BACHELOROPPGAVE

TITTEL:
Prediktivt vedlikehold

KANDIDATNUMMER(E):			
10010, 10025, 10030			
DATO:	EMNEKODE:	EMNE:	DOKUMENT TILGANG:
20.05.19	IE303612	Bacheloroppgave	
STUDIUM:		ANT SIDER/VEDLEGG:	BIBL. NR:
Bachelor i ingeniørfag - Data		58/8	

VEILEDER(E):
Kjell Inge Tomren

SAMMENDRAG:
I denne rapporten beskrives mulighetene for å bruke PdM (Predictive maintenance) som system for å oppdage feil på transportører produsert av Optimar AS. Formålet har vært både å finne feil før noe går galt samt avklare hvorfor en feil har skjedd. Gjennomføringen av oppgaven er utført i samarbeid med Optimar. For å skape feilstatistikk har vi laget et oppsett for datalogging og databehandling. Ved bruk av en Optimar transportør, en PLS og Docker, har det blitt satt opp et system for datalogging og loggført data har blitt brukt til å trene opp et nevralt nettverk. For visualisering ble det blitt laget en mobilapp og en nettside som gir informasjon videre til brukeren. Konklusjonen til gruppen er at en del typer feil kan oppdages ved å bare logge strømtrekket til motoren. Resultatet av prosjektet er et system som kan gi beskjed til brukeren når noe er feil med transportøren, men kan også utvides til å varsle før noe har gått galt.

Denne oppgaven er en eksamensbesvarelse utført av studenter ved NTNU i Ålesund.

Postadresse	Besøksadresse	Telefon	Telefax	Bankkonto
Høgskolen i Ålesund	Larsgårdsvegen 2	70 16 12 00	70 16 13 00	7694 05 00636
N-6025 Ålesund	Internett	Epostadresse		Foretaksregisteret
Norway	www.hials.no	postmottak@hials.no		NO 971 572 140

FORORD

Dette er vår bacheloroppgave om prediktivt vedlikehold.

Vi vil gjerne rette litt oppmerksomhet mot Optimar for fantastisk samarbeid gjennom hele oppgaven. Selv om ikke alt ikke har gått helt etter planen har dere alltid stilt opp med ressurser når vi har hatt behov for det.

Spesielt takk til Ole Andre Tomren, foruten deg hadde det ikke blitt noen oppgave. Vil også gi en spesiell takk til Marius Nedregård, Emil Dale Bjørlykhaug og Daniel Kvam som har vært til stor hjelp til oppsett av PLS og teori bak Anomaly Detection.

Ellers vil vi gjerne takke for oss for tre flotte år på NTNU Ålesund. Takk til alle lærere, administrasjonen og medelever som har bidratt til en fantastisk studietid og et fantastisk miljø for læring og sosialisering.

INNHOLD

Liste over figurer

SAMMENDRAG

I denne rapporten beskrives mulighetene for å bruke PdM (Predictive maintenance) som system for å oppdage feil på transportører produsert av Optimar AS. Formålet har vært både å finne feil før noe går galt samt avklare hvorfor en feil har skjedd. Gjennomføringen av oppgaven er utført i samarbeid med Optimar. For å skape feilstatistikk har vi laget et oppsett for datalogging og databehandling. Ved bruk av en Optimar transportør, en PLS og Docker, har det blitt satt opp et system for datalogging og loggført data har blitt brukt til å trene opp et nevralt nettverk. For visualisering ble det blitt laget en mobilapp og en nettside som gir informasjon videre til brukeren. Konklusjonen til gruppen er at en del typer feil kan oppdages ved å bare logge strømtrekket til motoren. Resultatet av prosjektet er et system som kan gi beskjed til brukeren når noe er feil med transportøren, men kan også utvides til å varsle før noe har gått galt.

TERMINOLOGI

Begreper

AI (Artificial intelligence): teknologien som gjør at datamaskiner er i stand til å løse samme tankeoppgaver som mennesker.

Deep learning: Deep learning er en type maskinlæring bygd opp av flere lag for å kunne håndtere komplekse problem.

Artificial neural network (ANN): Et nettverk av kunstige neurons, laget for å løse et problem.

Recurrent neural network (RNN): En type ANN med et internt minne for å kunne håndtere en sekvens med data.

Long short-term memory: Long short-term memory (LSTM) er en RNN arkitektur brukt for deep learning.

Convolutional neural network (CNN/ConvNet): Convolutional neural network er en type deep neural network.

Datasett: En samling med data. Ofte hentet fra en database.

Prediktivt vedlikehold (Predictive maintenance): En teknikk som er utformet for å avgjøre tilstanden til driftsutstyr for å kunne estimere når vedlikehold skal utføres.

Unsupervised learning: Unsupervised learning algoritme trenger kun data som input, algoritmen finner struktur i dataene, og kan derfor lære uten at data er klassifisert.

Avvik gjenkjenning (Anomaly detection): Identifikasjon av datapunkter, gjenstander, observasjoner eller hendelser som ikke samsvarer med det forventede mønsteret.

PLS: Programmerbar logisk styring. En datamaskin som brukes til å automatisere oppgaver som produksjon og kontroll.

Versjonskontrollsystem: Et system som holder styr på forandringer i filer over tid. Ved bruk av et slikt system kan du hente opp igjen spesifikke versjoner ved et seinere tidspunkt.

GIT: Versjonskontrollsystem for sporing av endringer i koden under programvareutvikling. Kan brukes for å koordinere arbeidet blant programmerere.

MQTT (Message Queue Telemetry Transport): En ISO standard publisering- og abonneringsbasert meldingsprotokoll. Fungerer på toppen av TCP/IP protokollen og er designet for enheter som har begrenset nettverksbåndbredde, som IoT-enheter og sensorer.

Agile: Agile arbeidsmetodikk omhandler å være åpen for og legge til rette for endringer underveis i prosjektet.

Back-end: Back-end er ofte ansvarlig for å behandle og lagre data. Brukeren har ikke direkte kontakt med denne delen av systemet.

Front-end: Den delen av et datasystem som brukeren samhandler med.

Forkortelser

UML - Unified Modeling Language

IDE - Integrated Development Environment

PdM - Predictive Maintenance

XP – Extreme programming

AI – Artificial Intelligence

NN – Neural network

ANN – Artificial neural network

RNN – Recurrent neural network

CNN – Convolutional neural network

LSTM – Long short-term memory

MQTT - Message Queue Telemetry Transport

PLS – Programmerbar logisk styring

AD – Anomaly Detection

1 INNLEDNING

Oppgaven går ut på å lage et prediktivt vedlikeholdssystem for frekvensstyrte motorer som i hovedsak benyttes av fiskeindustrien. Gruppen vil i første omgang fokusere på motorer brukt i transportører men med mulighet for i fremtiden (etter endt bachelor) utvide til andre motorer. Oppgaven vil bli utført for Optimar og er relatert til systemet Optimar Commander. Optimar Commander er laget for at man skal ha bedre kontroll over produksjon og prosesser, man kan enkelt få oversikt og statistikk som er bearbeidet av data fra de aktuelle anleggene. Commander vil i fremtiden ha støtte for prediktivt vedlikehold, men Optimar ønsker allerede nå å utforske hvordan dette lar seg gjøre i praksis.

1.1 Problemstilling

Problemstillingen går ut på å sjekke om det er mulighet for å avdekke feil, mulige feil og eventuelt kommende feil kun ved å lese strømtrekket fra en transportør ved bruk av AI. Er det mulighet å avdekke mangel på service? Feil gjennomført service? Og eventuelt feil belastning i forhold til spesifikasjoner som har blitt angitt.

1.2 Utfordringer

Utspringet til oppgaven startet med bedriftenes ønske om 100% oppetid av utstyr for fiske, landbruk og produksjon. Oppdragsgiveren Optimar ønsker på sikt å kunne tilby dette til kundene og de ønsker derfor å ta i bruk teknologi som kan tilrettelegge for dette i fremtiden. I første omgang vil oppgaven bestå av å teste muligheten for feilsøking og feilestimering ved bruk av AI. Det som har blitt tilrettelagt for problemstillingen er muligheter for å avdekke fremtidige feil ved hjelp av en transportør og signalene som kan hentes ut fra denne.

En av de store utfordringene med oppgaven har vært å avklare om det i det hele tatt er mulig å oppdage feil kun ved hjelp av strømmåling. I tidligere prosjekter har de funnet feil ved hjelp av sensorer i samhandling med trekk av strøm.

2 TEORETISK GRUNNLAG

2.1 Maskinlæring

Maskinlæring handler om at en maskin lærer basert på erfaring, eksempel og av analogi. Et system som kan lære vil ha mulighet til å utvikle et mer intelligent system over tid. De mest vanlige tilnærmingene til maskinlæring er kunstige nevrale nettverk og genetiske algoritme. (Michael Negnevitsky)

2.1.1 Historie

I over 2000 år har filosofer prøvd å forstå og svare på to store spørsmål: Hvordan fungerer hjernen til menneske, og kan ikke-menneske ha hjerne? (Michael Negnevitsky)

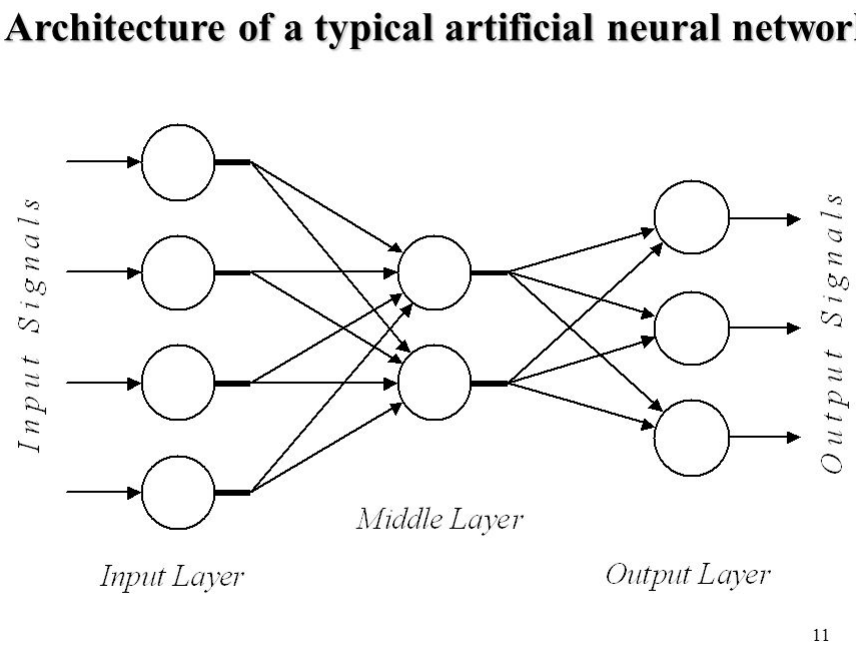
Det er tre "forfedre" til kunstig intelligens. Den første var Alan Turing. Han skrev en av de første og viktigste papirene om maskinlæring, Computing machinery and intelligence. Selv om det ble skrevet i 1950 er Turings fremgangsmåte fortsatt relevant. Alan Turing hadde en sentral rolle i knekking av den Tyske enigma koden under andre verdenskrig. Senere stilte han mange spørsmål som ledet opp til et fundamentalt spørsmål i maskinlæring: Kan en datamaskin tenke? Turing svarte aldri på spørsmålet, men i stedet lagde han et spill kalt The Turing Imitation game. (Michael Negnevitsky)

Warren McCulloch blir sett på som den andre forfaderen til kunstig intelligens. McCulloch sammen med Walter Pitts beskrev en modell for et kunstig nevralt nettverk der hver nevron hadde en binær tilstand, på eller av. Beskrevet i McCulloch and Pitts, 1943. (Michael Negnevitsky)

Den tredje grunnleggeren av maskinlæring var John von Neumann. Den ungarske matematikeren var kollega og venn av Alan Turing og var en nøkkelperson i Manhattan Project da de bygde atombomben. Neumann var med på å bygge den første nevrale nettverk maskinen i 1951. Maskinen var sterkt påvirket av McCulloch og Pitts nevrale nettverksmodell. (Michael Negnevitsky)

2.1.2 Kunstig nevralt nettverk

Michael Negenevitsky sier i læreboken Artificial Intelligence: a guide to intelligent systems at et nevralt nettverk kan defineres som en modell basert på menneskehjernen. Nettverket består av et sett med nevroner eller basic information-processing units. Selv om hver enkelt nevron har en simpel struktur, kan et sett med nevroner håndtere relativt avanserte problem. Dagens nevrale nettverk ligner like mye på hjernen som et papirfly ligner på et jetfly, likevel har nevrale nettverk evnen til å lære. Et nevralt nettverk kan lære ved at hver kobling imellom nevroner har en verdi for vekt. Vekten representerer graden av viktighet for hver input, nettverket lærer ved å oppdatere disse vektene. (Michael Negnevitsky)



Figur 1 Oversikt over hvordan et typisk nevralt nettverk ser ut

Læreboken Artificial Intelligence: a guide to intelligent systems sier at en enkel nevron består av et eller flere innkommende signal, vekter til signalene, en aktiveringsfunksjon og utgående signal. De innkommende signalene kan være rå data som en piksel i et bilde eller det kan være det utgående signalet ifra en annen neuron i nettverket. Vektene blir brukt til å bestemme hvor viktig hvert innkommende signal er. De 4 mest brukte aktiveringsfunksjonene er Step, Sign, Sigmoid og lineær

funksjon. Det utgående signalet vil enten være en del av svaret eller fortsette som input til en eller flere nevroner som kommer etter i nettverket. (Michael Negnevitsky)

$$Y = \phi\left(\sum_{i=1}^{n} x_i \; w_i - \; \theta\right)$$

Formel 1: Formel for en kunstig nevron

Y = output signal

φ = aktiveringsfunksjon

n = antall innkommende signal

x_i = verdi av innkommende signal i

ω_i= vekten til innkommende signal i

θ = bias

2.1.3 Prediktivt vedlikehold

Prediktivt vedlikehold er ifølge *Machine Learning for Predictive Maintenance: a Multiple Classifier Approach* en metode for å bearbeide vedlikeholds problematikk. På grunn av det økende behovet for å minimere kostnader og nedetid, er PdM en smart måte å håndtere vedlikehold. Bruk av PdM gjør at det dukker opp flere utfordringer, en av de er det å generere kvantitative indikatorer på statusen til et system, som er knyttet til et gitt vedlikeholdsproblem. Like vanskelig kan det være å bestemme systemets forhold til driftskostnader og feilrisiko.

2.1.4 Anomaly Detection

Anomaly Detection (AD) er et viktig problem som det har vært forsket mye på. Det finnes flere forskjellige typer av AD, fra noe så enkelt som at alt utenfor en normalverdi blir sett på som en anomaly til avanserte analytiske deep learning algoritmer. Tre hovedtyper for AD er unsupervised anomaly detection, supervised

anomaly detection og semi-supervised anomaly detection som utdyper seg i hvordan en AI ser på data. [31]

2.2 Systemutvikling

2.2.1 Agile

Wikipedia beskriver Agile utviklingsmetodikk som en måte å utvikle software på som legger til rette for og fremmer endring underveis i et utviklingsprosjekt. Utviklingsmetodene er iterative og jobbes med i inkrement. (Wikipedia)

The Agile Manifesto forklarer agile med følgende hovedpunkt:

"Individuals and interactions over processes and tools
Working software over comprehensive documentation
Customer collaboration over contract negotiation
Responding to change over following a plan" [6]

2.2.2 XP Extreme Programming

Extreme Programming er en agil utviklingsmetodikk som fremmer smidighet over alt. Verdiene til Extreme Programming er kommunikasjon, enkelhet, tilbakemelding, respekt og modig. To av de viktige prinsippene for å kunne bruke Extreme Programming er om prosjektet har dynamisk endrende krav og om det har risiko for forsinkelser fra andre leveranser. [7]

2.2.3 Arkitektur patterns

Broker pattern

Towardsdatascience.com forklarer broker pattern som et strukturert distribuert system med dekopled komponenter. En broker passer på kommunikasjonen imellom enheter i systemet. En eller flere servere publiserer til en broker som videresender informasjonen til klienter.

Client-server pattern

Client-server pattern består ifølge towardsdatascience.com av en server og flere klienter. Serveren tilbyr tjenester som klientene kan spør etter. Serveren har mulighet til å ta imot spørringer flere klienter samtidig. Og fortsetter å vente på spørringer selv om den tilbyr en tjeneste til en klient.

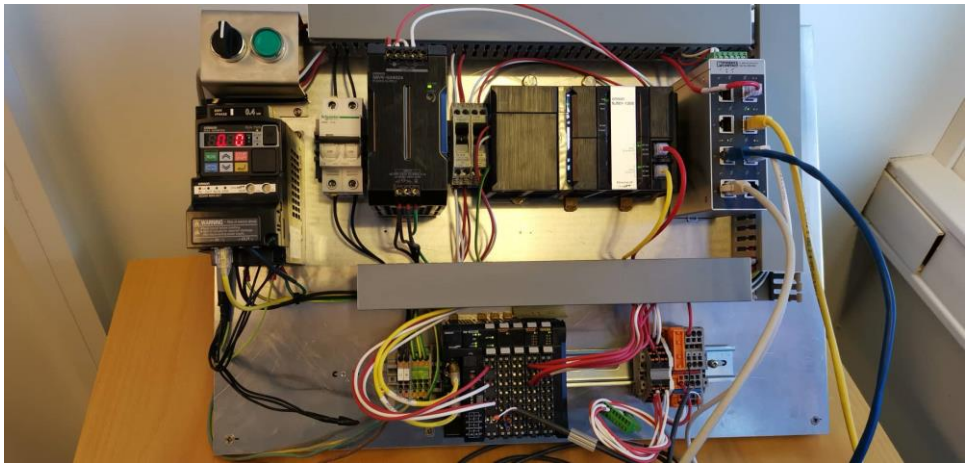
3 MATERIALER OG METODE

3.1 Materialer

3.1.1 PLS

Programmerbar logisk styring er en simpel datamaskin som blir brukt til styring og kontroll av prosesser. I prosjektet blir den brukt til å styre en frekvensbasert motor. Dagens industrisamfunn bruker PLS til å styre alt fra samleband på fabrikker til forskjellige elektriske prosesser i biler. [8]

3.1.2 Oppsettet av kontrollenheten



Figur 2 Oppsettet av kontrollenheten

3.1.2.1 Omron MX2 Inverter

Motorkontroller med innebygd logisk programmering. 200V 0,4kW. [9]

3.1.2.2 Omron S8VK-G Strømforsyning

En 24V strømforsyning fra Omron. Laget for å overleve i ekstreme omgivelser, eksempelvis på båter. [10]

3.1.3 Transportør

Prototypen av en transportør er bygd etter industristandarden i rustfritt aluminium og selve rammeverket i riggen skal tåle stor belastning. Rammen består av plater, rør, tre tverrstag og brak for motoroppheng. Oversiden av rammen er kledd med fire glidelister i solid PEHD med glatt overflate for en jevn bevegelse under last. Selve samlebåndet er laget i PET og tåler stor belastning i både press og trekk. I begge ender av transportøren sitter det en aksling, hvor den ene er en drivaksling. Begge akslinger sitter fast i glidelager som igjen er festet i et justerbart stag for justering av slark på båndet. På akslingene sitter det totalt 7 tannhjul konstruert i PEHD for bevegelse av båndet, 4 på drivakslingen og 3 på akslingen. Motoren på transportøren er en TpNordic 50hz motor som kan levere opp til 0,44KW i trekkraft. For overføring av kraft fra motoren til drivakslingen brukes det en snekke. Snekken er montert spesielt for en enkel demontering og montering av ny snekke. Transportøren står på 4 justerbare ben for justerbar arbeidshøyde som kan monteres på et annet objekt om ønskelig.

Item	Qty	Stock Number	Description	Note	Length	Width	Part Number	Material	Mass
23	1	62935-024-11	Drive shaft				62935-024-11		0
22	1	62935-024-10	Frame				62935-024-10		5,8
21	4	55052	Bein justerb. M75-16-150 u/Feste				55052	AISI 304	2,2
20	1	44051	Kilestål RF 8x7 SS2350		70		44051-70	SS2350	0
19	1	42801-26	Worm Gear VSF 050 i=26,P/FB/C/Ø/B14 Ø=25 Coated				42801-26	Alu.	2,7
18	1	40104	Motor elektrisk 0.37kW 4P B14 IP56				40104	SS 2333	22,3
17	1	29005	M8 Fjærskive DIN 127B A4				29005	Steel, Mild	0
16	3	29003	M6 Fjærskive DIN 127B A4				29003	Steel, Mild	0
15	3	29002	M6 Underlagskive DIN 125 A4				29002	Steel, Mild	0
14	8	24161	Mutter M16 DIN 934 A4 80				24161	A4	0,3
13	1	21084	Skru M8 x 25 6Kt DIN 933 A4				21084	A4	0
12	3	21065	Skru M6 x 16 6Kt DIN 933 A4				21065	A4	0
11	4	21-100041	Glidelist PEHD 1000 30x25		705		62935-024-00-01	PEHD1000	2
10	1	204-0300	S.Belt S50-801 PE/Blue W=300		2500		62935-024-00-02	PM Blue	4,8
9	4	20-102113	Boss M/strammeskru Ø50 L=150				20-102113		2,4
8	4	20-101710	Lager Glide Ø50/30X35 Pet-Tx				20-101710	PET	0,2
7	8	20-101601	Mutter M20 m/Kon				20-101601	Generic	0,1
6	8	20-101600	Hylse M/Krans Ø35/Ø28/Ø21				20-101600	Generic	0
5	1	20-101417	Aksel Vende B300,Lager 206				20-101417	SS 2333	4,8
4	4	12-41504	Distansestykke - aksling 40x40				12-41504		0,3
3	1	12-40865	Festeskive for Snekkev. t.o.m 063 Ø40xØ8.4				12-40865	AISI 304	0
2	1	12-31940	Motorfundament VSF 050				12-31940	AISI 304	0,7
1	7	11527	S.Belt 6T-Hjul S50-801 40x40				11527	-	0

MASS 48 kg

Drawn according to General Tolerances:
NS-ISO 2768-1 Class m
EN-ISO 13920 Class 1B, 2A and 3F

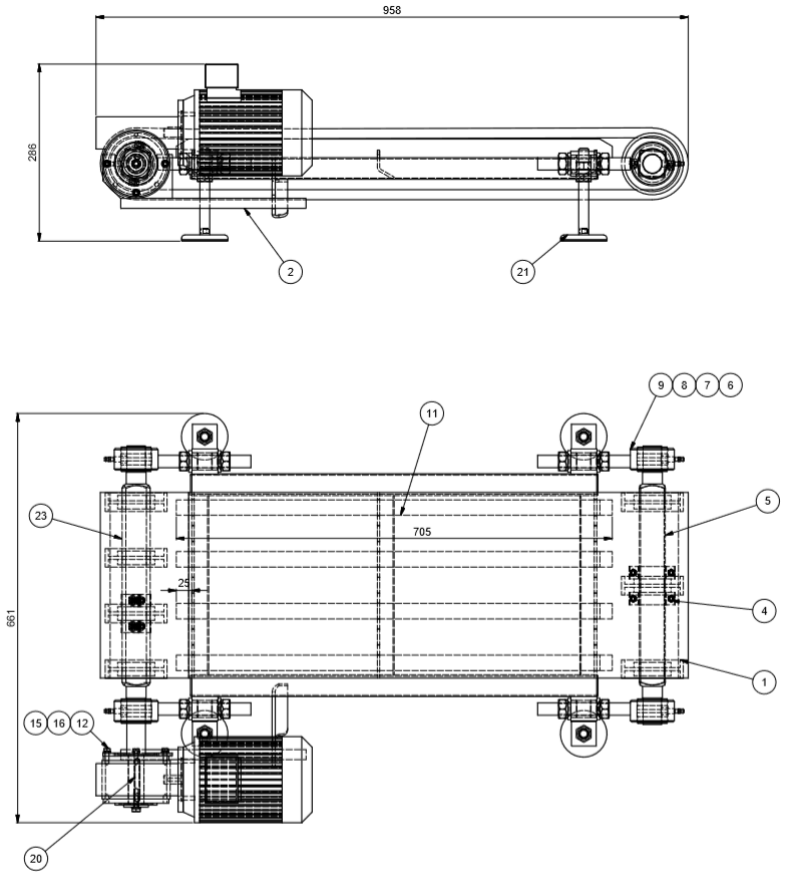
SCALE 1 : 5

OPTIMAR
FISH HANDLING WITH CARE
www.optimar.no

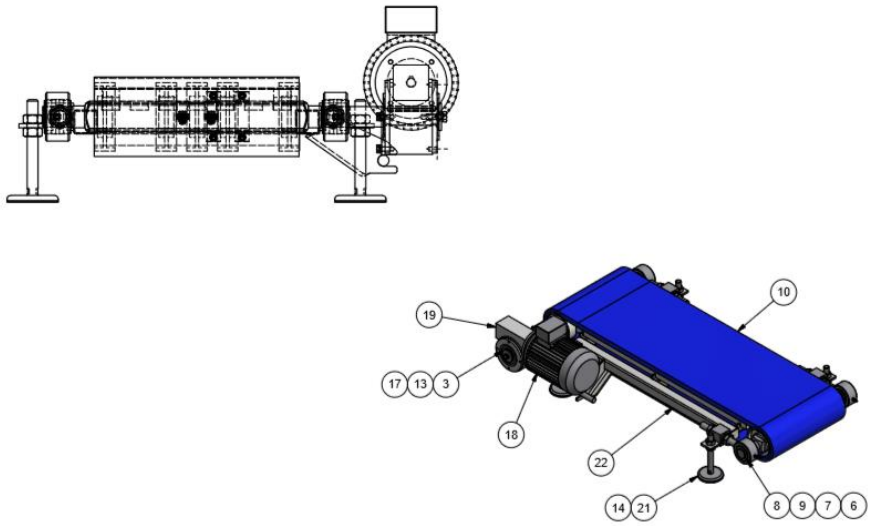
DRAWN BY	DATE	CHECKED BY	CHECKED DATE
olstor	3/11/2019		

Conveyor Pm, L=950 mm

Figur 3 Oversikt over alle komponenter på transportør



Figur 4 Produkttegning 1 av transportøren



Figur 5 Produkttegning 2 av transportør

3.1.4 Docker

Docker blir brukt til å kjøre containers. Containers kjører isolerte applikasjoner, de inneholder verktøy og biblioteket som er nødvendig for å kjøre applikasjonen. Det å kjøre containers er mye enklere for maskinen enn å kjøre en hel virtuell maskin, fordi alle containers deler en kernel. [11]

3.1.5 Docker-Compose

Docker-Compose er et verktøy for å enkelt kunne lage og kjøre flere containers samtidig. Docker-Compose gjør det enkelt å administrere et miljø som krever flere docker containers. [12]

3.1.6 MQTT.fx

MQTT.fx er skrevet i Java. Det er en MQTT klient basert på Eclipse Paho. [13]

3.1.7 PostgreSQL

PostgreSQL er et håndteringssystem for relasjonsdatabaser laget med åpen kildekode. [14]

3.1.8 TimeScaleDB

TimescaleDB er en time-series database som kjører på toppen av PostgreSQL spesielt laget for å lagre time-series data. [15]

3.1.9 ASP.NET

Et gratis rammeverk laget av Microsoft for å utvikle web applikasjoner og tjenester med .NET og C#. Kan også brukes til å utvikle REST API. ASP.NET utvider .NET plattformen til å utvikle for web. [16]

3.1.10 Tensorflow

Tensorflow er et bibliotek for maskinlæring. Tensorflow gjør utvikling av maskinlæringsmodeller enkelt. Tensorflow er åpen kildekode og utvikles av google. [17]

3.1.11 Keras

Keras er et bibliotek for nevrale nettverk skrevet i Python. Keras er laget for å kunne raskt eksperimentere med deep neural networks. Keras gir et høyt abstraksjonslag når det kommer til utvikling av nevrale nettverk. Keras kan kjøre på toppen av Tensorflow, PlaidML, Theano og Microsoft Cognitive Toolkit. [18]

3.1.12 Matplotlib

Matplotlib er et bibliotek for å tegne figurer i Python. Matplotlib kan brukes til alt fra tegning av geometriske figurer til å visualisere 3-dimensionell data. [19]

3.1.13 Python

Python er et objektorientert programmeringsspråk. Det er et generelt programmeringsspråk som kan brukes til å utvikle alt fra enkle skript til større system. Det ble lansert for første gang i 1991 og oppdateres kontinuerlig. Python er lett å bruke på både liten og stor skala. Python støttes av de fleste operativsystemer og er integrert i Linux-kjernen. [20]

3.1.14 React Native

React Native baserer seg på React og er et JavaScript-rammeverk som brukes i hovedsak for å lage mobile applikasjoner for Android og iOS. React er Facebook sitt JavaScript-bibliotek for å bygge brukergrensesnitt til web. Ved å bruke React Native kan det meste av koden du skriver deles mellom plattformer. Dette bidrar til enkel utvikling av applikasjoner både for Android og iOS samtidig. React Native bruker native-komponenter istedenfor web-komponenter som blir brukt i React. [21]

3.1.15 ReactXP

ReactXP er et bibliotek til React Native som er skrevet av Microsoft for å kunne bygge applikasjoner, som også går til Web og Windows samtidig som til Android og iOS. [22]

3.1.16 PyCharm

PyCharm er en IDE som er laget for Python og utvikles av JetBrains. PyCharm har kodeanalysering og debugging for Python noe som gjør utviklingen enklere. Det finnes

to versjoner av Pycharm, en gratis community-edition og en profesjonell utgave med ekstra funksjoner. [23]

3.1.17 Microsoft Visual Studio

Microsoft Visual Studio er en IDE som er laget av Microsoft. Programmet har muligheter for utvikling, testing, analyse, feilsøking og distribuering av kode. Det har støtte for flere programmeringsspråk, deriblant C++, C# og Java. Visual studio finnes i tre versjoner; Community, Professional og Enterprise. [24]

3.1.18 Microsoft Visual Studio Code

Visual Studio Code er en IDE som støtter vanlige utviklingsoperasjoner som feilsøking, oppgavekjøring og versjonskontroll. Den støtter de fleste vanlige programmeringsspråk, enten innebygd, eller gjennom utvidelser. [24]

3.1.19 Intellisence

IntelliSense er et generelt begrep som blir brukt om redigeringsfunksjoner for kode, dette inkluderer: kode ferdigstillelse, parameter info, og medlemslister.

IntelliSense-funksjonene er drevet av en språktjeneste som gir intelligente kodeutførelser basert på språksemantikk og analyse av kildekoden din. [41]

3.1.20 C#

C# er et programmeringsspråk utviklet av Microsoft rundt år 2000 sammen med .NET. C# kan bli brukt til små applikasjoner og større systemer. C# er et multiparadigme språk som kan brukes til å programmere funksjonelt, generisk, objekt-orientert m.m. [25]

3.1.21 Javascript

Javascript er et programmeringsspråk som er mest brukt som skriptspråk for nettsider, brukes også i utviklingsmiljø som Node.js til å lage mobilapplikasjoner. Javascript er et multiparadigme språk som støtter blant annet objektorientert kodestil og funksjonell programmering. Javascript bruker ECMAScript som standard. [26]

3.1.22 Node.js

Node.js er en plattform som eksekverer Javascript kode utenfor en nettleser. Det har åpen kildekode og bruker en event-driven og non-blocking I/O modell. [27]

3.1.23 NPM

NPM (Node package manager) er standard package manager for Javascript og runtime miljøet Node.js. Det er en online database der man bruker kommandolinje for å installere pakker. [28]

3.1.24 Webpack

Webpack er en modul pakker primært for Javascript. Den kan også håndtere HTML, CSS og bilder hvis riktige tilleggsmoduler er inkludert. Webpack tar moduler med avhengigheter og genererer statiske ressurser som representerer disse modulene. [29]

3.1.25 TypeScript

Typescript er et programmeringsspråk med åpen kildekode som er utviklet av Microsoft. Det er en streng syntaktisk superset av Javascript, og legger til en valgfri statisk skriving til språket.

Typescript ble utviklet for å forbedre utviklingen av store applikasjoner, og kan brukes til både klienter og server (Node.js)

3.1.26 GitLab

GitLab er en web-basert service for Git-basert versjonskontroll. Det er brukt verden over og er mest brukt til programmeringsrelaterte oppgaver. Git gir mulighet for å jobbe med forskjellige aspekter av koden samtidig, uten at det påvirker andres arbeid. Det gir også mulighet for skylagring av alle filer i prosjektet under alle stadier av utviklingen. [30]

3.1.27 dbForge Studio for PostgreSQL

DbForge Studio for PostgreSQL er et utviklingsverktøy for PostgreSQL databaser. Programmet gjør det enkelt å utvikle og kjøre queries i et brukergrensesnitt. [32]

3.1.28 Sysmac Studio

Sysmac Studio er en maskinautomasjonsplattform til bruk for styring av Omron PLS. Sysmac Studio brukes på en datamaskin og kobles til PLS via en TP-kabel. Programmet er skapt for å enkelt kunne kontrollere maskinvare, kontrollere, sensorer og kamerautstyr på maskiner. Ved hjelp av et grafisk brukergrensesnitt kan man enkelt se hvordan forskjellige enheter skal oppføre seg. Med mulighet for å programmere i strukturert tekst. [33]

3.1.29 Draw.io

Draw.io er en online editor for å lage diagram. Det har støtte for alt fra business diagram til UML diagram for programvare utvikling. [34]

3.2 Data

All data i prosjektet er det gruppen som har produsert. Ved å logge strømtrekket via PLS har transportøren blitt brukt på forskjellige måter. Forskjellige typer data som er forsøkt produsert er riktige data og feildata. Under produksjon av riktig data har transportøren vært i normal drift. Gruppen har forsøkt å sette seg inn i hva som regnes som normal drift og kommet frem til to forskjellige bruksområder. Det ene bruksområdet er når transportør står i riktig stand og kjører uten last. I en vanlig prosessoperasjon er det ikke unormalt med transportører som venter på arbeid slik at dette vil være reelt fra arbeidssituasjon. Det andre bruksområdet er normal drift av transportør der den transporterer masse innenfor normale verdier slik som den er tiltenkt. Begge typer data er klassifisert som riktige data. Under generering av feildata har gruppen forsøkt å finne ut hva som kan være tilfeller av feil bruk av maskinen. En av tingene som har blitt gjort for å fremprovosere feil er å stramme det justerbare båndet på transportøren til det ikke lenger er noe slakk. En annen ting har vært å gi transportøren en for stor belastning i forhold til hva den er ment til å transportere. Det har også vært forsøkt å stanse transportøren som skal simulere at noe har kilt seg fast en plass i maskineriet.

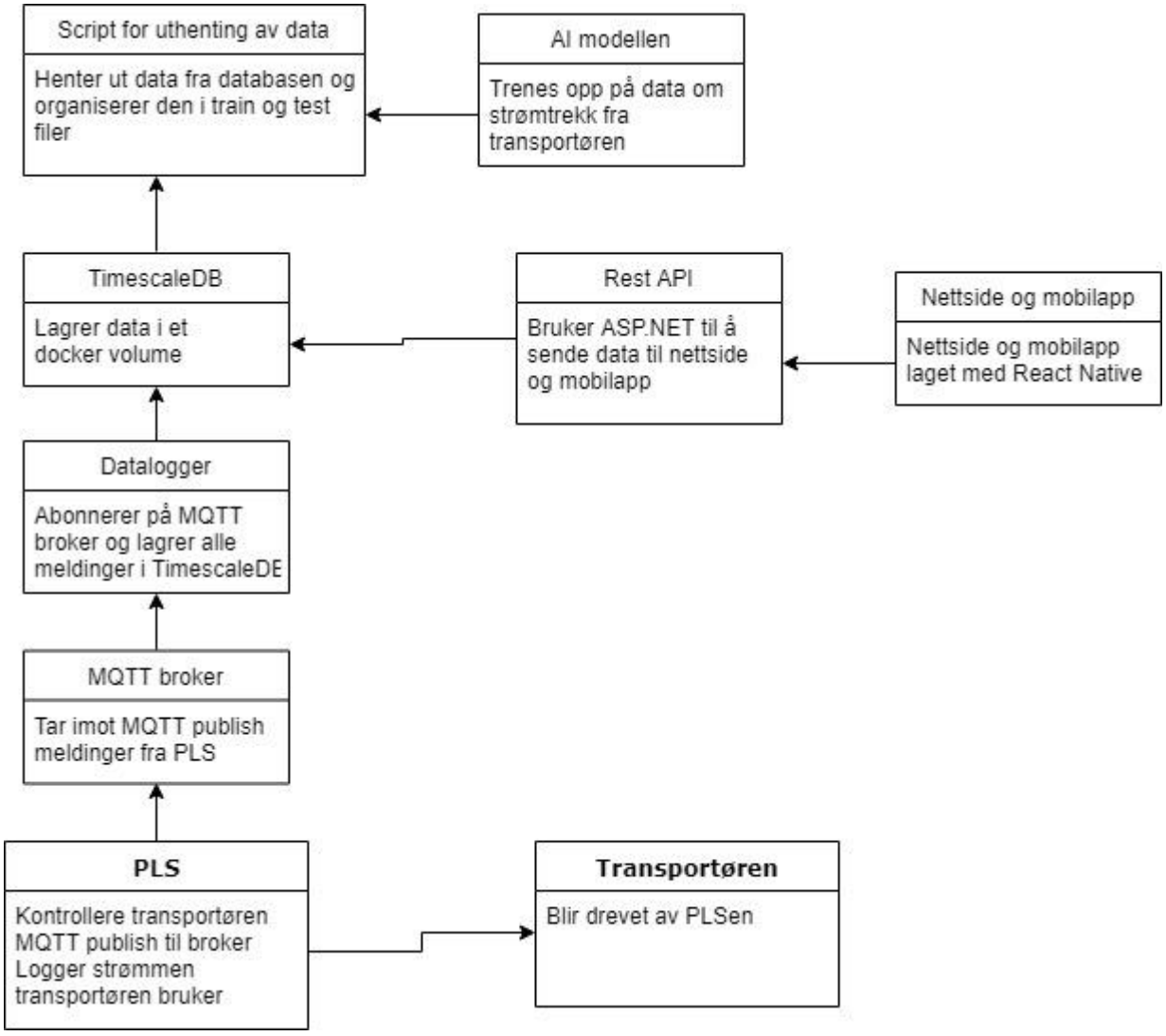
3.3 Prosjektorganisering

Gruppen ble gitt frie tøyler når det kom til valg av teknologier og arbeidsflyt. Optimar har stilt i stand et stort fint kontor som gruppen har fritt benyttet seg av og hatt som arbeidsplass store deler av prosjektet. Det har også vært positivt engasjement fra Optimar AS der de har kommet med innspill og hjelp dersom studentene har bedt om det. Det har spesielt vært viktig med enkel opplæring i bruk av PLS. Gruppen har også hatt en enighet i hvilke teknologier som skulle benyttes i oppgaven og det har blitt opprettholdt etter beste evne. Noen små endringer har blitt gjort, og i henhold til planen har gruppen forsøkt å komme til enighet og i verste fall brukt majoritetsklausulen fra forprosjektet.

4 RESULTATER

4.1 Systemets oppbygging

Oversikt over det endelige systemet:

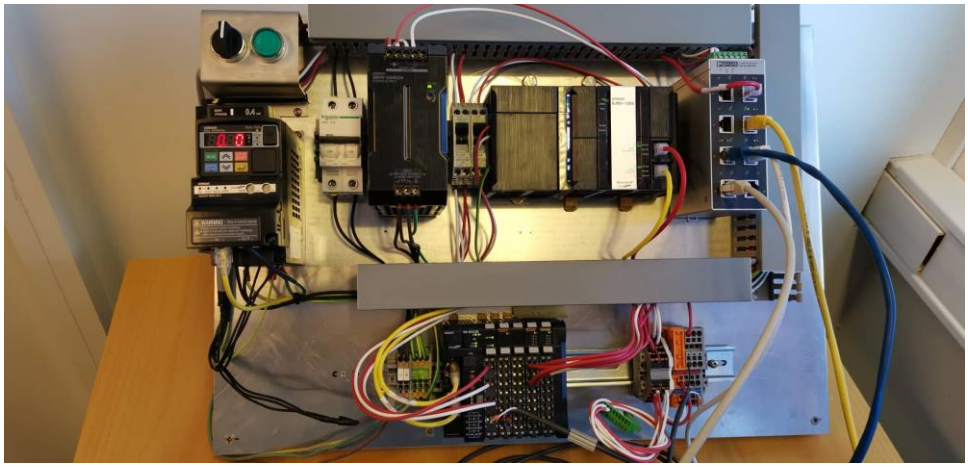


Figur 6 Oversikt over det endelige systemet

Dataflyten mellom PLS og databasen ble løst ved hjelp av broker pattern som beskrevet i kapittel 2.2.3 Dette ble gjort for å ha en løs kobling mellom komponentene slik at systemet kan fungere for andre typer datalogging eller annen løsning på

databehandlingen. Koblingen mellom rest API og visning av data i front-end er client-server patternet som beskrevet i kapittel 2.2.3 tatt i bruk. Dette gir en frihet i måten data blir presentert på i front-end.

4.2 PLS oppsettet



Figur 7 Kontrollenhet med PLS som ble brukt til bachelor oppgaven

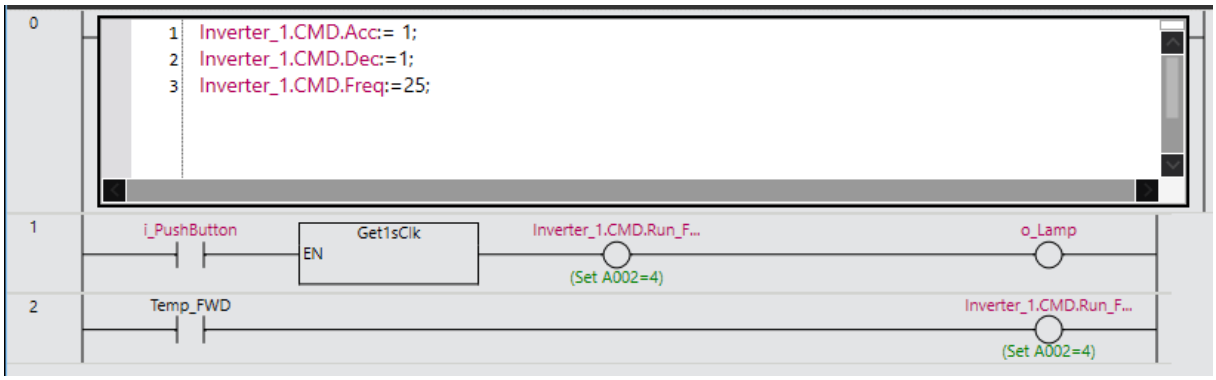
Oppsettet av PLS, transformator og switch.

4.2.1 PLS

PLS som beskrevet i kapittel 3.1.1 er programmert ved hjelp av strukturert tekst. Det har blitt utviklet et program for å starte og stoppe transportøren. Ved oppstart av transportøren må PLS informere transformatoren om en rekke gitte verdier: hvilken frekvens motoren skal kjøre på, hvor mange sekund den skal bruke på å akselerere opp til den gitte

en og hvor mange sekund den skal bruke på å stoppe. Transportøren blir startet og stoppet ved å sette boolean verdien Temp_FWD til True eller False, i tillegg finnes også en trykk-knapp som er programmert til å stoppe transportøren.

Programmet som er utviklet ser slik ut:



Figur 8 Utsnitt av programmet som kjører på PLS

PLS kjører også en MQTT klient som sender data til MQTT serveren, den er programmert i strukturert tekst for Omron PLS. MQTT klienten kobler til MQTT broker containeren og sender en MQTT PUBLISH i hver syklus, en syklus er satt til 500 millisekund. I løpet av denne syklusen blir strømtekket som motoren bruker målt og sendt til serveren.

4.2.2 Transformator

Transformatoren som beskrevet i 3.1.2.1 omformer fra 240V 50/60Hz til 24V DC som PLS bruker.

4.2.3 Switch

Det blir brukt en Switch til å koble sammen datamaskiner med PLS oppsettet. Switchen brukes også til testkommunikasjon mellom rest API og web.

4.3 Docker oppsett

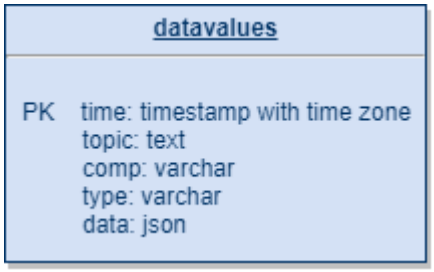
Oppsettet består av tre containere som jobber sammen for å lagre data fra PLS. Containerene ble hentet fra optimar.azure, de blir konfigurert og startet ved hjelp av docker compose 3.1.5.

```
PS C:\Users\OEM\Downloads\bachelor> docker ps --format "table {{.ID}}\t{{.Image}}\t{{.Names}}\t{{.Ports}}"
CONTAINER ID        IMAGE                                       NAMES               PORTS
4cc64b1bef2b        optimar.azurecr.io/timescaledb:latest       timescaledb         0.0.0.0:5432->5432/tcp
e28b1289baa1        optimar.azurecr.io/mqttbroker               mqttbroker          0.0.0.0:1883->1883/tcp
57e6b00f78d7        optimar.azurecr.io/commander_edge:latest    datalogger
```

Figur 9 Tabell som viser en list av containers som kjører

4.3.1 TimescaleDB

Den første containeren består av en TimescaleDB server, og alle sensormålingene blir lagret i databasen. Målingene lagres i tabellen datavalues.



Figur 10 ER-diagram fra TimeScaleDB

- **Time:** Tiden når data ble målt.
- **Topic:** Hvilken MQTT topic data ble sendt til.
- **Comp:** Hvilken enhet som sendte data.
- **Type:** Hvilken type data ble målt
- **Data:** Sensordata

Eksempel på en måling i databasen:

time timestamp with time zone	topic text	comp varchar	type varchar	data json
04/08/2019 12:57:34 PM	OC1/DATA/PLS/Freq	pls	freq	0.82

Figur 11 Utsnitt av en måling fra databasen

4.3.2 MQTT broker

MQTT broker er en MQTT server som MQTT klienten på PLS sender til og som dataloggeren abonnerer på.

4.3.3 Datalogger

Datalogger containeren består av en MQTT klient som abonnerer på en MQTT broker for så å legge dataene inn i en tabell i TimescaleDB.

4.3.4 Commander_postgres

Commander_postgres er et docker data-volume som blir brukt til å lagre databasen. Selv om TimescaleDB blir satt opp på nytt vil docker volumet bli beholdt og dataene forbli lagret.

4.3.5 Docker-compose

```
# https://docs.docker.com/compose/compose-file/

services:
  db:
    container_name: timescaledb
    image: timescale/timescaledb:latest-pg10
    restart: always
    ports:
      - 5432:5432
    volumes:
      - commander_postgres:/var/lib/postgresql/data

  broker:
    container_name: mqttbroker
    image: optimar.azurecr.io/mqttbroker
    restart: always
    ports:
      - 1883:1883

  datalogger:
    container_name: datalogger
    image: optimar.azurecr.io/commander_edge:latest
    restart: always
    entrypoint: "python app_service_runner.py datalogger"
    volumes:
      - ./edge_config.json:/usr/src/app/edge_config.json
    tty: true

volumes:
  commander_postgres:
    external: true
```

Figur 12 Konfigurasjonsfilen som blir brukt for å kjøre bildet i Docker (docker-compose.yml)

Bildene blir hentet fra optimar.azure og konfigurasjonen til containerene blir lest fra docker-compose.yml (Figur 12) når "docker-compose up" kommandoen blir kjørt.

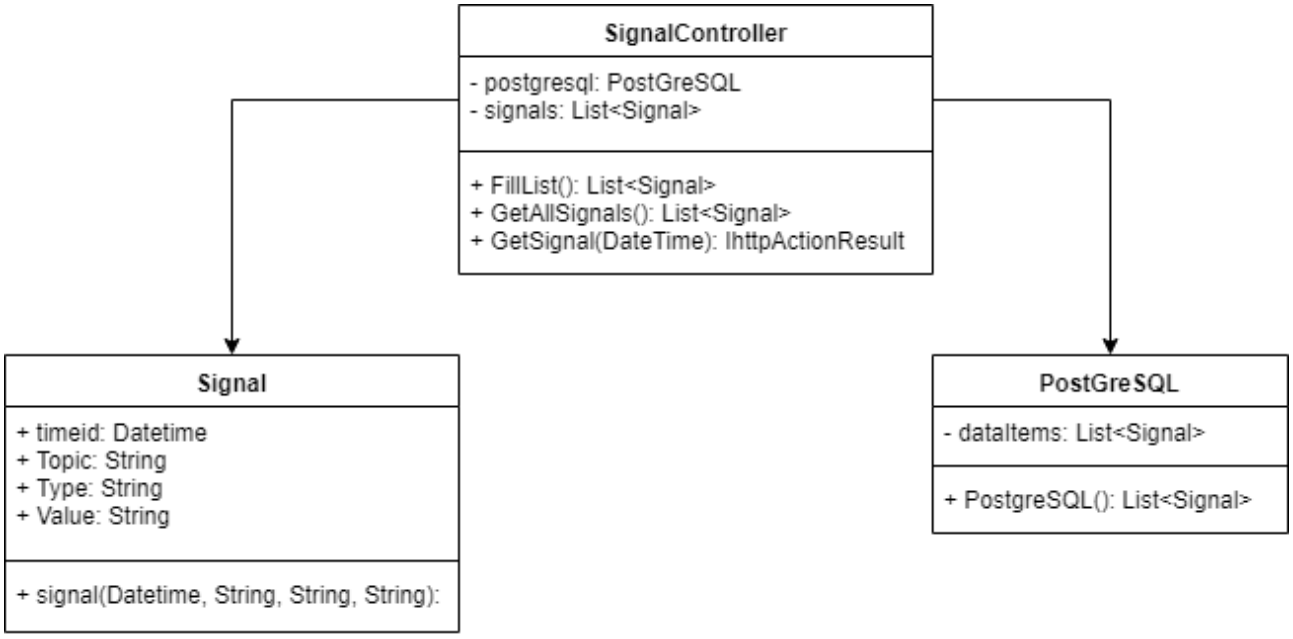
```
{
  "plantkey": "XXXXX",
  "mqtt": {
    "broker": "mqttbroker"
  },
  "postgres_db": {
    "host": "timescaledb",
    "user": "postgres",
    "password": "xxxxxxx",
    "db": "edge"
  },
  "services": {
    "datalogger": {
      "type": "datalogger",
      "client_name": "APP-LOGGER-PostgreSql",
      "topics": ["OC1/DATA/#"]
    }
  }
}
```

Figur 13 Konfigurasjonsfilen som datalogger bruker (edge_config.json)

Konfigurasjonsfil (Figur 13) som dataloggeren bruker for å vite hvilken database den skal logge til og hvilken MQTT broker den skal abonnere på.

4.4 Rest API

4.4.1 UML



Figur 14 UML klassediagram av rest API

4.4.2 API kall

Endpoint	Request type	POST data	Headers	Response	Protected
/webapi/api/signal	GET	None	None	Application/JSON	No
/webapi/api/status	GET	None	None	Application/JSON	No

Responsen som sendes når det blir sendt en GET request til serveren vil være i form av et JSON Array. Under følger et JSON objekt fra et utsnitt av responsen av /signal som blir mottatt fra serveren:

```
{
    "timeid": "2019-04-08T12:57:34.676378+02:00",
    "Topic": "OC1/DATA/PLS/Freq",
    "Type": "pls",
    "Value": "0.06"
}
```


Figur 15 JSON Object hentet fra databasen

Hvert objekt består av fire nøkler med fire tilhørende verdier. Under kan dere se en beskrivelse av hver nøkkel i objektet:

- **timeid:** Tiden med tidssone når dataverdien ble målt.
- **Topic:** Her blir det sett hvilket MQTT topic som data ble sendt til.
- **Type:** Viser hvilken datatype som ble målt.
- **Value:** Verdien av strømtrekket til motoren på transportøren gitt i Hz.

I Rest API ble det også utviklet en funksjonalitet for å kommunisere med AI, denne får en varsling om statusen til transportøren og en prediksjon om tiden som følger. Siden AI ikke fungerer optimalt ligger det nå en funksjon for å vise hvordan transportøren har blitt operert den siste tiden. Dette er ikke koblet opp mot AI, men mot data innsamlet. For å unngå varsling av enkeltsignaler kjører vi en enkel funksjon som regner ut snittet av de 20 siste målingene. Den varsler ikke under normal drift, men kun om styrken fra signalet overstiger normalverdiene vi har bestemt.

4.5 React Native Front-end

Som front-end har det blitt brukt Visual Studio Code (3.1.183.1.18) som IDE. Det ble brukt React Native (3.1.14) som rammeverk, og sammen med ReactXP (3.1.15) biblioteket ble det laget to applikasjoner. En applikasjon til Android og en til web. De nevnte applikasjonene ble bygget ut fra samme kodebase, og koden ble kompilert ned

til native kode samt tilpasset de brukergrensesnittene som de forskjellige plattformene bruker.

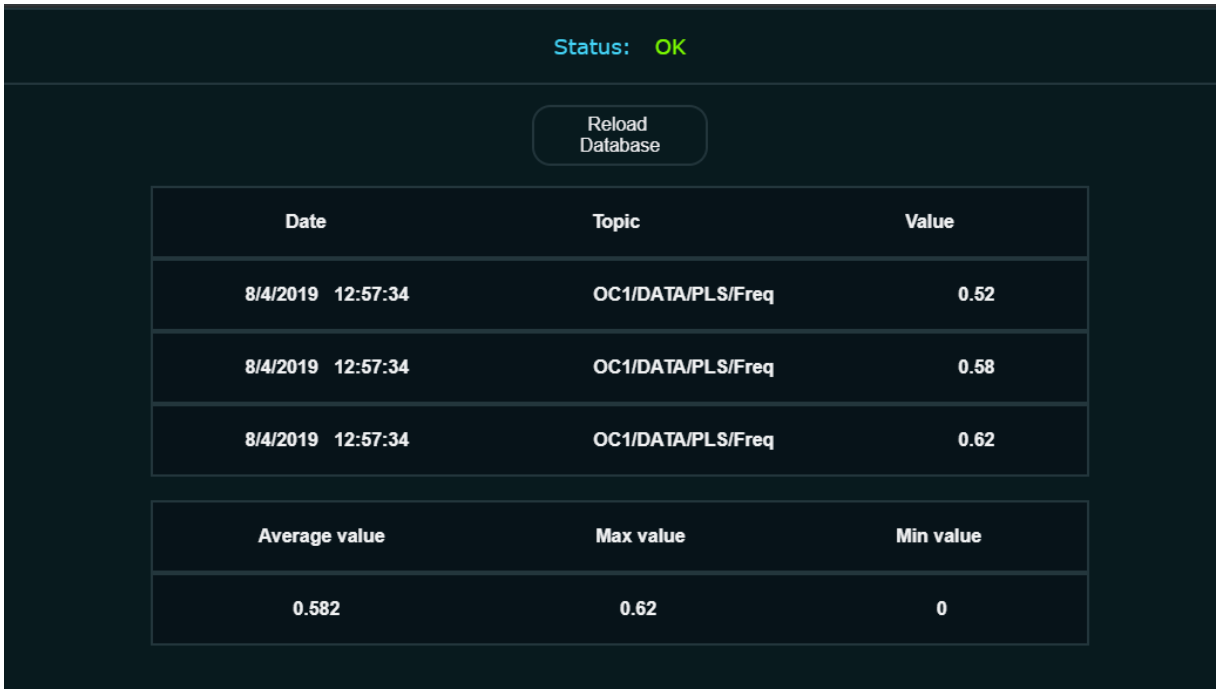
Ved å benytte biblioteket ReactXP trengte ikke gruppen flere abstraksjonslag for å bygge webapplikasjonen.

Som vist til i figurene under (Figur 16-19) utviklet gruppen en type for direktevisning av statusen for transportøren. Denne statusen varsler hvis noen av tilfellene der transportøren har problemer inntreffer.

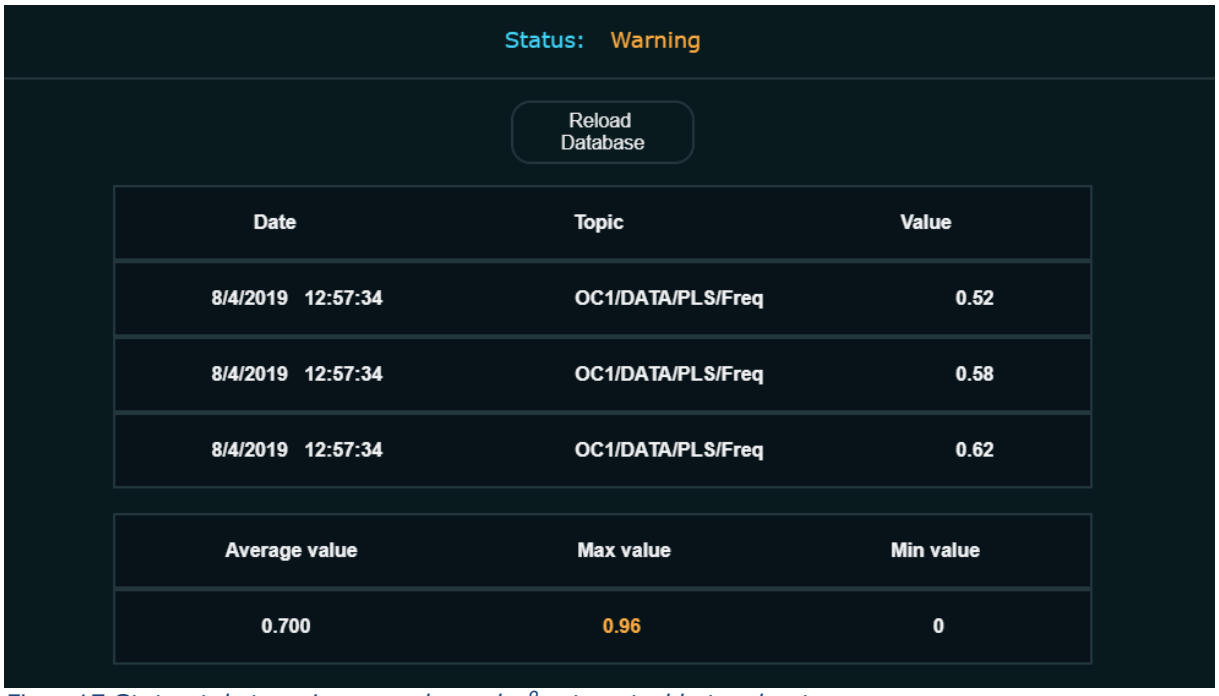
4.5.1 Utforming

Web

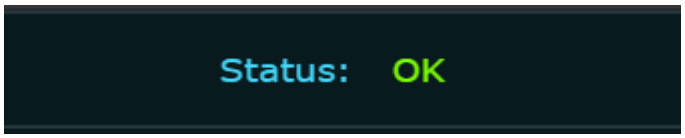
Utformingen av webapplikasjonen. Status tekst endrer etter hvilken status det har:



Figur 16 Nettside med database lastet.



Figur 17 Status teksten viser en advarsel når strømtrekket er høyt



Figur 18 Bilde av status i webapplikasjon

Mobil

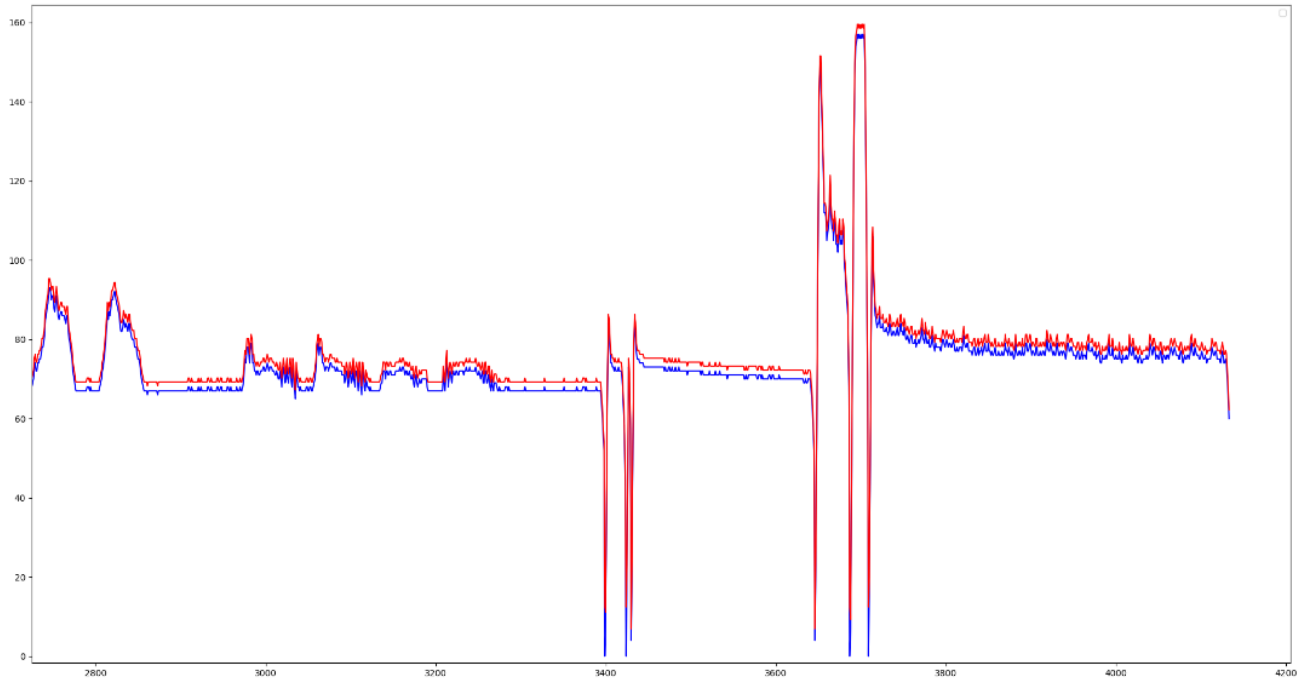
Utseende til Android appen ser ut som bildet under:

Figur 19 Utseende på en Nexus 6P som kjører Android som operativsystem

4.6 AI

4.6.1 Datavisualisering

Det har blitt utviklet et Python-program for å direkte visualisere strømtrekket til motoren. Ved hjelp av dette programmet kunne gruppen se hvordan motoren reagerte på forskjellig last og andre forhold. Programmet henter seneste loggført strømtrekk i et gitt intervall og bruker matplotlib for å visualisere det. Grafen oppdateres automatisk i et gitt intervall.



Figur 20 Graf som visualiserer strømtrekk over tid

Ved å studere grafen (Figur 18), observeres det at etter siste oppstart av transportøren normaliserte strømtekken seg mye høyere enn ved de to tidligere gangene den ble startet. Denne observasjonen viser at transportøren kjører tyngre på det angitte tidspunktet. I det angitte tilfellet ble transportbåndet strammet, og dette viser at grafen også kan benyttes til å observere når transportøren kjører komfortabelt.

4.6.2 Treningsdata

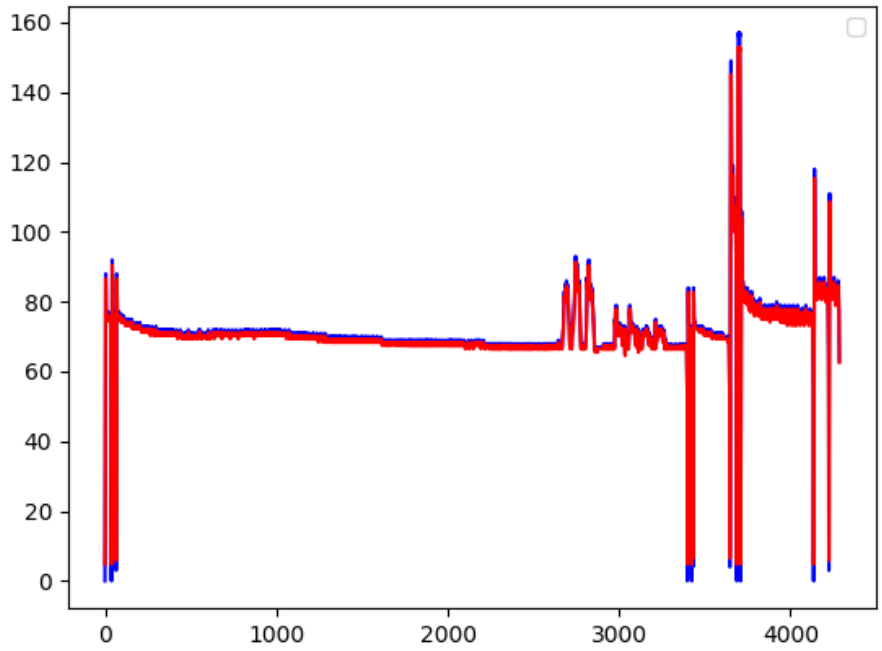
Treningsdata til AI blir hentet ut fra TimescaleDB og deretter blir data behandlet før den lagres i tekstfiler, dette slik at modellen kan trenes og testes. Treningsdataene er en indeksert liste med float-verdier som representerer strømtrekket og dataene blir splittet i to. Den første delen blir brukt til opptrening av modellen på hvordan strømtrekket forandrer seg over tid, og den andre delen av datasettet blir brukt til å teste hvor bra modellen kan klare å se fremover.

4.6.3 Modellen

AI modellen er laget ved hjelp av Keras med Tensorflow back-end. Nettverket tar inn en rekke med 32 verdier for strømtrekk. Det som angis ved bruk av modellen er en forutsetning over hvilken verdi som kommer etter rekken den tok inn.

Layer type	Layer size	Activation
Input layer	Dense (32)	ReLU
Hidden Layer	Dense (64)	ReLU
Hidden Layer	Dense (32)	ReLU
Output Layer	Dense (1)	ReLU

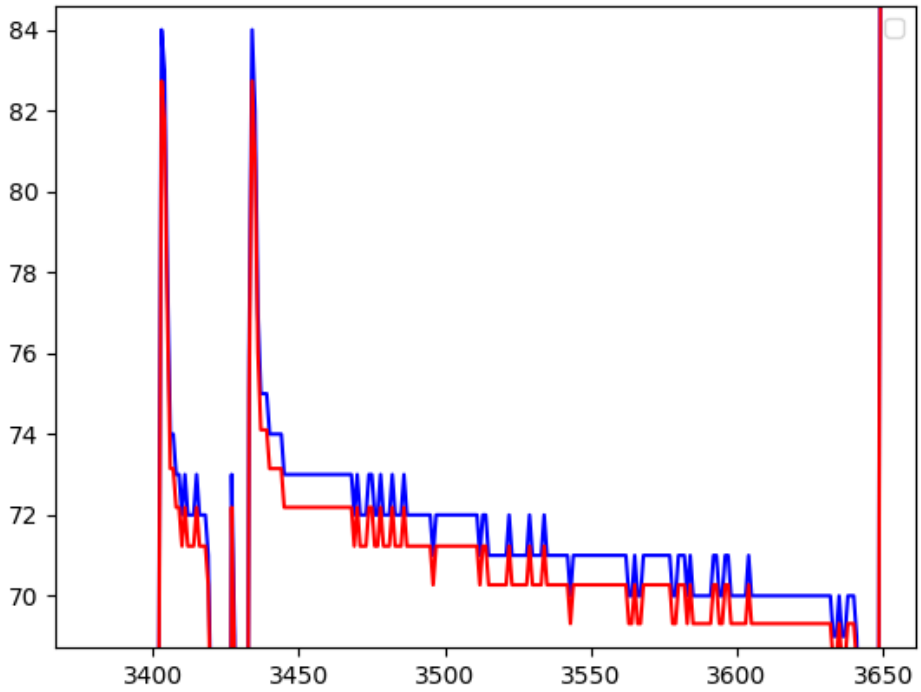
Tabell 1: viser en oversikt over det nevrale nettverket.



Figur 21 Graf som viser verdiene for strømtrekk som modellen tok inn

Den blå grafen (Figur 19) i bakgrunnen viser rekken med verdier som ble sendt til modellen. Den røde grafen (Figur 19) i forgrunnen er modellens forutsigelse om hva

som kommer til å skje fremover. I figur 19 ligger grafene over hverandre for å vise hvor like de er.



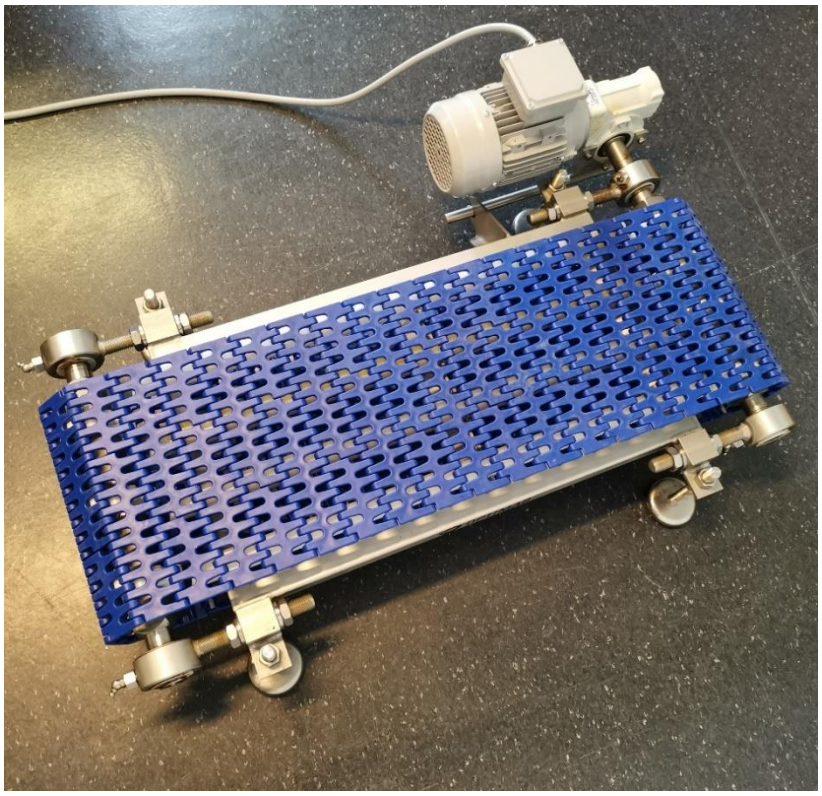
Figur 22 Utsnitt av grafen vist i Figur 19

Et nærmere utsnitt av samme graf (Figur 20). Her blir det også observert hvor lik forutsigelsen er de faktiske verdiene som modellen fikk inn.

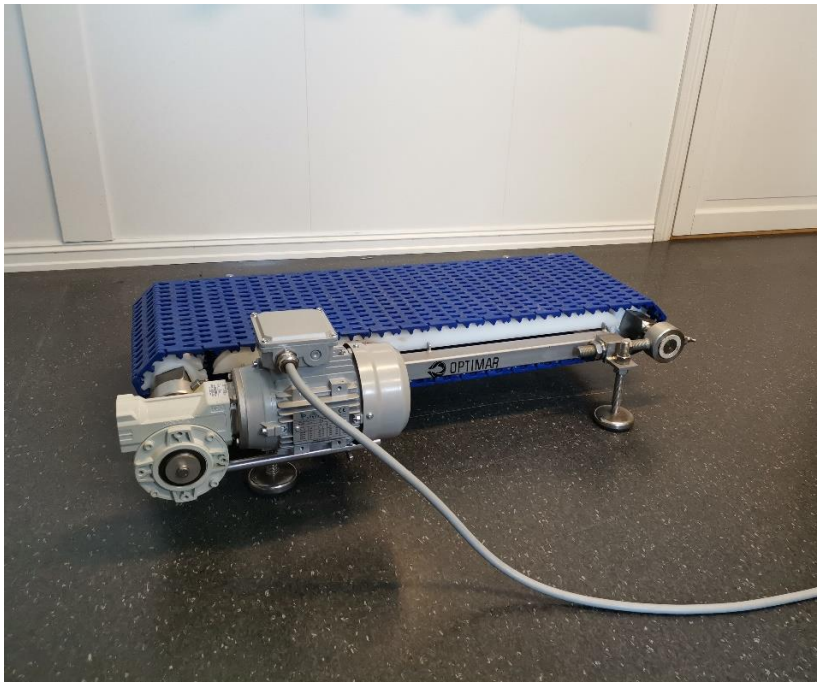
4.7 Transportøren

Den ferdige transportøren:

Figur 23 Transportør med test objekt



Figur 24 Transportør sett ovenfra



Figur 25 Transportør sett fra siden

5 DRØFTING

5.1 Rest API

Fra starten av hadde gruppen en plan om å utvikle et Rest API for å kommunisere med data logget i databasen, og vurderingen fra AI om behov for fremtidig vedlikehold. Samtidig måtte back-end ha mulighet for enkelt publisering av data til front-end. Valget falt på .NET og ASP.NET til publisering av data. Selve kommunikasjonen med PostgreSQL serveren blir gjort i .NET og lagret som en liste med objekter der hvert objekt er en måling fra PLS. Deretter har ASP.NET enkel mulighet til å kalle ut objektene fra .NET og publisere de som JSON objekter slik at de kan hentes ut og benyttes i applikasjoner eller nettlesere.

En av hovedgrunnene til at gruppen valgte å bruke .NET/ASP.NET er at Optimar allerede bruker Microsofts tjenester i form av Azure. For enkelt å kunne integrere vår prototype falt valget på Microsofts tjenester.

5.2 Front end

Når det kom til front-end hadde gruppen flere muligheter ettersom det ikke var noe krav fra oppdragsgiver om hvordan informasjon skulle framvises.

Gruppen hadde behov for å fortelle brukere av systemet om eventuelle feil som kunne oppstå, det ble dermed besluttet at en nettside var en god ide. En mobilapplikasjon kunne også være et nyttig hjelpemiddel siden den har bedre mulighet til å sende push notifikasjoner. Det ble vurdert ulike rammeverk for utvikling av programvaren, deriblant Java, React, React Native og Ionic.

Gruppen gikk for å bruke React Native av den grunn at det ville spare gruppen for tiden som det ville tatt å lage en applikasjon til hver plattform. Med React Native kunne gruppen skrive en kodebase som igjen kunne kompileres ned til flere forskjellige plattformer. Det tok ganske lang tid å sette seg inn i denne måten å kode på, ettersom gruppen ikke hadde brukt verken React eller Typescript fra før.

Ved bruk av React Native kunne gruppen produsere en applikasjon til iOS og Windows, men for å gjøre dette ville det ha krevd en Apple-maskin og Windows software development kit.

React Native støtter både Typescript og Javascript, men valget falt etter hvert på Typescript. Dette ble gjort for å få en ekstra sikkerhet når det kom til typer, samtidig som man fikk feildeteksjon ved kompilering og bruken av IntelliSence som er beskrevet i punkt (3.1.19).

5.3 AI

Gruppens vurdering er at AI-modellen ikke opererer like bra som den burde gjøre, det er to hovedgrunner til dette. Den ene grunnen er at selve det nevrale nettverket ikke er godt nok optimalisert, men hovedgrunnen er mangel på data.

Gruppen hadde fra starten av en plan om å lage en modell ved hjelp av unsupervised learning. Etter noen samtaler med Emil Dale Bjørlykhaug ble det besluttet at gruppen skulle se bort fra unsupervised learning. Årsaken til dette er at ingen publiserte kilder har fått til å benytte unsupervised learning med positivt resultat. Gruppen startet da utviklingen av et nevralt nettverk med LSTM komponenter. LSTM nettverk krever at data er i tre dimensjoner. Siden det bare blir logget strømtrekk i en dimensjon i systemet, passer det ikke å benytte LSTM nettverk til denne oppgaven. Gruppen besluttet derfor å utvikle et enkelt nevralt nettverk som skulle forutsi hvordan trenden på strømtrekket kom til å bli fremover i tid. Det viste seg at denne modellen gir et resultat som ikke holder helt mål. Dersom du angir data om hvordan strømtrekket har vært i perioden frem til nå vil modellen gjøre noen små justeringer, deretter vil den gjenta nesten den samme forutsigelsen og angi at det samme vil skje videre fremover. Gruppen har også utviklet et endimensjonalt convolutional neural network, men den siste modellen genererte heller ikke noe resultat som kunne brukes til vårt formål. Til slutt endte gruppen med å bruke det enkle nevrale nettverket. Selv om modellen ikke gir ønsket forutsigelse er dette det beste resultatet gruppen kom frem til med den tiden, de dataene og den kompetansen gruppen hadde til rådighet.

5.4 Docker

Docker ble valgt fremfor å benytte virtuelle maskiner for å kjøre serverne som gruppen hadde behov for. Det å bruke Docker containere gjør oppsett og test av dette veldig raskt og enkelt. Alternativet var å bruke virtuelle servere der endring av parametere ofte krever at systemet må settes opp på nytt, noe som kan ta flere timer.

For å gjøre endringer og sette opp flere Docker containere samtidig har det blitt tatt i bruk docker-compose. Den bruker en enkel fil til å sette alle parameterne til containerene, denne filen gir en enkel oversikt over alle containerene og alle parametere.

5.5 PLS oppsettet

Gruppen fikk hjelp av Ole Andre Tomren til oppsettet av PLS. Uten denne hjelpen hadde det tatt mye lengre tid for gruppen, ettersom den består av tre datastudenter uten så mye automasjonserfaring.

Marius Nedrelid har bistått gruppen med programmering av den strukturerte teksten som benyttes av PLS. Dette var til stor hjelp, ettersom gruppen ikke hadde vært borti programmering med strukturert tekst før.

5.6 Håndtering av problemer

5.6.1 Transportøren

Gruppen fikk ikke transportøren før sent i mars. Årsaken til dette var at Optimar hadde en testrigg med transportør som studentene skulle benytte, men når prosjektet startet ble denne riggen opptatt til et annet prosjekt. Gruppen måtte da sammen med diverse ansatte på Optimar få tegnet en ny transportør som ble produsert av fabrikken til dette prosjektet. Prosessen startet med å finne vanlige feil som forekommer på transportørene slik at de enkelt kunne gjenskapes på den nye transportøren, deretter ble transportøren tegnet og sendt til produksjon. Dette

forsinket en del av gruppens utvikling, og selv om systemet er i stor grad selvstendig uten transportør var det vanskelig å få testet ut helheten før transportøren var i drift. Når ny transportør var installert og gruppen fikk testet systemet var det flere deler av utviklet system som måtte endres og gjøres på nytt for at dataflyten skulle fungere som tiltenkt.

5.6.2 Uforutsette problem

Gruppen sto flere ganger fast på problemer som ikke kunne løses på egenhånd. Dette skyldes mangel på kompetanse og situasjoner som gruppen selv ikke kunne styre.

Første problem var mangel på kunnskap om strukturert tekst. Her fikk gruppen en innføring og hjelp Marius Nedrelid til å få systemet klart til å logge data fra transportøren.

Det neste problemet som gruppen støtte på var nettverkskommunikasjonen mellom nettverkskomponenten til PLS og MQTT brokeren. Grunnen var at windows ikke ville ta imot pakkene som kom fra PLS. Dette ble fikset med å åpne porten som MQTT (1883 og 8883) bruker i brannmuren og fjerne MQTT brokeren fra windows defender.

Når det gjelder front-end så møtte gruppen på problem med missmatch av node modules. Forskjellige moduler og versjoner for disse fungerte ikke sammen. Enkelte moduler måtte rulles tilbake til tidligere versjoner for at de skulle fungere med hverandre. Gruppen brukte en god del tid på å analysere og finne ut av dette.

React Native rammeverket har to typer data som kan styre en komponent, det er State og Props. Det var ganske utfordrende i starten å finne ut av hvilke man benyttet til hva. Etter en del utprøving og research fant gruppen ut at State skal brukes til endring av element som allerede er laget, og props benyttes for å sende parameter til child-elementer.

Et annet problem som oppsto med front-end var at ReactXP biblioteket hadde manglende støtte for mange av React Native sine komponenter. Derfor var det begrenset hvilke komponenter gruppen kunne bruke til utviklingen.

Kommandoene for å kompilere og bygge koden er annerledes ved bruk av ReactXP. Gruppen måtte analysere, teste og verifisere for å finne riktige kommandoer og sammensetning.

5.7 Datainnsamling

Som referert til i forrapporten og i møteloggen fra de tidligste møtene var den opprinnelige planen å sette opp måleutstyr på produksjonsområdet til Fjordlaks AS. Om dette hadde blitt gjennomført ville gruppen hatt tilgang til data fra en rekke transportører som går i daglig produksjon. Opptreningen av AI ville da hatt helt andre data å basere seg på.

Prosjektet på Fjordlaks lot seg ikke gjennomføre og prosjektet ble endret til å logge data fra en enkel transportør. Grunnet dette måtte gruppen selv imitere både normal bruk og ekstrem bruk av transportøren. Tidspress gjorde at det ikke ble brukt nok tid i denne prosessen og klassifisering av data ble ikke like bra som vi hadde forespeilet oss i planleggingsfasen. Sett ut ifra dataene gruppen har samlet inn, kan vi med sikkerhet si at dersom datamengden hadde vært i et større omfang hadde den ferdige modellen gitt et langt bedre resultat.

5.8 Planlagte funksjoner

5.8.1 Logge flere typer data

Dersom flere typer data logges, vil AI modellen kunne bli trent til å få en bedre forståelse for når transportøren kommer til å trenge fremtidig vedlikehold. For eksempel kan temperaturen til kjøleluften eller kjernetemperatur i motor bidra til å oppnå denne forståelsen.

5.8.2 Kommunikasjon imellom AI og rest API

Det var planlagt å lage mulighet for at AI modellen skulle si ifra til rest API dersom transportøren jobber utenfor optimale forhold. Med denne funksjonaliteten kan brukeren få en tilbakemelding om for eksempel transportbandet er for stramt eller om

transportøren løfter for mye vekt. Brukeren ville få en advarsel i form av at noe var galt på transportøren og at det ville betraktelig redusere tiden det ville tatt frem mot neste vedlikehold eller eventuell nedetid.

5.8.3 Visualisere data i front-end

Gruppen planla å visualisere data ved hjelp av en graf innenfor et gitt tidsintervall som brukeren kan skrive som input direkte i nettleseren eller på app.

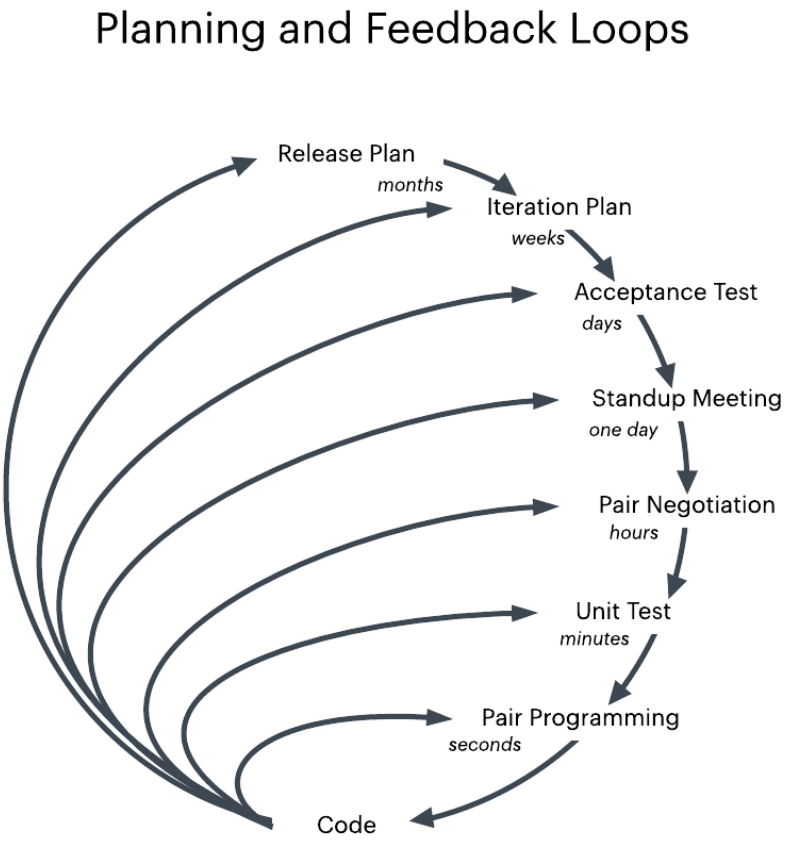
Det var også planlagt å endre statusvinduet som i dag ligger øverst i applikasjonen (Figur 16-19). Etter planen skulle det varsles om estimert tid til vedlikehold og status for hvor hardt transportøren jobbet slik at det enkelt kunne gjøres justeringer i et større system for å justere arbeidslasten.

5.9 Utviklingsmetodikk

Gruppen planlagte fra starten av å følge en agil utviklingsmetodikk. Grunnen for valget var at siden størstedelen av oppgaven besto av software utvikling vil en agil metode passe prosjektet langt bedre enn ved bruk av tradisjonell fossefallsmetodikk. Valget falt på Scrum metodikken og gruppen startet med planleggingen av sprinter som skulle følges.

Siden prosjektet kort tid etter prosjektoppstart fikk en stor uplanlagt forsinkelse, som referert til i 5.6.1. valgte gruppen å gå bort fra Scrum. I tidsperioden frem til transportøren ble levert, jobbet gruppen med planlegging og informasjonsinnsamling i forbindelse med transportører. I tiden fra bestillingen og fram til gruppen hadde fått ny transportør, ble det besluttet en endring i utviklingsmetodikk og gruppen valgte å benytte XP (Extreme Programming). Årsaken til at metodikken ble endret var at Extreme Programming er mer agil enn Scrum og med usikkerhet rundt leveringstid måtte gruppen planlegge å jobbe langt mer intensivt med selve programvaren på et senere tidspunkt. Et av hovedprinsippene til XP er prototyping, og siden gruppen ennå ikke hadde fått utstyr ble det jobbet en del med prototyping av hvordan prosjektet skulle flettes sammen. Etter leveranse av transportør ble det arbeidet mot et annet av prinsippene bak XP, nemlig enkelhet (simplicity). Gruppen måtte først ferdigstille det som under planleggingen ble sett på som minstekriterier for oppgaven. Etter

minstekravet var nådd utviklet gruppen i planning loops med å bygge på flere funksjoner, for bedre kommunikasjon mellom enheter og for bedre presentasjon av data.



Figur 26 Utviklingsmetodikken til XP

6 KONKLUSJON

Sammen med oppdragsgiver har gruppen utforsket muligheten til implementering av PdM (predictive maintenance) for transportører, tilsvarende de oppdragsgiver som består av en transportør, PLS med kontrollpanel og et system for logging, behandling og visualisering.

Forskjellige tester utført viser til at måling av strømtrekket alene *kan* påpeke forskjellige typer feil: om motoren blir kjørt for hardt, om transportbåndet er strammet for mye eller om det er for mye vekt på transportøren. Gruppen har kommet fram til at en AI med tilstrekkelig datainformasjon kan oppdage og varsle om slike feil.

Selv om gruppen ikke kan vise til en ferdig og helt fungerende prototype, kan det trekkes noen konklusjoner. Ved bruk av en mer optimalisert modell, flere sensorer og større mengder klassifisert data vil det være mulig å produsere et bedre PdM system ved hjelp av Anomaly detection.

Alt i alt er gruppen godt fornøyd med resultatet av sin oppgave, både i form av produkt og læringsutbytte, spesielt med tanke på de utfordringene som oppsto underveis. Dette er læring om hvordan prosjektutvikling fungerer i praksis samt at kandidatene har fått god erfaring som kan tas med ut i arbeidslivet.

7 REFERANSER

[1] Negnevitsky, Michael. *Artificial Intelligence A Guide to Intelligent Systems*. ISBN 9781408225745, Pearson Education Limited, 2011.

[2] GIT. 2019. *Om versjonskontroll* [Nettside] [oppsøkt mai. 2019]. Tilgjengelig på: *https://git-scm.com/book/no-nb/v1/Komme-i-gang-Om-versjonskontroll*

[3] Susto, G. A., Schirru, A., Pampuri, S., McLoone, S., & Beghi, A. 2015. *Machine Learning for Predictive Maintenance: A Multiple Classifiers Approach* 3. utgave [Nettside] [oppsøkt mai. 2019]. Tilgjengelig på: https://pure.qub.ac.uk/portal/files/17844756/machine.pdf

[4] Mallawaarachchi, Vijini. 2017. *10 Common Software Architectural Patterns in a nutshell* [Nettside] [oppsøkt mai. 2019]. Medium.com Tilgjengelig på: https://towardsdatascience.com/10-common-software-architectural-patterns-in-a-nutshell-a0b47a1e9013

[5] Wikipedia. 2019. *Agile software development* [Nettside] [oppsøkt mai. 2019] Tilgjengelig på: https://en.wikipedia.org/wiki/Agile_software_development

[6] Ward Cunningham. 2001. *Manifesto for Agile Software Development* [Nettside] [oppsøkt mai. 2019]. Tilgjengelig på: https://agilemanifesto.org/

[7] Agile Alliance. 2019. *Glossary, Extreme Programming* [Nettside] [oppsøkt mai. 2019]. Tilgjengelig på: https://www.agilealliance.org/glossary/xp/

[8] Wikipedia. 2019. *Programmerbar logisk styring* [Nettside] [oppsøkt april. 2019]. Tilgjengelig på: https://no.wikipedia.org/wiki/Programmerbar_logisk_styring

[9] Omron. 2019. *MX2* [Nettside] [oppsøkt april. 2019]. Tilgjengelig på: https://industrial.omron.no/no/products/mx2

[10] Omron. 2019. *S8VK-G* [Nettside] [oppsøkt april. 2019]. Tilgjengelig på: https://industrial.omron.no/no/products/s8vk-g

[11] Wikipedia. 2019. *Docker* (software) [Nettside] [oppsøkt februar. 2019]. Tilgjengelig på: https://en.wikipedia.org/wiki/Docker_(software)

[12] Docker Compose. 2019. *Docker Compose* [Nettside] [Oppsøkt februar. 2019]. Tilgjengelig på: https://docs.docker.com/compose/

[13] MQTT.fx. 2019. *MQTT.FX* [Nettside] [oppsøkt februar. 2019]. Tilgjengelig på: https://mqttfx.jensd.de/

[14] Postgresql. 2019. *PostgreSQL* [Nettside] [oppsøkt april. 2019]. Tilgjengelig på: https://www.postgresql.org/

[15] Timescale. 2019. *TimescaleDB* [Nettside] [oppsøkt april. 2019]. Tilgjengelig på: https://www.timescale.com/

[16] Microsoft. 2019. *ASP.NET* [Nettside] [oppsøkt april. 2019]. Tilgjengelig på: https://dotnet.microsoft.com/apps/aspnet

[17] TensorFlow. 2019. *TensorFlow* [Nettside] [oppsøkt mars. 2019]. Tilgjengelig på: https://www.tensorflow.org/

[18] Keras. 2019. *Keras* [Nettside] [oppsøkt mars. 2019]. Tilgjengelig på: https://keras.io/

[19] Matplotlib. 2019. *Matplotlib* [Nettside] [oppsøkt april. 2019]. Tilgjengelig på: https://matplotlib.org/

[20] Wikipedia. 2019. *Python* [Nettside] [oppsøkt mars. 2019]. Tilgjengelig på: https://no.wikipedia.org/wiki/Python

[21] Facebook. 2019. *React Native* [Nettside] [oppsøkt april. 2019]. Tilgjengelig på: https://facebook.github.io/react-native/

[22] Microsoft. 2019. *ReactXP* [Nettside] [oppsøkt april. 2019]. Tilgjengelig på: https://microsoft.github.io/reactxp/

[23] Wikipedia. 2019. *PyCharm* [Nettside] [oppsøkt april. 2019]. Tilgjengelig på: https://en.wikipedia.org/wiki/PyCharm

[24] Microsoft. 2019. *Visual Studio* [Nettside] [oppsøkt mars. 2019]. Tilgjengelig på: https://visualstudio.microsoft.com/

[25] Wikipedia. 2019. *C Sharp (programming language)* [Nettside] [oppsøkt mars. 2019]. Tilgjengelig på: https://en.wikipedia.org/wiki/C_Sharp_(programming_language)

[26] Wikipedia. 2019. *JavaScript* [Nettside] [oppsøkt april. 2019]. Tilgjengelig på: https://no.wikipedia.org/wiki/JavaScript

[27] NodeJS. 2019. *Node.JS* [Nettside] [oppsøkt april. 2019]. Tilgjengelig på: https://nodejs.org/en/about/

[28] Wikipedia. 2019. *npm* (software) [Nettside] [oppsøkt april. 2019]. Tilgjengelig på: https://en.wikipedia.org/wiki/Npm_(software)

[29] Wikipedia. 2019. *Webpack* [Nettside] [oppsøkt april. 2019]. Tilgjengelig på: https://en.wikipedia.org/wiki/Webpack

[30] GitLab. 2019. *GitLab* [Nettside] [oppsøkt mars. 2019]. Tilgjengelig på: https://about.gitlab.com/

[31] Wikipedia. 2019. *Anomaly detection* [Nettside] [oppsøkt april. 2019]. Tilgjengelig på: https://en.wikipedia.org/wiki/Anomaly_detection

[32] Devart. 2019. *dbForge Studio* [Nettside] [oppsøkt april. 2019]. Tilgjengelig på: https://www.devart.com/dbforge/postgresql/studio/

[33] Omron. 2019. *Sysmac Studio* [Nettside] [oppsøkt februar. 2019]. Tilgjengelig på: https://industrial.omron.no/no/products/sysmac-studio#featureshttps://industrial.omron.no/no/products/sysmac-studio

[34] Draw.io. 2019. *Draw.io* [Nettside] [oppsøkt mai. 2019]. Tilgjengelig på: https://about.draw.io/

[35] Wikipedia. 2019. Machine Learning [Nettside] [oppsøkt mai. 2019]. Tilgjengelig på: https://en.wikipedia.org/wiki/Machine_learning

[36] Wikipedia. 2019. Long short-term memory [Nettside] [oppsøkt mai. 2019]. Tilgjengelig på: https://en.wikipedia.org/wiki/Long_short-term_memory

[37] Wikipedia. 2019. Deep Learning [Nettside] [oppsøkt mai. 2019]. Tilgjengelig på: https://en.wikipedia.org/wiki/Deep_learning

[38] Wikipedia. 2019. Artificial neural network [Nettside] [oppsøkt mai. 2019]. Tilgjengelig på: https://en.wikipedia.org/wiki/Artificial_neural_network

[39] Wikipedia 2019. Recurrent neural network [Nettside] [oppsøkt mai. 2019]. Tilgjengelig på: https://en.wikipedia.org/wiki/Recurrent_neural_network

[40] Wikipedia 2019. Convolutional neural network [Nettside] [oppsøkt mai. 2019]. Tilgjengelig på: https://en.wikipedia.org/wiki/Convolutional_neural_network

[41] Microsoft. 2019. *Intellisence* [Nettside] [oppsøkt mars. 2019]. Tilgjengelig på: https://code.visualstudio.com/docs/editor/intellisense

8 VEDLEGG

8.1 Møte ang. Bachelor Oppgave

Lokasjon: Valderøy

Dato: 10.01.19

Tid: 13:00

Tilstede: Ole Andre Tomren, Andreas Rekdal, Rasmus Drønnen

MØTETS AGENDA:

1. Definere oppgaven, hva ønsker Optimar av studentene?
2. Hvilket utstyr og hvilke målinger av data kan vi gjøre?
3. Videre møter, kontaktpersoner for Automasjon

MØTEREFERAT:

1. Det ble under møtet stilt krav av studentene at Optimar skulle være litt mer klar i hva de ønsket ut fra oppgaven. Det har tidligere vært snakk om målinger av normalverdier, bruk av AI på testrigg eller egne målinger. For å kunne definere en oppgave må studentene ha mer klare linjer om hva det skal jobbes mot. Blir bestemt opptrening av AI mot innhenting av data.
2. Litt spørsmål angående hvilke data og hvilke sensorer som trengs for å hente data. Fjordlaks AS har lenge vært nevnt som mulig objekt for uthenting av data siden de har en blanding av brukte og godt brukte maskiner. Må sjekkes opp internt om det er muligheter for å sette opp måleutstyr her (og eventuelt hvilke målinger).
3. Avtales nytt møte en uke senere, studentene skal invitere veileder. Kontaktpersoner for automasjon blir invitert til møtet, samt noen fra avdelingen for vision (dataingeniører)

8.2 Møte med veileder

Lokasjon:	NTNU Ålesund
Dato:	15.01.19
Tid:	13:00
Tilstede:	Andreas Rekdal, Rasmus Drønnen, Stian Natland, Kjell Inge Tomren

MØTETS AGENDA:

1. Presentere oppgaven for veileder
2. Diskutere arbeidsmetodikk
3. Planlegge møte med veileder og oppdragsgiver

MØTEREFERAT:

1. Oppgaven ble presentert for veileder. I lag med gruppens plan på en løsning.
2. Scrum ble valgt som arbeidsmetodikk. Det ble planlagt møter med veileder hver 14. Dag på mandager.
3. Det ble planlagt møte hvor veileder og oppdragsgiver skulle møtes 21. Januar.

8.3 Møte Bachelor med Veileder og Oppdragsgiver

Lokasjon: Signalen. Valderøy

Dato: 21.01.19

Tid: 14:00

Tilstede: Ole Andre Tomren, Andreas Rekdal, Rasmus Drønnen, Stian Natland, Marius Nedrelid, Emil Dale Bjørlykhaug, Daniel Kvam

MØTETS AGENDA:

1. Hva trengs for å komme I gang med prosjektet?
2. Introduksjon mellom oppdragsgiver og veileder
3. Innspill fra andre organisasjoner I Optimar, automasjonsingeniører og dataingeniører
4. Datakvalitet, hvilke problemer er mest vanlige?

MØTEREFERAT:

1. Transportør og PLS trengs, det finnes allerede utstyr som studentene kan bruke. (Se ref. fra Optimar under)
2. Veileder hadde ikke anledning til å delta på møtet denne gangen.
3. Litt innspill om hva/hvordan vi kan bruke PLS til å hente ut data og hvilke data vi kan hente ut og relevans for dataene. Enkelte som jobber i Optimar har skrevet om lignende tidligere. (Se referat fra Optimar under)
4. Setter opp nytt møte der studentene kan intervjue forskjellige kontaktpersoner i Optimar når det kommer til hvilke feil som er normale. Ingen på møtet har noen erfaring med dette fra tidligere så det er bedre om det blir intervjuet direkte fra kilden!

Tilgjengelige ressurser hos Optimar:

- Starte med å bruke transportøren for pakkebord
- Ole Andre: Ordner PLS
- Marius: Hjelper med PLS og automasjonsoppsett
- Emil og Daniel: Bistår med kompetanse rundt maskinlæring og data analyse

Emil sjekker hvem (Anne-Marte eller Kenneth) som har gjort oppgave rundt vibrasjonsanalyse.

Ole sjekker om der er ledige kontor på brakkeriggen

8.4 Møte Bachelor med installatører/reparatører

Lokasjon: Signalen. Valderøy

Dato: 08.02.19

Tid: 14:00

Tilstede: Kenneth Synes, Hans-Kyrre Rørvik, Andreas Rekdal, Rasmus Drønnen, Stian Natland,

MØTETS AGENDA:

1. Intervju om de mest vanlige feil på transportører hos Optimar
2. Kostnader I forbindelse med utstyret som blir ødelagt
3. Er det noe operatører mener vi burde bestille som ekstrautstyr

MØTEREFERAT:

1. De mest vanlige formene for feil på utstyr fra Optimar er feilmontering eller feil under produksjon/design. De fleste maskiner fungerer godt over lengre tid men det er en del som begynner å bli utslitt når det har gått noen år. Det er da ofte I "snekka" som er giret eller kraftoverføringen fra motor til drivakslingen. Det er også en del feil som oppstår i andre typer lagre og med bånd en stund ut i livsløpet til en transportør men disse er mer betegnet som en slitedel. Litt småprat om vi kunne (med nok data) gitt de forskjellige delene «liv» som vi kunne trukket fra etterhvert som det var bruk av transportøren.
2. De største kostnadene kommer fra feilproduksjon hos Optimar, dette kan føre til forsinkelser og i verste fall forsinkelse av større prosesser (om det er deler som er kritiske for videreutvikling). For kunden er ikke pris på nye deler ofte et stort problem, men et større problem vil være nedetid av en fabrikk eller av en linje som følge av dette. Dette er det som er viktigst for Optimar og de ser lønnsomheten i å kunne gi beskjed til et fartøy om at de har deler som må byttes neste gang de er i havn (kanskje få til et system for å planlegge dette bedre).
3. Maskinistene har ikke hatt så mye med selve instrumentene å gjøre tidligere men etter litt diskusjon om hva slags forhold maskineriet utsettes for blir det enighet om at en vibrasjonsmåler kan være bra for hjullager. Ellers vil det å

kunne bruke temperatursensorer og logging av kraft brukt på forskjellige transportører nok være enkleste alternativ.

8.5 Møte Bachelor med teknisk tegner for Optimar

Lokasjon: Signalen. Valderøy

Dato: 22.02.19

Tid: 14:00

Tilstede: Bjørnar Andre Vik, Rasmus Drønnen, Stian Natland, Andreas Rekdal

MØTETS AGENDA:

1. Utviklingen av Testrigg for bachelorgruppe
2. Ekstra utstyr?
3. Estimert tid for ferdigstilling

MØTEREFERAT:

1. Siden det nå har vist seg at teststasjonen for pakking er i bruk for utvikling av et visionsystem må det bygges en ny testenhet til studentene. Det planlegges for hvor stor den trenger å være, samt hvor mye den må kunne frakte. Studentene sier den kan gjerne være liten og mobil så lenge det vil være mulig å produsere godt med data på den.
2. Ekstra utstyr ble ikke bestilt, det har ikke vært gjort tidligere så det vil øke tiden betraktelig.
3. Estimert tid for ferdigstilling, studentene har ut neste uke å komme med eventuelle endringer/justeringer de ønsker gjort. Ellers vil normal prosess være ca. 3 uker, litt avhengig av arbeidsmengde på både teknisk tegning og i produksjonsanlegget.

8.6 Møte med veileder

Lokasjon:	NTNU Ålesund
Dato:	16.05.19
Tid:	13:00
Tilstede:	Andreas Rekdal, Stian Natland, Kjell Inge Tomren

MØTETS AGENDA:

1. Hjelp fra veileder til endelig oppbygging og fullføring av oppgaven.

MØTEREFERAT:

1. Veileder svarte på diverse spørsmål angående oppbygging på oppgaven.
2. Veileder skal lese igjennom og påpeke eventuelle mangler i oppgaven.

8.7 Oversikt over perioder i XP

Periode (Loop)	Oppgave	Resultat
18.03-01.04	Prototype av datalogging. Oppsett og programmering av PLS. Lage treningsdata for AI modell.	Fungerende datalogging. PLS sender data til database.
01.04-15.04	Fungerende dataflyt PLS til front-end. Lage treningsdata for AI modell. Utvikle LSTM nettverk.	Fungerende visning av data i front-end. Fungerende trening og test data for AI. Ny modell for AI må utvikles.
15.04-29.04	Videreutvikle front-end datavisualisering. Trene opp AI modell. Kommunikasjon mellom AI og back-end	Formatert datavisualisering i front-end i for av en tabell. Fungerende AI modell.
29.04-13.05	Alarm til front-end Endring fra AI til sensordatavarsling. Optimalisere AI modell.	Fungerende varsling i front-end ved for tung belastning.

8.8 Forprosjektrapport

Forprosjekt: Prediktivt vedlikehold fra Optimar

KANDIDATNUMMER(E):			
997490, 997492, 997463			
DATO:	EMNEKODE:	EMNE:	DOKUMENT TILGANG:
02.02.19	IE303612	Bacheloroppgave	- Åpen
STUDIUM:		ANT SIDER/VEDLEGG:	BIBL. NR:
Bachelor i ingeniørfag - Data		9/0	- Ikke i bruk -

OPPDRAGSGIVER(E)/VEILEDER(E):
Optimar Giske AS/Kjell Inge Tomren

OPPGAVE/SAMMENDRAG:
Oppgaven går ut på å lage et prediktivt vedlikeholdssystem for frekvensstyrte motorer som benyttes i hovedsak av fiskeindustrien. Vi vil i første omgang fokusere på motorer brukt i transportører men med mulighet for å i fremtiden (etter endt bachelor) utvide til andre motorer. Oppgaven vil bli utført for Optimar og er relatert til Optimar Commander. Optimar Commander er et system laget for at man skal ha bedre kontroll over produksjon og prosesser, man kan enkelt få oversikt og statistikk som er bearbeidet av data fra de aktuelle anleggene. Commander vil i fremtiden ha støtte for Prediktivt vedlikehold, og Optimar ønsker å utforske hvordan dette lar seg gjøre i praksis allerede nå.

Denne oppgaven er en eksamensbesvarelse utført av student(er) ved NTNU i Ålesund.

INNHOLD

8.8.1 INNLEDNING

Oppgaven går ut på å lage et prediktivt vedlikeholdssystem for frekvensstyrte motorer som benyttes i hovedsak av fiskeindustrien. Vi vil i første omgang fokusere på motorer brukt i transportører men med mulighet for å i fremtiden (etter endt bachelor) utvide til andre motorer. Oppgaven vil bli utført for Optimar og er relatert til Optimar Commander. Optimar Commander er et system laget for at man skal ha bedre kontroll over produksjon og prosesser, man kan enkelt få oversikt og statistikk som er bearbeidet av data fra de aktuelle anleggene. Commander vil i fremtiden ha støtte for Prediktivt vedlikehold, og Optimar ønsker å utforske hvordan dette lar seg gjøre i praksis allerede nå.

8.8.2 BEGREPER

AI (Artificial intelligence): teknologien som gjør at datamaskiner er i stand til å løse samme tankeoppgaver som mennesker.

Datasett: En samling med data. Ofte hentet fra en database.

Prediktivt vedlikehold (Predictive maintenance): En teknikk som er utformet for å avgjøre tilstanden til driftsutstyr for å kunne estimere når vedlikehold skal utføres.

Avvik gjenkjenning (Anomaly detection): Identifikasjon av datapunkter, gjenstander, observasjoner eller hendelser som ikke samsvarer med det forventede mønsteret.

PLS: Programmerbar logisk styring. En datamaskin som brukes til å automatisere oppgaver som produksjon og kontroll.

8.8.3 PROSJEKTORGANISASJON

8.8.3.1 Prosjektgruppe

Studentnummer(e)

997490
997492
997463

Tabell: Studentnummer(e) for alle i gruppen som leverer oppgaven for bedømmelse i faget ID 302906

8.8.3.2 Styringsgruppe (veileder og kontaktperson oppdragsgiver)

Veileder: Kjell Inge Tomren

Kontaktperson oppdragsgiver (*Optimar*): Ole Andre Tomren

8.8.4 AVTALER

8.8.4.1 Arbeidssted og ressurser

- Tilgang på arbeidssted
- En transportør med PC som kommuniserer med PLS
- Marius Nedrelid hjelper med PLS og automasjonsoppsett
- Emil Dale Bjørlykhaug kompetanse rundt maskinlæring og data-analyse
- Daniel Kvam kompetanse rundt maskinlæring og data-analyse
- Anne-Marte eller Kenneth vibrasjonsanalyse

8.8.4.2 Gruppenormer – samarbeidsregler – holdninger

Gruppen skal i de fleste tilfeller komme til enighet, men i tilfelle uenigheter skal flertallet (2/3) bestemme.

Alle på gruppen skal utføre lik arbeidsmengde, så langt det lar seg gjøre.

Godt samarbeid er viktig, derfor skal medlemmene hjelpe hverandre hvis de får problemer med enkelte oppgaver.

Alle har likt ansvar for at oppgavene skal bli riktig gjennomført.

Holdningene til gruppen er at AI kun skal brukes til gode formål.

Bruken av AI skal kunne forklares og helst være selvforklarende.

Ved bruk av AI i dette prosjektet vil ikke bidra til et samfunn uten jobber, men heller endre på hvilke jobber som blir utført av arbeidere.

Vi vil ikke bidra til at virksomheter som tar i bruk AI skal bare øke kompensasjonen til eiere av bedriften men heller gi mer tilbake for alle.

8.8.5 PROSJEKTBESKRIVELSE

8.8.5.1 Problemstilling - målsetting - hensikt

Det skal bli laget et prediktivt vedlikeholdssystem for frekvensstyrte motorer som benyttes i hovedsak av fiskeindustrien. Under arbeidet med dette systemet vil vi også undersøke viktigheten av å få riktige målinger, samt om flere typer data vil gjøre resultatet mer nøyaktig eller ikke. Målet er å lage et system som fungerer for motorer som blir brukt i transportører. Vi har tre delmål, første vil være å hente inn data, andre vil være å lære opp et system på de dataene og vurdere behovet for vedlikehold til slutt vil vi lage et webbasert brukergrensesnitt. Til slutt vil vi lage et brukergrensesnitt for å vise frem resultatet.

8.8.5.2 Krav til løsning eller prosjektresultat – spesifikasjon

Målet vårt med oppgaven er ikke å utvikle et ferdig system, vi ønsker å prøve ut forskjellige måter til å lete etter feil. Løsningen skal inneholde bevis på at et slikt system skal kunne fungere kostnadseffektivt for bedriften og også gi merverdi for kunden. For Optimar AS vil det å kunne forutse eventuelle feil/mangler ved maskinvare eller eventuelt feil ved vedlikehold være et nytt steg mot veien på null nedetid. Under prosjektet vil vi også legge vekt på å finne ut om mer/forskjellig sensor data vil gi bedre resultater eller tidligere resultater.

8.8.5.3 Planlagt framgangsmåte(r) for utviklingsarbeidet – metode(r)

Det vil bli brukt agile framgangsmåte, det vil bli holdt god kontakt med oppdragsgiver og veileder. Arbeidsflyten i agile fremgangsmåter består av å lage en liste av all ønsket funksjonalitet. Oppdragsgiver bestemmer viktigheten for hver oppgave og rekkefølgen det skal utføres i. Utviklingsarbeidet foregår i den rekkefølgen oppdragsgiver har bestemt. Etter en bestemt tidsperiode vil ny liste laget og nytt møte holdt. Utviklingsarbeider vil fortsette slik til lansering.

Fordelene med agile fremgangsmåte er at det senker risiko og øker kvalitet med at det er åpent for raske endringer underveis. Siden partene arbeider tett kan endring i prioritet enkelt påvirke utviklingsprosessen.

Svakhetene med agile fremgangsmåter er at det kreves tett kontakt i vårt tilfelle med oppdragsgiver, det kan kreve en del tid både av utviklergruppen og oppdragsgiver/veileder.

Smidig (agile) programvareutvikling oppnås ved å følge det agile manifestet:

Vi finner bedre måter å utvikle programvare på ved å gjøre det selv og ved å hjelpe andre med det.
Gjennom dette arbeidet har vi lært oss å verdsette følgende:

- Personer og samspill fremfor prosesser og verktøy
- Programvare som virker fremfor omfattende dokumentasjon
- Samarbeid med kunden fremfor kontraktsforhandlinger
- Å reagere på endringer fremfor å følge en plan (Cunningham, 2001)

8.8.5.4 Informasjonsinnsamling – utført og planlagt

Vi har samlet opp en del data fra tidligere prosjekter som har hatt lignende vinkling som det vi har. Målet vårt med denne informasjonen er å unngå å bruke tid på å finne opp noe som allerede eksisterer. Vi vil benytte oss av det eksisterende grunnlaget til å undersøke eventuelt forbedringspotensial på sluttproduktet eller andre måter å komme oss dit på.

Vi vet at prediktivt vedlikehold er et "hot topic" om dagen og at det jobbes med mange plasser, derfor finnes det også en god del informasjon om det som er tilgjengelig for alle. Vi kommer til å forsøke å tilegne oss det nyeste innenfor slik forskning, samt det som er mest relevant i forhold til vårt prosjekt. Mesteparten av denne informasjonen vil bli hentet på internett, ofte rapporter fra tidligere bachelor/master prosjekt fra andre universitet.

- Deep Learning for Anomaly Detection: A Survey
  - https://arxiv.org/pdf/1901.03407.pdf
- Remaining useful life predictions
  - https://www.sciencedirect.com/science/article/pii/S0951832018307506

8.8.5.5 Vurdering – analyse av risiko

For å realisere prosjektet må de to første delmålene være gjennomført. Risikoelementer vil være å ikke klare å finne en passende algoritme for å trene opp systemet i det andre delmålet.

En annen risiko vil være at vi ikke får tilgang til nok data til å trene opp modellen vår. Om vi ikke har tilgang til nok data vil ikke vi ha mulighet til å få et godt resultat.

Det vil være særlig viktig å lykkes med å hente inn data fra motorene, slik at vi har noe å trene AI systemet på.

Trusler mot suksessen kan være at vi ikke klarer å framprovosere feil med motorene, slik at vi ikke finner ut hva som gjør at feil oppstår.

Uforutsette feil kan forekomme på transportøren vi skal bruke, dette kan forsinke arbeidet.

8.8.5.6 Hovedaktiviteter i videre arbeid

- Oppdragsgiver skal sette opp en transportør som vi kan jobbe med, denne består av et samlebånd med en oppkoblet PLS som kommuniserer med en datamaskin dette bør være i orden når forprosjektet er ferdig.
- Det vil bli undersøkt hvilke avvik som kan kobles mot hvilke data. Det vil bli forsøkt å finne flest mulig avvik med data som allerede måles av PLS-en som er oppkoblet. Om nødvendig vil flere sensorer bli koblet opp mot transportøren. Dette kan ta imellom 2 og 4 uker.
- Når innsamlede data er koblet opp mot avvik vil det bli testet ut hvilke avvik gjenkjennings algoritmer som vil passe best for vår bruk. Utvikling og optimalisering av denne algoritmen vil vare ut resten av prosjektet.
- Det vil bli laget et brukergrensesnitt for å vise resultat. Dette vil foregå parallelt med optimalisering av algoritmen og vil også vare ut resten av prosjektet.

8.8.5.7 Framdriftsplan – styring av prosjektet

8.8.5.7.1 Hovedplan

- Oppsett av transportøren vil oppdragsgiver utføre. Fram til dette er gjort kan ikke resten av oppgaven bli startet på.
- Undersøkelsen av hvilke data som kan kobles mot hvilke avvik. Dette vil bli startet på så fort vi har transportøren tilgjengelig. Vi vil først starte med å få samlet inn normaldata, deretter vil vi prøve å fremprovosere feil og se hvordan og hvilke data som endrer seg.
- Utvikling og testing av systemet som skal gi tilbakemelding på hvilke avvik som forekommer. Dette vil være den største oppgaven og vil ta lengst tid. Det burde bli startet så tidlig så mulig.
- Utvikle et brukergrensesnitt for å vise frem potensielle avvik som systemet varsler om.

8.8.5.7.2 Styringshjelpemidler

- JIRA vil bli brukt til å holde orden på hvilke oppgaver som blir utført og som skal bli utført og i hvilken rekkefølge oppgavene skal bli utført.
- Confluence vil bli brukt til å dokumentere og holde styr på løsningene vil har laget.

8.8.5.7.3 Utviklingshjelpemidler

- Fungerende transportør
- PLS
- En datamaskin

8.8.5.7.4 Intern kontroll – evaluering

- Undersøkelsen av hvilke avvik som kan knyttes mot hvilke data vil være ferdig når vi i enighet med oppdragsgiver er fornøyd.
- Om systemet fungerer vil vi kunne se det ved at det gir rett tilbakemelding på om avvik forekommer.

8.8.5.8 Beslutninger – beslutningsprosess

Beslutninger om presisering av oppgaven har blitt tatt under forprosjektet i lag med oppdragsgiver.

Viktige beslutninger igjennom hovedprosjektet vil bli tatt i lag med oppdragsgiver og veileder av oppgaven.

8.8.6 DOKUMENTASJON

8.8.6.1 Rapporter og tekniske dokumenter

- Bachelorrapport

8.8.7 PLANLAGTE MØTER OG RAPPORTER

8.8.7.1 Møter

8.8.7.1.1 Møter med styringsgruppen

Det vil bli hold regelmessige møte med veileder og god kommunikasjon med oppdragsgiver og ressurspersonell. Møter med veileder vil bli hold annenhver uke. Møter med oppdragsgiver og ressurspersonell vil bli holdt etter behov.

- Møter vil bli holdt med veileder annenhver uke, mandager eller tirsdager.

Dato	Tid	Innhold
04.02.19	14:00 – 14:30	Planleggings- og statusmøte

18.02.19	14:00 – 14:30	Planleggings- og statusmøte
04.03.19	14:00 – 14:30	Planleggings- og statusmøte
18.03.19	14:00 – 14:30	Planleggings- og statusmøte
01.04.19	14:00 – 14:30	Planleggings- og statusmøte
15.04.19	14:00 – 14:30	Planleggings- og statusmøte
29.04.19	14:00 – 14:30	Planleggings- og statusmøte
13.05.19	14:00 – 14:30	Planleggings- og statusmøte

8.8.7.1.2 Prosjektmøter

- Prosjektgruppen vil holde møter hver dag vi jobber. Hensikten med dette er å passe på at vi til enhver tid jobber mot samme mål og er enig med fremgangsmåten.

8.8.8 PLANLAGT AVVIKSBEHANDLING

- Ved mangel på data enten grunnet problem med innsamling, tilgang til ressurser, problem med transportøren eller andre grunner kan tilgjengelige datasett som blir brukt til avviksgjenkjenning bli tatt i bruk. Der ligger flere slike datasett tilgjengelig på nettet. Ansvaret til slike feil kan ligge hos enten oppdragsgiver eller prosjektgruppen.

8.8.9 UTSTYRSBEHOV/FORUTSETNINGER FOR GJENNOMFØRING

- For at oppgaven skal bli gjennomført kreves det at oppdragsgiver stiller med en fungerende transportør oppkoblet med en PLS som snakker med en datamaskin.

8.8.10 Referanser

Cunningham, W. (2001). *agilemanifesto.org*. Hentet fra Manifestet for smidig programvareutvikling: https://agilemanifesto.org/iso/no/manifesto.html