

---

# Bacheloroppgave

TELE3001

Styre- og overvåkingsystem for distribusjon av drikkevann

Control and monitoring system for distribution of drinking water

---

## Hovedrapport

Trondheim, Vår 2020

Johan Haukalid  
Markus Raudberget  
Jone Vassbø  
Peder Ward

Oppdragsgiver:	Veileder:
Wago Norge AS	Kåre Bjørvik
Tor Erik Næbb	Universitetslektor
Distriktsjef Midt & Nord	Institutt for teknisk kybernetikk
Industri & Automasjon	NTNU, Trondheim



Fakultet for informasjonsteknologi og elektroteknikk  
Institutt for teknisk kybernetikk



---

# **Styre- og overvåkingsystem for distribusjon av drikkevann**

**Control and monitoring system for distribution of  
drinking water**

---

**Hovedrapport**



# Sammendrag

På bakrunn av oppdragsgivers mulige satsing innen vann og avløp, er det utarbeidet et styre- og overvåkingsystem for distribusjon av drikkevann. Systemet inneholder tre delsystemer. Ett system som detekterer lekkasjer i distribusjonsnettet, ett for å sikre vannkilden mot flom og ett for å leve rikttig trykk. Det legges vekt på modulær kode, moderne teknologi, bruk av høy nivå språk i kontainermiljø, kontinuerlig integrasjon, og innovative brukergrensesnitt. Prosjektet er avgrenset til fysiske komponenter som kontrollere og skjerm for brukergrensesnitt. Disse leveres montert på en demonstrasjonsrigg. Dette innebærer at utstyr og komponenter som pumper, vannkilden og lignende, blir simulert.

Systemet er designet for å kunne kommunisere på publiserte-abonner protokollen MQTT, via 4G. Dette innebærer utviklingen av en egen OPC UA til/fra MQTT link. Denne linken kjører i kontainermiljøet Docker, på alle delsystemene. Kjøring av kontainermiljø på komponentene er i den forbindelse kartlagt. Det er utarbeidet to brukergrensesnitt. Ett for å kontrollere systemet, utviklet i WAGO sin programvare, e!COCKPIT, og ett som visualiserer historisk data ved bruk av Grafana. For å muliggjøre dette er det laget en egen applikasjon som tar inn data fra MQTT og lagrer denne i en MySQL database. Da systemet er avgrenset, er det implementert simulering basert på virkelige verdier, slik at systemet oppfører seg mest mulig som om det skulle være koblet til et reelt anlegg.

På tvers av alle kontrollerne er det satt opp en standardisert arkitektur. I tillegg er det utviklet et objektbibliotek som tilbyr et bredt utvalg av funksjonalitet.

Det er implementert to metoder for å detektere lekkasje i distribusjonsnettet, hvor den ene er utarbeidet i dette prosjektet. Metoden er implementert i Python og visualisert i Grafana. Den benytter sammenligning av historisk data. Vannkilden er sikret mot flom, ved å bruke værmeldingen til å senke nivået i forkant av flommen. Det leveres stabilt trykk til forbrukerne ved hjelp av en reguleringsløyfe, samt en arbeidsdistributør som distribuerer arbeidet mellom tre pumper.

Selv om store deler av anlegget er simulert, leveres det et fullverdig kontrollsysten, som enkelt kan tas i bruk på et reelt anlegg. Det er tatt i bruk moderne teknologi som gjør systemet innovativt og kompatibelt med fremtiden.



# Forord

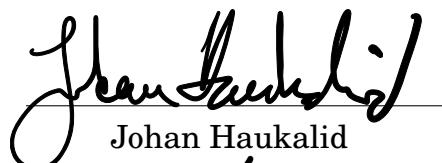
Denne rapporten er utarbeidet av undertegnede som en bacheloroppgave, tilsvarende 20 studiepoeng. Rapporten er utarbeidet ved Norges teknisk-naturvitenskapelige universitet, NTNU i Trondheim, våren 2020. Bacheloroppgaven ansees som et resultat av det treårige studiet elektroingeniør, med spesialisering i automatiseringsteknikk, på institutt for teknisk kybernetikk, ved fakultetet for informasjonsteknologi og elektroteknikk.

Oppgaven er utarbeidet i et samarbeid mellom WAGO Norge og undertegnede, den innebærer å utarbeide et styre- og overvåkingsystem for distribusjon av drikkevann. Systemet inneholder ulike deler der det finnes funksjonalitet for flomsikring av vannkilde, trykkregulering av forbruk og deteksjon av lekkasje. Systemet bruker moderne teknologi der kommunikasjonen foregår via MQTT over internett. I tillegg er deler av anlegget skrevet i høy nivåkode, Python, hvor koden er kjørt i kontainermiljøet Docker.

Oppgaven ble valgt på bakgrunn av muligheten til å teste ut moderne maskinvare, samarbeid med internasjonal leverandør, kompleksiteten og nytteverdien av systemet.

I et samarbeid med WAGO fikk prosjektgruppen en befatingsrunde hos Leksvik Vassverk for å få innsikt i distribusjon av drikkevann. Det rettes en ekstra takk til Leksvik Vassverk for en lærerik befaring på anlegget deres. Fra denne befaringen ble det hentet inspirasjon til å bygge opp en bedre og mer realistisk oppgave.

Takk til våre veiledere fra NTNU og WAGO, Kåre Bjørvik og Tor Erik Næbb i samarbeid med WAGO Support Norge, som alle har vært til hjelp under prosjektet.



---

Johan Haukalid



---

Jone Vassbø



---

Markus Raudberget



---

Peder Ward

Trondheim 20.05.2020

Sted/Dato



# Innhold

<b>Sammendrag</b>	<b>iv</b>
<b>Forord</b>	<b>vi</b>
<b>1 Innledning</b>	<b>1</b>
1.1 Introduksjon . . . . .	1
1.2 Bakgrunn . . . . .	1
1.3 Oppdragsgiver . . . . .	1
1.4 Problemstilling . . . . .	1
1.5 Prosjektmål . . . . .	2
1.6 Avgrensinger . . . . .	3
1.7 Definisjoner . . . . .	4
1.8 Rapportens oppbygging . . . . .	6
<b>2 Teori</b>	<b>7</b>
2.1 Høynivå- og lavnivåprogrammeringspråk . . . . .	7
2.2 Virtualisering med kontainere . . . . .	7
2.3 Publisér/Abonner protokoll . . . . .	7
2.4 Hash-funksjon . . . . .	7
2.5 Hash-tabell . . . . .	8
2.6 Objektorientert programmering . . . . .	8
<b>3 Systemoversikt</b>	<b>8</b>
3.1 Introduksjon . . . . .	8
3.2 Nettverkstopologi . . . . .	9
3.3 Kommunikasjon og dataflyt . . . . .	10
3.4 Kjøremiljø . . . . .	13
3.5 OPC UA - MQTT link . . . . .	15
3.6 Historisk data . . . . .	19
3.7 Sky-server . . . . .	20
3.8 PLS-programarkitektur og struktur . . . . .	22
3.9 Resultat . . . . .	28
3.10 Diskusjon . . . . .	29
3.11 Konklusjon . . . . .	32
<b>4 Simulering</b>	<b>32</b>
4.1 Introduksjon . . . . .	32
4.2 Værmelding fra YR . . . . .	33
4.3 Lekkasjedeteksjon . . . . .	35
4.3.1 Solcellepanelet . . . . .	35
4.3.2 Batteri . . . . .	36
4.3.3 Forbruk av vann . . . . .	37
4.4 Flomsikring . . . . .	40
4.4.1 Modellering av vannkilden . . . . .	40
4.4.2 Økning i vannivå . . . . .	41
4.4.3 Ventil . . . . .	42

4.5	Trykkregulering . . . . .	42
4.5.1	Trykkfall i distribusjonsnettet . . . . .	42
4.5.2	Strømtrekk på pumpe . . . . .	44
4.5.3	Prosess . . . . .	45
4.6	Implementasjon i sky . . . . .	45
4.7	Implementasjon i PLS og HMI . . . . .	48
4.8	Resultat . . . . .	50
4.9	Diskusjon . . . . .	50
4.10	Konklusjon . . . . .	51
<b>5</b>	<b>Lekkasjedeteksjon</b>	<b>52</b>
5.1	Introduksjon . . . . .	52
5.2	Målemetoder . . . . .	52
5.2.1	Teori rundt vannlekkasje . . . . .	52
5.2.2	Måling basert på akustikk . . . . .	54
5.2.3	Måling basert på strømningsmåling . . . . .	55
5.3	Deteksjonsmetoder . . . . .	55
5.3.1	Topp-ned vannbalanse metode . . . . .	55
5.3.2	Bunn-opp minimum nattforbruk metode . . . . .	57
5.3.3	Volumbalanse . . . . .	57
5.4	Resultater . . . . .	58
5.4.1	Valg av måler . . . . .	58
5.4.2	Lekkasjedeteksjon basert på historisk data (LHD) . . . . .	58
5.4.3	Lekkasjedeteksjon med volumbalanse . . . . .	65
5.4.4	Manuell avlesning i Grafana . . . . .	66
5.5	Implementasjon . . . . .	67
5.6	Diskusjon . . . . .	69
5.7	Konklusjon . . . . .	71
<b>6</b>	<b>Flomsikring</b>	<b>72</b>
6.1	Introduksjon . . . . .	72
6.2	Metoder og kilder . . . . .	72
6.2.1	Foroverkobling . . . . .	72
6.2.2	Kaskaderegulering . . . . .	73
6.2.3	Settpunktendring for fremtidig forstyrrelse . . . . .	75
6.3	Resultater . . . . .	76
6.3.1	Håndtering av forstyrrelse . . . . .	76
6.3.2	Innregulering . . . . .	77
6.4	Diskusjon . . . . .	78
6.5	Konklusjon . . . . .	79
<b>7</b>	<b>Trykkregulering</b>	<b>80</b>
7.1	Introduksjon . . . . .	80
7.2	Metoder og kilder . . . . .	80
7.3	Resultater . . . . .	81
7.4	Diskusjon . . . . .	82
7.5	Konklusjon . . . . .	82

<b>8 Brukergrensesnitt</b>	<b>83</b>
8.1 WAGO e!COCKPIT Visualization . . . . .	83
8.1.1 Introduksjon . . . . .	83
8.1.2 Objektstrukturer . . . . .	84
8.1.3 Oppbygningen av brukergrensesnittet . . . . .	84
8.1.4 Beskrivelse av brukergrensesnitt . . . . .	85
8.1.5 Diskusjon . . . . .	86
8.1.6 Konklusjon . . . . .	87
8.2 Grafana . . . . .	87
8.2.1 Introduksjon . . . . .	87
8.2.2 Beskrivelese av brukergrensesnitt . . . . .	88
8.2.3 Konklusjon . . . . .	89
<b>9 Demorigg</b>	<b>90</b>
9.1 Introduksjon . . . . .	90
9.2 Planlegging . . . . .	90
9.3 Resultat . . . . .	90
9.4 Konklusjon . . . . .	91
<b>10 Systemtesting</b>	<b>91</b>
<b>11 Administrativt</b>	<b>92</b>
11.1 Utviklingsmiljø . . . . .	92
11.2 Prosjektstyring . . . . .	93
11.3 Timeoversikt . . . . .	94
<b>12 Hovedkonklusjon</b>	<b>96</b>
<b>A Vedlegg system</b>	<b>A-1</b>
A.1 Globale objekter . . . . .	A-1
A.2 MQTT emne oversikt . . . . .	A-5
A.3 Signalliste . . . . .	A-7
A.4 PLS Objektbibliotek . . . . .	A-10
A.5 PLS program - Flomsikring . . . . .	A-38
A.6 PLS program - Lekkasjedeteksjon . . . . .	A-55
A.7 PLS program - Trykkregulering . . . . .	A-71
<b>B Vedlegg simulerte verdier</b>	<b>B-86</b>
B.1 Moodys diagram . . . . .	B-87
B.2 Ruhet for rør og tapskoeffisient for enkeltmotstander . . . . .	B-88
<b>C Vedlegg lekkasjedeteksjon</b>	<b>C-89</b>
C.1 Datablad strømningsmåler . . . . .	C-89
C.2 Resultat lekkasjedeteksjon . . . . .	C-99
<b>D Vedlegg flomsikring</b>	<b>D-101</b>
D.1 Innregulering flomsikring . . . . .	D-101
D.2 Resultat flomsikring . . . . .	D-103

<b>E Vedlegg trykkregulering</b>	<b>E-105</b>
E.1 Innregulering og resultat trykkregulering . . . . .	E-105
<b>F Vedlegg demorigg</b>	<b>F-108</b>
F.1 Arrangement tegninger demorigg . . . . .	F-108
F.2 Koblingskjema demorigg . . . . .	F-111
F.3 Bilde demorigg . . . . .	F-119
F.4 Deleliste demorigg . . . . .	F-121
<b>G Vedlegg systemtesting</b>	<b>G-123</b>
G.1 FAT . . . . .	G-123
<b>H Vedlegg timeforbruk</b>	<b>H-136</b>
H.1 Total timer per person . . . . .	H-137
H.2 Total timer tilsammen . . . . .	H-138
H.3 Total timer mot planlagt timer fra Gant . . . . .	H-139
<b>I Vedlegg brukergrensesnitt</b>	<b>I-140</b>
I.1 Brukergrensesnitt Grafana . . . . .	I-140
I.2 Brukergrensesnitt e!COCKPIT Visualization . . . . .	I-142
I.3 Brukermanual Grafana . . . . .	I-144
I.4 Brukermanual e!COCKPIT . . . . .	I-155

## Figurer

1 Oversiktsskisse med vannkilde . . . . .	2
2 Nettverkstopologi . . . . .	9
3 Kommunikasjon og kjøremiljøer . . . . .	11
4 Kommunikasjon med Simulering osv. . . . .	13
5 Flytdiagram OPC UA - MQTT link . . . . .	16
6 Lagring av data fra MQTT til MySQL . . . . .	20
7 PLS programarkitektur . . . . .	23
8 Ulempe med å ha settpunkt og prosessverdier i samme struktur	25
9 Flytskjema for innhenting og sending av værmelding . . . . .	35
10 Vannforbruk per person per time . . . . .	38
11 Virtuelt distribusjonsnett . . . . .	39
12 Trykkfall i distribusjonsnettet . . . . .	44
13 Lydbilde av lekkasje i plastrør [1, s.25] . . . . .	53
14 Lekkasjesøk med akustisk korrelator [2, s.22] . . . . .	54
15 Topp ned vannbalanse metode [3, s.19] . . . . .	56
16 Dag en verdier fra strømningsmåler . . . . .	59
17 Dag en time-gjennomsnittsverdier . . . . .	60
18 Dag en og dag to med gjennomsnittsverdier . . . . .	61
19 Dag tre og gjennomsnittsverdier for dag en og to . . . . .	62
20 Dag tre og gjennomsnittsverdier av dag 1-2 med avviksgrense	63
21 Dag ni og gjennomsnittsverdier for dag 4-8 med avviksgrense	64
22 Sammenligning av gjennomsnittsverdier uten gradvis lekkasje	65

23	Distribusjonsnett utklipp . . . . .	66
24	Oversiktsbilde flomsikring . . . . .	72
25	Blokkskjema av tilbakekobling og foroverkobling [4, s.271] . .	73
26	Implementering av kaskaderegulering . . . . .	74
27	Blokkdiagram av kaskaderegulering [4, s.282] . . . . .	75
28	Blokkdiagram av settpunktendring . . . . .	76

## Tabeller

1	MotorDigitalHmiPv objektstruktur . . . . .	26
2	Konvertering av tekst til tall for værmelding . . . . .	34
3	Værtype med værfaktor . . . . .	36
4	Utregning av simulerte verdier i strømningsmålere . . . . .	40
5	Spesifikasjoner for vannkilde . . . . .	41
6	Simulering av for økning i vannivå basert på nedbør . . . . .	41
7	Tallverdier for trykkfall . . . . .	43

## Liste av kodebiter

1	Rekursiv metode for lesing av OPC UA variabler . . . . .	17
2	Restartlogikk kommando OPC UA - MQTT link . . . . .	18
3	Restartlogikk OPC UA - MQTT link . . . . .	18
4	Trådsikring i OPC UA - MQTT link . . . . .	19
5	Docker-compose konfigurasjon for sky-server . . . . .	21
6	Oppgavebehandler hovedlogikk . . . . .	47
7	Trykkregulering: Simulering av forsinkelse . . . . .	49
8	Trykkregulering: Simulering av forstyrrelse . . . . .	49
9	Datastruktur strømmningsmålere . . . . .	68
10	Metode for oppdatering av datastruktur strømningsmålere . .	68



# **1 Innledning**

## **1.1 Introduksjon**

I dette kapitlet er det forklart hva prosjektet handler om, hva som er bakgrunnen for oppgaven og hvilken problemstilling det jobbes med. Prosjektets mål er forklart og hvilke avgrensinger som er satt. Mye av innholdet i dette kapitlet er utarbeidet i et forprosjekt og er dermed hentet fra forprosjekt-rapporten, rapporten ligger vedlagt i PA-perm.

## **1.2 Bakgrunn**

Oppdragsgiver er i dag en etablert leverandør innen flere bransjer, blant annet innenfor byggautomasjon og energi. For en mulig fremtidig satsing innen vann og avløp er det ønskelig med et demonstrasjonsanlegg til bruk på messer og lignende. Demonstrasjonsanlegget vil vise hvordan oppdragsgivers produkter også kan brukes i vann- og avløpsbransjen. Det vil demonstrere produktene i produksjon, samt hvor effektive og robuste teknologiene de bruker er.

## **1.3 Oppdragsgiver**

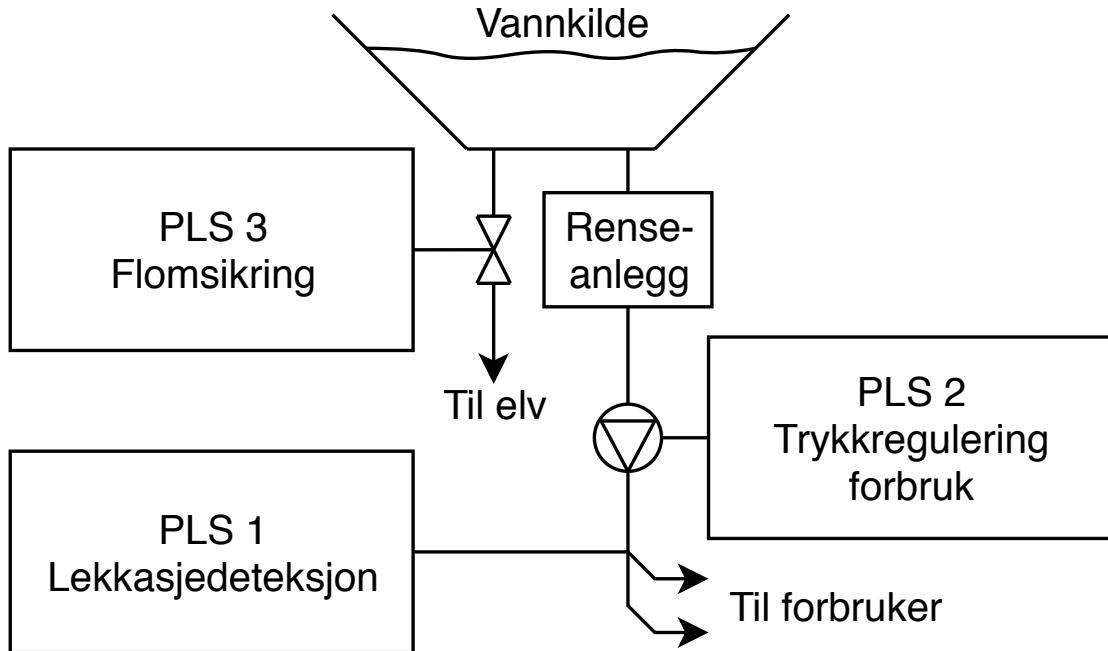
Prosjektets oppdragsgiver er WAGO Norge, som er en kjent leverandør innen løsninger av elektronikk og automasjon. Selskapet ble stiftet i Norge i 2005 og er et datterselskap av det tyske konsernet WAGO. Oppdragsgiveren har bidratt til prosjektet med deres utstyr, programvare og support igjennom deres tekniske kundeservice. Oppdragsgiver har også bidratt med en bacheloroppgave som har blitt skrevet i samarbeid med studenter fra NTNU tidligere. Innen utstyr har de bidratt med PLS'er, WAGO Cloud, panel for visualisering av brukergrensesnitt, koblingsutstyr og monteringslokale. For å arbeide med WAGOs maskinvare har de bistått med lisensbasert programvare, e!COCKPIT. WAGOs support har bidratt med å løse utfordringer innen bruk av maskinvare og utarbeiding av kode.

## **1.4 Problemstilling**

For å demonstrere oppdragsgivers produkter skal det lages et styre- og overvåkingsystem for tiltenkt distribusjon av drikkevann. Systemet, dokumentasjonen og prosessen med å utvikle systemet skal demonstrere oppdragsgivers produkter. Systemet skal demonstrere kvaliteten på produktene og de teknologiene som er brukt i dem. Ytelse og responsider settes på prøve for å kartlegge om produktene egner seg for et kritisk system, som drikkevann.

Systemet skal bestå av tre mindre systemer slik det er representert i figur 1. For å kunne detektere lekkasjer i ledningsnettet, vil det være et system for å måle strømning og prosessere dataen. For å sikre vannkilden mot flom, vil det være et system som styrer utsippet til elven og måler

nivået. For å kunne levere drikkevann med rett trykk, vil det være et system som regulerer pumper og opprettholder trykket, ved svingninger i forbruket.



Figur 1: Oversiktsskisse med vannkilde

Det legges vekt på modulær og gjenbruksbar kode, slik at oppdragsgivers kunder kan ta den i bruk. Det skal testes ut kjøring av produksjonskode i kontainermiljø skrevet i Python, for å kartlegge hvordan maskinvaren yter i dette miljøet. Det skal kartlegges i hvilken grad et er mulig å oppnå kontinuerlig integrasjon av ny funksjonalitet i de ulike delene av systemet. Det skal simuleres de verdiene som er nødvendige for at systemet skal se operativt ut under demonstrasjoner. I tillegg skal det testes ut visualisering av data i Grafana etter ønske fra oppdragsgiver.

## 1.5 Prosjektmål

Prosjektets hovedmål er å levere et demonstrasjonssystem for distribusjon av drikkevann.

Oppdragsgiver vil stille ut systemet på messer i vann- og avløpsbransjen, noe som resulterer i følgende effektmål:

- Økt synlighet for oppdragsgiver og produktene.
- Økt salg.

Prosjektet vil resultere i et demonstrasjonssystem, med tilhørende dokumentasjon. Dette er prosjektets resultatmål, som er delt inn i følgende delmål:

- System for å detektere lekkasjer i ledningsnettet, som publiserer informasjonen til skyen.

- Flomsikring som opererer basert på værmeldinger.
- Stabilt trykk på drikkevannet hos forbruker.
- Overvåking av systemet med trender, alarmer og alarmhistorikk.
- Utveksling av data foregår gjennom skyen.
- Standard funksjonalitet må være modulær og gjenbrukbar.
- Deler av anlegget skal utvikles i Python.

Gjennom prosessen med å produsere produktet er det satt opp følgende felles prosessmål:

- Få kunnskap om og erfaring med å bruke produkter som er ledende innen bruk av moderne teknologi.
- Få erfaring med å samarbeide med en landsledende automasjonsleverandør.
- Få erfaring med å jobbe og samarbeide i en gruppe.

## 1.6 Avgrensinger

Alt av utstyr monteres på en demonstrasjonsrigg. Prosjektet er derfor avgrenset til å bestå av kontrollere, skjerm for brukergrensesnitt, strømforsyninger og sikringer. Fysisk utstyr som ventiler, trykkmålere, pumper og nivåmålere vil bli simulert. Det vil ikke bli implementert noe i forbindelse med renseanlegg. Vannkilden med drikkevann, forbruket, været og tidsforholdet vil bli simulert av en egen programvare. Avgrensningene er satt for å kunne levere en realistisk løsning innen de gitte tidsrammene.

## 1.7 Definisjoner

Under følger definisjoner som er brukt i denne rapporten.

Open Source	Programvare med åpen kildekode [5]
WBM	Internettbasert håndtering av WAGO utstyr [6]
FAT	Fabrikk Aksepterings Test [7]
MQTT	Message queuing telemetry transport - Publisér/abonner meldings protokoll [8]
VPN	Virtuelt Privat Nettverk [9]
TCP	Transmission Control Protocol [10]
OPCUA	Object Linking and Embedding for Process Controll, Unified Architecture [11]
Real-time	Sanntidssystem [12]
Database	Organisert og strukturert lagring av data [13]
Docker	Kontainer platform for kjøring av applikasjoner i et lettvekts virtuelt miljø[14]
Docker-compose	Verktøy for å kjøre flertalls Docker applikasjoner [15]
WinSCP	Filhåndteringssystem mellom en lokal og fjern datamaskin [16]
Suffix	Orddel som etterstilles et helt ord. [17]
Database klient	Destinasjonen for dataen som blir hentet fra databasen [18]
Google Cloud	Ekstern lagring av data, i Google sky. [19]
Ubuntu	Operativsystem for Linux [20]
Adminer	Databaseadministrasjonsverktøy [21]
Portainer	Open source administrasjonsverktøy for å håndtere docker miljø[22]
Kontainer	Pakke med programvare for å kjøre program i docker [23]
GUI	Graphical User Interface, annet ord for brukergrensesnitt [24]
DHCP	Dynamic Host Configuration Protocol [25]
IP	Internet Protocol [26]
MAC	Media Access Control [27]
Inkscape	Digitalt tegneprogram [28]

API	Grensesnitt for å samhandle mellom ulike programvarer [29]
Json	Standard for formatert dokumenter til datautveksling [30]
CDP-studio	Utviklingsverktøy for å bygge automatiserte og industrielle brukergrensesnitt. [31]

## 1.8 Rapportens oppbygging

Rapportens oppbygging baserer seg på “Veiledning rapportskriving” utdelt fra NTNU. Denne veiledning er hentet fra heftet “Prosjektarbeid - En Veiledning for studenter” [32]. Målgruppen for denne rapporten er satt til 3. års studenter på elektroingeniør med spesialisering i automatiseringsteknikk.

Prosjektoppgaven er i hovedsak delt opp i tre hoveddeler, lekkasjedeksjon, flomsikring og trykkregulering. Disse har fått hvert sitt hovedkapittel og er avgrenset og bygget opp slik at man skal kunne lese hver del for seg selv. Hovedkapitlene på disse tre hoveddelene inneholder samme delkapitteloppdelingen. I tillegg har oppgaver som er felles for hoveddelene fått hvert sitt hovedkapittel, slik som teori, systemoversikt, simulering, brukergrensesnitt, demorigg og systemtesting. Det gjør det f.eks. mulig å lese bare systemoversikt eller simulerte verdier, dersom man ikke er interessert i f.eks. flomsikring. Ikke-tekniske kapitler som innledning, administrativt og hovedkonklusjon har også fått egne hovedkapitler.

Rapporten er skrevet i Latex. Ved å bruke Latex har man muligheter til å sette opp og bruke funksjoner som estetiske hjelpeMidler. F.eks. har det blitt brukt “equations” for å vise matematiske utregninger. Forsiden og tittelsiden er også egenprodusert i Latex.

Gjennom hele rapporten er det referert til kilder ved hjelp av Latex verktøyet BibTex. Disse kildene finner man igjen i referanselisten og er referert til med IEEE standard. Vedleggene er navnsatt med nummer og bokstav. Bokstaven forklarer hvilket hovedkapittel det tilhører og nummeret er unikt for hvert vedlegg i det kapittelet. Dersom man trykker på vedlegg referansen i teksten kommer man direkte til vedlegget.

I rapporten er det lagt vekt på estetikk, ryddighet og leservennlighet. Ved å dele opp kapitlene slik det er gjort, er det enkelt å finne fram til riktig lesestoff ut fra innholdsfortegnelsen. Det er tilordnet slik at tabeller, grafer og figurer har samme stil og farger. Alle figurer, tabeller og grafer er egenproduserte, men kan være inspirert av andre figurer. Disse finnes også i innholdsfortegnelsen, noe som gjør rapporten ryddigere.

Gjennom prosjektet har “git” blitt brukt som versjonskontrollsysten for koden som er skrevet i Python. Github [33] er brukt som ekstern kodebase. For å holde koden oversiktlig er den ikke lagt ved rapporten som pdf, det henvises heller til Github. Det er opprettet en “tag” som definerer den versjonen som gjelder for innleveringen. Denne har versjon v1.0.0. I tillegg er koden lagt ved som kodelfiler i PA-permen.

## 2 Teori

I dette kapitlet presenteres teorien det er ansett at leseren trenger utdypning om. Det gis en innføring i teori rundt generelle programmeringsbegreper, virtualisering, samt en publiser/abonner protokoll.

### 2.1 Høynivå- og lavnivåprogrammeringspråk

Høynivå- og lavnivåprogrammeringspråk er begge språk som skrives av mennesker. Det finnes flere meninger rundt hva som er hva, og det er derfor gitt en generell forklaring i dette kapitlet. Høynivåspråk er kjent som lett for et menneske å lese og forstå. Når koden er skrevet i et høynivå må man typisk bruke en tolker og/eller en kompilator for å lage en kode som maskinen skal kunne forstå. Et lavnivå derimot, er vanskeligere for et menneske og forstå, men lettere for maskinen. For å få maskinen til å forstå lavnivå språk brukes typisk en assembler, som bare konverterer språket [34]. Man sier at lavnivåspråk ligger nærmere maskinen enn et høynivå språk og at en linje med kode typisk gjør mer i et høynivåspråk enn i et lavnivåspråk.

### 2.2 Virtualisering med kontainere

Virtualisering med kontainere skiller seg fra tradisjonell virtualisering med virtuelle maskiner, ved at kontainerene virtualiserer bare operativsystemet og ikke hele maskinen [35]. For å gjøre dette mulig kjøres det en programvare som tar seg av å allokkere ressurser til de kontainerene som kjører. Når denne programvaren er installert kan kontainere kjøres opp med noen enkle kommandoer.

### 2.3 Publiser/Abonner protokoll

En protokoll av typen publiser/abonner fungerer slik som navnet tilsier. Klienter publiserer og/eller abonnerer på data. Det settes opp en server, denne serveren holder rede på hvem som skal ha hvilke data samt rutingen av denne dataen. Rutingen er mulig da data publiseres på emner. Emnene kan bygges opp slik brukeren ønsker. Om man samler inn temperatur data fra en bygning kan det være naturlig å bygge opp emnene med *byggX\etasjeY\romZ\sensor1*. De klientene som ønsker å abonnere på denne dataen registrerer et abonnement på serveren. Serveren vil da sørge for at dataen som blir publisert på dette emnet sendes videre til denne abonnenten [36].

### 2.4 Hash-funksjon

En hash-funksjon tar en input av variabel lengde og genererer en unik verdi med fast lengde. Det brukes matematikk for å få utført denne

konverteringen og det er typisk vanskelig å generere tilbake den verdien man puttet inn [37]. Bruken av hash-funksjoner er utbredt, å brukes i f.eks. kryptovaluta og hash-tabeller.

## 2.5 Hash-tabell

En hash-tabell er en datastruktur. Datastrukturen er bygget opp av en tabell med en indeks. Det brukes en hash-funksjon til å velge hvilken indeks dataen skal lagres på i tabellen [38]. Om man f.eks. skal lagre måleverdier fra en sensor en gang i døgnet, kan datastrukturen kjøre dato-en gjennom en hash-funksjon, få en indeks, og lagre måleverdien på den indeksen i tabellen. Datoen blir da kalt nøkkelen og måleverdien blir kalt verdien. Fordelen ved å bruke en hash-tabell er at man kan hente ut data uten å måtte rulle gjennom tabellen [38]. Fra eksempelet som er brukt, kan man hente ut hvilken som helst verdi, så lenge man vet dato-en.

## 2.6 Objektorientert programmering

Objektorientert programmering innebærer at man anser alt som objekter. Man kan be disse objektene utføre handlinger og lagre data. Objekter kan også utføre handlinger for andre objekter [39]. Et objekt kan f.eks. være en minibank, og et annet objekt kan være en person. Minibanken kan inneholde data om hvor mye penger personen har i banken, og personen kan inneholde data om hvor mye den har i pengeboken. En person kan f.eks. be minibanken om et uttak, mens minibanken kan be personen om å identifisere seg. I forbindelse med industriell programmering kan man f.eks. se på en analog inngang som et objekt. Objektet kan da f.eks. inneholde data om rå-verdien, skaleringsverdier og skalert verdi. Det kan f.eks. tilby funksjonalitet som å skalere rå-verdien eller evaluere alarmkriterier.

Objektorientert programmering skiller seg fra tradisjonell programmering. Hvor tradisjonell programmering ikke inneholder objekter, men heller en liste av instruksjoner. Hvor disse instruksjonene forteller hva datamaskinen skal gjøre [39].

# 3 Systemoversikt

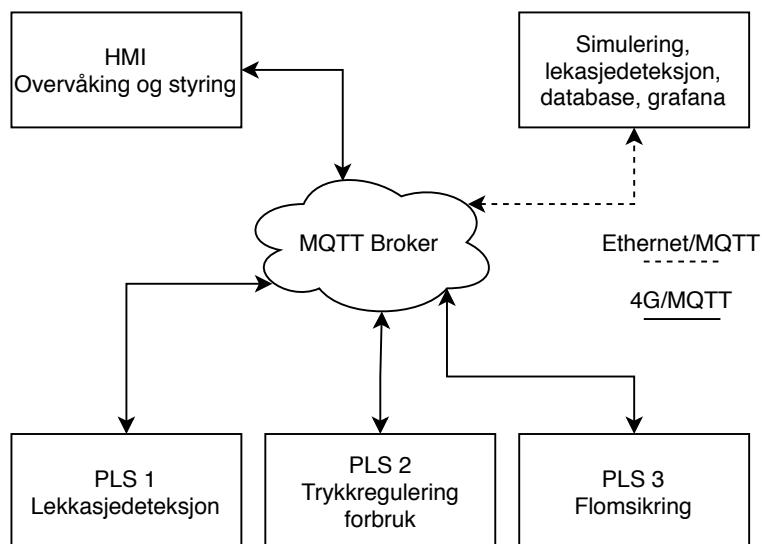
## 3.1 Introduksjon

I dette kapittelet fremstilles det hvordan systemet er satt sammen med tanke på nettverkstopologi, kommunikasjon, arkitektur og struktur innad i hver del. Kommunikasjonen er standardisert og lik mellom de forskjellige PLS'ene og HMI'en. Arkitekturen og strukturen er også standardisert og er derfor tilnærmet identisk i de forskjellige PLS programmene, samt følger HMI'en samme oppsett så langt det lar seg gjøre. Dataflyten er presentert, samt de delene som produserer eller konsumere data. Det er laget diverse programmer for å forenkle systemet, disse er også presentert i de kommende

kapitlene. Det finnes utallige detaljer rundt systemoversikten. Det er ansett som for omfattende å ta med alle detaljene i dette kapittelet, det er derfor brukt et redusert detaljnivå og det henvises til vedlegger.

### 3.2 Nettverkstopologi

I et distribusjonssystem for drikkevann er det store avstander mellom de ulike delene av systemet. Utløpet til vannet vil ofte ha stor geografisk avstand fra renseanlegget og så videre. Det er derfor satt opp en nettverkstopologi som innebærer at de forskjellige delene av systemet er forbundet ved hjelp av en internett tilkobling. For å koble delene i systemet til internett er det brukt 4G teknologi, da drikkevann ofte ligger utenfor bebygget strøk og har dårlig forbindelse til kablet internett. Topologien blir derfor enkel, den er visualisert i figur 2. Unntaket er de delene av systemet som kjører i skyen, det vil si simuleringsprogramvaren, lekkasjedeksjon, database for historisk data og Grafana. Disse systemene kjører allerede på en server i skyen og er derfor tilkoblet internett via kablett ethernet, og ikke 4G.



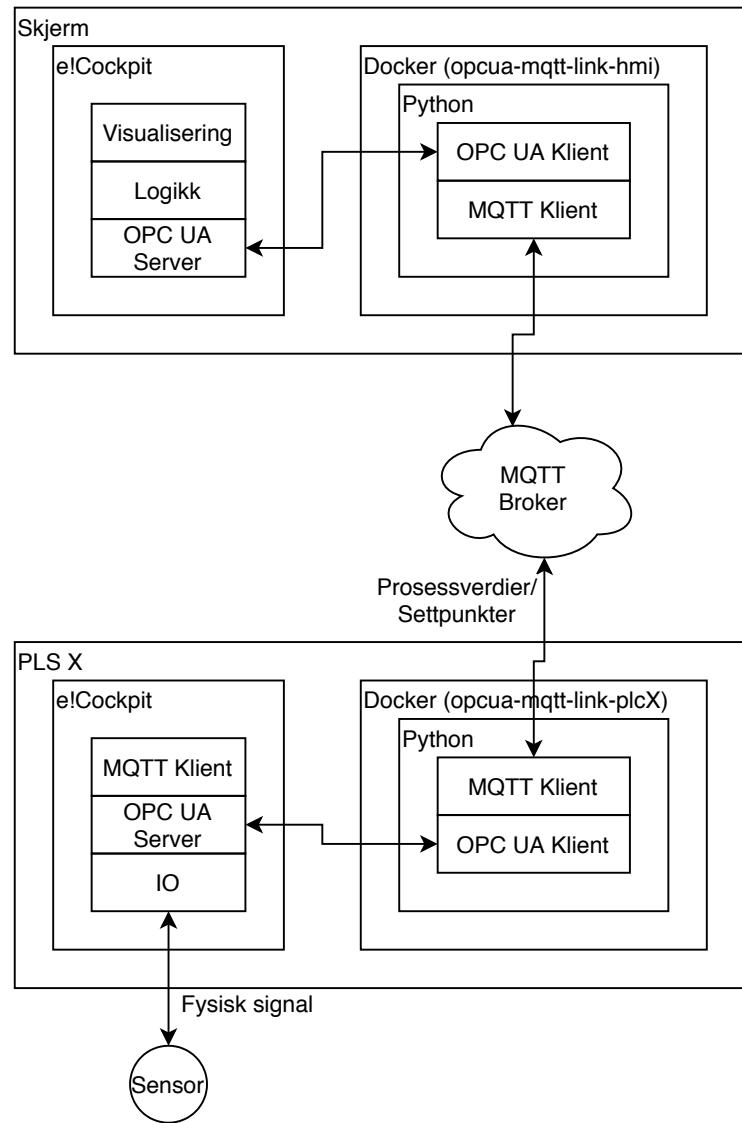
Figur 2: Nettverkstopologi

Som applikasjonsprotokoll er det valgt MQTT. Dette er en lettvekts publisér/abonner protokoll som er utviklet i forbindelse med “Tingenes internett”. Detaljer rundt denne typen protokoll finnes i kapittel 2.3. En av de store fordelene med MQTT er at klientene ikke trenger noen form for konfigurasjon i brannmurene eller en VPN, så lenge klienten har internett kommet dataen gjennom. Serveren, også kalt brokeren, må derimot konfigureres opp til å kunne nås fra internett, men når en broker er konfigurert er det vanlig å kjøre denne i skyen og ikke flytte den. Dette er avgjørende faktorer da delsystemene er koblet opp via 4G, ofte kan ha dårlig forbindelse, samt har dynamiske IP-adresser siden det er snakk om en demorigg. Det kjøres MQTT klienter på PLS'ene og på HMI'en. Disse

klientene publiserer og abонerer på emner i henhold til vedlegg A.2. Hvilken type eller hvilke spesifikasjoner MQTT-brokeren har er ikke avgjørende, så lenge den er tilgjengelig på internett. Det er i dette tilfellet brukt en gratis broker fra HiveMQ som kjører i skyen, og er tilgjengelig for alle.

### 3.3 Kommunikasjon og dataflyt

Da kommunikasjonen mellom HMI og PLS går over internett er det som nevnt valgt å bruke applikasjonsprotokollen MQTT. For å opprette kommunikasjon mellom komponentene, som krever minimalt med endring i koden ved endringer i tags, samt er enkel å sette opp og gjenbrukbar, er det valgt å lage en OPC UA - MQTT link. Linken er en dynamisk mapping mellom den innebygde OPC UA serveren i e!COCKPIT kjøremiljøet og MQTT. Den overordnede dataflyten mellom PLS og HMI er visualisert i figur 3. Som visualisert i figuren, kjører denne linken som et Python-program i en kontainer. Kontaineren kjøres i virtualiserings implementasjonen Docker. Docker kjører på PLS'ens operativsystem og kontaineren kommuniserer med PLS'en over TCP. Slik figuren viser går prosessverdier og settpunkter via denne OPC UA - MQTT linken.

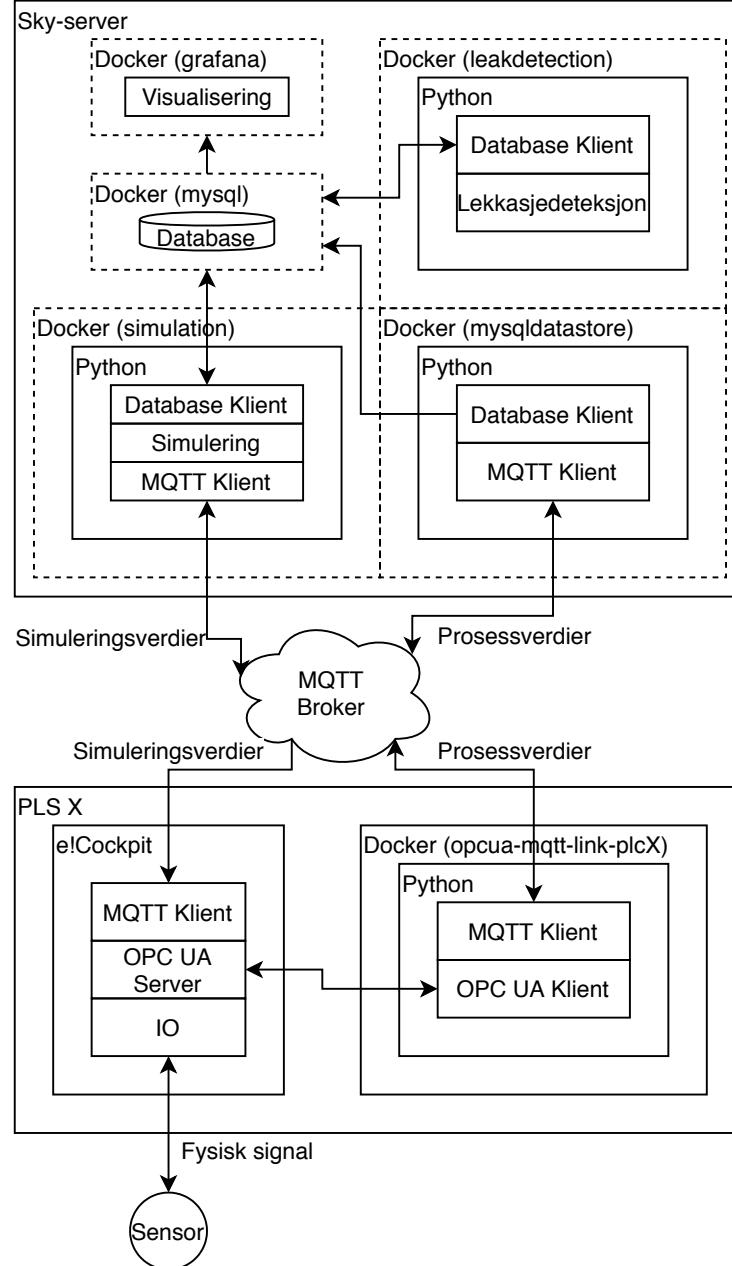


Figur 3: Kommunikasjon og kjøremiljøer

Systemet er satt opp slik at PLS'ene opererer som de som eier dataen. Med dette menes det at PLS'ene forespør aldri data, de publiserer kontinuerlig prosessverdier samt tar i mot settpunkter fra HMI. HMI forespør data fra PLS'ene om den mangler noe, samt sender eller mottar settpunkter. All data som sendes mellom HMI og PLS er dokumentert i signallisten i vedlegg A.3. All data er lagret i PLS'ene på en slik måte at selv om det oppstår strømbrudd eller det blir foretatt en nedlasting av nytt program, vil ikke dataen gå tapt. Det er satt opp tellere i alle PLS'ene, det vil si et tall som hele tiden økes. Dette tallet sendes til HMI for å kunne detektere brudd i kommunikasjonen. HMI overvåker dette tallet. Om dette tallet ikke endrer seg i HMI i løpet av 60 sekunder indikerer dette at HMI ikke har mottatt noe data fra PLS de siste 60 sekundene. Denne metoden er brukt til å generere en alarm i HMI slik at operatøren får beskjed om at kommunikasjonen med en PLS er brutt.

Kommunikasjonen og dataflyten mellom PLS'ene,

simuleringsprogramvaren og så videre, er visualisert i figur 4. I tillegg til kommunikasjonen som går via OPC UA - MQTT linken går simuleringsverdiene direkte fra simuleringsprogrammet til MQTT klienter som er implementert i e!COCKPIT miljøet. Dette er tydelig visualisert i figuren ved at pilene som representerer dataflyt er merket med "Simuleringsverdier". Denne dataen bruker allikevel samme MQTT broker da dette også er standard MQTT. Sky-serveren kjører fem Docker kontainere som utveksler data med systemet eller mellom hverandre. Fra øverst til venstre har man en container som kjører Grafana, denne henter data direkte fra databasen. Databasen lagrer all data og opererer kun som en slave som gjør som den får beskjed om fra de andre applikasjonene. Lekkasjedeksjonen henter ut historisk data fra databasen, deretter kalkulerer den gjennomsnittlige data og lagrer denne dataen i databasen igjen, slik at Grafana kan visualisere dataen. Simuleringsprogrammet skriver simulerte sensordata direkte i databasen på grunn av ytelsesutfordringer, samt sender ut nåverdier på MQTT. Kontaineren med navn "mysqldatastore" tar i mot all sensordata på MQTT og lagrer denne dataen i databasen. Sensordataen blir lagret slik at den skal kunne visualiseres i historiske grafer. Da systemet skal bli brukt som en flyttbar demorigg, er all MQTT kommunikasjon konfigurert til Qos 0. Qos står for "Quality of Service" og sier noe om hvor sikker protokollen skal være på at dataen blir mottatt. Qos 0 betyr at dataen blir sendt i håp om at mottakeren får den, men den sendes ikke om igjen om dataen ikke kommer frem [40].



Figur 4: Kommunikasjon med Simulering osv.

### 3.4 Kjøremiljø

I utgangspunktet kommer PLS'ene med to kjøremiljøer. Det ene er Codesys 2 og det andre er e!COCKPIT. Begge disse miljøene er real-time miljøer. Det som er forskjellen på de, er hvilket program som brukes for å utvikle programmet. Som et tredje miljø er det mulig å kjøre hvilken som helst programvare direkte på linux operativsystemet.

Kontrollogikken i PLS'ene kjøres i e!COCKPIT. Med kontrollogikken menes den logikken som kan styre fysiske komponenter i systemet. Dette innebærer for eksempel regulering og styring av ventiler. e!COCKPIT sitt kjøremiljø tilbyr å kjøre flere oppgaver på forskjellige tidsintervaller,

samt prioritere de forskjellige oppgavene etter hva som er viktigst. Denne funksjonaliteten er tilrettelagt for styresystemer. Oppdragsgiver er leverandør av denne programvaren og er derfor valgt i dette tilfellet. Codesys er mye av det samme som e!COCKPIT, men siden den ikke leveres av oppdragsgiver er den ikke utforsket. For å kjøre logikk på PLS'ene via e!COCKPIT koder man rett i programvaren, kobler opp til PLS'en via IP adressen, laster programmet over og setter PLS'en i kjør. Denne metoden å utvikle på er tilrettelagt for rask utvikling, men dette kommer med restriksjoner som f.eks. at den mest kompliserte koden kan være litt vanskelig å få til, samt at programvaren er lisensiert.

Det tredje miljøet er brukt til å kjøre kontainer virtualisering med Docker. Docker kjører igjen Python applikasjonen OPC UA - MQTT link, dette er visualisert i figur 3, og linken er utdypet i kommende kapittel. For å ta i bruk dette miljøet må Docker installeres på PLS'ene. Oppdragsgiver har på sine nettsider lagt ut en ferdig pakke for å få installert Docker. Først laster man ned installasjonsfilene, disse lastes opp i WBM og aktiveres. Når dette er gjort må man aktivere videresending av data, deretter kan man ta i bruk Docker. Samme metode brukes om man ønsker å ta i bruk Docker-compose, som i dette tilfellet er tatt i bruk for å forsikre at en kontainer starter om den stopper. Docker-compose er en programvare for å administrere kontainere. Den kan kjøre opp flere kontainere og sette opp nettverk mellom de.

For å kunne kjøre Docker kontaineren på PLS'en må det bygges et bilde som er kompatibelt. Et bilde er en pakke med operativsystem og applikasjon i ett, før det har blitt startet. Når et bilde startes kalles det en kontainer og man kan starte flere kontainere av samme bilde. Da PLS'ene ikke har samme prosessorarkitektur som en vanlig datamaskin krever det en spesiell byggeprosess for kunne bygge bildene på datamaskinen. Denne byggeprosessen er enda i eksperimentell modus fra Docker sin side. Den normale måten å bygge bilder på er å bygge de på den prosessor arkitekturen de skal brukes på. Det hadde i dette tilfellet vært mulig, men da byggingen er en ressurskrevende operasjon, så er dette gjort på en datamaskin, som har mye høyere ytelse en PLS'en. Når bildet er ferdig bygget kan det gjøres tilgjengelig på internett via tjenesten Docker hub. Dette gjøres ved å trykke bildet opp i denne tjenesten. Når det er gjort kan det hentes ned fra hvor som helst, så lenge man har internett.

Byggeren kjører også i Docker, dermed er den enkel å sette opp. Byggeren lages ved følgende kommando:

```
Docker buildx create --name builder-for-plc
```

Deretter velger man å bruke den med:

```
Docker buildx use builder-for-plc
```

For å bygge et bilde for arkitekturen arm/v7, som er PLS'en sin prosessorarkitektur, og trykke bildet opp i Docker hub brukes følgende kommando:

```
Docker buildx build -f Dockerfile-name --platform  
→ linux/arm/v7 -t username/imagename:tag --push .
```

For å kjøre bildet på en PLS brukes Docker-compose med en enkel konfigurasjon som følger:

```
version: '3.1'  
services:  
  Pythonapp:  
    image: username/imagename:tag  
    restart: always
```

Denne konfigurasjonen legges inn på PLS'en. Den overføres med et filoverførings verktøy, f.eks. "WinSCP" for Windows. Når den er overført kan følgende kommando kjøres fra filplasseringen:

Docker-compose up

Bildet vil da hentes ned fra Docker hub og startes. Det vil også alltid restartes slik det er konfigurert i Docker-compose konfigurasjonen. Trenger man å oppdatere programvaren som kjører i Docker, bygger man enkelt bildet på ny, samt trykker det opp i Docker hub. Deretter stopper man den kontaineren som allerede kjører med:

Docker-compose down

Henter ned det nye bildet med:

Docker-compose pull

Og starter opp igjen med det nye bildet med:

Docker-compose up

På denne måten får man raskt rullet ut nye versjoner av programvaren.

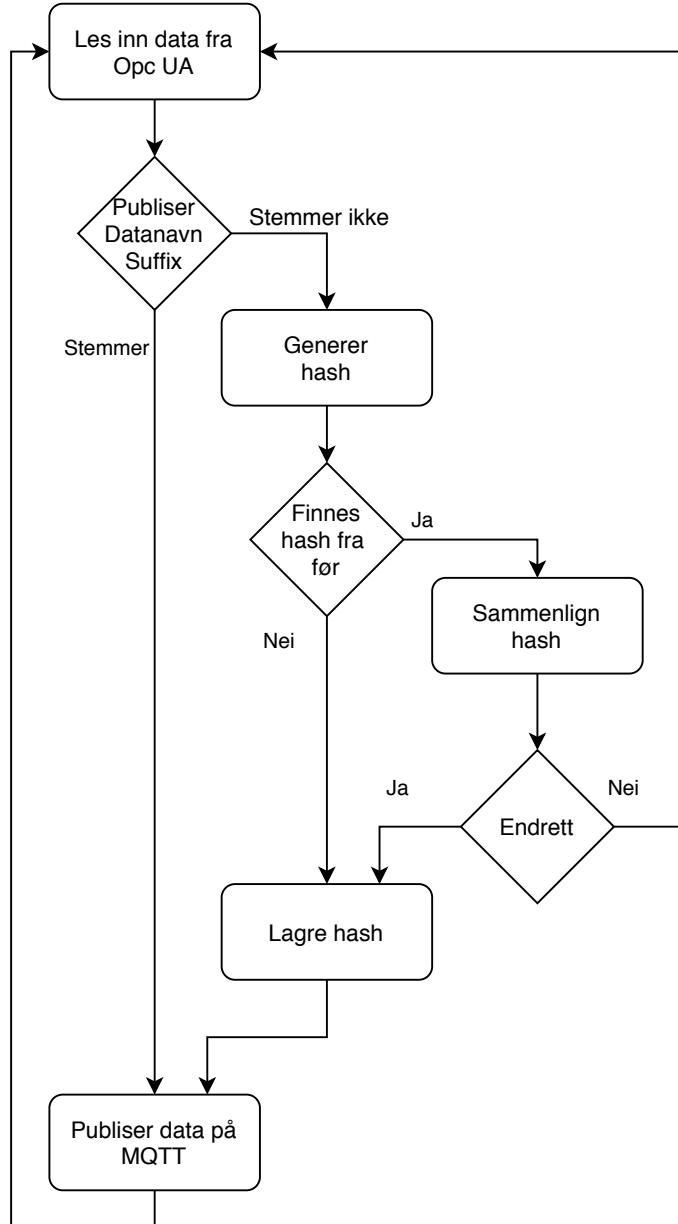
### 3.5 OPC UA - MQTT link

Som nevnt i kapittel 3.3 er det laget en OPC UA - MQTT link. Programvaren er utviklet i Python og hovedlogikken følger flyten som er visualisert i figur 5. I tillegg til hovedlogikken kjøres det en initialisering, samt en tråd for å ta i mot data.

Før hovedlogikken starter, kjøres det en initialisering. Denne initialiseringen utforsker seg gjennom alle objektene som blir publisert i OPC UA serveren, med deres underliggende trestruktur av objekter. Alle objektene med tilhørende OPC UA adresser blir lagret i to hash-tabeller. En tabell med objekter og en med node-adresser. Dette gjøres for å øke ytelsen til linken. Å utforske objektene i OPC UA serveren er en tidkrevende jobb. Når det i hovedlogikken skal leses eller skrives data fra eller til OPC UA serveren, gjøres dette ved å slå opp i hash-tabellene, i steden for å kjøre en

utforsking hver syklus. Dette innebærer at det kreves en omstart av linken hvis det gjøres endringer i OPC UA objektstrukturen i e!COCKPIT.

Hovedlogikken ruller gjennom hash-tabellen av objekter så fort den klarer. Når den er kommet til det siste objektet så starter den om igjen. For hvert objekt fullføres flyten som er visualisert i figur 5.



Figur 5: Flytdiagram OPC UA - MQTT link

Et objekt, f.eks. et pumpeobjekt leses ut fra serveren. Selve OPC UA lesingen/skrivingen er behandlet av et bibliotek, *opcua* [41]. Først sjekkes objektnavnet mot et predefinert suffix, som i dette tilfellet er \*Pv. Det vil si at alle objekt som slutter på Pv stemmer med publisering suffixet. Disse objektene publiseres ut på MQTT hver syklus. Om objektnavnet ikke slutter på Pv stemmer det ikke og ut i fra valgfirkanten går flyten til høyre. Det

genereres da en hash ut i fra alt datainnholdet i objektet. Dette er en unik ID for objektets tilstand. Dette gjøres for å raskere kunne finne ut om et objekt har endrett seg. Objektnavnene med deres hash-verdier lagres i en hash-tabell etter hvert som de lages. Om hashen ikke finnes fra før blir den lagret og objektet blir publisert på MQTT. Om hashen finnes fra før, som vil si at det ikke er første syklusen, sammenlignes den nye hashen med den sist lagrede hashen for det objektet. Om hashene er like, har objektet ikke forandret seg mellom denne og sist syklus. Objektet blir da ikke publisert på MQTT. Om hashene ikke er like har objektet endret seg og det blir publisert på MQTT.

Flyten er satt opp slik for å kunne publisere prosessverdier hele tiden, mens settpunkter kun publiseres når de har endret seg. Settpunkter publiserer kun når de har endret seg for å redusere dataen som skal sendes mellom systemene, samtidig som det muliggjør en lese/skrive funksjonalitet. Med dette menes det at både PLS og HMI kan endre et settpunkt og endringen vil oppdateres i den andre enden. Om begge endrer på samme tidspunkt vil den som endret sist bli gjeldende, dette er ikke sett på som et problem i praksis, siden man i utgangspunktet ikke endrer settpunkter hele tiden. PLS'en vil i utgangspunktet ikke endre settpunkter, logikken brukes til validering av settpunkter. Da HMI tar i mot prosessverdier og ikke har noen egne, er det bare høyre del av flyten som er implementert i HMI. Dette gjør at HMI og PLS linken ikke er identisk, men tilnærmet lik. Linken er implementert med dynamiske miljøvariabler og er derfor identisk i de forskjellige PLS'ene. Når programmet starter opp leses miljøvariablene inn og programmet tar hensyn til hvilken PLS (miljø) det kjører på.

Da alle objektene som er brukt er laget av en trestruktur som inneholder variabler, må alle disse variablene skrives/leses for hvert objekt. Da programvaren ikke vet noe om hvordan denne trestrukturen ser ut på forhånd, må den rulles gjennom. For å få til dette er det brukt en rekursiv metode. For lesing av variabler er metoden definert i kode 1.

Kode 1: Rekursiv metode for lesing av OPC UA variabler

```

1 def getValuesFromNodes (pObject, nodeStore):
2     for tagname, pObje in pObject.items():
3         if type(pObje) is dict:
4             getValuesFromNodes (pObje, nodeStore[tagname])
5         else:
6             pObject [tagname] =
7                 nodeStore[tagname].get_value()

```

Den rekursive metoden henter ut alle “top-level” objekter i linje 2. Objektene ligger i en hash-tabell, som kalles “ordbok/dict” i Python. Om verdien også er av typen “ordbok” kaller funksjonen seg selv. Valideringen gjøres på linje 3 og kallet gjøres på linje 4. Om *pObje* ikke er en ordbok vil den være en verdi, da hentes denne verdien fra OPC UA serveren, dette foregår på linje 6. Slik hentes alle verdier som hører til et objekt, hvor objektet er en trestruktur.

Som nevnt kreves det en restart av linken om det gjøres en endring på OPC UA serveren. For å automatisere denne restarten er det brukt en funksjon i e!COCKPIT programmet som heter “system hendelser”. Denne funksjonen gjør det mulig å kjøre egendefinert kode på en slik system hendelse. To hendelser er tatt i bruk; “På nedlasting av nytt program” og “Endring i program uten nedlasting”. På begge disse hendelsene settes en variabel `høy`. Denne variabelen leses av linken gjennom OPC UA. Er denne variabelen `høy`, restartes linken samtidig som variabelen settes lav. Restarten eksekveres enkelt ved å kaste ett unntak som vist i kode 2.

Kode 2: Restartlogikk kommando OPC UA - MQTT link

```

1 if restartCmdNode.get_value():
2     restartCmdNode.set_value(False)
3     raise Exception("PLC: Restart demanded from e!COCKPIT
    → environment")

```

Å kaste et unntak gjør at linken restarter. Dette fordi det er implementert automatisk omstart. Med dette menes det at om det oppstår en feil i programmet, vil programmet koble fra OPC UA og MQTT, deretter vente litt for å så koble til gjen. Funksjonaliteten er implementert med logikken i kode 3. “Try” kjøres ved normal eksekvering av kode. Kastes det et unntak, fanges det i “except” og koden der kjøres. Siden unntaket ikke kastes videre kan koden kjøre videre. Når koden i “except” har kjørt, kjøres koden i “finally”. All denne koden er implementert i en “while True”, som er en løkke funksjon som vil gjentas i det uendelige.

Kode 3: Restartlogikk OPC UA - MQTT link

```

1 while True:
2     try:
3         # initialisering
4         while True:
5             # hovedlogikk
6             # kaster unntak om det er behov for restart
7         except:
8             # logging av feil
9         finally:
10            # frakobling

```

I parallel med denne hovedlogikkken kjøres det en egen tråd for mottak og sending av meldinger på MQTT. Når hovedlogikkken skal publisere data på MQTT benyttes denne tråden. Når data mottas på MQTT behandles dette av en “callback” funksjon. Denne funksjonen er ikke lik i PLS og HMI. I HMI blir objekter tatt i mot og skrevet rett til OPC UA serveren. Bakgrunnen for dette er at det sendes kun data på endring, som vil si at dataen er ny å må inn til HMI. I PLS finnes det en svar funksjon i tillegg til vanlig mottak av data. Da PLS er eier av data er det HMI som spør

etter data og PLS svarer på forespørsel. Da dataen sendes kun på endring er det nødvendig for HMI å spørre etter data i det systemet starter opp. De siste settpunktene som ligger i PLS leses inn i HMI når den starter opp. Dette gjøres ved at HMI sender forespørsler til PLS om å få data. I disse forespørslene inneholder det objektet HMI'en ønsker, uten verdier. Så fort PLS mottar et objekt uten verdier sendes det objektet i retur med PLS'ens siste verdier.

Når data mottas, lagres det en ny hash av det nye objektet, i hash-tabellen. Da mottaket av data kjører i en annen tråd, er denne tabellen trådbeskyttet. Dette gjøres med kode 4. "HashLock" er av typen "Lock" som er en innebygget type i Python, laget for trådsikring. "With" er et nøkkelord i Python som brukes i forbindelse med ressurser, hvor ressursen lukkes når koden innkapslet i "with" er ferdig å kjøre. Dette gjør at "hashLock" låses før den nye hashen blir lagret, og låst opp når den er blitt lagret. Samme låsen brukes i begge trådene. Dette gjør at kun en tråd kan endre i hash-tabellen med hasher om gangen. Denne trådsikringen gjør at det ikke oppstår "race-conditions" og andre feil i forbindelse med bruk av flere tråder.

#### Kode 4: Trådsikring i OPC UA - MQTT link

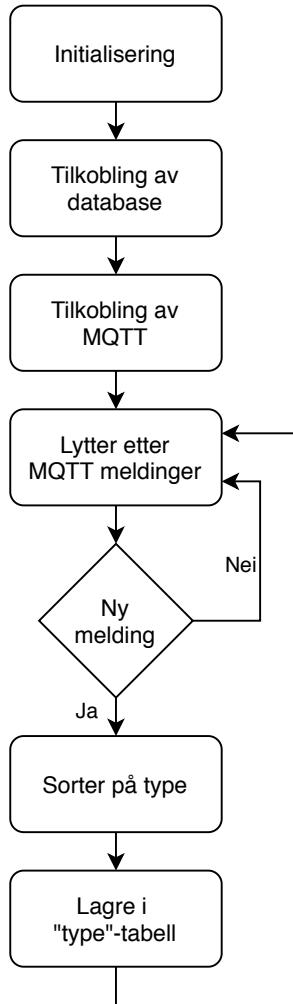
```
1 with hashsLock:  
2     hashs[receivedObject["_tagId"]] = newHash
```

Koden til begge programmene kan utforskes videre på Github [33] under "OPCUA\_MQTT\_Link".

### 3.6 Historisk data

For å kunne se på hendelser tilbake i tid, og dermed ha en formening om hvordan systemets tilstand vil være en periode inn i fremtiden, er all data lagret. Denne dataen er lagret i en database av typen MySQL. Denne måten å lagre data på er valgt da det er en standardisert metode. Dette gjør at det er mange klienter som kan hente ut denne dataen i etterkant. Dette tilrettelegger for videre analyse av dataen i fremtiden.

For å få dataen som sendes med MQTT inn i databasen er det laget en liten programvare. Denne programvaren inneholder en MQTT klient og en database klient. MQTT klienten abонnerer på all dataen som sendes ut av PLS'ene, deretter sender den dataen videre til database klienten som tar seg av å lagre dataen i databasen. Dataen lagres i flere tabeller, hvor hver tabell inneholder data fra objekter av samme type. Programmet er laget så enkelt som mulig og følger flyten i figur 6. Koden finnes på Github [33] under "DataStore".



Figur 6: Lagring av data fra MQTT til MySQL

For å visualisere historisk data i form av grafer, er en instans av Grafana satt opp. Denne instansen opererer som en slave i systemet hvor det eneste den gjør er å lese data fra databasen. Grafana er satt opp slik at den f.eks. lager historiske grafer av sensor data, hvor x-aksen er tid og y-aksen er verdi. Ved å visualisere data i slike grafer får man oversikt over hvordan systemets tilstand har utviklet seg, å på bagrunn av dette kan man forutse fremtiden. Om en sensor har økt sin verdi de siste to dagene, er det sannsynlig at den også vil øke den neste timen.

Den historiske dataen brukes i dette tilfellet til scenariet nevnt over, men også til å detektere lekkasjer i distribusjonen av drikkevann. Programvaren for lekkasjedeteksjon henter data direkte i databasen, samt skriver tilbake data.

### 3.7 Sky-server

For å kjøre de forskjellige programvarene som er nevnt i de forrige kapitlene er det satt opp en server i skyen, hos Google Cloud. Da prosjektet baserer seg

på å kjøre programvare i Docker, er oppsettet av denne enkelt. Det er satt opp en “Compute Engine”, denne fungerer som en helt vanlig datamaskin. Det er valgt et operativsystem av typen Ubuntu 18.04, da det kjøres uten lisens, samt at Docker og Docker-compose lar seg enkelt installere, med noen få kommandoer. De forskjellige programvarene er kjørt opp i Docker-compose, med konfigurasjonen i kode 5.

#### Kode 5: Docker-compose konfigurasjon for sky-server

```
1 version: '3.1'
2 services:
3     mqtttomysql:
4         image: jonev/mqtt-mysql-store
5         depends_on:
6             - db
7     simulation:
8         image: jonev/water-simulator
9         depends_on:
10            - db
11     leakdetection:
12         image: jonev/leak-detection
13         depends_on:
14            - db
15            - simulation
16     grafana:
17         image: jonev/grafana-with-plugins
18         ports:
19             - 3000:3000
20         depends_on:
21             - db
22     db:
23         image: mysql
24         command:
25             → --default-authentication-plugin=mysql_native_password
26         environment:
27             MYSQL_ROOT_PASSWORD: example
28     adminer:
29         image: adminer
30         ports:
31             - 8080:8080
32         depends_on:
33             - db
34     portainer:
35         image: portainer/portainer
36         command: -H unix:///var/run/Docker.sock
37         restart: always
38         ports:
39             - 9000:9000
```

```

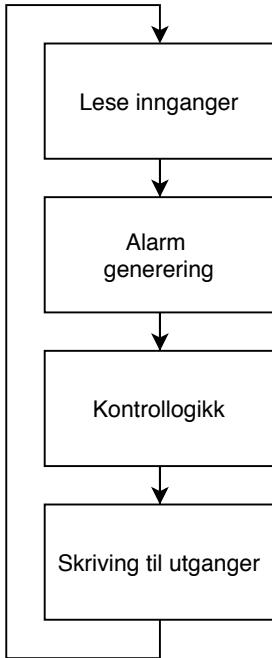
39      - 8000:8000
40  volumes:
41    - /var/run/Docker.sock:/var/run/Docker.sock
42    - portainer_data:/data
43 volumes:
44   portainer_data:

```

Konfigurasjonen er standard Docker-compose. Det er satt opp noen avhengigheter, slik at noen kontainere venter på at “db” skal starte før de starter opp. Disse avhengighetene er satt opp for å forhindre feilmeldinger i oppstarten. Det gir ingen mening å kjøre opp en programvare som skal hente ut data fra databasen før databasen kjører. Det er to tjenester som ikke er nevnt over, det er “adminer” og “portainer”. Adminer er en database klient, denne er i hovedsak brukt i utviklingssammenheng. Den brukes til å manuelt spørre etter data i databasen og få disse visualisert i tabeller. Portainer er et webgrensesnitt for å overvåke og administrere kontainere som kjører i Docker. Portainer brukes til å enkelt stoppe og starte kontainere, samt raskt hente ut loggfiler. Disse programvarene er ikke visualisert i figur 4 da de i hovedsak er brukt i utviklingssammenheng.

### 3.8 PLS-programarkitektur og struktur

Ved utvikling av arkitekturen og strukturen er det tatt utgangspunkt i problemstillingen som definerer at all programvare som utvikles skal være standardisert, modulær og gjenbrukbar. Det er derfor satt en standard som er fulgt i alle programmene. Denne standarden er i noen av programmene i overkant avansert, og dermed inneholder ikke alle delene like mye kode. På tross av dette er standarden fulgt, slik at utviklere kjenner seg igjen i koden, i alle deler av systemet. Standarden setter samtidig gode rammer for hvordan strukturen skal være om man skal utvide en eller flere deler av systemet. PLS-programmene er delt inn i forskjellige deler for å gjøre programmene mere oversiktlige. Delene følger en logisk tankegang som er representert i figur 7. Når et system skal styres gir det mest mening at man leser systemets tilstand, som vil si at man leser innganger til PLS'en. Deretter genererer man alarmer, som er avhengig av inngangene. Alarmene genereres før kontrollogikken, for å kunne handle på disse alarmene så fort som mulig. Deretter kjøres kontrollogikken, denne er avhengig av både inngangene og alarmene. Kontrollogikken bestemmer hvilke endringer som skal gjøres med systemet, f.eks. stoppe en pumpe og så videre. Når systemets status er bestemt av kontrollogikken skrives utgangene til systemet slik at statusen oppnås.



Figur 7: PLS programarkitektur

Når de fysiske inngangene leses, skrives de til det interne minnet i PLS'en. Når alarmgenereringen og kontrollogikken kjøres, leser de inngangene fra det interne minnet, og ikke direkte fra inngangene. Dette forsikrer at ikke noen innganger endrer seg i løpet av en programsyklus. Om f.eks. kontrollogikken hadde lest de fysiske inngangene direkte, kunne noen innganger endret seg midt i eksekveringen av logikken. Dette kunne fått konsekvenser som f.eks. to sylinder som skulle beveget seg samtidig, ender opp med å bevege seg på to forskjellige programsykuser. Dette får dog i mange tilfeller få konsekvenser, men i noen maskiner kan det få fatale konsekvenser. Da det i henhold til problemstillingen skal lages gjen brukbar kode, er dette tatt hensyn til. Det samme gjelder når det kommer til utganger. Alarmgenereringen og kontrollogikken skriver utganger til et internt minne, og rutinen for å skrive utganger skriver det interne minnet til de fysiske utgangene. Utenom rutinene som er visualisert i figur 7 finnes det en hovedroutine som kjører disse underrutinene, en rutine som initialiserer data, en rutine som tar seg av simuleringen av fysiske innganger og utganger, samt en rutine som mottar og sender data til en simuleringsprogramvare. Initialiseringrutinen er ikke tatt med i visualiseringen da den bare kjører en gang ved oppstart. I et virkelig anlegg ville denne kjørt når all data blir slettet, men siden dette er et demonstrasjonsanlegg, kjøres denne ved hver oppstart. Dette gjør at ved eventuelle brukerfeil, kan systemet gjøres strømløst, da vil alle innstillingene være tilbake til utgangspunktet. De to sistnevnte rutinene er ikke tatt med i visualiseringen da disse er spesifikke for denne applikasjonen siden fysisk utstyr simuleres. Simuleringskoden påvirker ikke arkitekturen, den er lagt helt på siden, for at arkitekturen skal kunne

brukes i et reelt system. I de fleste tilfeller vil koden i simuleringen kun skrive simulerte verdier til innganger, mens i f.eks. trykkregulering finnes det noe logikk for å simulere en prosess.

Dataen i de forskjellige PLS'ene er strukturert i følgende deler:

- HMI
- IO
- Local
- SIM
- Status

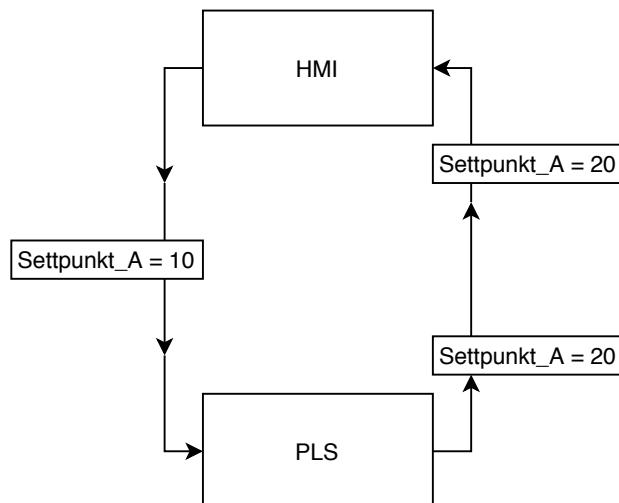
Mappen "HMI" inneholder all data som utveksles mellom PLS og HMI. Dette er data som gjør det mulig for HMI å sette alarmgrenser, starte/stoppe pumper samt motta prosessverdier som nivå i vann. Da man ønsker at settpunktene skal være de samme etter et strømbrudd eller en nedlasting av nytt program er disse taggene av typen "Persistent Retain". Dette er en innstilling som settes på dataen. Mappen "IO" inneholder data som representerer det interne minnet for innganger og utganger, som nevnt i forrige avsnitt. Denne dataen skal mappes direkte til fysiske innganger og utganger i rutinene for det. Mappen "Local" inneholder data som er lokal for den PLS'en man jobber i. I denne mappen lagres typisk funksjonsblokkinstanser for kontrollobjekter som pumper og så videre. Mappen "SIM" inneholder data som kommer fra simulatingsprogrammet, eller har med simulering å gjøre. Mappen "Status" holder intern data som forteller om status til systemet man er logget på.

For eksekvering av PLS-programmet er det satt opp to oppgaver. En oppgave som kjører hovedlogikken som nevnt i figur 7. Denne oppgaven kjører med prioritet 10 og på et syklusk intervall på 500ms. Det sykluse intervallet er oppgavens samplingstid som blir brukt i kalkuleringer, i f.eks. regulatoren. Da systemet representerer et veldig tregt system, er denne samplingstiden satt. Den andre oppgaven som kjører er MQTT kommunikasjon for å motta simulerte verdier. Denne oppgaven kjører med en prioritet på 15. Det vil si at kontrollogikken trumfer denne oppgaven. Kjører kommunikasjonsoppgaven, og kontrollogikkoppgaven trenger å kjøre, avbrytes kommunikasjonsoppgaven. Dette er satt opp slik da kontrollogikken må kjøre på fast syklus for å få korrekte kalkulasjoner samt kjøre systemet. Kommunikasjonsoppgaven er bare kommunikasjon og anses ikke som like viktig. Denne oppgaven er satt opp til å kjøre syklusk på fem sekunders intervall siden simulatingsprogrammet publiserer ny data på samme intervall.

Da modulær og gjen brukbar programvare er essensielt i dette prosjektet er det brukt objektbasert programmering så langt det lar seg gjøre i PLS-programmene. Dette innebærer at alt blir sett på som objekter og disse objektene består av tre komponenter. Hovedkomponenten er en funksjonsblokk som har tilkoblede innganger, utganger og kode.

Hovedkomponenten har to tilhørende datastrukturer som går til og fra HMI. Den ene strukturen inneholder alle prosessverdier og den andre inneholder alle settpunkt/verdier som HMI kan endre. Strukturene til og fra HMI er delt opp i to deler for å få til skrive funksjonalitet i både HMI og PLS. Med dette menes at både HMI og PLS kan endre ett settpunkt og det vil da oppdateres i den andre enden.

Uten å dele disse struktturene opp i to får man to utfordringer. Den ene utfordringen er visualisert i 8. Figuren illustrerer et scenario hvor HMI har endrett et settpunkt til 10. Utfordringen er at i den tiden dette blir kommunisert til PLS, har PLS allerede publisert to runder med data, som også inneholder settpunkter. Dette gjør at HMI tror at PLS har endrett settpunktet tilbake til 20. Den andre utfordringen er at man ikke enkelt kan detektere når et settpunkt har endrett seg å få HMI til å kun publisere på endring. Dette fordi prosessverdier finnes i samme struktur, og disse endres på hver syklus, da det er snakk om flyttall.



Figur 8: Ulempe med å ha settpunkt og prosessverdier i samme struktur

Biblioteket som er implementert består av følgende komponenter:

- AlarmDigital
- ControlAnalog
- MotorDigital
- PID
- Scaling
- SignalAnalog
- SignalDigital
- Timer

- WorkDistributor

Alle objektene med unntak av “PID” er utviklet i dette prosjektet. “PID” blokken er hentet inn fra et prosjekt tidligere i studiet. Noen av objektene, slik som “SignalAnalog” og “SignalDigital” har en unaturlig navngiving, dette er fordi typer som representerer nesten det samme skal legge seg etter hverandre på alfabetisk sortering. Objektene er brukt inni hverandre, f.eks. inneholder “AlarmDigital” en “Timer”, og “SignalAnalog” inneholder “Scaling”. Slik er alle objektene bygget opp av objekter som former en trestruktur. Denne objektbaserte tankegangen gjør at objekt, f.eks. en pumpe trenger tre tags, i steden for ett tag for hver funksjonalitet, slik som; pumpe har startet, pumpe har stoppet osv. Alle tag som er naturlig forbundet med objektet finnes inni objektet. Tag som kan være overflødig er normalt ikke tatt med, som f.eks. er ikke alarm for motorvern implementert i “ControlAnalog” da det ikke er sikkert at en liten ventil har overvåking av motorvernet.

I tabell 1 viser objektet for prosessverdiene til en digital motor som sendes til HMI. Som nevnt får man mye informasjon ut av et tag, slik som her, finnes alt fra om motoren er forhindrett i å starte (Interlocked\_Pv) til all informasjon rundt start og stopp alarmer.

<b>Tag navn</b>	<b>Tag type</b>
_tagId	STRING
_type	STRING
_owner	STRING
Interlocked_Pv	BOOL
ControlValue_Pv	BOOL
AlarmStartFailed	AlarmDigitalHmiPv
AlarmStopFailed	AlarmDigitalHmiPv
Started_Pv	BOOL
Stopped_Pv	BOOL

Tabell 1: MotorDigitalHmiPv objektstruktur

Som visualisert i tabell 1 har de forskjellige taggene i et objekt en gjennomtenkt navngiving. Det er tilstrebet å gjøre navnene så informative som mulig, men to forkortelser er ansett som nødvendig. Forkortelsene som er brukt er “Pv” for prosessverdi og “Cmd” for kommando. I tillegg finnes det tre tag som brukes til identifikasjon. Disse taggene starter på en understrek, for å indikere at de ikke kan endres og at de er systemtag. “\_tagId” er en identifikasjon som er unik for hver instans av et objekt. “\_type” inneholder hvilken type instansen er, som f.eks. “MotorDigital”. Dette tagget brukes til å mappe tagget til rett tabell i databasen for historisk data. “\_owner” inneholder hvilken PLS instansen hører til. Dette tagget forteller HMI hvilken PLS den skal spørre, om den mangler data på en instans, eller hvor et nytt settpunkt skal sendes.

For å sette en standard og gjøre det enkelt å jobbe med biblioteket, finnes det flere valg som har blitt tatt. Det er valgt ut et fåtall av de mange mulige datatypene som finnes i e!COCKPIT. Standardobjektene tar bevisst i bruk kun "String", "Bool", "Real", "Int" som representerer tekst, av/på, flyttall og heltall. Disse typene er ansett som de som er nødvendige for å få til all funksjonalitet. Dette valget gjør at man noen ganger bruker mer data enn det er behov for, men i dette biblioteket er brukervennligheten satt foran dataforbruket. Innad i objektene er det brukt andre datatyper. Denne standarden gjelder for dataen som flyter rundt i PLS og mellom PLS og HMI.

For at utviklere skal kjenne seg igjen er det konsekvent brukt strukturert tekst innad i objektene. Koden i de forskjellige blokkene følger også samme struktur, som følger listen under sekvensielt.

- Informasjon - Blokkens navn, hvem som har laget det, versjon, dato og beskrivelse.
- Inn rutine - Validering av f.eks. sett punkt for åpning av ventil i % ikke er over 100. Samt lesing av HMI variabler.
- Alarmgenerering - Alarmer som genereres inni blokken gjøres her.
- Kontroll - Alt av kontrollogikk som hører til for å få den funksjonaliteten som trengs.
- Ut rutine - F.eks. skriving fra interne variabler til HMI variabler.

Det er tilstrebet at denne strukturen er mest mulig lik arkitekturen som brukes i det overordnede programmet, nevnt i starten av dette kapitlet. Dette er igjen fordi brukere av biblioteket skal kjenne seg igjen og dermed skal koden bli enklere å forstå. I henhold til listen over er det tilrettelagt for validering av f.eks. sett punkter og alarmgrenser. Dette er implementert i liten grad. Det finnes noe validering av sett punkter, men mer omfattende og avansert validering er ikke blitt prioritert.

For implementasjonen av de forskjellige objektene refereres det til vedlegg A.4.

Alarm og alarmkvittering er håndtert innad i alarmblokken "AlarmDigital". Dette betyr at PLS'ene tar seg av å produsere alarmer, stoppe utstyr på grunn av alarmer og sørge for at ikke utstyr starter gjen uten at alarmer er kvittert. Det er to årsaker til at kvitteringen av alarmer ligger i PLS. Den ene fordelen er at man kan stoppe utstyr når en alarm går høy, og selv om alarmen forsvinner, starter ikke utstyret igjen. Alarmen må kvitteres før utstyret kan starte. Dette gjør at utstyr ikke starter etter en alarm, uten en aksjon fra en operatør. Den andre fordelen er at det er mulig å få synkronisert kvittering mellom flere skjermssystemer. Når kvitteringen ligger i PLS vil denne kunne oppdateres fra flere skjermssystemer uten ekstra logikk. Når det kommer til generell arkitektur, er det valgt at PLS skal eie all data, dette er også en årsak til at kvitteringen ligger i PLS.

Analoge alarmer er bygget opp av "AlarmDigital" og en tilhørende grense. Dette er gjort for å redusere antallet objekter, som igjen reduserer antallet

potensielle feil og potensielt hvor mange objekter som må bygges om ved endringer.

Da all data i biblioteket blir sett på som objekter, gjøres også dette for innganger. Alle innganger er derfor koblet rett til et objekt av typen “SignalDigital” eller “SignalAnalog”. Dette er gjort for å enkelt kunne simulere alle enkeltverdier. Denne typen simulering er implementert i signalblokkene og har en enkel oppgave. Fra et HMI tag kan man velge om objektet skal sende videre det fysiske signalet, eller en simulert verdi. Det finnes også et tag for å kunne sette denne simulerte verdien. Denne funksjonaliteten er nyttig når man f.eks. skal teste systemet. Man kan da f.eks. sette en nivåføler over i simulering, deretter legge inn den verdien man ønsker og dermed få den systemtilstanden man vil teste, som f.eks. hva som skal skje ved høyt nivå. Denne funksjonaliteten er også nyttig ved vedlikehold. Over tid blir følere utslitt, når disse skal byttes blir det brudd i et signal. Om dette signalet er kritisk for å holde prosessen igang, kan det være at en hel prosess stopper opp om man kobler fra dette signalet. Med simuleringfunksjonaliteten kan man enkelt sette signalet over i simulering og sette ønsket verdi, mens man bytter føleren.

Siden prosjektet resulterer i en demonstrasjonsrigg, er fullverdig objektsimplementering kun implementert i systemet for flomsikring. Dette betyr at det finnes noe redusert antall objekter i de andre systemene. Det innebefatter f.eks. at det ikke er implementert objekter for innganger og utganger. Dette er objekter som har som hovedfunksjon å skalere innganger og utganger. Når innganger og utganger er simulerte er ikke disse objektene kritiske og de er derfor utelatt. For en nærmere utforskning av implementasjonen av objekter henvises det derfor til systemet for flomsikring i vedlegg A.5.

### 3.9 Resultat

Nettverkstopologien med tilhørende dataflyt er sterkt avhengig av internett tilkoblingen og gir derfor varierende resultater. Kommunikasjon mellom PLS og HMI gir en gjennomsnittlig responstid på mellom 20 til 30 sekunder ved endringer i settpunkt. Publiseringen av data fra PLS avhenger også av CPU’ens ytelse på grunn av OPC UA - MQTT linken, samt datamengden. Da PLS’ene har varierende datamengder, varierer publiseringstiden fra 2 til 10 sekunder. Denne publiseringstiden definerer oppløsningen på den historiske dataen.

Historisk data er lagret i skyen. Dette resulterer i at systemets data er tilgjengelig fra hele verden, samt ingen begrensinger på lagringskapasitet og antall lesere av data. Er det behov for mer ytelse eller lagring, konfigureres dette enkelt opp.

PLS’enes kjøremiljø, arkitektur og struktur har resultert i en god arbeidsflyt hvor utviklere enkelt har arbeidet på tvers av de forskjellige delene av systemet. Ved endringer og utvidelser har det vist seg at strukturen fungerer bra, man utvider de forskjellige delene av programmet

enkelt ved å følge samme struktur. Koden har vist seg å være modulær og gjenbrukbar. Objektbiblioteket tilbyr bred funksjonalitet samtidig som det er enkelt å ta i bruk.

### 3.10 Diskusjon

På forhånd av prosjektet var det bestemt at all kommunikasjon mellom de forskjellige delene skulle gå via Wago Cloud ved bruk av MQTT. Når prosjektet var kommet igang og denne funksjonaliteten skulle settes opp viste det seg at Wago Cloud ikke kunne operere som en vanlig broker. Wago Cloud er designet for å koble seg til PLS'er, hente ut data og sende enkle kommandoer. Det er ikke laget for å sende data fra en PLS til en annen, slik en vanlig MQTT broker er [36]. Da dette systemet kjører en egen skjerm som henter inn data fra de tre delsystemene, er dette essensiell funksjonalitet for å kunne utveksle data. Det ble derfor besluttet å bruke en ordinær MQTT broker.

Kommunikasjonsmetoden kan betraktes som overkomplisert. Den inneholder mange komponenter som kan gjøre det vanskelig å holde oversikten. Det finnes mange alternative løsninger. Kommunikasjon over internett i stedet for LAN gjør hele kommunikasjonen mer kompleks. Man kan f.eks. ikke sette opp en direkte modbus kommunikasjon mellom komponentene. Dette er ikke mulig da komponentene har dynamiske IP-adresser. En mulighet er å sette opp en VPN mellom komponentene, for å få de på et og samme LAN, deretter bruke en av de integrerte protokollene. Det første som ble vurdert var å sette opp en OPC UA kommunikasjon. I utgangspunktet var dette en god ide, men ved videre utforskning, ble det funnet ut at både PLS'er og HMI har integrerte OPC UA servere. Serverene svarer på forespørsel om å få data, og dermed støtter de ikke server til server kommunikasjon. Det måtte i dette tilfellet blitt satt opp en OPC UA klient til klient applikasjon.

Direkte kommunikasjon med modbus TCP over VPN ble også vurdert. Dette hadde hvert en god løsning, men alle tag måtte da ha blitt mappet fra en tag verdi over til en adresse. I et system med mange tag er dette ansett som for mye arbeid. Ved mapping menes at man må skrive en kodelinje per verdi. Som et eksempel har man:

```
Motor1.Interlocked_Pv := ModbusAdresse1;
```

En slik mapping måtte man implementert for hvert tag i hvert objekt. I eksempelet er det brukt et tag fra "MotorDigital", de to HMI taggene som tilhører dette objektet har tilsammen 14 enkle tag. I tillegg har de fire tag av typen "AlarmDigital" som igjen har 11 enkle tag og to "Timer" tag. Timer taggene har til sammen 14 tag. Dette blir til sammen 39 tag, som vil si 39 linjer med kode. Dette gjelder for kun ett objekt. Totalt sett ville løsningen med å bruke modbus TCP kreve mye kode som mapper tag, og ble derfor ikke brukt.

Som nevt i tidligere kapitel, ble det laget en OPC UA - MQTT link. Under utviklingen av linken oppstod det flere komplekse utfordringer. Alt fra å

bygge Docker bilder som tok en time, bygge for rett prosessorarkitektur, til rett logikk, men den største utfordringen var ytelsen. Ved første utkast av programvaren fikk systemet en responstid på flere minutter. Programvaren kjørte da en kode som utforsket seg gjennom OPC UA tagstrukturen hver syklus. Dette gjorde programvaren kompatibel med endringer i strukturen. Om strukturen endret seg, ville disse endringene tre i kraft neste programsyklus. Dette ble ansett som en fordel, og ble derfor implementert i første omgang. I andre omgang ble det innført lagring av strukturen, slik det er nevnt tidligere. Ytelsen økte da til et akseptabelt nivå. Ytelsen er ansett til å være akseptabel, men det finnes muligheter for forbedring. Det kjøres nå en løkke for publisering av både prosessverdier og settpunkter. Det er mest kritisk å få publisert systemets status så ofte så mulig, det vil si at prosessverdiene blir publisert. For å potensielt øke ytelsen kan man kjøre to løkker, hvor en løkke vil kontinuerlig publisere prosessverdier. Den andre løkken kan kjøre sjeldnere og sjekke om settpunkter har endret seg, samt publisere de om nødvendig. Jobben med å sjekke settpunkter tas da bort fra hovedløkken og den kan potensielt få kjørt oftere.

Ett av målene med oppgaven var å finne ut hva som var mulig å få til med tanke på at Wago har åpnet opp for at utvikleren kan kjøre hva den vil på Linux operativsystemet. Det var i den forbindelse planlagt at operativsystemet skulle kjøre Docker og at man skulle kjøre all logikk i Python inne i en Docker kontainer. Dette ble startet på og det viste seg å være en del utfordringer med denne måten å kjøre systemene på, deriblant:

1. Lagring av tilstand.
2. Oppdatering av program uten å stoppe kjøringen.
3. Kommunikasjon med andre systemer.
4. Lesing og skriving av innganger og utganger.
5. Real-time kjøring.

De fleste utfordringene ble det mer eller mindre funnet akseptable løsninger for. 1. Lagring av tilstand ble løst ved at tilstanden til hvert objekt ble lagret i en database, som også kjørte i Docker, mellom hvert sample. Ved eventuelle feil eller strømbrudd ville programmet lese inn siste tilstanden fra databasen å fortsette kjøringen av programmet i den siste tilstanden det hadde før feilen oppstod. Denne løsningen fungerte bra, men PLS'en fikk noe redusert ytelse da den ble nødt til å kjøre et databasesystem i parallel med kontrollogikken, samt skrive til dette systemet hver programsyklus.

Punkt 2: Dette punktet kan høres irrelevant ut for dette systemet. Noen millisekunder stopp i vannforsyningen er som regel uproblematisk, men med fokus på gjenbrukbar programvare på tvers av bransjer og applikasjoner, ble også denne utfordringen utforsket. Det ble funnet en løsning hvor Python tar hånd om denne utforingen. Ved å lage et lite masterprogram som kjører et annet slaveprogram, var det mulig å bytte

ut dette slaveprogrammet uten at det ble stopp i systemet. Dette ble gjort ved at det nye programmet ble lastet ned til PLS'en, og når nedlastingen var ferdig, byttet masterprogrammet mellom å kjøre det gamle programmet og det nye, basert på at "sist lagret" tidstemplingen var nyere.

Punkt 3: Kommunikasjon med andre systemer ble løst ved at en MQTT klient ble implementert direkte inn i Python programmet som skulle kjøre kontrollogikken. Løsningen fungerte til tross for at man fikk en asynkron kommunikasjon.

Lesing og skriving av innganger og utganger, punkt 4, løste seg ved at et program i e!COCKPIT publiserte disse verdiene via den innebygde OPC UA serveren, deretter ble disse verdiene lest med en OPC UA klient i Python. Det ble først testet å bruke modbus TCP, men det viste seg å være noe tungvint da hver eneste variabel måtte mappes til og fra adresser. Til forskjell fra OPC UA hvor dette gjøres av seg selv både i server og klient. På serveren skjer dette i bakgrunnen av e!COCKPIT, mens på klienten er det implementert en rekursiv algoritme som går gjennom alle variabler og deres underliggende variabler å henter inn, eller skriver dataen. Det var ønskelig med en løsning hvor man ikke trengte noe program i e!COCKPIT, men det viste seg å være utfordrende å få tilgang til innganger og utganger uten. Skulle dette blitt gjort måtte man ha skrevet programmet i C++ og kjørt det direkte på Linux operativsystemet, og ikke i Docker. Docker introduserer et ekstra abstraksjonsnivå som kompliserer kommunikasjonen mellom kontrolleren og kontaineren. Da utgangspunktet var å bruke Docker og Python ble ikke denne C++ løsningen videre utforsket.

Punkt 4: Real-time kjøring viste seg å være den største utfordringen. Operativsystemet som kjører på kontrolleren kjører er av typen "Linux RT Preempt", som i utgangspunktet gjør det mulig å kunne kjøre programvare i real-time, men da abstraksjonslaget med Docker legges på blir det vanskeligere en først tiltenkt. I tillegg til mye forskning på utfordringen ble Wago support Norge kontaktet. De kontaktet Wago support i Tyskland, hvor det ble konkludert med at det ikke ville være mulig å kjøre en Python programvare inni en Docker kontainer i real-time.

På bakgrunn av konklusjonen i punkt 4 ble det besluttet at kontrollogikken skulle kjøres med e!COCKPIT. Løsningene som ble funnet i punkt 1-3 ble derfor lagt bort og funksjonaliteten e!COCKPIT tilbyr ble brukt.

For å få en kontinuerlig levering av nye Docker bilder, var det i utgangspunktet planlagt at disse bildene skulle bygges i skyen. Man ville da ha kunnet endre konfigurasjonen og fått sky-tjenesten til å bygge bildet. Siden bildene må bygges for prosessorarkitekturen som finnes på PLS'ene var ikke dette rett fram. Som nevnt, ble det brukt en funksjonalitet som fantes i eksperimentell versjon av Docker. Dette gjorde at støtten for å bygge disse bildene i skyen ble noe redusert. I henhold til ett blogginnlegg [42] skulle det være mulig å få til, men denne løsningen anses som midlertidig da den bruker den eksperimentelle versjonen og ble derfor ikke brukt. Bildene ble derfor bygget i det lokale utvikermiljøet.

Den historiske dataen samt de andre tjenestene som kjører i sky-serveren var først tiltenkt å kjøre på skjermen. Under utvikling ble de kjørt i skyen, for å frigjøre skjermen til utvikling av HMI. Dette gjorde at oppsettet av sky-serveren allerede var utført, samtidig som skjermen allerede hadde ytelses utfordringer med å motta alle signalene på OPC UA - MQTT linken. Det ble derfor, uten videre testing, besluttet at skyserveren ble værende som en komponent i systemet. Dette resulterte også i andre fordeler som offentlig IP adresse og mye maskinkraft.

Ved design av programarkitektur og strukturering av data oppstod det ikke noen store utfordringer. Oppdelingen av rutiner faller naturlig og det ble ikke vanskelig å dele opp koden slik at den passet inn i de forskjellige delene.

### 3.11 Konklusjon

Det konkluderes med at systemets kommunikasjon fungerer, og at responstiden er tilfredstillende med tanke på at dataen transporteres over internett. Dataflyten er kompleks, men det anses som nødvendig for å opprettholde systemets modulærbarhet som er spesifisert i problemstillingen. Oppdragsgivers kjøremiljø, samt kjøring av Python i Docker er brukt, men er ikke klar for kontrollogikk. Noen tilpasninger er gjort, men miljøene er ansett som vell utprøvd. Det er satt opp kontinuerlig integrasjon i den grad det er mulig for begge miljøene. Et funksjonsblokkbibliotek er utviklet, som har høy grad av modulærbarhet, samtidig som det ikke blir for komplisert å bruke. Det konkluderes derfor med at delene i dette kapitlet er levert i henhold til problemstillingen.

For videre arbeid finnes det flere utfordringer å forske videre på. Da PLS'ene kan kjøre C++ kode i Linux operativsystemet, ville det vært interessant å utforske hva dette innebefatter og hvor omfattende det er å få kjørt kontrollogikk i "real-time" med tilgang til lesing og skriving av innganger og utganger.

Det er bekreftet med WAGO support Tyskland at kontrollerene ikke er klar for å kjøre kontrollogikk i "real-time", i Docker. Det ville vært spennende å gått i dialog med leverandøren å prøvd å fått klargjort dette. Videre burde det blitt brukt tid på å øke ytelsen til OPC UA - MQTT linken, slik at den kan få en akseptabel publiseringstid i anlegg med mer data.

## 4 Simulerings

### 4.1 Introduksjon

Prosjektet innebefatter å lage et demonstrasjonssystem. Demonstrasjonssystem skal demonstrere ulike løsninger på problemstillinger i vann- og avløpsbransjen. Siden det ikke er ett reelt anlegg tilgjengelig med prosessverdier er disse verdiene simulert. For å simulere disse verdiene er det blitt valgt å gjøre dette hovedsaklig i Python.

Python er valgt som teknologien for å simulere verdier blant annet fordi det har støtte for diverse matematiske utregninger. Python har også flere ulike bibliotek som kan brukes som hjelpemiddel. Verdier der en lav samplingstid er essensiell, f.eks. for trykkregulering, er verdiene simulert i e!COCKPIT.

I dette kapittelet er det presentert hvordan simulering av verdier er løst. Kapittelet er delt opp i flere delkapittel, der hver del presenterer resultatet for sin simuleringsdel. Unntaket er kapittel 4.2, værmelding fra Yr, som ikke er simulerte verdier. Dette kapittelet er plassert her da verdiene fra værmelding brukes til å beregne andre simulerte verdier, samt at koden kjøres i samme Python program for å begrense kompleksiteten.

Teorien og formlene vist i dette kapittelet er hovedsaklig blitt brukt til å finne faktorer som kan brukes for lettere beregninger. I praksis brukes ikke disse formlene, men heller en faktor eller forhåndsberegnehede verdier ut ifra formlene. I utregninger er det blitt tatt noen antagelser og satt faste konstanter på noe som reelt kan variere. Det er ikke tatt hensyn til små variasjoner slik som f.eks. tap i kabel og temperaturforskjeller. Verdiene er ikke utregnet i dette kapittelet, men framgangsmåten er vist.

## 4.2 Værmelding fra YR

Oppdragsgiver ønsker å få vist værmelding for tre dager fremover i tid i brukergrensesnittet. Siden det finnes ett Python-bibliotek for å ta ned været fra yr.no, er det bestemt at værmeldingstjenesten Yr skal brukes. Etter at biblioteket [43] er lastet ned og installert, velges det hvilken type værmelding man vil ta inn, for hvilket sted. I dette tilfellet er det valgt værmelding for åtte dager og for Trondheim. Biblioteket henter ned dataen fra yr.no, via deres API. Deretter parser det dataen fra xml til json format.

Når json filen er hentet ned må man ta ut informasjonen man vil ha. Det er ønskelig å vise fram værsymbol, temperatur og millimeter nedbør for tre dager fram i tid, i brukergrensesnittet. Json filen inneholder en liste med en periode i hvert element. For å hente ut informasjonen man trenger brukes det en løkke til å gå gjennom listen og ta ut denne informasjonen. Værmeldingen er delt opp i fire perioder gjennom en dag, her trengtes bare periode tre, 12.00-18.00. Dette er valgt fordi det er også denne perioden som er vist i værmeldingen på langtidsvarslet hos yr.no. Deretter hentes det ut værsymbol, temperatur og nedbør i millimeter for de aktuelle periodene. Siden det finnes over 80 værsymboler i Yr, som er en del jobb å implementere i brukergrensesnittet, er det bestemt at det kun er nødvendig med seks symboler. Disse symbolene er; sol, delvis overskyet, overskyet, regn, kraftig regn og snø. Det er laget en funksjon i Python programmet som tar værsymbol som en tekst fra json filen og sjekker opp mot listen i programmet for å finne hvilket værsymbol i brukergrensesnittet de tilhører. F.eks. et værsymbol med tekst "Kraftig sludd og regn" omgjøres til "Kraftig regn".

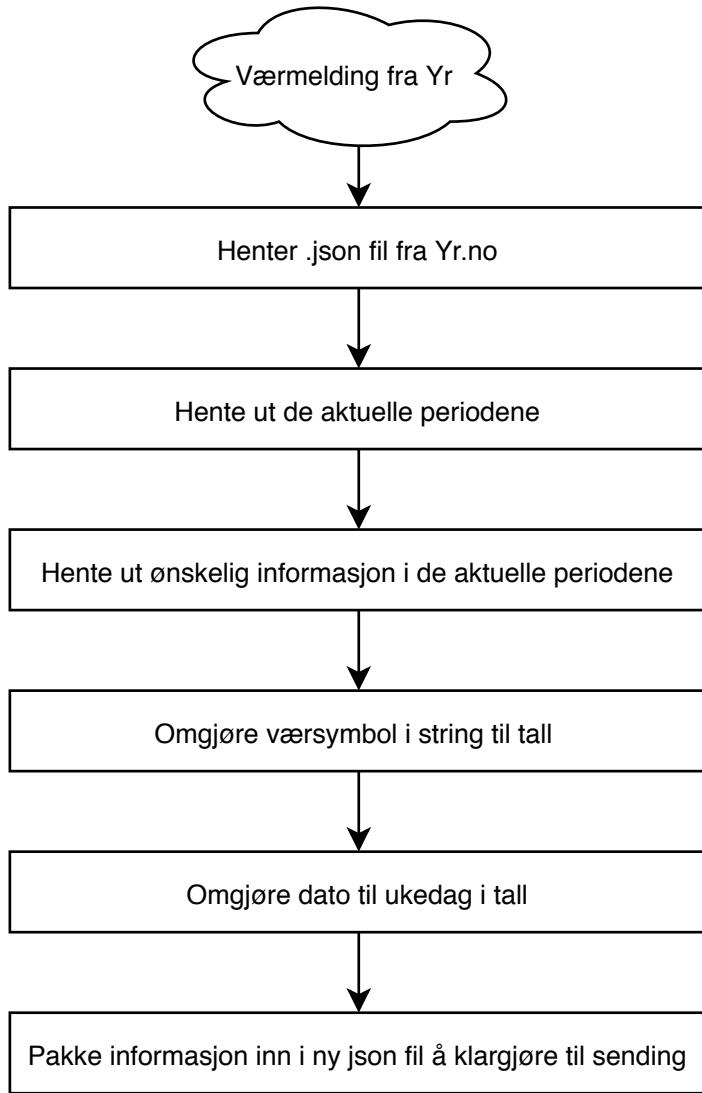
På grunn av enklere behandling av data med tall istedet for tekst, er det ønskelig å sende ukedag og symbol i tall. Dette gjøres i samme funksjon, ved å sjekke hvilken posisjon f.eks. "sol" har i listen. Siden "sol" er først i listen

får den tallet 0. Se venstre del av tabell 2. Fra json filen er det bare dato og tid i format ”DD-MM-YYYYTHH:MM:SS” som indikerer hvilken dag det er. Det er laget en funksjon som henter og splitter dato fra dato og tid teksten. Dato og tid sjekkes opp mot ett bibliotek i Python, ”datetime”, for å finne hvilken ukedag denne datoene tilhører. Funksjonen returnerer ukedagen i tall. Se høyre del av tabell 2.

Tekst	Tall	Tekst	Tall
Sol	0	Søndag	0
Delvis skyet	1	Mandag	1
Overskyet	2	Tirsdag	2
Regn	3	Onsdag	3
Kraftig regn	4	Torsdag	4
Snø	5	Fredag	5
		Lørdag	6

Tabell 2: Konvertering av tekst til tall for værmelding

Til slutt pakkes alt inn i en ny json struktur, med den informasjonen som skal sendes til brukergrensesnittet. Json strukturen klargjøres og sendes med MQTT hvert 10 sekund. For å få et bedre innblikk i programmets flyt, se figur 9.



Figur 9: Flytskjema for innhenting og sending av værmelding

### 4.3 Lekkasjedeteksjon

I distribusjonsnettet er det valgt å ha tre lekkasjedeteksjonsmoduler. Hver lekkasjedeteksjonsmodul inneholder ett batteri og ett solcellepanel. I et virkelig anlegg ville disse verdiene kommet fra strømmålere og en batterimåler. I dette delkapittelet er det vist hvordan disse verdiene er simulert. For å finne lekkasjer må man også ha en form for målinger på rørerene. I dette prosjektet er det valgt å bruke strømningsmåling som målemetode, det må derfor simuleres strømningsverdier i rør også.

#### 4.3.1 Solcellepanelet

Simulerte verdier fra solcellepanelet baserer seg på været og størrelse på solcellepanelet. For å vite hvor mye strøm solcellepanelet tilfører batteriet er det i dette tilfellet tatt hensyn til tre parametrer; type vær, hvor mye

effekt solcellepanelet produserer,  $P_{solcelle}$ , og hvilken spenning det kjører på,  $P_{solcelle}$ . Type vær er tatt inn på samme måte som i kapittel 4.2, "Værmelding fra YR", bortsett fra at her er det kun tatt inn værmelding for dagen idag. For å se hvor mye omtrentlig været påvirker solcellepanelet [44] bruker man tabell 3. Tekniske data på solcellepanelet er:  $P_{solcelle} = 100W$  og  $U_{solcelle} = 24V$ .

Værtype	Vær faktor ( $K_V$ )
Sol	1,0
Delvis skyet	0,8
Overskyet	0,5
Regn	0,3
Kraftig regn	0,0
Snø	0,0

Tabell 3: Værtype med værfaktor

Man beregner strøm ut av solcellepanelet,  $I_{ut,solcelle}$  ved og bruke effektformelen sammen med værfaktoren,  $K_V$ :

$$I_{ut,solcelle} = \frac{P_{solcelle}}{U_{solcelle}} \cdot K_V \quad [A] \quad (1)$$

#### 4.3.2 Batteri

I modulene er batteriet tilkoblet to strømningsmålere og en PLS. Det er forhåndsberegnet hvor mye disse enhetene trekker av strøm. Den forhåndsberegnet verdien er brukt som en konstant i simuleringssprogrammet. I dette tilfellet er den forhåndsberegnet verdien,  $I_K = 1A$ . Strømtrekket multipliseres med en tilfeldig verdi,  $R_{tilfeldig}$  mellom 0,95 og 1,05 for å få litt variasjon i den simulerte verdien. Strømmen ut fra batteriet er da simulert slik:

$$I_{ut,batteri} = I_K \cdot R_{tilfeldig} \quad [A] \quad (2)$$

Når man vet strømmen som går inn og ut fra batteriet kan man simulere batterinivået. Størrelsen på på batteriet er satt til 24Ah. For å vite energien som går inn og ut fra batteriet sees det på strømforbruk hver sample isteden for strømforbruk hver time. Sampletiden er  $T_S$ . Total samples per time:

$$N_S = \frac{3600s}{T_S} \quad (3)$$

Man kan deretter finne batterinivået,  $B_N$  ved å se på hvor mye strøm som er tilført og trekt ut av batteriet hvert sample. Det settes en startverdi,  $B_{start}$ , på batteriet på 16Ah. For første kjøring i simuleringssprogrammet er  $B_N$ :

$$B_N = B_{start} + \frac{I_{ut,solcelle} - I_{ut,batteri}}{N_S} \quad [Ah] \quad (4)$$

Etter første kjøring og utover er  $B_N$ :

$$B_N = B_{N-1} + \frac{I_{ut,solcelle} - I_{ut,batteri}}{N_S} \quad [Ah] \quad (5)$$

For å få verdiene batterinivået i prosent brukes det følgene utregning:

$$B_{N\%} = \frac{B_N}{B_{start}} \cdot 100 \quad [\%] \quad (6)$$

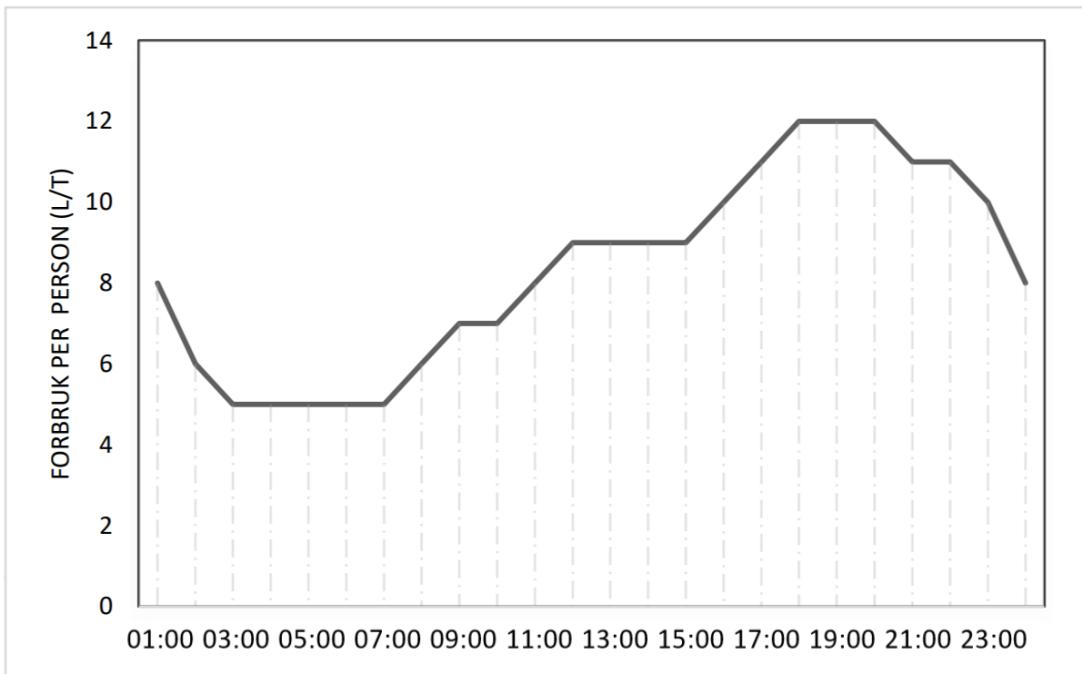
For å unngå at batterinivået skal få negative verdier eller verdier over maks nivå er negative verdier satt til 0 og verdier over makskapasitet satt til makskapasitet.

#### 4.3.3 Forbruk av vann

Strømningen av vann i rørene er avhengig av forbruket og eventuelle lekkasjer. For å lage simulerte verdier for forbruket er det brukt kilder fra et reelt vassverk og rapporter som omhandler forbruk av vann i Norge. I følge rapporten fra Norsk Vann [3] har hver person omtrentlig et forbruk på 200 liter per døgn i Norge.

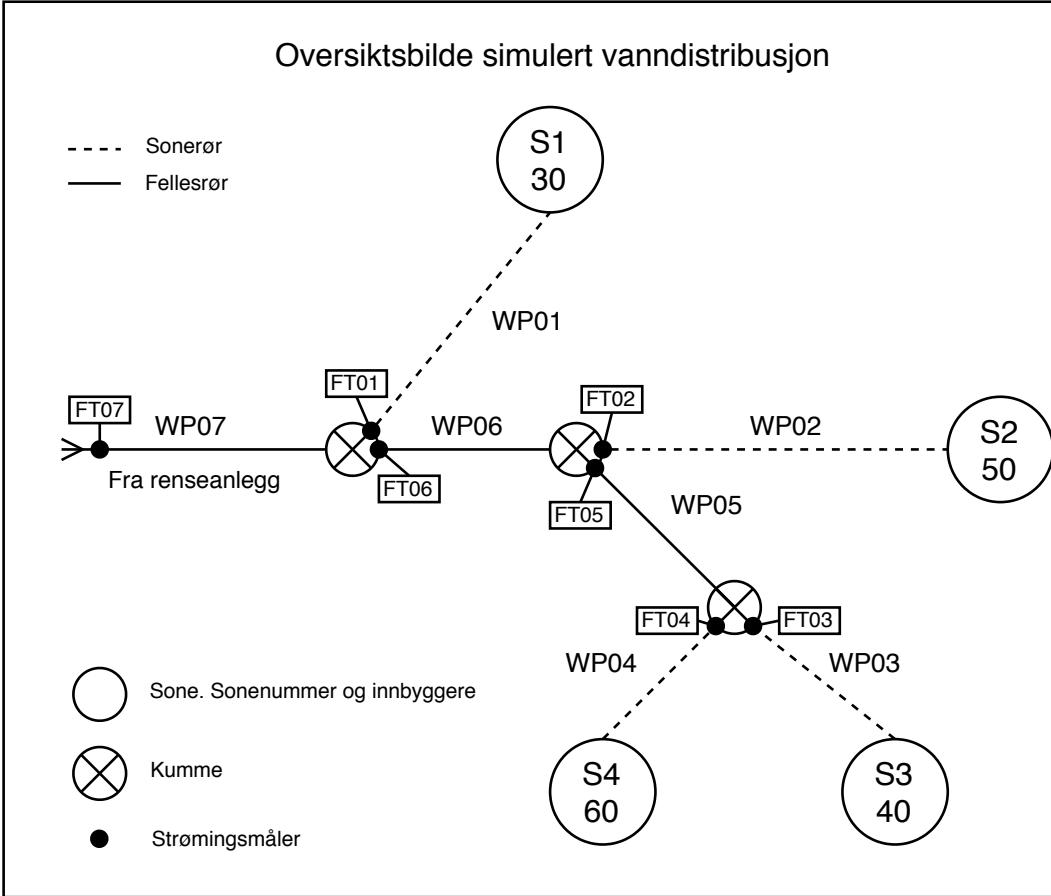
Leksvik Vassverk ga tilgang til en graf over et normalt forbruk for et boligfelt gjennom en dag. For å lage et normalt forbruk for en person for en dag er det satt opp en egen graf, se figur 10. Grafen er lik grafen som er tilsendt fra Leksvik Vassverk bortsett at verdiene på grafen samsvarer med forbruket for en person istedenfor. Det vil si at summen av grafen er omtrentlig 200 liter.

I praksis er verdiene delt opp i liter per time per person. Disse verdien legges i en liste i Python som har en lengde på 24, en plass for hver time. Python interpolerer verdiene mellom timesverdiene til ønskelig antall samples hvert døgn. Før verdiene sendes til PLS over MQTT, multipliseres de med en tilfeldig verdi mellom 0,95 og 1,05 for å få variasjoner.



Figur 10: Vannforbruk per person per time

Distribusjonsnettet inneholder flere strømningsverdier som skal simuleres. Så langt er det funnet strømningsverdiene for en person gjennom et døgn. Fra oversiktsbilde på figur 11 er strømningsmålerene vist ute i anlegget.



Figur 11: Virtuelt distribusjonsnett

Strømningsverdiene for strømningsmålere mot soner regnes ut slik:

$$q_{FT} = q_n \cdot N_{innbyggere} \quad \left[ \frac{l}{s} \right] \quad (7)$$

Der  $q_{FT}$  er strømningsverdiene i måleren,  $q_n$  er normal forbruk for en person og  $N_{innbyggere}$  er total innbyggere i sonen etter strømningsmålere.

Strømningsverdiene for målere som står på fellesrør regnes ut ved å se på utstrømningen av røret. I dette anlegget er det alltid to rør ut fra en kumme og ett inn. Dermed finner man strømningen slik:

$$q_{FT} = q_{ut_1} + q_{ut_2} \quad \left[ \frac{l}{s} \right] \quad (8)$$

Det er valgt å bruke  $\frac{l}{s}$  isteden for SI-enheten  $\frac{m^3}{s}$  fordi  $\frac{l}{s}$  gir bedre lesbarhet ved lave verdier. F.eks.  $5 \frac{l}{s}$  gir en raskere forståelse en  $0,005 \frac{m^3}{s}$ . Utregning for alle strømningsmålere er vist i tabell 4.

Strømningsmåler	Utregning
$q_{FT01}$	$q_n \cdot 30$
$q_{FT02}$	$q_n \cdot 50$
$q_{FT03}$	$q_n \cdot 40$
$q_{FT04}$	$q_n \cdot 60$
$q_{FT05}$	$q_{FT03} + q_{FT04}$
$q_{FT06}$	$q_{FT02} + q_{FT05}$
$q_{FT07}$	$q_{FT01} + q_{FT06}$

Tabell 4: Utregning av simulerte verdier i strømningsmålere

## 4.4 Flomsikring

### 4.4.1 Modellering av vannkilden

For å modellere vannkilden er det nødvendig å vite mengde inn og ut av vannkilden. Mengden ut kan man finne ved å bruke strømningsmålere, men mengde inn er vanskeligere å måle. Nedbøren og tilsiget er sett på som forstyrrelser på prosessen. Man kan finne ut hvor mye nedbøren og tilsiget påvirker prosessen ved å regne ut meter endring på nivået i vannkilde ved nedbør. Mengde nedbør i området kan man hente inn fra en værmeldingstjeneste. Endring av nivået følger volumbalanse [45, s.86].

$$\frac{dV}{dt} = q_{inn} - q_{ut} \quad \left[ \frac{m^3}{s} \right] \quad (9)$$

$$A \frac{dh}{dt} = q_{inn} - q_{ut} \quad \left[ \frac{m^3}{s} \right] \quad (10)$$

$$\frac{dh}{dt} = \frac{1}{A} (q_{inn} - q_{ut}) \quad \left[ \frac{m}{s} \right] \quad (11)$$

$$q_{inn} = q_{elv} + q_{tilsig} + q_{regn} \quad q_{ut} = q_{utslipp} + q_{forbruk} \quad \left[ \frac{m^3}{s} \right] \quad (12)$$

Forbruket og utsippet måles av strømningsmålere.  $q_{elv}$  regnes ut ved å multiplisere arealet av tversnittet på elva med farten i elva.

$$q_{elv} = A_{elv} \cdot v_{elv} \quad \left[ \frac{m^3}{s} \right] \quad (13)$$

For å finne  $q_{regn}$  og  $q_{tilsig}$  bruker man nedbørsmengde gitt av værmeldingstjeneste.

$$q_{regn} = A_{vannkilde} \cdot h_{regn} \cdot 1000 \quad \left[ \frac{m^3}{s} \right] \quad (14)$$

Her er  $h_{regn}$  mm regn per sekund og  $A_{vannkilde}$  er arealet av vannkilden.

$$q_{tilsig} = A_{tilsig} \cdot h_{regn} \cdot 1000 \quad \left[ \frac{m^3}{s} \right] \quad (15)$$

Tilsiget er beregnet ut ifra nedbørsområde  $A_{tilsig}$  og  $h_{regn}$ , mm regn per sekund. Man ender da opp med formel for nivået i vannkilden:

$$\frac{dh}{dt} = \frac{1}{A_{vannkilde}} (q_{elv} + q_{tilsig} + q_{regn} - q_{utsipp} - q_{forbruk}) \quad \left[ \frac{m^3}{s} \right] \quad (16)$$

Vannkilden som er valgt har spesifikasjoner lik tabell 5.

Areal	$22000m^2$
Dybde	$25m$
Nedbørsfelt	$180000m^2$

Tabell 5: Spesifikasjoner for vannkilde

#### 4.4.2 Økning i vannivå

Nivået i vannkilden er påvirket av flere faktorer, blant annet nedbør over og rundt vannkilden. Opprinnelig var det tenkt å bruke værmelding fra Yr som nedbøren i vannkilden, men dette viste seg å by på flere utfordringer. Siden tiden i flomsikring er simulert og ett minutt er lik en dag i virkeligheten ville værmeldingen fra Yr bare hatt data nok for syv minutter (syv dager). En annen utfordring er at det skal mye nedbør til for å få store endringer i vannivået. Det er heller ingen garanti for å få nedbør når man bruker værmelding fra Yr. Det er derfor bestemt at det skulle lages ett scenario for nedbør fem dager frem i tid, som gjentar seg.

Scenariet med nedbør er løst enkelt. Det er laget en liste med fem verdier, der verdien på en plassering i listen viser økning i vannmengde for en dag. Dag en på posisjon null i listen, dag to på posisjon en osv. Listen endrer seg hver dag. Dag en flyttes til dag fem, dag to flyttes til dag en, dag tre flyttes til dag to osv. Dette gjør at det er stadig endring i nedbøren. Listen er fem dager lang og ikke syv dager (en uke) fordi da vil det variere hver uke hvilken dag nedbøren kommer på. Tabell 6 viser endring i vannivået for hver dag. Disse verdiene baserer seg på formlene vist i kapittel 4.4.1.

Dag	1	2	3	4	5	
Økning vannivå	0	0	0.5	0.5	0	[m]

Tabell 6: Simulering av for økning i vannivå basert på nedbør

Verdiene som er satt inn er endring i vannivå ved ekstremt nedbør, da det er det som er av interesse når det jobbes med flomsikring.

Endringen i vannivået kommer fra nedbøren direkte på vannet, tilsiget fra nedbørsområdet og økning i vanntilførselen fra elver grunnet tilsig til elvene.

#### 4.4.3 Ventil

For at systemet skal oppføre seg mest mulig som et reelt system er det laget simulering av prosessen som finnes i ventilen. Med prosessen i ventilen menes det at det finnes en overføringsfunksjon mellom det man ber ventilen om å gjøre, å det den gjør. Om man ber ventilen om å åpne fra 0% til 50% vil ikke ventilen oppnå 50% åpning umiddelbart. Ventilen er en mekanisk komponent som trenger tid til å utføre en handling. I mange tilfeller er denne overføringsfunksjonen tilnærmet 1, men i dette tilfellet ser man for seg at dette er en fysisk stor ventil, som i tillegg kan stå med mye trykk på. Det er derfor simulert en førsteordens prosess, men tidskonstanten og forsterkningen er urealistiske da hele systemet kjøres i en fiktiv tid hvor et døgn er 60 sekunder.

### 4.5 Trykkregulering

#### 4.5.1 Trykkfall i distribusjonsnettet

Trykket pumpene holder vil få et trykkfall på grunn av forbruket i distribusjonsnettet. Dette trykkfallet må simuleres i forhold til forbruket som også er simulert. For å få det mest teoretisk riktig er det brukt formler og lærestoff fra fysikken. Under utregninger er det tatt noen antagelser siden man ikke har full kontroll på alle faktorene i formlene og det er snakk om et fiktivt anlegg. Trykktapet baserer seg på Bernoullis likning med tapsledd.

$$\frac{p_1}{\rho g} + \frac{1}{2} \cdot \frac{v_1^2}{g} + y_1 = \frac{p_2}{\rho g} + \frac{1}{2} \cdot \frac{v_2^2}{g} + y_2 + h_f + h_e \quad (17)$$

Det er ønskelig å finne  $p_1 - p_2$  som vil gi trykktapet. For enkelhetens skyld anser man at rørene ligger på lik høyde og at farten er lik i begge punktene og setter  $y_1 = y_2$  og  $v_1 = v_2$ . Trykktapet er da:

$$p_1 - p_2 = (h_f + h_e)\rho g \quad (18)$$

Tapsleddene  $h_f$  og  $h_e$  finner man med formlene:

$$h_f = f \cdot \frac{L}{D} \cdot \frac{1}{2} \cdot \frac{v^2}{g} \quad h_e = \xi \cdot \frac{1}{2} \cdot \frac{v^2}{g} \quad (19)$$

Tapskoeffisient,  $\xi$ , leses ut fra tabell i vedlegg B.2. For å finne friksjonsfaktoren,  $f$ , kan man bruke Moodys diagram. Diagrammet ligger ved som vedlegg B.1. For å avlese Moodys diagram trenger man Reynoldstall  $N_R$  og den relative ruheten. Disse finner man med følgende formler.

$$N_R = \frac{\rho v D}{\eta} \quad \text{Relativ ruhet : } \frac{\epsilon}{D} \quad (20)$$

Ruhet,  $\epsilon$ , leses ut fra tabell i vedlegg B.2. Fra Moodys diagram leser man av Reynoldstall på x-aksen og den relative ruheten på y-aksen på den høyere siden. Der de to linjene krysser hverandre kan man finne friksjonsfaktoren,  $f$ . Ved å bruke strømningsverdiene fra avsnitt 4.3.3 kan finne trykktapet. Omgjøring fra  $q \left[ \frac{l}{t} \right]$  som strømningsverdiene er oppgitt i til  $v \left[ \frac{m}{s} \right]$  som brukes i formlene gjøres slik:

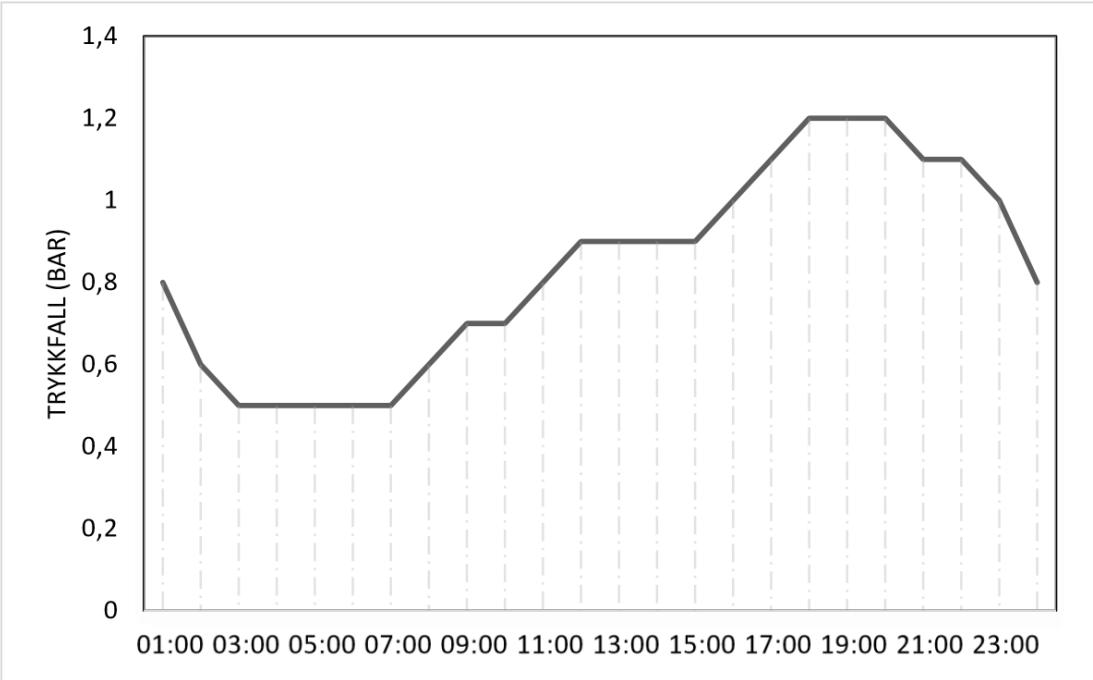
$$v = \frac{q \cdot 2,77 \cdot 10^{-7}}{\pi r^2} \quad \left[ \frac{m}{s} \right] \quad (21)$$

For dette tilfellet er det valgt å bruke tallene fra tabell 7.

Beskrivelse	Symbol	Verdi
Diameter rør	$D$	$0,15m$
Lengde rør	$L$	$20000m$
Ruhet plastrør	$\epsilon$	$0.01$
Tetthet vann	$\rho$	$997 \frac{kg}{m^3}$
Viskositet vann	$\eta$	$8,9 \cdot 10^{-4} Pa \cdot s$
Tapskoeffisient utløp	$\xi_u$	$1.0$
Tapskoeffisient bend $90^\circ$	$\xi_b$	$0.2$
Antall utløp		$180$
Antall bend		$180$

Tabell 7: Tallverdier for trykkfall

Etter utregning og tilnæringer er resultatet for trykkfallet lik figur 12. Det er valgt å legge disse verdiene i en liste og deretter kjøre gjennom denne listen for hver simulert dag for å finne trykkfallet for den aktuelle timen. Teorien ovenfor er brukt for å finne verdiene som legges i listen.



Figur 12: Trykkfall i distribusjonsnettet

#### 4.5.2 Strømtrekk på pumpe

Strømtrekket på pumpene kan måles med strømmålere i et reelt anlegg. Strømmålerne kan sende denne informasjonen til PLS'en og igjen vise den i brukergrensesnittet. Disse verdiene må her simuleres. Man kan finne effekten som må tilføres for at distribusjonsnettet skal få riktig trykk og strømning. Dette gjøres ved å bruke Bernoullis likning med tapsled. Det må legges til et ekstra ledd på venstresiden av likhetsteget,  $h_p$ . Dette ledet er løftehøyden pumpen må gi for at ligningen skal være lik på venstre og høyre side. Siden pumpen tilfører energi til systemet er  $h_p$  plassert på venstre side av formelen.

$$\frac{p_1}{\rho g} + \frac{1}{2} \cdot \frac{v_1^2}{g} + y_1 + h_p = \frac{p_2}{\rho g} + \frac{1}{2} \cdot \frac{v_2^2}{g} + y_2 + h_f + h_e \quad (22)$$

Det antas at  $y_1 = y_2$ ,  $p_1 = 0$  og  $v_1 = 0$ .  $p_2$  er ønskelig trykk og  $v_2$  er farten på vannet som blir forbrukt. Det er allerede forklart hvordan  $v_2$ ,  $h_f$  og  $h_e$  er utregnet i simulerte verdier for trykkfall, kapittel 4.5.1.  $h_p$  er dermed:

$$h_p = \frac{p_2}{\rho g} + \frac{1}{2} \cdot \frac{v_2^2}{g} + y_2 + h_f + h_e \quad (23)$$

Deretter finner man formelen for effekt som må tilføres [46, s.194].

$$P_v = \rho \cdot g \cdot q \cdot h_p \quad [W] \quad (24)$$

Effekten som tilføres en motor og deretter tilføres til pumpa blir ikke direkte overført til vannet. Det oppstår tap gjennom denne prosessen.

Virkningsgraden,  $\eta$ , forteller hvor godt prosessen overfører energien. For å finne effekten pumpen må tilføre divideres  $P_v$  med virkningsgraden for pumpa.

$$P_p = \frac{P_v}{\eta} \quad [W] \quad (25)$$

I dette prosjektet er det satt opp tre pumper som alternerer. Det er hele tiden to pumper som kjører mens den siste har pause. Effekten tilført av en pumpe er dermed:

$$P_{p_1} = \frac{P_p}{2} \quad [W] \quad (26)$$

Til slutt kan man finne strømtrekket ved å bruke effektformelen for trefase motor.

$$I = \frac{P_{p_1}}{\sqrt{3} \cdot U \cdot \cos\phi} \quad [A] \quad (27)$$

Her er  $U$  spenningen pumpen kjører på og  $\cos\phi$  er effektfaktoren som finnes på motorskiltet på pumpen.

Teorien ovenfor er brukt for å finne en faktor som brukes til å finne strømtrekket med hensyn på hastigheten på pumpen. Denne faktoren gjør at programmet slipper å ta store beregninger for hver sample, men multipliserer bare hastigheten på pumpa med faktoren for å finne strømtrekket. Dette er ikke helt teoretisk korrekt da ikke alle leddene er linære, men det vil være godt nok for et simuleringssprogram. Verdiene brukes bare for visualisering og ikke til regulering.

#### 4.5.3 Prosess

Av samme grunner som at ventilen i flomsikring er simulert, forklart i kapittel 4.4.3, er også prosessen i trykkreguleringen simulert ved bruk av en førsteordens funksjon. Pumpene og prosessen er implementert i en og samme overføringsfunksjon. Med prosessen menes det hele rørsystemet i distribusjonsnettet. Om man introduserer et sprang i trykket inn på nettet vil det være en overføringsfunksjon mellom det spranget man introduserer og det som oppleves hos forbrukeren, det er denne som er simulert. Av samme grunn som for ventilen, er ikke tidskonstanten og forsterkningen inkludert i rapporten.

### 4.6 Implementasjon i sky

De simulerte verdiene forklart i de forrige kapitlene, er produsert i en programvare skrevet i Python. Denne programvaren kjører i Docker, som igjen kjører i skyen. Programvaren kommuniserer med PLS'ene og HMI ved bruk av MQTT, samt produserer historisk data som legges rett i databasen.

Programvaren er implementert så modulær som mulig, og inneholder derfor følgende moduler:

- Værmelding - Henter værmelding og formaterer den

- Batterinivå - Kalkulering av batteriverdier og formaterer de
- Vann - Holder data om vann, som f.eks. vannnivå
- Nedbørsværmelding - Inneholder fem dager fiktiv værmelding om nedbør til bruk i flomsikring
- Vann-distribusjonsrør - Holder data om hvor mye strømning det er i distribusjonsrørene, samt et fiktivt scenario for lekkasje

Modulene tilbyr den funksjonaliteten som trengs for å tilfredsstille kravene til simulerte verdier, forklart i forrige kapitler.

For at programvaren skal få utført det den skal, kjører den først en initialisering, deretter tre periodiske oppgaver, som følger:

- Hovedoppgave
- Datasletting
- Henting og sending av værmelding

Initialiseringen kobler MQTT klient-instansen til MQTT brokeren. Deretter opprettes instanser av de forskjellige modulene. Når dette er gjort, er programmet klart for å starte, men da alle systemer kjører på tid, venter programmet med å starte til sekundtelleren er på 0. Dette gjøres fordi ett døgn er 60 sekunder, og ved å starte på sekund 0 vet programmet hvilken time det er, ut i fra sekundtelleren. Deretter starter de tre periodiske oppgavene.

Om det skulle oppstå feil i tilkobling til MQTT eller andre feil, er det ønskelig at programvaren venter litt og deretter prøver igjen. Det er derfor implementert samme restart logikk som i OPC UA - MQTT linken i kode 3. Deloppgavene avsluttes i "finally" og restartes igjen i "try" blokken.

Da systemet skal kjøres i demonstrasjonsammenheng kjøres hele systemet i en fiktiv tid. Dette innebærer at et døgn er satt til 60 sekunder i systemet. Dette er fordi tilskuere av systemet skal kunne få demonstrert flere døgn i løpet av tiden de har tilgjengelig. Det er satt opp ett lekkasje scenario som inntreffer hvert 3. døgn, altså hvert 3. minutt, slik at tilskuere skal kunne se systemet i aksjon.

Hovedoppgaven tar seg av den faktiske simuleringen. Oppgaven kjører hvert 5. sekund, som resulterer i hver 2. time, i systemets tidsrom. Hovedoppgaven har i hovedsak fire deloppgaver; beregning av nivå i vann, beregning av strømning i distribusjonsnett, produsering av historisk data for strømning, samt beregning av batteri-nivåer for lekkasjedeteksjonen. Beregningene gjøres ved å kalle på metoder i objektene som er forklart over. Produseringen av historisk data skriver data direkte i databasen. Denne dataen går ikke via PLS siden kommunikasjonen er for treg. Ytelsen er presset til fire måleverdier i timen. En måleverdi per 15 minutt, som blir fire per 2.5 sekund. Kommunikasjonen er for treg til at det kan sendes en verdi per strømningsmåler til PLS'en, og den igjen skal publisere den. Da

hovedoppgaven kjører hvert 5. sekund, produseres det åtte måleverdier per måler. Det finnes syv målere, dette resulterer i 56 måleverdier, som utgjør 56 rader i SQL databasen. For å få denne ytelsen settes syv verdier inn i databasen om gangen, i en “batch” operasjon, åtte ganger. Måleverdiene som produseres tidsstemples med lik fordeling på de fem sekundene, slik at disse opptrer som om de var reelle. Denne oppløsningen er nødvendig for at lekkasjedeksjonsprogrammet skal kunne generere gjennomsnittsverdier for hver time.

Oppgaven “Datasletting” tar seg av å slette data som er eldre enn den dataen som blir brukt av systemet. Da ett minutt er ett døgn, blir det produsert mye data av hovedoppgaven. Det er derfor behov for å slette data. Det vil også oppstå gammel data om man flytter systemet fra en messe til en annen messe, som man ønsker å slette.

Oppgaven “Henting og sending av værmelding” kjører en gang i minuttet, noe som blir en gang i døgnet. Oppgaven henter inn værmeldingen fra yr.no og sender den til skjermsystemet. Da værmeldingen blir oppdatert relativt sjeldent, kjører ikke denne oppgaven ofte.

Simuleringsprogrammets oppgaver kjører som nevnt på tids-intervall. For å forsikre at oppgavene kjører på intervallet som er ønsket er det implementert en egen oppgavebehandler. Intervallene kunne ha vært implementert ved å innføre en tidsforsinkelse. Utfordringen med kun en tidsforsinkelse er at programvaren kjører på et intervall som er  $eksekveringstid + forsinkelse = intervall$  og da eksikversingstiden varierer, vil det si at oppgaven vil kjøres på forskjellig intervall hver gang. For å unngå dette ble det implementert en oppgavebehandler. Det er flere bibliotek som tilbyr denne funksjonaliteten, men gjengangeren med disse bibliotekene var at de ikke utførte noen som helst handling om intervallet ikke ble overholdt. Det vil si at eksekveringstiden overstiger intervalltiden. Det ble derfor laget en enkel oppgavebehandler selv.

Oppgavebehandleren er laget så enkel som mulig. Funksjonaliteten den tilbyr er å kjøre en metode på et gitt tidspunkt rundet av til intervallet. Med dette menes det at om intervallet er på 10 sekunder, kjører oppgavebehandleren oppgaven når klokken er 00:00:10, 00:00:20 osv. Den starter ikke oppgaven på sekund seks f.eks. I tillegg kan man spesifisere en forsinkelse. Et forsinkelsen på f.eks ett sekund og intervallet på 10 sekunder, kjører oppgaven på 00:00:01; 00:00:11, osv. For å være i stand til å utføre en handling når eksekveringstiden er lengre en ønsket intervall, er det brukt en tråd som sjekker om dette har skjedd. For at denne tråden ikke skal kreve for mye ressurser, er det mulig å justere hvor ofte denne sjekker. Dette er gjort da en oppgave kan f.eks. kjøre hvert 60. sekund, og da er det ikke nødvendig å sjekke om eksekveringstiden har oversteget intervalltiden så ofte som for en oppgave som kjører f.eks. hvert 2. sekund.

#### Kode 6: Oppgavebehandler hovedlogikk

```
1 while self.__runflag:  
2     if self.__starttime < time.time():
```

```

3      raise Exception("Last task was not done ->
4          ↳ increase the interval")
5  while self.__starttime > time.time():
6      time.sleep(self.__checkIntervalInSeconds)
7  worker = threading.Thread(
8      target=self.__task,
9          ↳ args=(datetime.fromtimestamp(self.__starttime), )
10 )
11 self.__starttime = self.__starttime +
12     ↳ self.__runIntervalInSeconds
13 worker.start()
14 worker.join()

```

Hovedlogikken til oppgavebehandleren viser i kode 6. Denne koden kjører i en egen tråd som er kalt starter-tråden, da den tar seg av å starte oppgaven på de gitte intervallene. Koden kjører i en løkke, som kun avbrytes av kjøre-flagget. Dette flagget kan settes lavt fra hovedtråden, for å kunne stoppe oppgaven i forbindelse med å stenge ned hele programmet, eller restarte det. Den første starttiden kalkuleres før man kommer inn i denne løkken, dermed går den første “if” sjekken på linje 2 forbi når oppgaven skal startes for første gang. Deretter kjøres eksekveringen inn en i en “while” løkke, på linje 4, hvor eksekveringen venter til det er på tide å starte oppgaven. Eksekveringen venter så lenge oppgavebehandleren er konfigurert til å vente, før den igjen sjekker om det er på tide å starte oppgaven. Når tiden er inne, tilegnes oppgaven til en arbeidstråd på linje 6. Deretter kalkuleres det når oppgaven skal kjøres neste gang. Arbeidstråden startes og starter-tråden venter til arbeidstråden er ferdig med oppgaven. Dette gjøres ved en “join” på linje 11. Når arbeidstråden er ferdig med å utføre oppgaven slås den sammen med starter-tråden. Deretter er starter-tråden klar for å starte oppgaven på ny, men først sjekker den om den allerede skulle ha startet oppgaven. Skulle den det, kastes det et unntak, som gjør at programmet krasjer. Dette gjøre at man kan være sikker på at man får beskjed om oppgavene ikke kjører på det intervallet man ønsker. Unntaket er om oppgaven aldri fullføres første gangen den kjører, da vil man ikke få beskjed, men da vil aldri oppgaven utføres heller, noe som gjør at det er enkelt å oppdage. Om ikke oppgaven burde ha startet, venter starter-tråden igjen i “while” løkken på linje 4.

## 4.7 Implementasjon i PLS og HMI

Implementasjonen som finnes i PLS består av to rutiner. En som mottar og sender verdier på MQTT og en som har simuleringslogikk. For HMI finnes bare delen som mottar verdier. Koden er implementert ved bruk av WAGO sine biblioteker. “WagoAppCloud” tar seg av kommunikasjonen og “WagoAppJSON” tar seg av å gjøre om dataen som blir mottatt som en tekst, over til tags. Koden er hentet fra Kurt Braun sin video [47], men noen justeringer er blitt gjort for å få den til å passe med tagstrukturen.

Simuleringslogikken avhenger hvilken PLS man er i, men samme prinsipper om at fysiske innganger og utganger skal simuleres, er brukt. PLS 1 for lekkasjedeksjon har logikk som flytter alle simuleringsverdier til tag som er tiltenkt å knytte rett til fysiske innganger. Koden er derfor 16 linjer med flytting av verdier. En linje for hver strømningsmåler, samt verdier for batterinivå og strømmer fra solcellepaneler.

PLS 2 for trykkregulering har man mapping av pumpe hastigheter, men det er også implementert en førsteordens prosess med forsinkelse og forstyrrelse. For å simulere prosessen er det brukt en funksjon for den deriverte, hvor den er approksimert med Eulers metode [48]. Denne metoden og koden for den, er hentet fra et prosjekt tidligere i studiet. Forsinkelsen er simulert ved å utsette pådraget før det når prosessen. Dette gjøres ved å skrive pådraget inn i en tabell, hvor det leses ut x programsykuser senere. Implementasjonen er representert i kode 7. På linje 1 beregnes prosessverdien hvor pådraget inn på  $u$  er forsinket i forhold til verdien pumpene gir ut på linje 4. Linje 2 og 5 brukes til å rulle gjennom tabellen med pådragsverdier. ”MOD” er en systemfunksjon for modulus regning. Ved å ta modulus av indeks verdien forsikrer man at den alltid er innenfor størrelsen på tabellen. Selv når indeksen ruller over vil modulus funksjonen sørge for at den er innenfor tabellens størrelse.

#### Kode 7: Trykkregulering: Simulering av forsinkelse

```

1 SIM.Process_y := process(u := _simDelay[_idelay MOD 19],
  ↳ Ts := 0.1, y := SIM.Process_Y, yMin := 0, yMax := 8.0,
  ↳ T := 5, Kp := 1.0);
2 _idelay := _idelay + 1;
3
4 _simDelay[_iprocess MOD 19] := ((IO.AI_DeliveryPump1SpeedPv
  ↳ + IO.AI_DeliveryPump2SpeedPv +
  ↳ IO.AI_DeliveryPump3SpeedPv) / 10);
5 _iprocess := _iprocess + 1;

```

Forstyrrelsen består av en liste med timesverdier som tilsvarer et normalt trykkgfall i løpet av et døgn. For å gjøre implementasjonen lettere, er denne listen utvidet til 30 verdier. Det er da mulig å styre forstyrrelsen med sekundteller. Dette er gjort slik at f.eks. time 10 havner på samme plass hvert døgn. Samtidig som man ikke får forskyvning av døgnet mellom denne simuleringen og resten av systemet. Da et døgn er 60 sekunder, er sekundtelleren delt på to med heltallsdivisjon hvor resultatet brukes til å rulle gjennom listen med forstyrrelser. Denne logikken er implementert i kode 8. På linje 2 hentes sekundtelleren ut, på linje 3 deles den på to og på linje 4 brukes den til å hente ut rett verdi fra listen med forstyrrelser.

#### Kode 8: Trykkregulering: Simulering av forstyrrelse

```

1 timeNow := TIME();
2 secondsNow := (TIME_TO_LWORD(timeNow) / 1000) MOD 60;

```

```

3 fakeHour := secondsNow/2;
4 SIM.Disturbance := pressureDisturbance[fakeHour];

```

PLS 3 for flomsikring har litt flytting av data, samt simulering av prosessen som finnes i ventilen for utslipp av vann. Denne er simulert på samme måte som trykkreguleringsprosessen, bare uten forsinkelse og forstyrrelse. Prosessen er simulert i programmet i skyen, kommunikasjonen introduserer da en forsinkelse siden det tar tid for dataen å komme frem, samt høy syklustid i programvaren. Forstyrrelsen er også simulert i skyen i form av regn og kommer på samme kommunikasjonsform.

## 4.8 Resultat

Simuleringsprogrammet leverer simulerte verdier hvert 5. sekund via MQTT. Verdiene er basert på virkeligheten i henhold til kalkulasjoner gjort i de forrige kapitlene. Programmet er implementert i Python, og kjøres via Docker. Fra programmet sendes det følgende verdier til PLS'ene:

- PLS 1 Lekkasjedeteksjon: Verdier for strømmninger i rør, FT01-07. Verdier for batterier og solcellepanel, CI01-03, CO01-03, BL01-03.
- PLS 3 Flomsikring: Vannivå, PT01. Utløpstrømning, FT08, og værmelding for styring av ventil.

I tillegg mottar programmet ventilåpningen fra PLS 3 flomsikring.

## 4.9 Diskusjon

Det ble brukt en god del ressurser på å finne teoretisk korrekte simulerte verdier. Dette er i hovedsak utført slik at systemet skal ha en tilhørighet til virkeligheten. Med dette menes det at verdiene ikke har blitt tatt ut i fra ingenting, men at det en viss grad av virkelighet i dem. Ved å gjøre dette har det gjort jobben med innregulering enklere.

Under implementasjonen av simulerte verdier oppstod det flere utfordringer. Det var i utgangspunktet tenkt at det skulle være mulig å justere hvor lang tid et døgn skulle være. Da flere av systemer kjører på times verdier oppstod det et behov for å synkronisere døgnet mellom systemene. Det ble da besluttet at et døgn skulle være 60 sekunder. Så lenge klokken er synkronisert vil da alle systemer ha døgnet synkronisert. Det hadde vært mulig å fått synkronisert døgnet etter en endring i hvor lang tid et døgn er, men ved å veie jobben med å få dette til, mot behovet for funksjonaliteten, ble utbyttet for lite.

For å holde simuleringslogikken adskilt, kjøres mye av denne i skyen. I dette tilfellet introduserer dette en forsinkelse i kommunikasjonen. Denne forsinkelsen i sammen med at ett døgn er 60 sekunder introduserte en annen utfordring. I noen tilfeller holder det å få simulerte verdier hvert 5. sekund, slik som i flomsikring. Ved trykkregulering derimot, er dette for sjeldent, da det utgjør at man får verdier hver 2. time i systemet

sitt tidsrom. Trykkregulering har endring i forstyrrelsen hver time, og bruker timen på å hente seg inn. Det er da for sjeldent å få oppdaterte verdier hver 2. time. Dette ble løst ved å implementere denne simuleringen lokalt i PLS'en. Dette er en god løsning rent ytelsesmessig, men løsningen introdusere mer kompleksitet til systemets helhet. Løsningen gjør at det finnes simuleringslogikk både i skyen samt lokalt. Målet var å holde simuleringslogikken samlet i skyen, men dette lot seg ikke gjøre.

## 4.10 Konklusjon

De simulerte verdiene konkluderes til å være av tilfredstillende nøyaktighet. Implementasjonen av simuleringen er i overkant kompleks, men er ansett som nødvendig. Systemene har løse koblinger mot simuleringen, det er derfor enkelt å ta i bruk programvaren i systemene i et anlegg hvor det skal brukes fysiske sensorer og signalgivere. Ved å sette en fast verdi på 60 sekunder til å være et døgn, mister man muligheten til å stille på denne. Dette er funksjonalitet som ville vært nyttig å ha i demonstrasjonssammenheng, men det er konkludert med at tilskuerne har fire minutter å bruke. De vil da få med seg det aller meste av systemet funksjonalitet. Simuleringen gjør at systemet opptrer som om det skulle være et operativt system. Det konkluderes derfor at det leveres simulering i tråd med problemstillingen.

## 5 Lekkasjedeteksjon

### 5.1 Introduksjon

Lekkasjedeteksjon er et område innenfor vann og avløpsbransjen som har fått stadig økende fokus de senere årene. Norge er et land med god tilgang til store mengder rent vann per person. Overfloden av vann har gjort at leting etter lekkasjer ikke har blitt prioritert. Dette har gjort at man over tid har fått et etterslep innenfor utbygging og vedlikehold av vannforsyningens nettet [49]. Den stadige utviklingen av teknologi har gjort at man i den senere tid har fått bedre oversikt over mengden vann som forsvinner mellom vannkilden og forbrukeren, samt muligheter for å finne lekkasjene.

Da oppdragstakere sammen med oppdragsgiver var på besøk til Leksvik Vassverk ble det diskutert rundt hvilke utfordringer vann- og avløpsbransjen har. Etter besøket hadde oppdragstaker og oppdragsgiver et internt møte der det ble besluttet at lekkasjedeteksjon var en god og relevant oppgave å se på. I ettertid har kontakten med Borgar Berg, driftsleder ved Leksvik Vassverk SA fortsatt. Han har gitt tilgang til ressurser slik at man kan legge et kunnskapsgrunnlag rundt problematikken lekkasjedeteksjon.

Ressursene som er tilsendt fra Borgar er kunnskap rundt oppbygningen av ledningsnettet og forbruket av nettvann i Leksvik. Grunnet dette er det besluttet at oppgavene er begrenset til anlegget i Leksvik eller anlegg av tilsvarende størrelse.

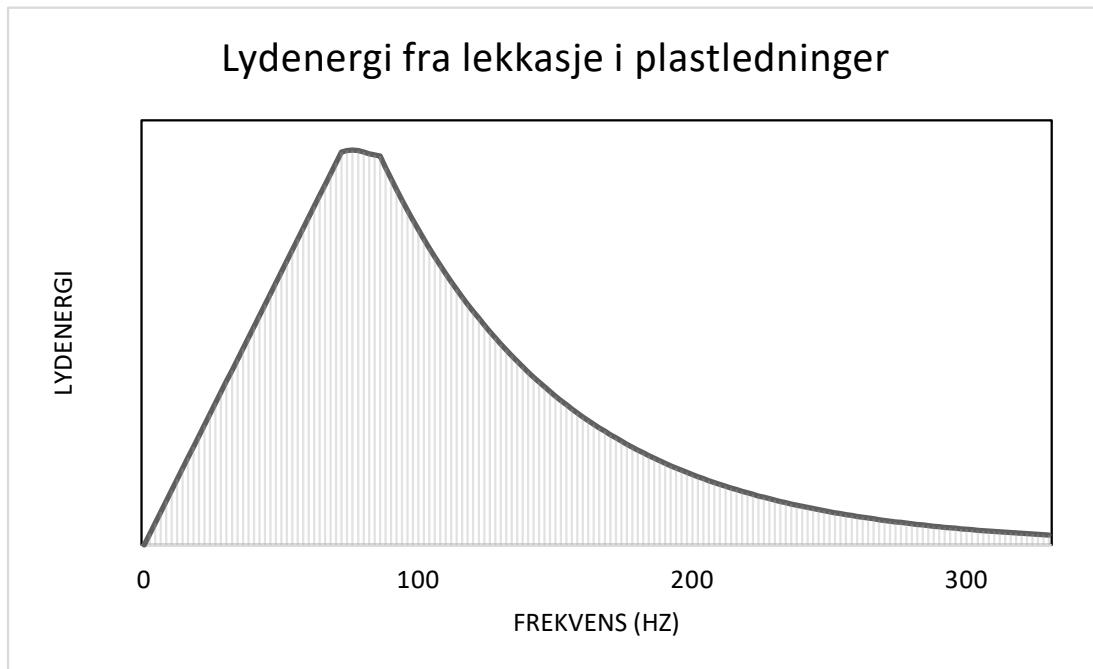
Gjennom prosjektperioden har gruppen jobbet med problemstillinger tilknyttet deteksjon av vannlekkasjer. Dette kapittelet tar for seg arbeidet som har blitt gjort innenfor lekkasjedeteksjon som valg av måleprinsipp, valg av deteksjonsmetoder som bygger på måleprinsippene samt generelle undersøkelser i forbindelse med miljøer utstyr skal plasseres i på et reelt anlegg.

### 5.2 Målemetoder

#### 5.2.1 Teori rundt vannlekkasje

Når det oppstår en vannlekkasje, skapes det vibrasjoner og det forsvinner vann. Vibrasjonene forplanter seg i vannet, så vel som i vannrøret [1]. Når det gjelder rørets evne til å lede lyd, er det stor forskjell ut fra hvilke materialer røret består av. Metallrør virker bra som lydleder, mens rør laget av plast materialer er tilnærmet lyddøde. Ved lekkasjer på metallrør skapes det høyfrekvent lyd i materealet, mens det i vannet forplanter seg lavfrekvente vibrasjoner. På den andre siden, i plastrør, vil det være tilnærmet ingen forplantning av vibrasjoner i veggene på røret, mens i vannet vil det dannes lavfrekvente vibrasjoner. Figuren 13 viser lydbildet til en vannlekkasje på et plastrør. Her kan man se at lydsignalet ligger i området omkring 75 Hz. Ved lekkasjer på metallrør og eternittrør vil kurven til lydbilde ha tilsvarende form, mens her vil lydsignalet henholdsvis ligge

rundt 1000 Hz og 150 Hz.



Figur 13: Lydbilde av lekkasje i plastrør [1, s.25]

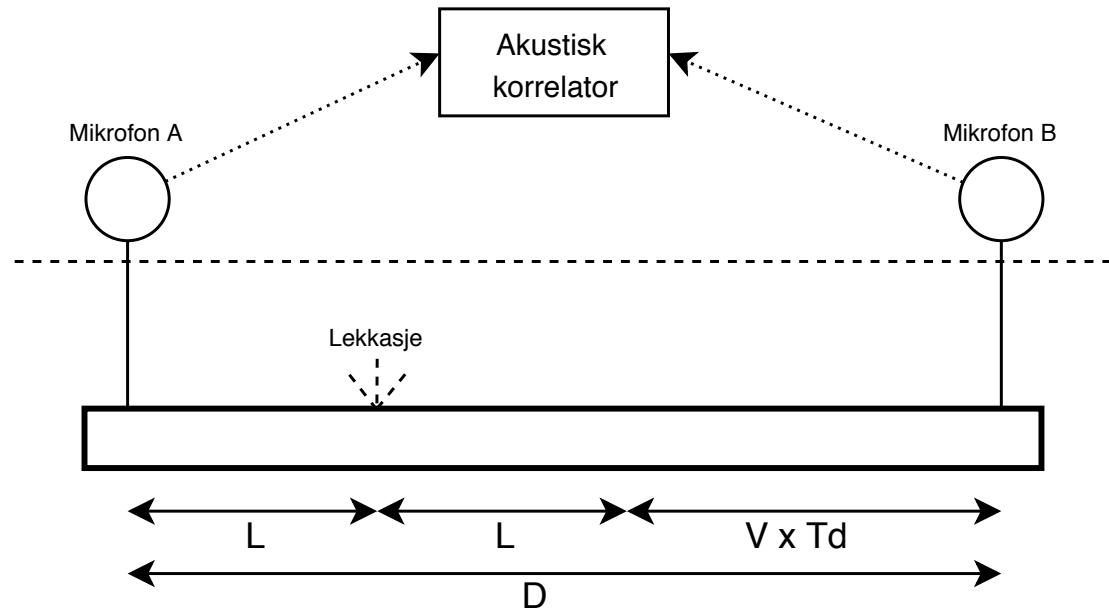
I tillegg til rørets materialer har også størrelsen på lekkasjen betydning for lydbildet som dannes. Ved en liten lekkasje i størrelsesorden 0,01-0,5 l/s dannes det mye høyfrekvent og lite lavfrekvent lyd. En større lekkasje fra 0,5 l/s og oppover lager mer lavfrekvent og mindre høyfrekvent jo større lekkasjen er. Siden vannmassene inne i røret leder lavfrekvent lyd bedre enn høyfrekvent, vil man kunne oppfatte store lekkasjer på lengre avstand en små, med akustiske deteksjonsmetoder [1]. En utfordring med den lavfrekvente lyden, er at den ofte er i samme frekvensområde som målt signalstøy. Dette kan gjøre det vanskelig å skille mellom lekkasje og støy. Trykket på vannledningen har også betydning for lyden ved en lekkasje, da svingningene til lyden vil få større amplitud desto høyere trykket er.

Ved tradisjonell leting etter lekkasjer brukes det vanligvis å skille mellom grovsøking og finsøking. Grovsøking vil si at man ved hjelp av ulike målinger finner ut hvilken del eller sone lekkasjen befinner seg. Dette kan typiske være at man finner ut at det befinner seg en lekkasje på et rørstrekke mellom to kommer [50]. Etter man har funnet ut hvilket rørstrekke lekkasjen befinner seg på, blir det finsøkt på det bestemte røret for å finne det eksakte lekkasjepunktet.

Ved grovsøking brukes det ofte tradisjonelle måleprinsipper som strømning og trykk. Vannforsyningssanlegg som har stasjonære målere plassert rundt omkring i sitt distribusjonsnett kan benytte disse i forbindelse med grovsøking. Alternativt finnes det mobile løsninger i form av flyttbare strømningsmålere i "clamp on" utforming [50].

### 5.2.2 Måling basert på akustikk

De akustiske metodene baserer seg på fenomenet nevnt innledningvis, nemlig at det oppstår lyd ved vannlekkasjer. Innenfor de akustiske metodene finnes det flere ulike prinsipper, blant de mest brukte er korrelasjonsmåling og marklytting. Korrelasjonsmåling går ut på at man monterer en mikrofon i hver ende av et rørstrekke. Dette gjøres ofte i kummene. Man legger også inn data for material-type og dimensjoner for røret. Hvis en av mikrofonene oppfatter lekkasjestøy registreres tidsforskjellen til den samme lyden treffer neste mikrofon. Tidsforsinkelsen sammen med rørdata gjør det mulig for korrelatoren å regne ut avstanden fra en av mikrofonene til lekkasjestedet [2]. Nedenfor vises en prinsippskisse for akustisk korrelasjonsmåling.



Figur 14: Lekkasjesøk med akustisk korrelator [2, s.22]

- L: Avstand fra nærmeste sensor til lekkasjen
- V: Lydens forplantningshastighet
- D: Rørlengden mellom sensorer
- Td: Tidsforsinkelsen

$$D = L + L + (V \cdot T_d) \quad (28)$$

Som gir

$$L = \frac{D}{2} - \frac{(V \cdot T_d)}{2} \quad (29)$$

Etter man har funnet ca. avstand fra en av kummene til lekkasjen, brukes det gjerne en marklytter for å dobbeltsjekke at lekkasjen er

reell. En marklytter er kort forklart en mikrofon med en signalforsterker. Forsterkeren forsterker lyden opp til ca. 20000 ganger. Operatøren vil da høre lyden av lekkasjen med sine egne ører [50].

### 5.2.3 Måling basert på strømningsmåling

Strømningsmåling går ut på å finne ut hvor mye væske som strømmer i gjennom et rør. I forbindelse med lekkasjedeteksjon kan strømningsmålere brukes til blant annet å utarbeide en volumbalanse. Hvis man måler strømningen inn og ut av et rør, kan ulikeheter mellom disse være et tegn på lekkasje. En annet moment med strømningsmåling er at man ved bruk av historiske strømningsdata, kan lage algoritmer for å finne tegn til lekkasje.

## 5.3 Deteksjonsmetoder

Siden Leksvik Vassverk ikke har noen form for automatisk lekkasjedeteksjon, er kunnskapen rundt dette temaet blitt innhentet fra andre kilder. Rapport fra Norsk Vann [3] er brukt som en viktig kilde for å komme fram til et resultat for lekkasjedeteksjon. "Erfaring med lekkasjekontroll" er en rapport med formål om å kartlegge status og erfaringer innen lekkasjesøking i Norge. Her diskuteres to ulike internasjonale metoder fra IWA (International Waterworks Association) for beregning av lekkasjenivå, topp-ned vannbalanse metode og bunn-opp minimum nattforbruk metode.

### 5.3.1 Topp-ned vannbalanse metode

Topp-ned vannbalanse metode er en metode som deler opp vannmengde levert i to deler, legalt forbruk og vanntap. Oppdelingen for denne metoden er vist i figur 15. Legalt forbruk er forbruk man enten har stipulert til f.eks. privathjem og målt forbruk til f.eks. næringslivet. Under vanntap ser man først på tilsynelatende tap som illegalt forbruk og vannmålerfeil, noe som kan se ut som en lekkasje fordi man ikke har kontroll på mengden. Deretter ser man på virkelig tap, der det deles opp i lekkasje og overløp i basseng og lekkasje i offentlige og private anlegg. Denne metoden inneholder mye usikkerhet og man har heller ikke oversikt på hvor eventuelt lekkasjen finner sted. Metoden brukes som et estimat på lekkasjemengde på forsyningssystemet og brukes ofte til å sammenligne opp mot andre anlegg. Sammenligningen inneholder mye usikkerhet dersom partene bruker forskjellige antagelser, som stipulert forbruk og tilsynelatende tap.

Total vannmengde (mengde vann målt ut fra vannverk)	Legalt forbruk	Legalt forbruk, fakturert	Fakturert, målt forbruk	Fakturert vannmengde
			Fakturert, ikke målt forbruk	
		Legalt forbruk, ikke fakturert	Ikke fakturert, målt forbruk	Ikke fakturert vannmengde
		Tilsynelatende tap	Ikke fakturert, ikke målt forbruk	
	Vanntap	Virkelig tap (lekkasje)	Illegalt forbruk	
			Vannmålerfeil	
			Lekkasje / overløp bassenger	
		Lekkasje offentlig ledninger	Lekkasje offentlig ledninger	
			Lekkasje private ledninger	

Figur 15: Topp ned vannbalanse metode [3, s.19]

### 5.3.2 Bunn-opp minimum nattforbruk metode

Bunn-opp minimum nattforbruk metode bruker en matematisk ligning og beregner lekkasjenivå etter minimum nattforbruk [3, s.20]. Formelen for lekkasjemengde  $q_{lm}$  er slik:

$$q_{lm} = (q_{mnf} - (q_{dt} + q_{bf} + q_{nf})) \cdot K_{trykk} \quad \left[ \frac{\text{liter}}{\text{time}} \right] \quad (30)$$

Målt nattforbruk,  $q_{mnf}$   $\left[ \frac{\text{liter}}{\text{time}} \right]$ : Målt vannmengde for store måleområder (minimum natt-time)

Dråpetap,  $q_{dt}$   $\left[ \frac{\text{liter}}{\text{time}} \right]$ : ((Antall påkoblinger  $\cdot$  0,8) + (km offentlige ledninger  $\cdot$  18) + (km private ledninger  $\cdot$  25))  $\cdot$  trykk (mVs) / 24 timer

Boligforbruk,  $q_{bf}$   $\left[ \frac{\text{liter}}{\text{time}} \right]$ : Antall bosatte  $\cdot$  nattforbruk (ca 0,6 l pr bosatte/time)

Næringsforbruk,  $q_{nf}$   $\left[ \frac{\text{liter}}{\text{time}} \right]$ : Beregnet fra målte data fra store bedrifter med nattaktivitet + spesifikt forbruk fra mindre bedrifter

Time til dag trykkfaktor,  $K_{trykk}$  [mVs]: Forhold mellom gjennomsnittstrykk over 24 timer og trykk ved minimum natt-time

Denne metoden inneholder også usikkerheter i forhold til parametre som blir satt og egner seg best til sammenligning mellom anlegg. Ifølge rapporten [3, s.20] er ikke parametrene egnet norske forhold da den ikke er entydig på hvilke påkoblinger som skal inkluderes. Man er også avhengig av en trykkfaktor som beskrives som "time til dag trykkfaktor". Denne trykkfaktoren kan være vanskelig å oppnå hvis man har et mindre anlegg med trykkregulering slik det er i Leksvik. Grunnen til dette er at ved et mindre anlegg vil ikke forbruket være såpass stort at det vil danne så stort trykkfall i ledningsnettet, samt at man har trykkregulering som prøver og opprettholder trykket kontinuerlig.

### 5.3.3 Volumbalanse

Volumbalanse er en balanse man setter opp der man ser på volumstrøm inn i et system og volumstrøm ut i et system [45, s.85]. Avviket mellom de to volumstrømmene anser man som endring i volumet for systemet.

$$\text{Volumendring i system} = \text{volumstrøm inn} - \text{volumstrøm ut} \quad (31)$$

$$\frac{d}{dt}V = q_{inn} - q_{ut} \quad \left[ \frac{m^3}{s} \right] \quad (32)$$

Volumbalanse i forhold til lekkasjedeksjon er en metode der man ser på volumstrømmen som går inn i et rør og volumstrømmen som går ut av samme røret. Hvis røret har en inngang og en utgang og det normalt ikke

er noe mulighet for å akkumulere volum i et rør, og man kan dermed anse endringen i volumet som et tap. Man kan derfor skrive om uttrykket.

$$\text{Volum tap} = \text{volumstrøm inn} - \text{volumstrøm ut} \quad (33)$$

Denne metoden lar seg enklest gjøre dersom man har en form for strømningsmåling i både inn- og utløp på røret. Om tapet er en lekkasje må vurderes opp mot hvor stort avviket er, antall koblinger på røret og eventuelle målerfeil. Normalt settes en avviksgrense for å unngå falske alarmer.

## 5.4 Resultater

I dette delkapittelet presenteres løsningene på lekkasjedekksjon som har blitt jobbet frem gjennom prosjektperioden. I neste delkapittel, ”Diskusjon” blir valgene som ble tatt under prosjektperioden drøftet. Resultatet er tre lekkasjedekksjonsmetoder som baserer seg på strømningsmåler som målemetode. To av metodene gjennomføres ved å sammenligne verdier fra dag til dag og den siste er en automatisert volumbalanse. For å utvikle deteksjonsmetodene har det ikke vært mulig å bruke strømningsverdier fra et reelt anlegg. Det er derfor satt opp et virtuelt distribusjonsnettverk som ligner distribusjonsnettet i Leksvik. For å få tak i verdier som kan behandles med lekkasjedekksjonsmetodene er det satt opp et simuleringsprogram som sender simulerte strømningsverdier. Metodene tar deretter inn de simulerte strømningsverdiene og bruker algoritmer i Python sammen med database i MySQL for å finne en eventuell lekkasje. For grafisk visning av verdier og prosessert data brukes Grafana.

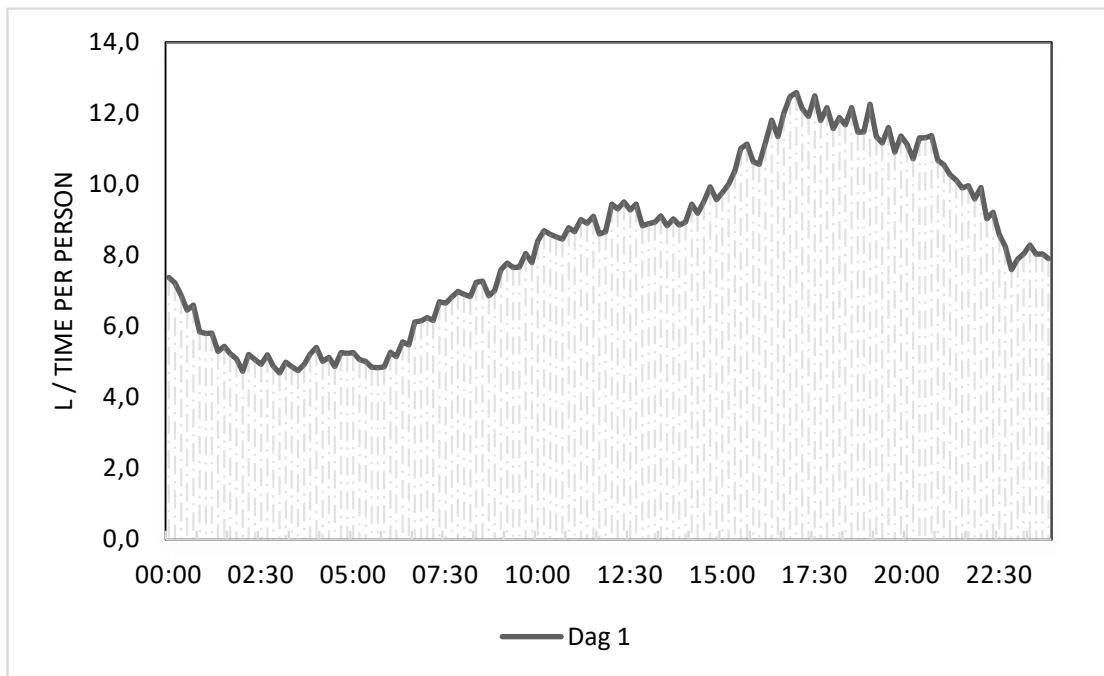
### 5.4.1 Valg av måler

Som tidligere nevnt hadde gruppen lite kunnskap om akustikk. Siden målere basert på akustikk virket mer omfattende med mye ukjent teori som ligger bak, ble det bestemt å gå videre med strømningsmåling. Det finnes en rekke ulike strømningsmålere å velge mellom, derfor ble det undersøkt litt mer spesifikt hvilken type måler som kan være aktuell hvis anlegget skulle vært montert i virkeligheten. Etter noen undersøkelser og telefoner rundt om til ulike leverandører, ser ”Ultrasonics TFX Ultra” ut som en god kandidat. Denne måleren leveres i ”clamp on” utforming som gjør at den kan monteres utenpå eksisterende rør. ”Clamp on” utformingen gjør også at den kan leveres tilpasset flere ulike rørdimensjoner, alt etter kundens ønsker. Den er i tillegg velegnet for å måle på væsker med lavt innhold av partikler. Se vedlegg C.1 for ytterligere detaljer.

### 5.4.2 Lekkasjedekksjon basert på historisk data (LHD)

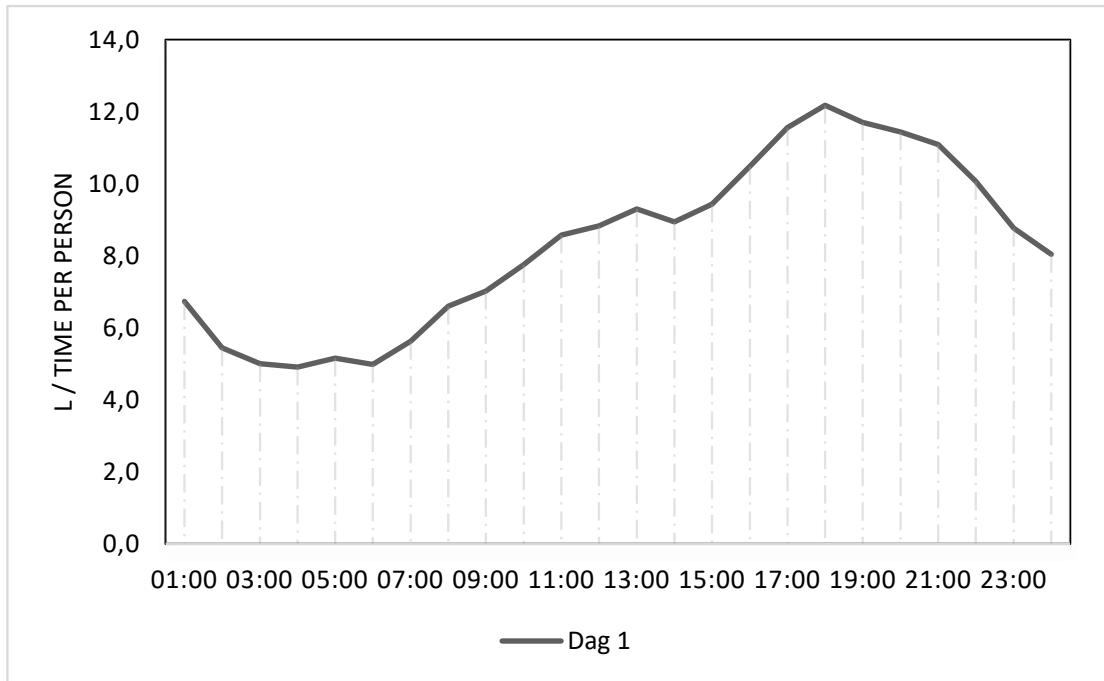
Lekkasjedekksjon basert på historisk data, heretter forkortet til LHD, er en metode som er utviklet gjennom prosjektperioden. Det er en metode som ser

på gjennomsnittsverdier i timen og sammenligner den timen med samme timen på dagene før. Metoden er tiltenkt til rør som går inn i en sone der man bare har strømningsmåling inn til sonen. En sone kan f.eks. være et boligfelt eller deler av en by. Siden en sone har stort sett et fast antall innbyggere vil ikke forbruket plutselig øke gjennom et døgn. Forbruket kan øke plutselig gjennom noen timer i et døgn, f.eks. ved en varm dag, men stort sett vil nattforbruket være det samme. For å få mest effektiv lekkasjedekksjon inneholder LHD to forskjellige funksjoner; en som finner øyeblikkelige lekkasjer og en som finner gradvise lekkasjer. LHD forklares lettere med grafer som blir presentert her.



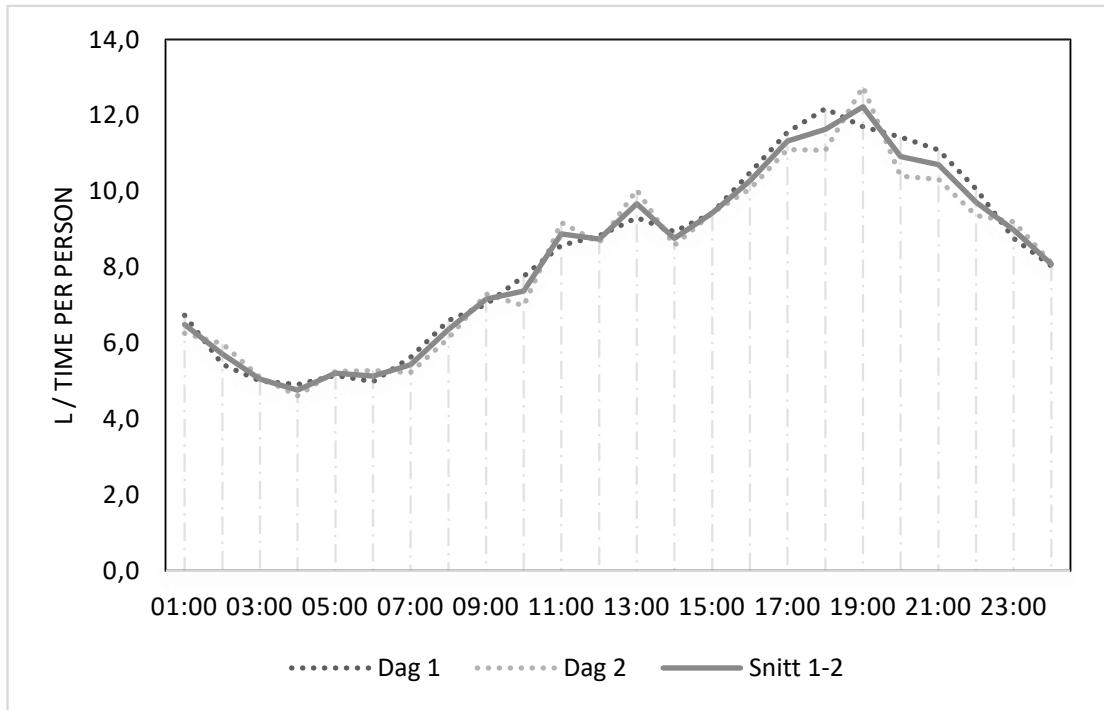
Figur 16: Dag en verdier fra strømningsmåler

Det første som skjer i denne metoden er at man tar inn verdier fra det siste døgnet fra databasen. Denne dataen er illustrert i grafen på figur 16. For å ikke behandle for mye data blir disse verdiene gjort om til time-gjennomsnittsverdier og deretter lagt inn i en ny tabell i databasen. Se figur 17 for time-gjennomsnittsverdier av figur 16



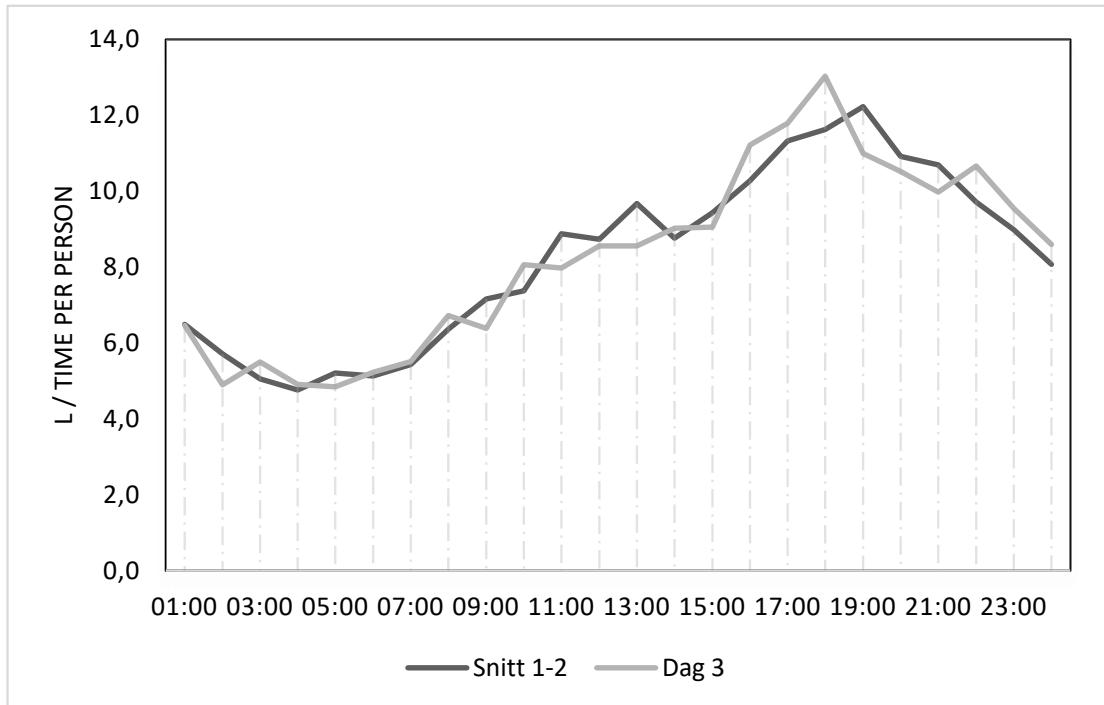
Figur 17: Dag en time-gjennomsnittsverdier

Etter at dag to er ferdig, legges den også inn i databasen med time-gjennomsnittsverdier. Dag to sammenlignes deretter mot dag en. Denne sammenligningen vil ikke gi et troverdig resultat fordi en dag ikke er nok grunnlag til å betrakte som et normalt forbruk. Men i praksis er ikke dette et problem da denne operasjonen vil bare skje ved første oppstart av metoden. For å ha noe å sammenligne dag tre med, finner Python programmet gjennomsnittsverdiene for dag en og dag to og legger det i en ny tabell i databasen. Se figur 18.



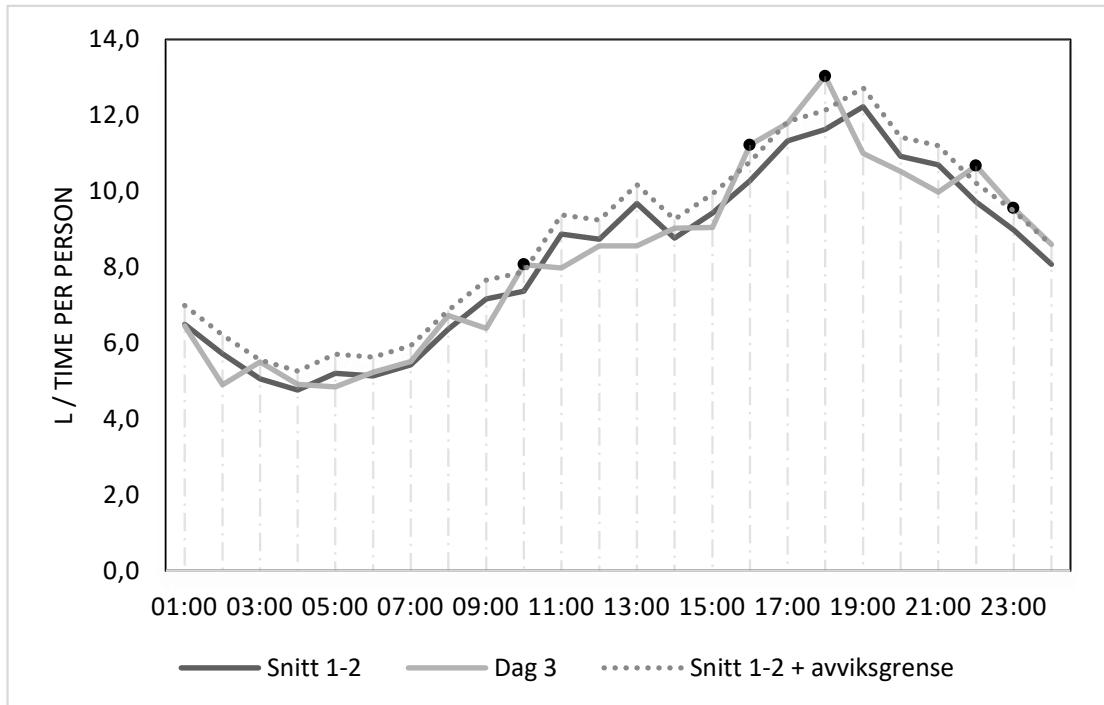
Figur 18: Dag en og dag to med gjennomsnittsverdier

Når dag tre er kommet inn i databasen sammenlignes den opp mot gjennomsnittsverdi for dag en og to, heretter forkortet til snitt 1-2. Se figur 19. Snitt 1-2 er den dataen som nå regnes som et normal forbruk og man vil oppdage et avvik dersom dag tre har en lekkasje i forhold til de tidligere dagene.



Figur 19: Dag tre og gjennomsnittsverdier for dag en og to

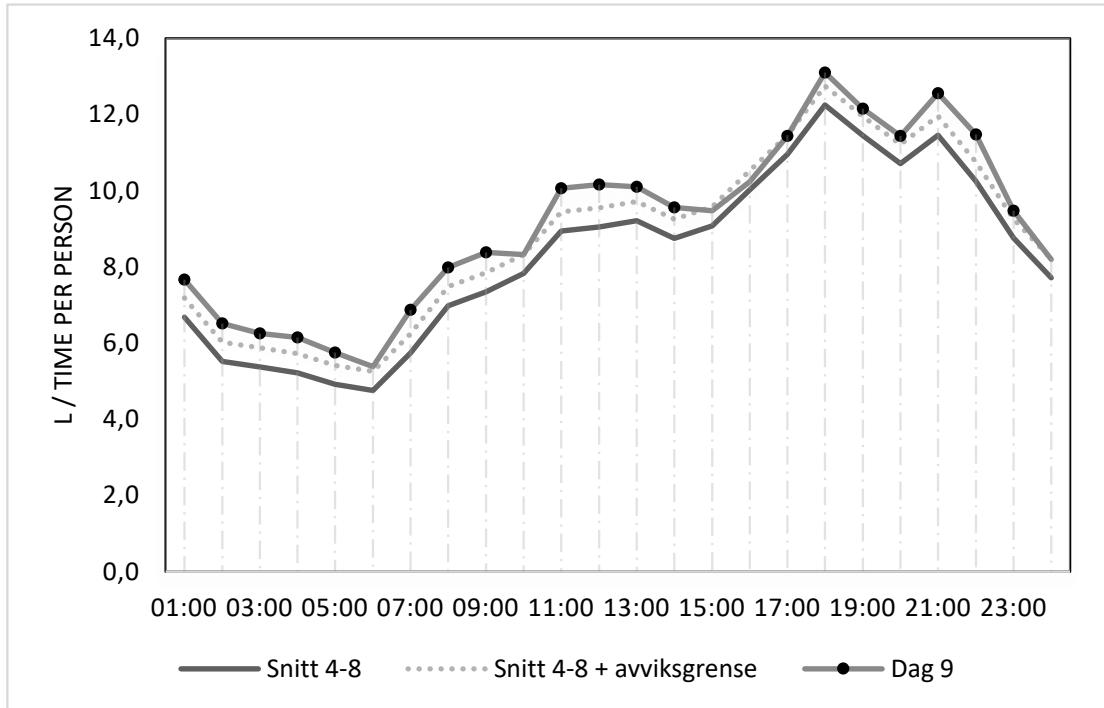
Man kan se at i figur 19 at det er avvik mellom grafene. Det er ønskelig at en alarm blir aktivert dersom det er avvik, men ettersom forbruket hele tiden vil variere settes det en avviksgrense. Avviksgrensen gjør at metoden tåler variasjoner i forbruket uten at den gir alarm. Grensen er noe operatøren selv må sette i forhold til hvor stor variasjon anlegget har på forbruket. I figur 20 er avviksgrensen lagt til.



Figur 20: Dag tre og gjennomsnittsverdier av dag 1-2 med avviksgrense

Det er nå mulig å se på hvilke timesverdier/punkter som er over avviksgrensen. I algoritmen velger man selv hvor mange timesverdier som må være over avviksgrensen før man anser det som en lekkasje. I figur 20 kan man se at fem punkter (svart prikker) er over avviksgrensen, det anses ikke som en lekkasje. Funksjonen ved å se på timesverdier over avviksgrense isteden for å sammenligne summen av begge grafene gjør at lekkasjedekjonsmetoden tåler et plutselig høyt forbruk uten å gi alarm om lekkasje.

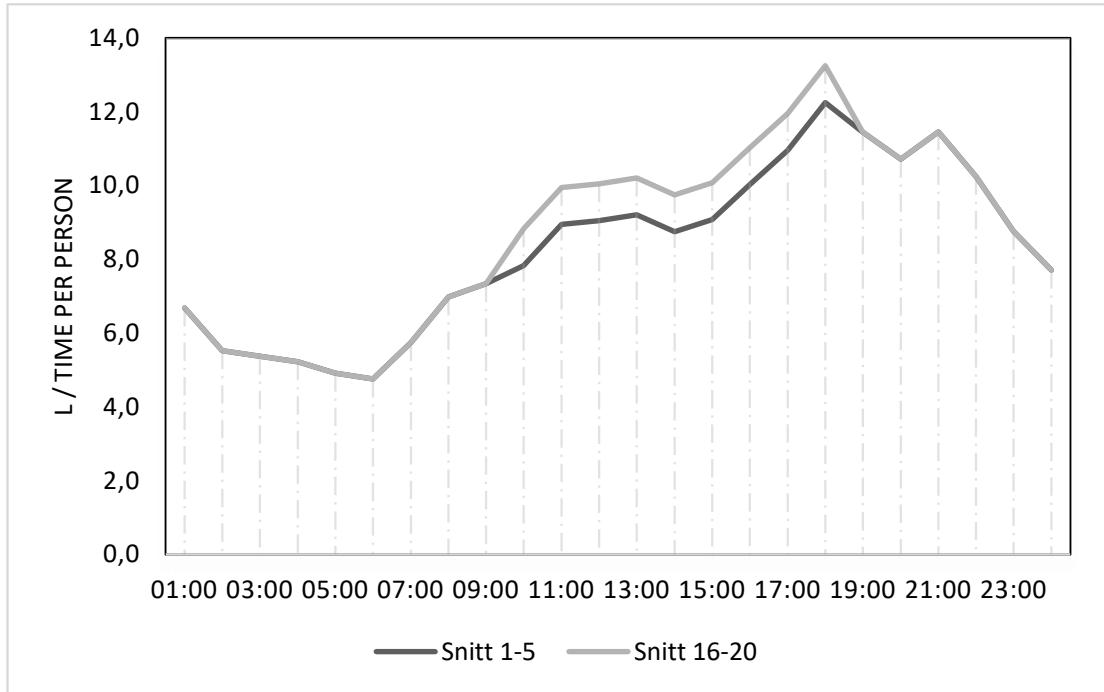
Etter at dag tre er ferdig legges den også inn i gjennomsnittsverdiene og neste dag sammenlignes dag fire mot snitt 1-3. Dette kjører kontinuerlig, men ser maks tilbake på fem dager. Ved å bare sammenligne dagsverdier med gjennomsnittsverdier fra maks fem dager bakover sikrer man at gjennomsnittsverdiene representerer et mest mulig normalt forbruk i den aktuelle tidsperioden. Det vil si at gjennomsnittsverdien raskt tilpasser seg en gradvis endring i forbruket slik som årstider og endring på antall innbyggere i sonen. På figur 21 vises et tilfelle der det er en lekkasje.



Figur 21: Dag ni og gjennomsnittsverdier for dag 4-8 med avviksgrense

I figuren ovenfor kan man se at dag ni sammenlignes med gjennomsnittsverdien fra de fem siste dagene, snitt 4-8. Her er det flere timesverdier som er over avviksgrensen, 19 av 24 timesverdier er over avviksgrensen. Dette anses som en lekkasje. Dette er et godt eksempel på at det må være et kontinuerlig avvik for at metoden skal gi alarm, slik som en lekkasje vil være.

Metoden som er beskrevet så langt i kapittelet fungerer på øyeblikkelige lekkasjer, der det plutselig vil bli et avvik i forbruket. Metoden vil ikke fungere på en gradvis lekkasje da en slik type lekkasje utvikler seg for langsomt. Det vil gjøre at det aldri vil komme en dag der det gjennomsnittlige forbruket vil gå over avviksgrensen på flere punkter. Dersom den gradvise lekkasjonen ikke blir oppdaget vil den bli lagt til i gjennomsnittsverdiene og bli ansett som et normal forbruk. For å oppdage gradvise lekkasjer har denne metoden enda en funksjon. Funksjonen er at man ser på og sammenligner forskjellige gjennomsnittsverdier. Det vil si at man f.eks. sammenligner gjennomsnittsverdier for dag en til fem med gjennomsnittsverdier for dag 16 til 20, altså snitt 1-5 mot snitt 16-20.

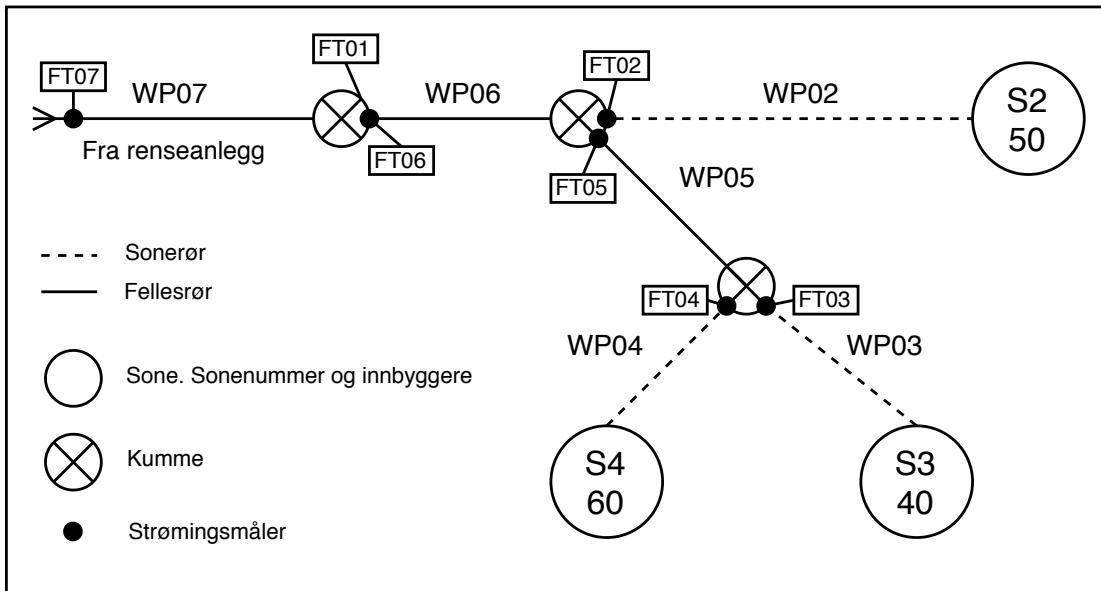


Figur 22: Sammenligning av gjennomsnittsverdier uten gradvis lekkasje

Figur 22 sammenligner snitt 1-5 mot snitt 16-20. Ved samme metodene som har blitt beskrevet tidligere, med avviksgrense på timesverdier, sjekker den om det har vært en gradvis og/eller en øyeblikkelig lekkasje. I tilfellet ved figur 22 vil ikke metoden gi varsel på lekkasje selv om man kan se at det er avvik på dagtid for den gitte tidsperioden. Dette avviket kan f.eks. være på grunn av en varmere periode der det er større forbruk av vann. Slik metoden er bygd opp, ved å se på totalt antall avvikspunkter gjennom en hel dag, vil ikke slike avvik som vist i figur 22 skape falske alarmer. Dersom det blir endring i årstid vil det bli minimale endringer i nattforbruket, noe som vil tilsi at denne funksjonen tilpasser seg slike endringer. Metoden med lekkasjedeteksjon på gradvis lekkasje er ikke implementert som en metode i dette prosjektet, men ansees som et resultat allikevel. Figur en i vedlegg C.2 viser lekkasjedeteksjon basert på historisk data implementert i Grafana. Denne grafen viser antall avvik de siste 24 timene. Dersom det er over 18 avvik på de siste 24 timene vil det gi en alarm.

#### 5.4.3 Lekkasjedeteksjon med volumbalanse

Slikt det er beskrevet i delkapittel 5.3.3, er volumbalanse en metode for å detektere en lekkasje. Den er også implementert i denne lekkasjedeteksjonsmodulen. Som vist i figur 11 er det satt opp et virtuelt distribusjonsnettverk. For å bruke volumbalanse i lekkasjedeteksjon er man avhengig av å ha kontroll på hva som går ut og inn av røret.



Figur 23: Distribusjonsnett utklipt

Volumbalansefunksjonen fungerer på rør man har kontroll på strømningen inn og ut av røret, slik som fellesrørene WP07, WP06 og WP05, vist i figur 23. F.eks. på WP05 kan det leses av strømningsnverdi inn i røret ved å se på FT05. Strømningen ut av røret er summen av FT03 og FT04. Formelen er da:

$$\frac{d}{dt}V = q_{inn} - q_{ut} \quad q_{inn} = FT05 \quad q_{ut} = FT03 + FT04 \quad \left[ \frac{m^3}{s} \right] \quad (34)$$

Tapet i røret er  $\frac{d}{dt}V$ . Tapet inneholder lekkasjemengde og feilmåling. Ved å sette en grense og en alarm dersom  $\frac{d}{dt}V$  er større en grensen vil dette fungere som lekkasjedeksjon. Figur to i vedlegg C.2 viser volumbalanse implementert i Grafana. Grafen viser avviket mellom inn- og utløp for røret WP06. Det er satt en avviksgrense på  $10 \frac{l}{s}$ . Man kan se at det er en lekkasje på  $100 \frac{l}{s}$ , som resulterer i en alarm.

#### 5.4.4 Manuell avlesning i Grafana

Leksvik Vassverk har idag grafer som et hjelpemiddel for å oppdage lekkasjer. Ved å sammenligne grafene kan man finne lekkasjer. Dette var noe oppdragstaker så muligheter til å bygge videre på. Ved hjelp av Python, en database og Grafana kan man sette opp ulike grafer, tabeller og andre diagrammer som gjør det enklere å oppdage lekkasjer. Her kan kunder ønske seg diagram over hva de vil. Noen eksempler kan være gjennomsnittsverdier over dager, uker eller år. Disse grafene kan igjen legges over hverandre for å enklere sammenlignes. Her er det flere muligheter og lite begrensinger. Etter gitte spesifikasjoner, kan det utarbeides en algoritme som passer, regne ut verdier i Python og sende de til databasen. Grafana leser dataen fra databasen og presenterer den som

ønskelig i ett såkalt “dashboard”. De ønskelige verdiene blir kontinuerlig utregnet, slik at man vil hele tiden få oppdatert informasjon. I tillegg til å lese av grafer kan man også i Grafana legge inn alarmbetingelser som gir en alarm dersom betingelsene blir oppfylt. Denne metoden for å finne lekkasjer gir operatørene på vannverkene mulighet til å tilpasse brukergrensesnittet selv, da Grafana krever lite erfaring og opplæring for å brukes. Figur tre i vedlegg C.2 viser et eksempel på hvordan diverse paneler kan visualisere nyttige data i Grafana, som igjen kan brukes til å detektere lekkasje manuelt.

## 5.5 Implementasjon

På samme måte som simuleringsprogrammet i skyen, fra kapittel 4.6, er implementert, er også algoritmen for å detektere lekkasje implementert. Programmet er implementert i Python, kjøres i Docker og bruker samme oppgavebehandler som simuleringsprogrammet. Koden finnes på Github [33] under “LeakDetection”.

Programmet består av følgende moduler:

- Datalagring - Lager data for hver strømningsmåler
- Databaseklient - Kommuniserer med databasen
- Diverse kalkulasjoner - Generelle matematiske uttrykk, tilpasset datastrukturen som er brukt

Programmet kjører en oppgave på et periodisk intervall på 2,5 sekunder, som i systemets tidsrom betyr hver time. Før oppgaven startes kjøres det en initialiseringskode. Denne koden oppretter de tabellene programmet trenger i databasen, for å kunne lagre den dataen som blir produsert. Deretter opprettes det objekter for å lagre midlertidig data for kalkuleringer.

Den periodiske oppgaven behandler data for alle strømningsmålerene. Dette gjelder time-gjennomsnittsverdier, i henhold til metoden LHD, forklart i kapittel 5.4.2. I tillegg kalkuleres det volumbalanse i forklart i kapittel 5.3.3. Hver periode hentes det inn siste time måleverdier, for alle syv strømningsmålerene, fra databasen. Det kalkuleres et time-gjennomsnitt av disse verdiene og det lagres i databasen. De siste fem verdiene for den gitte timen hentes ut. Disse kan f.eks. tilsvare verdien for kl. 10 igår, dagen før osv. opptil fem dager. Det kalkuleres et gjennomsnitt av disse fem verdiene og det lagres i databasen. Time-gjennomsnittet som gjelder for nåværende time sammenlignes mot gjennomsnittet for de siste fem verdiene. Fra denne sammenligningen lagres det om verdien er fortrinnsvis 10%, 20% og 30% over gjennomsnittet fra de fem verdiene. Etter dette kalkuleres avviket for volumbalansen, på gjennomsnittet, mellom strømningsmålerene som står på inn- og utløp på samme rør.

Antall verdier over de gitte prosentgrensene gjelder for de siste 24 timene. Dette betyr at det finnes 24 verdier, hvor hver verdi er over, eller

under grensen, en eller null. Det som lagres, er hvor mange verdier som er over grensen, av disse 24 verdiene.

Det enkleste er å lagre om hver time er over eller under grensen, hente inn disse 24 verdiene og kalkulere summen av de. Det må da gjøres hver time, for alle syv strømningsmålerene. Da en time er 2.5 sekund krever dette mye ressurser og det er derfor implementert en egen datastruktur i programmet. Strukturen holder kontroll på hvor mange verdier som har vært over eller under grensen de siste 24 timene. Datastrukturen er representert i kode 9.

Kode 9: Datastruktur strømmningsmålere

```
1 class FtData:
2     def __init__(self, _tagId):
3         self._tagId = _tagId
4         self.pointsOver10 = 0
5         self.pointsOver20 = 0
6         self.pointsOver30 = 0
7         self.queuePointsOver10 = queue.Queue()
8         self.queuePointsOver20 = queue.Queue()
9         self.queuePointsOver30 = queue.Queue()
10    for i in range(24):
11        self.queuePointsOver10.put(0)
12        self.queuePointsOver20.put(0)
13        self.queuePointsOver30.put(0)
```

Datastrukturen inneholder tre tall, "pointsOver\*", dette er antall punkt over den gitte grensen. I tillegg inneholder den tre køer, og logikk for å initialisere disse køene til 24 tall med null i, som representerer at ingen verdier de siste 24 timene har vært over grensen. En kø er en datastruktur som fungere akkurat som en hverdaglig kø, først inn, først ut. For å oppdater denne datastrukturen hver time er metoden i kode 10 implementert.

Metoden for å kunne detektere gradvis lekkasje er ikke implementert. Dette er begrunnes med at det leveres en demonstrasjonsrigg, hvor tilskuerene har begrenset tid.

Kode 10: Metode for oppdatering av datastruktur strømningsmålere

```
1 def updatePoints(diff, limit, points, queue):
2     if diff > limit:
3         points = points + 1
4         queue.put(1)
5     else:
6         queue.put(0)
7         removedItem = queue.get()
8         if removedItem == 1:
9             points = points - 1
10    return points
```

Metoden tar inn differansen mellom verdien fra time-gjenomsnittet og verdien for gjennomsnittet av siste fem dagene, hvor mange verdier som er over grensen til nå, og køen som lagrer de tidligere verdiene. På linje en verifiseres det om differansen er over grensen. Er den det, legges det til ên på “points”, og det settes inn en êner i køen. Er ikke differansen over grensen gjøres det ikke noe med “points”, men det settes inn en nuller i køen. Deretter tas elementet, som representerer den 25 timen, ut av køen. Det er nå for gammelt, da man ser på siste 24 timene. Om det elementet som tas ut inneholder en ener, reduseres “points” med en, siden et punkt over grensen fjernes. Er ikke elementet som tas ut av køen en ener, gjøres det ingenting. Deretter returneres antallet verdier over grensen, “points”. Denne verdien lagres i databasen og visualiseres i Grafana, hvor det også er konfigurert alarmer på verdien.

## 5.6 Diskusjon

Ett usikkerhetsmoment som dukket opp i forbindelse med strømningsmåling var dersom målingene ble gjort på en ringledning. Da var det en viss usikkerhet om hvilken retning i ringledningen vannet ville strømme, og om det eventuelt vil snu retning noen gang. For å avklare dette ble Borgar Berg på Leksvik Vassverk kontaktet for en faglig samtale om tematikken. Han sa i generelle ordelag at han syntes strømningsmåling som måleprinsipp virket som et fornuftig måleprinsipp å bygge den videre oppgaven på. Videre kunne han fortelle at vannstrømmen i en ringledning vanligvis kun vil strømme i en retning. Når en ny ringledning blir lagt ned i jorden og vannet blir skrudd på, vil vannet “finne den retningen som gir minst motstand” og vil fortsette å strømme slik inntil noe unormalt inntreffer. Med unormalt viste han til to konkrete eksempler. Hvis et rørstrekk byttes ut, kan “motstanden” i ringledningen forandres, slik at motsatt strømningsretning gir minst motstand for vannet. Det andre eksemplet var at det plutselig ble en stor forandring i vannforbruket. En stor forandring kunne være at en brannhydrant ble åpnet for fullt, eller at et rør var sprukket. Han sa videre at en plutselig endring i strømningsretningen ble betraktet som en unormal hendelse, som eventuelt kan brukes som et tegn på lekkasje i en ringledning.

Selv om målet med dette prosjektet er å lage et demonstrasjonsanlegg, bør de valgene gjennom prosjektet i en viss grad være forankret i virkeligheten. Derfor ble det undersøkt litt hvilke plasseringer, miljøer og påvirkninger som påvirker et slikt system hvis det skal brukes i den virkelige verden. Magnus Raudberget jobber som formann hos entreprenørfirmaet Tore Løkke AS. De holder for tiden på med en større oppgradering av vannforsyningsnettet i Ørland kommune. Han kan fortelle at de legger vannledningene på mellom 2-3 meters dybde. Kummene blir dimensjonert og montert i henhold til “VA-miljøblad nr 112-Basal”. Han sier videre at diameter og høyde på kummen varier etter hvilken dimensjon det er på rørene som blir benyttet. Avslutningsvis sier han at det legges

dreneringsrør sammen med vannledningen. Dette dreneringsrøret går opp til bunnen av kummen. Ved en eventuell lekkasje inne i kummen vil vannet føres ut gjennom dreneringsrøret slik at teknisk utstyr ikke blir oversvømt av vann. I forhold til temperaturer i kummene sier han vannet som strømmer gjennom rørene varmer opp omgivelsene. Han kan ikke garantere at det er plussgrader kummen til enhver tid, men det vil aldri gå nå mye under 0°C. Med andre ord kan det virke som om utstyr som blir montert i vannkummer vanligvis vil stå tørt, med temperaturer som kan komme under 0°C på vinteren. Utfordringen med temperaturen kan løses ved å velge en PLS og I/O kort med utvidet temperaturområde.

Ett moment å merke seg ved valg av måler til renset vann, er at mange målere benytter partiklene i vannet for å måle hastigheten. Hvis vannet er godt filtrert for humus og partikler kan man oppleve problemer med stabiliteten til måleren, da vannet er for rent for deres målemetode. I tillegg til å se på strømningsmåler ble det undersøkt litt rundt ulike metoder for å forsyne et styreskap bestående i en kumme med fornybar energi. Det finnes svært mange ulike leverandører med et vidt spekter av produkter innenfor småskala vind og solkraft. Etter hvert ble det presisert fra oppdragsgiver at det ikke var aktuelt å kjøpe inn noe slikt utstyr, så lenge demonstrasjonssystemet ikke fysisk skulle monteres.

Gjennom prosjektpersonalen har det blitt sett på flere metoder å finne lekkasje. Først ble det sett på metodene topp-ned vannbalanse metode og bunn-opp minimum nattforbruk metode. Disse metodene ble ikke implementert på grunn av stor usikkerhet og man har bare kontroll på hva tapet er i distribusjonsnettverket og ikke i spesifikke rør. Bunn-opp minimum nattforbruk metode krevde også en trykkfaktor som ikke er aktuell i dette anlegget da det har trykkregulering og er ett mindre anlegg med lite trykkfall. I Norsk Vann rapporten [3] ble det skrevet at metoden ikke var tilpasset norske forhold da metoden ikke var entydig på hvilke påkoblinger som skal inkluderes.

Oppdragstaker ønsket å finne en metode der man ikke trenger å sette parameter for et forbruk, men heller lære av informasjonen den har tilgjengelig hvordan et normalt forbruk er. Etter å ha lest en del rapporter ble det ikke funnet en slik metode. Det ble da heller satt igang å utvikle en slik metode selv. Resultatet ble LHD metoden. Denne motoden kan implementeres på hvilket som helst rør inn til en sone uten å vite noe om innbyggeretall, offentlige og private vannledninger, trykk osv. Metoden tar til høyde for både gradvise og øyeblikkelige lekkasjer og man har mulighet til å tilpasse selv hvilke dager og dagsgjennomsnitt man vil sammenligne.

I tillegg til LHD metoden har lekkasjedekksjon med volumbalanse blitt implementert. Dette er en metode som allerede er i bruk Leksvik Vassverk i form av manuell avlesning, det vil si at man ser på hva som går inn i røret og sammenligner med det som går ut. Oppdragstakere ønsket å automatisere denne metoden slik at den kontinuerlig overvåker rørene der det er mulighet for det. Med tanke på at metoden har blitt brukt før i form av manuell avlesning vil det ikke være mye arbeid med å få implementert metoden da

strømningsmålerne og verdiene fra de allerede finnes. Lekkasjedeteksjon med volumbalanse er en presis lekkasjedeteksjonsmetode som med stor sikkerhet, dersom målingene er riktig, kan antyde lekkasje eller ikke. Metoden vil være sikrere jo flere strømningsmålere man har i anlegget.

## 5.7 Konklusjon

Det konkluderes med at bruk av strømningsmåler gir det beste utgangspunktet for arbeidet med lekkasjedeteksjon. Akustikk er for omfattende, med mye avansert og ukjent teori. En erfaring å ta med seg videre er at det er brukt mye ressurser på å undersøke diverse utstyr innenfor fornybar energi og strømmåling, som i liten grad kom til nytte under prosjektet. Det er brukt mer tid en planlagt på lekkasjedeteksjonsmetodene da metodene som allerede eksisterer ikke er egnert for norske forhold, har stor usikkerhet eller bare ser på tapet i hele distribusjonsnettet. Dette førte til at det måtte utvikles en ny metode. Metoden som er utviklet har blitt kalt "Lekkasjedeteksjon basert på historisk data" eller forkortelsen LHD. Denne metodene kan med større sikkerhet enn tidligere metoder antyde en lekkasje. Den er også utviklet slik at den er dynamisk i forhold til at det ikke er nødvendig og sette noen parameter og at den selv lærer seg hva et normalt forbruk er. Dette gjør metoden tilpasningsdyktig. Lekkasjedeteksjon med volumbalanse er også implementert. Dette er en metoden som kan med stor sikkerhet antyde lekkasjer, men krever at man har kontroll på volumet som går inn og ut av et rør. Ved flere strømningsmålere vil metoden bli sikkert. Et ekstra hjelpemiddel i forhold til lekkasjedeteksjon er grafene i Grafana, her kan det tilpasses slik som operatører selv ønsker. Ved bruk av egenutviklet og eksisterende lekkasjedeteksjonsmetoder er det mulig å detektere lekkasje i distribusjonsnettet, med høy sikkerhet.

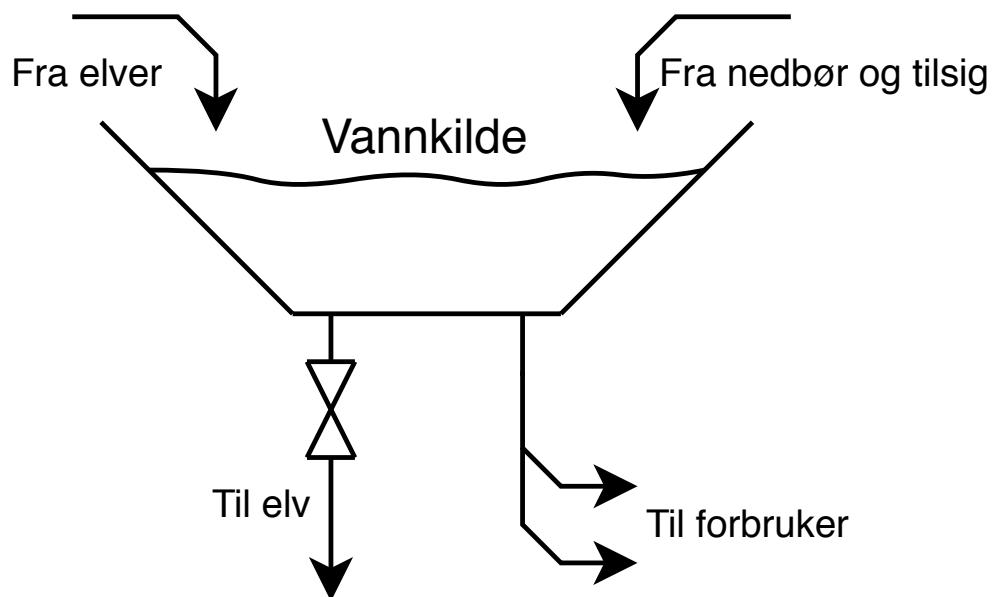
Som videre arbeid innefor lekkasjedeteksjon ville det være interessant å utvikle eller finne nye metoder. For å kvalitetsikre lekkasjedeteksjonen kunne det vært en ide å samkjøre flere deteksjonsmetoder istedenfor å kjøre en og en. Dette hadde ført til en bedre metode totalt sett, men krever en del ekstra arbeid.

## 6 Flomsikring

### 6.1 Introduksjon

I dette kapittelet er det innhentet informasjon, sett på metoder og til slutt funnet en god løsning på flomsikring av vannkilder. Flom i vannkilder kan oppstå dersom vannmengden som kommer inn i vannet er større en det som går ut. Det er derfor ønskelig å ha kontroll hva som kommer inn og hva som går ut av vannet. For å se på dette problemet er det blitt satt opp en fiktiv vannkilde.

Den fiktive vannkilden som skal flomsikres har flere innløp som fører vannmengde inn til vannet, men bare et utløp i tillegg til vannforbruket av innbyggere. Utløpet av vannet reguleres med en reguleringventil slik at man kan holde vannivået regulert rundt et gitt nivå. Regulering av vannet er viktig for å ha nok opplagret vann til forbrukere og samtidig hindre flom. I tillegg til elvene som har utløp til vannet kommer det også vannmengde fra nedbør direkte på vannet og nedbør rundt vannet som kommer inn som tilsig. Tilsiget varierer fra vannkilde til vannkilde, ettersom vannkildens geografiske plassering avgjør hvor stort nedbørsfelt vannet har som igjen avgjør hvor stort tilsiget er.



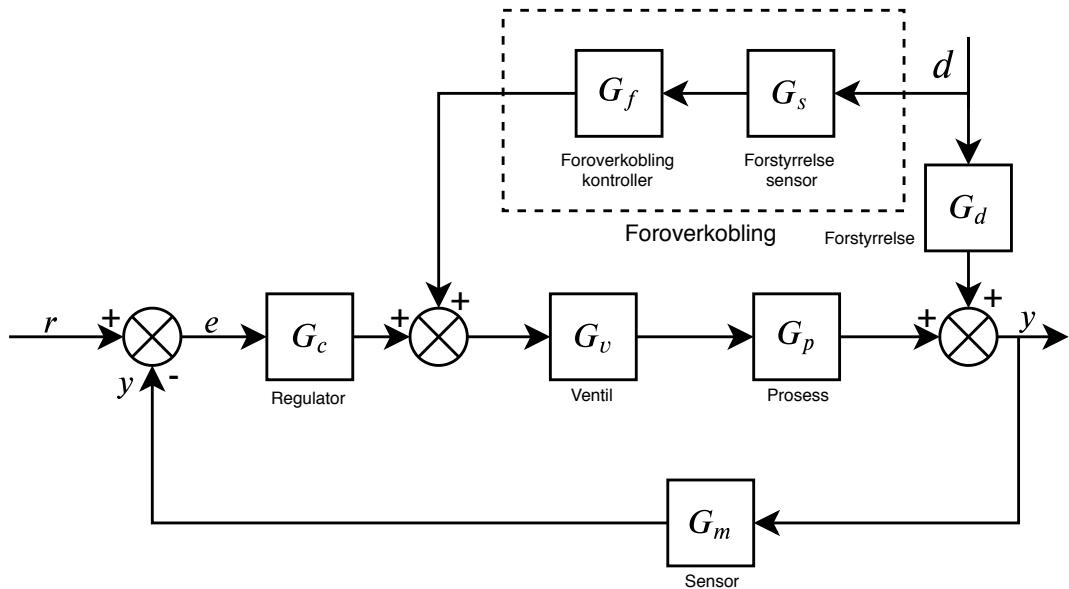
Figur 24: Oversiktsbilde flomsikring

### 6.2 Metoder og kilder

#### 6.2.1 Foroverkobling

Hensikten med å bruke foroverkobling i en prosess er å måle viktige forstyrrelser og ta hensyn til disse forstyrrelsene før det påvirker prosessen

[4, s.263]. For en vanlig prosess med bare tilbakekobling vil ikke regulatoren ta hensyn til forstyrrelsen før etter forstyrrelsen har påvirket prosessen. For å bruke foroverkobling bør forstyrrelsen kontinuerlig måles av en sensor. Man bør også ha en overføringsfunksjon som er tilnærmet lik den reelle prosessen. Foroverkoblingens effektivitet samsvarer med hvor eksakt overføringsfunksjonen er.



Figur 25: Blokkskjema av tilbakekobling og foroverkobling [4, s.271]

I figur 25 er det tegnet opp et blokkskjema av en prosess med tilbakekobling og foroverkobling. Ideelt kan man få en perfekt regulering dersom man har alle overføringsfunksjonene helt korrekt. For å bruke foroverkobling må man finne  $G_f$ . Forstyrrelsens påvirkning på utgangen er vist i ligning 35.

$$\frac{y(s)}{d(s)} = \frac{G_d + G_s G_f G_v G_p}{1 + G_c G_v G_d G_p G_m} \quad (35)$$

Det er ønskelig at forstyrrelsen ikke påvirker utgangen, man kan derfor sette ligning lik null og deretter løse for  $G_f$ .

$$\frac{y(s)}{d(s)} = \frac{G_d + G_s G_f G_v G_p}{1 + G_c G_v G_d G_p G_m} = 0 \quad (36)$$

$$G_f = -\frac{G_d}{G_s G_v G_p} \quad (37)$$

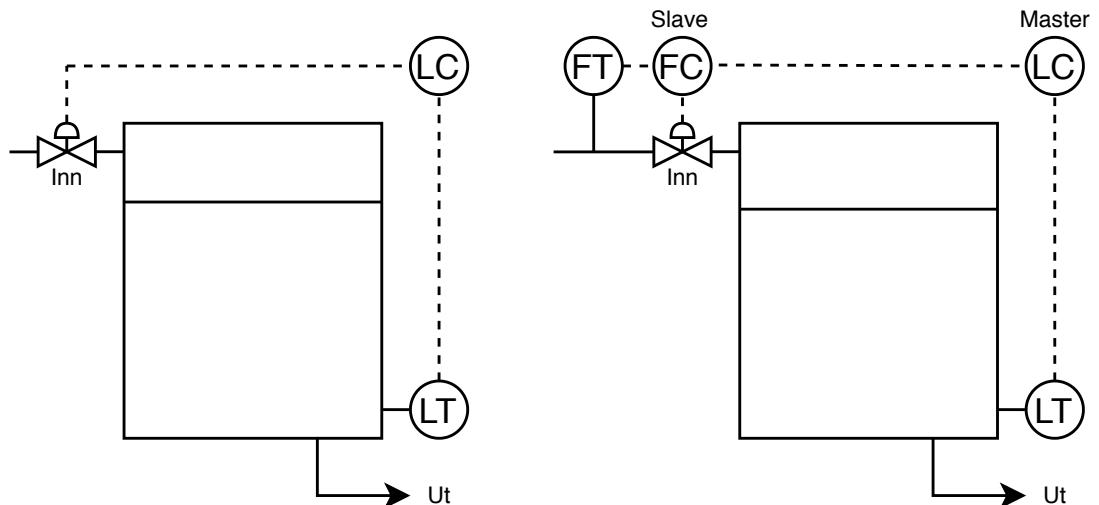
Deretter kan  $G_f$  settes inn i prosessen.

### 6.2.2 Kaskaderegulering

Kaskaderegulering er en type regulering som kan brukes til å ta hensyn til forstyrrelser [4, s.279]. Reguleringen bruker to reguleringsløyper som

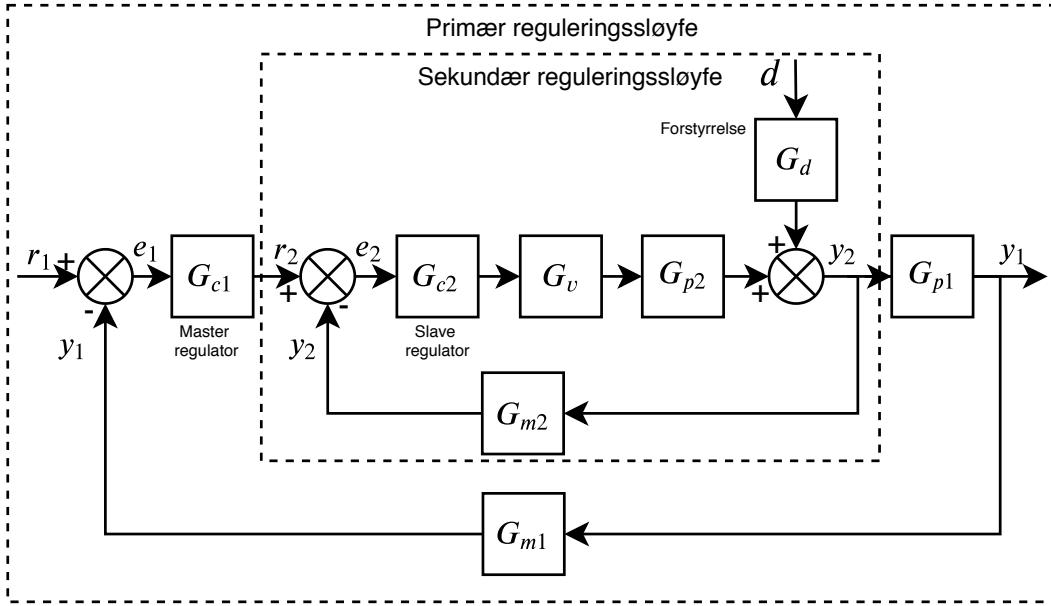
sammen styrer prosessen. Den første reguleringsløyfen, primærsløyfen har master regulator og målingen på prosessverdien som skal reguleres. Den sekundære reguleringsløyfen har slave regulator og målingen på forstyrrelser som påvirker det pådragsorganet styrer. Det vil si at målingen på den sekundære reguleringsløyfen vil oppdage forstyrrelsene før den påvirke selve prosessen og slaveregulatoren kan ta hensyn til dette.

Et eksempel kan være regulering av nivået på en tank. Der har man har en reguleringeventil som styrer strømningen inn til tanken. Pådraget til ventilen kommer fra en kontroller som får data fra en nivåtransmitter plassert på tanken. Trykkendringen i røret inn til tanken kan påvirke strømningen inn i tanken. Ved å plassere en sekundær reguleringsløyfe med en slave regulator og en sensor som måler eventuelle strømningsendringer i røret kan man ta hensyn til disse forstyrrelsene før det endrer nivået i tanken. På figur 26 ser man til høyre at kaskaderegulering er implementert for problemstillingen som ble nevnt i dette eksempelet.



Figur 26: Implementering av kaskaderegulering

Utgangen av master regulator fungerer som settpunkt for slave regulator. Reguleringsløyfene er flettet inn i hverandre med den sekundære reguleringsløyfen inne i den primære reguleringsløyfen. Dette er illustrert i blokkskjemaet for kaskaderegulering i figur 27.

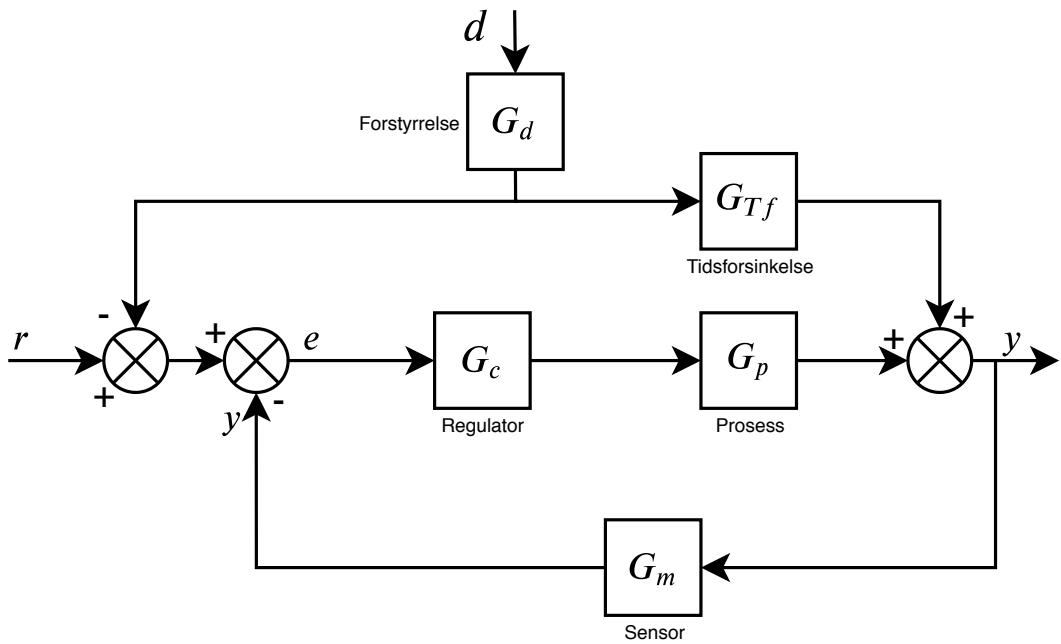


Figur 27: Blokkskjema for kaskaderegulering [4, s.282]

### 6.2.3 Settpunktendring for fremtidig forstyrrelse

Endring av settpunkt for fremtidig forstyrrelse er en metode for å unngå at forstyrrelsen påvirker prosessen for mye. Målet med denne metoden er å unngå at forstyrrelsen gir oversving i prosessen. Prosessen må tåle undersving. Metoden innebærer at man vet om forstyrrelsen som kommer i fremtiden. Man trenger ikke nødvendigvis å ha en prosessmåling på forstyrrelsen, selv om det blir mest korrekt, men kan matematisk regne ut hvordan den påvirker prosessen. Når man vet om forstyrrelsen og hvordan den vil påvirke prosessen kan man endre settpunktet slik at prosessen tilpasser seg og er klar for forstyrrelsen som kommer.

Et eksempel kan være at man vet at det kommer sterkt sol og man har en prosess som er følsom for oversving i temperatur, men undersving er ingen problem. Dersom man vet hvor mye temperaturøkning man får den dagen solen kommer kan man kompensere for dette ved å trekke den temperaturen fra settpunktet tidligere. I figur 28 er denne metoden illustrert med blokkskjema.



Figur 28: Blokkdiagram av settpunktendring

I figur 28 er forstyrrelsen,  $d$ , trukket fra settpunktet så fort man vet om forstyrrelsen, men påvirker ikke prosessen før etter tidsforsinkelsen,  $G_{Tf}$ . Tidsforsinkelsen er altså den tiden mellom at man får informasjon om forstyrrelsen og har trukket den fra settpunktet til forstyrrelsen påvirker prosessen.

## 6.3 Resultater

### 6.3.1 Håndtering av forstyrrelse

Metoden som er valgt for å håndtere forstyrrelsen i flomsikring er ”Settpunktendring for fremtidig forstyrrelse” som beskrevet i kapittel 6.2.3. I dette delkapittelet er det forklart hvordan denne metoden er implementert i flomsikring, og i neste delkapittel, ”Diskusjon”, er det begrunnet hvorfor.

I delkapittel 4.4.1 ble det funnet to forstyrrelser som påvirker prosessen,  $q_{regn}$  og  $q_{tilsig}$ . Det ble også funnet matematiske metoder for å regne ut disse forstyrrelse viss man viste om værmeldingen fram i tid.

Når man har regnet ut hvor mye  $q_{regn}$  og  $q_{tilsig}$  vil påvirke nivået i vannkilden ved hjelp av værmeldingen kan man bruke den verdien til å endre settpunktet. Utregningen vil gi et estimat på hvor mange meter nivået øker når forstyrrelsen kommer. Siden det er en treg prosess blir settpunktet endret to dager før nedbøren kommer slik at nivået i vannkilden får tid til å stabilisere seg på det nye nivået. Dersom det ikke kommer noe nedbør de neste to dagene, eller at prosessverdien allerede er under settpunkt grunnet høyt vannforbruk, vil ikke settpunktet endre seg.

Metoden følger figur 28, der  $d$  er værmelding og  $G_d$  er de matematiske omregningene til å finne hvor mye forstyrrelsen påvirker prosessen.  $G_{Tf}$  er i dette tilfellet to dager siden det tar to dager før forstyrrelsen påvirker prosessen etter settpunktendring.

I vedlegg D.2 kan man se at metoden er implementert. Verdiene som er vist i grafen er; vannivå i kilden (Waterlevel\_Pv), pådraget ut fra regulator (Waterlevel\_reg\_output) og settpunktet i vannkilden (Waterlevel\_Sv). Grafen viser at ved et gitt tidspunkt, når værmeldingen kommer, settes settpunktet ned slik at det forhindrer flom ved ekstrem nedbør. Etter settpunktendering åpner ventilen ved at pådraget fra regulator øker og tømmer vannkilden ned til gitt settpunkt. To dager senere kan man se at nedbøren kommer, og settpunktet settes tilbake.

### 6.3.2 Innregulering

For å innregulere anlegget er det valgt å bruke Ziegler-Nichols metode [45, s.44]. Regulatoren som er valgt er en PI-regulator. Grunnen for dette valget er at I-leddet trengs for å unngå stasjonært avvik [45, s.33]. D-leddet ikke er nødvendig siden det ikke er noe raske endringer i avviket på prosessen [45, s.37]. Prosessen i dette anlegget er treg og endringer i referansen er liten, det er heller ingen kritisk regulering så lenge nivået holder seg mellom alarmgrensene. I vedlegg D.1 viser det utklipt fra trend i Visualization. Det er dårlig oppløsning på prosessverdien grunnet tiden det tar for å sende simulerte verdier over MQTT.

Første steg i Ziegler-Nichols metode er å bruke bare P-leddet i regulatoren og sette forsterkningen,  $K_p$ , slik at prosessverdien oppnår stasjonære svingninger uten at pådraget går i metning, også kalt grensesvingninger. Etter man har funnet grensesvingningene reduserer  $K_p$  slik at man får oscillasjoner som dempes svært langsomt.  $K_k$  er den minste  $K_p$  verdien som gir grensesvingninger. Graf 1 i vedlegg D.1 viser grensesvingningene ved  $K_p = -460$ .  $K_p$  er negativ fordi regulatoren er bygd opp for generell bruk å finne avviket ved  $e = r - y$ , altså reversert modus [45, s.46]. Det vil si at avviket er positivt om settpunktet er over prosessverdien, noe som gjør at pådraget blir positivt. I dette tilfellet er det ønskelig å bruke direkte modus,  $e = y - r$  [45, s.47], fordi pådraget ut fra regulatorene bør være negativ dersom settpunktet er større en prosessverdien, slik at ventilen kan stenge og dermed fylle vannkilden. Siden regulatoren er bygd opp for generell bruk og benytter reversert modus vil det bli samme funksjon som en direkte modus ved å sette  $K_p$  negativ.

Etter at man har funnet grensesvingningene leses kritisk forsterkning ( $K_k$ ) og kritisk periodetid ( $T_k$ ) fra grafen. Kritisk forsterkning er den forsterkningen som gir grensesvingninger og kritisk periodetid er periodetiden ved grensesvingninger. Etter avlesning finner man verdiene:

$$K_k = -460 \quad T_k = 35s \quad (38)$$

Ut ifra tabell for innjustering etter Ziegler-Nichols metode finner man

parametrene for en PI-regulator med følgene utregninger [45, s.46] :

$$K_p = 0,45 \cdot K_k \quad T_i = 0,85 \cdot T_k \quad (39)$$

$$K_p = 0,45 \cdot -460 = -207 \quad T_i = 0,85 \cdot 35 = 29,75 \quad (40)$$

Normalt bør man regne ut regulatparametrene for flere arbeidspunkt i prosessen siden de sjeldent er lineære [45, s.45]. I dette tilfellet er ikke dette nødvendig siden prosessen vil arbeide stort sett rundt samme arbeidspunkt. Neste steg er å bruke disse utregnende verdiene i en PI-regulator. Graf 2 i vedlegg D.1 viser resultatet av dette. Prosessverdien svinger seg raskt inn på settpunkt, lite oscillasjoner i prosessverdien og ventilen jobber lite. Ved etterjustering kan innsvingningstiden reduseres litt, men det gjør at ventilen må jobbe mere. I denne prosessen har ikke innsvingningstiden så mye å si innen visse rammer og velger heller derfor å spare ventilen for unødvendig arbeid. Parametrene for PI-regulator er som følgende:

$$K_p = -207 \quad T_i = 30 \quad (41)$$

## 6.4 Diskusjon

Det har blitt undersøkt flere metoder for å håndtere forstyrrelser i denne deloppgaven. Først ble det sett på to kjente metoder, kaskaderegulering og foroverkobling. Til slutt ble det sett på settpunktendring for fremtidig forstyrrelse. I dette delkapittelet skal det komme frem hva som var bakgrunnen for at resultatet ble som det ble.

Foroverkobling krever at man hele tiden har en kontinuerlig måling på forstyrrelsen som påvirker prosessen [4, s.279]. I praksis er det utfordrende å måle  $q_{regn}$  og  $q_{tilsig}$  direkte i nåtid. Teoretisk sett er det også utfordrene å finne  $q_{regn}$  og  $q_{tilsig}$  helt korrekt, da de er avhengig av mange faktorer. På grunn av problemene med å finne forstyrrelsene korrekt i nåtid ble denne metoden vanskelig å implementere. Foroverkobling krever også en overføringsfunksjon for prosessen for å finne foroverkobling kontrolleren. I forbindelse med dette prosjektet der prosessen er valgt selv er ikke denne modellen vanskelig å finne, men kan være vanskelig å finne en nøyaktig overføringsfunksjon for en reell vannkilde. Foroverkobling blir mer nøyaktig dersom overføringsfunksjonen for prosessen er nøyaktig. Hovedgrunnen for at foroverkobling ble forkastet som en metode for håndtering av forstyrrelser er at man ikke hadde kontinuerlig måling på disse forstyrrelsene.

Kaskaderegulering trenger også en direkte måling av forstyrrelsen som påvirker prosessen. Kaskaderegulering brukes når det er en forstyrrelse som påvirker verdien pådragsorganet styrer [4, s.280]. Pådragsorganet i dette tilfellet er utslippsventilen som slipper ut vann om nivået er for høyt, og verdien pådragsorganet styrer er utstrømningen. Utstrømning ut fra utslippsventilen påvirkes ikke av innstrømningen i vannkilden, bortsett fra at trykket blir høyere dersom nivået stiger. På grunn av vannkildens

lave endring i nivå, ansees denne trykkøkningen som neglisjerbar. Siden trykkøkningen ikke påvirker verdien i pådragsorganet, ble ikke denne metoden implementert.

Settpunktendring for fremtidig forstyrrelse var den metoden som tilslutt ble valgt for håndtering av forstyrrelsen. Grunnen for at denne ble valgt er at den ikke krever kontinuerlig måling på forstyrrelsen, men trenger bare vite hvor mye forstyrrelsen påvirker prosessen på en dag for eksempel. Prosessen i dette tilfellet er følsom for oversving, da det er det som skaper flom, men er ikke følsom for undersving så lenge det holder seg rundt settpunktnivået. Det betyr at det ikke går utover prosessen om nivået senker seg for å kompansere mot fremtidig forstyrrelse, som gjør at denne metoden passer godt.

## 6.5 Konklusjon

Vannkilden er sikret mot flom ved at nivået senkes to dager i forveien, på bakgrunn av værmelding. Løsningen kan introdusere unødvendige svingninger i vannkilden, i tilfeller hvor værmeldingene er feil. Konsekvensene av disse svingningene er ansett som mildere enn flom i vannkilden. For vannkilden som er brukt er det tatt utgangspunkt i et ekte vann. Ved å gjøre dette bygges det mer troverdighet til metoden. På bakgrunn av dette konkluderes det med at vannkilden er sikret mot flom på en tilfredstillende måte.

Etter en del undersøkelse i ettertid ble det funnet en ny metode, som antageligvis ville fungert optimalt for denne problemstillingen. Metodene har forkortelse GPC og heter Generalized Predictive Control [51]. Denne metoden håndterer forstyrrelser som ligger i en prognose, slik som værmelding gjør. Som videre arbeid ville det vært fornuftig å undersøke GPC metoden å se om det er mulig å implementere denne i flomsikring. I denne problemstillingen har vi tatt hensyn til bare nedbør i regn. Som et annet punkt på videre arbeid kunne man ha sett på hvordan snøsmelting påvirker prosessen og funnet metoder for å håndtere denne forstyrrelsen også.

# 7 Trykkregulering

## 7.1 Introduksjon

Ved distribusjon av drikkevann er det nødvendig å opprettholde et konstant vanntrykk, slik at vannet kommer fram og forbrukeren har trykk i dusj og vannkraner. I dette kapittelet er det beskrevet hvordan tre pumper deler på jobben ved å opprettholde trykket på et ønsket nivå. Da denne delen av prosjektet har fått den laveste prioriteten, er fokuset på å få til noe som fungerer på demoriggen. Det er derfor brukt lite ressurser på forskjellige metoder.

## 7.2 Metoder og kilder

Ved programmering av PLS er arkitekturen og strukturen som er forklart i kapittel 3.8 fulgt. For kontrollering av pumpene er det brukt objektet "ControlAnalog". Objektet tilbyr funksjonaliteten det er behov for, som f.eks. automatisk og manuell kjøring. Pumpene blir kontrollert av en standard "PID" regulator, så lenge de er i auto. For å kunne fordele arbeidet er objektet "WorkDistributor" utviklet. Dette objektet tar kontroll over alle tre pumpene og bruker to til å opprettholde trykket. Objektet ser på hvor mange timer pumpene har vært i drift, og bruker de to pumpene som har den minste driftstiden. Endringen i arbeidsfordelingen er satt til å justeres hvert døgn, som i dette tilfellet er hvert minutt. Det er satt inn tre pumper, men i utgangspunktet brukes to, dette er begrunnet i fokus på oppetid. Med oppetid menes det hvor mange timer i året forbrukeren har tilgang på drikkevann med brukbart trykk. Ved å overdimensjonere systemet med tre pumper og bare bruke to er det mulig å drive full service på en pumpe uten at det blir noe redusering i oppetiden. I tillegg til denne funksjonaliteten finnes det simulerte innganger for trykket i renseanlegget, trykket hos forbrukeren, samt pådrags- og hastigetsverdier til pumper.

For metoden for innregulering av anlegget har det blitt sett på sprangresponsmetoden og Ziegler-Nichols metoden. Sprangresponsmetoden er forklart i neste kapittel og Ziegler-Nichols metode er allerede vist i kapittel 6.3.2 ved innregulering av flomsikringen.

Fremgangsmetoden for å finne parametrene for regulatoren ifølge sprangresponsmetoden er forklart i Reguleringsteknikk [45, s.52]. For å bruke denne oppskriften må man først ha en stabil prosess og deretter sette regulatoren i manuell for så å kjøre et sprang i pådrag på ca. 10%.

1. Trekk tangenten til prosessverdikurva der denne er brattest.
2. Trekk en forlengelse av prosessverdien fra før spranget kom.
3. Skjæringspunktet mellom disse to linjene marker slutten på tidforsinkelsen.
4. Tidsforsinkelse starter i det spranget i pådraget kommer.

5. Finn den ekvivalente tidsforsinkelsen,  $\tau_e$ .
6. Finn størrelsen på spranget i pådraget.
7. Finn ut hvor lang tid tangenten til prosessverdikurva bruker på å endre seg like mye som spranget i pådraget. Dette er den ekvivalente intigrasjonstiden,  $T_{ei}$ .

Etter at man har funnet  $\tau_e$  og  $T_{ei}$  bruker man disse verdiene i formlene nedenfor for å finne  $K_k$  og  $T_k$ .

$$K_k = \frac{2 \cdot T_{ei}}{\tau_e} \quad T_k = 4 \cdot \tau_e \quad (42)$$

Til slutt bruker man  $K_k$  og  $T_k$  i Ziegler-Nichols tommelfingerforslag [45, s.46] for å finne parametrerne for regulatoren man har valgt.

### 7.3 Resultater

For innregulering av trykkregulering er det valgt å bruke sprangresponsmetoden [45, s.50]. Resultatet og fremgangen for innregulering med sprangresponsmetoden er vist grafisk med grafer i vedlegg E.1 og blir beskrevet i neste avsnitt. For regulering av denne prosessen er det valgt å bruke PID regulator.

Graf 1 i vedlegg E.1 er det påtegnet resultatet etter å ha fulgt oppskriften for sprangresponsmetoden beskrevet i kapittel 7.2. Dette gir  $\tau_e = 1,38\text{sek}$  og  $T_{ei} = 2,77\text{sek}$ . Regner deretter ut  $K_k$  og  $T_k$  som videre skal brukes i Ziegler-Nichols tommelfingerforslag.

$$K_k = \frac{2 \cdot 2,77}{1,38} = 4,01 \quad T_k = 4 \cdot 1,38 = 5,52 \quad (43)$$

Bruker resultatet ovenfor i Ziegler-Nichols tommelfingerforslag for PID:

$$K_p = 0,65 \cdot K_k \quad T_i = 0,5 \cdot T_k \quad T_d = 0,12 \cdot T_k \quad (44)$$

$$K_p = 0,65 \cdot 4,01 = 2,61 \quad T_i = 0,5 \cdot 5,52 = 2,76 \quad T_d = 0,12 \cdot 5,52 = 0,66 \quad (45)$$

Graf 2 i vedlegg E.1 viser resultatet i et sprang etter å ha satt inn parametrerne i PID regulator funnet av sprangresponsmetoden. Graf 3 viser resultatet etter etterjustering med parametrerne  $K_p = 8$ ,  $T_i = 5,5$  og  $T_d = 0,5$ . Graf 4 viser resultatet etter innsetting av forstyrrelsen som er i graf 5. Man kan se at regulatoren greier å holde prosessverdien rundt referansen, men at prosessverdien og pådraget er hakkete, grunnen for dette sees på i diskusjonskapittelet.

Som nevnt deles jobben mellom de tre pumpene av en arbeidsdistributør. Denne arbeidsdistributøren er laget i form av en funksjonsblokk og finnes i vedlegg A.4.

## 7.4 Diskusjon

Ziegler-Nichols metode er ikke optimal å bruke som innreguleringsmetode på grunn av at den krever stående svingninger i prosessverdiene. For et reelt anlegg vil Ziegler-Nichols metode skape for mye oversving på trykket som kan skade rør og komponenter i distribusjonsnettet. Derfor er Ziegler-Nichols metode utelukket og sprangresponsmetoden er et godt alternativ, siden den ikke krever svingninger i prosessen over lengre tid. For regulering av denne prosessen er det valgt å bruke PID regulator. I-leddet er ønskelig slik at prosessen ikke ender opp med stasjonært avvik [45, s.33]. D-leddet hjelper til med å motvirke raske endringer i avviket [45, s.37]. Denne prosessen har raske endringer i avviket på grunn av trykkfallet som kommer av forbruket på distribusjonsnettet.

Etter innregulering med sprangresponsmetodene, der resultatet er vist i graf 2 i vedlegg E.1, er ikke innreguleringen optimal. Det er ønskelig med en raskere innreguleringstid uten større oversving. Det ble derfor forsøkt å etterjustere parametrerne. Graf 3 viser prosessen i et sprang etter etterjustering. Det kan ligne på at det er et lite stasjonært avvik, men det stemmer ikke. Det er grafen i Visualization som legger den blåe grafen ovenfor den røde selv om avviket er null. Når forstyrrelsen ble satt inn så man at grafene ble ujevn. Dette kommer av at forstyrrelsen ikke påvirker prosessen kontinuerlig, men har en samplingstid større en prosessen. Hvordan trykkfallet er simulert kan leses om i kapittel 4.5.1.

## 7.5 Konklusjon

Trykket til forbrukeren leveres av to pumper, med en i reserve. Arbeidet distribueres ut i fra hvor mange timer pumpene har vært i drift. Pumpene er innregulert og leverer stabilt trykk ved varierende forbruk. Forbruket er simulert basert på virkelige verdier. Det konkluderes derfor med at systemet oppnår kravene i problemstillingen i høy grad.

Etter innregulering er resultatet ansett som tilfredstillende, men poengterer at simuleringen av forstyrrelsen påvirker resultatet mer enn ønsket. Optimalt sett burde det vært forsket mer på hvilken metode som kan brukes for denne type regulering. Grafene etter innregulering hadde blitt glattere dersom det hadde vært et reelt anlegg med mulighet for å måle forstyrrelsen kontinuerlig.

Som videre arbeid i simuleringssammenheng kunne vært nyttig å lage en lengre liste med forstyrrelse verdier. Denne listen hadde ført til at forstyrrelsen hadde fått en mindre samplingstid og påvirket prosessen jevnere. Rent reguleringsteknisk var det ønskelig å implementere foroverkobling beskrevet i kapittel 6.2.1, men grunnet nedprioritering av denne delen, ble det ikke tid. Implementering av foroverkobling kan være et naturlig neste steg for videre arbeid.

## 8 Brukergrensesnitt

Da systemet skal leveres som en pakke til oppdragsgiver, kreves det at de tre hoveddelene enkelt kan monitoreres og manipuleres. Dette gjøres mulig ved hjelp av et brukergrensesnitt som gir operatøren av systemet en grafisk fremstilling av systemene. Brukergrensesnittet skal også gi muligheten til å lese av tilstander og endre på parametere. Systemet vil bestå av to brukergrensesnitt, e!COCKPIT og Grafana. Hvilke teknologier som ble brukt, hvordan og hvorfor designet fikk de spesifikke stilene, blir gjennomgått i dette kapittelet. For å få størst mulig utbytte av brukergrensesnitt kapittelet anbefales det å lese vedlagt brukermanualer for Grafana, vedlegg I.3, og WAGO e!COCKPIT Visualization, vedlegg I.4, først.

### 8.1 WAGO e!COCKPIT Visualization

#### 8.1.1 Introduksjon

For å produsere ett av brukergrensesnittene, er det valgt WAGOs egne programvare, e!COCKPIT, Visualization miljøet, som er hvordan e!COCKPIT visualiserer brukergrensesnittet. For å bygge brukergrensesnittet ble det designet unike bilder. Disse bildene er hva som visualiseres på panelet, altså det som gjør systemet interaktivt med operatøren. Bildene blir bygd i e!COCKPIT og lastes direkte ned til panelet. WAGOs programvare er basert på kjøremiljøet CODESYS. Dette ga muligheter og ressurser i form av CODESTORE som hadde flere tilgjengelige eksempler på funksjoner og et aktivt forum. I tillegg hadde oppdragstakerne WAGO Norge support tilgjengelig via mail og telefon, som kunne bistå ved behov. Sammen med bildene ligger det kode som skal detektere kommunikasjonsalarm og lignende. For å få dette til, måtte det legges til en e!RUNTIME lisens på panelet, slik at panelet kunne brukes som en PLS.

De seks bildene i visualization består av følgende:

- Systemoversikt - Har hensikt til å gi operatøren et raskt inntrykk over totaltilstanden i systemet
- Lekkasje- og distribusjonsoversikt - Delt mellom en link mot Grafana, som ivaretar lekkasjedekksjonen og et bilde av strømningsmålerne.
- Flomsikring - Værinformasjon for tre kommende dager og hva tilstanden er på vannkilden.
- Trykkregulering - Oversikt over pumpe- og trykktilstander.
- Alamer - Fellesbilde hvor alle alarmene og notifikasjonsendringene blir loggført, med mulighet for kvittere alarmer og hente ut historikk.
- Trender - For å lese av tilstander og pådrag over tid.

### **8.1.2 Objektstrukturer**

Under oppbyggingen av systemet er det bestemt at prosjektet benytter seg av en bestemt arkitektur, beskrevet i kapittel 3.8. Sensorer og pådragsorgan vil derav bygges opp som et objekt. Det vil si at all informasjon for en sensor vil skrives til et objekt med en bestemt struktur. Ved hjelp av objektstrukturen kan prosjektet enklere duplisere sensorer og holde oversikt. For å utnytte denne strukturen har systemet egne symboler for pådragsorganer og sensorer. Dermed ble det bygget et komplekst symbol for ventiler og pumper. Med komplekst symbol, menes det at symbolet indikerer alle tilstandene, åpen, feil og stengt sammen med variabler og konstanter som tilhører f.eks. ventilen.

### **8.1.3 Oppbygningen av brukergrensesnittet**

Før arbeidet med brukergrensesnittet ble påbegynt ble det lagt ned mye tid i design. Designet for hvert unike bilde ble skissert for hånd og deretter delt for tilbakemelding. Bildene ble først delt innad prosjektet som en første godkjenning. Da bildene ble godkjent innad i prosjektet ble de tegnet på nytt ved hjelp av Inkscape. Inkscape ble brukt for illustrere hvordan grafikken skulle bli. Illustrasjonene tegnet i Inkscape ble lagt til grunn for godkjenningen av oppdragsgiver. På denne måten ble det spart tid siden oppdragsgiver fikk et tydelig inntrykk om hvordan designet var tiltenkt og kunne dermed gi hyplige og konstruktive tilbakemeldinger. Resultatet ble et brukergrensesnitt som er delt opp i totalt syv unike skjermbilder. Hvor seks av disse bildene er rene e!COCKPIT bilder og det syvende er en peker mot Grafana.

For at systemet skal driftes trygt for personell og utstyr er det behov for et alarmsystem. Alarmsystemet er e!COCKPITS løsning, hvor en kan lage egne grupper og klasser for variablene som skal overvåkes. Alarmvariablene ble lagret i tre forskjellige alarmklasser. Alarmklassene forteller systemet om hvilken del av systemet alarmene hører til og hvilken prioritet de har, disse heter LeakDetection, FloodControl og PresControl. Tilhørende alarmklassene ble det dannet tre forskjellige alarmgrupper, for å sortere de forskjellige objektene alarmene tilhører. Alarmene blir konfigurert i disse klassene ved observere en alarmvariabel. På denne måten fikk prosjektet delt inn alarmene på unike alarmlister tilhørende de tre forskjellige systemene. Det ble gjennomgått om hvilke tilstander som krevde alarmer, for å redusere støy i form av unødvendige alamer for operatøren. Dermed ble det vedtatt internt i prosjektet at f.eks. strømningsmålerne i lekkasjedeksjonsbildet ikke skulle ha konfigurerde alarmer, da lekkasjedeksjonens strømningsmålere ikke har akutte egenskaper. Alarmene til pådragsorganene og tilstandsmålerne blir skrevet fra PLS'en til brukergrensesnittet, det vil si at PLS'en eier alarmflagget. På denne måten vil alarmgrensene være ivaretatt ved kommunikasjonsbrudd mellom brukergrensesnitt og system PLS. Hvis et pådragsorgan går i "fail safe" på grunnlag av en alarmgrense er brutt, vil ikke pådragsorganet

kunne driftes før en operatør har gjort tiltak for å stabilisere tilstanden som førte til alarm og deretter kvittert alarmen. Dette er for å unngå uønskede hendelser. Det er også konfigurert slikt at endringer av grenser og parametre blir loggført i en kollektiv liste for alle alarmklassene inne på alarmbildet. Dette gir en digital historikk av endringer operatører gjør.

Etterhvert som de unike systembildene kunne lese av variabler fra PLS'en og hadde interaktive objekter, ble det startet løpende tester. Testingen innebar sjekk av grafikk, hvor godt systemet kjørte, brukervennlighet, sjekk av funksjoner, alarmfunksjoner og endring i parametre. Testene ble koordinert mellom prosjektdeltagerne, for å holde kontroll på hvilken versjon av brukergrensesnittet som ble testet og resultatet. Disse testene ble loggført i et felles dokument med signert "testet" og dato. På denne måten unngikk prosjektet dobbel loggføring av feil og avvik, som førte til færre tidstyver. Ved å ha løpende tester av brukergrensesnittet opprettholdt prosjektet en fremgang på utviklingen og optimaliseringen av brukeropplevelsen.

#### 8.1.4 Beskrivelse av brukergrensesnitt

Dette underkapittelet tar for seg hvordan bildene er bygd opp og filosofien bak designet. Felles for alle bildene er at de har en tittelbar horisontalt øverst i bildet. Her kan operatøren manøvrere seg til menyen over tilgjengelige bilder, identifisere om det er initiert nye alarmer, hvilket bilde som er synlig, språkvalg og hvilken versjon av brukergrensesnittet som kjører. Menyen blir synlig ved å trykke på bildet helt til venstre på tittelbaren, illustrert ved fire firkanter i en kassestruktur. Her kan som nevnt operatøren manøvrere seg til følgende bilder:

System Oversikt/System Overview: Ved første oppstart møter operatøren "System Oversikt" bildet. Dette bilde gir operatøren et raskt inntrykk av hvordan den totale tilstanden er, altså et komprimert bilde som tar inn elementer fra de tre systemene som blir levert. På dette bildet er det en felles alarmliste hvor alle nyttige hendelser på tvers systemene blir samlet opp.

Lekkasjedeteksjon/Leak detection: Lekkasjedeteksjonsbildet blir kalt lekkasje og distribusjonsbildet. Dette er fordi selve lekkasjedeteksjonfunksjonen og varslingen ligger i Grafana. Derfor er det laget et "Distribusjonsbilde" som viser tilstander på strømningsmålere og batteripakker som er plassert i distribusjonsnettet. På denne måten kan operatøren kjapt få informasjon om alle målerne som er med i lekkasjedeteksjonsalgoritmen fungerer som normalt, slikt at algoritmen blir korrekt. Da bildet i menyen heter lekkasje og distribusjon, består bildet av to sider. I tittelbaren i toppen av bildet, kan operatøren manøvrere seg til lekkasjedeteksjonssiden. Ved å manøvrere seg hit, møter operatøren en peker til Grafana.

Flomsikring/Flood Control: Her har operatøren indikasjon på hvor mange meter vannsøyle og prosent det er i vannkilden, og hvor mye vann som går til utslipps som flomforhindrende tiltak. Operatøren får også

visualisert værmeldingen for tre kommende dager, som blir levert av yr.no.

Forbruketrykk/Consumer Pressure: Dette bildet viser vanntrykket fra renseanlegget, pumpetilstander og vanntrykk som leveres til forbrukernettet. Pumpetilstandene består blant annet av driftstiden og strømtrekket til pumpen. Dette er ment for å gi operatøren informasjon om hvor lenge pumpen har kjørt og kan forberede en vedlikeholdstopp.

Trend: For at operatøren kan se utvikling over tid, er det et trendbildet hvor operatøren kan visualisere variabler over tid. Dette ble løst ved å bruke e!COCKPITS egne trendfunksjoner.

Alarm: Alarmsiden samler hendelsene på samme måte som alarmlisten i system oversiktbildet. Her kan operatøren kvittere ut alarmer på tvers av systemene og hente ut hendelseshistorikken. Alarmlistene på de unike systembildene tar kun inn hendelser i korrespondanse med systembildet.

Popups: For å verne operatøren for visuell støy, er interaktiviteten mot pådragsorgan og sensorer begrenset til popups. Med dette menes det at operatøren må trykke på eksempelvis en pumpe, for å få synliggjort en meny hvor operatøren kan manipulere status og eventuelt utgangsverdi.

Widgets: For å gi bildene, System Oversikt, Lekkasjedeteksjon, Flomsikring og Forbruketrykk, dypere detaljer. Er det dannet widgets. Widgets er små og informative ruter som enkelt kan byttes ut på bildene ved hjelp av e!COCKPIT. Disse kan bestå av f.eks. trace av verdier og pumpestatuser.

Utklipp av systembildene inspiseres i vedlegg I.2, og hvordan operatøren kan operere systemet blir forklart i brukermanualen i vedlegg I.4.

### 8.1.5 Diskusjon

Designet ble direkte utviklet etter kundens forventninger og ønsker. Ønsket var å produsere en moderne og attraktiv stil. På tross at dette kunne stride mot kjent forskning for å oppnå et optimalt brukergrensesnitt. Det var en krevende oppgave for å få dette stilrent, med tanke på piksler og hvordan de forskjellige tilstandene og konstantene skulle visualiseres. Oppløsningen skulle tilpasses et 15,6" full HD panel, som ligger til grunn for den valgte oppløsningen 1920x1080. 15,6" panelet ble ikke levert i tide, som førte til at systemet ble levert med et 10" panel.

Alarmfilosofien i systemet fungerer slik at PLS'ene eier alarmene. Det vil si at HMI'en må sende et kvitteringsflagg til PLS'en for å behandle alarmene i PLS'en. Det ble forsøkt å finne løsninger ved å rute alarmstatusen fra en spesifikk ID direkte til kvitteringsflagget som skrives til PLS'en. Etter undersøkelser igjennom CODESYS sitt alarmeksempl, brukerforum og utforskning av programvaren, viste det seg at denne løsningen ikke ble vellykket. Det ble dermed programmert en egen algoritme for å sende et kvitteringsflag fra brukergrensesnittet til PLS'ene. Algoritmen baserer seg på at operatøren velger en alarm fra listen, trykker kvittere valgt og dermed vil skjermen kjøre en "WHILE-løkke" en gang. I "WHILE-løkken" er det tre forskjellige "CASE" betingelser. "CASE" er en utvidelse av "IF"-betingelse,

som blir aktivert etter hvilken alarmgruppe alarmen tilhører. Innad CASE-en blir kvitteringsflagget satt aktivt alt etter hva alarm ID'en tilsier.

### 8.1.6 Konklusjon

Oppdragstakerne anser resultatmålet med å produsere en visuell stil som oppleves moderne som vellykket. Brukergrensesnittet har oversiktlige bilder med tilpasset informasjon iht. hvilket system som visualiseres. Brukersnittet har logging på alarmer og hendelser, og sender kvitteringssignal til PLS'ene. Med logging av hendelser menes det at endring av moduser og parametre blir loggført i alarmlisten. Ved hjelp av estetikk, riktig fargevalg, fargekombinasjonene og et godt gjennomtenkt design oppleves bruakergrensesnittet som et moderne brukergrensesnitt. Grafikken oppleves som noe skjev og trykt ut på 10" skjerm. Det er ikke testet opp mot 15,6" skjerm for å se hvordan det endte opp, men det er testet på PC-monitor med full HD oppløsning. Med skjev menes det at f.eks. at teksten i indikatorrutene ikke har like mye "luft" mellom topp og bunn. Oppdragstakerne har ikke lyktes med å visualisere flere enn et popup-vindu av gangen. Dette resulterer i en HMI med mye trykking frem og tilbake ved driftsforstyrrelser. Da dette er en HMI som i grunn er tiltenkt demonstrasjon, er det ikke lagt mye tid for å optimalisere dette.

Alarmsystemet fungerer som tiltenkt, ved utvidelser til et større system kan kvitteringslogikken bli noe uoversiktlig for utvikleren.

CDP-studio er et verktøy for å bygge brukergrensenett til automasjon og industrielle systemer. Etter forskningen av CDP, kunne det antas at CDP kunne gjøre brukergrensesnittet så interaktivt som ønsket. CDP ble lagt til side til fordel for WAGOs e!COCKPIT og det ble besluttet at det ville tatt for mye tid å utvikle noe i CDP parallelt med e!COCKPIT. Videre arbeid hadde likevel vært å utforske CDP for å identifisere om utfordringene i e!COCKPIT kunne vært løst bedre i CDP.

## 8.2 Grafana

### 8.2.1 Introduksjon

Oppdragsgiver ønsket at en del av prosjektet skulle innebære å utforske Grafana. Etter å ha testet det ut ble det avgjort at det skulle brukes som brukergrensesnitt for en av deloppgavene, lekkasjedeteksjon.

Grafana er en open source analyse og interaktiv visualiserings programvare. Det brukes hovedsaklig til visualisering av data fra databaser. Grafana støtter flere databaser og blant annet MySQL som er valgt å bruke i dette prosjektet. Brukergrensesnittet i Grafana er bygget opp av et dashbord som inneholder et eller flere paneler. Panelene kan visualisere grafer, diagrammer, verdier og mye mer. Siden det er open source har samfunnet rundt Grafana laget flere tredjeparts paneler som er tilgjengelige. Det er også mulig og implementere alarmer i Grafana. For å få alarmer settes det opp alarmgrenser og dersom disse grensene blir brutt

genereres det en alarm. Foreløpig er det bare panelet med grafer som støtter alarmer.

### 8.2.2 Beskrivelese av brukergrensesnitt

Som nevnt i introduksjonen blir Grafana brukt som brukergrensesnitt for lekkasjedeksjon. Dette er gjort på grunn av at det er den delen i prosjektet som inneholder mest data. Det gjør at Grafana kan utforskes ved å visualisere dataen på forskjellige måter.

Det har vært fokus på å ikke bruke for mye farger i brukergrensesnittet, men å heller holde seg til rundt fire farger. Lyse grønn og rødt er forbeholdt bruk for noe som er bra eller dårlig. For eksempel blir lyse rødt brukt på alarmer og dersom avviket er alt for stort.

Det er ønskelig at panelene skal være selvforklarende, men med et såpass avansert system som lekkasjedeksjon har blitt, er det essensielt at hvert panel har fått "informasjon" knapp. Under informasjonsknappen står det bedre beskrevet hva panelene visualiserer. Brukergrensesnittet har en variasjon av flere paneler. Dette er for å vise hvilke muligheter man har i Grafana. Det er ikke støtte for å ha flerspråklig brukergrensesnitt i Grafana, det er derfor bare laget på norsk.

Brukergrensesnittet er delt opp i tre deler. I rapporten blir hensiktene med delene beskrevet, beskrivelse av delene er bedre beskrevet i brukermanualen, vedlegg I.3. I vedlegg I.1 er det vist utklipp fra Grafana, av de tre delene fra brukergrensesnittet.

- Oversiktsbilde
- Rørovervåkning
- Soneoversikt

Oversiktsbildet skal gi en rask oversikt over distribusjonsnettet slik at den viktigste informasjonen blir visualisert først. Dette er bildet operatøren normalt vil ha fremfor seg på en arbeidsdag. Det er derfor også hensiksmessig å ha alarmlisten for distribusjonsanlegget her. Kartet på oversiktsbilde er laget i SVG format med samme fargene som blir brukt i Grafana.

Rørovervåkning er satt sammen av fire paneler. Tre paneler er like og viser avviket i graf mellom inn- og utstrømning i fellesrør basert på volumbalansen beskrevet i kapittel 5.4.3. Det fjerde panelet viser også avviket, men i et bar panel. Med alle rørene samlet på samme panel blir det enklere å sammenligne avviket på rørene.

Soneoversikt består av fire rader. En rad for hver sone. Her blir metoden beskrevet i kapittel 5.4.2, lekkasjedeksjon basert på historisk data visualisert. Panelene viser også annen informasjon for sonene, slik som nåverdi og totalverdi, maks og min de siste 24 timene.

### 8.2.3 Konklusjon

Under arbeidet av brukergrensesnittene har WAGO e!COCKPIT Visualization hatt hovedprioritet. Selv om ikke Grafana har hatt størst prioritet har det blitt brukt en god del tid på et godt resultat og brukergrensesnitt. Grafana har visst seg å være et godt analyseverktøy og fungerer godt som et brukergrensesnitt i form av visualisering. Dersom man har en database er det enkelt å hente inn data og visualisere denne i Grafana. Designet i Grafan gir inntrykk av et moderne brukergrensesnitt som kan være ettertraktet i salgssammenheng. Etter å ha jobbet med Grafana gjennom prosjektpersonen er det også gjort noen negative oppdagelser. Man har merket flere mindre "bugs" som har gjort arbeidet mer krevende. Bedre brukermanualer og et mer aktivt forum, for å lettere få konfigurert programvaren, er savnet. Grafana oppleves som ressurskrevende. Ved ett lavt oppdateringsintervall og mye data på "dashboardet", oppleves brukergrensesnittet tregt og lite responsivt. Disse funnene er naturligvis avhengig av maskinvaren Grafana kjører på, men er basert på en maskin med 15GB RAM. Prosjektgruppen har utforsket Grafana og har samtidig bygget et eget brukergrensesnitt som er i henhold til ønskene og forventningene fra oppdragsgiver.

Som videre arbeid innenfor brukergrensesnittet i Grafana hadde det vært nyttig å innhente og visualisere mer sensor data, og dermed lage et komplett brukergrensesnitt av hele riggen. Siden dataen allerede ligger i databasen krever dette lite arbeid. Grafana har også annonsert at det kommer en ny versjon, versjon 7.0, med nye funksjoner. Når denne versjonen blir tilgjengelig hadde det vært interessant å utforsket de nye funksjonene.

# **9 Demorigg**

## **9.1 Introduksjon**

Et av hovedmålene ved denne oppgaven er å lage en demorigg. Denne demoriggen skal benyttes av oppdragsgiveren, for å vise fram utstyr, applikasjoner og løsninger for eksisterende og fremtidige kunder. Aktuelle bruksområder for en slik demorigg kan være fagmesser, kundesbesøk eller presentasjoner.

I dette kapittelet presenteres arbeidet som er gjort i forbindelse med demonstrasjonsriggen.

## **9.2 Planlegging**

Siden demoriggen skal brukes for å promotere oppdragsgivers produkter, er det viktig at de er med å bestemmer hvilke komponenter som skal benyttes. Det er gjerne noen komponenter de heller vil vise frem en andre. En del av komponentvalgene er tatt allerede i forprosjektet, deriblant at den nye 15,6" skjermen skal benyttes. Andre komponentvalg har vært mer selvforklarende, for eksempel i forhold til PLS, da det er kun PFC200 serien til Wago som har nok ressurser til å kjøre Docker.

Etter ønske fra oppdragsgiver er en eksisterende demonstrasjonsrigg benyttet som utgangspunkt for arbeidet. Denne ble demontert for alt av eksisterende utstyr og komponenter før de nye ble montert.

For å planlegge og dokumentere arbeidet på demoriggen er programmet Eplan Education benyttet. Dette er velkjent "CAD" program for elektro- og automasjonsbransjen, som er vederlagsfritt å bruke for studenter. Eplan er hovedsakelig brukt for å lage 3D arrangement tegning av riggens framplate i tillegg til elektriske koblingskjema. Hensikten med å lage en arrangement tegning er å verifisere at de komponentene man har planlagt å benytte, får plass på det området som er tilgjengelig. Vedlegg F.1 viser tegningen for framplaten av demoriggen.

En av fordelene med Eplan er at det har et bibliotek av komponenterliggende på internett, Eplan Data Portal. Her legger mange produsenter av utstyr, deriblant WAGO, ut ferdige 3D tegninger og macroer for sine respektive komponenter. Dette gjør at man enkelt kan sette inn ferdige komponenter fra Eplan Data Portal til prosjektet i Eplan. Disse komponentene har riktig opplosning i forhold til reell størrelse i tillegg til at deres elektriske egenskaper samsvarer med virkeligheten. Vedlegg F.2 viser elektriske skjema for demoriggen.

## **9.3 Resultat**

Den ferdige demoriggen vises i F.3. Man kan legge merke til at det er forholdsvis god plass omkring HMI skjermen. Dette er på grunn av at den planlagte 15,6" skjermen ikke var ferdig utviklet på det tidspunktet demoriggen ble satt sammen, og vil trenger større plass en 10" skjermen

som er montert. Området hvor den nye skjermen kommer er oppmerket med teip, og den vil skjule de fleste av hull fra tidligere oppsett. Som tidligere nevnt i kapittel 4 er alt av elektriske signaler i dette prosjektet simulert. Dette gjør at alle I/O modulene på PLS'ene og er å anse som en forberedelse til en eventuelt senere utvidelse av demoriggen. Som beskrevet i kapittel 3 er koden i PLS'ene til en viss grad forberedt for å bruke fysiske signaler i stedet for simulerte, og kan ved relativt enkle grep gjøres om. For deleliste henvises det til vedlegg F.4. Forbruksmateriell som din-skinner, kanaler, niter og lignende er ikke med i denne listen.

## 9.4 Konklusjon

Demoriggen fremstår som et godt og gjennomført produkt. Den viser frem nytt og avansert utstyr fra WAGO. Det er tatt høyde for at det skal monteres en annen skjerm enn den som står der i dag og det er ledig plass for oppdragsgiver til å tilføre mer utstyr om ønskelig. Arrangement-tegninger og elektriske skjema gjør det enkelt for brukeren og feilsøke og planlegge eventuelle utvidelser. Alt i alt en demonstrasjonsrigg som er godt egnet for framtidige messer og kundebesøk.

Videre arbeid med demonstrasjonsriggen vil være å integrere den tidligere nevnte 15,6" skjermen når den blir ferdigstilt fra oppdragsgivers fabrikk i Tyskland. En annen videreutvikling kan være integrasjon av en Eltorque ventil med CANBUS kommunikasjon. Dette var noe oppdragsgiver flagget som et ønske mot slutten av prosjektet.

# 10 Systemtesting

Systemet inneholder flere sensormålinger og pådragsvariabler. Disse kommuniseres mellom HMI og PLS'ene. For å forsikre oppdragstakere og oppdragsgiver at systemet fungerer som tiltenkt, er det utført en formell test. Testen verifiserer at verdier vises på riktig plass, parameterendringer blir utført og at feilhåndtering fungerer. Testen er lagt ved i vedlegg G.1. Hvordan og hvorfor testen er bygd opp, er forklart i dette kapittelet.

For at systemet skal fungere som tiltenkt må PLS'ene manipulere de riktige verdiene mot riktig pådrag. F.eks. må pumperguleringen vite trykket i linjen for å regulere pådraget. For å sikre dette har alle pådragsorganer og sensorer sine unike tag, som er ført opp i en signalliste for å holde oversikt. Signallisten kan inspiseres i vedlegg A.3. På grunn av arkitekturen som er nevnt i kapittel 3.8, vil det holde med å teste f.eks. en sensor grundig og deretter ta stikkprøver. Det er i tillegg kritisk at systemet er robust ved strømbrudd, siden det skal kobles opp og ned ved salgssammenheng. Testen tar ikke forbehold til sløyfetesting fra feltutsyr til inngang da alle verdier er simulerte. Med feltutsyr menes det sensorer og pådragsorgan. Som nevnt i innledningen i dette kapittelet vil det utføres en formell test. FAT'en er et dokument signert av oppdragstakere. Dette dokumentet viser testprosedyren som ivaretar alle funksjonene og feilhåndteringen

som systemet inneholder. Dokumentet er som nevnt en garanti for både oppdragstakere og oppdragsgiver, og bekrefter at systemet er testet og fungerer som tiltenkt.

## 11 Administrativt

### 11.1 Utviklingsmiljø

Tidlig i prosjektet ble det kartlagt at prosjektets bærbarer datamaskiner ikke hadde nok ressurser til å drive hovedprogramvaren i prosjektet, e!COCKPIT. Samtidig som COVID-19 gjorde at hele samfunnet stengte ned. For å få en god arbeidsflyt, og forholde seg til smittevernsreglene, ble det brukt ressurser på utviklingsmiljøet. Det ble raskt anskaffet en dedikert datamaskin, med nok ressurser til å drive hovedprogramvaren for alle prosjektdeltakerene, samtidig. Denne maskinen, heretter kalt serveren, ble satt opp slik at alle deltagerene kunne koble seg opp og jobbe direkte på den.

For å gjøre det mulig å koble seg til serveren fra internett på en sikker måte er det blitt implementert VPN. Det er installert en OPEN VPN server på serveren, og klienter på alle deltakerenes bærbarer maskiner, alle med sine egne nøkler. Da serveren står koblet opp til internett fra et hjemmenett med dynamisk IP, er det implementert en dynamisk DNS løsning. Løsningen gjør at det er mulig å koble seg opp til serveren med en url i steden for en IP, da IP adressen kan endre seg. Løsningen krever at det kjøres en dynamisk DNS klient på serveren. Denne klienten snakker med en server i skyen som sørger for å oppdatere IP adressen til url'en, når den endrer seg. For å rute trafikken som går gjennom VPN løsningen til serveren, er det satt opp port videresending i ruteren på hjemmenettet. Denne løsningen gjør at de bærbarer maskinene til prosjektets deltagere havner på samme lokale nettverket som serveren. Når denne forbindelsen er koblet opp, aksesseres serveren ved bruk av eksternt skrivebord. Dette gjør det mulig å kjøre programvaren på serveren som har ressurser nok og ikke på de bærbarer maskinene.

I prosjektet finnes det tre kontrollere og en trykkbar skjerm. Selv om disse kommuniserer med hverandre over internett, er det nødvendig å kunne koble seg opp via et lokalt nettverkt for å laste ned programmene som skal kjøre på de. Disse komponentene står derfor på samme hjemmenett som serveren, nevnt i forrige avsnitt. Dette gjør at alle prosjektdeltakerene har direkte tilgang til alle komponentene når de er koblet opp med eksternt skrivebord.

Da komponentene skal være på demoriggen vil de flyttes, noe som resulterer i at de må kunne operere med en dynamisk IP. Dette er ikke noe problem når systemet er satt i drift, for da brukes MQTT. Under utvikling er programvaren som skal laste ned kode til komponentene avhengig av en IP adresse til å sende koden mot. For å ha utviklingsmiljøet mest mulig likt miljøet det skal stå i når prosjektet er levert, er komponentene konfigurert til å motta IP adresse via DHCP i utviklingsmiljøet også. Utfordringen med

dette er at alle komponentene kan potensielt få en ny IP adresse hver gang de restartes. Dette er løst ved å konfigurere ruteren til å gi ut de samme IP adressene hver gang, dette gjøres ved å knytte komponentenes MAC adresse til spesifikke IP adresser. Når dette er konfigurert vil ruteren alltid gi den samme IP adressen til den MAC adressen den er knyttet til.

## 11.2 Prosjektstyring

I forprosjektet ble det nevnt flere metoder for å kvalitetsikre og holde styring på prosjektet fra start til slutt. Etter situasjonen med COVID-19 har disse metodene vært enda viktigere for å få et tilfredstillende resultat, da man stort sett har sittet hver for seg å jobbet med prosjektet. Dette har ført til et større behov for å holde kontroll på hva alle jobber med og fremgangen i prosjektet. I forprosjektet ble det satt opp Gantt-diagram i MS project som et hjelpemiddel for planlegging av prosjektet. Etter oppstart ble det etterhvert konkludert at oppdeling av resurser ikke var optimal. Planen var at flere personer skulle jobbe på hver arbeidspakke, men erfarte etterhvert at opplæringstiden var for ressurskrevende på noen av programmene. Dette har da ført til en del avvik i forhold til Gantt-diagrammet, slik som arbeidspakkene brukergrensesnitt og lekkasjedekksjon.

For å ha kontroll på alle dokumenter i prosjektperioden ble det tidlig opprettet en skybasert lagring med god mappestruktur i Google Disk. Fordelene med dette er selvsagt at dokumentene er sikkert lagret, men også at de er lett tilgjengelige. I tillegg til mappene som brukes under arbeid av prosjektet er det også opprettet en prosjektadministrasjons-perm (PA-perm). I PA-permen tilføres alt som blir ansett som resultat av prosjektet. Før forprosjektet ble påbegynt ble det satt opp en timeliste i Google Regneark. Formålet med timelisten er å holde kontroll på timeforbruket for hver person og timeforbruket opp mot arbeidspakkene som ble satt opp i forprosjektet. Listen brukes også som loggføring der hver deltaker har skrevet inn hva man har jobbet med for hver dag. Funksjonene i timelisten har gjort det enklere å se hvordan fremgangen i prosjektet er og hvor mange timer som er lagt ned i arbeidspakkene i forhold til planlagte timer. Timelisten ligger vedlagt i PA-permen.

Det har også blitt opprettet andre lister i Google Regneark, slik som signalliste, systemtestingliste, kapitteliste osv. Listene synkronises, slik at alle har tilgang til disse. Dette fungerer som ett ekstra hjelpemiddel for å holde kontroll på delt informasjon.

Daglige statusmøter har vært essensiell for å holde kontroll i prosjektet. Fra forprosjektet var det planlagt at disse møtene skulle være i rundt femten minutter der hver person skulle fortelle hva man jobbet med å hva man skulle jobbe med videre. På grunn av at man etterhvert satt hver for seg og ikke hadde mye kontakt angående prosjektet utenom disse møtene førte det til at møtetiden økte betraktelig. Her ble problemer man hadde hatt og trodde man kom til å få diskutert. Dersom det var større problemer eller diskusjoner der man trengte mere møtetid ble det satt opp tid til nytt møte.

Statusmøtene har blitt gjennomført på Whereby.

Etter hjelp fra veileder ble det tilordnet tilgang til arbeidsrom på NTNU. Nødvendighet for et slik arbeidsrom kom etterhvert som delene fra Wago ankom. Det var ønskelig med et felles møtested og et rom for å oppbevare delene etterhvert som de ble koblet opp. Etter å ha brukt rommet to ganger ble NTNU nedstengt grunnet COVID-19. For å bevare smittevernreglene ble det besluttet å flytte alle delene hjem til en av deltakerene og opprette et utviklingsmiljø der alle deltakerne hadde tilgang til delene over nettet. Selv om dette ikke var optimalt fungerte utviklingsmiljøet godt nok.

Det har ved jevne mellomrom blitt holdt møter med oppdragsgiver og veileder. Disse møtene har hovedsaklig blitt gjennomført for å ta større avgjørelser eller for å fremvise resultater. For hvert møte har det blitt utsendt møteinkalling før møtet og ettersendt møtereferat etter møtet. Møtene ble i starten holdt på NTNU, men ble etterhvert tatt over Skype grunnet COVID-19. Hver 14. dag, dersom det ikke har hvert møte i mellomtiden, har det også blitt sendt en statusoppdatering på mail til oppdragsgiver og veileder. Denne statusoppdateringen har inneholdt fremgangen for hver del i prosjektet og eventuelle avklaringer som må bli tatt. Møtereferat og møteinkallinger er vedlagt i PA-permen.

Et kanbanbrett har blitt brukt til statusoppfølging på hver del i prosjektet. Det har kontinuerlig blitt oppdatert slik at alle kunne følge med og komme med innspill. Det har også blitt brukt som en huskeliste da det er raskt gjort å opprette en deloppgaven som tilhører en hovedoppgave. Ved å se på kortene som er satt opp kan man enkelt se om ressursene er skeiwt fordelt, og kan dermed forflytte ressursene. Etterhver fikk også oppdragsgiver tilgang til brettet slik at han også hadde mulighet til å følge med å komme med innspill.

Hovedrapporten er skrevet i Latex, ved bruk av Overleaf, en nettbasert editor. Overleaf gir muligheten til at flere kan skrive på det samme dokument. Man har også muligheten til å kommentere i dokumentet slik at man kan foreslå endringer eller legge til innspill.

### 11.3 Timeoversikt

Gjennom hele prosjektperioden er det utført regelmessig loggføring ved hjelp av ett ark i Google Regnark. Arket er utviklet i prosjektet og bruker flere funksjoner for å holde kontroll på timeforbruk på hver person, men også for hver arbeidspakke som ble satt opp i forprosjektet. Arket med tilførte timer er vedlagt som vedlegg i PA-permen. Resultatet av timeforbruket på hver arbeidspakke ligger som vedlegg H.3. Som man kan se i vedlegg H.3 har ikke timeforbruket på hver arbeidspakket gått helt som planlagt. Selv om prosjektet består av flere deloppgaver ble det tidlig oppdaget at det måtte settes opp en prioriteringsliste. HMI og lekkasjedeteksjon ble satt med høyest prioritet grunnet ønske fra oppdragsgiver og mulighet for å jobbe parallelt med disse oppgavene. HMI har blitt jobbet med gjennom hele prosjektperioden. Etter lekkasjedeteksjon

ble flomsikring prioritert og deretter trykkregulering.

Overforbruket på HMI og lekkasjedeksjon kan delvis begrunnes av at de var først prioritert og fikk dermed mye utfordringer i forhold til oppstartskommunikasjon for hele systemet. Siden det ikke var satt opp en egen arbeidspakke på problemer rundt kommunikasjon og ytelse har disse timene blitt skrevet på den arbeidspakken man jobbet med akkurat da. Disse arbeidspakkene fikk også mye av opplæringestiden av diverse program, siden de var først prioritert. Det har vært en del utfordringer i forhold til utvikling av brukergrensesnittet som har krevd mer ytelse en hva man hadde tilgjengelig på private datamaskiner i starten. Etter at utviklingsmiljøet ble satt opp ble ytelseproblemet løst. For lekkasjedeksjon viste det seg å være vanskelig å løse problemstillingen rundt lekkasjedeksjon med de metodene som allerede var utviklet. Det ble derfor brukt en god del tid på å utvikle en egen metode.

Prosjektet innebar å teste ut ny teknologi. I hovedsak gjelder det kjøring av kontrolllogikk i Python, hvor Python kjører i en Docker kontainer. I henhold til timelisten er det brukt ca. 100 timer på denne testingen. Da resultatet av denne testingen førte til at løsningen ikke kunne brukes, er ikke disse timene direkte implementert i resultatet av prosjektet. Det ble tilegnet kunnskap som kunne brukes videre, men f.eks. koden som ble skrevet måtte skrives om igjen i e!COCKPIT.

Det ble som nevnt satt opp et utviklingsmiljø da samfunnet stengte ned. Disse timene er innbakt i de fleste arbeidspakkene. Dette er tid som er vanskelig å logge. Oppsettet kunne vært logget, men alle timene som har gått bort på grunn av dårlig internett forbindelse, eksternt skrivebord som mister tilkoblingen, programvarer som stopper, responsstider på fysiske restarter, uforventede hendelser, osv., er tid som har blitt implementert i den oppgaven man jobber med. I dette prosjektet har det blitt brukt flere forskjellige programvarer. Det har ført til at mye tid har gått til opplæring av disse programvarene.

Som nevnt i prosjektstyring, kapittel 11.2, ble det brukt en god del tid på møter etter COVID-19 stengte ned samfunnet. Totalt har det blitt brukt ca. 200 timer på møter.

Målet var å komme så langt som mulig på oppgavene gitt av oppdragsgiver ved å bruke det anbefalte timeforbruket gitt av NTNU. Anbefalt timeforbruket på en bachelor har NTNU satt til 500 timer per person. I vedlegg H.1 ser man at anbefalt timeforbruk på hver person er oppnådd.

Under prosjektperioden var det aldri noe problem med å oppnå 500 timer per person, da oppdragsgiver hele tiden kunne by på nye utfordringer. På grunn av COVID-19, ytelseproblemer og at andre oppgaver tok lengre tid en planlagt, slik som lekkasjedeksjon og brukergrensesnitt har ikke alle oppgavene blitt fullført som ønskelig. Dette gjelder i hovedsak brukergrensesnitt for bruk på mobil og brukergrensesnitt med et mer konservativt design.

I vedlegg H.2 vises det totale arbeidstimene brukt gjennom

prosjektperioden. I perioden 6.april - 12.april var det satt opp påskeferie og derfor lite aktivitet. I forhold til det planlagte timeforbruket, altså S-kurven, satt opp i forprosjekt og vist i vedlegg H.2, ser man at timeforbruket følger kurven godt.

## 12 Hovedkonklusjon

Det finnes delkonklusjoner i hvert av hovedkapitlene. I dette kapitlet samles disse delkonklusjonene til en hovedkonklusjon. Det legges vekt på i hvor stor grad problemstillingen, og dens spesifikasjoner, blir løst.

Det er utarbeidet et komplett styre- og overvåkingssystem. I tillegg er det utarbeidet funksjonalitet som simulerer en ekte prosess. Systemet som en helhet leveres i henhold til problemstillingen. Bruken av produktene i systemet er demonstrert i stor grad, i både sluttresultatet og i metodekapitlene. Store deler av teknologien som er implementert i produktene er testet ut i forskjellige deler av systemet. Alt fra forskjellige kjøremiljøer, MQTT kommunikasjon, brukergrensesnitt, til WAGO Cloud. Ytelsen og responstider er presset til maks. Det er gjort grep i algoritmer for å få ned responstider, da systemet ikke klarte å yte tilfredstillende. Etter finjusteringer av algoritmer, og kartlegging av ytre påvirkninger, konkluderes det med at systemet og produktene egner seg for distribusjon av drikkevann.

Drikkevannsdistribusjonen overvåkes for lekkasje, er sikret mot flom, leverer rett trykk til forbrukeren og kontrolleres via ett intuitivt brukergrensesnitt. Detaljene rundt dette finnes i delkonklusjonene i de gitte hovedkapitlene. Konklusjoner rundt kode, kontainermiljø, og kontinuerlig integrasjon er detaljert i kapittel 3.11. Detaljer rundt konklusjonene for de simulerte verdiene finnes i kapittel 4.10 og for visualisering i Grafana i kapittel 8.2.3.

Videre arbeide er spesifikt tatt opp i delkonklusjonen for de gitte systemene.

## Referanser

- [1] S. A. Valdor, “Lekkasjesøking på vannledninger med trykk.” <https://docplayer.me/15848893-Lekkasjesoking-pa-vannledninger-med-trykk-hvordan-lokalisere-lekkasjer.html>. (Accessed on 15/02/2020).
- [2] J.-H. Høvset, “Effektiv lekkasjelokalisering og valg av utstyr og metode for søk på lydsvakelekkasjer.” <https://docplayer.me/8258355-Eskeland-electronics-as.html>, 02 2015. (Accessed on 27/03/2020).
- [3] A. F. m. fl., “A171 - erfaringer med lekkasjekontroll,” tech. rep., Norsk Vann, 2009.
- [4] D. A. M. F. J. D. I. Dale E. Seborg, Thomas F. Edgar, *Process Dynamics and Control - 4th edition*. John Wiley & Sons, Inc, 2016.
- [5] “Open-source software - wikipedia.” [https://en.wikipedia.org/wiki/Open-source\\_software](https://en.wikipedia.org/wiki/Open-source_software). (Accessed on 01/24/2020).
- [6] “Cloud data control - fra feltnivået opp til skyen | wago no.” <https://www.wago.com/no/automasjon-teknologi/software/cloud-data-control>. (Accessed on 05/20/2020).
- [7] “What is factory acceptance testing, and how is fat done | carelabz.com.” <https://carelabz.com/what-factory-acceptance-testing-how-fat-done/>. (Accessed on 05/04/2020).
- [8] “What is mqtt? definition and details.” <https://www.paessler.com/it-explained/mqtt>. (Accessed on 01/28/2020).
- [9] “Virtual private network - wikipedia.” [https://en.wikipedia.org/wiki/Virtual\\_private\\_network](https://en.wikipedia.org/wiki/Virtual_private_network). (Accessed on 05/14/2020).
- [10] “Tcp (transmission control protocol) definition.” <https://techterms.com/definition/tcp>. (Accessed on 05/14/2020).
- [11] “Opc unified architecture - wikipedia.” [https://en.wikipedia.org/wiki/OPC\\_Unified\\_Architecture](https://en.wikipedia.org/wiki/OPC_Unified_Architecture). (Accessed on 05/14/2020).
- [12] “Real-time - wikipedia.” <https://en.wikipedia.org/wiki/Real-time>. (Accessed on 01/24/2020).
- [13] “Database – store norske leksikon.” <https://snl.no/database>. (Accessed on 05/14/2020).
- [14] “Docker - debug your app, not your environment.” <https://www.docker.com/>. (Accessed on 01/22/2020).
- [15] “Overview of docker compose | docker documentation.” <https://docs.docker.com/compose/>. (Accessed on 05/14/2020).

- [16] “WinSCP :: Official site :: Download.” <https://winscp.net/eng/download.php>. (Accessed on 05/14/2020).
- [17] “Suffix - wikipedia.” <https://en.wikipedia.org/wiki/Suffix>. (Accessed on 05/14/2020).
- [18] “What is the difference between a client, database and server? - quora.” <https://www.quora.com/What-is-the-difference-between-a-client-database-and-server>. (Accessed on 05/14/2020).
- [19] “Cloud computing services | google cloud.” <https://cloud.google.com/>. (Accessed on 05/14/2020).
- [20] “The leading operating system for pcs, iot devices, servers and the cloud | ubuntu.” <https://ubuntu.com/>. (Accessed on 05/14/2020).
- [21] “Adminer - database management in a single php file.” <https://www.adminer.org/>. (Accessed on 05/14/2020).
- [22] “Portainer management, docker user interface, container software - auckland, singapore, san francisco | emerging technology partners.” <https://www.portainer.io/>. (Accessed on 05/19/2020).
- [23] “What is a container? | app containerization | docker.” <https://www.docker.com/resources/what-container>. (Accessed on 05/19/2020).
- [24] “Gui (graphical user interface) definition.” <https://techterms.com/definition/gui>. (Accessed on 05/14/2020).
- [25] “Dynamic host configuration protocol - wikipedia.” [https://en.wikipedia.org/wiki/Dynamic\\_Host\\_Configuration\\_Protocol](https://en.wikipedia.org/wiki/Dynamic_Host_Configuration_Protocol). (Accessed on 05/14/2020).
- [26] “Ip address - wikipedia.” [https://en.wikipedia.org/wiki/IP\\_address](https://en.wikipedia.org/wiki/IP_address). (Accessed on 05/14/2020).
- [27] “Mac address - wikipedia.” [https://en.wikipedia.org/wiki/MAC\\_address](https://en.wikipedia.org/wiki/MAC_address). (Accessed on 05/14/2020).
- [28] “Draw freely | inkscape.” <https://inkscape.org/>. (Accessed on 05/14/2020).
- [29] “Application programming interface - wikipedia.” [https://en.wikipedia.org/wiki/Application\\_programming\\_interface](https://en.wikipedia.org/wiki/Application_programming_interface). (Accessed on 05/14/2020).
- [30] “Json - wikipedia.” <https://en.wikipedia.org/wiki/JSON>. (Accessed on 05/14/2020).
- [31] “Cdp technologies as – control and monitoring systems.” <https://cdpstudio.com/>. (Accessed on 05/16/2020).

- [32] E. S. og Eva Schwencke, *Prosjektarbeid - En veiledning for studenter.* nki, 2001.
- [33] “jonev/drinking-water-distribution-demo-ba-46-wago-ntnu-2020 at v1.0.0.” <https://github.com/jonev/drinking-water-distribution-demo-ba-46-wago-ntnu-2020/tree/v1.0.0>. (Accessed on 05/19/2020).
- [34] “Difference between high level and low level languages - geeksforgeeks.” <https://www.geeksforgeeks.org/difference-between-high-level-and-low-level-languages/>. (Accessed on 05/03/2020).
- [35] “Virtualization via containers.” [https://insights.sei.cmu.edu/sei\\_blog/2017/09/virtualization-via-containers.html](https://insights.sei.cmu.edu/sei_blog/2017/09/virtualization-via-containers.html). (Accessed on 05/04/2020).
- [36] “Publish & subscribe - mqtt essentials: Part 2.” <https://www.hivemq.com/blog/mqtt-essentials-part2-publish-subscribe/>. (Accessed on 04/13/2020).
- [37] “What is hashing? | binance academy.” <https://www.binance.vision/security/what-is-hashing>. (Accessed on 05/04/2020).
- [38] “Basics of hash tables tutorials & notes | data structures | hackerearth.” <https://www.hackerearth.com/practice/data-structures/hash-tables/basics-of-hash-tables/tutorial/>. (Accessed on 05/04/2020).
- [39] “Object-oriented programming - simple english wikipedia, the free encyclopedia.” [https://simple.wikipedia.org/wiki/Object-oriented\\_programming](https://simple.wikipedia.org/wiki/Object-oriented_programming). (Accessed on 05/08/2020).
- [40] “Quality of service 0.1 & 2 - mqtt essentials: Part 6.” <https://www.hivemq.com/blog/mqtt-essentials-part-6-mqtt-quality-of-service-levels/>. (Accessed on 05/03/2020).
- [41] “Freeopcua/python-opcua: Lgpl pure python opc-ua client and server.” <https://github.com/FreeOpcUa/python-opcua>. (Accessed on 04/16/2020).
- [42] “Docker cross cpu architecture builds with travis ci and buildx.” <https://medium.com/@quentin.mcgaw/cross-architecture-docker-builds-with-travis-ci-arm-s390x-etc-8f754e20aaef>. (Accessed on 05/09/2020).
- [43] “Get the forecast from the norwegian wheather service yr.no in python.” <https://pypi.org/project/python-yr/>. (Accessed on 12/02/2020).
- [44] “Effect of the weather on solar panels.” <https://www.leadingedgepower.com/shop/help-advice-faq/help-with-solar-panels/effect-of-weather-on-solar-panels.html>. (Accessed on 04/05/2020).

- [45] P. H. Kåre Bjørvik, *Reguleringssteknikk*. Kybernetes forlag, 2014.
- [46] F. White, *Fluid Mechanics - 8th edition*. McGraw-Hill Education, 2015.
- [47] “(13) e!cockpit native mqtt tutorial - youtube.” <https://www.youtube.com/watch?v=F7tVQ5xspAc>. (Accessed on 05/01/2020).
- [48] “Eulers metode – store norske leksikon.” [https://snl.no/Eulers\\_metode](https://snl.no/Eulers_metode). (Accessed on 05/14/2020).
- [49] P. Qvale, “Med dagens tempo vil det ta 50 år å fikse vann- og avløpsnettet i norge.” <https://www.tu.no/artikler/med-dagens-tempo-vil-det-ta-50-ar-a-fikse-vann-og-avlopsnettet-i-norge/275648>. (Accessed on 04/30/2020).
- [50] “Lekkasjekontroll for dummies - godt vann drammensregionen.” <https://www.godtvann.no/lekkasjesoek-for-dummies>. (Accessed on 05/14/2020).
- [51] F. R. M. B. J. E. N.-R. Andrzej Pawłowski, Jose L. Guzman, “Predictive control with disturbance forecasting for greenhouse diurnal temperature control,” tech. rep., University of Almeria, 2011.

## **A Vedlegg system**

### **A.1 Globale objekter**

## Globale objekter

Implementert i HMI

Block name:	TimerHmiPv	Block name:	TimerHmiCmd
Tag name	Tag type	Tag name	Tag type
_tagId	STRING	_tagId	STRING
_type	STRING	_type	STRING
_owner	STRING	_owner	STRING
Input_Pv	BOOL	Tag	STRING
Output_Pv	BOOL	Description	STRING
OnDelayElapsed	REAL	OnDelayLimit	REAL
OffDelayElapsed	REAL	OffDelayLimit	REAL
Block name:	AlarmDigitalHmiPv	Block name:	AlarmDigitalHmiCmd
Tag name	Tag type	Tag name	Tag type
_tagId	STRING	_tagId	STRING
_type	STRING	_type	STRING
_owner	STRING	_owner	STRING
Input_Pv	BOOL	Tag	STRING
Delay	TimerHmiPv	Description	STRING
Alarm	BOOL	Delay	TimerHmiCmd
		Acknowledged	BOOL
Block name:	ControlAnalogHmiPv	Block name:	ControlAnalogHmiCmd
Tag name	Tag type	Tag name	Tag type
_tagId	STRING	_tagId	STRING
_type	STRING	_type	STRING
_owner	STRING	_owner	STRING
ControlValue_Pv	REAL	Tag	STRING
Interlocked_Pv	BOOL	Description	STRING
AlarmControlFailed	AlarmDigitalHmiPv	Auto	BOOL
Feedback_Pv	REAL	ManualControlValue_C	REAL
RunHours_Pv	INT	AlarmControlFailed	AlarmDigitalHmiCmd
RunMinutes_Pv	INT	AlarmControlFailedLim	REAL
Block name:	MotorDigitalHmiPv	Block name:	MotorDigitalHmiCmd
Tag name	Tag type	Tag name	Tag type
_tagId	STRING	_tagId	STRING
_type	STRING	_type	STRING
_owner	STRING	_owner	STRING
Interlocked_Pv	BOOL	Tag	STRING
ControlValue_Pv	BOOL	Description	STRING
AlarmStartFailed	AlarmDigitalHmiPv	Auto	BOOL
AlarmStopFailed	AlarmDigitalHmiPv	AlarmStartFailed	AlarmDigitalHmiCmd
Started_Pv	BOOL	AlarmStopFailed	AlarmDigitalHmiCmd
Stopped_Pv	BOOL	ManualStart_Cmd	BOOL

Block name:	ScalingHmiPv	Block name:	ScalingHmiCmd
Tag name	Tag type	Tag name	Tag type
_tagId	STRING	_tagId	STRING
_type	STRING	_type	STRING
_owner	STRING	_owner	STRING
Input_Pv	REAL	InputUnit	STRING
Output_Pv	REAL	InputMin_Cmd	REAL
		InputMax_Cmd	REAL
		OutputUnit	STRING
		OutputMin_Cmd	REAL
		OutputMax_Cmd	REAL
Block name:	SignalDigitalHmiPv	Block name:	SignalDigitalHmiCmd
Tag name	Tag type	Tag name	Tag type
_tagId	STRING	_tagId	STRING
_type	STRING	_type	STRING
_owner	STRING	_owner	STRING
Input_Pv	BOOL	Tag	STRING
Output_Pv	BOOL	Description	STRING
		SimulationMode	BOOL
		SimulationValue	BOOL
Block name:	SignalAnalogHmiPv	Block name:	SignalAnalogHmiCmd
Tag name	Tag type	Tag name	Tag type
_tagId	STRING	_tagId	STRING
_type	STRING	_type	STRING
_owner	STRING	_owner	STRING
RawInput_Pv	REAL	Tag	STRING
Output_Pv	REAL	Description	STRING
AlarmHigh	AlarmDigitalHmiPv	AlarmHigh	AlarmDigitalHmiCmd
AlarmHighHigh	AlarmDigitalHmiPv	AlarmHighHigh	AlarmDigitalHmiCmd
AlarmLow	AlarmDigitalHmiPv	AlarmLow	AlarmDigitalHmiCmd
AlarmLowLow	AlarmDigitalHmiPv	AlarmLowLow	AlarmDigitalHmiCmd
InputScaling	ScalingHmiPv	AlarmHighLimit	REAL
		AlarmHighHighLimit	REAL
		AlarmLowLimit	REAL
		AlarmLowLowLimit	REAL
		AlarmHysteresis	REAL
		SimulationMode	BOOL
		SimulationValue	REAL
		InputScaling	ScalingHmiCmd
Block name:	AliveHmiPv		
Tag name	Tag type		
_tagId	STRING		

_type	STRING
_owner	STRING
CounterPv	INT

Block name:	PIDHmiPv	Block name:	PIDHmiCmd
Tag name	Tag type	Tag name	Tag type
_tagId	STRING	_tagId	STRING
_type	STRING	_type	STRING
_owner	STRING	_owner	STRING
InputReference	REAL	Tag	STRING
ProcessMeasure	REAL	Description	STRING
Error	REAL	K	REAL
Output	REAL	EnableK	BOOL
up	REAL	Ti	REAL
ui	REAL	EnableI	BOOL
ud	REAL	Td	REAL
es	REAL	N	REAL
u	REAL	EnableD	BOOL
		Auto	BOOL
		ManualValue_Cmd	REAL
		u0	REAL
		TrackingGain	REAL
		HmiReference_Cmd	REAL
		HmiReference	BOOL

Block name:	WorkDistributorHmiPv
Tag name	Tag type
_tagId	STRING
_type	STRING
_owner	STRING
Tag1	STRING
Tag2	STRING
Tag3	STRING
RunHours1	INT
RunHours2	INT
RunHours3	INT
AlarmNotAuto	BOOL

## A.2 MQTT emne oversikt

MQTT Topic Overview			
What	Topic	Publishers	Subscribers
All values on opc ua	ba/wago/opcua/plc3/plcpub	PLC3	HMI
All values on opc ua	ba/wago/opcua/plc3/plcsub	HMI	PLC3
All values on opc ua	ba/wago/opcua/plc2/plcpub	PLC2	HMI
All values on opc ua	ba/wago/opcua/plc2/plcsub	HMI	PLC2
All values on opc ua	ba/wago/opcua/plc1/plcpub	PLC1	HMI
All values on opc ua	ba/wago/opcua/plc1/plcsub	HMI	PLC1
Values from PLC 1 sent to Simulation	ba/wago/sim/plc1/plcpub	PLC1	SIM
Simulated values sent to PLC 1	ba/wago/sim/plc1/plcsub	SIM	PLC1
Values from PLC 2 sent to Simulation	ba/wago/sim/plc2/plcpub	PLC2	SIM
Simulated values sent to PLC 2	ba/wago/sim/plc2/plcsub	SIM	PLC2
Values from PLC 3 sent to Simulation	ba/wago/sim/plc3/plcpub	PLC3	SIM
Simulated values sent to PLC 3	ba/wago/sim/plc3/plcsub	SIM	PLC3
Simulation program information	ba/wago/sim/info	SIM	
Weather from python to hmi	ba/wago/sim/hmi/hmisub		HMI

### **A.3 Signalliste**

Signalliste				
BA	Datatype	Tags	Connected to	
<b>PLS1</b>				<b>PLS1</b>
AliveCounterPlc1Pv	AliveHmiPv			PLS1
FlowLevel1Cmd	SignalAnalogHmiCmd	FT01	PLS1	
FlowLevel1Pv	SignalAnalogHmiPv	FT01	PLS1	
FlowLevel2Cmd	SignalAnalogHmiCmd	FT02	PLS1	
FlowLevel2Pv	SignalAnalogHmiPv	FT02	PLS1	
FlowLevel3Cmd	SignalAnalogHmiCmd	FT03	PLS1	
FlowLevel3Pv	SignalAnalogHmiPv	FT03	PLS1	
FlowLevel4Cmd	SignalAnalogHmiCmd	FT04	PLS1	
FlowLevel4Pv	SignalAnalogHmiPv	FT04	PLS1	
FlowLevel5Cmd	SignalAnalogHmiCmd	FT05	PLS1	
FlowLevel5Pv	SignalAnalogHmiPv	FT05	PLS1	
FlowLevel6Cmd	SignalAnalogHmiCmd	FT06	PLS1	
FlowLevel6Pv	SignalAnalogHmiPv	FT06	PLS1	
FlowLevel7Cmd	SignalAnalogHmiCmd	FT07	PLS1	
FlowLevel7Pv	SignalAnalogHmiPv	FT07	PLS1	
BatteryMonitoring1Cmd	PowerMonitoringHmiCmd	BM01	PLS1	
BatteryMonitoring1Pv	PowerMonitoringHmiPv	BM01	PLS1	
BatteryMonitoring2Cmd	PowerMonitoringHmiCmd	BM02	PLS1	
BatteryMonitoring2Pv	PowerMonitoringHmiPv	BM02	PLS1	
BatteryMonitoring3Cmd	PowerMonitoringHmiCmd	BM03	PLS1	
BatteryMonitoring3Pv	PowerMonitoringHmiPv	BM03	PLS1	
<b>PLS2</b>				<b>PLS2</b>
AliveCounterPlc2Pv	AliveHmiPv			PLS2
DeliveryPump1Cmd	ControlAnalogHmiCmd	P01	PLS2	
DeliveryPump1Pv	ControlAnalogHmiPv	P01	PLS2	
DeliveryPump2Cmd	ControlAnalogHmiCmd	P02	PLS2	
DeliveryPump2Pv	ControlAnalogHmiPv	P02	PLS2	
DeliveryPump3Cmd	ControlAnalogHmiCmd	P03	PLS2	
DeliveryPump3Pv	ControlAnalogHmiPv	P03	PLS2	
DeliveryPumpPIDCmd	PIDHmiCmd	PID01	PLS2	
DeliveryPumpPIDPv	PIDHmiPv	PID01	PLS2	
DeliveryPumpWorkDistPv	WorkDistributorHmiPv	WD01	PLS2	
TreatmentPressureCmd	SignalAnalogHmiCmd	PT02	PLS2	
TreatmentPressurePv	SignalAnalogHmiPv	PT02	PLS2	
ConsumerPressureCmd	SignalAnalogHmiCmd	PT03	PLS2	
ConsumerPressurePv	SignalAnalogHmiPv	PT03	PLS2	
<b>PLS3</b>				<b>PLS3</b>
AliveCounterPlc3Pv	AliveHmiPv			PLS3
ValveEmission1Cmd	ControlAnalogHmiCmd	VE01	PLS3	
ValveEmission1Pv	ControlAnalogHmiPv	VE01	PLS3	
ValveEmission1AlarmMotorProtectionCmd	AlarmDigitalHmiCmd	VE01	PLS3	

Signalliste			
BA	Datatype	Tags	Connected to
ValveEmission1AlarmMotorProtectionPv	AlarmDigitalHmiPv	VE01	PLS3
ValveEmission1SignalDigitalMotorProtectionCmd	SignalDigitalHmiCmd	VE01	PLS3
ValveEmission1SignalDigitalMotorProtectionPv	SignalDigitalHmiPv	VE01	PLS3
ValveEmission1ScalingPositionCmd	ScalingHmiCmd	VE01	PLS3
ValveEmission1ScalingPositionPv	ScalingHmiPv	VE01	PLS3
ValveEmission1ScalingControlValueCmd	ScalingHmiCmd	VE01	PLS3
ValveEmission1ScalingControlValuePv	ScalingHmiPv	VE01	PLS3
ValveEmission1PIDCmd	PIDHmiCmd	PID02	PLS3
ValveEmission1PIDPv	PIDHmiPv	PID02	PLS3
WaterLevelCmd	SignalAnalogHmiCmd	PT01	PLS3
WaterLevelPv	SignalAnalogHmiPv	PT01	PLS3
FlowLevelEmmissionCmd	SignalAnalogHmiCmd	FT08	PLS3
FlowLevelEmmissionPv	SignalAnalogHmiPv	FT08	PLS3

## **A.4 PLS Objektbibliotek**

## Project Documentation

File: Objekt Bibliotek 09.05.2020.ecp

Date: 5/9/2020

Profile: e!COCKPIT

---

Table of Contents

---

## Table of Contents

1 Folder: LibFB	4
1.1 POU: AlarmDigitalFB	4
1.2 POU: ControlAnalogFB	5
1.3 POU: MotorDigitalFB	6
1.4 POU: PIDFB	8
1.5 POU: PowerMonitoringFb	10
1.6 POU: ScalingFB	12
1.7 POU: SignalAnalogFB	12
1.8 POU: signalDigitalFB	15
1.9 POU: TimerFB	16
1.10 POU: WorkDistributorFB	17
2 Folder: LibHmiStructs	19
2.1 DUT: AlarmDigitalHmiCmd	19
2.2 DUT: AlarmDigitalHmiPv	19
2.3 DUT: AliveHmiPv	20
2.4 DUT: ControlAnalogHmiCmd	20
2.5 DUT: ControlAnalogHmiPv	20
2.6 DUT: MotorDigitalHmiCmd	20
2.7 DUT: MotorDigitalHmiPv	21
2.8 DUT: PIDHmiCmd	21
2.9 DUT: PIDHmiPv	22
2.10 DUT: PowerMonitoringHmiCmd	22
2.11 DUT: PowerMonitoringHmiPv	22
2.12 DUT: ScalingHmiCmd	23
2.13 DUT: ScalingHmiPv	23
2.14 DUT: SignalAnalogHmiCmd	23
2.15 DUT: SignalAnalogHmiPv	24
2.16 DUT: SignalDigitalHmiCmd	24
2.17 DUT: SignalDigitalHmiPv	24
2.18 DUT: TimerHmiCmd	25
2.19 DUT: TimerHmiPv	25
2.20 DUT: WorkDistributorHmiPv	25
3 Folder: Utils	26
3.1 DUT: OpcUa_Mqtt_Status	26
3.2 POU: eulers	26
3.3 POU: firstOrderSystem	26
3.4 POU: process	27
3.5 POU: String_To_ForecastArray	27

## 1 Folder: LibFB

## 1 Folder: LibFB

## 1.1 POU: AlarmDigitalFB

```

1 FUNCTION_BLOCK AlarmDigitalFB
2 VAR_INPUT
3   _id : STRING ;
4   _owner : STRING ;
5   Input : BOOL ;
6 END_VAR
7 VAR_OUTPUT
8   Alarm : BOOL ;
9 END_VAR
10 VAR
11   _Delay : TimerFB ;
12   _AlarmLastScan : BOOL ;
13   _type : string := 'AlarmDigital' ;
14 END_VAR
15 VAR_IN_OUT
16   HmiCmd : AlarmDigitalHmiCmd ;
17   HmiPv : AlarmDigitalHmiPv ;
18 END_VAR
19

20
21 (*
22 Name: AlarmDigitalFB
23 Author: Jone Vassbø
24 Version: 0.3
25 Date: 09.03.2020
26 Description:
27 - Function block for a digital alarm, with commands from HMI in seperate
28 struct
29 *)
30
31 // In routine - start -
32 // Validate input
33 // In routine - end -
34 // Alarm routine - start
35 // Alarm routine - end -
36
37 // Control routine - start -
38 _Delay(_id := _id, _owner := _owner, Input := Input, Output => Alarm
39 , HmiCmd := HmiCmd.Delay, HmiPv := HmiPv.Delay) ;
40 IF Alarm AND NOT _AlarmLastScan THEN // Flank
41   HmiCmd.Acknowledged := FALSE ;
42 END_IF
43 _AlarmLastScan := Alarm ;
44 // Control routine - end -
45
46 // Out routine - start -
47 HmiPv.Input_Pv := Input ;

```

---

 1.1 POU: AlarmDigitalFB
 

---

```

26      HmiPv . Alarm := Alarm OR NOT HmiCmd . Acknowledged ;
27      HmiCmd . _tagId := CONCAT (_id , 'Cmd' ) ;
28      HmiPv . _tagId := CONCAT (_id , 'Pv' ) ;
29      HmiPv . _type := _type ;
30      HmiCmd . _type := _type ;
31      HmiPv . _owner := _owner ;
32      HmiCmd . _owner := _owner ;
33 // Out routine - end -
34

```

## 1.2 POU: ControlAnalogFB

```

1   FUNCTION_BLOCK ControlAnalogFB
2     VAR_INPUT
3       _id : STRING ;
4       _owner : STRING ;
5       ControlValueAuto_Cmd : REAL ;
6       Interlock : BOOL ;
7       Feedback : REAL ;
8     END_VAR
9     VAR_OUTPUT
10    ControlValue : REAL ;
11    Alarm : BOOL ;
12  END_VAR
13  VAR
14    _AlarmControlFailed : AlarmDigitalFB ;
15    _hoursTimer : TON ;
16    _type : STRING := 'ValveAnalog' ;
17  END_VAR
18  VAR_IN_OUT
19    HmiCmd : ControlAnalogHmiCmd ;
20    HmiPv : ControlAnalogHmiPv ;
21  END_VAR
22

```

---

```

1   /*
2    * Name: ValveAnalogFB
3    * Author: Jone Vassbø
4    * Version: 0.3
5    * Date: 09.03.2020
6    * Description:
7    * - Function block to control a analog valve, with commands from HMI in
8    *   seperate struct
9    */
10   // In routine - start -
11   // Validate input
12   IF HmiCmd . ManualControlValue_Cmd > 100.0 THEN
13     HmiCmd . ManualControlValue_Cmd := 100.0 ;
14   END_IF
15   IF HmiCmd . ManualControlValue_Cmd < 0.0 THEN
16     HmiCmd . ManualControlValue_Cmd := 0.0 ;
17   END_IF
18   IF ControlValueAuto_Cmd > 100.0 THEN
19     ControlValueAuto_Cmd := 100.0 ;

```

## 1.2 POU: ControlAnalogFB

```

20      END_IF
21      IF ControlValueAuto_Cmd < 0.0 THEN
22          ControlValueAuto_Cmd := 0.0 ;
23      END_IF
24      // In routine - end -
25      // Alarm routine - start
26      _AlarmControlFailed (_id := _id, _owner := _owner, Input := ABS (
27          Feedback - ControlValue ) > HmiCmd . AlarmControlFailedLimit ,
28          HmiCmd := HmiCmd . AlarmControlFailed , HmiPv := HmiPv .
29          AlarmControlFailed ) ;
30      // Alarm routine - end -
31
32      // Control routine - start -
33      IF Interlock OR Alarm THEN
34          HmiCmd . Auto := FALSE ;
35          HmiCmd . ManualControlValue_Cmd := 0.0 ;
36      END_IF
37
38      IF HmiCmd . auto THEN
39          ControlValue := ControlValueAuto_Cmd ;
40      ELSE
41          ControlValue := HmiCmd . ManualControlValue_Cmd ;
42      END_IF
43
44      _hoursTimer ( In := ( ControlValue > 0.3 ) AND NOT _hoursTimer . Q , PT :=
45          T#0.041666667S ) ;
46      // Using HmiPv because the tag is retain
47      IF _hoursTimer . Q THEN
48          HmiPv . RunMinutes_Pv := HmiPv . RunMinutes_Pv + 1 ;
49      END_IF
50      IF HmiPv . RunMinutes_Pv = 60 THEN
51          HmiPv . RunHours_Pv := HmiPv . RunHours_Pv + 1 ;
52          HmiPv . RunMinutes_Pv := 0 ;
53      END_IF
54      IF HmiPv . RunHours_Pv < 0 THEN
55          HmiPv . RunHours_Pv := 0 ;
56      END_IF
57
58      // Control routine - end -
59
60      // Out routine - start -
61      Alarm := _AlarmControlFailed . Alarm OR NOT HmiCmd . AlarmControlFailed .
62      Acknowledged ;
63      HmiPv . ControlValue_Pv := ControlValue ;
64      HmiPv . Interlocked_Pv := Interlock ;
65      HmiPv . Feedback_Pv := Feedback ;
66      HmiCmd . _tagId := CONCAT ( _id , 'Cmd' ) ;
67      HmiPv . _tagId := CONCAT ( _id , 'Pv' ) ;
68      HmiPv . _type := _type ;
69      HmiCmd . _type := _type ;
70      HmiPv . _owner := _owner ;
71      HmiCmd . _owner := _owner ;
72      // Out routine - end -
73

```

## 1.3 POU: MotorDigitalFB

**1.3 POU: MotorDigitalFB**

```

1      FUNCTION_BLOCK MotorDigitalFB
2      VAR_INPUT
3          _id : STRING ;
4          _owner : STRING ;
5          Interlock : BOOL ;
6          ControlValueAuto_Cmd : BOOL ;
7          Started : BOOL ;
8          Stopped : BOOL ;
9      END_VAR
10     VAR_OUTPUT
11         ControlValue : BOOL ;
12         Alarm : BOOL ;
13     END_VAR
14     VAR
15         _StartFailed : AlarmDigitalFB ;
16         _StopFailed : AlarmDigitalFB ;
17         _type : string := 'MotorDigital' ;
18     END_VAR
19     VAR_IN_OUT
20         HmiCmd : MotorDigitalHmiCmd ;
21         HmiPv : MotorDigitalHmiPv ;
22     END_VAR
23

1      /*
2      Name: MotorDigitalFB
3      Author: Jone Vassbø
4      Version: 0.3
5      Date: 09.03.2020
6      Description:
7      - Function block to control a digital motor, with commands from HMI in
seperate struct
8      */
9
10    // In routine - start -
11    // Validate input
12    // In routine - end -
13    // Alarm routine - start
14    _StartFailed (_id := _id, _owner := _owner, Input := ( NOT Started AND
ControlValue ) OR ( Started AND NOT ControlValue ),
15        HmiCmd := HmiCmd . AlarmStartFailed , HmiPv := HmiPv . AlarmStartFailed )
;
16    _StopFailed (_id := _id, _owner := _owner, Input := ( Stopped AND
ControlValue ) OR ( NOT Stopped AND NOT ControlValue ),
17        HmiCmd := HmiCmd . AlarmStopFailed , HmiPv := HmiPv . AlarmStopFailed ) ;
18    // Alarm routine - end -
19
20    // Control routine - start -
21    IF Interlock OR Alarm THEN
22        HmiCmd . auto := FALSE ;
23        HmiCmd . ManualStart_Cmd := FALSE ;
24    END_IF

```

---

### 1.3 POU: MotorDigitalFB

---

```

25
26   IF HmiCmd . auto  THEN
27     ControlValue  :=  ControlValueAuto_Cmd ;
28   ELSE
29     ControlValue  :=  HmiCmd . ManualStart_Cmd ;
30   END_IF
31   // Control routine - end -
32
33   // Out routine - start -
34   Alarm  :=  _StartFailed . Alarm  OR NOT HmiCmd . AlarmStartFailed . Acknowledged
35   OR _StopFailed . Alarm  OR NOT HmiCmd . AlarmStopFailed . Acknowledged ;
36   HmiPv . ControlValue_Pv  :=  ControlValue ;
37   HmiPv . Interlocked_Pv  :=  Interlock ;
38   HmiPv . Started_Pv  :=  Started ;
39   HmiPv . Stopped_Pv  :=  Stopped ;
40   HmiCmd . _tagId  :=  CONCAT ( _id ,  'Cmd' ) ;
41   HmiPv . _tagId  :=  CONCAT ( _id ,  'Pv' ) ;
42   HmiPv . _type  :=  _type ;
43   HmiCmd . _type  :=  _type ;
44   HmiPv . _owner  :=  _owner ;
45   HmiCmd . _owner  :=  _owner ;
46   // Out routine - end -

```

### 1.4 POU: PIDFB

```

1   FUNCTION_BLOCK  PIDFB
2   VAR_INPUT
3     _id : STRING ;
4     _owner : STRING ;
5     InputReference : REAL ;
6     ProcessMeasure : REAL ;
7     Ts : REAL ;
8     uMax : REAL ;
9     uMin : REAL ;
10    END_VAR
11    VAR_OUTPUT
12      Output : REAL ;
13      Out_e : REAL ;
14      Out_up : REAL ;
15      Out_ui : REAL ;
16      Out_ud : REAL ;
17      Out_es : REAL ;
18      Out_u : REAL ;
19    END_VAR
20    VAR
21      _type : STRING  :=  'PID' ;
22
23      y_last : REAL  :=  0.0 ;
24      u_limited : REAL  :=  0.0 ;
25      u : REAL  :=  0.0 ;
26      e : REAL  :=  0.0 ;
27      es : REAL  :=  0.0 ;
28      manual_last : BOOL  :=  FALSE ;
29      // P

```

## 1.4 POU: PIDFB

```

30      up : REAL := 0.0 ;
31      // I
32      alfa : REAL := 0.0 ;
33      ui : REAL := 0.0 ;
34      ui_last : REAL := 0.0 ;
35      enableI_last : BOOL := TRUE ;
36      // D
37      ud : REAL := 0.0 ;
38      ud_last : REAL := 0.0 ;
39      beta : REAL := 0.0 ;
40      ref : REAL := 0 ;
41  END_VAR
42  VAR_IN_OUT
43      HmiCmd : PIDHmiCmd ;
44      HmiPv : PIDHmiPv ;
45  END_VAR
46

```

```

1      /*
2      Name: PIDFB
3      Author: Jone Vassbø
4      Version: 0.1
5      Date: 09.03.2020
6      Description:
7      - Standard PID regulation block
8      */
9
10     IF HmiCmd . HmiReference THEN
11         ref := HmiCmd . HmiReference_Cmd ;
12     ELSE
13         ref := InputReference ;
14     END_IF
15
16
17     e := ref - ProcessMeasure ;
18     // K
19     up := ( HmiCmd . K * e ) * BOOL_TO_REAL ( HmiCmd . EnableK ) ;
20
21     // I
22     alfa := Ts / HmiCmd . Ti ;
23     ui := ( ui_last + ( HmiCmd . K * alfa * e ) + ( es * ( 1 / HmiCmd .
24     TrackingGain ) ) ) * BOOL_TO_REAL ( HmiCmd . EnableI ) ;
25     ui_last := ui ;
26     // On enable I
27     IF HmiCmd . EnableI AND NOT enableI_last THEN
28         ui := HmiCmd . u0 ;
29     END_IF ;
30     // On disable I
31     IF NOT HmiCmd . EnableI AND enableI_last THEN
32         HmiCmd . u0 := ui ;
33     END_IF ;
34
35     enableI_last := HmiCmd . enableI ;
36     // D
37     beta := HmiCmd . Td / ( HmiCmd . Td + ( Ts * HmiCmd . N ) ) ;

```

## 1.4 POU: PIDFB

```

37      ud := ( ( beta * ud_last ) - ( HmiCmd . K * ( HmiCmd . Td / Ts ) * ( 1 -
38          beta ) * ( ProcessMeasure - y_last ) ) * BOOL_TO_REAL ( HmiCmd . enabledD ) ;
39      ud_last := ud ;
40      y_last := ProcessMeasure ;
41
42      u := HmiCmd . u0 + up + ui + ud ;
43
44      // On enable manual
45      IF NOT HmiCmd . Auto AND NOT manual_last THEN
46          HmiCmd . ManualValue_Cmd := u ;
47      END_IF ;
48      IF NOT HmiCmd . Auto THEN
49          u_limited := HmiCmd . ManualValue_Cmd ;
50      ELSE
51          u_limited := u ;
52      END_IF ;
53      IF u_limited > uMax THEN
54          u_limited := uMax ;
55      ELSIF u_limited < uMin THEN
56          u_limited := uMin ;
57      END_IF ;
58
59      manual_last := NOT HmiCmd . Auto ;
60      es := u_limited - u ;
61
62      Output := u_limited ;
63      Out_e := e ;
64      Out_up := up ;
65      Out_ui := ui ;
66      Out_ud := ud ;
67      Out_es := es ;
68      Out_u := u ;
69
70      HmiPv . Reference_Pv := ref ;
71      HmiPv . ProcessMeasure := ProcessMeasure ;
72      HmiPv . Error := e ;
73      HmiPv . Output := Output ;
74      HmiPv . up := up ;
75      HmiPv . ui := ui ;
76      HmiPv . ud := ud ;
77      HmiPv . es := es ;
78      HmiPv . u := u ;
79
80      HmiCmd . _tagId := CONCAT ( _id , 'Cmd' ) ;
81      HmiPv . _tagId := CONCAT ( _id , 'Pv' ) ;
82      HmiPv . _type := _type ;
83      HmiCmd . _type := _type ;
84      HmiPv . _owner := _owner ;
85      HmiCmd . _owner := _owner ;

```

---

 1.5 POU: PowerMonitoringFb
 

---

**1.5 POU: PowerMonitoringFb**

```

1   FUNCTION_BLOCK PowerMonitoringFb
2     VAR_INPUT
3       _id : STRING ;
4       _owner : STRING ;
5       Input_CurrentIn : REAL ;
6       Input_CurrentOut : REAL ;
7       Input_CurrentStored : REAL ;
8     END_VAR
9     VAR_OUTPUT
10    END_VAR
11    VAR
12      _type : STRING := 'PowerMonitoring' ;
13      Current_balance : REAL ;
14    END_VAR
15    VAR_IN_OUT
16      HmiCmd : PowerMonitoringHmiCmd ;
17      HmiPv : PowerMonitoringHmiPv ;
18    END_VAR
19

20
21  /*
22  Name: PowerMonitoringFB
23  Author: Markus Raudberget
24  Version: 0.1
25  Date: 14.03.2020
26  Description:
27  - Function block for Power monitoring of a battery package, with commands
28  from hmi i seperate struct
29  */
30

31 //Control routine - start -
32 Current_balance := (Input_CurrentIn - Input_CurrentOut) ;
33 IF Current_balance < 0 THEN
34   HmiPv . Status_bool := FALSE ;
35   HmiPv . Status_string := 'Decharging' ;
36   HmiPv . Time_pv := Input_CurrentStored / - Current_balance ;
37   HmiPv . Time_string := 'Time for empty battery' ;
38 ELSE
39   HmiPv . Status_bool := TRUE ;
40   HmiPv . Status_string := 'Charging' ;
41   HmiPv . Time_pv := (HmiCmd . BatterySize_Cmd - Input_CurrentStored) /
42 Current_balance ;
43   HmiPv . Time_string := 'Time for full battery' ;
44 END_IF
45
46 HmiPv . Input_Current_In := Input_CurrentIn ;
47 HmiPv . Input_Current_Out := Input_CurrentOut ;
48 HmiPv . Input_Battery_Level := Input_CurrentStored ;
49
50 HmiCmd . _tagId := CONCAT (_id, 'Cmd') ;
51 HmiPv . _tagId := CONCAT (_id, 'Pv') ;
52 HmiPv . _type := _type ;

```

---

 1.5 POU: PowerMonitoringFb
 

---

```

31      HmiCmd . _type  :=  _type ;
32      HmiPv . _owner  :=  _owner ;
33      HmiCmd . _owner  :=  _owner ;
34

```

## 1.6 POU: ScalingFB

```

1   FUNCTION_BLOCK  ScalingFB
2   VAR_INPUT
3       _id : STRING ;
4       _owner : STRING ;
5       Input : REAL ;
6   END_VAR
7   VAR_OUTPUT
8       Output : REAL ;
9   END_VAR
10  VAR
11      _type : string  :=  'Scaling' ;
12  END_VAR
13  VAR_IN_OUT
14      HmiCmd : ScalingHmiCmd ;
15      HmiPv : ScalingHmiPv ;
16  END_VAR
17

```

---

```

1      /*
2      Name: ScalingFB
3      Author: Jone Vassbø
4      Version: 0.3
5      Date: 09.03.2020
6      Description:
7      - Function block for a scaling analog value, with commands from HMI in
8      separate struct
9      */
10
11     // Control routine - start -
12     Output  :=  HmiCmd . OutputMin_Cmd * ( 1 - ( ( Input - HmiCmd . InputMin_Cmd
13         ) / ( HmiCmd . InputMax_Cmd - HmiCmd . InputMin_Cmd ) ) ) + HmiCmd .
14     OutputMax_Cmd * ( ( Input - HmiCmd . InputMin_Cmd ) / ( HmiCmd . InputMax_Cmd
15         - HmiCmd . InputMin_Cmd ) );
16     // Control routine - end -
17
18     // Out routine - start -
19     HmiPv . Input_Pv  :=  Input ;
20     HmiPv . Output_Pv  :=  Output ;
21     HmiCmd . _tagId  :=  CONCAT ( _id , 'Cmd' ) ;
22     HmiPv . _tagId  :=  CONCAT ( _id , 'Pv' ) ;
23     HmiPv . _type  :=  _type ;
24     HmiCmd . _type  :=  _type ;
25     HmiCmd . _owner  :=  _owner ;
26     HmiPv . _owner  :=  _owner ;
27     // Out routine - end -
28

```

## 1.7 POU: SignalAnalogFB

**1.7 POU: SignalAnalogFB**

```

1   FUNCTION_BLOCK  SignalAnalogFB
2     VAR_INPUT
3       _id : STRING ;
4       _owner : STRING ;
5       RawInput : REAL ;
6     END_VAR
7     VAR_OUTPUT
8       Output : REAL ;
9       AlarmHigh : BOOL ;
10      AlarmHighHigh : BOOL ;
11      AlarmLow : BOOL ;
12      AlarmLowLow : BOOL ;
13    END_VAR
14    VAR
15      _AlarmHigh : AlarmDigitalFB ;
16      _AlarmHighHigh : AlarmDigitalFB ;
17      _AlarmLow : AlarmDigitalFB ;
18      _AlarmLowLow : AlarmDigitalFB ;
19      _Scaling : ScalingFB ;
20
21      _High : BOOL ;
22      _HighHigh : BOOL ;
23      _Low : BOOL ;
24      _LowLow : BOOL ;
25
26      _type : string := 'SignalAnalog' ;
27    END_VAR
28    VAR_IN_OUT
29      HmiCmd : SignalAnalogHmiCmd ;
30      HmiPv : SignalAnalogHmiPv ;
31    END_VAR
32

```

```

1   /*
2    * Name: SignalAnalogFB
3    * Author: Jone Vassbø
4    * Version: 0.3
5    * Date: 09.03.2020
6    * Description:
7    * - Function block for a analog signal, with alarms and commands from HMI in
8    *   seperate struct
9    */
10   // In routine - start -
11   // Validate input
12   _Scaling (_id := _id, _owner := _owner, Input := RawInput, HmiCmd := HmiCmd . InputScaling, HmiPv := HmiPv . InputScaling) ;
13   IF HmiCmd . SimulationMode THEN
14     Output := HmiCmd . SimulationValue ;
15   ELSE
16     Output := _Scaling . Output ;
17   END_IF

```

## 1.7 POU: SignalAnalogFB

```

18      // In routine - end -
19      // Alarm routine - start
20      IF Output > HmiCmd . AlarmHighLimit THEN
21          _High := TRUE ;
22      END_IF
23      IF Output < ( HmiCmd . AlarmHighLimit - HmiCmd . AlarmHysteresis ) THEN
24          _High := FALSE ;
25      END_IF
26      _AlarmHigh ( _id := _id , _owner := _owner , Input := _High ,
27                  HmiCmd := HmiCmd . AlarmHigh , HmiPv := HmiPv . AlarmHigh ) ;
28
29      IF Output > HmiCmd . AlarmHighHighLimit THEN
30          _HighHigh := TRUE ;
31      END_IF
32      IF Output < ( HmiCmd . AlarmHighHighLimit - HmiCmd . AlarmHysteresis ) THEN
33          _HighHigh := FALSE ;
34      END_IF
35      _AlarmHighHigh ( _id := _id , _owner := _owner , Input := _HighHigh ,
36                      HmiCmd := HmiCmd . AlarmHighHigh , HmiPv := HmiPv . AlarmHighHigh ) ;
37
38      IF Output < HmiCmd . AlarmLowLimit THEN
39          _Low := TRUE ;
40      END_IF
41      IF Output > ( HmiCmd . AlarmLowLimit + HmiCmd . AlarmHysteresis ) THEN
42          _Low := FALSE ;
43      END_IF
44      _AlarmLow ( _id := _id , _owner := _owner , Input := _Low ,
45                  HmiCmd := HmiCmd . AlarmLow , HmiPv := HmiPv . AlarmLow ) ;
46
47      IF Output < HmiCmd . AlarmLowLowLimit THEN
48          _LowLow := TRUE ;
49      END_IF
50      IF Output > ( HmiCmd . AlarmLowLowLimit + HmiCmd . AlarmHysteresis ) THEN
51          _LowLow := FALSE ;
52      END_IF
53      _AlarmLowLow ( _id := _id , _owner := _owner , Input := _LowLow ,
54                      HmiCmd := HmiCmd . AlarmLowLow , HmiPv := HmiPv . AlarmLowLow ) ;
55      // Alarm routine - end -
56
57      // Control routine - start -
58
59      // Control routine - end -
60
61      // Out routine - start -
62      AlarmHigh := _AlarmHigh . Alarm OR NOT HmiCmd . AlarmHigh . Acknowledged ;
63      AlarmHighHigh := _AlarmHighHigh . Alarm OR NOT HmiCmd . AlarmHighHigh .
64      Acknowledged ;
65      AlarmLow := _AlarmLow . Alarm OR NOT HmiCmd . AlarmLow . Acknowledged ;
66      AlarmLowLow := _AlarmLowLow . Alarm OR NOT HmiCmd . AlarmLowLow .
67      Acknowledged ;
68
69      HmiPv . Output_Pv := Output ;
70      HmiPv . RawInput_Pv := RawInput ;
71      HmiCmd . _tagId := CONCAT ( _id , 'Cmd' ) ;
72      HmiPv . _tagId := CONCAT ( _id , 'Pv' ) ;
73      HmiPv . _type := _type ;

```

---

 1.7 POU: SignalAnalogFB
 

---

```

72      HmiCmd . _type   :=  _type ;
73      HmiPv . _owner  :=  _owner ;
74      HmiCmd . _owner  :=  _owner ;
75      // Out routine - end -
76
  
```

## 1.8 POU: signalDigitalFB

```

1      FUNCTION_BLOCK  signalDigitalFB
2      VAR_INPUT
3          _id : STRING ;
4          _owner : STRING ;
5          Input : BOOL ;
6      END_VAR
7      VAR_OUTPUT
8          Output : BOOL ;
9      END_VAR
10     VAR
11         _type : string  :=  'SignalDigital' ;
12     END_VAR
13     VAR_IN_OUT
14         HmiCmd : SignalDigitalHmiCmd ;
15         HmiPv : SignalDigitalHmiPv ;
16     END_VAR
17
  
```

```

1      /*
2      Name: signalDigitalFB
3      Author: Jone Vassbø
4      Version: 0.3
5      Date: 09.03.2020
6      Description:
7      - Function block for a digital signal, with commands from HMI in separate
8      struct
9      *)
10
11     // In routine - start -
12     // Validate input
13     // In routine - end -
14     // Alarm routine - start
15     // Alarm routine - end -
16
17     // Control routine - start -
18     IF HmiCmd . SimulationMode THEN
19         Output  :=  HmiCmd . SimulationValue ;
20     ELSE
21         Output  :=  Input ;
22     END_IF
23     // Control routine - end -
24
25     // Out routine - start -
26     HmiPv . Input_Pv  :=  Input ;
27     HmiPv . Output_Pv  :=  Output ;
28     HmiCmd . _tagId  :=  CONCAT ( _id ,  'Cmd' ) ;
29     HmiPv . _tagId  :=  CONCAT ( _id ,  'Pv' ) ;
  
```

---

 1.8 POU: signalDigitalFB
 

---

```

29      HmiPv . _type  :=  _type ;
30      HmiCmd . _type  :=  _type ;
31      HmiPv . _owner  :=  _owner ;
32      HmiCmd . _owner  :=  _owner ;
33      // Out routine - end -
34

```

## 1.9 POU: TimerFB

```

1      FUNCTION_BLOCK  TimerFB
2      VAR_INPUT
3          _id : STRING ;
4          _owner : STRING ;
5          Input :  BOOL ;
6      END_VAR
7      VAR_OUTPUT
8          Output :      BOOL ;
9      END_VAR
10     VAR
11        OnDelay :  TON ;
12        OffDelay :  TOF ;
13        ToSec : REAL  :=  1000.0 ;
14        _type : string  :=  'Timer' ;
15    END_VAR
16    VAR_IN_OUT
17        HmiCmd :      TimerHmiCmd ;
18        HmiPv :  TimerHmiPv ;
19    END_VAR
20

```

```

1      /*
2       * Name: TimerFB
3       * Author: Jone Vassbø
4       * Version: 0.4
5       * Date: 09.03.2020
6       * Description:
7       * - Function block operating as a Timer with both on and off delay
8      *)
9
10     // In routine - start -
11     // Validate input
12     IF HmiCmd . OnDelayLimit < 0  THEN
13         HmiCmd . OnDelayLimit  :=  0.0 ;
14     END_IF
15     IF HmiCmd . OffDelayLimit < 0  THEN
16         HmiCmd . OnDelayLimit  :=  0.0 ;
17     END_IF
18     // In routine - end -
19     // Alarm routine - start
20     // Alarm routine - end -
21     // Control routine - start -
22     OnDelay ( IN  :=  Input ,
23             PT  :=  TO_TIME ( HmiCmd . OnDelayLimit * ToSec ) ) ;
24     HmiPv . OnDelayElapsed  :=  TO_REAL ( OnDelay . ET ) / ToSec ;
25

```

## 1.9 POU: TimerFB

```

26      OffDelay ( IN := OnDelay . Q ,
27                  PT := TO_TIME ( HmiCmd . OffDelayLimit * ToSec ) ) ;
28      HmiPv . OffDelayElapsed := TO_REAL ( OffDelay . ET ) / ToSec ;
29
30      IF OnDelay . Q THEN
31          Output := TRUE ;
32      END_IF
33      IF NOT OffDelay . Q THEN
34          Output := FALSE ;
35      END_IF
36      // Control routine - end -
37
38      // Out routine - start -
39      HmiPv . Input_Pv := Input ;
40      HmiPv . Output_Pv := Output ;
41      HmiCmd . _tagId := CONCAT ( _id , 'Cmd' ) ;
42      HmiPv . _tagId := CONCAT ( _id , 'Pv' ) ;
43      HmiPv . _type := _type ;
44      HmiCmd . _type := _type ;
45      HmiPv . _owner := _owner ;
46      HmiCmd . _owner := _owner ;
47      // Out routine - end -
48

```

## 1.10 POU: WorkDistributorFB

```

1      FUNCTION_BLOCK WorkDistributorFB
2      VAR_INPUT
3          _id : STRING ;
4          _owner : STRING ;
5          InputControlSignal : REAL ;
6          Tag1 : STRING ;
7          Tag2 : STRING ;
8          Tag3 : STRING ;
9          InAuto1 : BOOL ;
10         InAuto2 : BOOL ;
11         InAuto3 : BOOL ;
12         RunHours1 : INT ;
13         RunHours2 : INT ;
14         RunHours3 : INT ;
15         ChangeInterval : REAL ;
16     END_VAR
17     VAR_OUTPUT
18         Out1 : REAL ;
19         Out2 : REAL ;
20         Out3 : REAL ;
21
22     END_VAR
23     VAR
24         _intervalTimer : TON ;
25         _alarm : BOOL ;
26         _type : STRING := 'WorkDistributor' ;
27         _ToSec : REAL := 1000.0 ;
28         _maxWorkingHours : INT ;
29         _run1 : BOOL ;

```

## 1.10 POU: WorkDistributorFB

```

30      _run2 : BOOL ;
31      _run3 : BOOL ;
32  END_VAR
33  VAR_IN_OUT
34      HmiPv : WorkDistributorHmiPv ;
35  END_VAR
36

1      /*
2      Name: WorkDistributorFB
3      Author: Jone Vassbø
4      Version: 0.4
5      Date: 09.03.2020
6      Description:
7      - Function block to distribute work between three components
8      */
9
10     _intervalTimer ( IN := NOT _intervalTimer . Q ,
11                      PT := TO_TIME ( ChangeInterval * _ToSec ) ) ;
12
13     _alarm := NOT InAuto1 OR NOT InAuto2 OR NOT InAuto3 ;
14
15     // These are located here because of the return statements
16     HmiPv . Tag1 := Tag1 ;
17     HmiPv . Tag2 := Tag2 ;
18     HmiPv . Tag3 := Tag3 ;
19     HmiPv . RunHours1 := RunHours1 ;
20     HmiPv . RunHours2 := RunHours2 ;
21     HmiPv . RunHours3 := RunHours3 ;
22     HmiPv . AlarmNotAuto := _alarm ;
23
24     HmiPv . _tagId := CONCAT ( _id , 'Pv' ) ;
25     HmiPv . _type := _type ;
26     HmiPv . _owner := _owner ;
27
28     IF _alarm THEN
29         Out1 := InputControlSignal ;
30         Out2 := InputControlSignal ;
31         Out3 := InputControlSignal ;
32         RETURN ;
33     END_IF
34
35     Out1 := ( InputControlSignal * BOOL_TO_REAL ( _run1 ) ) ;
36     Out2 := ( InputControlSignal * BOOL_TO_REAL ( _run2 ) ) ;
37     Out3 := ( InputControlSignal * BOOL_TO_REAL ( _run3 ) ) ;
38
39
40     // Time to change who is working
41     IF _intervalTimer . Q THEN
42         _run1 := TRUE ;
43         _run2 := TRUE ;
44         _run3 := TRUE ;
45
46         _maxWorkingHours := RunHours1 ;
47         IF RunHours2 > _maxWorkingHours THEN

```

## 1.10 POU: WorkDistributorFB

```

48      _maxWorkingHours := RunHours2 ;
49  END_IF
50  IF RunHours3 > _maxWorkingHours THEN
51      _maxWorkingHours := RunHours3 ;
52  END_IF
53
54  IF RunHours1 = _maxWorkingHours THEN
55      _run1 := FALSE ;
56      RETURN ; // Prevents two from stopping
57  END_IF
58  IF RunHours2 = _maxWorkingHours THEN
59      _run2 := FALSE ;
60      RETURN ;
61  END_IF
62  IF RunHours3 = _maxWorkingHours THEN
63      _run3 := FALSE ;
64      RETURN ;
65  END_IF
66 END_IF
67
68

```

## 2 Folder: LibHmiStructs

### 2.1 DUT: AlarmDigitalHmiCmd

```

1  // These are autogenerated, do not manual modify it
2  TYPE AlarmDigitalHmiCmd :
3  STRUCT
4      _tagId : STRING ;
5      _type : STRING ;
6      _owner : STRING ;
7      Tag : STRING ;
8      Description : STRING ;
9      Delay : TimerHmiCmd ;
10     Acknowledged : BOOL ;
11  END_STRUCT
12 END_TYPE
13

```

### 2.2 DUT: AlarmDigitalHmiPv

```

1  // These are autogenerated, do not manual modify it
2  TYPE AlarmDigitalHmiPv :
3  STRUCT
4      _tagId : STRING ;
5      _type : STRING ;
6      _owner : STRING ;
7      Input_Pv : BOOL ;
8      Delay : TimerHmiPv ;
9      Alarm : BOOL ;
10 END_STRUCT

```

---

## 2.2 DUT: AlarmDigitalHmiPv

---

```

11      END_TYPE
12

```

### 2.3 DUT: AliveHmiPv

```

1      // These are autogenerated, do not manual modify it
2      TYPE AliveHmiPv :
3          STRUCT
4              _tagId : STRING ;
5              _type : STRING ;
6              _owner : STRING ;
7              CounterPv : INT ;
8          END_STRUCT
9      END_TYPE
10

```

### 2.4 DUT: ControlAnalogHmiCmd

```

1      // These are autogenerated, do not manual modify it
2      TYPE ControlAnalogHmiCmd :
3          STRUCT
4              _tagId : STRING ;
5              _type : STRING ;
6              _owner : STRING ;
7              Tag : STRING ;
8              Description : STRING ;
9              Auto : BOOL ;
10             ManualControlValue_Cmd : REAL ;
11             AlarmControlFailed :     AlarmDigitalHmiCmd ;
12             AlarmControlFailedLimit :     REAL ;
13         END_STRUCT
14     END_TYPE
15

```

### 2.5 DUT: ControlAnalogHmiPv

```

1      // These are autogenerated, do not manual modify it
2      TYPE ControlAnalogHmiPv :
3          STRUCT
4              _tagId : STRING ;
5              _type : STRING ;
6              _owner : STRING ;
7              ControlValue_Pv :     REAL ;
8              Interlocked_Pv :     BOOL ;
9              AlarmControlFailed :     AlarmDigitalHmiPv ;
10             Feedback_Pv : REAL ;
11             RunHours_Pv : INT ;
12             RunMinutes_Pv : int ;
13         END_STRUCT
14     END_TYPE
15

```

---

 2.6 DUT: MotorDigitalHmiCmd
 

---

```

1   // These are autogenerated, do not manual modify it
2   TYPE MotorDigitalHmiCmd :
3     STRUCT
4       _tagId : STRING ;
5       _type : STRING ;
6       _owner : STRING ;
7       Tag : STRING ;
8       Description : STRING ;
9       Auto : BOOL ;
10      AlarmStartFailed : AlarmDigitalHmiCmd ;
11      AlarmStopFailed : AlarmDigitalHmiCmd ;
12      ManualStart_Cmd : BOOL ;
13    END_STRUCT
14  END_TYPE
15

```

**2.7 DUT: MotorDigitalHmiPv**

```

1   // These are autogenerated, do not manual modify it
2   TYPE MotorDigitalHmiPv :
3     STRUCT
4       _tagId : STRING ;
5       _type : STRING ;
6       _owner : STRING ;
7       Interlocked_Pv : BOOL ;
8       ControlValue_Pv : BOOL ;
9       AlarmStartFailed : AlarmDigitalHmiPv ;
10      AlarmStopFailed : AlarmDigitalHmiPv ;
11      Started_Pv : BOOL ;
12      Stopped_Pv : BOOL ;
13    END_STRUCT
14  END_TYPE
15

```

**2.8 DUT: PIDHmiCmd**

```

1   // These are autogenerated, do not manual modify it
2   TYPE PIDHmiCmd :
3     STRUCT
4       _tagId : STRING ;
5       _type : STRING ;
6       _owner : STRING ;
7       Tag : STRING ;
8       Description : STRING ;
9       K : REAL ;
10      EnableK : BOOL ;
11      Ti : REAL ;
12      EnableI : BOOL ;
13      Td : REAL ;
14      N : REAL ;
15      EnableD : BOOL ;
16      Auto : BOOL ;

```

---

## 2.8 DUT: PIDHmiCmd

---

```

17      ManualValue_Cmd : REAL ;
18      u0 : REAL ;
19      TrackingGain : REAL ;
20      HmiReference_Cmd : REAL ;
21      HmiReference : BOOL ;
22  END_STRUCT
23  END_TYPE
24

```

### **2.9 DUT: PIDHmiPv**

```

1 // These are autogenerated, do not manual modify it
2 TYPE PIDHmiPv :
3 STRUCT
4     _tagId : STRING ;
5     _type : STRING ;
6     _owner : STRING ;
7     Reference_Pv : REAL ;
8     ProcessMeasure : REAL ;
9     Error : REAL ;
10    Output : REAL ;
11    up : REAL ;
12    ui : REAL ;
13    ud : REAL ;
14    es : REAL ;
15    u : REAL ;
16  END_STRUCT
17  END_TYPE
18

```

### **2.10 DUT: PowerMonitoringHmiCmd**

```

1 //These are autogenerated, do not manual modify it
2 TYPE PowerMonitoringHmiCmd :
3 STRUCT
4     _tagId : STRING ;
5     _type : STRING ;
6     _owner : STRING ;
7     BatterySize_Cmd : REAL ; //Size of the battery in mAh
8  END_STRUCT
9  END_TYPE
10

```

### **2.11 DUT: PowerMonitoringHmiPv**

```

1 //These are autogenerated, do not manual modify it
2 TYPE PowerMonitoringHmiPv :
3 STRUCT
4     _tagId : STRING ;
5     _type : STRING ;
6     _owner : STRING ;
7     Status_string : STRING ;
8     Status_bool : BOOL ;
9     Time_string : STRING ;
10    Time_pv : REAL ;

```

---

## 2.11 DUT: PowerMonitoringHmiPv

---

```

11      Alarm : BOOL ;
12      Input_Current_In : REAL ;
13      Input_Current_Out : REAL ;
14      Input_Battery_Level : REAL ;
15      END_STRUCT
16      END_TYPE
17

```

## 2.12 DUT: ScalingHmiCmd

```

1      // These are autogenerated, do not manual modify it
2      TYPE ScalingHmiCmd :
3          STRUCT
4              _tagId : STRING ;
5              _type : STRING ;
6              _owner : STRING ;
7              InputUnit : STRING ;
8              InputMin_Cmd : REAL ;
9              InputMax_Cmd : REAL ;
10             OutputUnit : STRING ;
11             OutputMin_Cmd : REAL ;
12             OutputMax_Cmd : REAL ;
13         END_STRUCT
14         END_TYPE
15

```

## 2.13 DUT: ScalingHmiPv

```

1      // These are autogenerated, do not manual modify it
2      TYPE ScalingHmiPv :
3          STRUCT
4              _tagId : STRING ;
5              _type : STRING ;
6              _owner : STRING ;
7              Input_Pv : REAL ;
8              Output_Pv : REAL ;
9          END_STRUCT
10         END_TYPE
11

```

## 2.14 DUT: SignalAnalogHmiCmd

```

1      // These are autogenerated, do not manual modify it
2      TYPE SignalAnalogHmiCmd :
3          STRUCT
4              _tagId : STRING ;
5              _type : STRING ;
6              _owner : STRING ;
7              Tag : STRING ;
8              Description : STRING ;
9              AlarmHigh : AlarmDigitalHmiCmd ;
10             AlarmHighHigh : AlarmDigitalHmiCmd ;
11             AlarmLow : AlarmDigitalHmiCmd ;
12             AlarmLowLow : AlarmDigitalHmiCmd ;
13             AlarmHighLimit : REAL ;

```

---

## 2.14 DUT: SignalAnalogHmiCmd

---

```

14      AlarmHighHighLimit : REAL ;
15      AlarmLowLimit : REAL ;
16      AlarmLowLowLimit : REAL ;
17      AlarmHysteresis : REAL ;
18      SimulationMode : BOOL ;
19      SimulationValue : REAL ;
20      InputScaling : ScalingHmiCmd ;
21      END_STRUCT
22      END_TYPE
23

```

## 2.15 DUT: SignalAnalogHmiPv

```

1      // These are autogenerated, do not manual modify it
2      TYPE SignalAnalogHmiPv :
3          STRUCT
4              _tagId : STRING ;
5              _type : STRING ;
6              _owner : STRING ;
7              RawInput_Pv : REAL ;
8              Output_Pv : REAL ;
9              AlarmHigh : AlarmDigitalHmiPv ;
10             AlarmHighHigh : AlarmDigitalHmiPv ;
11             AlarmLow : AlarmDigitalHmiPv ;
12             AlarmLowLow : AlarmDigitalHmiPv ;
13             InputScaling : ScalingHmiPv ;
14             END_STRUCT
15             END_TYPE
16

```

## 2.16 DUT: SignalDigitalHmiCmd

```

1      // These are autogenerated, do not manual modify it
2      TYPE SignalDigitalHmiCmd :
3          STRUCT
4              _tagId : STRING ;
5              _type : STRING ;
6              _owner : STRING ;
7              Tag : STRING ;
8              Description : STRING ;
9              SimulationMode : BOOL ;
10             SimulationValue : BOOL ;
11             END_STRUCT
12             END_TYPE
13

```

---

 2.17 DUT: SignalDigitalHmiPv
 

---

```

1  // These are autogenerated, do not manual modify it
2  TYPE SignalDigitalHmiPv :
3  STRUCT
4    _tagId : STRING ;
5    _type : STRING ;
6    _owner : STRING ;
7    Input_Pv : BOOL ;
8    Output_Pv : BOOL ;
9    END_STRUCT
10   END_TYPE
11

```

## 2.18 DUT: TimerHmiCmd

```

1  // These are autogenerated, do not manual modify it
2  TYPE TimerHmiCmd :
3  STRUCT
4    _tagId : STRING ;
5    _type : STRING ;
6    _owner : STRING ;
7    Tag : STRING ;
8    Description : STRING ;
9    OnDelayLimit : REAL ;
10   OffDelayLimit : REAL ;
11   END_STRUCT
12   END_TYPE
13

```

## 2.19 DUT: TimerHmiPv

```

1  // These are autogenerated, do not manual modify it
2  TYPE TimerHmiPv :
3  STRUCT
4    _tagId : STRING ;
5    _type : STRING ;
6    _owner : STRING ;
7    Input_Pv : BOOL ;
8    Output_Pv : BOOL ;
9    OnDelayElapsed : REAL ;
10   OffDelayElapsed : REAL ;
11   END_STRUCT
12   END_TYPE
13

```

---

 2.20 DUT: WorkDistributorHmiPv
 

---

**2.20 DUT: WorkDistributorHmiPv**

```

1   // These are autogenerated, do not manual modify it
2   TYPE WorkDistributorHmiPv :
3     STRUCT
4       _tagId : STRING ;
5       _type : STRING ;
6       _owner : STRING ;
7       Tag1 : STRING ;
8       Tag2 : STRING ;
9       Tag3 : STRING ;
10      RunHours1 : INT ;
11      RunHours2 : INT ;
12      RunHours3 : INT ;
13      AlarmNotAuto : BOOL ;
14    END_STRUCT
15  END_TYPE
16

```

### 3 Folder: Utils

**3.1 DUT: OpcUa\_Mqtt\_Status**

```

1   TYPE OpcUa_Mqtt_Status :
2     STRUCT
3       LastRun : STRING ;
4       AliveCounterPv : INT := 0 ;
5       Restart_Cmd : BOOL := FALSE ;
6     END_STRUCT
7   END_TYPE
8

```

**3.2 POU: eulers**

```

1   FUNCTION eulers : REAL
2     VAR_INPUT
3       y : REAL ;
4       f : REAL ;
5       h : REAL ;
6     END_VAR
7     VAR
8     END_VAR
9

```

---

```

1   eulers := y + ( h * f ) ;
2

```

---

 3.3 POU: firstOrderSystem
 

---

**3.3 POU: firstOrderSystem**

```

1   FUNCTION firstOrderSystem : REAL
2   VAR_INPUT
3       T : REAL ;
4       Kp : REAL ;
5       y : REAL ;
6       u : REAL ;
7   END_VAR
8   VAR
9   END_VAR
10
11
12   firstOrderSystem := - ( y / T ) + ( ( Kp / T ) * u ) ;
13

```

---

**3.4 POU: process**

```

1   FUNCTION process : REAL
2   VAR_INPUT
3       u : REAL ;
4       Ts : REAL ;
5       y : REAL ;
6       yMin : REAL ;
7       yMax : REAL ;
8       T : REAL ;
9       Kp : REAL ;
10  END_VAR
11  VAR
12      _out : real ;
13  END_VAR
14
15
16  _out := eulers ( y , firstOrderSystem ( T , Kp , y , u ) , Ts ) ;
17  IF _out > yMax THEN
18      _out := yMax ;
19  ELSIF _out < yMin THEN
20      _out := yMin ;
21  END_IF ;
22  process := _out ;
23

```

---

### 3.5 POU: String\_To\_ForecastArray

### **3.5 POU: String\_To\_ForecastArray**

```

1   FUNCTION String_To_ForecastArray : ARRAY [ 0 .. 4 ] OF REAL
2   VAR_INPUT
3       InputString : STRING ;
4   END_VAR
5   VAR
6       temp : STRING ;
7       temp1 : STRING ;
8       temp2 : STRING ;
9       pos : INT ;
10      i : INT ;
11      values : ARRAY [ 0 .. 4 ] OF REAL ;
12  END_VAR
13

1   i := 0 ;
2
3   pos := LEN ( InputString ) ;
4   temp := DELETE ( InputString , 1 , 1 ) ;
5   temp1 := temp ;
6   temp := DELETE ( temp , 1 , pos - 1 ) ;
7   temp2 := temp ;
8   pos := FIND ( temp , ',' ) ;
9   FOR i := 0 TO 3 BY 1 DO
10       values [ i ] := STRING_TO_REAL ( LEFT ( temp , pos - 1 ) ) ; // Denne tar
ikke hensyn til siste verdi, då den leter etter komma men det finnes ikke
11       temp := DELETE ( temp , pos + 2 , 0 ) ;
12       pos := FIND ( temp , ',' ) ;
13   END_FOR ;
14   values [ 4 ] := STRING_TO_REAL ( temp ) ;
15   String_To_ForecastArray := values ;
16

```

## **A.5 PLS program - Flomsikring**

## Project Documentation

File: PLS Flomsikring 09.05.2020.ecp

Date: 5/9/2020

Profile: e!COCKPIT

---

Table of Contents

## Table of Contents

1 Device: PFCx00_SmartCoupler	3
1.1 PLC Logic: Plc Logic	3
1.1.1 Application: PLS3_Flomsikring	4

---

1 Device: PFCx00\_SmartCoupler

---

## 1 Device: PFCx00\_SmartCoupler

### Users and Groups

Users:

Groups:

---

### Access Rights

View  
Modify  
Execute  
Add/remove children

---

### Symbol Rights

---

### Parameters

#### Parameters:

Name:	Type:	Value:	Default Value:	Unit:	Description:
Processor Load Lower Limit	DWORD	80	80		
Processor Load Upper Limit	DWORD	90	90		
Processor Load Processor Share	DWORD	90	90		
Processor Load Should Throw	bool	FALSE	FALSE		
ProcessorLoadWatchdog_Exception					

---

### Information

Name: 750-8214 PFC200 G2 2ETH RS CAN  
Vendor: WAGO  
Categories: PLCs  
Type: 4096  
ID: 1006 120a  
Version: 5.13.2.22  
Order number: 0750-8214  
Description: Programmable Ethernet fieldbus coupler

---

---

 1.1 PLC Logic: Plc Logic
 

---

## 1.1 PLC Logic: Plc Logic

### 1.1.1 Application: PLS3\_Flomsikring

#### 1.1.1.1 Folder: GlobalData

##### 1.1.1.1.1 Global Variable List: HMI

```

1  {attribute 'qualified_only'}
2  VAR_GLOBAL PERSISTENT RETAIN
3      AliveCounterPlc3Pv : AliveHmiPv ;
4      ValveEmission1Cmd : ControlAnalogHmiCmd ;
5      ValveEmission1Pv : ControlAnalogHmiPv ;
6      ValveEmission1AlarmMotorProtectionCmd : AlarmDigitalHmiCmd ;
7      ValveEmission1AlarmMotorProtectionPv : AlarmDigitalHmiPv ;
8      ValveEmission1SignalDigitalMotorProtectionCmd : SignalDigitalHmiCmd ;
9      ValveEmission1SignalDigitalMotorProtectionPv : SignalDigitalHmiPv ;
10     ValveEmission1ScalingPositionCmd : ScalingHmiCmd ;
11     ValveEmission1ScalingPositionPv : ScalingHmiPv ;
12     ValveEmission1ScalingControlValueCmd : ScalingHmiCmd ;
13     ValveEmission1ScalingControlValuePv : ScalingHmiPv ;
14     ValveEmission1PIDCmd : PIDHmiCmd ;
15     ValveEmission1PIDPv : PIDHmiPv ;
16
17     WaterLevelCmd : SignalAnalogHmiCmd ;
18     WaterLevelPv : SignalAnalogHmiPv ;
19
20     FlowLevelEmissionCmd : SignalAnalogHmiCmd ;
21     FlowLevelEmissionPv : SignalAnalogHmiPv ;
22
23 END_VAR
24

```

##### 1.1.1.1.2 Global Variable List: IO

```

1  {attribute 'qualified_only'}
2  VAR_GLOBAL
3      DI_ValveEmission1MotorProtectionOk : BOOL ;
4
5      AI_ValveEmission1PositionPv : REAL ;
6      AI_WaterLevelPv : REAL ;
7      AI_FlowLevelEmmissionPv : REAL ;
8
9      AO_ValveEmission1ControlValue : REAL ;
10 END_VAR
11

```

---

 1.1.1.1.3 Global Variable List: Local
 

---

```

1   {attribute 'qualified_only'}
2   VAR_GLOBAL
3     ValveEmission1 : ControlAnalogFB ;
4     ValveEmission1PID : PIDFB ;
5     ValveEmission1AlarmMotorProtection : AlarmDigitalFB ;
6     ValveEmission1MotorProtectionOk : signalDigitalFB ;
7     ValveEmission1ScalingPosition : ScalingFB ;
8     ValveEmission1ScalingControlValue : ScalingFB ;
9
10    WaterLevel : SignalAnalogFB ;
11    FlowLevelEmission : SignalAnalogFB ;
12
13    RainForcast : ARRAY [ 0 .. 4 ] OF REAL ;
14 END_VAR
15

```

#### 1.1.1.1.4 Global Variable List: SIM

```

1   {attribute 'qualified_only'}
2   VAR_GLOBAL
3     WaterLevel_Pv_m : REAL ;
4     EmissionFlow_Pv_m3_s : REAL ;
5     ValveEmission1Position_Pv : REAL ;
6     RainForcast_m_day : ARRAY [ 0 .. 4 ] OF REAL ;
7 END_VAR
8

```

#### 1.1.1.1.5 Global Variable List: STATUS

```

1   {attribute 'qualified_only'}
2   VAR_GLOBAL
3     OpcUaMqttLinkStatus : OpcUa_Mqtt_Status ;
4 END_VAR
5

```

#### 1.1.1.2 POU: Alarms

```

1   PROGRAM Alarms
2   VAR
3   END_VAR
4

```

---

```

1
2   Local . ValveEmission1AlarmMotorProtection ( _id := 
3     'ValveEmission1AlarmMotorProtection' , _owner := 'plc3' ,
4     HmiCmd := HMI .
5     ValveEmission1AlarmMotorProtectionCmd ,
6     HmiPv := HMI .
7     ValveEmission1AlarmMotorProtectionPv ,
8     Input := NOT Local .
9     ValveEmission1MotorProtectionOk . Output ) ;
6

```

---

### 1.1.1.2 POU: Alarms

---

7

### 1.1.1.3 POU: Control

```

1  PROGRAM Control
2  VAR
3      _reference_default_m : REAL := 22.0 ;
4      _reference_m : REAL ;
5  END_VAR
6

11 // Adjusting setpoint to prevent to much water in the river
12 _reference_m := _reference_default_m - Local . RainForcast [ 2 ] ;
13
14 // Regulator block
15 Local . ValveEmission1PID ( _id := 'ValveEmission1PID' , _owner := 'plc3' ,
16     InputReference := _reference_m ,
17     ProcessMeasure := Local . WaterLevel . Output ,
18     Ts := 0.5 ,
19     uMax := 100.0 ,
20     uMin := 0.0 ,
21     HmiCmd := Hmi . ValveEmission1PIDCmd ,
22     HmiPv := Hmi . ValveEmission1PIDPv ) ;
23
24 // Valves
25 Local . ValveEmission1 ( _id := 'ValveEmission1' , _owner := 'plc3' ,
26     HmiCmd := HMI . ValveEmission1Cmd ,
27     HmiPv := HMI . ValveEmission1Pv ,
28     Interlock := NOT Local . ValveEmission1MotorProtectionOk . Output ,
29     Feedback := Local . ValveEmission1ScalingPosition . Output ,
30     ControlValueAuto_Cmd := Local . ValveEmission1PID . Output ) ; //ControlValueAuto_Cmd from PID regulator
31
32

```

### 1.1.1.4 POU: Initialize

```

1  PROGRAM Initialize
2  VAR
3  END_VAR
4

11 // First run setpoints - are stored while power failure, but not for program
12 updates
13 Hmi . ValveEmission1SignalDigitalMotorProtectionCmd . Tag := 'VE01' ;
14 Hmi . ValveEmission1SignalDigitalMotorProtectionCmd . Description := 'Digital
15 signal for Emission Valve 1 motor protection' ;
16
17 Hmi . ValveEmission1ScalingPositionCmd . InputMax_Cmd := 100.0 ;
18 Hmi . ValveEmission1ScalingPositionCmd . InputMin_Cmd := 0.0 ;
19 Hmi . ValveEmission1ScalingPositionCmd . OutputMax_Cmd := 100.0 ;
20
21

```

---

#### 1.1.1.4 POU: Initialize

---

```

8   Hmi . ValveEmission1ScalingPositionCmd . OutputMin_Cmd := 0.0 ;
9
10  Hmi . ValveEmission1ScalingControlValueCmd . InputMax_Cmd := 100.0 ;
11  Hmi . ValveEmission1ScalingControlValueCmd . InputMin_Cmd := 0.0 ;
12  Hmi . ValveEmission1ScalingControlValueCmd . OutputMax_Cmd := 100.0 ;
13  Hmi . ValveEmission1ScalingControlValueCmd . OutputMin_Cmd := 0.0 ;
14
15  Hmi . ValveEmission1AlarmMotorProtectionCmd . Tag := 'VE01' ;
16  Hmi . ValveEmission1AlarmMotorProtectionCmd . Description := 'Motor
protection alarm Emission Valve 1' ;
17  Hmi . ValveEmission1AlarmMotorProtectionCmd . Delay . OnDelayLimit := 1.0 ;
18  Hmi . ValveEmission1AlarmMotorProtectionCmd . Delay . OffDelayLimit := 5.0 ;
19
20  HMI . ValveEmission1Cmd . Tag := 'VE01' ;
21  HMI . ValveEmission1Cmd . Description := 'Emission valve 1' ;
22  HMI . ValveEmission1Cmd . Auto := TRUE ;
23  HMI . ValveEmission1Cmd . ManualControlValue_Cmd := 50.0 ;
24  HMI . ValveEmission1Cmd . AlarmControlFailedLimit := 10.0 ;
25  HMI . ValveEmission1Cmd . AlarmControlFailed . Description := 'Failed to
control Emission valve 1' ;
26  HMI . ValveEmission1Cmd . AlarmControlFailed . Delay . OnDelayLimit := 100.0 ;
27  HMI . ValveEmission1Cmd . AlarmControlFailed . Delay . OffDelayLimit := 10.0 ;
28  HMI . ValveEmission1Cmd . AlarmControlFailed . Acknowledged := TRUE ;
29
30  Hmi . ValveEmission1PIDCmd . Tag := 'PID02' ;
31  Hmi . ValveEmission1PIDCmd . Description := 'Regulator for controling
emission valve 1' ;
32  Hmi . ValveEmission1PIDCmd . HmiReference := FALSE ;
33  Hmi . ValveEmission1PIDCmd . HmiReference_Cmd := 20.0 ;
34  Hmi . ValveEmission1PIDCmd . Auto := TRUE ;
35  Hmi . ValveEmission1PIDCmd . EnableK := TRUE ;
36  Hmi . ValveEmission1PIDCmd . EnableI := TRUE ;
37  Hmi . ValveEmission1PIDCmd . EnableD := FALSE ;
38  Hmi . ValveEmission1PIDCmd . K := -0.5 ;
39  Hmi . ValveEmission1PIDCmd . Ti := 20.0 ;
40  Hmi . ValveEmission1PIDCmd . Td := 0.0 ;
41  Hmi . ValveEmission1PIDCmd . N := 10.0 ;
42  Hmi . ValveEmission1PIDCmd . TrackingGain := 1.0 ;
43  Hmi . ValveEmission1PIDCmd . u0 := 50.0 ;
44
45  Hmi . WaterLevelCmd . Tag := 'PT01' ;
46  Hmi . WaterLevelCmd . Description := 'Level of supply water' ;
47  Hmi . WaterLevelCmd . AlarmHysteresis := 1.0 ;
48  Hmi . WaterLevelCmd . AlarmHighHighLimit := 25.0 ;
49  Hmi . WaterLevelCmd . AlarmHighLimit := 24.5 ;
50  Hmi . WaterLevelCmd . AlarmLowLowLimit := 10.0 ;
51  Hmi . WaterLevelCmd . AlarmLowLimit := 5.0 ;
52  Hmi . WaterLevelCmd . AlarmHigh . Description := 'High level supply water' ;
53  Hmi . WaterLevelCmd . AlarmHigh . Delay . OnDelayLimit := 1.0 ;
54  Hmi . WaterLevelCmd . AlarmHigh . Delay . OffDelayLimit := 1.0 ;
55  Hmi . WaterLevelCmd . AlarmHighHigh . Description := 'High high level supply
water' ;
56  Hmi . WaterLevelCmd . AlarmHighHigh . Delay . OnDelayLimit := 1.0 ;
57  Hmi . WaterLevelCmd . AlarmHighHigh . Delay . OffDelayLimit := 1.0 ;
58  Hmi . WaterLevelCmd . AlarmLow . Description := 'Low level supply water' ;
59  Hmi . WaterLevelCmd . AlarmLow . Delay . OnDelayLimit := 1.0 ;

```

---

#### 1.1.1.4 POU: Initialize

---

```

60      Hmi . WaterLevelCmd . AlarmLow . Delay . OffDelayLimit  := 1.0 ;
61      Hmi . WaterLevelCmd . AlarmLowLow . Description  := 'Low low level supply
water' ;
62      Hmi . WaterLevelCmd . AlarmLowLow . Delay . OnDelayLimit  := 1.0 ;
63      Hmi . WaterLevelCmd . AlarmLowLow . Delay . OffDelayLimit  := 1.0 ;
64      Hmi . WaterLevelCmd . InputScaling . InputUnit  := 'na' ;
65      Hmi . WaterLevelCmd . InputScaling . OutputUnit  := 'm' ;
66      Hmi . WaterLevelCmd . InputScaling . InputMax_Cmd  := 100.0 ;
67      Hmi . WaterLevelCmd . InputScaling . InputMin_Cmd  := 0.0 ;
68      Hmi . WaterLevelCmd . InputScaling . OutputMax_Cmd  := 100.0 ;
69      Hmi . WaterLevelCmd . InputScaling . OutputMin_Cmd  := 0.0 ;
70
71      Hmi . FlowLevelEmissionCmd . Tag  := 'FT08' ;
72      Hmi . FlowLevelEmissionCmd . Description  := 'Level of flow out of water' ;
73      Hmi . FlowLevelEmissionCmd . AlarmHysteresis  := 3.0 ;
74      Hmi . FlowLevelEmissionCmd . AlarmHighHighLimit  := 90.0 ;
75      Hmi . FlowLevelEmissionCmd . AlarmHighLimit  := 80.0 ;
76      Hmi . FlowLevelEmissionCmd . AlarmLowLowLimit  := 10.0 ;
77      Hmi . FlowLevelEmissionCmd . AlarmLowLimit  := 20.0 ;
78      Hmi . FlowLevelEmissionCmd . AlarmHigh . Delay . OnDelayLimit  := 1.0 ;
79      Hmi . FlowLevelEmissionCmd . AlarmHigh . Delay . OffDelayLimit  := 1.0 ;
80      Hmi . FlowLevelEmissionCmd . AlarmHighHigh . Delay . OnDelayLimit  := 1.0 ;
81      Hmi . FlowLevelEmissionCmd . AlarmHighHigh . Delay . OffDelayLimit  := 1.0 ;
82      Hmi . FlowLevelEmissionCmd . AlarmLow . Delay . OnDelayLimit  := 1.0 ;
83      Hmi . FlowLevelEmissionCmd . AlarmLow . Delay . OffDelayLimit  := 1.0 ;
84      Hmi . FlowLevelEmissionCmd . AlarmLowLow . Delay . OnDelayLimit  := 1.0 ;
85      Hmi . FlowLevelEmissionCmd . AlarmLowLow . Delay . OffDelayLimit  := 1.0 ;
86      Hmi . FlowLevelEmissionCmd . InputScaling . InputUnit  := 'na' ;
87      Hmi . FlowLevelEmissionCmd . InputScaling . OutputUnit  := 'l/s' ;
88      Hmi . FlowLevelEmissionCmd . InputScaling . InputMax_Cmd  := 100.0 ;
89      Hmi . FlowLevelEmissionCmd . InputScaling . InputMin_Cmd  := 0.0 ;
90      Hmi . FlowLevelEmissionCmd . InputScaling . OutputMax_Cmd  := 100.0 ;
91      Hmi . FlowLevelEmissionCmd . InputScaling . OutputMin_Cmd  := 0.0 ;
92

```

#### 1.1.1.5 POU: Inputs

```

1  PROGRAM  Inputs
2  VAR
3
4  END_VAR
5

1
2  Local . ValveEmission1MotorProtectionOk ( _id  :=
3    'ValveEmission1SignalDigitalMotorProtection' , _owner  := 'plc3' ,
4    Input  := IO .
DI_ValveEmission1MotorProtectionOk ,
5    HmiCmd  := HMI .
ValveEmission1SignalDigitalMotorProtectionCmd ,
6    HmiPv  := HMI .
ValveEmission1SignalDigitalMotorProtectionPv ) ;
7 // We dont need the scaling, but a real system would

```

---

### 1.1.1.5 POU: Inputs

---

```

8   Local . ValveEmission1ScalingPosition (_id := 
9     'ValveEmission1ScalingPosition' , _owner := 'plc3' ,
10    Input := IO . AI_ValveEmission1PositionPv ,
11    HmiCmd := HMI . ValveEmission1ScalingPositionCmd ,
12    HmiPv := HMI . ValveEmission1ScalingPositionPv ) ;
13
14  Local . WaterLevel (_id := 'WaterLevel' , _owner := 'plc3' ,
15    RawInput := IO . AI_WaterLevelPv ,
16    HmiCmd := HMI . WaterLevelCmd ,
17    HmiPv := HMI . WaterLevelPv ) ;
18
19  Local . FlowLevelEmission (_id := 'FlowLevelEmission' , _owner := 'plc3' ,
20    RawInput := IO . AI_FlowLevelEmmissionPv ,
21    HmiCmd := HMI . FlowLevelEmissionCmd ,
22    HmiPv := HMI . FlowLevelEmissionPv ) ;
23
24

```

### 1.1.1.6 POU: Outputs

```

1  PROGRAM Outputs
2  VAR
3  END_VAR
4

```

---

```

1  // We dont need the scaling, but a real system would
2  Local . ValveEmission1ScalingControlValue (_id :=
3    'ValveEmission1ScalingControlValue' , _owner := 'plc3' ,
4      Input := Local . ValveEmission1 . ControlValue ,
5      HmiCmd := HMI . ValveEmission1ScalingControlValueCmd
6
6      ,
7      HmiPv := HMI . ValveEmission1ScalingControlValuePv ,
8      Output => IO . AO_ValveEmission1ControlValue ) ;

```

### 1.1.1.7 POU: PLC\_PRG

```

1  PROGRAM PLC_PRG
2  VAR RETAIN
3    _firstScanAfterDownload : BOOL  :=  TRUE ;
4  END_VAR
5  VAR
6    _counter : INT ;
7    _lastRun : STRING ;
8  END_VAR
9

```

---

```

1
2  // Signaling that the PLC is alive
3  Hmi . AliveCounterPlc3Pv . _tagId := 'AliveCounterPlc3Pv' ;
4  Hmi . AliveCounterPlc3Pv . _type := 'AliveHmi' ;

```

---

### 1.1.1.7 POU: PLC\_PRG

---

```

5      Hmi . AliveCounterPlc3Pv . _owner := 'plc3' ;
6      HMI . AliveCounterPlc3Pv . CounterPv := HMI . AliveCounterPlc3Pv . CounterPv +
7          1 ;
8
9      _counter := STATUS . OpcUaMqttLinkStatus . AliveCounterPv ;
10     _lastRun := STATUS . OpcUaMqttLinkStatus . LastRun ;
11
12    IF _firstScanAfterDownload THEN
13        Initialize () ;
14    END_IF
15    Inputs () ;
16    Alarms () ;
17    Control () ;
18    Outputs () ;
19    Simulation () ;
20
21
22
23
24
25 // Always last
26 _firstScanAfterDownload := FALSE ;
27
28
29

```

### 1.1.1.8 POU: RestartOpcUaMqttLinkOnDownload

```

1   FUNCTION RestartOpcUaMqttLinkOnDownload : DWORD
2   VAR_IN_OUT
3       EventPrm : CmpApp . EVTPARAM_CmpApp ;
4   END_VAR
5   VAR
6   END_VAR
7

1   /*
2   This function is called when a download is about to happen.
3   This will force the Opc Ua - MQTT link to restart.
4   The restart will take 10 seconds, allowing e!Cockpit to download and start
5   before the link starts.
6   */
7   Status . OpcUaMqttLinkStatus . Restart_Cmd := TRUE ;

```

---

---

 1.1.1.9 POU: RestartOpcUaMqttLinkOnOnlineChange
 

---

## 1.1.1.9 POU: RestartOpcUaMqttLinkOnOnlineChange

```

1   FUNCTION RestartOpcUaMqttLinkOnOnlineChange : DWORD
2   VAR_IN_OUT
3     EventPrm : CmpApp . EVTPARAM_CmpApp ;
4   END_VAR
5   VAR
6   END_VAR
7

1   /*
2   This function is called when a online change is about to happen.
3   This will force the Opc Ua - MQTT link to restart.
4   The restart will take 10 seconds, allowing e!Cockpit to download and start
5   before the link starts.
6   */
7   Status . OpcUaMqttLinkStatus . Restart_Cmd := TRUE ;

```

---

## 1.1.1.10 POU: Simulation

```

1   PROGRAM Simulation
2   VAR
3   END_VAR
4

1   // Simulation because we need to be able to change the forecast to fit our
2   // need
3   Local . RainForcast := SIM . RainForcast_m_day ;
4
5   // Simulation because no physical IO is connected
6
7   IO . DI_ValveEmission1MotorProtectionOk := True ;
8   // Simulating valve follows controlvalue
9   //IO.AI_ValveEmission1PositionPv := IO.AO_ValveEmission1ControlValue; // //
10  // TODO add first order delay function
11
12  IO . AI_ValveEmission1PositionPv := process ( u := IO .
13    AO_ValveEmission1ControlValue ,
14    Ts := 0.5 , y := IO . AI_ValveEmission1PositionPv , yMin := 0.001 ,
15    yMax := 100.0 , T := 2.0 , Kp := 1.0 ) ;
16
17  SIM . ValveEmission1Position_Pv := IO . AI_ValveEmission1PositionPv ;
18  IO . AI_WaterLevelPv := SIM . WaterLevel_Pv_m ;
19  IO . AI_FlowLevelEmmissionPv := SIM . EmissionFlow_Pv_m3_s ;

```

---

 1.1.1.11 POU: Simulation\_MQTT
 

---

## 1.1.1.11 POU: Simulation\_MQTT

```

1   PROGRAM Simulation_MQTT
2   VAR
3       MqttStatus           : WagoAppCloud . FbStatus_NativeMQTT ;
4       MqttError            : BOOL    := FALSE ;
5       MqttConnectedToMyCloud : BOOL    := FALSE ;
6       MqttFillLevel         : REAL   := 0 ;
7       MqttOutgoingBlocks    : ULINT  := 0 ;
8
9
10
11      MqttSubscribe          : WagoAppCloud . FbSubscribeMQTT ;
12      MqttSubscriptionDataBytes : ARRAY [ 0 .. 254 ] OF BYTE ;
13      MqttSubscriptionDataJSONString : STRING ( JSON_MAX_STRING ) ;
//      Resulting JSON string
14      MqttSubscriptionDataJSONPointer : WagoAppJSON . typJSON_Pointer ;
15      MqttSubscriptionJSONParser    : Fb_JSON_ParseAndModify ; // 
//      JSON parser function block
16      MqttSubscribeNewData : BOOL ;
17
18      oParserStatus     : WagoAppJSON . WagoSysErrorBase . FbResult ;
19      xParserError      : BOOL ;
20      xParserDone        : BOOL ;
21      diParserToken      : DINT ;
22      xParserBuildup    : BOOL ;
23      xParserTrigger     : ARRAY [ 0 .. 5 ] OF BOOL ;
24
25      MqttPublish          : WagoAppCloud . FbPublishMQTT ;
26      MqttPublishParameters : ARRAY [ 0 .. 10 ] OF STRING ;
27      MqttPublishTemplate   : STRING ( 512 )  := '{"#Parameter": "#Parameter"}'
;
28      JSONWriter          : FB_JSON_Writer_01 ;
29      MqttPublishPayload    : STRING ( 1024 ) ;
30      MqttPublishBuffer     : ARRAY [ 0 .. 1999 ] OF BYTE ;
31      MqttPublishTrigger    : BOOL ;
32      JSONWriter_Done      : BOOL ;
33      JSONWriter_Error     : BOOL ;
34      JSONWriter_Status     : WagoSysErrorBase . FbResult ;
35      JSONWriterTrigger    : BOOL ;
36  END_VAR
37

1   // Status
2   MqttStatus ( xEnabled := TRUE ,
3               xError  => MqttError ,
4               xCloudConnected => MqttConnectedToMyCloud ,           // Will be
FALSE after network timeout occurred
5               rCacheFillLevel  => MqttFillLevel ,                  //
Increasing value indicates bad network or overload
6               uliOutgoingDataBlocks => MqttOutgoingBlocks ) ;    // Only for
QoS 1 and 2. Increasing value indicates bad network or overload
7
8   // Subscribe to data

```

---

## 1.1.1.11 POU: Simulation\_MQTT

```

9      MqttSubscribe ( xSubscribe := TRUE ,
10         sTopic := 'ba/wago/sim/plc3/plcsub' ,
11         eQoS := eQualityOfService . QoS1 ,
12         aPayloadData := MqttSubscriptionDataBytes ,
13         xDataReceived => MqttSubscribeNewData ) ;
14
15     MemCopySecure ( pDest := ADR ( MqttSubscriptionDataJSONString ) , udiDestSize
16                     := 255 , pSource := ADR ( MqttSubscriptionDataBytes ) , udiSourceSize :=
17                     MqttSubscribe . dwRxNBytes , bPadding := 0 ) ;
18
19     // Execute this function block once to complete the parser buildup
20     process
21     // This must be done before calling the parser methods
22
23     IF MqttSubscribeNewData THEN
24       xParserBuildup := TRUE ;
25     END_IF
26
27     MqttSubscriptionJSONParser (
28       pData := ADR ( MqttSubscriptionDataJSONString ) , // JSON string to parse
29       udiSizeData := length ( MqttSubscriptionDataJSONString ) ,
30       oStatus => oParserStatus ,
31       xError => xParserError ,
32       xDone => xParserDone ,
33       diToken => diParserToken ,
34       xTrigger := xParserBuildup ) ; // Set xParserBuildup TRUE to
35     execute the function block
36
37     IF xParserDone THEN
38       xParserTrigger [ 0 ] := TRUE ; // Set xParserTrigger TRUE
39     to execute the parser method
40       xParserTrigger [ 1 ] := TRUE ;
41       xParserTrigger [ 2 ] := TRUE ;
42       xParserTrigger [ 3 ] := TRUE ;
43       xParserDone := FALSE ;
44     END_IF
45
46     // Call the parser method GetValueByPath for each value to be
47     // extracted from the JSON string
48     MqttSubscriptionDataJSONPointer := MqttSubscriptionJSONParser .
49     GetValueByPath ( sPointer := '/WaterLevel_Pv' , xTrigger := xParserTrigger [ 0 ] ) ;
50
51     SIM . WaterLevel_Pv_m := STRING_TO_REAL ( MqttSubscriptionDataJSONPointer .
52     sValue ) ;
53     MqttSubscriptionDataJSONPointer := MqttSubscriptionJSONParser .
54     GetValueByPath ( sPointer := '/EmissionFlow_Pv' , xTrigger := xParserTrigger [ 1 ] )
55   ) ;
56     SIM . EmissionFlow_Pv_m3_s := STRING_TO_REAL (
57     MqttSubscriptionDataJSONPointer . sValue ) ;
58     MqttSubscriptionDataJSONPointer := MqttSubscriptionJSONParser .
59     GetValueByPath ( sPointer := '/RainForcast' , xTrigger := xParserTrigger [ 2 ] ) ;
60     SIM . RainForcast_m_day := String_To_ForecastArray (
61     MqttSubscriptionDataJSONPointer . sValue ) ;
62
63     // Publish data

```

---

### 1.1.1.11 POU: Simulation\_MQTT

---

```

50      MqttPublishParameters [ 0 ] := 'ValveEmission1Position_Pv' ;
51      MqttPublishParameters [ 1 ] := REAL_TO_STRING ( SIM . ValveEmission1Position_Pv
52      ) ;
53      JSONWriterTrigger := TRUE ; // Parses every time this routine executes
54      JSONWriter (
55          sJSON_BaseFrame := MqttPublishTemplate ,
56          oStatus => JSONWriter_Status ,
57          xError => JSONWriter_Error ,
58          xDone => JSONWriter_Done ,
59          aParameterValues := MqttPublishParameters ,
60          xTrigger := JSONWriterTrigger ,
61          sOutput := MqttPublishPayload ) ;
62
63      MemCopy ( pDest := ADR ( MqttPublishBuffer ) , pSource := ADR (
64          MqttPublishPayload ) , udiSize := Length ( MqttPublishPayload ) ) ;
65      MqttPublishTrigger := TRUE ; // Parses every time this routine executes
66      MqttPublish ( sTopic := 'ba/wago/sim/plc3/plcpub' ,
67          eQualityOfService := 1 ,
68          dwSize := Length ( MqttPublishPayload ) ,
69          aData := MqttPublishBuffer ,
70          xTrigger := MqttPublishTrigger ) ;
71

```

### 1.1.1.12 Symbol Configuration: Symbols

### 1.1.1.13 Task Configuration: Task configuration

Max. number of tasks: 15

Max. number of cyclic tasks: 15

Max. number of freewheeling tasks: 15

Max. number of event tasks: 15

Max. number of external event tasks: 8

Max. number of status tasks: 15

System Events:

1. PrepareOnlineChange (Called before application online change. Context=Communication task.

Debugging=Disabled), active: RestartOpcUaMqttLinkOnOnlineChange

2. PrepareDownload (Called before application download. Context=Communication task. Debugging=Disabled), active: RestartOpcUaMqttLinkOnDownload

---

 1.1.1.13.1 Task: MQTT
 

---

### 1.1.1.13.1 Task: MQTT

Priority: 15  
 Type: Cyclic  
 Interval: 5000 Unit: ms  
 Watchdog: Active  
 Watchdog Time: 4500 Unit: ms  
 Watchdog Sensitivity: 1  
 POUs: Simulation\_MQTT

#### 1.1.1.13.1.1 Program call: Simulation\_MQTT

### 1.1.1.13.2 Task: PLC\_Task

Priority: 10  
 Type: Cyclic  
 Interval: 500 Unit: ms  
 Watchdog: Active  
 Watchdog Time: 400 Unit: ms  
 Watchdog Sensitivity: 1  
 POUs: PLC\_PRG

#### 1.1.1.13.2.1 Program call: PLC\_PRG

### 1.1.1.13.3 Task: VISU\_TASK

Priority: 15  
 Type: Cyclic  
 Interval: 100 Unit: ms  
 Watchdog: Inactive  
 POUs: VisuElems.Visu\_Prg

#### 1.1.1.13.3.1 Program call: VisuElems.Visu\_Prg

### 1.1.1.14 Persistent Variables: PersistentVars

```

1  {attribute 'qualified_only'}
2  VAR_GLOBAL PERSISTENT RETAIN
3
4      // Generated instance path of persistent variable
5      HMI . AliveCounterPlc3Pv : AliveHmiPv ;
6      // Generated instance path of persistent variable
7      HMI . ValveEmission1Cmd : ControlAnalogHmiCmd ;
8      // Generated instance path of persistent variable
9      HMI . ValveEmission1Pv : ControlAnalogHmiPv ;
10     // Generated instance path of persistent variable
11     HMI . ValveEmission1AlarmMotorProtectionCmd : AlarmDigitalHmiCmd ;

```

---

1.1.1.14 Persistent Variables: PersistentVars

---

```
12      // Generated instance path of persistent variable
13      HMI . ValveEmission1AlarmMotorProtectionPv : AlarmDigitalHmiPv ;
14      // Generated instance path of persistent variable
15      HMI . ValveEmission1SignalDigitalMotorProtectionCmd : SignalDigitalHmiCmd
16      ;
17      // Generated instance path of persistent variable
18      HMI . ValveEmission1SignalDigitalMotorProtectionPv : SignalDigitalHmiPv ;
19      // Generated instance path of persistent variable
20      HMI . ValveEmission1ScalingPositionCmd : ScalingHmiCmd ;
21      // Generated instance path of persistent variable
22      HMI . ValveEmission1ScalingPositionPv : ScalingHmiPv ;
23      // Generated instance path of persistent variable
24      HMI . ValveEmission1ScalingControlValueCmd : ScalingHmiCmd ;
25      // Generated instance path of persistent variable
26      HMI . ValveEmission1ScalingControlValuePv : ScalingHmiPv ;
27      // Generated instance path of persistent variable
28      HMI . ValveEmission1PIDCmd : PIDHmiCmd ;
29      // Generated instance path of persistent variable
30      HMI . ValveEmission1PIDPv : PIDHmiPv ;
31      // Generated instance path of persistent variable
32      HMI . WaterLevelCmd : SignalAnalogHmiCmd ;
33      // Generated instance path of persistent variable
34      HMI . WaterLevelPv : SignalAnalogHmiPv ;
35      // Generated instance path of persistent variable
36      HMI . FlowLevelEmissionCmd : SignalAnalogHmiCmd ;
37      // Generated instance path of persistent variable
38      HMI . FlowLevelEmissionPv : SignalAnalogHmiPv ;
39
END_VAR
```

## **A.6 PLS program - Lekkasjedeteksjon**

## Project Documentation

File: PLS Lekkasjedeteksjon 09.05.2020.ecp

Date: 5/9/2020

Profile: e!COCKPIT

---

Table of Contents

## Table of Contents

1 Application: PLS1_Lekasje	3
1.1 Folder: GlobalData	3
1.1.1 Global Variable List: HMI	3
1.1.2 Global Variable List: IO	3
1.1.3 Global Variable List: Local	4
1.1.4 Global Variable List: SIM	4
1.1.5 Global Variable List: STATUS	5
1.2 POU: Alarms	5
1.3 POU: Control	5
1.4 POU: Initialize	6
1.5 POU: Inputs	9
1.6 POU: Outputs	9
1.7 POU: PLC_PRG	10
1.8 POU: RestartOpcUaMqttLinkOnDownload	11
1.9 POU: RestartOpcUaMqttLinkOnOnlineChange	11
1.10 POU: Simulation	11
1.11 POU: Simulation_MQTT	12

---

1 Application: PLS1\_Lekasje

---

# 1 Application: PLS1\_Lekasje

## 1.1 Folder: GlobalData

### 1.1.1 Global Variable List: HMI

```
1 {attribute 'qualified_only'}
2 VAR_GLOBAL PERSISTENT RETAIN
3
4     FlowLevel1Cmd : SignalAnalogHmiCmd ;
5     FlowLevel2Cmd : SignalAnalogHmiCmd ;
6     FlowLevel3Cmd : SignalAnalogHmiCmd ;
7     FlowLevel4Cmd : SignalAnalogHmiCmd ;
8     FlowLevel5Cmd : SignalAnalogHmiCmd ;
9     FlowLevel6Cmd : SignalAnalogHmiCmd ;
10    FlowLevel7Cmd : SignalAnalogHmiCmd ;
11
12    BatteryMonitoring1Cmd : PowerMonitoringHmiCmd ;
13    BatteryMonitoring2Cmd : PowerMonitoringHmiCmd ;
14    BatteryMonitoring3Cmd : PowerMonitoringHmiCmd ;
15 END_VAR
16
17 VAR_GLOBAL
18     AliveCounterPlc1Pv : AliveHmiPv ;
19
20     FlowLevel1Pv : SignalAnalogHmiPv ;
21     FlowLevel2Pv : SignalAnalogHmiPv ;
22     FlowLevel3Pv : SignalAnalogHmiPv ;
23     FlowLevel4Pv : SignalAnalogHmiPv ;
24     FlowLevel5Pv : SignalAnalogHmiPv ;
25     FlowLevel6Pv : SignalAnalogHmiPv ;
26     FlowLevel7Pv : SignalAnalogHmiPv ;
27
28     BatteryMonitoring1Pv : PowerMonitoringHmiPv ;
29     BatteryMonitoring2Pv : PowerMonitoringHmiPv ;
30     BatteryMonitoring3Pv : PowerMonitoringHmiPv ;
31 END_VAR
32
```

---

 1.1.2 Global Variable List: IO
 

---

```

1   {attribute 'qualified_only'}
2   VAR_GLOBAL
3     AI_FlowLevel1Pv : REAL ;
4     AI_FlowLevel2Pv : real ;
5     AI_FlowLevel3Pv : real ;
6     AI_FlowLevel4Pv : real ;
7     AI_FlowLevel5Pv : real ;
8     AI_FlowLevel6Pv : real ;
9     AI_FlowLevel7Pv : REAL ;
10
11
12     AI_CurrentIn01Pv : REAL ;
13     AI_CurentOut01Pv : REAL ;
14     AI_BatteryLevel01Pv : REAL ;
15     AI_CurrentIn02Pv : REAL ;
16     AI_CurentOut02Pv : REAL ;
17     AI_BatteryLevel02Pv : REAL ;
18     AI_CurrentIn03Pv : REAL ;
19     AI_CurentOut03Pv : REAL ;
20     AI_BatteryLevel03Pv : REAL ;
21   END_VAR
22

```

**1.1.3 Global Variable List: Local**

```

1   {attribute 'qualified_only'}
2   VAR_GLOBAL
3     FlowLevel1 : SignalAnalogFB ;
4     FlowLevel2 : SignalAnalogFB ;
5     FlowLevel3 : SignalAnalogFB ;
6     FlowLevel4 : SignalAnalogFB ;
7     FlowLevel5 : SignalAnalogFB ;
8     FlowLevel6 : SignalAnalogFB ;
9     FlowLevel7 : SignalAnalogFB ;
10    BatteryMonitoring1 : PowerMonitoringFB ;
11    BatteryMonitoring2 : PowerMonitoringFB ;
12    BatteryMonitoring3 : PowerMonitoringFB ;
13  END_VAR
14

```

**1.1.4 Global Variable List: SIM**

```

1   {attribute 'qualified_only'}
2   VAR_GLOBAL
3     WaterFlowFT01_Pv : REAL ;
4     WaterFlowFT02_Pv : REAL ;
5     WaterFlowFT03_Pv : REAL ;
6     WaterFlowFT04_Pv : REAL ;
7     WaterFlowFT05_Pv : REAL ;
8     WaterFlowFT06_Pv : REAL ;
9     WaterFlowFT07_Pv : REAL ;
10    CurrentIn01_Pv : REAL ;
11    CurentOut01_Pv : REAL ;

```

---

#### 1.1.4 Global Variable List: SIM

---

```

12      BatteryLevel01_Pv : REAL ;
13      CurrentIn02_Pv : REAL ;
14      CurentOut02_Pv : REAL ;
15      BatteryLevel02_Pv : REAL ;
16      CurrentIn03_Pv : REAL ;
17      CurentOut03_Pv : REAL ;
18      BatteryLevel03_Pv : REAL ;
19  END_VAR
20

```

### 1.1.5 Global Variable List: STATUS

```

1 {attribute 'qualified_only'}
2 VAR_GLOBAL
3     OpcUaMqttLinkStatus : OpcUa_Mqtt_Status ;
4 END_VAR
5

```

### 1.2 POU: Alarms

```

1 PROGRAM Alarms
2 VAR
3 END_VAR
4

```

---

### 1.3 POU: Control

```

1 PROGRAM Control
2 VAR
3 END_VAR
4

```

---

```

1 Local . BatteryMonitoring1 (_id := 'BatteryMonitoring1' , _owner := 'plc1'
2
3     Input_CurrentIn := IO . AI_CurrentIn01Pv ,
4     Input_CurrentOut := IO . AI_CurentOut01Pv ,
5     Input_CurrentStored := IO . AI_BatteryLevel01Pv ,
6     HmiCmd := HMI . BatteryMonitoring1Cmd ,
7     HmiPv := HMI . BatteryMonitoring1Pv ) ;
8
9 Local . BatteryMonitoring2 (_id := 'BatteryMonitoring2' , _owner := 'plc1'
10
11    Input_CurrentIn := IO . AI_CurrentIn02Pv ,
12    Input_CurrentOut := IO . AI_CurentOut02Pv ,
13    Input_CurrentStored := IO . AI_BatteryLevel02Pv ,
14    HmiCmd := HMI . BatteryMonitoring2Cmd ,
15    HmiPv := HMI . BatteryMonitoring2Pv ) ;
16
17 Local . BatteryMonitoring3 (_id := 'BatteryMonitoring3' , _owner := 'plc1'
18

```

---

### 1.3 POU: Control

---

```

17      Input_CurrentIn := IO . AI_CurrentIn03Pv ,
18      Input_CurrentOut := IO . AI_CurentOut03Pv ,
19      Input_CurrentStored := IO . AI_BatteryLevel03Pv ,
20      HmiCmd := HMI . BatteryMonitoring3Cmd ,
21      HmiPv := HMI . BatteryMonitoring3Pv ) ;
22

```

## 1.4 POU: Initialize

```

1   PROGRAM Initialize
2   VAR
3   END_VAR
4
5
6   HMI . BatteryMonitoring1Cmd . BatterySize_Cmd := 20 ;
7   HMI . BatteryMonitoring2Cmd . BatterySize_Cmd := 20 ;
8   HMI . BatteryMonitoring3Cmd . BatterySize_Cmd := 20 ;
9
10
11  Hmi . FlowLevel1Cmd . Tag := 'FT01' ;
12  Hmi . FlowLevel1Cmd . Description := 'Level of flow out of FT01' ;
13  Hmi . FlowLevel1Cmd . AlarmHysteresis := 10.0 ;
14  Hmi . FlowLevel1Cmd . AlarmHighHighLimit := 900.0 ;
15  Hmi . FlowLevel1Cmd . AlarmHighLimit := 800.0 ;
16  Hmi . FlowLevel1Cmd . AlarmLowLowLimit := 10.0 ;
17  Hmi . FlowLevel1Cmd . AlarmLowLimit := 20.0 ;
18  Hmi . FlowLevel1Cmd . AlarmHigh . Delay . OnDelayLimit := 1.0 ;
19  Hmi . FlowLevel1Cmd . AlarmHigh . Delay . OffDelayLimit := 1.0 ;
20  Hmi . FlowLevel1Cmd . AlarmHighHigh . Delay . OnDelayLimit := 1.0 ;
21  Hmi . FlowLevel1Cmd . AlarmHighHigh . Delay . OffDelayLimit := 1.0 ;
22  Hmi . FlowLevel1Cmd . AlarmLow . Delay . OnDelayLimit := 1.0 ;
23  Hmi . FlowLevel1Cmd . AlarmLow . Delay . OffDelayLimit := 1.0 ;
24  Hmi . FlowLevel1Cmd . AlarmLowLow . Delay . OnDelayLimit := 1.0 ;
25  Hmi . FlowLevel1Cmd . AlarmLowLow . Delay . OffDelayLimit := 1.0 ;
26  Hmi . FlowLevel1Cmd . InputScaling . InputUnit := 'na' ;
27  Hmi . FlowLevel1Cmd . InputScaling . OutputUnit := 'l/s' ;
28  Hmi . FlowLevel1Cmd . InputScaling . InputMax_Cmd := 100.0 ;
29  Hmi . FlowLevel1Cmd . InputScaling . InputMin_Cmd := 0.0 ;
30  Hmi . FlowLevel1Cmd . InputScaling . OutputMax_Cmd := 100.0 ;
31  Hmi . FlowLevel1Cmd . InputScaling . OutputMin_Cmd := 0.0 ;
32
33
34
35
36
37
38
39
40
41  Hmi . FlowLevel2Cmd . Tag := 'FT02' ;
42  Hmi . FlowLevel2Cmd . Description := 'Level of flow out of FT02' ;
43  Hmi . FlowLevel2Cmd . AlarmHysteresis := 10.0 ;
44  Hmi . FlowLevel2Cmd . AlarmHighHighLimit := 900.0 ;
45  Hmi . FlowLevel2Cmd . AlarmHighLimit := 800.0 ;
46  Hmi . FlowLevel2Cmd . AlarmLowLowLimit := 10.0 ;
47  Hmi . FlowLevel2Cmd . AlarmLowLimit := 20.0 ;
48  Hmi . FlowLevel2Cmd . AlarmHigh . Delay . OnDelayLimit := 1.0 ;
49  Hmi . FlowLevel2Cmd . AlarmHigh . Delay . OffDelayLimit := 1.0 ;
50  Hmi . FlowLevel2Cmd . AlarmHighHigh . Delay . OnDelayLimit := 1.0 ;
51  Hmi . FlowLevel2Cmd . AlarmHighHigh . Delay . OffDelayLimit := 1.0 ;
52  Hmi . FlowLevel2Cmd . AlarmLow . Delay . OnDelayLimit := 1.0 ;
53  Hmi . FlowLevel2Cmd . AlarmLow . Delay . OffDelayLimit := 1.0 ;
54  Hmi . FlowLevel2Cmd . AlarmLowLow . Delay . OnDelayLimit := 1.0 ;

```

## 1.4 POU: Initialize

```

42   Hmi . FlowLevel2Cmd . AlarmLowLow . Delay . OffDelayLimit := 1.0 ;
43   Hmi . FlowLevel2Cmd . InputScaling . InputUnit := 'na' ;
44   Hmi . FlowLevel2Cmd . InputScaling . OutputUnit := 'l/s' ;
45   Hmi . FlowLevel2Cmd . InputScaling . InputMax_Cmd := 100.0 ;
46   Hmi . FlowLevel2Cmd . InputScaling . InputMin_Cmd := 0.0 ;
47   Hmi . FlowLevel2Cmd . InputScaling . OutputMax_Cmd := 100.0 ;
48   Hmi . FlowLevel2Cmd . InputScaling . OutputMin_Cmd := 0.0 ;
49
50   Hmi . FlowLevel3Cmd . Tag := 'FT03' ;
51   Hmi . FlowLevel3Cmd . Description := 'Level of flow out of FT03' ;
52   Hmi . FlowLevel3Cmd . AlarmHysteresis := 10.0 ;
53   Hmi . FlowLevel3Cmd . AlarmHighHighLimit := 900.0 ;
54   Hmi . FlowLevel3Cmd . AlarmHighLimit := 800.0 ;
55   Hmi . FlowLevel3Cmd . AlarmLowLowLimit := 10.0 ;
56   Hmi . FlowLevel3Cmd . AlarmLowLimit := 20.0 ;
57   Hmi . FlowLevel3Cmd . AlarmHigh . Delay . OnDelayLimit := 1.0 ;
58   Hmi . FlowLevel3Cmd . AlarmHigh . Delay . OffDelayLimit := 1.0 ;
59   Hmi . FlowLevel3Cmd . AlarmHighHigh . Delay . OnDelayLimit := 1.0 ;
60   Hmi . FlowLevel3Cmd . AlarmHighHigh . Delay . OffDelayLimit := 1.0 ;
61   Hmi . FlowLevel3Cmd . AlarmLow . Delay . OnDelayLimit := 1.0 ;
62   Hmi . FlowLevel3Cmd . AlarmLow . Delay . OffDelayLimit := 1.0 ;
63   Hmi . FlowLevel3Cmd . AlarmLowLow . Delay . OnDelayLimit := 1.0 ;
64   Hmi . FlowLevel3Cmd . AlarmLowLow . Delay . OffDelayLimit := 1.0 ;
65   Hmi . FlowLevel3Cmd . InputScaling . InputUnit := 'na' ;
66   Hmi . FlowLevel3Cmd . InputScaling . OutputUnit := 'l/s' ;
67   Hmi . FlowLevel3Cmd . InputScaling . InputMax_Cmd := 100.0 ;
68   Hmi . FlowLevel3Cmd . InputScaling . InputMin_Cmd := 0.0 ;
69   Hmi . FlowLevel3Cmd . InputScaling . OutputMax_Cmd := 100.0 ;
70   Hmi . FlowLevel3Cmd . InputScaling . OutputMin_Cmd := 0.0 ;
71
72   Hmi . FlowLevel4Cmd . Tag := 'FT04' ;
73   Hmi . FlowLevel4Cmd . Description := 'Level of flow out of FT04' ;
74   Hmi . FlowLevel4Cmd . AlarmHysteresis := 10.0 ;
75   Hmi . FlowLevel4Cmd . AlarmHighHighLimit := 900.0 ;
76   Hmi . FlowLevel4Cmd . AlarmHighLimit := 800.0 ;
77   Hmi . FlowLevel4Cmd . AlarmLowLowLimit := 10.0 ;
78   Hmi . FlowLevel4Cmd . AlarmLowLimit := 20.0 ;
79   Hmi . FlowLevel4Cmd . AlarmHigh . Delay . OnDelayLimit := 1.0 ;
80   Hmi . FlowLevel4Cmd . AlarmHigh . Delay . OffDelayLimit := 1.0 ;
81   Hmi . FlowLevel4Cmd . AlarmHighHigh . Delay . OnDelayLimit := 1.0 ;
82   Hmi . FlowLevel4Cmd . AlarmHighHigh . Delay . OffDelayLimit := 1.0 ;
83   Hmi . FlowLevel4Cmd . AlarmLow . Delay . OnDelayLimit := 1.0 ;
84   Hmi . FlowLevel4Cmd . AlarmLow . Delay . OffDelayLimit := 1.0 ;
85   Hmi . FlowLevel4Cmd . AlarmLowLow . Delay . OnDelayLimit := 1.0 ;
86   Hmi . FlowLevel4Cmd . AlarmLowLow . Delay . OffDelayLimit := 1.0 ;
87   Hmi . FlowLevel4Cmd . InputScaling . InputUnit := 'na' ;
88   Hmi . FlowLevel4Cmd . InputScaling . OutputUnit := 'l/s' ;
89   Hmi . FlowLevel4Cmd . InputScaling . InputMax_Cmd := 100.0 ;
90   Hmi . FlowLevel4Cmd . InputScaling . InputMin_Cmd := 0.0 ;
91   Hmi . FlowLevel4Cmd . InputScaling . OutputMax_Cmd := 100.0 ;
92   Hmi . FlowLevel4Cmd . InputScaling . OutputMin_Cmd := 0.0 ;
93
94   Hmi . FlowLevel5Cmd . Tag := 'FT05' ;
95   Hmi . FlowLevel5Cmd . Description := 'Level of flow out of FT05' ;
96   Hmi . FlowLevel5Cmd . AlarmHysteresis := 10.0 ;
97   Hmi . FlowLevel5Cmd . AlarmHighHighLimit := 900.0 ;

```

## 1.4 POU: Initialize

```

98     Hmi . FlowLevel5Cmd . AlarmHighLimit   := 800.0 ;
99     Hmi . FlowLevel5Cmd . AlarmLowLowLimit := 10.0 ;
100    Hmi . FlowLevel5Cmd . AlarmLowLimit   := 20.0 ;
101    Hmi . FlowLevel5Cmd . AlarmHigh . Delay . OnDelayLimit := 1.0 ;
102    Hmi . FlowLevel5Cmd . AlarmHigh . Delay . OffDelayLimit := 1.0 ;
103    Hmi . FlowLevel5Cmd . AlarmHighHigh . Delay . OnDelayLimit := 1.0 ;
104    Hmi . FlowLevel5Cmd . AlarmHighHigh . Delay . OffDelayLimit := 1.0 ;
105    Hmi . FlowLevel5Cmd . AlarmLow . Delay . OnDelayLimit := 1.0 ;
106    Hmi . FlowLevel5Cmd . AlarmLow . Delay . OffDelayLimit := 1.0 ;
107    Hmi . FlowLevel5Cmd . AlarmLowLow . Delay . OnDelayLimit := 1.0 ;
108    Hmi . FlowLevel5Cmd . AlarmLowLow . Delay . OffDelayLimit := 1.0 ;
109    Hmi . FlowLevel5Cmd . InputScaling . InputUnit := 'na' ;
110    Hmi . FlowLevel5Cmd . InputScaling . OutputUnit := 'l/s' ;
111    Hmi . FlowLevel5Cmd . InputScaling . InputMax_Cmd := 100.0 ;
112    Hmi . FlowLevel5Cmd . InputScaling . InputMin_Cmd := 0.0 ;
113    Hmi . FlowLevel5Cmd . InputScaling . OutputMax_Cmd := 100.0 ;
114    Hmi . FlowLevel5Cmd . InputScaling . OutputMin_Cmd := 0.0 ;
115
116    Hmi . FlowLevel6Cmd . Tag := 'FT06' ;
117    Hmi . FlowLevel6Cmd . Description := 'Level of flow out of FT06' ;
118    Hmi . FlowLevel6Cmd . AlarmHysteresis := 10.0 ;
119    Hmi . FlowLevel6Cmd . AlarmHighHighLimit := 900.0 ;
120    Hmi . FlowLevel6Cmd . AlarmHighLimit := 800.0 ;
121    Hmi . FlowLevel6Cmd . AlarmLowLowLimit := 10.0 ;
122    Hmi . FlowLevel6Cmd . AlarmLowLimit := 20.0 ;
123    Hmi . FlowLevel6Cmd . AlarmHigh . Delay . OnDelayLimit := 1.0 ;
124    Hmi . FlowLevel6Cmd . AlarmHigh . Delay . OffDelayLimit := 1.0 ;
125    Hmi . FlowLevel6Cmd . AlarmHighHigh . Delay . OnDelayLimit := 1.0 ;
126    Hmi . FlowLevel6Cmd . AlarmHighHigh . Delay . OffDelayLimit := 1.0 ;
127    Hmi . FlowLevel6Cmd . AlarmLow . Delay . OnDelayLimit := 1.0 ;
128    Hmi . FlowLevel6Cmd . AlarmLow . Delay . OffDelayLimit := 1.0 ;
129    Hmi . FlowLevel6Cmd . AlarmLowLow . Delay . OnDelayLimit := 1.0 ;
130    Hmi . FlowLevel6Cmd . AlarmLowLow . Delay . OffDelayLimit := 1.0 ;
131    Hmi . FlowLevel6Cmd . InputScaling . InputUnit := 'na' ;
132    Hmi . FlowLevel6Cmd . InputScaling . OutputUnit := 'l/s' ;
133    Hmi . FlowLevel6Cmd . InputScaling . InputMax_Cmd := 100.0 ;
134    Hmi . FlowLevel6Cmd . InputScaling . InputMin_Cmd := 0.0 ;
135    Hmi . FlowLevel6Cmd . InputScaling . OutputMax_Cmd := 100.0 ;
136    Hmi . FlowLevel6Cmd . InputScaling . OutputMin_Cmd := 0.0 ;
137
138    Hmi . FlowLevel7Cmd . Tag := 'FT07' ;
139    Hmi . FlowLevel7Cmd . Description := 'Level of flow out of FT07' ;
140    Hmi . FlowLevel7Cmd . AlarmHysteresis := 10.0 ;
141    Hmi . FlowLevel7Cmd . AlarmHighHighLimit := 900.0 ;
142    Hmi . FlowLevel7Cmd . AlarmHighLimit := 800.0 ;
143    Hmi . FlowLevel7Cmd . AlarmLowLowLimit := 10.0 ;
144    Hmi . FlowLevel7Cmd . AlarmLowLimit := 20.0 ;
145    Hmi . FlowLevel7Cmd . AlarmHigh . Delay . OnDelayLimit := 1.0 ;
146    Hmi . FlowLevel7Cmd . AlarmHigh . Delay . OffDelayLimit := 1.0 ;
147    Hmi . FlowLevel7Cmd . AlarmHighHigh . Delay . OnDelayLimit := 1.0 ;
148    Hmi . FlowLevel7Cmd . AlarmHighHigh . Delay . OffDelayLimit := 1.0 ;
149    Hmi . FlowLevel7Cmd . AlarmLow . Delay . OnDelayLimit := 1.0 ;
150    Hmi . FlowLevel7Cmd . AlarmLow . Delay . OffDelayLimit := 1.0 ;
151    Hmi . FlowLevel7Cmd . AlarmLowLow . Delay . OnDelayLimit := 1.0 ;
152    Hmi . FlowLevel7Cmd . AlarmLowLow . Delay . OffDelayLimit := 1.0 ;
153    Hmi . FlowLevel7Cmd . InputScaling . InputUnit := 'na' ;

```

---

#### 1.4 POU: Initialize

---

```

154     Hmi . FlowLevel7Cmd . InputScaling . OutputUnit   := '1/s' ;
155     Hmi . FlowLevel7Cmd . InputScaling . InputMax_Cmd  := 100.0 ;
156     Hmi . FlowLevel7Cmd . InputScaling . InputMin_Cmd  := 0.0 ;
157     Hmi . FlowLevel7Cmd . InputScaling . OutputMax_Cmd := 100.0 ;
158     Hmi . FlowLevel7Cmd . InputScaling . OutputMin_Cmd := 0.0 ;
159

```

### 1.5 POU: Inputs

```

1      PROGRAM  Inputs
2      VAR
3      END_VAR
4

```

---

```

1
2      Local . FlowLevel1 (_id  := 'FlowLevel1' , _owner  := 'plc1' ,
3                           RawInput  := IO . AI_FlowLevel1Pv ,
4                           HmiCmd    := HMI . FlowLevel1Cmd ,
5                           HmiPv     := HMI . FlowLevel1Pv ) ;
6
7      Local . FlowLevel2 (_id  := 'FlowLevel2' , _owner  := 'plc1' ,
8                           RawInput  := IO . AI_FlowLevel2Pv ,
9                           HmiCmd    := HMI . FlowLevel2Cmd ,
10                          HmiPv     := HMI . FlowLevel2Pv ) ;
11
12     Local . FlowLevel3 (_id  := 'FlowLevel3' , _owner  := 'plc1' ,
13                           RawInput  := IO . AI_FlowLevel3Pv ,
14                           HmiCmd    := HMI . FlowLevel3Cmd ,
15                           HmiPv     := HMI . FlowLevel3Pv ) ;
16
17     Local . FlowLevel4 (_id  := 'FlowLevel4' , _owner  := 'plc1' ,
18                           RawInput  := IO . AI_FlowLevel4Pv ,
19                           HmiCmd    := HMI . FlowLevel4Cmd ,
20                           HmiPv     := HMI . FlowLevel4Pv ) ;
21
22     Local . FlowLevel5 (_id  := 'FlowLevel5' , _owner  := 'plc1' ,
23                           RawInput  := IO . AI_FlowLevel5Pv ,
24                           HmiCmd    := HMI . FlowLevel5Cmd ,
25                           HmiPv     := HMI . FlowLevel5Pv ) ;
26
27     Local . FlowLevel6 (_id  := 'FlowLevel6' , _owner  := 'plc1' ,
28                           RawInput  := IO . AI_FlowLevel6Pv ,
29                           HmiCmd    := HMI . FlowLevel6Cmd ,
30                           HmiPv     := HMI . FlowLevel6Pv ) ;
31
32     Local . FlowLevel7 (_id  := 'FlowLevel7' , _owner  := 'plc1' ,
33                           RawInput  := IO . AI_FlowLevel7Pv ,
34                           HmiCmd    := HMI . FlowLevel7Cmd ,
35                           HmiPv     := HMI . FlowLevel7Pv ) ;
36

```

---

 1.6 POU: Outputs
 

---

**1.6 POU: Outputs**

```

1   PROGRAM Outputs
2   VAR
3   END_VAR
4

```

---

**1.7 POU: PLC\_PRG**

```

1   PROGRAM PLC_PRG
2   VAR _RETAIN
3       _firstScanAfterDownload : BOOL := TRUE ;
4   END_VAR
5   VAR
6       _counter : INT ;
7       _lastRun : STRING ;
8   END_VAR
9

10
11  // Signaling that the PLC is alive
12  Hmi . AliveCounterPlc1Pv . _tagId := 'AliveCounterPlc1Pv' ;
13  Hmi . AliveCounterPlc1Pv . _type := 'AliveHmi' ;
14  Hmi . AliveCounterPlc1Pv . _owner := 'plc1' ;
15  HMI . AliveCounterPlc1Pv . CounterPv := HMI . AliveCounterPlc1Pv . CounterPv +
16  1 ;
17
18  _counter := STATUS . OpcUaMqttLinkStatus . AliveCounterPv ;
19  _lastRun := STATUS . OpcUaMqttLinkStatus . LastRun ;
20
21
22
23
24
25
26
27
28
29
30
31  // Always last

```

---

---

 1.7 POU: PLC\_PRG
 

---

```

32      _firstScanAfterDownload := FALSE ;
33

```

### 1.8 POU: RestartOpcUaMqttLinkOnDownload

```

1   FUNCTION RestartOpcUaMqttLinkOnDownload : DWORD
2   VAR_IN_OUT
3       EventPrm : CmpApp . EVTPARAM_CmpApp ;
4   END_VAR
5   VAR
6   END_VAR
7

1   /*
2   This function is called when a download is about to happen.
3   This will force the Opc Ua - MQTT link to restart.
4   The restart will take 10 seconds, allowing e!Cockpit to download and start
5   before the link starts.
6   */
7   Status . OpcUaMqttLinkStatus . Restart_Cmd := TRUE ;

```

### 1.9 POU: RestartOpcUaMqttLinkOnOnlineChange

```

1   FUNCTION RestartOpcUaMqttLinkOnOnlineChange : DWORD
2   VAR_IN_OUT
3       EventPrm : CmpApp . EVTPARAM_CmpApp ;
4   END_VAR
5   VAR
6   END_VAR
7

1   /*
2   This function is called when a online change is about to happen.
3   This will force the Opc Ua - MQTT link to restart.
4   The restart will take 10 seconds, allowing e!Cockpit to download and start
5   before the link starts.
6   */
7   Status . OpcUaMqttLinkStatus . Restart_Cmd := TRUE ;

```

### 1.10 POU: Simulation

```

1   PROGRAM Simulation
2   VAR
3   END_VAR
4

1   // Simulation because no physical IO is connected
2   IO . AI_FlowLevel1Pv := SIM . WaterFlowFT01_Pv ;
3   IO . AI_FlowLevel2Pv := SIM . WaterFlowFT02_Pv ;
4   IO . AI_FlowLevel3Pv := SIM . WaterFlowFT03_Pv ;
5   IO . AI_FlowLevel4Pv := SIM . WaterFlowFT04_Pv ;

```

---

### 1.10 POU: Simulation

---

```

6   IO . AI_FlowLevel5Pv := SIM . WaterFlowFT05_Pv ;
7   IO . AI_FlowLevel6Pv := SIM . WaterFlowFT06_Pv ;
8   IO . AI_FlowLevel7Pv := SIM . WaterFlowFT07_Pv ;
9
10  IO . AI_CurrentIn01Pv := SIM . CurrentIn01_Pv ;
11  IO . AI_CurentOut01Pv := SIM . CurentOut01_Pv ;
12  IO . AI_BatteryLevel01Pv := SIM . BatteryLevel01_Pv ;
13  IO . AI_CurrentIn02Pv := SIM . CurrentIn02_Pv ;
14  IO . AI_CurentOut02Pv := SIM . CurentOut02_Pv ;
15  IO . AI_BatteryLevel02Pv := SIM . BatteryLevel02_Pv ;
16  IO . AI_CurrentIn03Pv := SIM . CurrentIn03_Pv ;
17  IO . AI_CurentOut03Pv := SIM . CurentOut03_Pv ;
18  IO . AI_BatteryLevel03Pv := SIM . BatteryLevel03_Pv ;
19

```

### 1.11 POU: *Simulation\_MQTT*

```

1  PROGRAM Simulation_MQTT
2  VAR
3      MqttStatus          : WagoAppCloud . FbStatus_NativeMQTT ;
4      MqttError           : BOOL := FALSE ;
5      MqttConnectedToMyCloud : BOOL := FALSE ;
6      MqttFillLevel        : REAL := 0 ;
7      MqttOutgoingBlocks   : ULINT := 0 ;
8
9
10     MqttSubscribe         : WagoAppCloud . FbSubscribeMQTT ;
11     MqttSubscriptionDataBytes : ARRAY [ 0 .. 512 ] OF BYTE ;
12     MqttSubscriptionDataJSONString : STRING ( JSON_MAX_STRING ) ;
13     // Resulting JSON string
14     MqttSubscriptionDataJSONPointer : WagoAppJSON . typJSON_Pointer ;
15     MqttSubscriptionJSONParser    : Fb_JSON_ParseAndModify ; // 
16     // JSON parser function block
16     MqttSubscribeNewData : BOOL ;
17
18     oParserStatus       : WagoAppJSON . WagoSysErrorHandler . FbResult ;
19     xParserError        : BOOL ;
20     xParserDone         : BOOL ;
21     diParserToken       : DINT ;
22     xParserBuildup     : BOOL ;
23     xParserTrigger      : ARRAY [ 0 .. 15 ] OF BOOL ;
24
25     MqttPublish          : WagoAppCloud . FbPublishMQTT ;
26     MqttPublishParameters : ARRAY [ 0 .. 10 ] OF STRING ;
27     MqttPublishTemplate   : STRING ( 512 ) := '{"#Parameter": "#Parameter"}'
;
28     JSONWriter          : FB_JSON_Writer_01 ;
29     MqttPublishPayload    : STRING ( 1024 ) ;
30     MqttPublishBuffer     : ARRAY [ 0 .. 1999 ] OF BYTE ;
31     MqttPublishTrigger    : BOOL ;
32     JSONWriter_Done      : BOOL ;
33     JSONWriter_Error     : BOOL ;
34     JSONWriter_Status    : WagoSysErrorHandler . FbResult ;
35     JSONWriterTrigger    : BOOL ;

```

## 1.11 POU: Simulation\_MQTT

---

```

36      END_VAR
37

1      // Status
2      MqttStatus ( xEnabled := TRUE ,
3                  xError => MqttError ,
4                  xCloudConnected => MqttConnectedToMyCloud ,           // Will be
5                  FALSE after network timeout occurred
6                  rCacheFillLevel => MqttFillLevel ,                   //
7                  Increasing value indicates bad network or overload
8                  uIOutgoingDataBlocks => MqttOutgoingBlocks ) ;     // Only for
9                  QoS 1 and 2. Increasing value indicates bad network or overload
10
11
12
13
14
15      MemCopySecure ( pDest := ADR ( MqttSubscriptionDataJSONString ) , udiDestSize
16      := 255 , pSource := ADR ( MqttSubscriptionDataBytes ) , udiSourceSize :=
17      MqttSubscribe . dwRxNBytes , bPadding := 0 ) ;
18
19
20      // Execute this function block once to complete the parser buildup
21      process
22      // This must be done before calling the parser methods
23
24      IF MqttSubscribeNewData THEN
25          xParserBuildup := TRUE ;
26      END_IF
27
28      MqttSubscriptionJSONParser (
29          pData := ADR ( MqttSubscriptionDataJSONString ) ,           //
30          JSON string to parse
31          udiSizeData := length ( MqttSubscriptionDataJSONString ) ,
32          oStatus => oParserStatus ,
33          xError => xParserError ,
34          xDone => xParserDone ,
35          diToken => diParserToken ,
36          xTrigger := xParserBuildup ) ;           // Set xParserBuildup TRUE to
37          execute the function block
38
39      IF xParserDone THEN
40          xParserTrigger [ 0 ] := TRUE ;           // Set xParserTrigger TRUE
41          to execute the parser method
42          xParserTrigger [ 1 ] := TRUE ;
43          xParserTrigger [ 2 ] := TRUE ;
44          xParserTrigger [ 3 ] := TRUE ;
45          xParserTrigger [ 4 ] := TRUE ;
46          xParserTrigger [ 5 ] := TRUE ;
47          xParserTrigger [ 6 ] := TRUE ;
48          xParserTrigger [ 7 ] := TRUE ;
49          xParserTrigger [ 8 ] := TRUE ;
50          xParserTrigger [ 9 ] := TRUE ;

```

## 1.11 POU: Simulation\_MQTT

```

44     xParserTrigger [ 10 ] := TRUE ;
45     xParserTrigger [ 11 ] := TRUE ;
46     xParserTrigger [ 12 ] := TRUE ;
47     xParserTrigger [ 13 ] := TRUE ;
48     xParserTrigger [ 14 ] := TRUE ;
49     xParserTrigger [ 15 ] := TRUE ;
50     xParserDone := FALSE ;
51 END_IF
52
53 //      Call the parser method GetValueByPath for each value to be
54 //      extracted from the JSON string
55 MqttSubscriptionDataJSONPointer := MqttSubscriptionJSONParser .
56 GetValueByPath ( sPointer := '/FT01' , xTrigger := xParserTrigger [ 0 ] ) ;
57 SIM . WaterFlowFT01_Pv := STRING_TO_REAL ( MqttSubscriptionDataJSONPointer .
58 sValue ) ;
59
60 MqttSubscriptionDataJSONPointer := MqttSubscriptionJSONParser .
61 GetValueByPath ( sPointer := '/FT02' , xTrigger := xParserTrigger [ 1 ] ) ;
62 SIM . WaterFlowFT02_Pv := STRING_TO_REAL ( MqttSubscriptionDataJSONPointer .
63 sValue ) ;
64
65 MqttSubscriptionDataJSONPointer := MqttSubscriptionJSONParser .
66 GetValueByPath ( sPointer := '/FT03' , xTrigger := xParserTrigger [ 2 ] ) ;
67 SIM . WaterFlowFT03_Pv := STRING_TO_REAL ( MqttSubscriptionDataJSONPointer .
68 sValue ) ;
69
70 MqttSubscriptionDataJSONPointer := MqttSubscriptionJSONParser .
71 GetValueByPath ( sPointer := '/FT04' , xTrigger := xParserTrigger [ 3 ] ) ;
72 SIM . WaterFlowFT04_Pv := STRING_TO_REAL ( MqttSubscriptionDataJSONPointer .
73 sValue ) ;
74
75 MqttSubscriptionDataJSONPointer := MqttSubscriptionJSONParser .
76 GetValueByPath ( sPointer := '/FT05' , xTrigger := xParserTrigger [ 4 ] ) ;
77 SIM . WaterFlowFT05_Pv := STRING_TO_REAL ( MqttSubscriptionDataJSONPointer .
78 sValue ) ;
79
80 MqttSubscriptionDataJSONPointer := MqttSubscriptionJSONParser .
81 GetValueByPath ( sPointer := '/CI01' , xTrigger := xParserTrigger [ 7 ] ) ;
82 SIM . CurrentIn01_Pv := STRING_TO_REAL ( MqttSubscriptionDataJSONPointer .
83 sValue ) ;
84
85 MqttSubscriptionDataJSONPointer := MqttSubscriptionJSONParser .
86 GetValueByPath ( sPointer := '/CO01' , xTrigger := xParserTrigger [ 8 ] ) ;
87 SIM . CurrentOut01_Pv := STRING_TO_REAL ( MqttSubscriptionDataJSONPointer .
88 sValue ) ;
89
90

```

1.11 POU: Simulation\_MQTT

---

```
81     MqttSubscriptionDataJSONPointer := MqttSubscriptionJSONParser .  
82     GetValueByPath ( sPointer := '/BL01' , xTrigger := xParserTrigger [ 9 ] ) ;  
83     SIM . BatteryLevel01_Pv := STRING_TO_REAL ( MqttSubscriptionDataJSONPointer .  
84     sValue ) ;  
85  
86     MqttSubscriptionDataJSONPointer := MqttSubscriptionJSONParser .  
87     GetValueByPath ( sPointer := '/CI02' , xTrigger := xParserTrigger [ 10 ] ) ;  
88     SIM . CurrentIn02_Pv := STRING_TO_REAL ( MqttSubscriptionDataJSONPointer .  
89     sValue ) ;  
90  
91     MqttSubscriptionDataJSONPointer := MqttSubscriptionJSONParser .  
92     GetValueByPath ( sPointer := '/CO02' , xTrigger := xParserTrigger [ 11 ] ) ;  
93     SIM . CurrentOut02_Pv := STRING_TO_REAL ( MqttSubscriptionDataJSONPointer .  
94     sValue ) ;  
95  
96     MqttSubscriptionDataJSONPointer := MqttSubscriptionJSONParser .  
97     GetValueByPath ( sPointer := '/CI03' , xTrigger := xParserTrigger [ 13 ] ) ;  
98     SIM . CurrentIn03_Pv := STRING_TO_REAL ( MqttSubscriptionDataJSONPointer .  
99     sValue ) ;  
100    MqttSubscriptionDataJSONPointer := MqttSubscriptionJSONParser .  
101    GetValueByPath ( sPointer := '/CO03' , xTrigger := xParserTrigger [ 14 ] ) ;  
102    SIM . CurrentOut03_Pv := STRING_TO_REAL ( MqttSubscriptionDataJSONPointer .  
103    sValue ) ;  
104
```

## A.7 PLS program - Trykkregulering

## Project Documentation

File: PLS Trykkregulering 02.05.2020.ecp

Date: 5/9/2020

Profile: e!COCKPIT

---

Table of Contents

## Table of Contents

1 Device: PFCx00_SmartCoupler	3
1.1 PLC Logic: Plc Logic	3
1.1.1 Application: PLS2_Trykkregulering	4

---

## 1 Device: PFCx00\_SmartCoupler

---

### Users and Groups

Users:

Groups:

---

### Access Rights

View  
Modify  
Execute  
Add/remove children

---

### Symbol Rights

---

### Parameters

#### **Parameters:**

Name:	Type:	Value:	Default Value:	Unit:	Description:
Processor Load Lower Limit	DWORD	80	80		
Processor Load Upper Limit	DWORD	90	90		
Processor Load Processor Share	DWORD	90	90		
Processor Load Should Throw	bool	FALSE	FALSE		
ProcessorLoadWatchdog_Exception					

---

### Information

Name: 750-8213 PFC200 G2 2ETH CAN  
Vendor: WAGO  
Categories: PLCs  
Type: 4096  
ID: 1006 120C  
Version: 5.13.2.32  
Order number: 0750-8213  
Description: Programmable Ethernet fieldbus coupler

---

---

1.1 PLC Logic: Plc Logic

---

## 1.1 PLC Logic: Plc Logic

### 1.1.1 Application: PLS2\_Trykkregulering

#### 1.1.1.1 Folder: GlobalData

##### 1.1.1.1.1 Global Variable List: HMI

```
1 {attribute 'qualified_only'}
2 VAR_GLOBAL PERSISTENT RETAIN
3 AliveCounterPlc2Pv : AliveHmiPv ;
4
5 DeliveryPump1Cmd : ControlAnalogHmiCmd ;
6 DeliveryPump1Pv : ControlAnalogHmiPv ;
7 DeliveryPump2Cmd : ControlAnalogHmiCmd ;
8 DeliveryPump2Pv : ControlAnalogHmiPv ;
9 DeliveryPump3Cmd : ControlAnalogHmiCmd ;
10 DeliveryPump3Pv : ControlAnalogHmiPv ;
11 DeliveryPumpPIDCmd : PIDHmiCmd ;
12 DeliveryPumpPIDPv : PIDHmiPv ;
13 DeliveryPumpWorkDistPv : WorkDistributorHmiPv ;
14 TreatmentPressureCmd : SignalAnalogHmiCmd ;
15 TreatmentPressurePv : SignalAnalogHmiPv ;
16 ConsumerPressureCmd : SignalAnalogHmiCmd ;
17 ConsumerPressurePv : SignalAnalogHmiPv ;
18 END_VAR
19
```

##### 1.1.1.1.2 Global Variable List: IO

```
1 {attribute 'qualified_only'}
2 VAR_GLOBAL
3 AI_DeliveryPump1SpeedPv : REAL ;
4 AI_DeliveryPump2SpeedPv : Real ;
5 AI_DeliveryPump3SpeedPv : Real ;
6
7 AO_DeliveryPump1ControlValue : real ;
8 AO_DeliveryPump2ControlValue : real ;
9 AO_DeliveryPump3ControlValue : REAL ;
10
11 AI_ConsumerPressurePv : REAL ;
12 AI_TreatmentPressurePv : REAL ;
13 END_VAR
14
```

---

1.1.1.1.3 Global Variable List: Local

---

```
1 {attribute 'qualified_only'}
2 VAR_GLOBAL
3 DeliveryPump1 : ControlAnalogFB ;
4 DeliveryPump2 : ControlAnalogFB ;
5 DeliveryPump3 : ControlAnalogFB ;
6 DeliveryPumpPID : PIDFB ;
7 DeliveryPumpWorkDist : WorkDistributorFB ;
8 TreatmentPressure : SignalAnalogFB ;
9 ConsumerPressure : SignalAnalogFB ;
10 END_VAR
11
```

#### 1.1.1.1.4 Global Variable List: SIM

```
1 {attribute 'qualified_only'}
2 VAR_GLOBAL
3     Disturbance : REAL ;
4     Process_y : REAL ;
5     Process : REAL ;
6     Ref : REAL ;
7     Output : REAL ;
8 END_VAR
9
```

#### 1.1.1.1.5 Global Variable List: STATUS

```
1 {attribute 'qualified_only'}
2 VAR_GLOBAL
3     OpcUaMqttLinkStatus : OpcUa_Mqtt_Status ;
4 END_VAR
5
```

#### 1.1.1.2 POU: Alarms

```
1 PROGRAM Alarms
2 VAR
3 END_VAR
4
```

---

## 1.1.1.3 POU: Control

```

1   PROGRAM Control
2   VAR
3
4   END_VAR
5

```

```

1
2   // Regulator block
3   Local . DeliveryPumpPID ( _id := 'DeliveryPumpPID' , _owner := 'plc2' ,
4     InputReference := 6.0 ,
5     ProcessMeasure := IO . AI_ConsumerPressurePv ,
6     Ts := 0.1 ,
7     uMax := 100.0 ,
8     uMin := 0.0 ,
9     HmiCmd := Hmi . DeliveryPumpPIDCmd ,
10    HmiPv := Hmi . DeliveryPumpPIDPv ) ;
11
12  // Work distribution
13  Local . DeliveryPumpWorkDist ( _id := 'DeliveryPumpWorkDist' , _owner :=
14    'plc2' ,
15    InputControlSignal := Local . DeliveryPumpPID . Output ,
16    Tag1 := Hmi . DeliveryPump1Cmd . Tag ,
17    Tag2 := Hmi . DeliveryPump2Cmd . Tag ,
18    Tag3 := Hmi . DeliveryPump3Cmd . Tag ,
19    InAuto1 := Hmi . DeliveryPump1Cmd . Auto ,
20    InAuto2 := Hmi . DeliveryPump2Cmd . Auto ,
21    InAuto3 := Hmi . DeliveryPump3Cmd . Auto ,
22    RunHours1 := Hmi . DeliveryPump1Pv . RunHours_Pv ,
23    RunHours2 := Hmi . DeliveryPump2Pv . RunHours_Pv ,
24    RunHours3 := Hmi . DeliveryPump3Pv . RunHours_Pv ,
25    ChangeInterval := 60.0 ,
26    HmiPv := HMI . DeliveryPumpWorkDistPv ) ;
27
28  // Pumps
29  Local . DeliveryPump1 ( _id := 'DeliveryPump1' , _owner := 'plc2' ,
30    HmiCmd := HMI . DeliveryPump1Cmd ,
31    HmiPv := HMI . DeliveryPump1Pv ,
32    Interlock := FALSE ,
33    Feedback := IO . AI_DeliveryPump1SpeedPv ,
34    ControlValueAuto_Cmd := Local . DeliveryPumpWorkDist . Out1 ) ;
35
36  Local . DeliveryPump2 ( _id := 'DeliveryPump2' , _owner := 'plc2' ,
37    HmiCmd := HMI . DeliveryPump2Cmd ,
38    HmiPv := HMI . DeliveryPump2Pv ,
39    Interlock := FALSE ,
40    Feedback := IO . AI_DeliveryPump2SpeedPv ,
41    ControlValueAuto_Cmd := Local . DeliveryPumpWorkDist . Out2 ) ;
42
43  Local . DeliveryPump3 ( _id := 'DeliveryPump3' , _owner := 'plc2' ,
44    HmiCmd := HMI . DeliveryPump3Cmd ,

```

---

 1.1.1.3 POU: Control
 

---

```

45      HmiPv := HMI . DeliveryPump3Pv ,
46      Interlock := FALSE ,
47      Feedback := IO . AI_DeliveryPump3SpeedPv ,
48      ControlValueAuto_Cmd := Local . DeliveryPumpWorkDist . Out3 ) ;
49

```

#### 1.1.1.4 POU: Initialize

```

1   PROGRAM Initialize
2   VAR
3   END_VAR
4

1
2   Hmi . DeliveryPumpPIDCmd . Tag := 'PID01' ;
3   Hmi . DeliveryPumpPIDCmd . Description := 'Regulator for controling pressure'
;
4   Hmi . DeliveryPumpPIDCmd . Auto := TRUE ;
5   Hmi . DeliveryPumpPIDCmd . EnableK := TRUE ;
6   Hmi . DeliveryPumpPIDCmd . EnableI := TRUE ;
7   Hmi . DeliveryPumpPIDCmd . EnableD := FALSE ;
8   Hmi . DeliveryPumpPIDCmd . K := 0.5 ;
9   Hmi . DeliveryPumpPIDCmd . Ti := 20.0 ;
10  Hmi . DeliveryPumpPIDCmd . Td := 0.0 ;
11  Hmi . DeliveryPumpPIDCmd . N := 10.0 ;
12  Hmi . DeliveryPumpPIDCmd . TrackingGain := 1.0 ;
13  Hmi . DeliveryPumpPIDCmd . u0 := 50.0 ;
14
15
16  HMI . DeliveryPump1Cmd . Tag := 'P01' ;
17  HMI . DeliveryPump1Cmd . Description := 'Delivery pump 1' ;
18  HMI . DeliveryPump1Cmd . Auto := TRUE ;
19  HMI . DeliveryPump1Cmd . ManualControlValue_Cmd := 50.0 ;
20  HMI . DeliveryPump1Cmd . AlarmControlFailedLimit := 10.0 ;
21  HMI . DeliveryPump1Cmd . AlarmControlFailed . Description := 'Failed to
control Delivery Pump 1' ;
22  HMI . DeliveryPump1Cmd . AlarmControlFailed . Delay . OnDelayLimit := 10.0 ;
23  HMI . DeliveryPump1Cmd . AlarmControlFailed . Delay . OffDelayLimit := 10.0 ;
24  HMI . DeliveryPump1Cmd . AlarmControlFailed . Acknowledged := TRUE ;
25
26  HMI . DeliveryPump2Cmd . Tag := 'P02' ;
27  HMI . DeliveryPump2Cmd . Description := 'Delivery pump 2' ;
28  HMI . DeliveryPump2Cmd . Auto := TRUE ;
29  HMI . DeliveryPump2Cmd . ManualControlValue_Cmd := 50.0 ;
30  HMI . DeliveryPump2Cmd . AlarmControlFailedLimit := 10.0 ;
31  HMI . DeliveryPump2Cmd . AlarmControlFailed . Description := 'Failed to
control Delivery Pump 2' ;
32  HMI . DeliveryPump2Cmd . AlarmControlFailed . Delay . OnDelayLimit := 10.0 ;
33  HMI . DeliveryPump2Cmd . AlarmControlFailed . Delay . OffDelayLimit := 10.0 ;
34  HMI . DeliveryPump2Cmd . AlarmControlFailed . Acknowledged := TRUE ;
35
36  HMI . DeliveryPump3Cmd . Tag := 'P03' ;
37  HMI . DeliveryPump3Cmd . Description := 'Delivery pump 2' ;
38  HMI . DeliveryPump3Cmd . Auto := TRUE ;

```

## 1.1.1.4 POU: Initialize

```

39   HMI . DeliveryPump3Cmd . ManualControlValue_Cmd := 50.0 ;
40   HMI . DeliveryPump3Cmd . AlarmControlFailedLimit := 10.0 ;
41   HMI . DeliveryPump3Cmd . AlarmControlFailed . Description := 'Failed to
control Delivery Pump 3' ;
42   HMI . DeliveryPump3Cmd . AlarmControlFailed . Delay . OnDelayLimit := 10.0 ;
43   HMI . DeliveryPump3Cmd . AlarmControlFailed . Delay . OffDelayLimit := 10.0 ;
44   HMI . DeliveryPump3Cmd . AlarmControlFailed . Acknowledged := TRUE ;
45
46   Hmi . TreatmentPressureCmd . Tag := 'PT02' ;
47   Hmi . TreatmentPressureCmd . Description := 'Treatment pressure' ;
48   Hmi . TreatmentPressureCmd . AlarmHysteresis := 3.0 ;
49   Hmi . TreatmentPressureCmd . AlarmHighHighLimit := 90.0 ;
50   Hmi . TreatmentPressureCmd . AlarmHighLimit := 80.0 ;
51   Hmi . TreatmentPressureCmd . AlarmLowLowLimit := 10.0 ;
52   Hmi . TreatmentPressureCmd . AlarmLowLimit := 20.0 ;
53   Hmi . TreatmentPressureCmd . AlarmHigh . Description := 'High Treatment
Pressure' ;
54   Hmi . TreatmentPressureCmd . AlarmHighHigh . Description := 'High high
Treatment Pressure' ;
55   Hmi . TreatmentPressureCmd . AlarmLow . Description := 'Low Treatment
Pressure' ;
56   Hmi . TreatmentPressureCmd . AlarmLowLow . Description := 'Low low Treatment
Pressure' ;
57   Hmi . TreatmentPressureCmd . AlarmHigh . Delay . OnDelayLimit := 1.0 ;
58   Hmi . TreatmentPressureCmd . AlarmHigh . Delay . OffDelayLimit := 1.0 ;
59   Hmi . TreatmentPressureCmd . AlarmHighHigh . Delay . OnDelayLimit := 1.0 ;
60   Hmi . TreatmentPressureCmd . AlarmHighHigh . Delay . OffDelayLimit := 1.0 ;
61   Hmi . TreatmentPressureCmd . AlarmLow . Delay . OnDelayLimit := 1.0 ;
62   Hmi . TreatmentPressureCmd . AlarmLow . Delay . OffDelayLimit := 1.0 ;
63   Hmi . TreatmentPressureCmd . AlarmLowLow . Delay . OnDelayLimit := 1.0 ;
64   Hmi . TreatmentPressureCmd . AlarmLowLow . Delay . OffDelayLimit := 1.0 ;
65   Hmi . TreatmentPressureCmd . InputScaling . InputUnit := 'na' ;
66   Hmi . TreatmentPressureCmd . InputScaling . OutputUnit := 'bar' ;
67   Hmi . TreatmentPressureCmd . InputScaling . InputMax_Cmd := 100.0 ;
68   Hmi . TreatmentPressureCmd . InputScaling . InputMin_Cmd := 0.0 ;
69   Hmi . TreatmentPressureCmd . InputScaling . OutputMax_Cmd := 100.0 ;
70   Hmi . TreatmentPressureCmd . InputScaling . OutputMin_Cmd := 0.0 ;
71
72   Hmi . ConsumerPressureCmd . Tag := 'PT03' ;
73   Hmi . ConsumerPressureCmd . Description := 'Consumer pressure' ;
74   Hmi . ConsumerPressureCmd . AlarmHysteresis := 3.0 ;
75   Hmi . ConsumerPressureCmd . AlarmHighHighLimit := 90.0 ;
76   Hmi . ConsumerPressureCmd . AlarmHighLimit := 80.0 ;
77   Hmi . ConsumerPressureCmd . AlarmLowLowLimit := 10.0 ;
78   Hmi . ConsumerPressureCmd . AlarmLowLimit := 20.0 ;
79   Hmi . ConsumerPressureCmd . AlarmHigh . Description := 'High Consumer
Pressure' ;
80   Hmi . ConsumerPressureCmd . AlarmHighHigh . Description := 'High high Consumer
Pressure' ;
81   Hmi . ConsumerPressureCmd . AlarmLow . Description := 'Low Consumer Pressure' ;
82   Hmi . ConsumerPressureCmd . AlarmLowLow . Description := 'Low low Consumer
Pressure' ;
83   Hmi . ConsumerPressureCmd . AlarmHigh . Delay . OnDelayLimit := 1.0 ;
84   Hmi . ConsumerPressureCmd . AlarmHigh . Delay . OffDelayLimit := 1.0 ;
85   Hmi . ConsumerPressureCmd . AlarmHighHigh . Delay . OnDelayLimit := 1.0 ;
86   Hmi . ConsumerPressureCmd . AlarmHighHigh . Delay . OffDelayLimit := 1.0 ;

```

---

#### 1.1.1.4 POU: Initialize

---

```

87  Hmi . ConsumerPressureCmd . AlarmLow . Delay . OnDelayLimit := 1.0 ;
88  Hmi . ConsumerPressureCmd . AlarmLow . Delay . OffDelayLimit := 1.0 ;
89  Hmi . ConsumerPressureCmd . AlarmLowLow . Delay . OnDelayLimit := 1.0 ;
90  Hmi . ConsumerPressureCmd . AlarmLowLow . Delay . OffDelayLimit := 1.0 ;
91  Hmi . ConsumerPressureCmd . InputScaling . InputUnit := 'na' ;
92  Hmi . ConsumerPressureCmd . InputScaling . OutputUnit := 'bar' ;
93  Hmi . ConsumerPressureCmd . InputScaling . InputMax_Cmd := 100.0 ;
94  Hmi . ConsumerPressureCmd . InputScaling . InputMin_Cmd := 0.0 ;
95  Hmi . ConsumerPressureCmd . InputScaling . OutputMax_Cmd := 100.0 ;
96  Hmi . ConsumerPressureCmd . InputScaling . OutputMin_Cmd := 0.0 ;
97

```

#### 1.1.1.5 POU: Inputs

```

1  PROGRAM Inputs
2  VAR
3
4  END_VAR
5

```

---

```

1  Local . TreatmentPressure (_id := 'TreatmentPressure' , _owner := 'plc2' ,
2      RawInput := IO . AI_TreatmentPressurePv ,
3      HmiCmd := HMI . TreatmentPressureCmd ,
4      HmiPv := HMI . TreatmentPressurePv ) ;
5
6
7  Local . ConsumerPressure (_id := 'ConsumerPressure' , _owner := 'plc2' ,
8      RawInput := IO . AI_ConsumerPressurePv ,
9      HmiCmd := HMI . ConsumerPressureCmd ,
10     HmiPv := HMI . ConsumerPressurePv ) ;
11

```

#### 1.1.1.6 POU: Outputs

```

1  PROGRAM Outputs
2  VAR
3  END_VAR
4

```

---

```

1  IO . AO_DeliveryPump1ControlValue := Local . DeliveryPump1 . ControlValue ;
2  IO . AO_DeliveryPump2ControlValue := Local . DeliveryPump2 . ControlValue ;
3  IO . AO_DeliveryPump3ControlValue := Local . DeliveryPump3 . ControlValue ;
4
5

```

---

1.1.1.7 POU: PLC\_PRG

---

### 1.1.1.7 POU: PLC\_PRG

```
1   PROGRAM PLC_PRG
2   VAR  RETAIN
3       _firstScanAfterDownload : BOOL  :=  TRUE ;
4   END_VAR
5   VAR
6       _counter : INT ;
7       _lastRun : STRING ;
8   END_VAR
9

1
2   // Signaling that the PLC is alive
3   Hmi . AliveCounterPlc2Pv . _tagId  :=  'AliveCounterPlc2Pv' ;
4   Hmi . AliveCounterPlc2Pv . _type  :=  'AliveHmi' ;
5   Hmi . AliveCounterPlc2Pv . _owner  :=  'plc2' ;
6   HMI . AliveCounterPlc2Pv . CounterPv  :=  HMI . AliveCounterPlc2Pv . CounterPv +
7   1 ;
8
9   _counter  :=  STATUS . OpcUaMqttLinkStatus . AliveCounterPv ;
10  _lastRun  :=  STATUS . OpcUaMqttLinkStatus . LastRun ;
11
12  IF  _firstScanAfterDownload  THEN
13      Initialize () ;
14  END_IF
15
16  Inputs () ;
17  Alarms () ;
18  Control () ;
19  Outputs () ;
20  Simulation () ;
21
22
23
24  // Always last
25  _firstScanAfterDownload  :=  FALSE ;
26
27
28
```

---

---

 1.1.1.8 POU: RestartOpcUaMqttLinkOnDownload
 

---

```

1   FUNCTION RestartOpcUaMqttLinkOnDownload : DWORD
2   VAR_IN_OUT
3       EventPrm : CmpApp . EVTPARAM_CmpApp ;
4   END_VAR
5   VAR
6   END_VAR
7

1   /*
2   This function is called when a download is about to happen.
3   This will force the Opc Ua - MQTT link to restart.
4   The restart will take 10 seconds, allowing e!Cockpit to download and start
5   before the link starts.
6   */
7   Status . OpcUaMqttLinkStatus . Restart_Cmd := TRUE ;

```

---

## 1.1.1.9 POU: RestartOpcUaMqttLinkOnOnlineChange

```

1   FUNCTION RestartOpcUaMqttLinkOnOnlineChange : DWORD
2   VAR_IN_OUT
3       EventPrm : CmpApp . EVTPARAM_CmpApp ;
4   END_VAR
5   VAR
6   END_VAR
7

1   /*
2   This function is called when a online change is about to happen.
3   This will force the Opc Ua - MQTT link to restart.
4   The restart will take 10 seconds, allowing e!Cockpit to download and start
5   before the link starts.
6   */
7   Status . OpcUaMqttLinkStatus . Restart_Cmd := TRUE ;

```

---

## 1.1.1.10 POU: Simulation

```

1   PROGRAM Simulation
2   VAR
3       _simDelay : ARRAY [ 0 .. 20 ] OF REAL ;
4       _iprocess : INT := 13 ; // Endret fra 19 til 13. -Peder
5       _idelay : INT := 0 ;
6       pressureDisturbance : ARRAY [ 0 .. 29 ] OF REAL := [ 0.9 ,
7           0.9 ,
8           0.85 ,
9           0.85 ,
10          0.8 ,
11          0.8 ,
12          0.75 ,
13          0.75 ,
14          0.7 ,

```

## 1.1.1.10 POU: Simulation

```

15      0.75 ,
16      0.75 ,
17      0.8 ,
18      0.8 ,
19      0.85 ,
20      0.85 ,
21      0.9 ,
22      0.9 ,
23      0.95 ,
24      0.95 ,
25      1.0 ,
26      1.0 ,
27      1.05 ,
28      1.05 ,
29      1.1 ,
30      1.1 ,
31      1.05 ,
32      1.05 ,
33      1.0 ,
34      0.95 ,
35      0.9 ] ;
36      timeNow : TIME ;
37      secondsNow : LWORD ;
38      fakeHour : LWORD ;
39      currentFakeHour : LWORD ;
40  END_VAR
41

```

```

1   IO . AI_DeliveryPump1SpeedPv := IO . AO_DeliveryPump1ControlValue ;
2   IO . AI_DeliveryPump2SpeedPv := IO . AO_DeliveryPump2ControlValue ;
3   IO . AI_DeliveryPump3SpeedPv := IO . AO_DeliveryPump3ControlValue ;
4
5   timeNow := TIME () ;
6   secondsNow := ( TIME_TO_LWORD ( timeNow ) / 1000 ) MOD 60 ;
7   fakeHour := secondsNow / 2 ;
8   SIM . Disturbance := pressureDisturbance [ fakeHour ] ;
9   Sim . Output := Local . DeliveryPumpPID . Output ;
10  Sim . Ref := HMI . DeliveryPumpPIDPv . Reference_Pv ;
11
12  SIM . Process_y := process ( u := _idelay MOD 19 ] ,
13      Ts := 0.1 , y := SIM . Process_Y , yMin := 0 , yMax := 8.0 , T := 5 ,
14      Kp := 1.0 ) ;
15  _idelay := _idelay + 1 ;
16  _simDelay [ _iprocess MOD 19 ] := ( ( IO . AI_DeliveryPump1SpeedPv + IO .
17  AI_DeliveryPump2SpeedPv + IO . AI_DeliveryPump3SpeedPv ) / 10 ) ;
18  _iprocess := _iprocess + 1 ;
19
20  SIM . Process := SIM . Process_y - SIM . Disturbance ;
21
22  IO . AI_ConsumerPressurePv := SIM . Process ;
23  IO . AI_TreatmentPressurePv := 5.0 ;
24

```

---

1.1.1.11 Symbol Configuration: Symbols

---

### 1.1.1.11 Symbol Configuration: Symbols

#### 1.1.1.12 Task Configuration: Task configuration

Max. number of tasks: 15  
Max. number of cyclic tasks: 15  
Max. number of freewheeling tasks: 15  
Max. number of event tasks: 15  
Max. number of external event tasks: 8  
Max. number of status tasks: 15

##### System Events:

1. PrepareDownload (Called before application download. Context=Communication task. Debugging=Disabled), active: RestartOpcUaMqttLinkOnDownload
2. PrepareOnlineChange (Called before application online change. Context=Communication task. Debugging=Disabled), active: RestartOpcUaMqttLinkOnOnlineChange

#### 1.1.1.12.1 Task: PLC\_Task

Priority: 10  
Type: Cyclic  
Interval: 100 Unit: ms  
Watchdog: Active  
Watchdog Time: 90 Unit: ms  
Watchdog Sensitivity: 1  
POUs: PLC\_PRG

#### 1.1.1.12.1.1 Program call: PLC\_PRG

#### 1.1.1.12.2 Task: TrendRecordingTask

Priority: 15  
Type: Cyclic  
Interval: 100 Unit: ms  
Watchdog: Inactive  
POUs: VisuTrendStorageAccess.GlobalInstances.g\_TrendRecordingManager.CyclicCall

---

### 1.1.1.12.2.1 Program call: VisuTrendStorageAccess.GlobalInstances.g\_T

#### 1.1.1.12.3 Task: VISU\_TASK

Priority: 15  
 Type: Cyclic  
 Interval: 100 Unit: ms  
 Watchdog: Inactive  
 POUs: VisuElems.Visu\_Prg

#### 1.1.1.12.3.1 Program call: VisuElems.Visu\_Prg

#### 1.1.1.13 Persistent Variables: PersistentVars

```

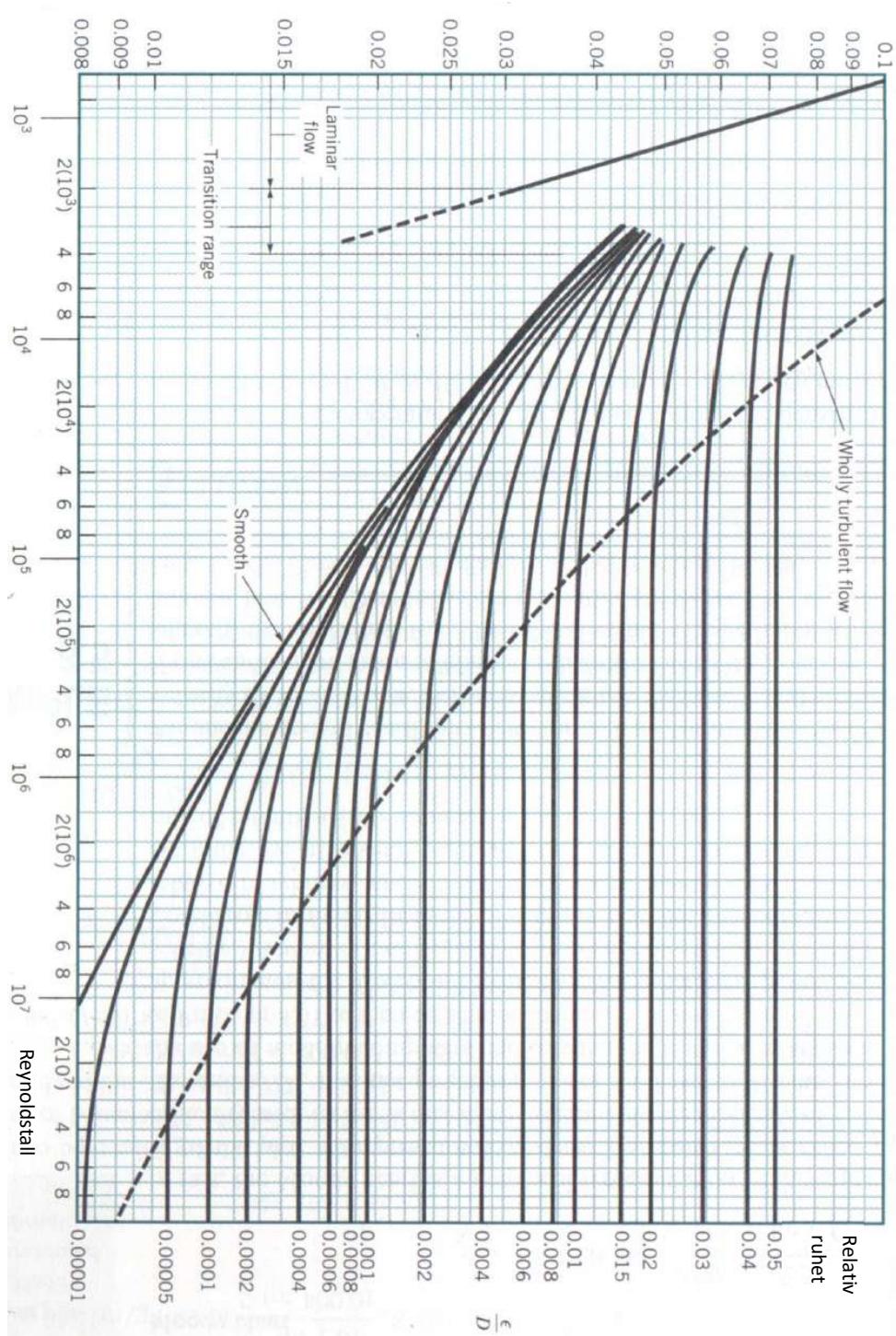
1   {attribute 'qualified_only'}
2   VAR_GLOBAL PERSISTENT RETAIN
3       // Generated instance path of persistent variable
4       HMI . AliveCounterPlc2Pv : AliveHmiPv ;
5       // Generated instance path of persistent variable
6       HMI . DeliveryPump1Cmd : ControlAnalogHmiCmd ;
7       // Generated instance path of persistent variable
8       HMI . DeliveryPump1Pv : ControlAnalogHmiPv ;
9       // Generated instance path of persistent variable
10      HMI . DeliveryPump2Cmd : ControlAnalogHmiCmd ;
11      // Generated instance path of persistent variable
12      HMI . DeliveryPump2Pv : ControlAnalogHmiPv ;
13      // Generated instance path of persistent variable
14      HMI . DeliveryPump3Cmd : ControlAnalogHmiCmd ;
15      // Generated instance path of persistent variable
16      HMI . DeliveryPump3Pv : ControlAnalogHmiPv ;
17      // Generated instance path of persistent variable
18      HMI . DeliveryPumpPIDCmd : PIDHmiCmd ;
19      // Generated instance path of persistent variable
20      HMI . DeliveryPumpPIDPv : PIDHmiPv ;
21      // Generated instance path of persistent variable
22      HMI . DeliveryPumpWorkDistPv : WorkDistributorHmiPv ;
23      // Generated instance path of persistent variable
24      HMI . TreatmentPressureCmd : SignalAnalogHmiCmd ;
25      // Generated instance path of persistent variable
26      HMI . TreatmentPressurePv : SignalAnalogHmiPv ;
27      // Generated instance path of persistent variable
28      HMI . ConsumerPressureCmd : SignalAnalogHmiCmd ;
29      // Generated instance path of persistent variable
30      HMI . ConsumerPressurePv : SignalAnalogHmiPv ;
31  END_VAR
32

```

## **B Vedlegg simulerete verdier**

## B.1 Moody's diagram

**Moodys diagram til bestemmelse av friksjonsfaktoren:**



## B.2 Ruhet for rør og tapskoeffisient for enkeltmotstander

Ruhet for visse typer rør (typiske verdier)

Materiale	Tilstand	Ruhet $\epsilon$ ( mm)
Messing, kopper	Slette og rene	0.03
Aluminium,	"	0.03
Plast, glass	"	0.001 – 0.01
Stål	Nye	0.05 – 0.10
	Rustne	0.20 – 0.30
	Forsinkede	0.13
Støpejern	Nye	0.25
	Rustne	1.0 – 1.5

Tapskoeffisienter for visse typer enkeltmotstander

Komponent	Karakteristikk	Tapskoeffisient $\zeta$
Utløp	skarpkantet	1.0
Utløp	avrundet	1.0
Innløp	skarpkantet	0.5
	avrundet	0.04
Bend 90 °	skarpkantet	1
Bend 90 °	liten krumning	0.2
Kraner kuleventil	åpen	0.05
	1/3 lukket	5.5
	2/3 lukket	200

## **C Vedlegg lekkasjedeteksjon**

### **C.1 Datablad strømningsmåler**



Badger Meter Europa

**Badger Meter Europa  
GmbH**  
Nürtinger Str. 76  
72639 Neuffen (Germany)  
Tel. +49-7025-9208-0  
Fax +49-7025-9208-15  
[www.badgermeter.de](http://www.badgermeter.de)

## Model TFX Ultra®

Clamp-on ultrasonic flow and energy meters for liquids



### Description

TFX Ultra® ultrasonic flow and energy meters clamp onto the outside of pipes and do not contact the internal liquid. The technology has inherent advantages over alternate devices including: low-cost installation, no pressure head loss, no moving parts to maintain or replace, no fluid compatibility issue, and a large, bi-directional measuring range that ensures reliable readings even at very low and high flow rates. TFX Ultra® is available in a variety of configurations that permit the user to select a meter with features suitable to meet particular application requirements.

The TFX Ultra® is available in two versions: a stand-alone flow meter, and an energy flow meter used in conjunction with dual clamp-on RTDs. The energy flow meter measures energy usage in BTU, MBTU, MMBTU, Tons, kJ, kW, MW and is ideal for retrofit, chilled water and other HVAC applications.

### Features

- Reduced material cost: Clamp-on sensor eliminates the need for in-line flanges, pipe fittings, strainers and filters
- Reduced installation time: The TFX Ultra® may be installed and fully operational within minutes
- Reduced maintenance costs: With no moving parts, there is nothing on the TFX Ultra® to wear down – no repair kits or replacement parts are needed
- No need to shut down the process for installation or maintenance due to clamp-on-sensor design

### Applications

- May be used to measure clean liquids as well as those with small amounts of suspended solids or aeration (e.g. surface water, sewage).
- Bi-directional flow measurement system. Totalizer options include forward, reverse and net total.
- ModBus® RTU and BACnet® MS/TP over RS485; Ethernet connection includes BACnet®/IP™, EtherNet/IP™ and ModBus® TCP/IP protocols.
- Large, easy-to-read digital display.
- Rugged, aluminium enclosure ensures a long service life in harsh environments.
- Certified for hazardous area installation in North America and Europe.



## Technical data

Flow meter	
Liquid types	Most clean liquids or liquids containing small amounts of suspended solids or gas bubbles
Velocity range	Bi-directional to greater than 12 m/s (40 ft/s)
Flow accuracy	DTTN/DTTH/DTTL: $\pm 1\%$ of reading or $\pm 0.003 \text{ m/s}$ (0.01 ft/s), whichever is greater DTTS/DTTC: DN 25 (1") and larger - $\pm 1\%$ of reading or $\pm 0.012 \text{ m/s}$ (0.04 ft/s), whichever is greater DTTS/DTTC: DN 19 (1/2") and smaller - 1% of full scale (refer to page dimensions)
Temperature accuracy (energy meters only)	Option A: 0 to + 50 °C (+ 32 to + 122 °F); absolute: 0,12 °C (0.22 °F) difference: 0,05 °C (0.09 °F) Option B: 0 to + 100 °C (+ 32 to + 212 °F); absolute: 0,25 °C (0.45 °F) difference: 0,1 °C (0.18 °F) Option C: -40 to + 175 to (-40 °C + 350 °F); absolute: 0,6 °C (1.1 °F) difference: 0,25 °C (0.45 °F) Option D: -20 to + 30 °C (-4 to + 86 °F); absolute: 0,12 °C (0.22 °F) difference: 0,05 °C (0.09 °F)
Sensitivity	<b>Flow:</b> 0,0003 m/s (0.001 ft/s) <b>Temperature:</b> option A: 0,012 °C (0.03 °F); option B: 0,025 °C (0.05 °F); option C: 0,06 °C (0.1 °F); option D: 0,012 °C (0.03 °F)
Repeatability	0,5 % of reading
Approvals	General safety (all models): UL® 61010-1, CSA® C22.2 No. 61010-1; (power options A and D only) EN 61010-1 Hazardous location (power supply options A and D only): class I division 2 groups C, D, T4; class II, division 2, groups F, G, T4; class III division 2 for US/CAN; ATEX II 2 G Ex nA II T4: UL® 1604, CSA® 22.2 No. 213, EN 60079-0 and EN 60079-15 CE: EN61326-1:2006 on meter systems with integral flow sensors, sensors constructed with twin axial cable [all sensors with cables 30 m (100 ft) and shorter] or remote sensor with conduit.
Transmitter	
Power supply	AC: 95-264 VAC 47-63 Hz @ 17 VA max. or 20-28 VAC 47-63 Hz @ 0,35 A max. DC: 10-28 VDC @ 5 W max. Protection: Auto resettable fuse, reverse polarity and transient suppression
Display	Two line LCD, LED backlit; top row 18 mm (0.7") height, 7-segment; bottom row 9 mm (0.35") height, 14-segment Icons: RUN, PROGRAM, RELAY1, RELAY2 Flow rate indication: 8-digit positive, 7-digit negative max.; auto decimal, lead zero blanking Flow accumulator (totalizer): 8-digit positive, 7-digit negative max. (reset via keypad press, ULTRALINK™, network command or momentary contact closure)
Enclosure	Type 4 (IP65) construction: Powder-coated aluminum, polycarbonate, stainless steel, polyurethane, nickel-plated steel mounting brackets Size (electronic enclosure only): 152 mm W x 112 mm H x 56 mm D (6.0" W x 4.4" H x 2.2" D) Conduit holes: (2) 1/2" NPT female; (1) 1/4" NPT female; optional cable gland kit
Temperature	-40 °C to + 85 °C (-40 °F to + 185 °F)
Configuration	Via optional keypad or PC running ULTRALINK™ software (Note: Not all configuration parameters are available from the keypad – i.e. flow and temperature calibration and advanced filter settings)
Engineering units	<b>Flow meter:</b> Feet, gallons, cubic feet, million gallons, barrels (liquid and oil), acre-feet, lbs., meters, cubic meters, liters, million liters, kg <b>Energy meter:</b> BTU, MBTU, MMBTU, tons, kJ, kW, MW and the flow meter list from above
Inputs/outputs	USB 2.0: for connection of a PC running ULTRALINK™ configuration utility RS485: ModBus® RTU command set; optional BACnet® MS/TP 10/100 Base-T: RJ45, communication via ModBus® TCP/IP, EtherNet/IP™ or BACnet®/IP 4-20mA: 12-bit, internal power, can span negative to positive flow/energy rates <b>Energy meter model only:</b> Total pulse option: Opto isolated open collector transistor <b>Flow meter model only:</b> 0-1000 Hz: open-collector, 12-bit, can span negative to positive rates; square-wave or turbine meter simulation outputs Two alarm outputs: Open-collector, configure as rate alarm, signal strength alarm or totalizer pulse
Sensors	
Sensor ratings	DTTN/DTTC/DTTL: NEMA 6* (IP67), CPVC, Ultem®, nylon cord grip, PVC cable jacket; -40 to + 120 °C (-40 to + 250 °F) DTTN/DTTL: NEMA 6P* (IP68) option, CPVC, Ultem®, nylon cord grip, polyethylene cable jacket; -40 to + 120 °C (-40 to + 250 °F) DTTH: NEMA 6* (IP67), PTFE, Vespel®, nickel-plated brass cord grip, PFA cable jacket; -40 to + 175 °C (-40 to + 350 °F) DTTS: NEMA 6* (IP67), PVC, Ultem®, nylon cord grip, PVC cable jacket; -40 to + 85 °C (-40 to + 185 °F) *NEMA 6 units: To a depth of 1 m (3 ft) for 30 days max. NEMA 6P units: To a depth of 30 m (100 ft) seawater equivalent density indefinitely.
Frequency	DTTS/DTTC: 2 MHz DTTN/DTTH: 1 MHz DTTL: 500 KHz
Cables	RG59 Coaxial, 75 ohms or twinaxial, 78 ohms (optional armored conduit)

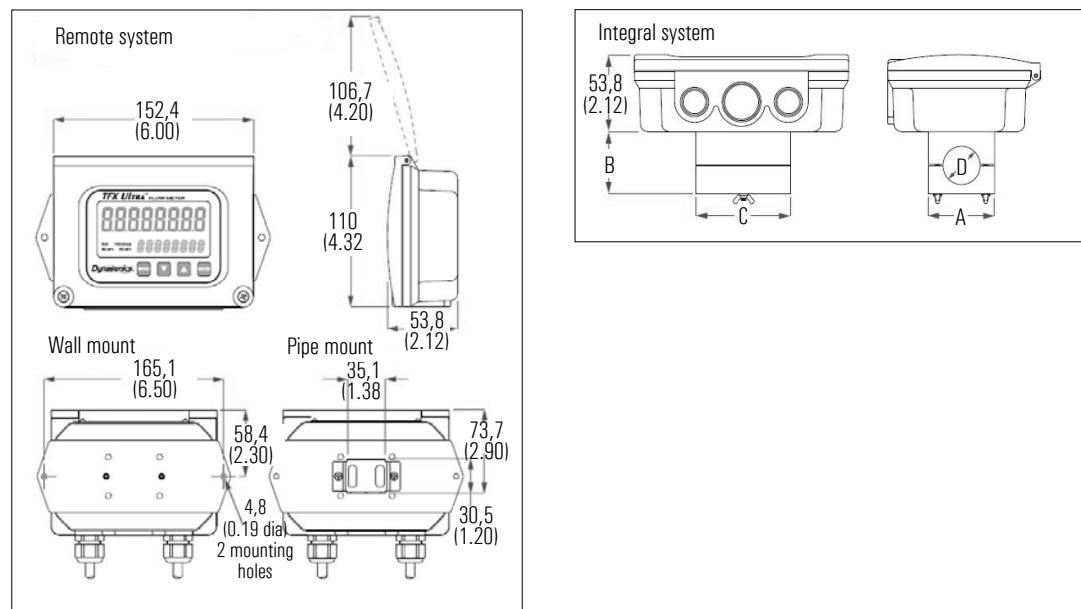


## Technical data (cont.)

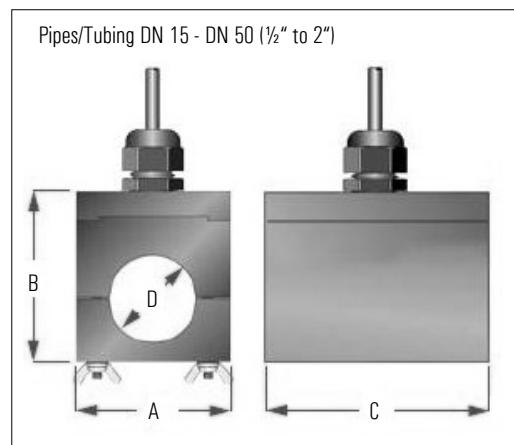
Sensors (cont.)	
Cable length	300 m (990 ft) max. in 3 m (10 ft) increments; submersible conduit limited to 30 m (100 ft)
RTDs	<b>Energy meters only:</b> Platinum 385, 1,000 ohms, 3-wire; PVC jacket cable
Installation	DTTN (-N option) /DTTS/DTTH/DTTC: General and hazardous location (see installation compliance above) DTTN sensor and IS barrier (-F option): Class I Div. 1, groups C&D T5 intrinsically safe Ex ia; CSA® C22.2 No.'s 142 & 157; UL® 913 & 916
Software utilities	
*ULTRALINK™	Utilized to configure, calibrate and troubleshoot flow and energy meters. Connection via USB A/B cable; software is compatible with Windows® 2000, Windows® XP, Windows Vista® and Windows® 7
*EnergyLink	Utilized to monitor a network of flow and energy meters. Connection via RS485. Operates within Microsoft Excel® 2003, Microsoft Excel® 2007, Microsoft Excel® 2010 (32-bit O.S. only).

\*Software available at no additional cost.

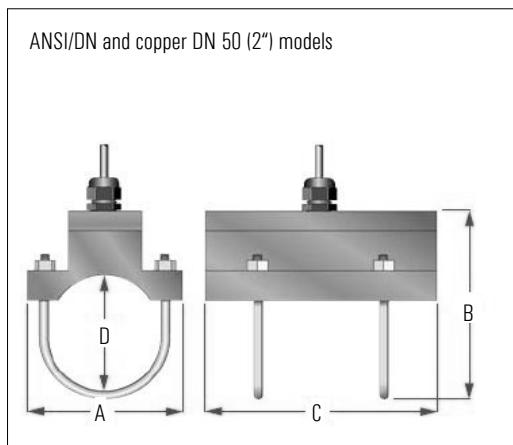
## Dimensions in mm (inches)



## DTTS/DTTC/ sensor



## DTTS/DTTC U-bolt connection



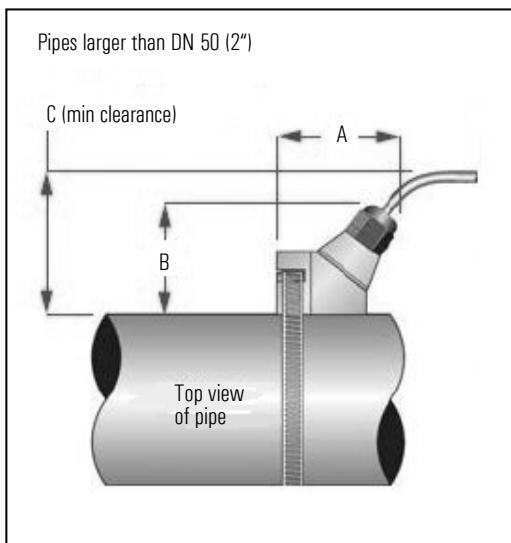


**DTTS /DTTC sensor**  
**Dimensions in mm (inches)**

Pipe size	Pipe material	A	B	C	D	Measuring range
$\frac{1}{2}$ "	ANSI/DN	62,5 (2.46)	59,9 (2.36)	67,6 (2.66)	21,3 (0.84)	8 - 144 l/min (2 - 38 gal/min)
	Copper	62,5 (2.46)	59,9 (2.36)	84,6 (3.33)	15,9 (0.63)	7 - 102 l/min (1.8 - 27 gal/min)
	Tubing	62,5 (2.46)	57,9 (2.28)	94,5 (3.72)	12,7 (0.50)	6 - 68 l/min (1.5 - 18 gal/min)
$\frac{3}{4}$ "	ANSI/DN	62,5 (2.46)	65,3 (2.57)	67,6 (2.66)	26,7 (1.05)	10 - 250 l/min (2.75 - 66 gal/min)
	Copper	62,5 (2.46)	63,5 (2.50)	90,4 (3.56)	22,2 (0.88)	10 - 204 l/min (2.5 - 54 gal/min)
	Tubing	62,5 (2.46)	63,5 (2.50)	90,4 (3.56)	19,0 (0.75)	10 - 170 l/min (2.5 - 45 gal/min)
1"	ANSI/DN	62,5 (2.46)	74,2 (2.92)	72,6 (2.86)	33,4 (1.32)	13 - 409 l/min (3.5 - 108 gal/min)
	Copper	62,5 (2.46)	72,9 (2.87)	96,5 (3.80)	28,6 (1.13)	13 - 320 l/min (3.5 - 95 gal/min)
	Tubing	62,5 (2.46)	69,9 (2.75)	96,5 (3.80)	25,4 (1.00)	13 - 320 l/min (3.5 - 85 gal/min)
$1\frac{1}{4}$ "	ANSI/DN	71,0 (2.80)	80,8 (3.18)	79,8 (3.14)	42,2 (1.66)	19 - 704 l/min (5 - 186 gal/min)
	Copper	62,5 (2.46)	76,2 (3.00)	102,6 (4.04)	34,9 (1.38)	17 - 575 l/min (4.5 - 152 gal/min)
	Tubing	62,5 (2.46)	76,2 (3.00)	102,6 (4.04)	31,8 (1.25)	15 - 514 l/min (4 - 136 gal/min)
$1\frac{1}{2}$ "	ANSI/DN	76,7 (3.02)	86,9 (3.42)	84,6 (3.33)	48,3 (1.90)	23 - 946 l/min (6 - 250 gal/min)
	Copper	68,8 (2.71)	72,6 (2.86)	108,7 (4.28)	41,3 (1.63)	19 - 814 l/min (5 - 215 gal/min)
	Tubing	68,8 (2.71)	84,1 (3.31)	108,7 (4.28)	38,1 (1.50)	19 - 757 l/min (5 - 200 gal/min)
2"	ANSI/DN	94,0 (3.70)	86,9* (3.42)	139,7 (5.50)	60,3* (2.38)	30 - 1590 l/min (8 - 420 gal/min)
	Copper	94,0 (3.70)	85,9* (3.38)	139,7 (5.50)	54,0* (2.13)	30 - 1419 l/min (8 - 375 gal/min)
	Tubing	81,5 (3.21)	98,0 (3.85)	120,7 (4.75)	50,8 (2.00)	30 - 1381 l/min (8 - 365 gal/min)

\* Varies due to U-bolt configuration

**DTTN/DTTH/DTTL sensor\***



	A	B	C
DTTN	74,9 (2.95)	69,8 (2.75)	76,2 (3.00)
DTTH	74,9 (2.95)	69,8 (2.75)	76,2 (3.00)
DTTL	86,4 (3.40)	74,7 (2.94)	81,3 (3.20)

\*One pair of sensors are supplied and are required for measurements

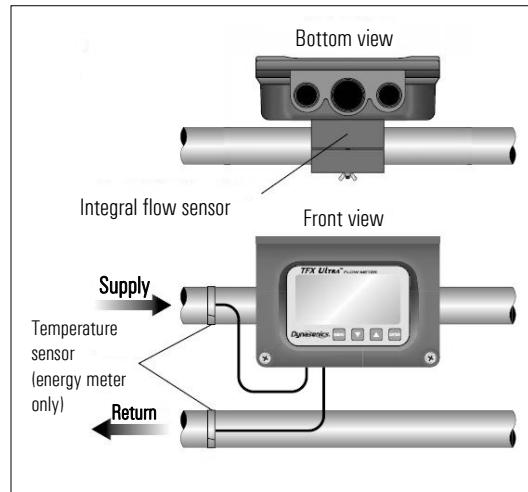


### Meter with integral flow sensor

For pipe/tubing sizes of DN 50 (2") and lower, TFX Ultra® is available with a clamp-on sensor mounted and wired directly to the flow meter display/electronics enclosure. This design provides a convenient installation in areas where the user requires local indication. PVC constructed sensors are rated to 85 °C (185 °F) and CPVC are rated to 120 °C (250 °F).

#### Common features:

- Rate-total backlit display
- 4-20 mA output
- 0-1,000 Hz rate pulse and dual alarm outputs (flow meter model only)
- USB programming port
- RS485 ModBus® network connection
- Remote totalizer reset



### Ordering matrix

DTFX	<input type="checkbox"/>	-	<input type="checkbox"/>	<input type="checkbox"/>	-	<input type="checkbox"/>	<input type="checkbox"/>	-	<input type="checkbox"/>	-	<input type="checkbox"/>	-	<input type="checkbox"/>
<b>Model</b>													<b>Language</b> (leave blank for English)
B) Flow meter model													F) French
E) Energy meter model													S) Spanish
<b>Pipe size / measurement range</b>													<b>Options</b>
A) ½" ANSI pipe (DN 15)													N) None
B) ¾" ANSI pipe (DN 20)													C) 4-pin (male)
C) 1" ANSI pipe (DN 25)													Brad Harrison®
D) 1-½" ANSI pipe (DN 32)													Micro-Change®
E) 1-½" ANSI pipe (DN 40)													(available for D/C power only)
F) 2" ANSI pipe (DN 50)													A) Cable gland kit
G) ½" Copper tube													<b>Area approvals</b>
H) ¾" Copper													F) General safety, hazardous locations, and CE (see technical data)
I) 1" Copper tube													N) General safety (power supply C only)
J) 1-½" Copper tube													<b>Energy temperature range</b>
K) 1-½" Copper tube													N) None (select for flow meter model B)
L) 2" Copper tube													A) 0 to +50 °C (+32 to +122 °F)
M) ½" OD standard tube													B) 0 to +100 °C (+32 to +212 °F)
N) ¾" OD standard tube													C) -40 to +175 °C (-40 to +350 °F)
P) 1" OD standard tube													D) -20 to +30 °C (-4 to +85 °F)
Q) 1-½" OD standard tube													
R) 1-½" OD standard tube													
S) 2" OD standard tube													
<b>Advanced communications</b>													
E) 10/100 Base-T (EtherNet/IP™, BACnet®/IP, ModBus® TCP/IP)													
B) BACnet® MS/TP													
C) BACnet® MS/TP; 10/100 Base-T (EtherNet/IP™, BACnet®/IP, ModBus® TCP/IP)													
N) None													
P) Total pulse output (energy meter only)													
H) BACnet® MS/TP 76800 Baud													

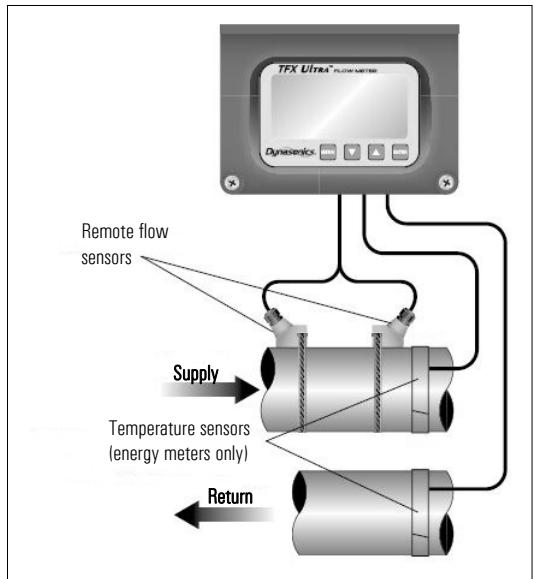


### Meter with remote flow sensor

TFX Ultra® is available with remote mounted sensors that permit separation of up to 300 m (990 ft). This design is utilized when pipes are located in areas that are not convenient for viewing, or on piping systems with severe vibration. PVC constructed sensors are rated to 85 °C (185 °F), CPVC to 120 °C (250 °F) and PTFE to 175 °C (350 °F).

#### Common features

- Rate-total backlit display
- 4-20 mA output
- 0-1,000 Hz rate pulse and dual alarm outputs (flow meter model only)
- USB programming port
- RS485 ModBus® network connection
- Remote totalizer reset



#### Ordering matrix

<b>DTFX</b>	<input type="checkbox"/>	-	<b>ZN</b>	<input type="checkbox"/>	-	<input type="checkbox"/>										
Model			Power supply			Keypad										
B) Flow meter model	<input type="checkbox"/>		A) A/C (95-264 VAC)	<input type="checkbox"/>		K) Keypad										
E) Energy meter model	<input type="checkbox"/>		C) A/C (20-28 VAC)	<input type="checkbox"/>		N) No keypad										
Remote sensor			D) D/C (10-28 VDC)	<input type="checkbox"/>												
Use with DTTN/DTTH/DTTL large pipe sensors (pipes larger than 2") or DTTS/DTTC small pipe sensors (pipes 1/2" - 2")																
Advanced communications																
E) 10/100 Base-T (EtherNet/IP™, BACnet®/IP, ModBus® TCP/IP)	<input type="checkbox"/>															
B) BACnet® MS/TP	<input type="checkbox"/>															
C) BACnet® MS/TP; 10/100 Base-T (EtherNet/IP™, BACnet®/IP, ModBus® TCP/IP)	<input type="checkbox"/>															
N) None	<input type="checkbox"/>															
P) Total pulse output (energy meter only)	<input type="checkbox"/>															
H) BACnet® MS/TP 76800 Baud	<input type="checkbox"/>															
Approvals																
F) General safety, hazardous locations, and CE (see technical data)	<input type="checkbox"/>															
N) General safety (power supply C only)	<input type="checkbox"/>															
Language (Leave blank for English)																
F) French	<input type="checkbox"/>															
S) Spain	<input type="checkbox"/>															
Options																
N) None	<input type="checkbox"/>															
C) 4-pin (male) Brad Harrison® Micro-Change® (available for D/C power only)	<input type="checkbox"/>															
A) Cable gland kit	<input type="checkbox"/>															
Energy temperature range																
N) None (select for flow meter model B)	<input type="checkbox"/>															
A) 0 to + 50 °C (+32 to +122 °F)	<input type="checkbox"/>															
B) 0 to + 100 °C (+32 to +212 °F)	<input type="checkbox"/>															
C) -40 to + 175 °C (-40 to + 350 °F)	<input type="checkbox"/>															
D) -20 to + 30 °C (-4 to + 85 °F)	<input type="checkbox"/>															



## Ordering matrix for flow sensor -

Pipes larger than DN 50 (2")

DTT	<input type="checkbox"/>	-	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	-	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	-	<input type="checkbox"/>
<b>Construction</b>											
N) Standard											
+ 120 °C (+ 250 °F)											
(CPVC, Ultem®)											
H) High temperature											
+ 175 °C (+ 350 °F)											
(PTFE, Vespel®)											
L) Large pipe – 500 KHz											
+ 120 °C (+ 250 °F)											
(CVPC, Ultem®)*											
<b>Cable length</b>											
020) 6 m (20 ft)											
050) 15 m (50 ft)											
100) 30 m (100 ft)											
<b>Conduit type</b>											
N) None											
A) Flexible armored											
S) Submersible											
(DTTN and DTTL only) limited to 30 m (100 ft)											
<b>Conduit length</b>											
(Standard construction; conduit length = cable length)											
000) None											
020) 6 m (20 ft)											
050) 15 m (50 ft)											
100) 30 m (100 ft)											
<b>Installation</b>											
N) General purpose											
F) Class I, Div. 1, Groups C & D (DTTN only)											

\* Recommended for pipe sizes larger than DN 600 (24")

## Ordering Matrix for flow sensor

Small pipes DN 12 to DN 50 (½" to 2")

DTT	<input type="checkbox"/>	-	<input type="checkbox"/>	-	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>				
<b>Model</b>											
S) Standard + 85 °C (+ 185 °F) (PVC, Ultem®)											
C) High temperature + 120 °C (+ 250 °F) (CPVC, Ultem®)											
<b>Nominal pipe size</b>											
D) ½"											
F) ¾ "											
G) 1"											
H) 1-¼"											
J) 1-½"											
L) 2"											
<b>Cable length</b>											
020) 6 m (20 ft)											
050) 15 m (50 ft)											
100) 30 m (100 ft)*											
<b>Conduit length</b>											
000) None											
020) 6 m (20 ft)											
050) 15 m (50 ft)											
100) 30 m (100 ft)*											
<b>Pipe type</b>											
P) ANSI pipe											
C) Copper pipe											
T) Rigid tubing											
<b>Conduit type</b>											
N) None											
A) Flexible armored											

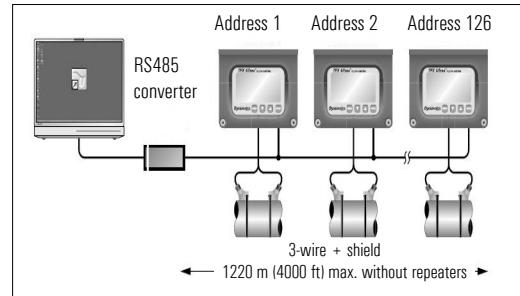
\* Maximum length: 300 m (990 ft) in 3 m (10 ft) increments.



## Network options

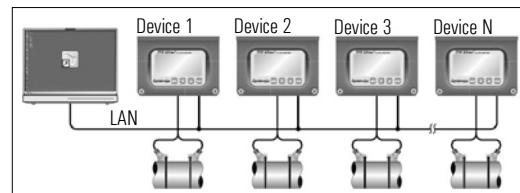
### TFX RS 485 Network

All TFX Ultra® meters come equipped with RS485 drivers and utilize a ModBus® RTU command set (data can be returned in single-precision, double-precision, integer or floating point values). Up to 126 TFX Ultra® products can be run on a single daisy-chain network and be individually queried for flow rate, positive flow accumulator, negative flow accumulator, supply temperature, return temperature and signal strength. Flow accumulators can be cleared at discrete addresses or globally. The RS485 network is also compatible with the EnergyLink, direct to Excel®, application detailed below.



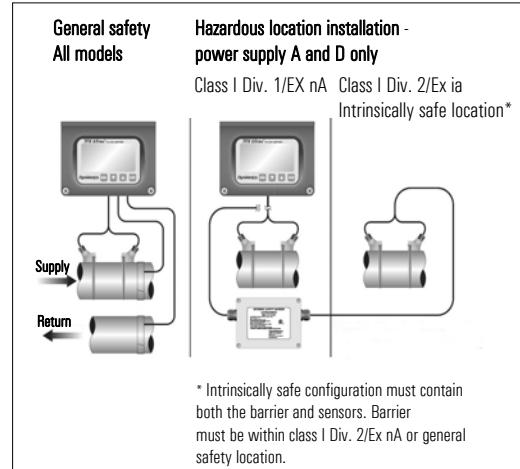
### TFX 10/100 Base-T network

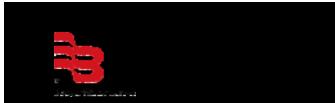
If equipped with the optional Ethernet communications module, the TFX Ultra® can be plugged into a LAN and queried for flow rate, positive flow accumulator, negative flow accumulator, supply temperature, return temperature and signal strength. The module contains ModBus® TCP/IP, EtherNet/IP™ and BACnet®/IP network compatibility.



## EnergyLink software

Operating from a standard, low-cost PC, EnergyLink software operates within Microsoft® Excel® and provides an efficient method of monitoring and archiving data from a network of TFX Ultra® energy meters. EnergyLink automatically backs up accumulated energy data every hour, day, month, quarter and year into convenient spreadsheet formats suitable for input into invoicing systems. The "Current Readings" screen provides real time measurements from all TFX Ultra® meters on the network (up to 126 meters can be connected on a single RS485 network). Data displayed includes: Location name, room number, TFX Ultra® address, a good/bad communication indicator, the time and date of the last reading, flow signal level, energy flow rate, energy accumulation, supply temperature and return temperature. The software can be configured to "auto run" should PC power be interrupted or the PC be turned off. The software can also be configured to reset the energy accumulators on all network meters at the beginning of every month or quarter.





## Parts

### RTD kits for integral and remote energy measurement meters

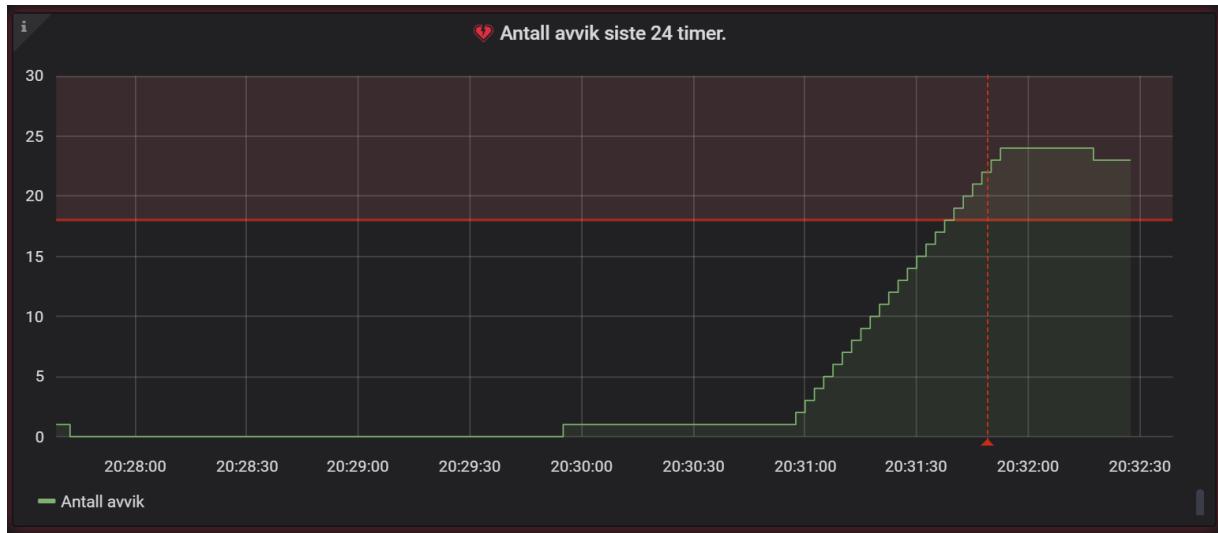
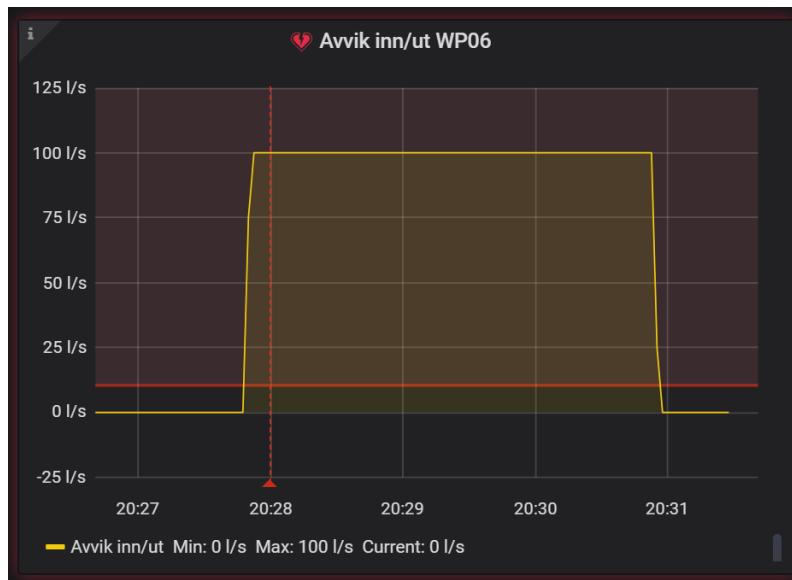
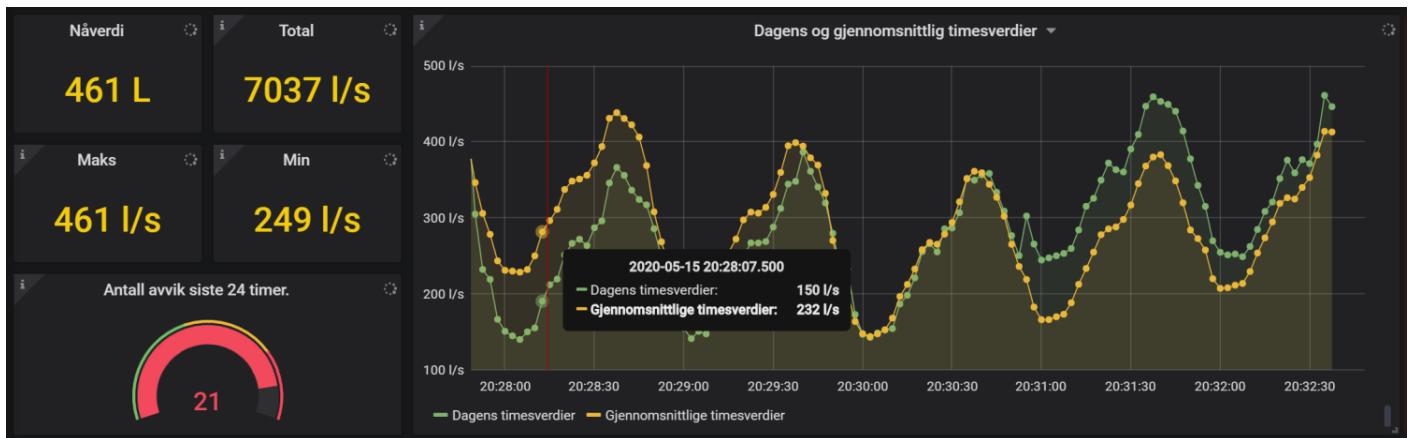
Part number	U.S. part number	Description
280003	D010-3000-120	RTD Kit <sup>1</sup> , clamp on, 130 °C, 1,000 ohms, 6 m (20 ft)
280032	D010-3000-121	RTD Kit <sup>1</sup> , clamp on, 130 °C, 1,000 ohms, 15 m (50 ft)
280033	D010-3000-122	RTD Kit <sup>1</sup> , clamp on, 130 °C, 1,000 ohms, 30 m (100 ft)
280034	D010-3000-123	RTD Kit <sup>1</sup> , clamp on, 200 °C, 1,000 ohms, 7,5 m (25 ft)
280035	D010-3000-124	RTD Kit <sup>1</sup> , clamp on, 200 °C, 1,000 ohms, 15 m (50 ft)
280036	D010-3000-125	RTD Kit <sup>1</sup> , clamp on, 200 °C, 1,000 ohms, 30 m (100 ft)
280037	D010-3000-200	Insertion RTD Kit <sup>2</sup> , 3", 1/4" O.D., 260 °C, 1,000 ohms, 6 m (20 ft)
280038	D010-3000-201	Insertion RTD Kit <sup>2</sup> , 3", 1/4" O.D., 260 °C, 1,000 ohms, 15 m (50 ft)
280039	D010-3000-202	Insertion RTD Kit <sup>2</sup> , 3", 1/4" O.D., 260 °C, 1,000 ohms, 30 m (100 ft)

<sup>1</sup>RTD kits include: 2 RTDs, heat sink compound and installation tape

<sup>2</sup>Insertion RTD kits include a set of 2 RTDs



## C.2 Resultat lekkasjedeteksjon

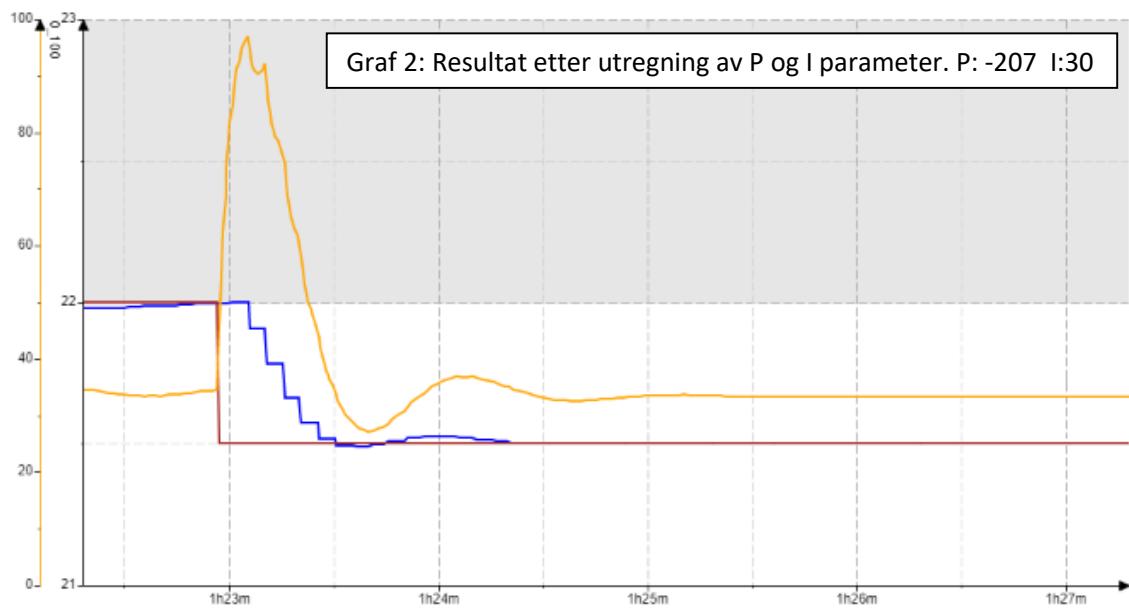
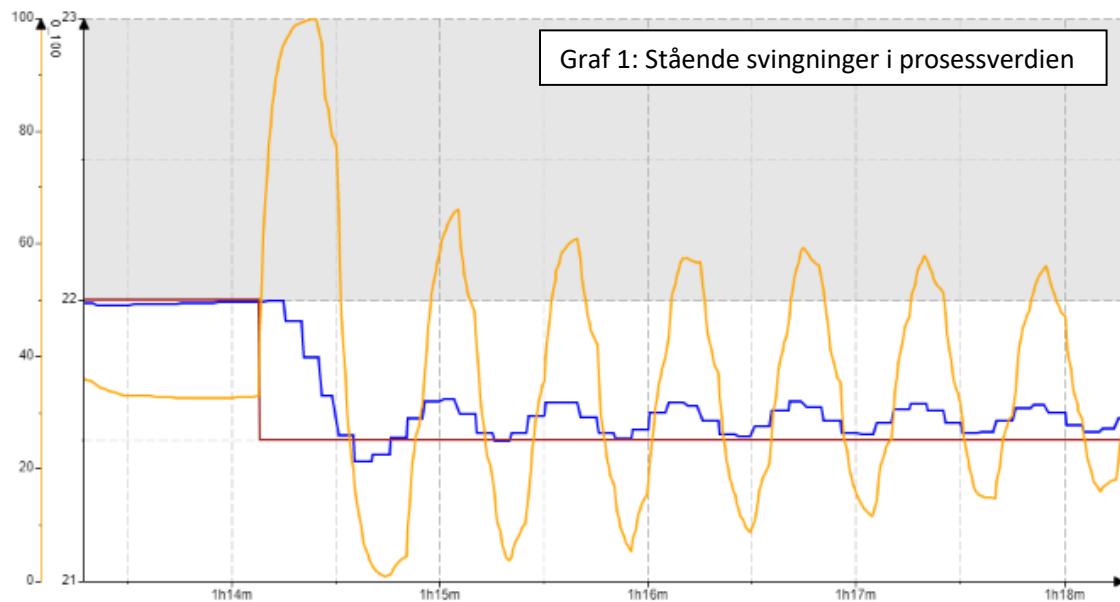
**Figur 1: Lekkasjedeteksjon basert på historisk data (LHD)****Figur 2: Lekkasjedeteksjon basert på volumbalanse****Figur 3: Manuell avlesning i Grafana**

## **D Vedlegg flomsikring**

### **D.1 Innregulering flomsikring**

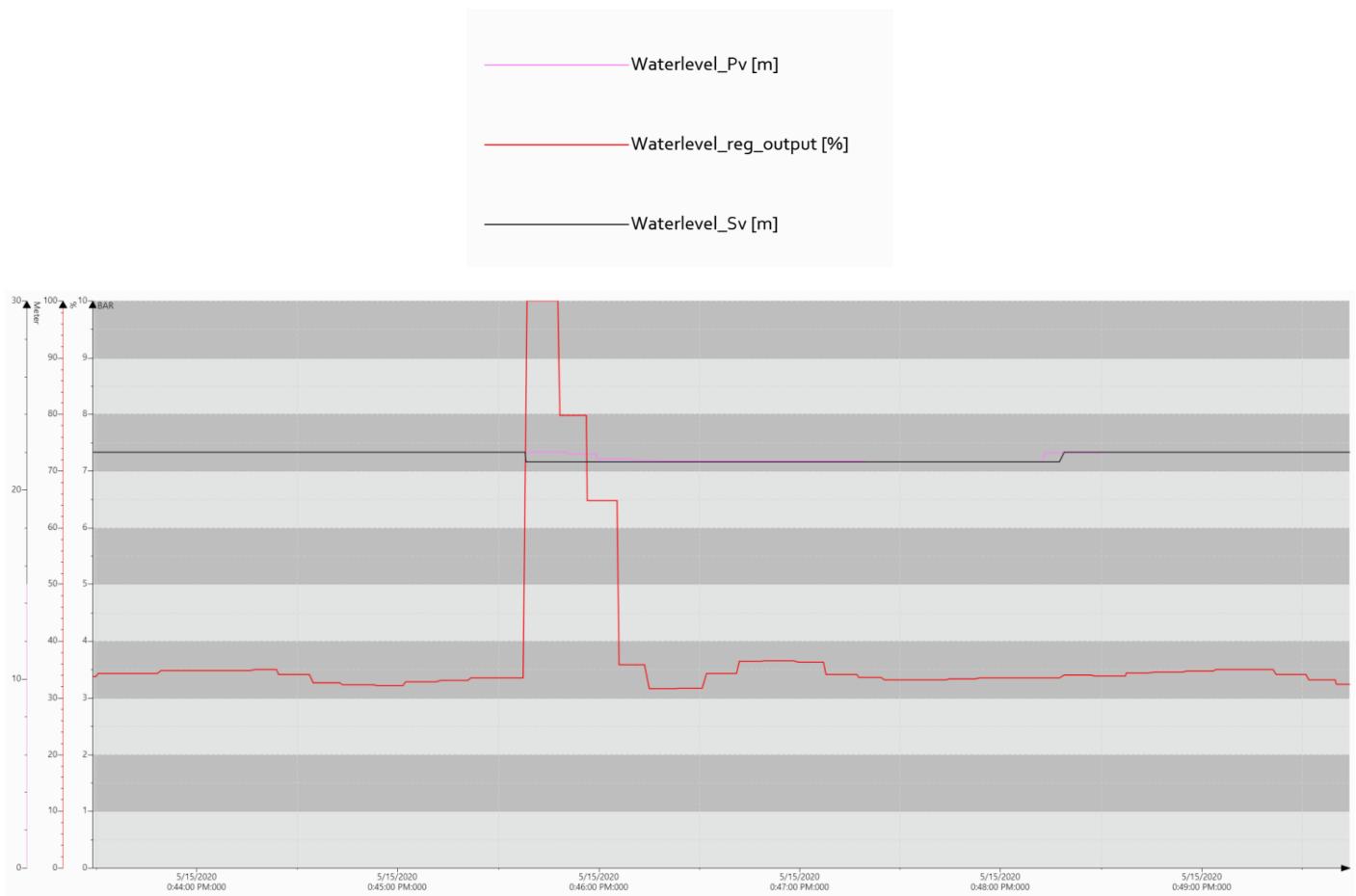
## Innregulering av flomsikring med Ziegler-Nichols metode

■ Prosessverdi [m]  
■ Ventilposisjon [%]  
■ Referanse [m]



## D.2 Resultat flomsikring

### Resultat flomsikring

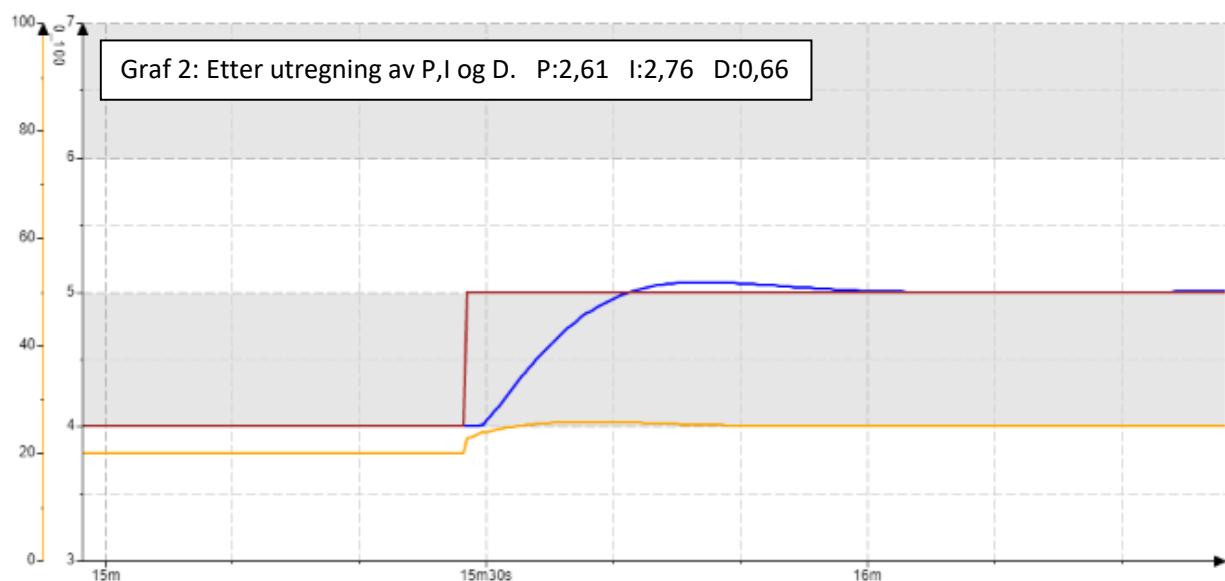
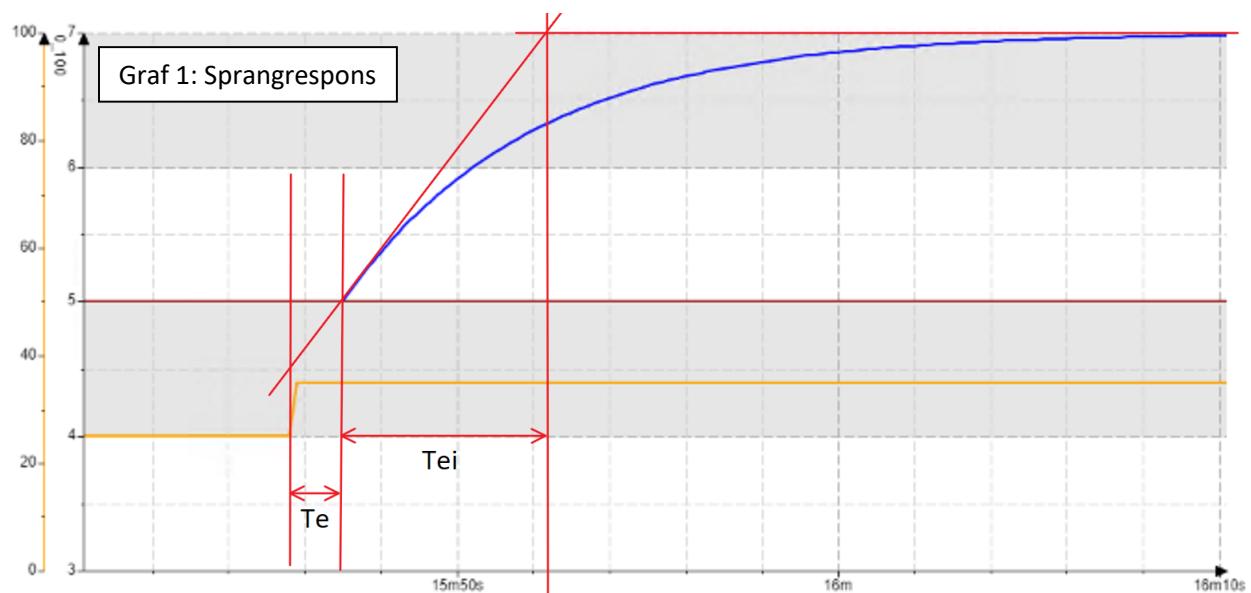


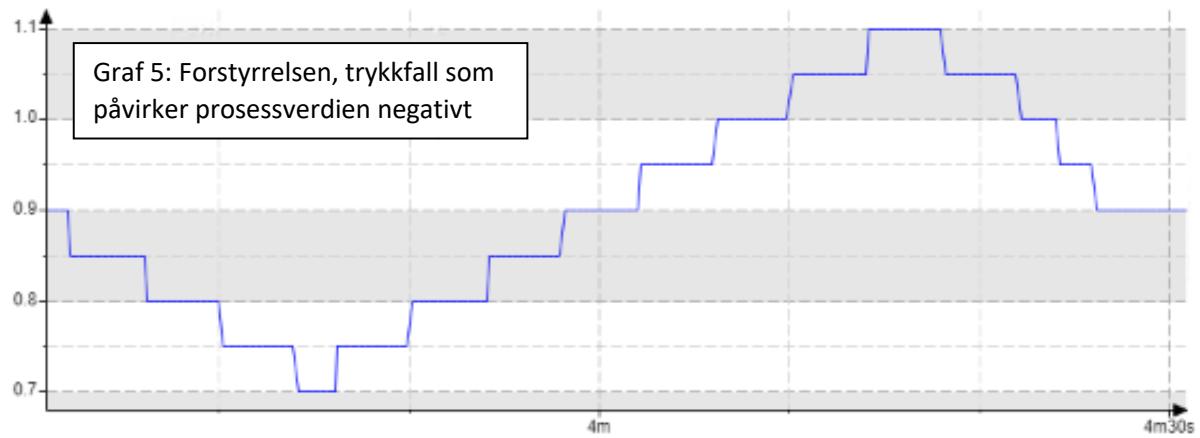
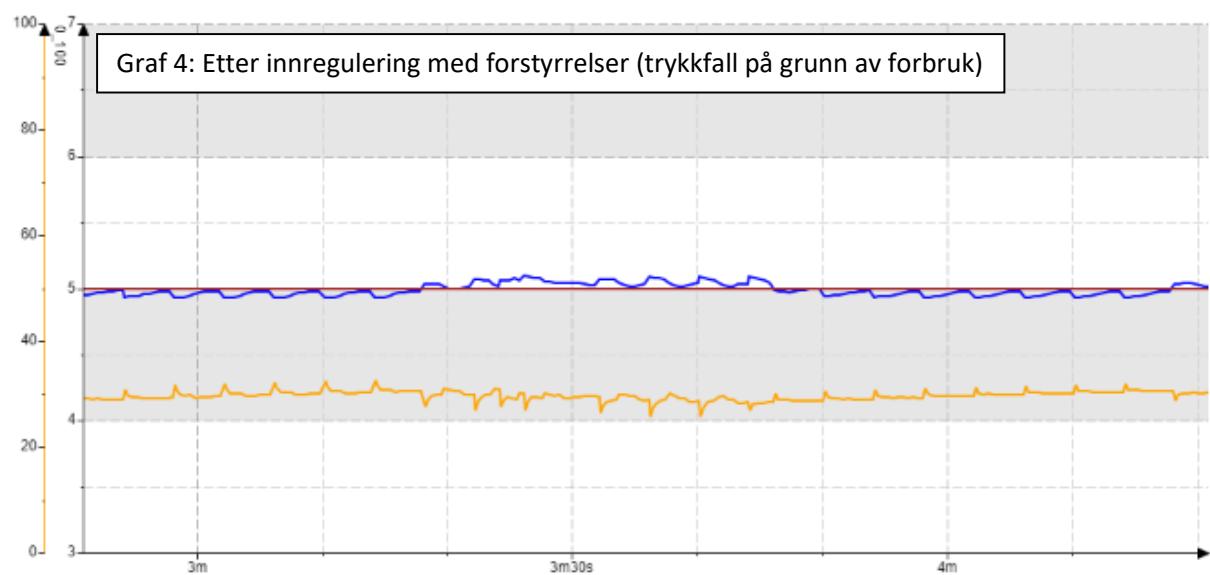
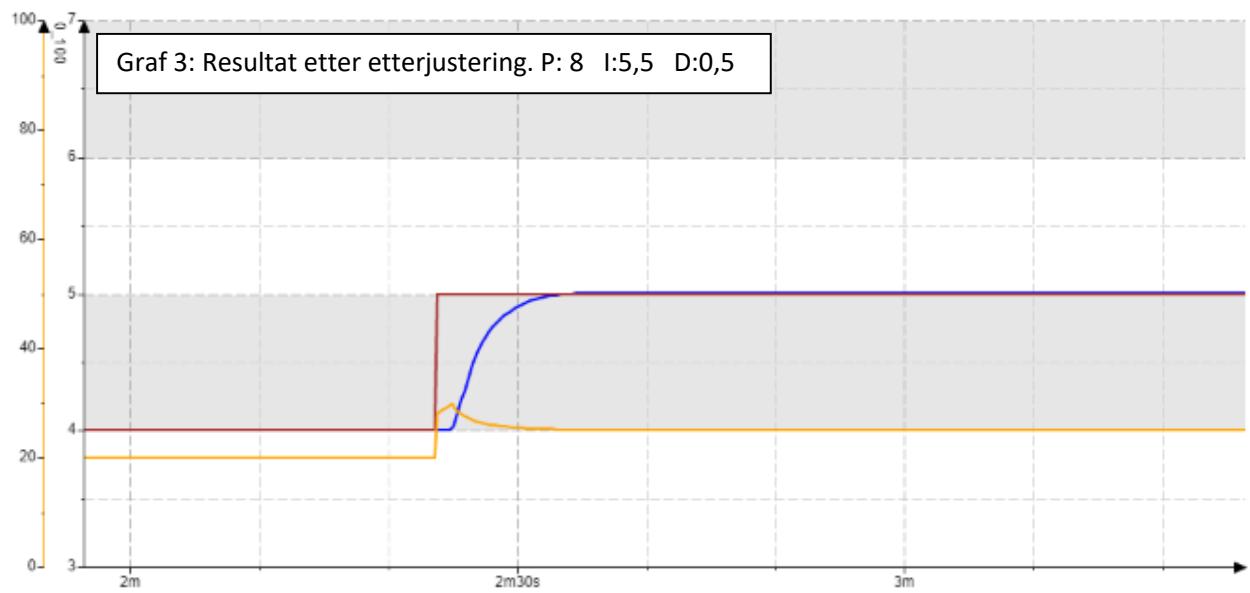
## **E Vedlegg trykkregulering**

### **E.1 Innregulering og resultat trykkregulering**

## Innregulering av trykkregulering med sprangresponsmetoden

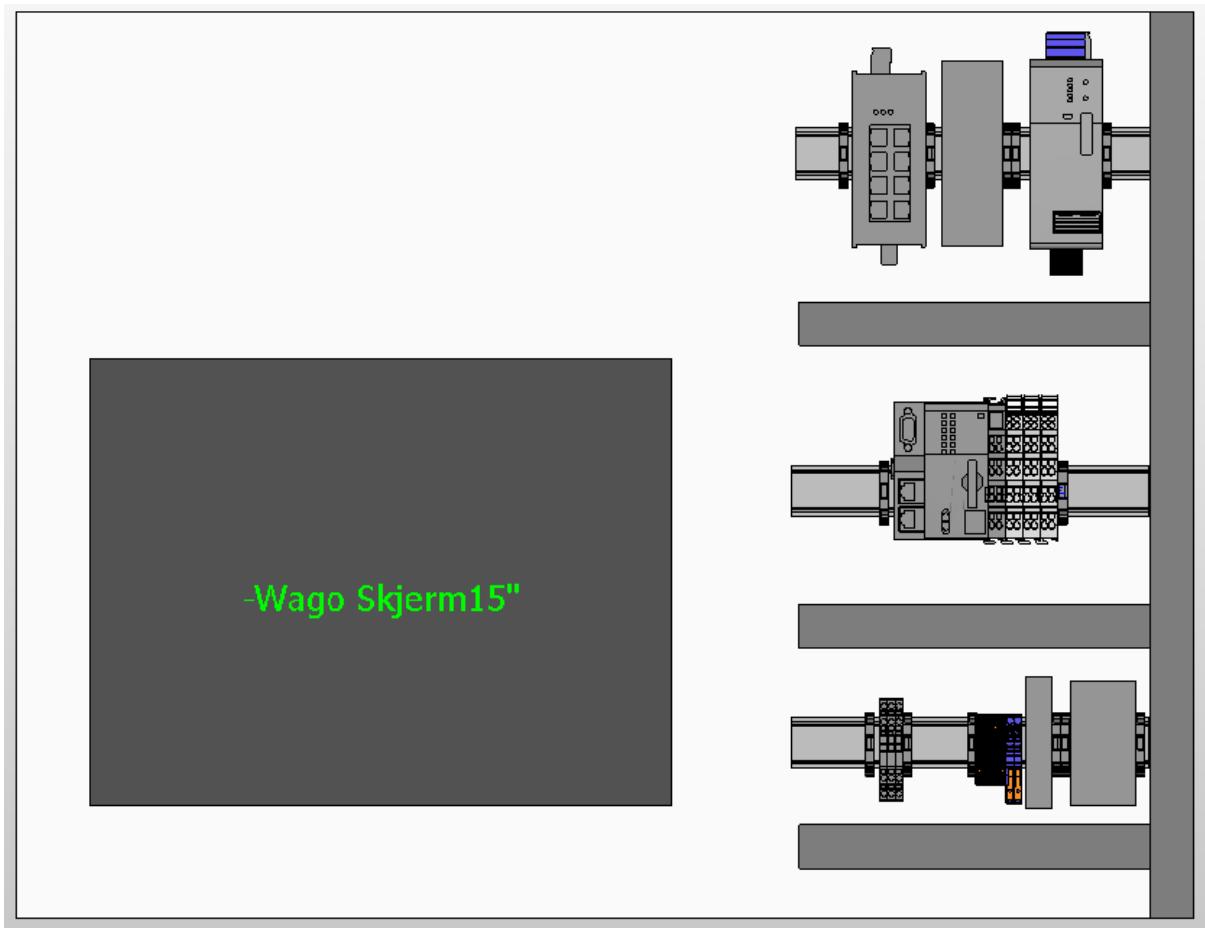
■ Prosessverdi [bar]  
■ Pådrag [%]  
■ Referanse [bar]

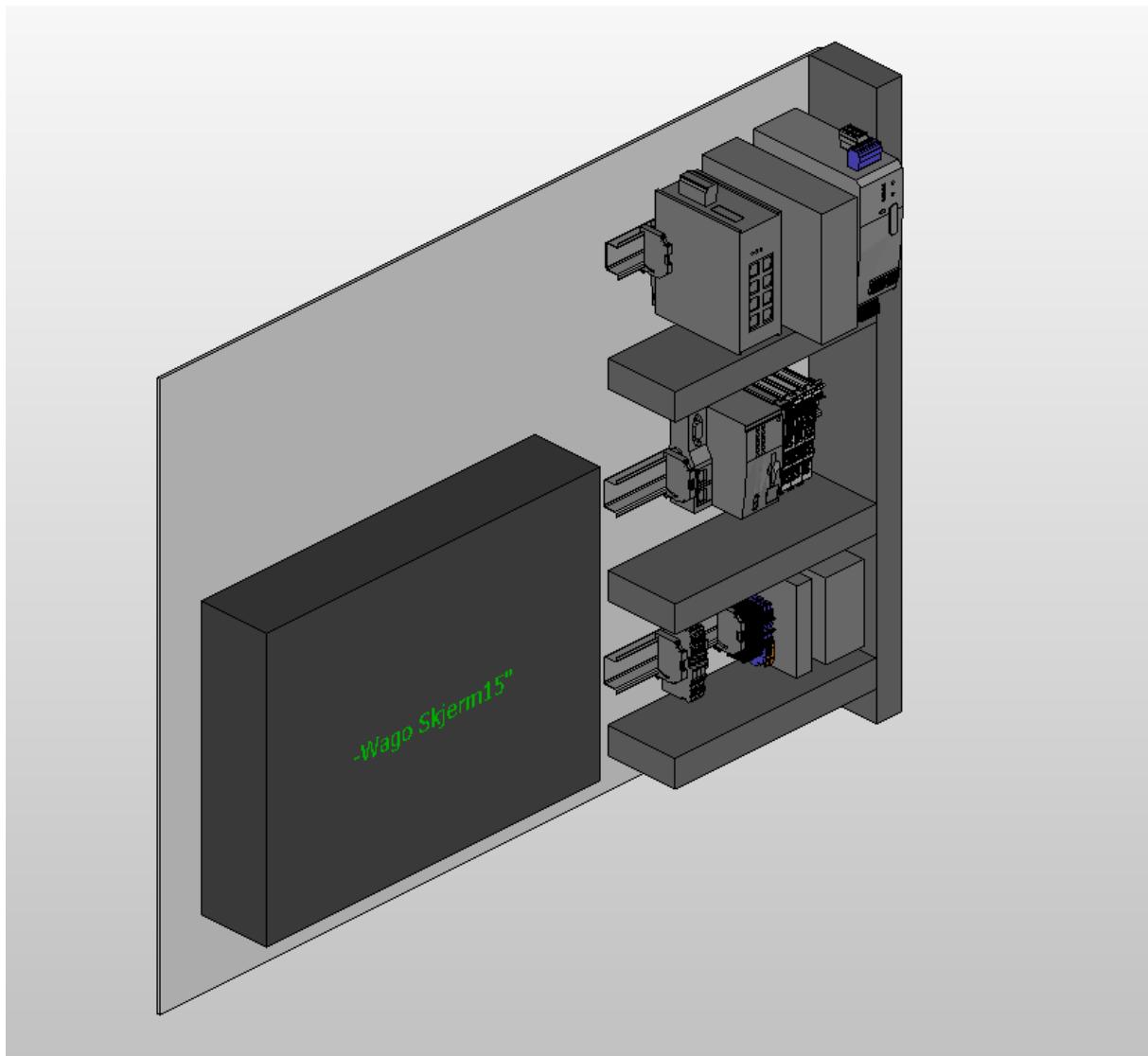




## **F Vedlegg demorigg**

### **F.1 Arrangement tegninger demorigg**

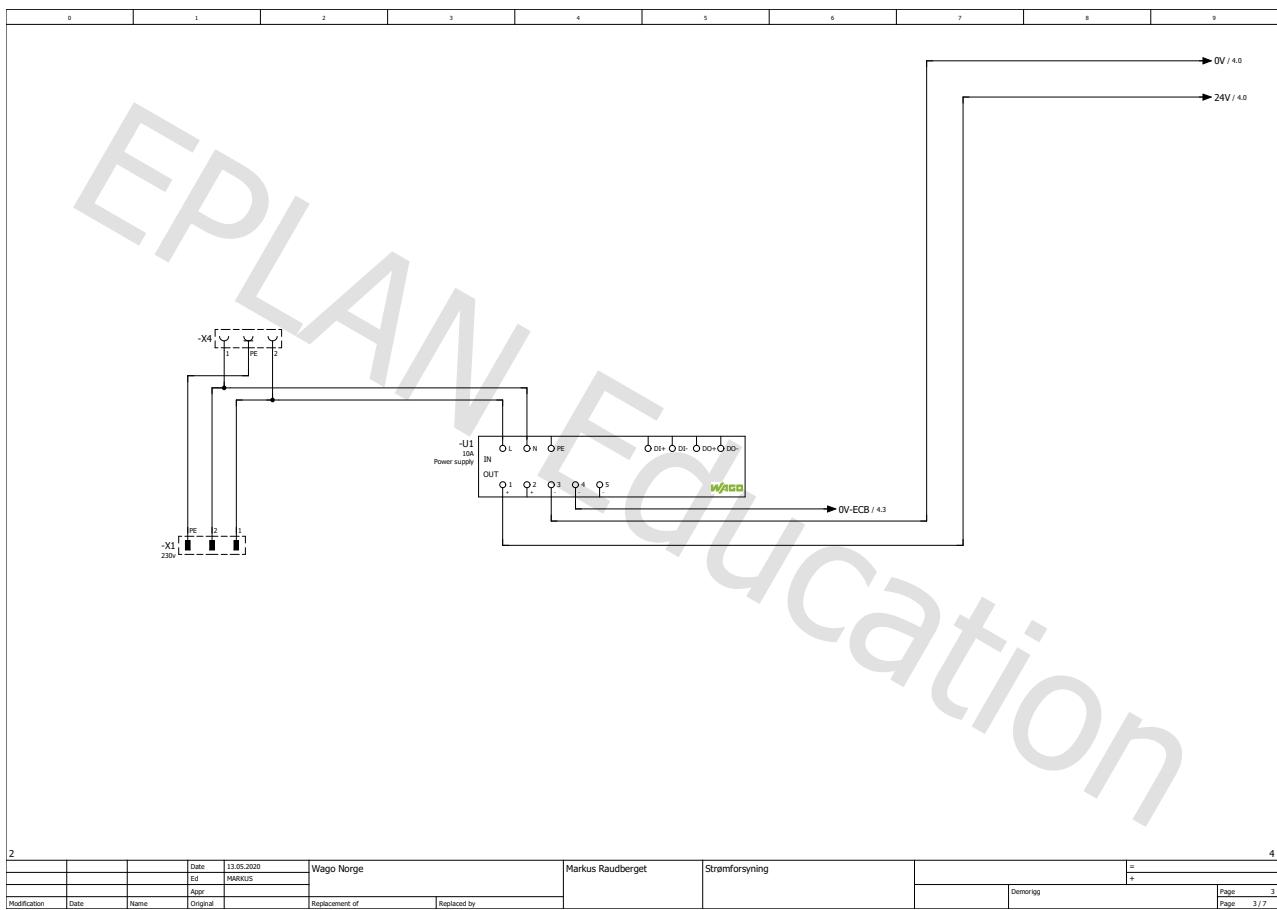


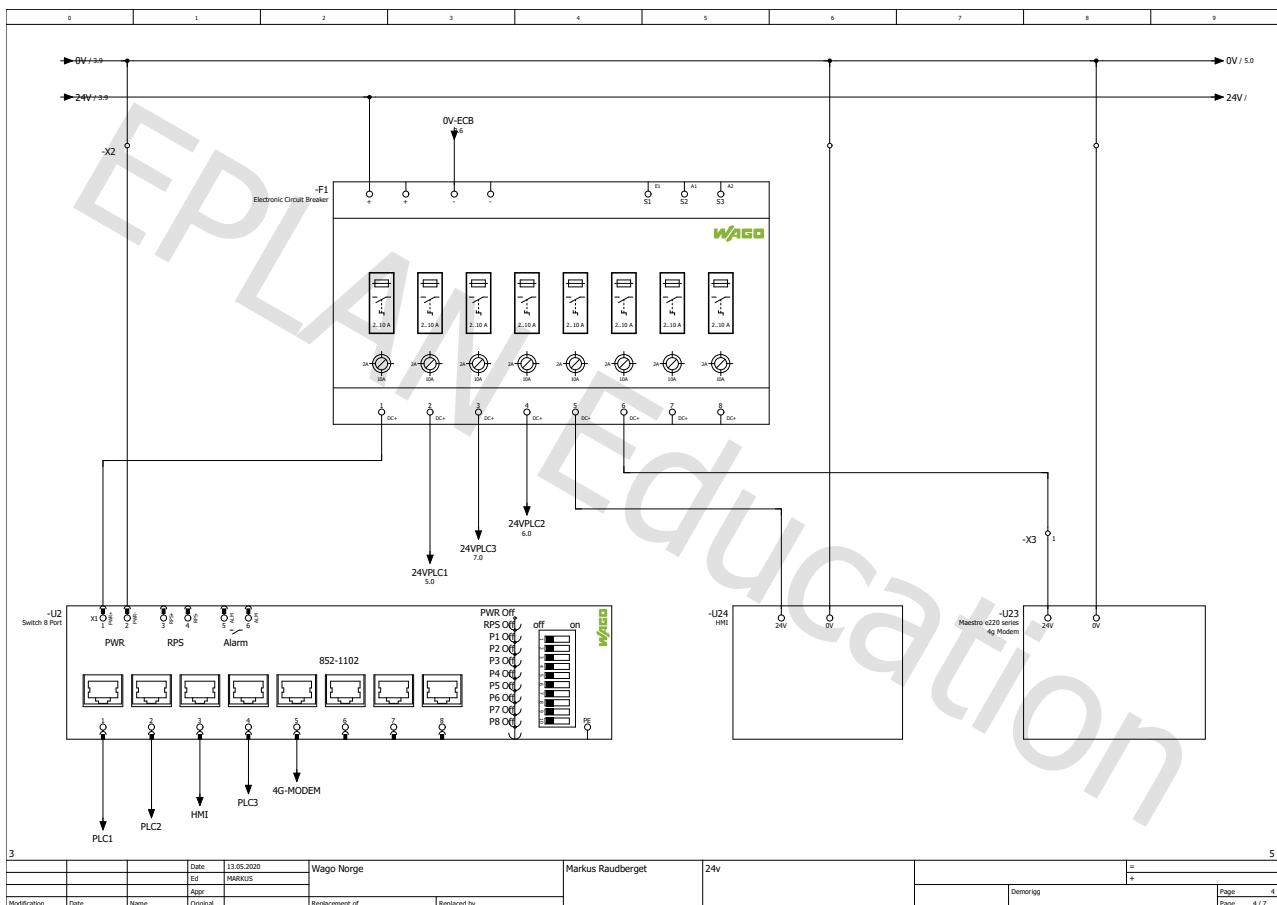


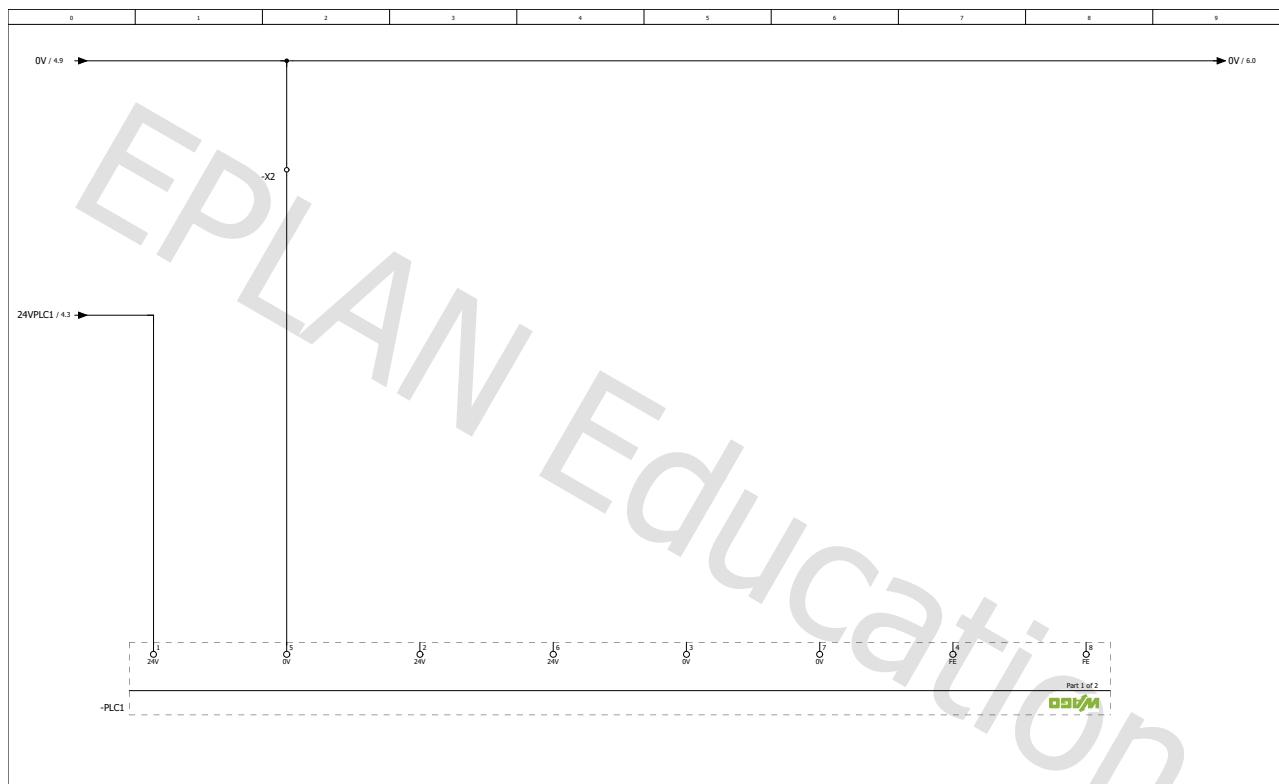
## **F.2 Koblingsskjema demorigg**

0	1	2	3	4	5	6	7	8	9																																																																																																																																								
F26_001																																																																																																																																																	
 <p><b>Markus Raudberget</b></p> <p>Nedre Ila 5 7018 Trondheim Phone +47 90873917</p>																																																																																																																																																	
<table border="1" style="width: 100%; border-collapse: collapse;"> <tr> <td style="width: 15%;">Company / customer</td> <td colspan="8"></td> </tr> <tr> <td>Project description</td> <td colspan="8"></td> </tr> <tr> <td>Job number</td> <td colspan="8">Demorigg</td> </tr> <tr> <td>Commission</td> <td colspan="8">Wago Norge</td> </tr> <tr> <td>Manufacturer (company)</td> <td colspan="8">Markus Raudberget</td> </tr> <tr> <td>Path</td> <td colspan="8">EPLAN sample project</td> </tr> <tr> <td>Project name</td> <td colspan="8">wago</td> </tr> <tr> <td>Make</td> <td colspan="8"></td> </tr> <tr> <td>Type</td> <td colspan="8"></td> </tr> <tr> <td>Place of installation</td> <td colspan="8"></td> </tr> <tr> <td>Responsible for project</td> <td colspan="8"></td> </tr> <tr> <td>Part feature</td> <td colspan="8"></td> </tr> <tr> <td>Created on</td> <td colspan="8">30.03.2020</td> </tr> <tr> <td>Edit date</td> <td colspan="8">13.05.2020 by (short name) MARKUS</td> </tr> <tr> <td colspan="8"></td> <td style="text-align: right;">Number of pages</td> <td style="text-align: right;">7</td> </tr> </table>										Company / customer									Project description									Job number	Demorigg								Commission	Wago Norge								Manufacturer (company)	Markus Raudberget								Path	EPLAN sample project								Project name	wago								Make									Type									Place of installation									Responsible for project									Part feature									Created on	30.03.2020								Edit date	13.05.2020 by (short name) MARKUS																Number of pages	7
Company / customer																																																																																																																																																	
Project description																																																																																																																																																	
Job number	Demorigg																																																																																																																																																
Commission	Wago Norge																																																																																																																																																
Manufacturer (company)	Markus Raudberget																																																																																																																																																
Path	EPLAN sample project																																																																																																																																																
Project name	wago																																																																																																																																																
Make																																																																																																																																																	
Type																																																																																																																																																	
Place of installation																																																																																																																																																	
Responsible for project																																																																																																																																																	
Part feature																																																																																																																																																	
Created on	30.03.2020																																																																																																																																																
Edit date	13.05.2020 by (short name) MARKUS																																																																																																																																																
								Number of pages	7																																																																																																																																								
2																																																																																																																																																	
<table border="1" style="width: 100%; border-collapse: collapse;"> <tr> <td>Date</td> <td>17.01.2019</td> <td>Ed</td> <td>MARKUS</td> <td>Wago Norge</td> <td rowspan="2">Markus Raudberget</td> <td rowspan="2">Title page</td> <td>=</td> <td></td> </tr> <tr> <td></td> <td></td> <td>Apor</td> <td></td> <td></td> <td></td> <td></td> <td>+</td> <td></td> </tr> <tr> <td>Modification</td> <td>Date</td> <td>Name</td> <td>Original</td> <td>Replacement of</td> <td>Replaced by</td> <td></td> <td></td> <td></td> </tr> </table>										Date	17.01.2019	Ed	MARKUS	Wago Norge	Markus Raudberget	Title page	=				Apor					+		Modification	Date	Name	Original	Replacement of	Replaced by																																																																																																																
Date	17.01.2019	Ed	MARKUS	Wago Norge	Markus Raudberget	Title page	=																																																																																																																																										
		Apor							+																																																																																																																																								
Modification	Date	Name	Original	Replacement of	Replaced by																																																																																																																																												





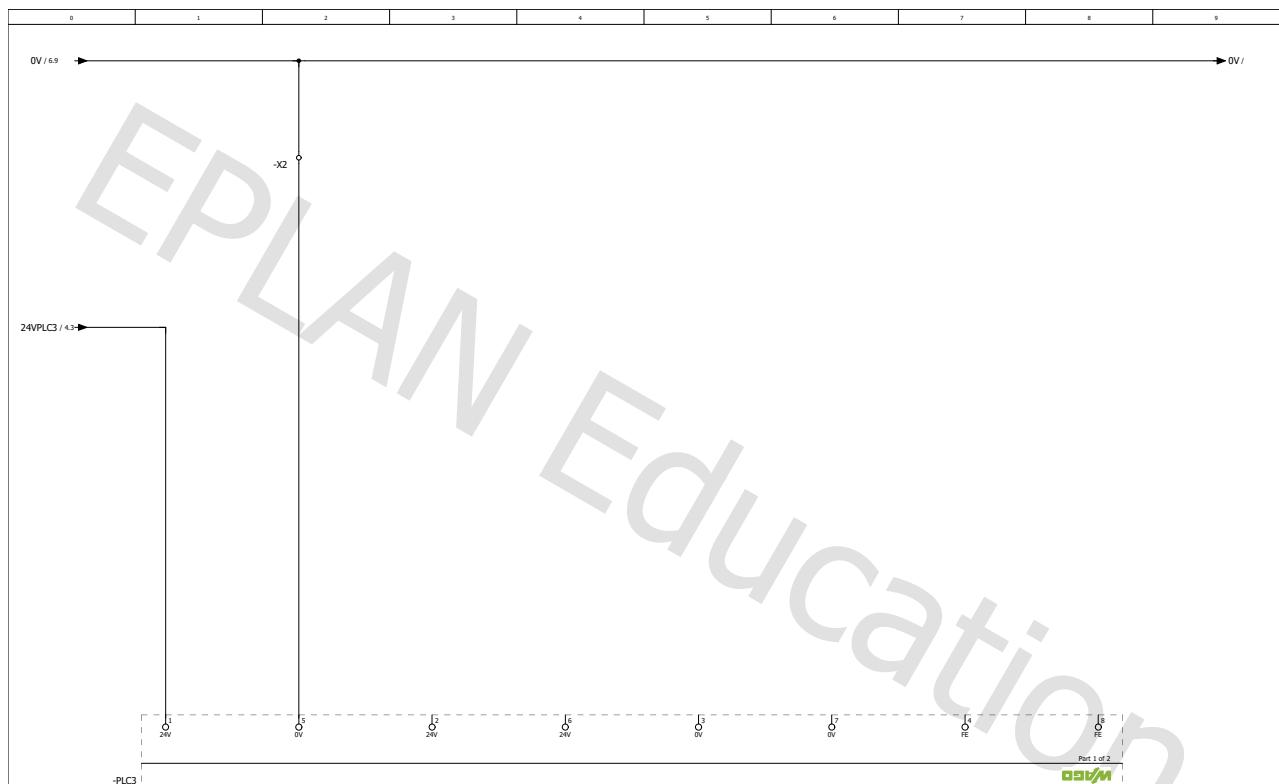




4			Date	13.05.2020	Wago Norge	Markus Raudberget	PLS1	=	6
			Ed	MARKUS				+	
			Appr						
Modification	Date	Name	Original	Replacement of	Replaced by				
								Demosig	Page 5 / 7

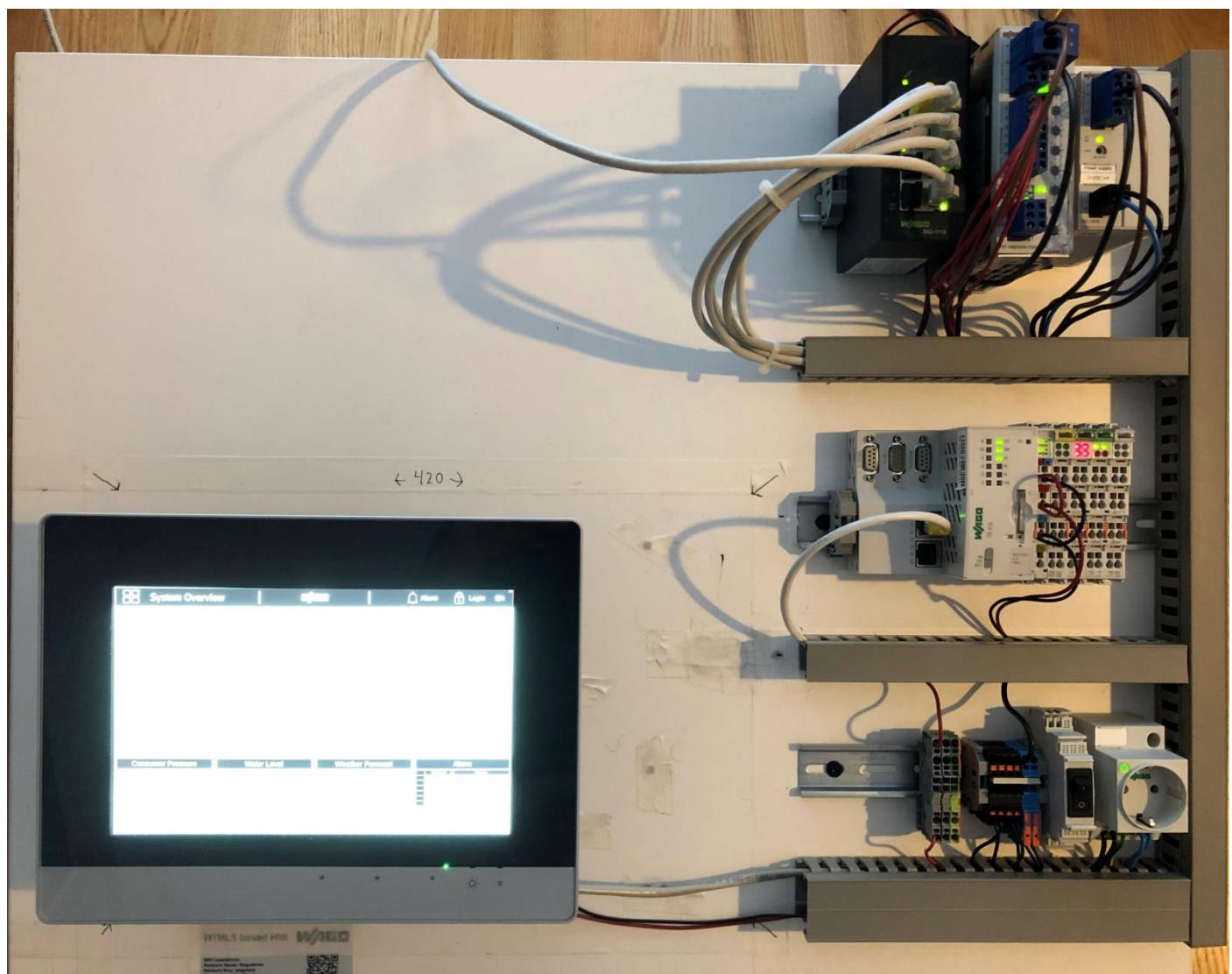


5			Date	13.05.2020	Wago Norge	Markus Raudberget	PLS2	=	7
			Ed	MARKUS				+	
			Appr						
Modification	Date	Name	Original	Replacement of	Replaced by				
								Demosig	Page 6 / 7



6			Date	13.05.2020	Wago Norge	Markus Raudberget	PLS3	=
			Ed	MARKUS				+
			Appr					+
Modification	Date	Name	Original	Replacement of	Replaced by		Demorg	Page 7/7

**F.3 Bilde demorigg**



## **F.4 Deleliste demorigg**

Funksjon	EI NR /leverandør	Antall	Navn	Leverandør
PLS 1 Lekkasjejedeksjon	750-8216	1	Kontroller PFC200; 2. generasjon; 2 x ETHERNET, RS-232/-485, CAN, CANopen, PROFIBUS Slave; lysegrå	Wago
PL2 2 Tryggregulering	750-8213	1	Kontroller PFC200; 2. generasjon; 2 x ETHERNET, CAN, CANopen; lysegrå	Wago
PLS 3 Flomsikring	750-8214	1	Kontroller PFC200; 2. generasjon; 2 x ETHERNET, RS-232/-485, CAN, CANopen; lysegrå	Wago
Strømforsyning	2787-2146	1	Strømforsyning; Pro 2; 1-fase; 24 VDC utgangsspenning; 10 A utgangsstrøm; TopBoost + PowerBoost; communication capability	Wago
Elektrisk sikringsmodul	787-1668/006-1000	1	electronic circuit breaker; 8-kanals; 24 VDC inngangsspenning; justerbar 0,5 ... 6 A; active current limitation; communication capability	Wago
4g Modem	Varenr: 111565	1	Lantronix E224 LTE-Router	Last mile
Switch	852-1112	1	Industriell ECO-svitsj; 8-ports 1000Base-T; sort	Wago
HMI	Ikke lansert	1	Ikke lansert fra Wago	Wago
Digital inn pls 1	750-402/025-000	1	4-kanals digital inngang; 24 V DC; 3 ms; Ekst. temperatur; lysegrå	Wago
Analog inn pls 1	750-455/025-000	1	4-kanals analog inngang; 4-20 mA; Ensidig; Ekst. temperatur; lysegrå	Wago
Analog inn pls 1	750-475/020-000	1	2AI; AC/DC 0-5 A; Differensialinnganger; lysegrå	Wago
Endemodul pls 1	750-600/025-000	1	Endemodul; Ekst. temperatur; lysegrå	Wago
Endemodul pls 2 og 3	750-600	2	Endemodul; lysegrå	Wago
Digital ut pls 3	750-530	1	8-kanals digital utgang; 24 V DC; 0,5 A; lysegrå	Wago
Analog inn pls 3	750-458	1	8AI; Termoelement; justerbar; -	Wago
Stikkontakt	709-581	1	Bryterutstyr-uttaksskap; for DIN-skinne- og skruemontering; for plugg, type F,	Wago
Bryter	789-801	1	Vekslingsmodul; med automatsikring; Kablingsspenning: 250 VAC; Koblingsstrøm: 16 A	Wago

## **G Vedlegg systemtesting**

### **G.1 FAT**

**FABRIKK  
AKSEPTERINGS  
TEST**

<b>Hensikt</b>	<b>3</b>
<b>Omfang</b>	<b>3</b>
<b>Prosedyre</b>	<b>3</b>
HMI:	4
System Overview/System Oversikt:	5
Leak Detection/Lekkasjedeteksjon:	7
Flood Contol/Flomsikring:	8
Consumer Pressure/Forbruker Trykk:	9
Trends/Trender:	10
Alarms/Alarmer:	11
<b>Avslutning</b>	<b>12</b>

## Hensikt

Hensikten med FAT er å forsikre utviklere og oppdragstaker at systemet fungerer etter gitte spesifikasjoner.

Det er spesifisert av oppdragsgiver at designet i HMI skal utvikles for et kommende panel, som ikke er tilgjengelig på nåværende tidspunkt. Dette fører til at bildet ser kan se noe "trykt" ut.

## Omfang

Testen tar for seg å teste interaktivitet og at funksjoner fungerer som de skal, samt teste feilhåndtering ved f.eks. kommunikasjonsbrudd.

Siden systemet kommuniserer over internett, vil det være merkbar tidsforsinkelse mellom HMI og PLS. Det betyr at ved endringer i HMI, vil det ta 5-12 sekunder før endringer blir skrevet i PLS. Dette er gjeldende for kommunikasjonen mellom PLS og HMI også.

## Prosedyre

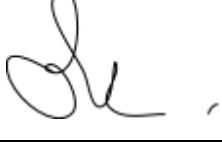
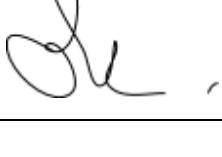
### HMI:

Ved første oppstart av systemet står brukergrensesnittet som default på engelsk språk. Testen foretar seg først en grundig test av et gitt objekt og deretter blir det tatt stikkprøver fra flere objekter. Eksempelvis strømningsmålinger i lekkasjedeteksjon.

**Manøvrering mellom bilder:** Menyen åpnes ved å trykke på menyknappen, fire firkanter, øverst til venstre. Velg deretter ønsket bildet.

**POPUP:** Ved aktivt popupvindu vil ikke bildet i bakgrunnen være tilgjengelig. Popupvinduet må først krysses ut, ved hjelp av hvitt kryss i tittelbaren.

## System Overview/System Oversikt:

Handling	Mål	Godkjent
Observer at panelet viser "System Overview/System Oversikt" bildet.	Oppstart har gått som normalt og at startsiden er konfigurert riktig.	
Se over tekst at alt står på engelsk.	Identifisere om noen tekster avviker fra valgt språk.	
Opp i høyre hjørne, er språkvalg. Ved oppstart vil denne indikere "EN". Trykk og velg "NO"	Språk på tekster blir endret til norsk.	
Se deretter om det er tekster som avviker fra valgt språk.	Sjekk om språkfunksjon fungerer som tiltenkt og identifisering av språkavvik.	
Velg tilbake til engelsk.	Sjekk av språkfunksjon fungerer tilbake fra norsk.	
Observer at grafiske fremstillingen av vannnivå og forbrukertrykk oppdateres.	Sikre at sampling til trace og trend funksjoner kjører.	
Trykk på ventilsymbolet som er ved bunnen av vannkilden.	En popup meny for innstillingar av ventilen blir synlig.	
Utfør valgfrie kommandoer i popup for å se om disse skrives.	Utførte handlinger skal skrives. Dette sikrer strukturen for pådragsobjektet for ventiler.	

Trykk deretter på det hvite krysset som er til høyre i det sorte feltet.	Popugen legges ned.	
Trykk på ruten med variabler på enden av linjen med ventilsymbolene.	Popup for mengdemålinginnstillingene blir synlig.	
Les av verdier som TAG navn og Description.	Sikre at strukturen er korrekt opp imot PLS. Siden konstantene er korrekte.	
Kryss ut på samme måte som ventilobjektet.	Popup legges ned.	
Observer at Weather Forecast/Værmelding har 3 påløpende dager.	Sikre at konfigurasjonen av widget er korrekt.	
Trykk på pumpesymbolene som er plassert etter renseanlegget.	Pumpe-popup blir synlig.	
Utfør valgfrie kommandoer	Eventuelle endringer blir aktivert.	
Trykk på det hvite krysset i toppen av popugen.	Popugen legges ned	

Kommentarer:

## Leak Detection/Lekkasjedeteksjon:

På hovedbildet i leakdetection er det tattstikkprøver av strømningsmålingene. Observer "Battery State/Batteri Status" widget at verdiene er beskrivende.

Handling	Mål	Godkjent
Trykk på den hvite pilen i tittelbaren på bildet for manøvrere til Grafana peker.	Manøvreres til en peker mot Grafana og lekkasjedeteksjonsalgoritmen kommer til syne.	
Observer inne på Grafana om at det kommer nye data ved gitt oppdateringstid	Sikre at det kommer data inn til Grafana	
Observer at en alarm blir gitt når systemet oppdager en lekkasje.	Sjekk av at alarmer ligger inne og at lekkasjedeteksjonsmetoden fungerer	
Trykk på den hvite pilen i tittelbaren for å manøvrere tilbake til HMI'en.	Manøvreres tilbake til HMI'en.	

Kommentarer:

-Grafanapeker fungerer på skjerm, men fungerer ikke i nettleser på datamaskin.

## Flood Control/Flomsikring:

Mange av funksjonene på dette bildet er allerede ivaretatt, slik som ventil og sensorer

Handling	Mål	Godkjent
Trykk på firkanten som er koblet til ventilen med en striplet linje.	Popup til strømningsregulatoren for flomforhindring blir synlig.	
Endre på verdier i popup.	Endringene blir utført.	
Kryss ut av popup ved hjelp av det hvite krysset i høyre øvre hjørne.	Popup legges ned.	
Se på parameter for PID.	Sjekke at riktige parameter er lagt inn for reguleringen.	
Følg med at vannet regulerer seg inn ved endring i settpunkt.	Sjekke at reguleringen fungerer.	

Kommentarer:

## Consumer Pressure/Forbruker Trykk:

Mange av funksjonene er allerede testet.

Handling	Mål	Godkjent
Se på parametrerne på PID.	Sjekke at riktige parameter er lagt inn for reguleringen.	
Se om pumpene alternerer og at to stk. bare kjører samtidig.	Sjekke at funksjonen med alternering av pumper fungerer.	
Se om trykket opprettholdes i forhold til settpunkt.	Sjekke at reguleringen fungerer som normalt.	

Kommentarer:

**Trends/Trender:**

<b>Handling</b>	<b>Mål</b>	<b>Godkjent</b>
Trykk på valgfrie variabler.	Se at det visualiseres verdier i trendvinduet.	

Kommentarer:

## Alarms/Alarmer:

MV: Maskinvare, alarmer som blir initiert av maskinvare feil. For å initiere alarm, utføres handlingen fysisk.

PV: Progra,vare, alarmer som blir initiert av brudd på alarmgrenser for initiere alarm, utføres ved å sette alarm flagg i den globale variabel listen HMI, ved hjelp av PC.

Alarmfilosofi: Rød tekst - Aktiv alarm. Gul tekst - Aktiv og kvittert alarm. Blå - Inaktiv og ukvittert.

Handling	Mål	Godkjent
HW: Dra ut nettverkstilkoblingen til PLS'ene, vent 60-70 sekunder.	Observer at kommunikasjonsalarmene til PLS'ene blir initiert. Ved rød tekst i alarmlisten.	
Kvitter en valgfri alarm.	Observer at alarmen skifter farge fra rød til gul tekst.	
HW: Koble til nettverktilkoblingen	Observer at røde alarmtekster går til blå og gule alarmtekster forsvinner.	

Kommentarer:

## Avslutning

Signering av gjennomført test og har funnet testprotokollen godkjent på samtlige punkter.

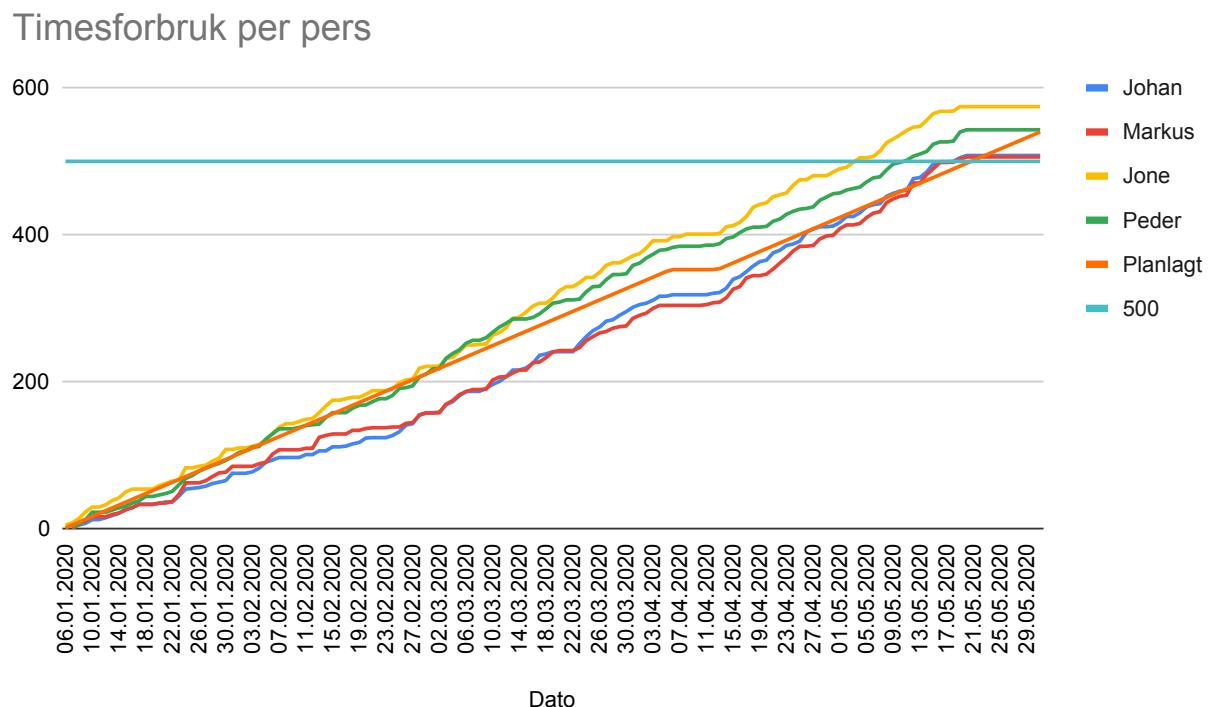
Sign.: 	Sign.: Markus Raudberget
Sign.: 	Sign.: Peder Wæland

Trondheim, 19.5.20

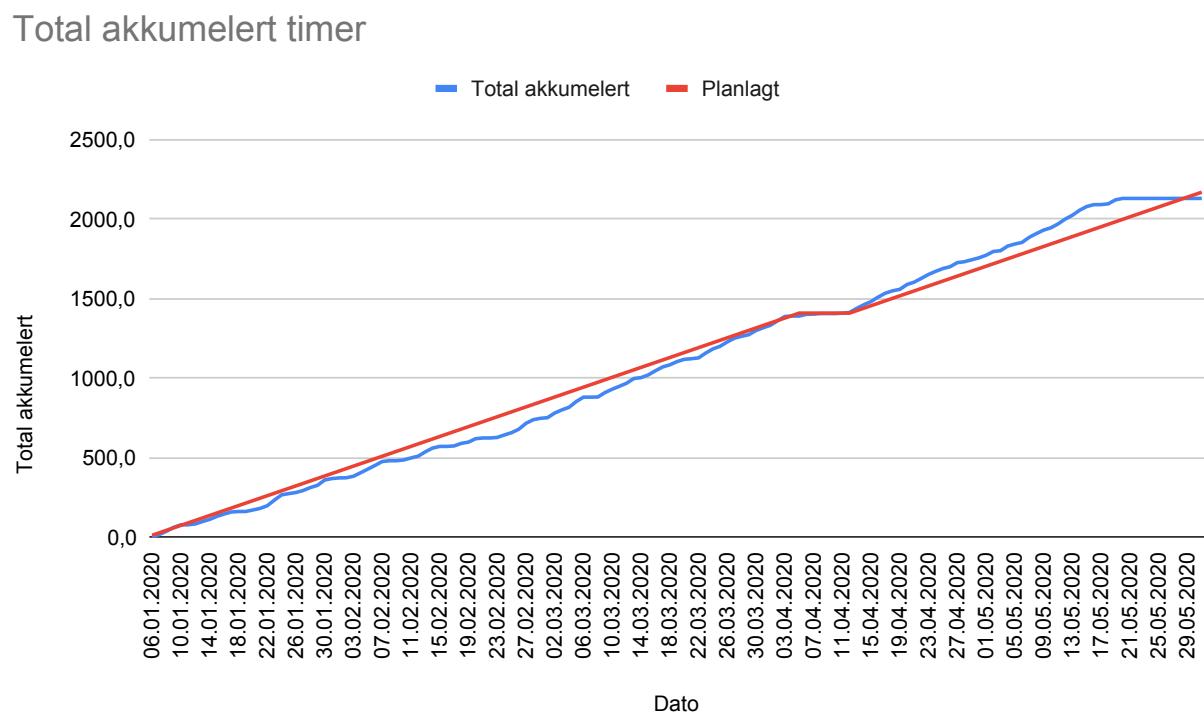
Sted, Dato

## **H Vedlegg timeforbruk**

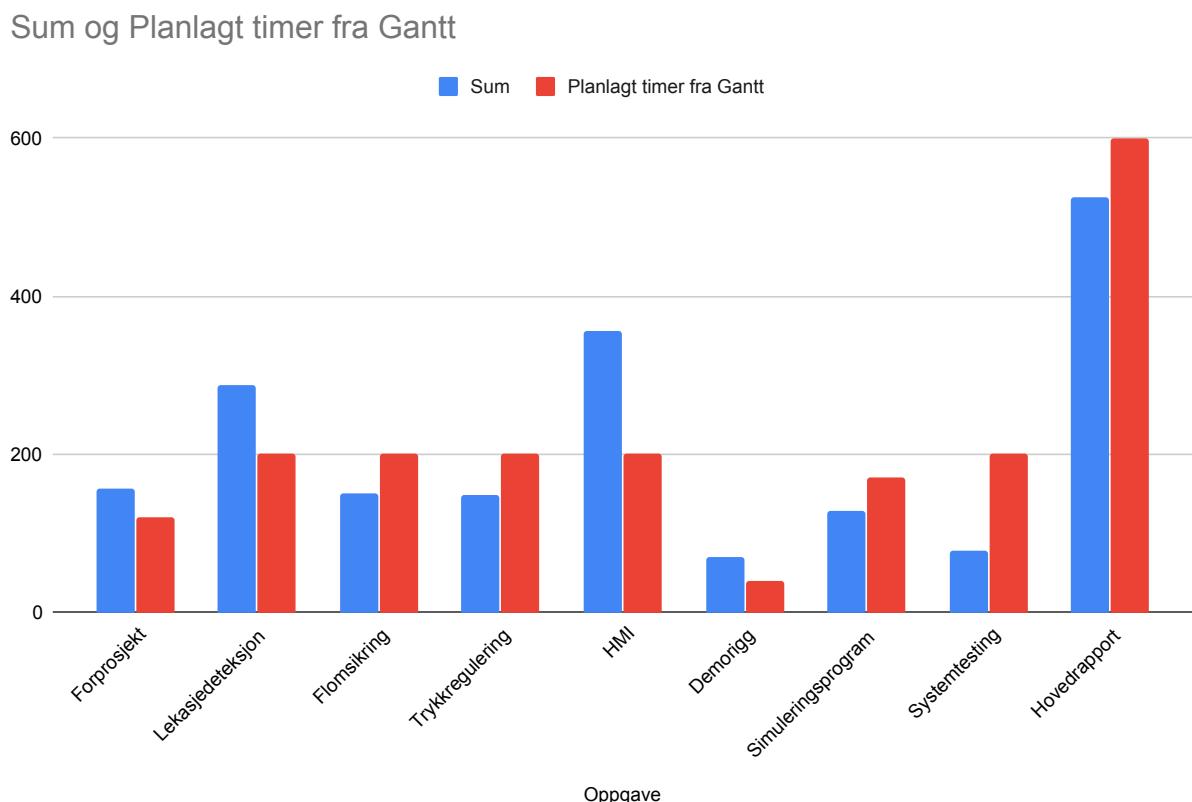
## H.1 Total timer per person



## H.2 Total timer tilsammen

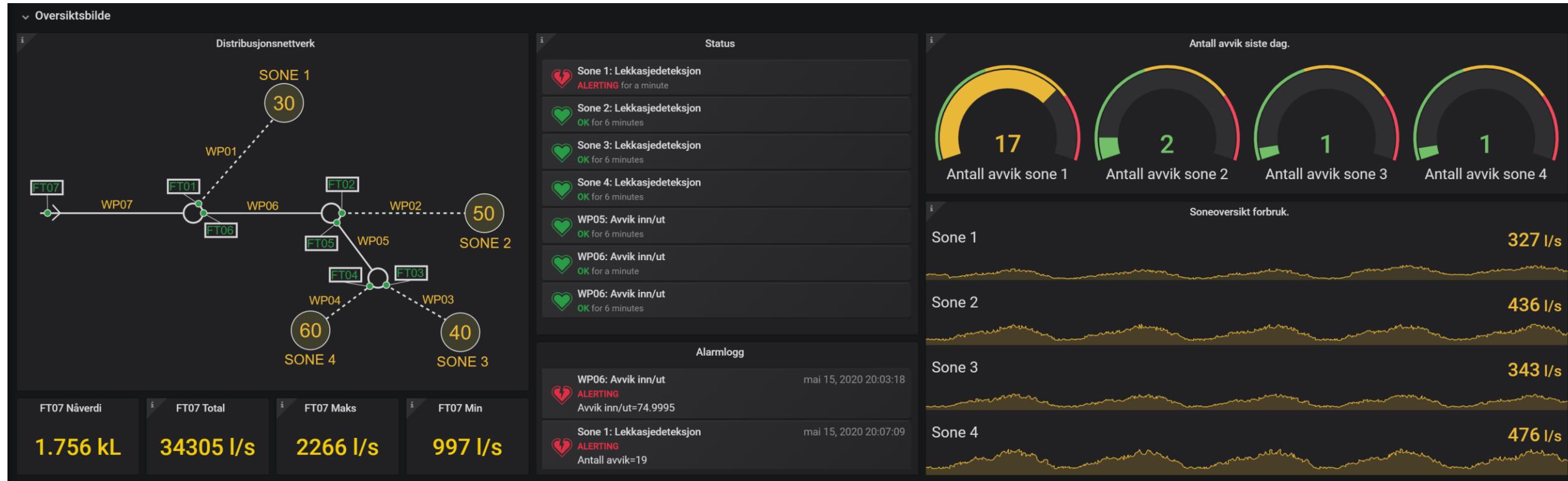
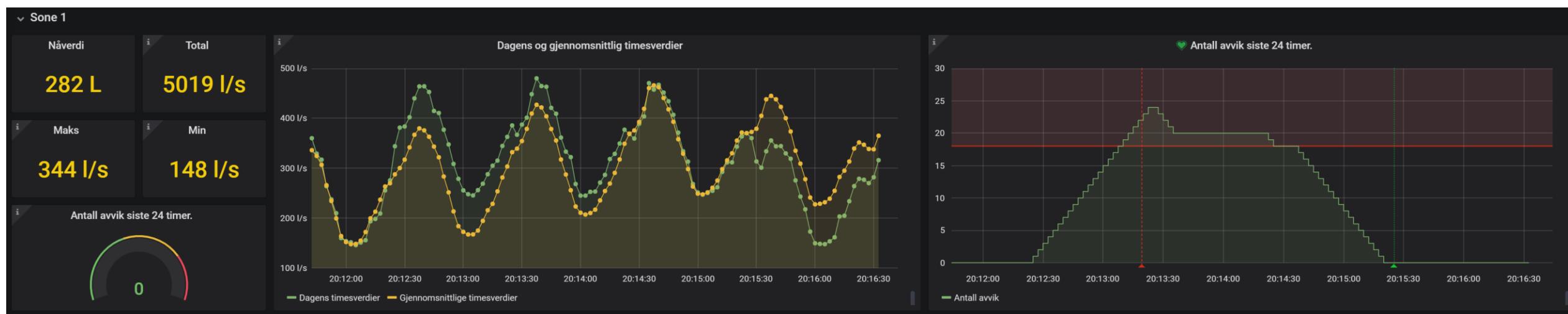


### H.3 Total timer mot planlagt timer fra Gant



## **I Vedlegg brukergrensesnitt**

### **I.1 Brukergrensesnitt Grafana**

**Oversiktsbilde****Rør overvåkning****Soneoversikt**

## I.2 Brukergrensesnitt e!COCKPIT Visualization

### System Overview

**Consumer Pressure**

**Water Level**

**Weather Forecast**

Day	Temp (°C)	Rain (mm)
Sunday	3	0
Monday	7	0
Tuesday	7*	0 mm

**Alarm**

Timestamp	Message
19.05.2020 04:52:47	UA - MQTT link had been down for 60 seconds
19.05.2020 04:52:36	No communication with PLC1 in 60 seconds
19.05.2020 04:52:28	No communication with PLC3 in 60 seconds
19.05.2020 04:52:23	No communication with PLC2 in 60 seconds
16.05.2020 15:03:52	Pump 3 mode set to TRUE
16.05.2020 15:03:47	Pump 2 mode set to TRUE
16.05.2020 14:53:49	Pump 1 manual Start: 0.00
16.05.2020 14:52:57	Pump 2 manual Start: 0.00
16.05.2020 14:52:50	Pump 3 manual Start: 0.00

**Distributed Water**

**Battery Status**

Zone	Level	Time Left	State
K1	14.37 Ah	1.61 h	Charging
K2	12.32 Ah	2.22 h	Charging
K3	17.23 Ah	0.81 h	Charging

**Weather Forecast**

**Alarm Leak Detection**

### Distribution Overview

**Consumer Pressure**

**Water Level**

**Weather Forecast**

Day	Temp (°C)	Rain (mm)
Sunday	3	0 mm
Monday	7	0 mm
Tuesday	7*	0 mm

**Alarm**

Timestamp	Message
16.05.2020 15:03:52	Pump 3 mode set to TRUE
16.05.2020 15:03:47	Pump 2 mode set to TRUE
16.05.2020 14:53:49	Pump 1 manual Start: 0.00
16.05.2020 14:52:57	Pump 2 manual Start: 0.00
16.05.2020 14:52:50	Pump 3 manual Start: 0.00

### Flood Control

**Emission Flow**

**Water Level**

**Weather Forecast**

Day	Temp (°C)	Rain (mm)
Sunday	3	0 mm
Monday	7	0 mm
Tuesday	7*	0 mm

**Alarm Flood Control**

### Consumer Pressure

**Treatment Pressure**

**Consumer Pressure**

**Pumps**

Pump	Current	Uptime
P01	0.00 A	22766 h
P02	2.53 A	22763 h
P03	2.52 A	22767 h

**Alarm Pressure Control**

### Trend

1s 10s 30s 1m 10m 30m All

Trend Variables

Pressure_Sv	Waterlevel_Sv
Pressure_reg_output	Waterlevel_reg_output
Pressure_Pv	Waterlevel_Pv

### Alarms

Alarms

Timestamp	Class	Message
0	Notifications	Pump 3 mode set to TRUE
1	Notifications	Pump 2 mode set to TRUE
2	Notifications	Pump 1 mode set to TRUE
3	Notifications	Pump 1 manual Start: 0.00
4	Notifications	Pump 2 manual Start: 0.00
5	Notifications	Pump 3 manual Start: 0.00

Acknowledge Selected

Acknowledge All

History

## **I.3 Brukermanual Grafana**

# Brukermanual Grafana

2020

## Innholdsfortegnelse

<b>Oppstart</b>	<b>3</b>
Velge database	3
Importere dashboard	3
<b>Teknisk</b>	<b>3</b>
Lage nytt dashboard	3
Lage nye paneler	4
Forklaring av paneler	5
Lage nye rad	5
Oppdateringstid	5
Tidsrom	5
<b>Forklaring av skjermbilder</b>	<b>6</b>
Oversiktsbilde	6
Rør overvåking	7
Soneoversikt	7
<b>Alarmer</b>	<b>8</b>
Lage alarmer	8
Status for alarmer	8
Flytte alarmgrenser	9
Kvittering alarmer	9
Forhåndssatte alarmer	9
Alarm for lekkasjedeteksjon i sone	9
Alarm for lekkasje i rør	10

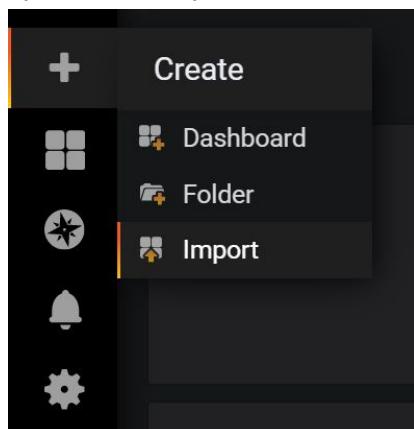
# Oppstart

## Velge database

For at Grafana skal visualisere data må programmet ha tilgang på data. Denne dataen tar Grafana gjennom databaser. Ved å trykke på “Create a data source” på “Home” i Grafana får man mulighet til å velge ønsket database. Under dette prosjektet er det brukt MySQL database, man velger derfor den i dette tilfellet. Etter å ha valgt MySQL må man sette inn informasjon om databasen. I dette tilfellet er “Host”: **db:3306**, “Database”: **processvalues**, “User”: **root** og “Password”: **example**. Etter at alt dette er lagt inn kan man trykke “Save and Test” for og sjekke kommunikasjonen.

## Importere dashboard

Ønsker man å importere et dashboard som tidligere har blitt lagret som .json fil kan man trykke på “+” symbolet “Create” til venstre i Grafana og velge “Import”.

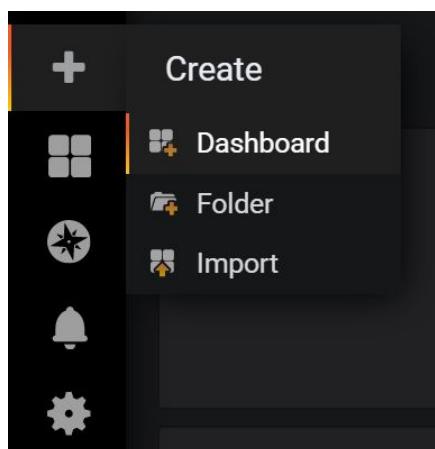


Man får da mulighet til å velge “Upload .json file”. Selv om man importerer et dashbord må riktig database være valgt for den konfigureringen innebære ikke i .json filen. For å få alarmene til å fungere etter importering må man lagre dashboardet en gang etter opplasting.

# Teknisk

## Lage nytt dashboard

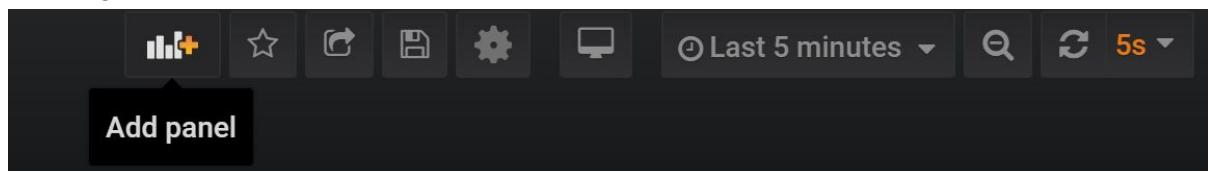
For å lage et nytt dashboard trykker man på “+” symbolet “Create” i menyen til venstre som vist i bildet under og velge “Dashboard”.



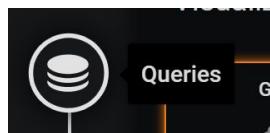
Du får muligheten til å lage ditt første panel på dashboardet. Etter at du har lagt inn et panel kan du lagre dashboardet ditt med et nytt navn.

## Lage nye paneler

For å lage et nytt panel på et dashbord må man trykke på symbolet med et diagram og plussstegn på, som vist på bilde.



Deretter velger man "Choose Visualization" og velger ønskelig visualisering. Her finnes det også tredjeparts paneler som kan installeres. Etter å ha valgt riktig visualisering må trykker man på "Queries" på venstre siden.

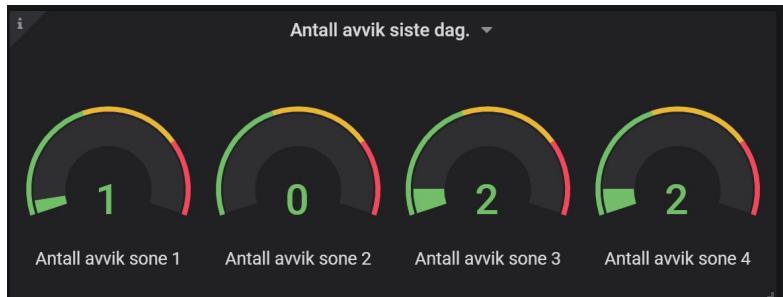


Selv om du har valgt database må man også velge hvilken tabell man skal hente dataene fra. Det velges her. Her er det en fordel og kunne SQL koding, da disse innstillingene baserer seg på SQL kode. Under vises et eksempel der det er tatt inn data fra tabellen "LeakDetection120SamplesHourlyAverage" og sortert på timestamp.

The image shows the 'Query' editor in Grafana. At the top, there is a dropdown menu set to 'default'. Below it, the 'FROM' clause is set to 'LeakDetection120SamplesHourlyAverage' with 'Time column' set to 'timestamp' and 'Metric column' set to 'none'. The 'SELECT' clause has 'Column: id' and an addition sign. The 'WHERE' clause contains 'Macro: \$\_\_timeFilter' and an addition sign. The 'GROUP BY' clause has an addition sign. At the bottom, the 'Format as' dropdown is set to 'Time series', and there are buttons for 'Edit SQL', 'Show Help', and 'Generated SQL'.

## Forklaring av paneler

Skjermbildene som er i Grafana er bygd opp av paneler. Slik som bildet under.



Når man lager et panel har man muligheten til å sette opp en forklaring for panelet. Dette kan være en viktig å gjøre dersom det er andre en deg selv som skal bruke dashbordet. Forklaringen for hvert panel finner du øverst til venstre på trekanten med en "i" på. Dersom det kommer et "loading" symbol øverst til høyre betyr dette at Grafana holder på å hente ny informasjon i dette panelet.

## Lage nye rad

Ønsker man og sortere panelene for å lage et mer brukervennlig brukergrensesnitt kan man lage rader. Dersom man lager en rad har man muligheten til å sette panelene som har noe til felles under samme rad. Man har også muligheten til å lukke og åpne raden, dette gjør brukergrensesnittet ryddigere. For å lage en rad trykker man på "Add panel" og deretter "Convert to row". Da vil alle panelene som er øverst automatisk legge seg i denne raden.

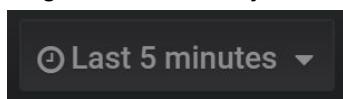
## Oppdateringstid

Oppdateringstiden er den tiden det tar mellom hver gang Grafana prøver å hente data fra databasen. Oppdateringstid velges øverst til høyre på dashbordet. Her kan man velge flere tider mellom fem sekunder og en dag.



## Tidsrom

Tidsrom er den "fra" og "til" tiden man ønsker å visualisere dataene for i panelene. Dette velges øverst til høyre. Her kan man velge ønsket tidsrom for panelene.



## Forklaring av skjermbilder

Under dette kapittelet blir skjermbildene forklart. For å forklare bildene enklest mulig er det tatt utklipp fra radene i dashbordet i Grafana og det brukes rammer med nummer på for å referere til disse bildene. Flere av panelene baserer seg på metoder beskrevet i rapporten "Styre- og overvåkningssystem for distribusjon av drikkevann" skrevet av studentene Johan Haukalid, Jone Vassbø, Markus Raudberget og Peder Ward fra NTNU. Disse metodene vil bli referert til ved å oppgi kapittelnavn.

### Oversiktsbilde



Oversiktsbilde gir en god oversikt over forbruket ut fra vassverket og forbruket rundt om i de forskjellige sonene.

**Ramme 1** kan man se et oversiktsbilde over distribusjonsnettverket. Der vises alle strømningsmålerne rundt omkring i nettverket. Man ser også antall forbrukere i hver sone. FT indikerer "Flow transmitter" og WP indikerer "Water pipe". Tallene bak forkortelsene er et unik nummer for akkurat den strømningsmåleren eller røret.

**Ramme 2** viser informasjon om vannmengden ut ifra vassverket siste 24 timene. Med første panelet som viser nåverdi, deretter total mengde ut og maks og minimumsverdier.

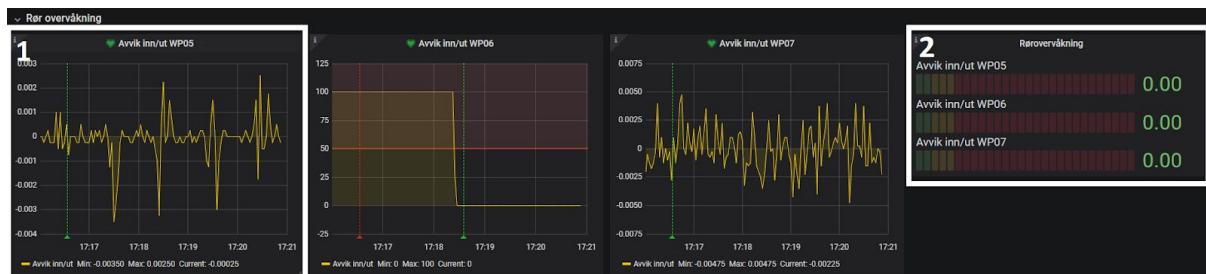
**Ramme 3** viser status over sonene og rør. Grønt hjerte og "OK" betyr at statusen er ok og ingen alarmer er aktivert. Et rødt hjerte og "ALERTING" betyr at det har en alarm for det aktuelle røret eller sonen. For å se hvilket alarmer som er forhåndssatt se "Forhåndssatte alarmer".

**Ramme 4** viser alarmlogg. Her vil alarmene legges inn med tidsstempel, slik at man kan gå tilbake for å se på alarmene som har vært aktiv.

**Ramme 5** viser antall avvik over avviksgrensen siste 24 timene. Dette er for lekkasjedeksjon for sonene. Hver sone sammenligner gjennomsnittlige timesverdier mot gjennomsnittlige timesverdier for dagene før. Dersom nåtidenes gjennomsnittlige timesverdier er over avviksgrensen, som er 10% av normal forbruk vil man få et avvik. Total kan man få 24 avvik gjennom en hel dag. For å forstå denne metoden for lekkasjedeksjons bedre anbefales det å lese kapittel "Lekkasjedeksjon basert på historisk data (LHD)".

**Ramme 6** viser oversikt av forbruket for hver sone. Der tallverdien viser nåverdi av strømning inn til sonene og grafene viser historien på forbruket for det aktuelle tidsrommet satt i Grafana. Fargene på grafene er satt til grønn under 300, gul mellom 300 til 600 og over 600 rødt. Dette er for å visualisere dersom det kommer endringer i forbruket.

## Rør overvåking

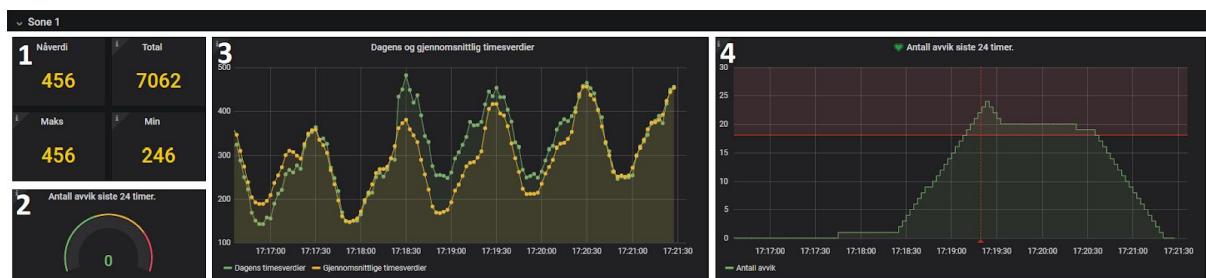


Raden “Rør overvåking” viser paneler som hjelper til og finne lekkasjer i rører

**Ramme 1** viser en graf av avviket mellom strømningen inn og ut i WP05, WP06 og WP07.. Dette er en metode for lekkasjedeteksjon på rør, og kan leses mer om i kapittel “Lekkasjedeteksjon med volumbalanse”. For WP06 kan man se at avviket er for stort, og en alarm har blitt aktivert. For å se mere om alarmer se “Forhåndssatte alarmer”. Den røde og grønne vertikale stiplede linjene viser når alarmen ble aktivert og når det var deaktivert.

**Ramme 2** viser også avviket mellom inn og ut for hvert rør i distribusjonsnettverket. Her er det valgt å visualisere dette i en “Bar” istedenfor. Her er avviket grønt dersom det holder seg under 2, gult mellom 2 og 4 og rødt over 4. Grunnen for at grønn er satt til 2 er fordi avlesningene på strømningsmålerne kan måle litt feil slik at det må være rom for litt variasjon før det blir sett på som unormalt, og antydet rødt og gult.

## Soneoversikt



Soneoversikt er et oversiktsbilde over den aktuelle sonen. På dette utkippet ser vi at vi har oversiktsbilde for sone 1. I dashbordet er det også slikt bilde for sone 2, 3 og 4 også.

**Ramme 1** viser informasjon for strømningen inn til sonene. Der det er et panel for nåverdi, totalverdi, maks verdi og minimumsverdi de siste 24 timene.

**Ramme 2** viser antall avvik over avviksgrensen de siste 24 timene. Denne er lik panelet som er forklart i ramme 5 på “Oversiktsbilde” bare at denne gjelder kun for sone 1.

**Ramme 3** viser dagens gjennomsnittlige timesverdier i grønn sammenlignet med gjennomsnittlige timesverdier for de siste dagene i gult for det aktuelle tidsrommet.

**Ramme 4** viser en graf av avvikene over avviksgrensen de siste 24 timene. Dersom det er totalt 18 avvik de siste 24 timene, så vil det aktivere en alarm. For å se mere om alarmer se ”Forhåndssatte alarmer”. Den røde og grønne vertikale stiplede linjene viser når alarmen ble aktivert og når det var deaktivert.

Ramme 3 og 4 bruker en lekkasjedeteksjonsmetode beskrevet bedre i kapittel ”Lekkasjedeksjon basert på historisk data (LHD)“.

## Alarmer

### Lage alarmer

Alarmer kan foreløpig bare settes i ”Graph“ panelet i Grafana. Etter å ha satt inn et ”Graph“ panel kan man sette opp alarmer ved å velge ”Alert“ på innstillingene for panelet.



Det er flere muligheter på hvordan man vil at alarmen skal aktiveres i Grafana. For eksempel har det blitt satt opp en alarm på bildet under. Her har det blitt valgt at alarmen skal sjekkes for hvert sekund og den trenger ikke å være aktiv lengre enn over null sekund. Alarmen blir aktivert dersom maksverdien for denne grafen er over 50 mellom nå og 10 sekunder tilbake.

**Dashbordet må lagres dersom det legges til en ny alarm.**

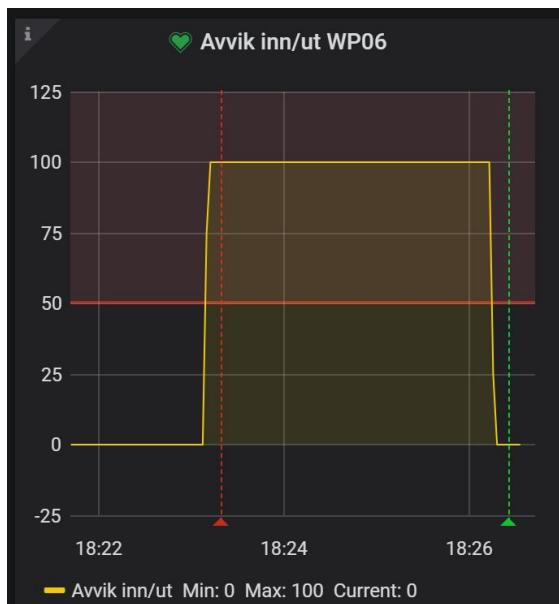
### Status for alarmer

Alarmene har i hovedsak to statuser, ”OK“ og ”Alerting“. ”OK“ indikerer at alarmen ikke er aktivert og ”Alerting“ indikerer at alarmen er aktivert.



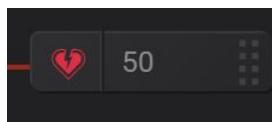
I tillegg kan alarmene ha to til tilstander, ”No data“ og ”Pending“. ”No data“ betyr at panelet ikke inneholder noe data. ”Pending“ betyr at alarmen er aktivert, men det er satt i innstillingene på at det skal være aktivert i f.eks 10 sekunder før den blir i status ”Alerting“.

Statusen på alarmene er også illustrert på grafen i panelet. Her er det en rød stiplet linje dersom alarmen blir aktivert og en grønn stiplet linje om den blir deaktivert. Se bilde under.



## Flytte alarmgrenser

Ønsker man å flytte alarmgrensene til en alarm som allerede er laget kan man trykke på navnet på panelet og velge "Edit" og deretter "Alert". Da vil man få opp en slider på høyre side av grafen der man kan endre på alarmgrensen.



## Kvittering alarmer

Det er ingen form for kvittering av alarmer i Grafana. Alarmen vil kvittere seg selv dersom alarmen ikke lenger er aktiv. Alarmen vil vises i alarmloggen selv om den ikke lenger er aktiv.

## Forhåndssatte alarmer

I dashbordet som har blitt laget for prosjektet "Styre- og overvåkningssystem for distribusjon av drikkevann" er i hovedsakelig to alarmer som er forhåndssatt.

### Alarm for lekkasjedeteksjon i sone

Denne alarmen aktiveres dersom det er en lekkasje i sonen. Metoden baserer seg på LHD metoden, beskrevet i "Lekkasjedeteksjon basert på historisk data (LHD)". Grafana tar inn total avvik siste 24 timene som blir utregnet av denne metodene og fremviser dette i en graf. Det er satt opp en alarm som aktiveres dersom det totale avviket kommer over alarmgrensen.

### Alarm for lekkasje i rør

Denne alarmen aktiveres dersom det er en lekkasje i et rør. Metoden baserer seg på volumbalansemetodene beskrevet i kapittel "Lekkasjedeteksjon med volumbalanse". Grafana tar inn avviket som blir utregnet av denne metodene og fremviser dette i en graf. Det er satt opp en alarm som aktiveres dersom avviket kommer over alarmgrensen.

## **I.4 Brukermanual e!COCKPIT**

# Brukermanual Brukergrensesnitt e!COCKPIT VISUALIZATION

2020

## Innholdsfortegnelse

<b>Introduksjon</b>	<b>3</b>
<b>Teknisk</b>	<b>4</b>
<b>Forklaring av skjermbilder</b>	<b>4</b>
Tittelbar	4
Prosessbilder	5
System Overview/System Oversikt	5
Leak Detection/Lekkasjedeteksjon	6
Grafana	6
Flood Control/Flomsikring	7
Forbruketrykk	8
Trends/Trends	9
Alarms/Alarmer	10
Widgets	11
Popups	11
Meny	11
Regulatorpopup	12

## Introduksjon

Denne brukermanualen skal gi operatøren tilfredsstillende kunnskaper for å tolke og operere brukergrensesnittet mot et kjørende “Control and monitoring system for distribution of drinking water”-systemet.

Brukergrensesnittet er i hovedsak designet for å stå på messer.

## Forklaring av skjermbilder

Bildene er bygd opp i tre deler. Tittelbar, prosessbilde og widgetlinje. For å forklare visualiseringen er funksjoner bundet opp med et tall, slik at man lettere kan refere mellom tekst og bildet.

### Tittelbar



Tittelbaren blir forklart en gang, siden den går igjen i alle bildene, bare at tittelen endrer seg.

1 - Menyknapp, åpner en egen popup for meny slikt operatøren kan manøvrere seg mellom bildene.

2 - Tittelen på valgt bildet.

3 - Alarmbjelle og alarmtekst. Bjellen er normalt hvit, men ved nylig initiert alarmer blir denne rød, som vist i eksemplet.

4 - For å velge mellom brukerrettigheter, kan operatørene logge seg inn med sin unike ID. Denne funksjonen er ikke satt i bruk i, siden det er en demonstrasjonrigg og er ment kun for å visualisere at funksjonen er kan legges til.

5 - Brukergrensesnittet kan velge mellom to språk, engelsk og norsk. Tallet opp til høyre for språkvalgene er brukergrensesnittets versjonsnummer.

### Prosessbilder

Det er totalt fire unike prosessbilder. Disse blir forklart løpende, på samme måte som tittelbaren.

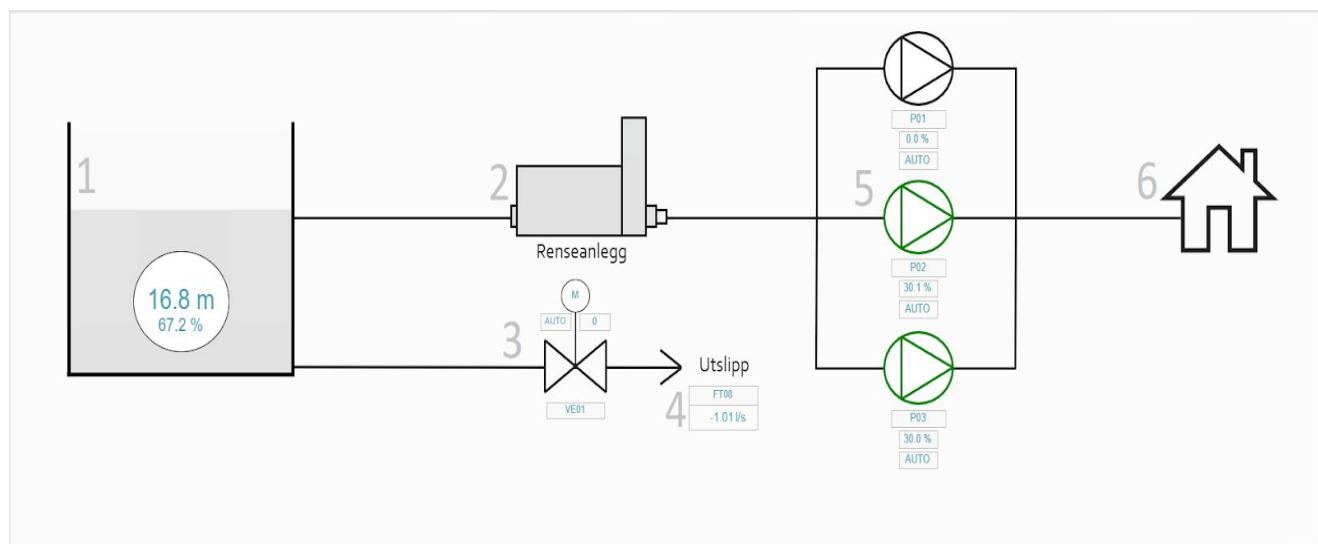
#### System Overview/System Oversikt

Bildet skal operatøren et kjapt inntrykk av tilstanden i alle systemene

1 - Vannkilden, indikerer nåværende nivå i antall meter og %. Ved å trykke på kilden får operatøren tilgang på sensorinnstillinger.

2 - Indikasjon for renseanlegg. Renseanlegget er ikke en del av pakken som prosjektet leverer, men er likevel tegnet inn for å få helhet i systemet.

3 - Reguleringsventil på utslipp. Viser modus,tagnavn og utgangsverdi. Ved å trykke på ventilen får operatøren tilgjengelig en popup som gir muligheter for å endre på ventilinnstillinger. Ved feil



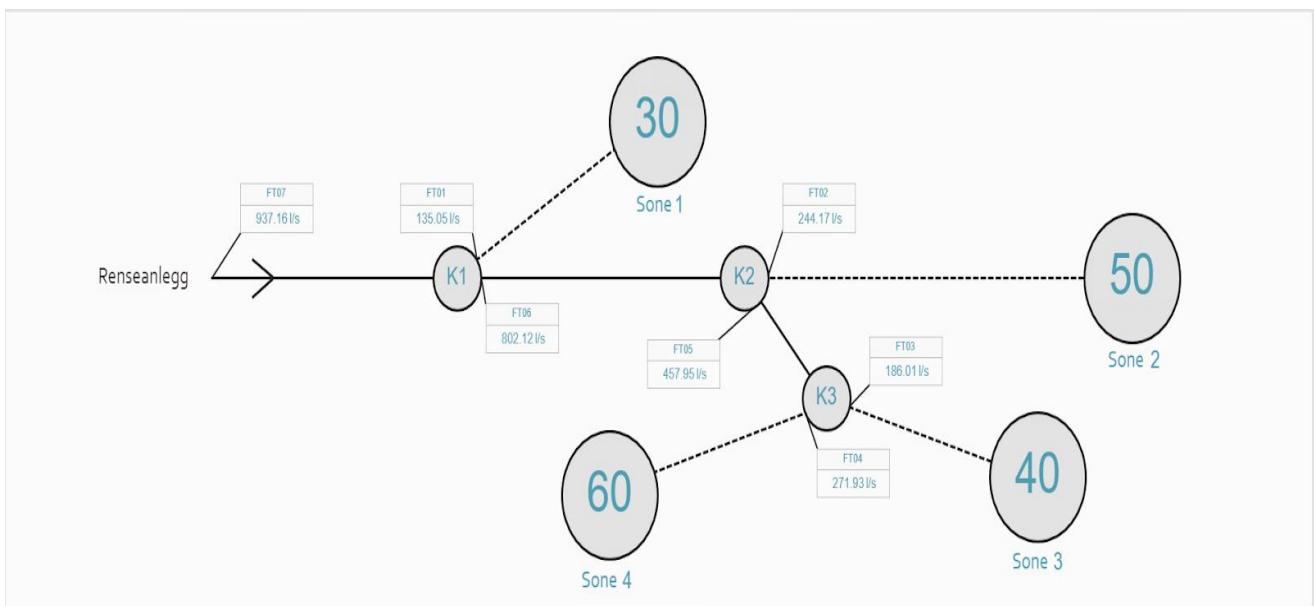
så blir ventilen rød, med rødt fyll i "pilen".

4 - Utslippsindikering. Strømningsindikering på vann som strømmer ut av vannkilden som flomforhindrende tiltak. Ved å trykke på måleren får operatøren opp en meny for sensorinnstillinger.

5 - Viser pumpene i trykkregulering, åpner et vindu for å manipulere pumpetilstand. Pumpene indikerer status med et svart symbol når pumpen ikke kjører, mens den lyser grønt når den kjører. Ved feil så blir pumpen rød, med rødt fyll i "pilen".

## Leak Detection/Lekkasjedeteksjon

Lekkasjedeteksjonsbildet gir operatøren et inntrykk av hvordan mengdene strømmer. Hovedfunksjonen ligger på det tilkoblede bildet operatøren kan manøvrere seg til ved hjelp av å trykke på haken på tittelbaren, denne bringer operatøren til en peker mot Grafana, som viser grafer.



Bildet ovenfor viser hva som visualiseres i e!COCKPIT. Dette bildet viser kun tilstanden på sensorene i lekkasjedeteksjonen. K1, K2 og K3 er navnet på kummene som batteripakkene og strømningsmålerne står i.

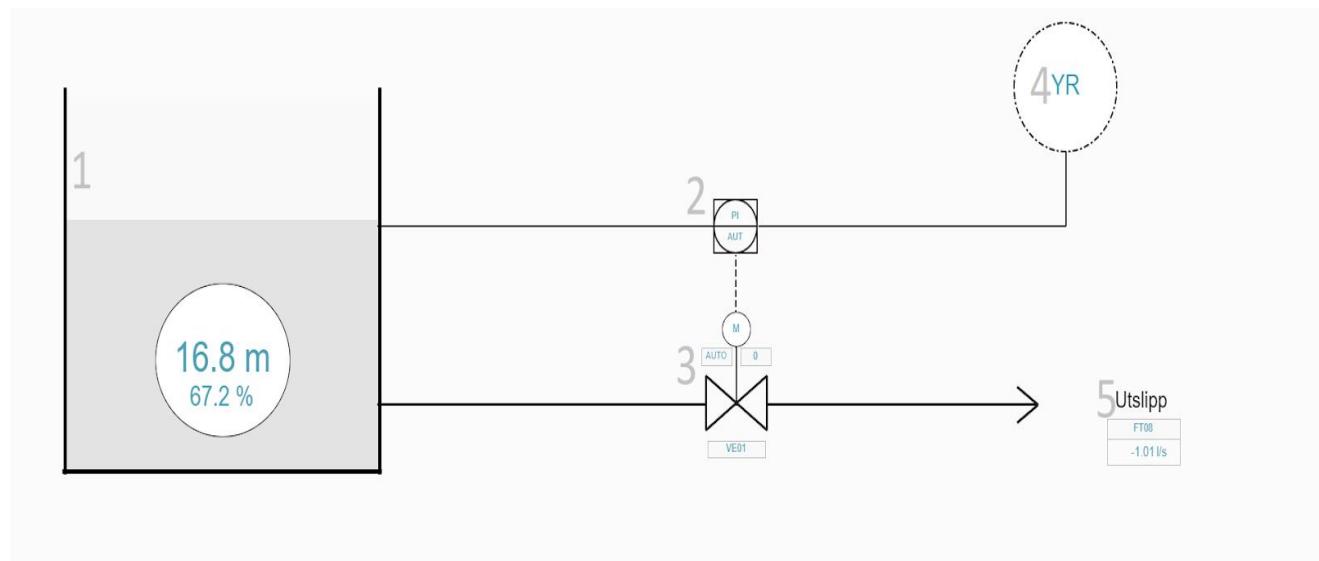
Sifferet i de store sirklene er antall forbrukere.

I dette bildet får operatøren indikasjoner om det er feil på målerne og et inntrykk av hvordan forbruket er på nettet.

## Grafana

Se egen manual.

## Flood Control/Flomsikring



1 - Vannkilde, indikerer nivået i vannkilden i prosent og meter. Ved trykk åpnes sensorinnsillingene for nivåsensoren.

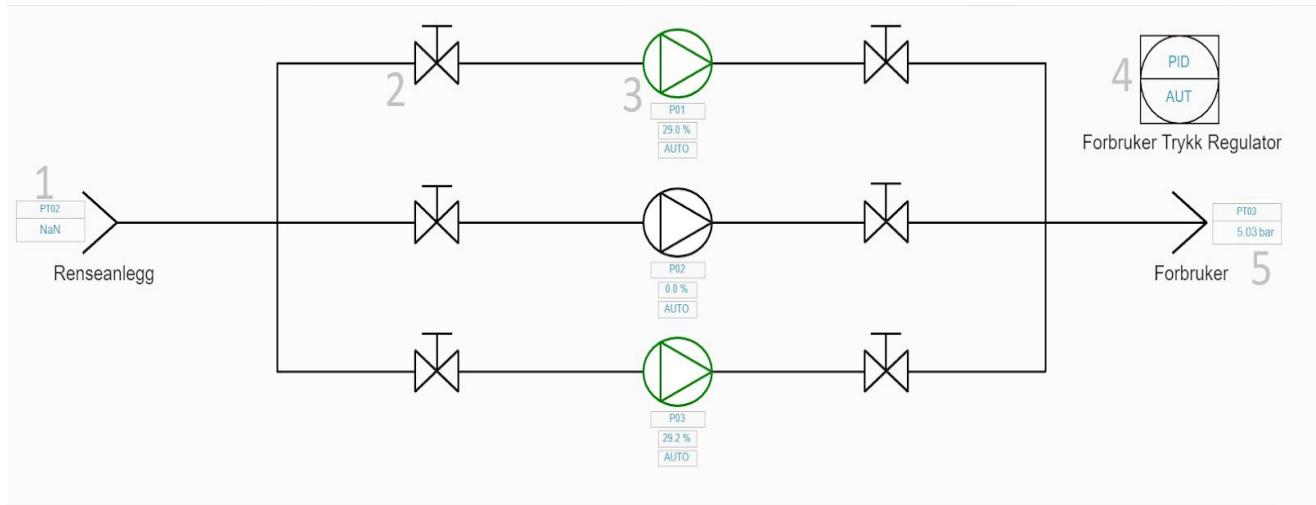
2 - Regulatorsymbol. Indikerer om regulatoren står i manuell eller auto, viser i tillegg hvilke led av P, I og D som er aktive. Ved trykk åpnes regulatorinnstillingene for valgt regulator.

3 - Regulatorventil. Viser om ventilen er i auto eller manuell, tag og utgangsverdi til ventilen. Ved trykk åpnes ventilinnstillingene.

4 - Bare et symbol for å vise at regulatoren får verdier fra YR. Ikke et objekt som er interaktivt.

5 - Strømningsmåling på vann ut av vannkilden. Ved trykk åpner sensorinnstillingene for sensoren.

## Forbrukertrykk



1 - Trykkmåling på vannet fra renseanlegget. Ved trykk åpnes sensorsinnstillingene for trykksensoren.

2 - Symbol for indikering av håndventil. Ikke interaktiv i brukergrensesnittet.

3 - Pumpesymbol. Grønn ved drift, svart ved stopp og rød ved feil. Viser tag, utgang og status. Ved trykk åpnes pumpeinnstillingene.

4 - Regulator. Viser om regulatoren står i manuell eller auto, hvis regulator står i auto indikerer den også PID valgene. Ved trykk åpnes regulatorinnstillingene.

5 - Samme som 1, bare for trykket ut fra pumpene og til forbrukernettet.

## Trends/Trends

Et bilde hvor operatøren får tilganger på variabler grafisk over en tidsakse.



1 - Setter tidsperspektivet på trenden.

2 - Valgknapper for mulige verdier som kan visualiseres.

3 - X-akse manøvrering.

## Alarms/Alarmer

For å kvittere en bestemt alarm velger man alarmen i alarmlisten og Kvitterer Valgt/Acknowledge Selected.

For å kvittere alle alarmer, trykker man Kvittere Alle/Acknowledge All

Alarmene er fargekodet ut fra statusen på alarmen.

Er teksten **rød** vil det si at det er en nylig initiert alarm, som er aktiv.

Er teksten **blå** vil det si at alarmen er ukvittert og er inaktiv.

Er teksten **gul** vil det si at alarmen er aktiv og kvittert.

Alarmlisten inneholder også hendelser, som ikke blir kvittert ved kvitter alle/acknowledge all funksjonen.

## Widgets

De fire firkantene ved bunnen av bildene blir kalt widgets. Disse består av trace, alarm og status widgeter.

Alarm widgetene er unike ut ifra hvilket bildet de er plassert. På de unike systemene vises kun alarmene tilhørende systemet, mens på System Oversikt/System Overview vises alarmene på tvers av systemet.

Eksempler på disse widgetene:

Batteri Status			
Sone	Nivå	Tid Igjen	Tilstand
K1	14.31 Ah	9.18 h	Lader
K2	12.32 Ah	12.39 h	Lader
K3	17.23 Ah	4.62 h	Lader

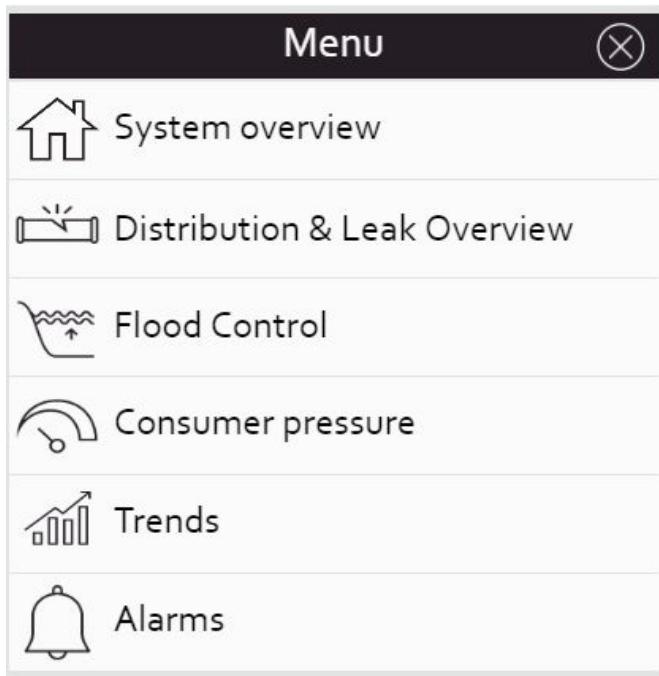
Visualiserer batteristatusene i sonene. Hvordan nivået er i batteriet, hvor lenge igjen før batteriet utlades eller oppladet, alt etter hva tilstanden er.

Pumper		
Pumpe	Current	Uptime
P01	0.00 A	5291 t
P02	2.41 A	5290 t
P03	2.34 A	5290 t

Pumper: Visualiserer navnet til pumpen, strømtrekk og hvor lenge pumpen har vært i drift.

## Popups

### Meny



Menyen viser tilgjengelige bilder i HMI'en.

System overview skal gi operatøren et helhetsinntrykk av systemet.

Distribution & Leak overview skal gi operatøren tilstand i lekkasjedeteksjonssystemet og viser til en peker mot Grafana som har algoritmen for lekkasjedeteksjonen.

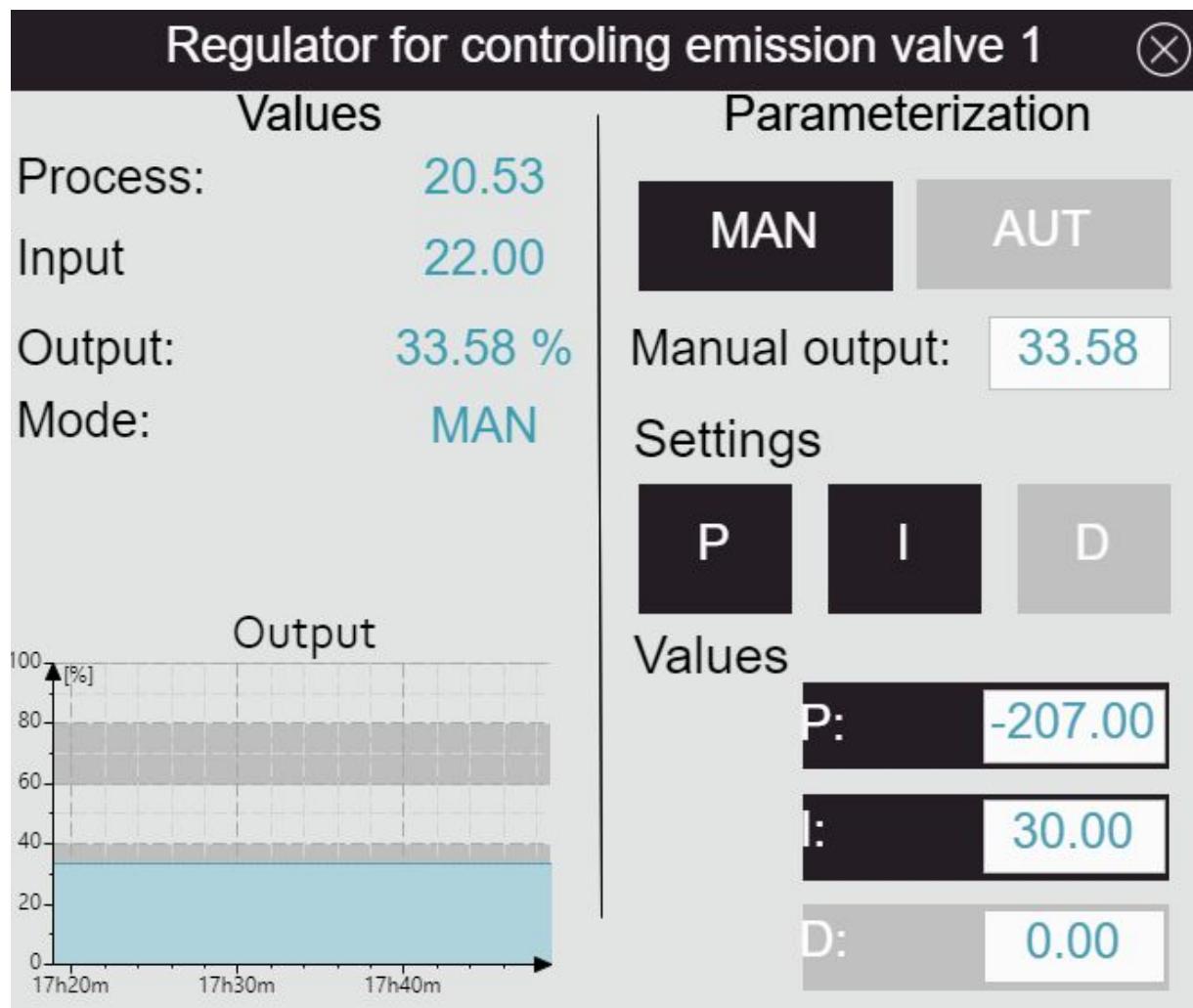
Flood control er alle tilstander og pådrag som påvirker det flomforhindrende systemet.

Consumer pressure er reguleringen på trykket i vannet som går til forbrukerne.

Trends gir operatøren muligheten å studere variabler over en gitt tid i form av en graf.

Alarms er en samlet alarmliste som gir operatøren av alarmer og hendelser på tvers av systemene.

## Regulatorpopup

**Values:**

Process: Viser prosessverdien.

Input: Viser hva inngangen er.

Output: Viser va regulatoren utgang til pådragsorganet er.

Mode: Viser om regulatoren står i auto eller manuell.

Trace Output: Viser utgangens verdi over en time.

**Parameterization:**

MAN/AUT: Knapper for å velge modus. Hvis det er en sort bakrunn, indikerer det at modusen er valgt. Er knappen "grået" ut, viser det at den ikke er valgt.

Manual output: Variabel for å styre utgangen manuelt.

Settings: Her velger man PID parametere. Sort bakrun indikerer valgt ledd.

Values: Input for endirng av parameterverdi