NTNU
Kunnskap for en bedre verden

Bruk av Raspberry Pi i tilstandskontroll.

En studie gjennomført i henhold til brukervennlighet
av Raspberry Pi i tilstandskontroll.

Erlend Randen

Kjell Erik Kalsvik

Frode Østby

Olav Oterholm Mære

Bacheloroppgave

Bachelor i maskiningeniør

Fakultet for ingeniørvitenskap (IV)

Norges teknisk-naturvitenskapelige universitet, 2019

NTNU
Kunnskap for en bedre verden

FAKULTET FOR INGENIØRVITENSKAP
Institutt for maskinteknikk og produksjon
7491 Trondheim

Besøksadresse:
R.Birkelands vei, 2B, Trondheim

RAPPORT bacheloroppgaven

Bruk av Raspberry Pi i tilstandskontroll

Condition based monitoring with Raspberry Pi

Prosjektnr.
MTP-D-2019-01

Forfattere
Frode Østby
Erlend Randen
Olav Mære
Kjell Erik Kalsvik

Oppdragsgiver eksternt

Oddvar Bjerkås v/ Trønder Energi Kraft

Dato levert	20	136
20.05.2019		

Veileder internt
Viggo Gabriel Borg Pedersen

ÅPEN

Kort sammendrag
Denne oppgaven har undersøkt muligheten til å bygge et rimelig, men også pålitelig tilstandssystem. Systemets spesifikasjoner er valgt på bakgrunn av turbinlagret til en Francis-turbin på Brattset kraftverk. Det er benyttet Raspberry Pi, temperatursensor, vibrasjonssensor og Python for å konstruere prototypen. Ytelsen av tilstandssystemet sammenlignes med troverdige kilder.

Stikkord fra prosjektet:
Tilstandsbasert vedlikehold, temperatur, vibrasjon, Raspberry Pi, Python, vertikal Francis-turbin, ADXL355, DS18B20
Condition based monitoring, maintenance, temperature, vibration, Raspberry Pi, Python, Vertical Francis turbine, ADXL355, DS18B20

NTNU
Kunnskap for en bedre verden

FORORD

Denne rapporten er et resultat av emnet TMAS3001 «Bacheloroppgave» fra utdanningsprogrammet maskiningeniør ved NTNU. «Bruk av Raspberry Pi i tilstandskontroll» er skrevet av fire studenter og gjennomført våren 2019 i samarbeid med NTNU og TrønderEnergi AS.

Prosessen med å gjennomføre en oppgave som alle på gruppen kan stille seg bak, har vært tidkrevende og utfordrende, men meget lærerik for alle medlemmer. Gruppen har benyttet mye av kunnskapen innenfor vår studieretning, drifts- og vedlikeholdsteknikk, i tillegg har oppgaven utfordret gruppen på fagområder som går utenfor pensum.

Gruppen ønsker å rette en stor takk til vår engasjerte og faglig dyktig veileder, Viggo Gabriel Borg Pedersen for oppfølging, veiledning og inspirerende diskusjoner. Takk til kontaktperson i TrønderEnergi, Oddvar Bjerkaas som har vært meget behjelpelig med svar på spørsmål og omvisning på Brattset Kraftverk.

Kjell Erik Kalsvik
Trondheim 20.Mai

Frode Østby
Trondheim 20.Mai

Erlend Randen
Trondheim 20.Mai

Olav Oterholm Mære
Trondheim 20.Mai

NTNU
Kunnskap for en bedre verden

Sammendrag

Denne bacheloroppgaven har undersøkt om det er mulig å bygge et lavkost elektronisk tilstandskontrollsystem med bruk av åpne kildekoder og rimelig elektronikk. Systemet skal måle flere tilstandsparametere samtidig. Utgangspunktet for valg av tilstandsparametere skal gjøres på bakgrunn av tilgjengelige målepunkter på en vertikal Francis-turbin hos TrønderEnergi AS. Det er videre undersøkt om tilstandskontrollsystemet gir pålitelige verdier, ved å sammenligne mot troverdige kilder.

For å besvare problemstillingen, er det identifisert mulige målepunkter og målområder som kan forventes. Ut fra dette er det laget en prototype som måler temperatur og vibrasjon. Prototypen benytter en Raspberry Pi 3B+ som datalogger, en digital temperatursensor og et MEMS-akselerometer. Programmeringen er gjennomført i Python 3.

Det digitale termometeret ble kalibrert mot en kjent kilde ved instituttet for energi og prosessteknikk. Kalibreringen av det digitale termometeret viste et avvik som stemte overens med databladet til sensoren. Avviket ble senere korrigert med programmering som benyttet kubisk interpolasjon.

Oppsettet med det digitale termometeret kan brukes til eksempelvis tilstandsovervåkning av turbinlager på Francis-turbin. Utprøving i felt må til for å verifisere funksjonsevnen til prototypen.

MEMS-akselerometeret er sammenlignet mot det portable instrumentet Leonova Diamond, hvor RMS-verdi og frekvensbilde er sammenlignet. Resultatet viser avvik mellom prototypen og Leonova Diamond. Avviket er så stort at det tyder på feil i målekjeden. Resultatene fra RMS-sammenligningen tyder på at integrasjonen av signalet ikke er utført korrekt. Sammenligningen av FFT-ene viser at signalbehandlingen, som er utført på signalet fra MEMS-akselerometeret, produserer et frekvensbilde som ligner på frekvensbildet fra Leonova Diamond. Avvik i amplitude og avvik i frekvens tyder på feil i signalbehandlingen.

Oppsettet med MEMS-akselerometeret trenger gjennomgang av hele målekjeden og signalbehandlingen i Python. Deretter må det utføres sammenligning av måleresultat mot verifiserbar målekjede. Utprøving i felt må til for å verifisere funksjonsevnen til prototypen.

SUMMARY

This bachelor thesis has examined the possibilities for building a low-budget unit for condition monitoring with the use of open source codes. The unit will measure several parameters simultaneously. Regarding the choice of sensors, we based the choice on available measuring points on the vertical Francis turbine at TrønderEnergi AS. In addition, the system will be compared to credible sources to determine the validity of the measurements.

To address this assignment, we have identified possible data acquisition locations and obtained expected parameter values. From this information, we have produced a prototype which collects vibration and temperature data. The prototype hardware consists in general of a Raspberry Pi 3B+, a digital thermometer, and a MEMS accelerometer. The software is developed in Python 3.

To calibrate the digital thermometer, the deviation errors were mapped using a calibration instrument at the Department of Energy and Process Engineering. The deviation was of a similar character of that described in the thermometer data sheet. The deviation was later minimized by applying a calibration function in the prototype software.

The temperature measuring system can be used for CBM on a Francis turbine shaft bearing. However, it is necessary to execute a field test to determine the prototype functionality.

The MEMS accelerometer got compared to the portable instrument Leonova Diamond, where we compared the RMS values and frequency domain. The results showed a deviation between the prototype and Leonova Diamond. The deviation error was so great that it implied an error in the measuring chain. The difference of the RMS values is so massive that they imply wrong integration of the acceleration signal. Comparison of the FFTs, shows that the FFT from the MEMS accelerometer is relatively similar to the FFT from the Leonova Diamond. However, amplitude- and frequency-deviations in the FFTs implies wrong signal processing

The vibration measuring chain requires complete troubleshooting. Afterward, it is necessary to compare the new measuring chain, to a validated measuring chain. In addition, it is necessary to execute a field test to determine the prototype functionality.

NTNU
Kunnskap for en bedre verden

FORKORTELSER

NTNU	Norges teknisk-naturvitenskapelige universitet
ISO	International Organization for Standardization
IoT	Internet of Things
NS	Norsk standard
FFT	Fast Fourier-transformasjon
LOR	Lines of resolution
RMS	Root mean square
RMS_V	Root mean square velocity
RMS_A	Root mean square acceleration
CE	Conformité Européenne
ADC	Analog-to-digital converter
ODR	Output data rate
SCL	Serial clock
SDA	Serial data
g	$9{,}81 m/s^2$
USB	Universal Serial Bus
Hz	Hertz
RPM	Revolutions per minute
VNC	Virtual network computing
GPIO	General purpose input output
HDMI	High Definition Multimedia Interface
I^2C	Inter-Integrated Circuit
WIFI	Wireless Fidelity
PC	Personal Computer
CRC	Cyclic Redundance Check
MSB	Most Significant Bit
LSB	Least Significant Bit
MEMS	Mikroelektromekaniske Systemer
EBL	EnergiBedriftens Landsforening
LCD	Liquid Crystal Display
SPI	Serial Peripheral Interface

UART	Universal asynchronous receiver-transmitter
SMBus	System Management Bus
IP	International Protection
PCB	Printed Circuit Board
IDLE	Integrated development environment
GUI	Graphical User Interface

TABELLISTE

NTNU
Kunnskap for en bedre verden

FIGURLISTE

Innholdsfortegnelse

1 Innledning

1.1 Motivasjon for å gjøre oppgaven

Gjennom flere tiår har vi sett en enorm utvikling innenfor vedlikehold, og spesielt innenfor Internet of Things (IoT) og Industri 4.0. Industri 4.0 går innenfor 4 hovedpunkt, som er tingene, nettforbindelse, data og analyse. Aktørene i markedet ønsker å kunne forutse vedlikehold før alvorlige hendelser inntreffer. På bakgrunn av dette har gruppen fått presentert en oppgave fra veileder Viggo B. Pedersen, som retter seg inn mot tilstandskontroll. Oppgaven kombinerer kjente emner fra studieløpet (Vedlikehold og driftssikkerhet) og ukjente fagområder, slik som programmering, digitalteknikk og produktutvikling.

1.2 Hensikten med oppgaven

Denne oppgaven skal undersøke muligheten for at en gruppe uten noe dyptgående kunnskaper eller erfaringer innen elektronikk, kan sette sammen et **lavkost** elektronisk tilstandskontroll-**system**. Systemet skal basere seg på overvåkning av temperatur, vibrasjon eller trykk eller en kombinasjon av disse. Systemet skal benytte **open source-koder** og lett tilgjengelige, billige og **kommersielle komponenter**.

1.3 Målgruppe for oppgaven

Interessenter	**Primær/sekundær**	**Kommentar**
Viggo Gabriel Borg Pedersen	Primær	Veileder.
TEK	Primær	TEK -støttespiller i forhold til prosjektoppgaven.
NTNU	Primær	Oppgaven skrives for NTNU.
Studenter	Sekundær	Mulig bruk i videre utdanning.

Tabell 1 Interessenter

Denne oppgaven er i samarbeid med TrønderEnergi AS og NTNU, som også er våre nøkkelpartnere. Det vil være en fordel å ha grunnleggende kunnskaper innenfor vedlikehold og driftssikkerhet, elektronikk, signalbehandling og vibrasjon.

1.4 Resultatmål

1	En ståstedsanalyse som inneholder men ikke er begrenset til å: Identifisere målepunkter på TEK sin turbin ved Brattset for måling av temperatur, trykk og vibrasjon med Raspberry Pi – identifisere måleområde for disse punktene Identifisere sensorer egnet for måling i forhold til de identifiserte målepunktene. Sensorene må være kompatible med Raspberry Pi. Identifisere Raspberry Pi sine kapasiteter og begrensninger **Ståstedsanalysen vil legge grunnlag for hvilken av de 3 måleparameterene, trykk, temperatur og vibrasjon som skal videreføres i oppgaven**
2	Utvikle en prototype datalogger med Raspberry Pi og utvalgte sensorer. Måledataene skal lagres i et praktisk format for videre analyse.
3	Lage en manual som beskriver programmeringen og sammenkoblingen av Raspberry Pi med sensorer. Datablad på komponentene legges ved oppgaven.
4	Verifisering av prototype mot kjent kilde.

Tabell 2 Resultatmål

1.5 Avgrensing

For at oppgaven skal bli gjennomførbar med tilgjengelige ressurser, er følgende avgrensninger satt av gruppen og veileder.

Raspberry Pi skal benyttes som datalogger og master for valgte sensorer. Tilstandskontrollsystemet konstrueres rundt Raspberry Pi sine kapasiteter og begrensninger.

Python skal benyttes som programmeringsspråk, både i system-software og signalbehandling.

Systemet tar utgangspunkt i et turbinlager på en vertikal Francisturbin med måleparameterene vibrasjon og temperatur.

Komponenter som skal benyttes skal ha lav kostnad og være kommersielt tilgjengelige.

1.6 Disposisjon

Kapittel 1 innleder med avsnitt for å forberede leseren på hva man kan forvente.

Kapittel 2 gir et overblikk over hvordan gruppen har gått frem og hvilke metoder som er brukt til informasjonsinnhenting.

Kapittel 3 og 4 presenterer grunnleggende teori om vedlikehold, vibrasjon og signalbehandling, for å kunne sette sammen et pålitelig tilstandssystem.

Kapittel 5 viser resultatmål 1 i form av en ståstedsanalyse.

Kapittel 6,7 og 8 presenterer resultatmål 2 og 3, i form av prototype, brukermanual og software-oversikt.

Kapittel 9 presenterer resultatmål 4, med fremgangsmetoden og resultatet fra verifiseringen til prototypen.

Kapittel 10 diskuteres valgene som er gjort for å besvare resultatmålene.

Til slutt oppsummeres arbeidet i en konklusjon og en anbefaling for videre arbeid i kapittel 11.

For å fremstille sentrale elementer av oppgaven er det benyttet bilder og figurer. Der hvor det ikke opplyses om kilde, er det gruppen som står som fotograf. Figurene i kap. 7 og 8 er ikke nummerert i figurlisten, slik at man har muligheten til å bruke kapitlene separert fra rapporten.

2 Forskningsmetode

I dette kapitlet gjør man rede for hvordan gruppen metodisk har håndtert denne bacheloroppgaven. Formålet er å fortelle hva gruppen har gjort i forskningsprosessen, forklare hvorfor det er blitt gjort, samt forsøke å reflektere over de valg som er tatt.

Valg av forskningsmetode er en av de avgjørende faktorene for suksess. Resultatmålene og problemstillingen som skal undersøkes vil fungere som en veiledning til valg av forskningsmetode. Oppgaver hvor det er gjort lite forskning på tidligere vil en kvalitativ metode egne seg godt, mens i forskning på mindre komplekse temaer hvor det er gjort flere tidligere undersøkelser, vil en kvantitativ metode som regel være best egnet. (Thagaard, 2018) Kvalitative data er ikke-tallfestbare, mens kvantitative data er målbare (Larsen, 2017).

Denne oppgaven kombinerer kvalitative og kvantitative metoder for å besvare resultatmålene. Ved å bruke de kvalitative metodene som forberedelse til kvantitative metodene vil en sikre seg et visst empirisk grunnlag for å konstruere best mulig forarbeid før selve hovedarbeidet. På den måten vil man styrke tilliten til analyseresultatene og gyldigheten til metoden som blir testet. (Holmen & Solvang, 1996). De aktuelle metodene som er brukt er vist i tabellen under.

Kvalitativ metode	**Kvantitativ metode**
Litteraturstudie	Verifisering (beskrevet i kapittel 9)
Dokumentstudier	
Intervju	
Produktutvikling	

Tabell 3 Oversikt over valgte metoder

2.1 Datainnsamling

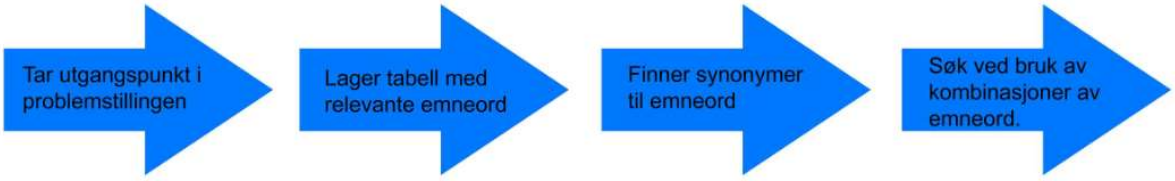
Datamaterialet vil utgjøre enhver kvalitativ forskningsartikkels ryggrad (Leseth & Tellemann, 2014). Oppgaven tar for seg både tilstandskontroll, elektroteknikk, vibrasjonsmåling, målemetoder, signalbehandling og programmering. Vibrasjonsmåling i seg selv blir i dag sett på som et eget fagfelt. Man er derfor avhengig av en god teoretisk forståelse for å kunne gjennomføre resultatmålene. Gruppen vil beskrive de tre datainnsamlingsmetodene som er valgt i denne oppgaven, for å skaffe seg en sterk faglig kompetanse.

2.1.1 Litteraturstudie

For å skaffe et teoretisk grunnlag til forankringen for gjennomføring av oppgaven, er det lagt tyngde på bruk av det som kan ansees som sikre kilder for å samle inn relevant teori.
Litteraturen er innhentet ved bruk av databaser ved NTNU universitetsbibliotek, og kvalitetssikret litteratur og anerkjente informasjonskilder på anbefaling fra veileder.
Følgende databaser er brukt:

- Oria
- Compendex
- Web of Science

Søkene som gjennomføres i universitetsbiblioteket basere seg på emneord. For å fremskaffe relevant litteratur, ble det brukt følgende prosedyre:



Figur 1 Prosedyre litteratursøk

Ved å bruke databaser tilknyttet NTNU sikrer man seg pålitelige kilder.
Dette bidrar til å styrke teorikapittelets reliabilitet og validitet, men personlige tolkninger av litteraturen kan være med å svekke teorikapitlenes tyngde. For å bedømme kildene ble VIKOs anbefaling om å vurdere nøyaktighet, troverdighet, egenhet og objektivitet fulgt (NTNU-VIKO, 2019)

Industri 4.0 er under utvikling og vil komme med rask og eksponentiell effekt, dermed er litteraturen under kontinuerlig utarbeidelse. Derfor har gruppa valgt å benytte seg av tidsartikler som går bedre i dybden, og kvalitetssikret litteratur utdelt av veileder.

2.1.2 Dokumentstudier

Når man bruker dokumentstudier, vil man benytte seg av dokumenter skrevet av andre i forskningen. Dokumentene vil skille seg fra data som forskeren henter inn gjennom feltarbeidet, hvor de er skrevet for et annet formål enn det forskeren skal benytte dem til (Thagaard, 2018). Det å velge dokumenter vil være en silingsprosess hvor man begrunner nøye hvilke dokument man tar med, og hvilke man utelukker. I forbindelse med denne oppgaven har gruppen valgt å benytte seg av standarden ISO 20816.

ISO 20816 gir retningslinjer for akseptabel vibrasjon på lageret, bærelageret og lagerhuset i hydrauliske kraft- og pumpeanlegg når de opererer innenfor normale driftsforhold. Ved utvelgelsen av vibrasjonssensor ble aksjonsgrensen for akseptabel RMS-verdi benyttet. ISO 20816 har med det hatt innvirkning på de empiriske slutningene gruppen har kommet frem til (ref. kap. 5.1.4).
En fordel med å bruke standarder er kvaliteten, standarden har innbakt kvalitetsnivå vurdert av spesialister. Den er et initiativ fra interessegrupper som sitter på en ekspertvurdering innenfor temaet, hvor den oppdateres kontinuerlig. Dette bidrar til å styrke validiteten til resultatet. Det at det kreves betydelig kompetanse for å anvende standarden, samt at noen standarder åpner for personlige tolkninger, kan bidra til å svekke troverdigheten. Gruppen vil derfor legge vekt på å bygge en faglig kompetanse innenfor standarden før man velger å benytte seg av den.

2.1.3 Intervju

I kvalitative metoder brukes det ofte strategiske utvalg. Man velger deltagere med egnede kvalifikasjoner eller egenskaper til å besvare resultatmålene og undersøkelsens teoretiske perspektiver (Thagaard, 2018) For å besvare resultatmålene i oppgaven, mener gruppen det var nødvendig å knytte kontakt med fagpersoner som sitter på bred kompetanse innenfor sitt fagfelt. Under er en tabell av personer som har stilt sin kompetanse til disposisjon.

Intervjuobjekt	**Stilling/funksjon**	**Kompetanse**
Viggo Gabriel Pedersen	Universitetslektor og veileder	Drift- og Vedlikeholdsteknikk Kvalitetssikring
Oddvar Bjerkås	Driftsingeniør og kontaktperson	Drift og vedlikehold Francis-turbin
Antoine Rauzy	Professor	Databehandling
Aleksander Mosand	Ingeniør	Prosessteknikk
Arild Sæther	Avdelingsingeniør	Mekanisk produksjon
Dominik Osiniski	Universitetslektor	Elektroniske systemer
Dag Roar Hjelme	Professor	Elektroniske systemer
Olav Aleksander Myrvang	Overingeniør	Elektroniske systemer
Berit Vinje Kramer	Overingeniør	Materialteknologi
John Inge Edvardsen	Overingeniør	Materialteknologi

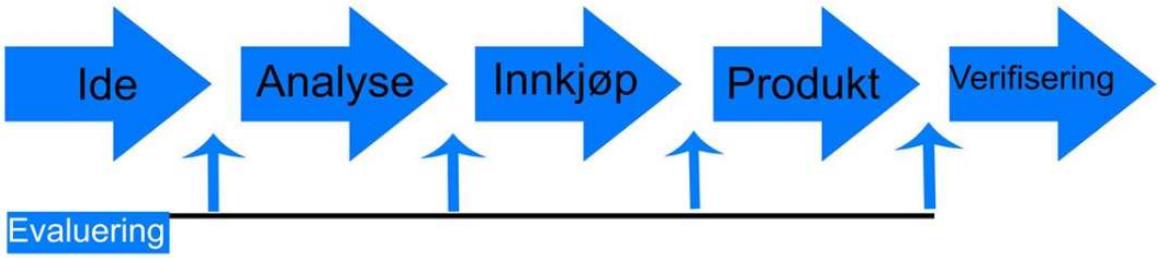
Tabell 4 Intervjuobjekter

Hvordan man vil utforme intervjuet i forskningsbasert sammenheng, vil være forskjellig fra hva man vil oppnå av intervjuet. Man skiller i hovedsak mellom tre metoder, uformelt, semi-strukturert og strukturert intervju. De fleste intervjuene gruppen har gjennomført har bestått av semi- strukturerte intervju, ansikt til ansikt. Et semi-strukturert intervju kjennetegnes ved at det under selve intervjuet ble det stilt åpne spørsmål og vist interesse for situasjonen, slik at man videre fikk belyst noen aktuelle innfallsvinkler. En styrke med å gjennomføre intervjuet på denne metoden er at intervjupersonen får en følelse av å være en ressurs for oppgaven. Svakheten ved å intervjue personen slik, er etterarbeidet og analysen (Dalland, 2007). Gruppen var derfor bevisste på å notere ned viktige forklaringer etter intervjuet.

Det at man kun har benyttet et fåtall intervjuobjekt ved hver problemstilling, kan argumenters som en svakhet for validiteten. Men gruppen ser på intervjuobjektene som meget faglig sterke innenfor sine fagfelt og som pålitelige kilder. En utfordring med kvalitative intervju er å gå i dybden, det innebærer at antall informanter ikke må være for omfattende (Dalland, 2007).

2.2 Produktutvikling

For å utvikle et produkt, ble modellen Stage-Gate fra Robert G. Cooper benyttet. Modellen er delt opp i 5 faser, med en kontinuerlig evaluering mellom fasene. (Cooper, 2011)



Figur 2 Produktutviklingsmodell

Fase 1: Komme opp med en ide på bakgrunn av markedets behov.
Fase 2: Utarbeide en ståstedsanalyse for å kartlegge produktspesifikasjonene.
Fase 3: Benytte innkjøpstrategi på bakgrunn av spesifikasjonene i ståstedsanalysen.
Fase 4: Utvikle produktet i form av en fysisk prototype.
Fase 5: Prototypen testes og verifiseres etter de gitte krav til funksjonalitet og ytelse.

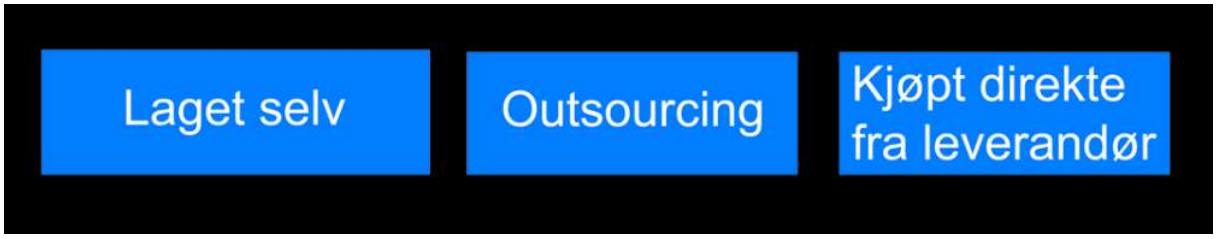
Bakgrunnen for valget av Coopers Stage Gate-modell, er fokuset den har på markedsorientering og forarbeid.

Dagens produkter innen tilstandskontroll kan koste veldig mye og brukeren kan bli avhengig av leverandøren. Ved å identifisere et behov for et billig tilstandskontrollapparat i markedet, ble ideen om å utvikle et lavkost tilstandskontrollsystem satt sammen med veileder. En av de viktigste suksesskriteriene for å lykkes med nye produkter er å se markedets behov (Ulrich & Eppinger, 2012)

Ved å gjennomføre en grundig ståstedsanalyse, vil tekniske utfordringer og problemer kunne oppdages tidlig. Kvaliteten på produktet blir i stor grad definert av analysen gjort i forkant av produktgenereringen. Et grundig forarbeid før utformingen av produktet vil gi en positiv innvirkning på suksessraten til produktet (Cooper, 2011)

2.2.1 Innkjøpsstrategi

På bakgrunn av spesifikasjonen utarbeidet i ståstedsanalysen, kunne man gå til anskaffelse av komponenter til prototypen. Komponentene ble anskaffet på 3 metoder:



Faktorer som bestemmer metoden:

- Pris
- Kvalitet
- Leveringstid
- Kompetanse

Elektroniske komponenter ble kjøpt fra pålitelige og kvalitetssikrede leverandører. Følgende leverandører ble brukt:

- Elfa
- Adafruit
- Mouser
- Kjell og Company

Ved å benytte seg av store auksjonsnettsteder som eBay eller Amazon, kunne man fått komponentene til en vesentlig lavere pris. Men usikkerhet knyttet til tollavgift, leveringstid og kvalitet, valgte gruppen å benytte seg av pålitelige leverandører som sikret rask levering. Kvalitet begrunnes opp mot CE-merking. CE-merking sender signal til offentlige organer som driver kontroll av produkter, om at produktet oppfyller de sikkerhetskrav som gjelder i EØS, og at dette kan dokumenteres (Norsk standard, 2019)

Utfresingen av vibrasjonssensor huset ble outsourcet. Lang leveringstid og usikkerhet knyttet til når produktet var ferdig, ga en negativ effekt på progresjonen til ferdigstillelse av systemet. Outsourcing øker avhengigheten og fører til kontinuerlig oppfølging av leverandøren (Weele, 2014). Men siden gruppens manglende kompetanse på fresing og tilgang til utstyr, står man for at dette var den beste løsningen. Man optimaliserte produktet ved bruken av kunnskap, utstyr og ekspertise hos en tredjepart (Weele, 2014)

3 Vedlikehold

I kapittelet 3 og 4 presenteres teori, som legger grunnlaget for å kunne gjennomføre oppgaven. Hvor oppgaven består av å lage et tilstandsovervåkningssystem med bruk av vibrasjons- og temperatur-sensor basert på Raspberry Pi 3B+ og MEMS-teknologi.
Den teknologiske utviklingen, med stadig billigere tilstandskontrollutstyr, kraftigere datamaskiner og raskere datanettverk, har åpnet for nye muligheter innen vedlikehold.

3.1: Historikk:

Maskiner i drift har alltid vært utsatt for slitasje, som gir behov for vedlikehold. Vedlikehold defineres som "En kombinasjon av alle tekniske og administrative aktiviteter, inkludert ledelsesaktiviteter, som har til hensikt å opprettholde eller gjenvinne en tilstand som gjør en enhet i stand til å utføre en krevd funksjon." (Bye, 2009)
De industrielle revolusjonene har påvirket hvordan vedlikehold utføres. Fra å fokusere på korrektivt vedlikehold, hvor reparasjoner utføres etter feil har oppstått, har vedlikeholdet gradvis gått over til å hindre feil i å oppstå, for å redusere nede-tida til maskineriet til et minimum. De industrielle revolusjonene kan deles inn i fire epoker, hvor vi er inne i den fjerde nå.

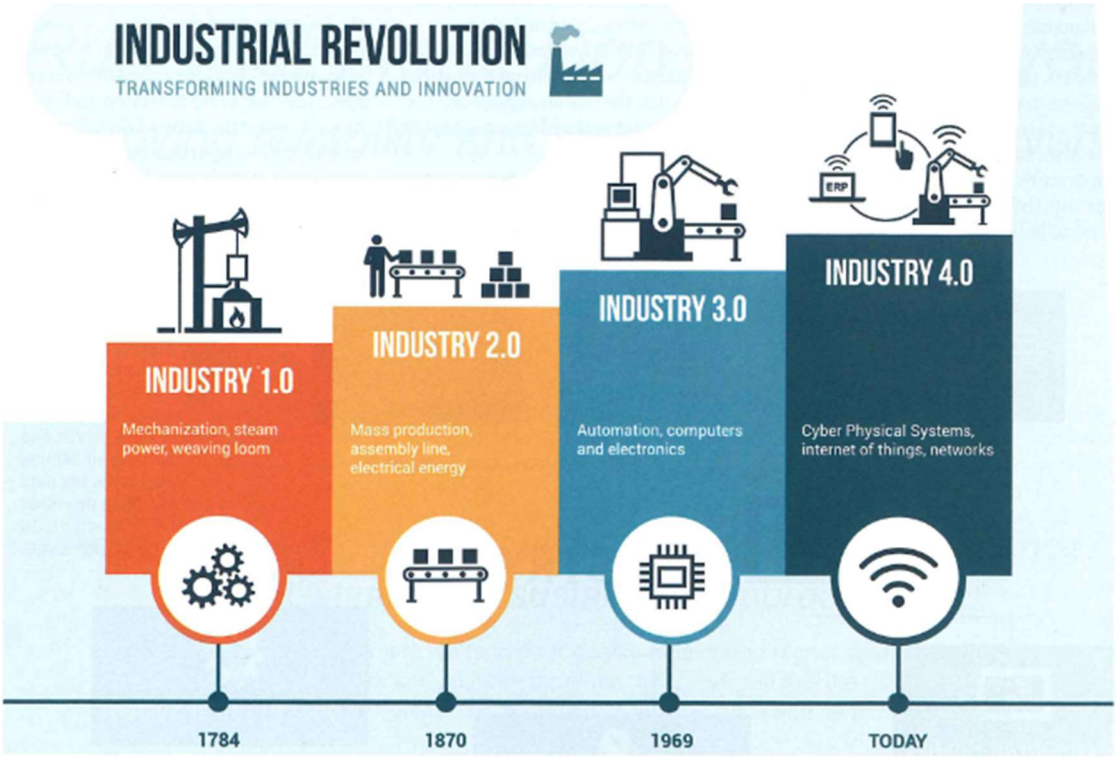
Den første industrielle revolusjon er det som tradisjonelt er sett på som den originale industrielle revolusjonen. Kjennetegnes ved bruk av vannkraft og dampkraft, mekanisering, og fabrikkproduksjon. Fra omtrent 1760 til 1840.

Den andre industrielle revolusjonen er videreutviklingen av den første. Den kjennetegnes ved masseproduksjon, industriell standardisering, samlebånd, elektrisitet og kommunikasjonsteknologi. Fra omtrent 1870 til 1914. Den andre industrielle revolusjonen, sammen med politiske omveltninger, ga grobunn til mye av vedlikeholdsteknologien som fortsatt brukes. Henry Ford introduserte 40-timers arbeidsuker for arbeiderne sine. Arbeidere fikk flere rettigheter, og mer ansvar knyttet til produksjonen, sammenlignet med den første industrielle revolusjonen. Større eierskap til eget arbeid, gjorde det mulig for arbeidere å ta vare på produksjonsmaskinene, og det bidrog til et større fokus på vedlikehold.

Før Andre Verdenskrig, var fokuset på korrektivt vedlikehold. Nedetider i industrien ga grobunn for preventivt vedlikehold, for å kunne produsere mest mulig uten stopp. Vedlikeholdsoperasjoner gikk fra å reparere feil, til å hindre feil fra å oppstå. Preventivt vedlikehold har gått fra å gjennomføres med faste tidsintervaller, til å gradvis gå over til å bli gjennomført etter tilstandsovervåkning, noe som gir oss prediktivt vedlikehold. Prediktivt vedlikehold gir et større fokus på å utføre vedlikehold når det er nødvendig, basert på utstyrets tilstand.

Den tredje industrielle revolusjonen kalles også den digitale revolusjonen. Kjennetegnes ved videreutviklingen av informasjonsteknologi og datamaskiner. Transistoren ble funnet opp i 1947, og mikroprosessoren i 1969. Transistoren og mikroprosessoren legger grunnlaget for starten på den tredje industrielle revolusjon. Dette la grunnlaget for automatisering og datastyring i produksjonsprosessen, og kontinuerlig tilstandsovervåkning i vedlikeholdsprosessen.

Den fjerde industrielle revolusjon er også den digitale revolusjonen. Kjennetegnes ved utviklingen av bruk av digitale nettverk, kunstig intelligens og mindre tydelige skillelinjer mellom fysiske, digitale og biologiske domener. Dette åpner for moderne vedlikehold.

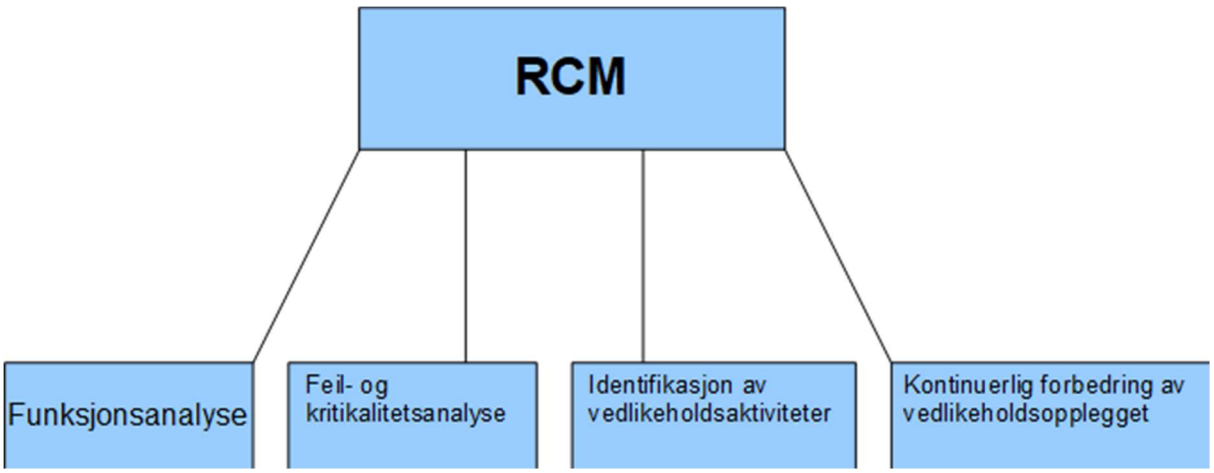


Figur 3 De industrielle revolusjoner (estudie, 2019)

3.2: Moderne vedlikehold:

Moderne vedlikehold er en del av den fjerde industrielle revolusjonen. Under de tidligere industrielle revolusjonene, har vedlikeholdsstrategien vært fokusert på korrigerende vedlikehold. Moderne vedlikehold skifter dette fokuset over til forebyggende og prediktivt vedlikehold.
Vedlikeholdsstrategi: *"Styringsmetode som brukes for å oppnå vedlikeholdsmålene";* (NS-EN 13306, 2017)
For å redusere feilhendelser, kostnader, mens man samtidig øker sikkerheten, påliteligheten, og avkastningen, trenger man en vedlikeholdsstrategi. Vedlikeholdsstrategien må tilpasses lovkrav, økonomi, sikkerhet og påliteligheten til systemet vi utvikler en vedlikeholdsstrategi for.

Vedlikeholdsstrategien legger basisen for vedlikeholdsprogrammet. Vedlikeholdsprogrammet er det spesifikke vedlikeholdsarbeidet som utføres for et anlegg eller en maskin. Optimalisering av moderne vedlikeholdsprogram er avhengig av en metodikk eller prosess som ivaretar forbedringsarbeid. For å forbedre påliteligheten til systemer, ble Reliability Centered Maintenance-konseptet utviklet av flyindustrien i USA på 70-tallet. RCM kan deles inn i fire aktiviteter;



Figur 4 RCM konsept

Vedlikeholdsevnen defineres som evnen til å ha riktig vedlikeholdsstøtte på nødvendig sted til å utføre den krevde vedlikeholdsaktiviteten på et gitt tidspunkt eller i et gitt tidsintervall. (Bye, 2009)
Vedlikeholdsevnen er avhengig av vedlikeholdsvennligheten til enheten.

Vedlikeholdsvennlighet: "enhetens evne under gitte bruksforhold til å opprettholde eller gjenopprette sin tilstand slik at den kan utføre krevd funksjon, når vedlikeholdet utføres under gitte forhold og med gitte prosedyrer og ressurser" (NS-EN 13306, 2017)
Beskriver hvor effektivt og enkelt det er å drive vedlikehold på enheten og systemet. Enkel tilgang på dokumentasjon, deler, og verktøy er viktig for vedlikeholdsvennligheten. (Bye, 2009)

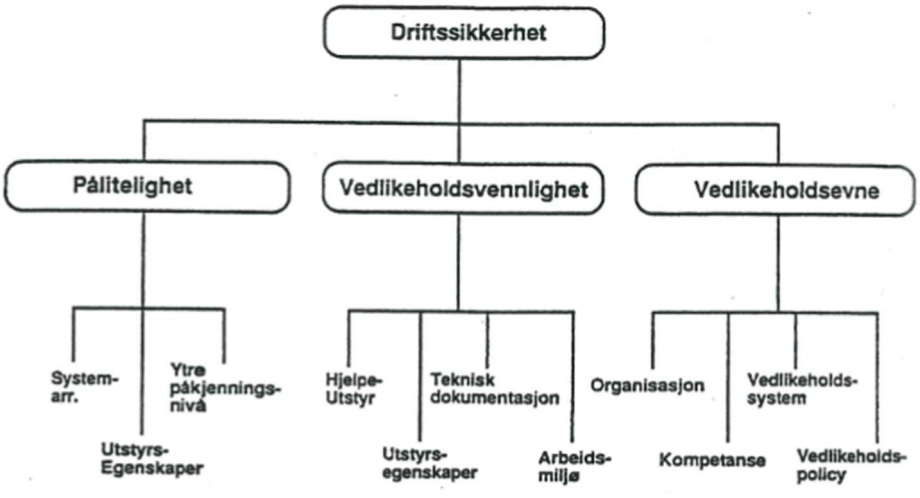
3.3 Vedlikeholdstyper:

3.3.1: Korrigerende vedlikehold; *"Vedlikehold som utføres etter at en feil er funnet, og som har som formål å gjenopprette en enhet til en tilstand der den kan oppfylle krevd funksjon"* - (NS-EN 13306, 2017)

Dette er den eldste formen for vedlikehold. Kjør maskineriet til det bryter sammen, og reparer det når det bryter sammen. Et ensidig fokus på korrigerende vedlikehold påvirker påliteligheten ved at vi får uforutsigbare stopp i produksjonen og nedetider.

Pålitelighet er definert som; "enhetens evne til å oppfylle krevd funksjon under gitte forhold innenfor et gitt tidsintervall". (NS-EN 13306, 2017)
Pålitelighet avhenger av levetid og feilrate. En lang levetid og en lav feilrate tyder på høy pålitelighet. Klar og tydelig oppfølging av spesifikasjonskrav er en av forutsetningene for å oppnå ønsket pålitelighet. I tillegg er korrekt drift og installasjon viktig for å oppnå den ønskede påliteligheten. Påliteligheten påvirkes i tre faser; innkjøp, installasjon, og drift. Under innkjøp er det kravspesifikasjonen og kvaliteten på det kjøpte produktet som bestemmer påliteligheten. Påliteligheten kan forverres under installasjon, selv om både drift og kravspesifikasjon er korrekt, grunnet feil i installasjonsprosessen. (Bye, 2009)

Påliteligheten, vedlikeholdsvennligheten og vedlikeholdsevnen påvirker driftssikkerheten, som er definert som; "evne til å fungere som det kreves og slik det kreves" (NS-EN 13306, 2017). Driftssikkerheten til et vannkraftverk er avhengig av generatoren, og enhetene knyttet til turbinen. For å øke driftssikkerheten, må vedlikeholdsstrategien gå fra et fokus på korrigerende vedlikehold, til forebyggende vedlikehold.



Figur 5 Driftssikkerhet (Bye, 2009)

3.3.2 Forebyggende vedlikehold;

"Vedlikehold som utføres for å vurdere og/eller minske degradering og redusere sannsynligheten for svikt i en enhet" (NS-EN 13306, 2017)

En vedlikeholdsstrategi fokusert på forebyggende vedlikehold gir mindre nedetid, høyere tilgjengelighet og bedre driftssikkerhet enn en vedlikeholdsstrategi utelukkende fokusert på korrigerende vedlikehold. Forebyggende vedlikehold hindrer stans, og man kan velge vedlikeholdsoperasjoner til faste tider, noe som gir forutsigbarhet om nedetider. Tilstandsbasert vedlikehold er en videre effektivisering av det forebyggende vedlikeholdet, ved at vedlikeholdsoperasjonene gjennomføres når de trengs, heller enn etter et gitt tidsintervall.

Tilstandsovervåkning: "aktivitet som utføres enten manuelt eller automatisk, og som måler, i forutbestemte intervaller, en enhets faktiske fysiske tilstand med hensyn til egenskaper og parametere" (NS-EN 13306, 2017)

En vedlikeholdsaktivitet hvor man ved forutbestemte intervaller måler tilstanden til en enhet. Det kan gjøres manuelt, eller automatisk. I Industri 4.0 og IoT, er det et stort fokus på automatisk tilstandsovervåkning. (Hentet fra undervisning: vedlikehold og driftsteknikk)

3.3.3 Prediktivt vedlikehold; "tilstandbasert vedlikehold som utføres etter en prognose utledet av gjentatt analyse eller kjente egenskaper og evaluering av de vesentlige parameterne for degradering av enheten" (NS-EN 13306, 2017)

Ved å bruke sensorer koblet opp til en sentral databehandlingssentral, kan man kontinuerlig overvåke tilstanden. Datagrunnlaget fra målingene, sammen med sannsynlighetsberegninger, åpner for muligheten å velge vedlikeholdstidspunkt basert på tilstanden til enheten, istedenfor faste intervaller. Dermed unngår man å gjøre vedlikehold for tidlig, når man ikke trenger det. I tillegg gir det muligheten for utføre vedlikehold tidlig, om det skulle oppstå uforutsette hendelser som kan bidra til hurtigere svikt enn tidligere anslått.

3.4 Industri 4.0 og digitalisering av industrien

Industri 4.0 er et konsept lansert av den tyske regjeringa i 2011 som et industrielt konseptsutviklingsprosjekt. Det blir sett på som en del av den Fjerde Industrielle Revolusjon. Industri 4.0 la grunnlaget for større bruk av automasjon enn i den tredje industrielle revolusjon, ved at sensorer og databehandling brukes i stort omfang både i drift og vedlikehold av industrielle anlegg. Bruk av IoT gir nye muligheter for effektivisering og optimalisering av vedlikeholdsprogram.

3.4.1 IoT

IoT betyr Internet of Things. Den digitale utviklingen har muliggjort at fysiske gjenstander kan kobles til et nettverk, hvor de kan utveksle data med hverandre og samhandle på en måte som ikke har vært mulig tidligere. Tidligere har tilstandskontroll i mindre grad vært oppkoblet internett, men heller vært gjennomført under tilstandskontrollrunder, hvor arbeidere fysisk er til stede og gjennomfører tilstandskontrollen. En IoT-tilnærming, vil da være å ha flere sensorer, koblet opp til internett, som kontinuerlig overvåker tilstanden til enheten. Mikrocomputere og sensorer blir stadig billigere, noe som åpner for bruk av fjernstyrt vedlikehold.

3.4.2 Fjernstyrt vedlikehold

"Vedlikehold av en enhet uten at personell har direkte fysisk tilgang til enheten" (NS-EN 13306, 2017).

Sensorer samler inn data, og sender dem til en sentral, hvor de analyseres, og brukes til å utforme vedlikeholdsplanen. Fjernovervåkning er nødvendig for preventivt vedlikehold for å kunne utarbeide en vedlikeholdsplan basert på sannsynlighet for svikt. Fjernovervåkning kan også muliggjøre utsetting av vedlikeholdsoperasjoner, som tidligere har vært fastsatt av tidsintervall.

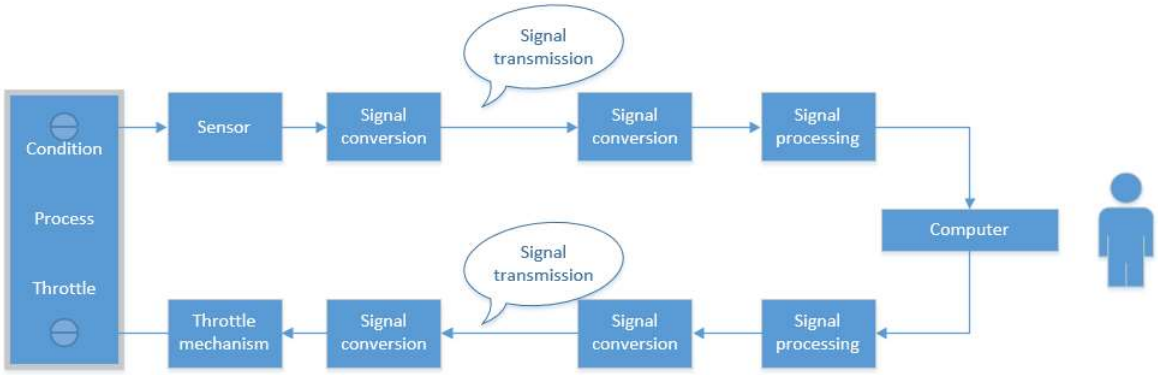
3.4.3 Tilstandsovervåkning

En vedlikeholdsaktivitet hvor man ved forutbestemte intervaller måler tilstanden til en enhet. Det kan gjøres manuelt, eller automatisk. I Industri 4.0 og IoT, er det et stort fokus på automatisk tilstandsovervåkning. (Hentet fra undervisning: vedlikehold og driftsteknikk)

3.5 Målekjede

For å kunne utvikle et godt vedlikeholdsprogram, må man ha kartlagt aktivitetene som inngår i vedlikeholdsprosessen, og kartlagt funksjonene til systemet, så en har oversikt over kritikaliteten til eventuelle svikt. Med god oversikt over funksjonene til systemet kan man identifisere aktuelle måleobjekt.

For å sette opp en pålitelig målekjede bør man først tilegnet seg teknisk forståelse av måleobjektet. Med en god teknisk forståelse av objektet, kan man velge riktige og nødvendige parameter på sensorer og tilhørende element i kjeden. Hele målekjeden må være planlagt og kontrollert, ved at de individuelle leddene i kjeden er tilpasset med tanke på nøyaktighet og pålitelighet. Mangel på kunnskap i ett ledd, kan redusere kvaliteten på målekjeden, siden det kan oppstå uforutsette problemer. Hele målekjeden må være riktig kalibrert og justert før den settes i drift. (Hentet fra undervisning: vedlikehold og driftsteknikk)

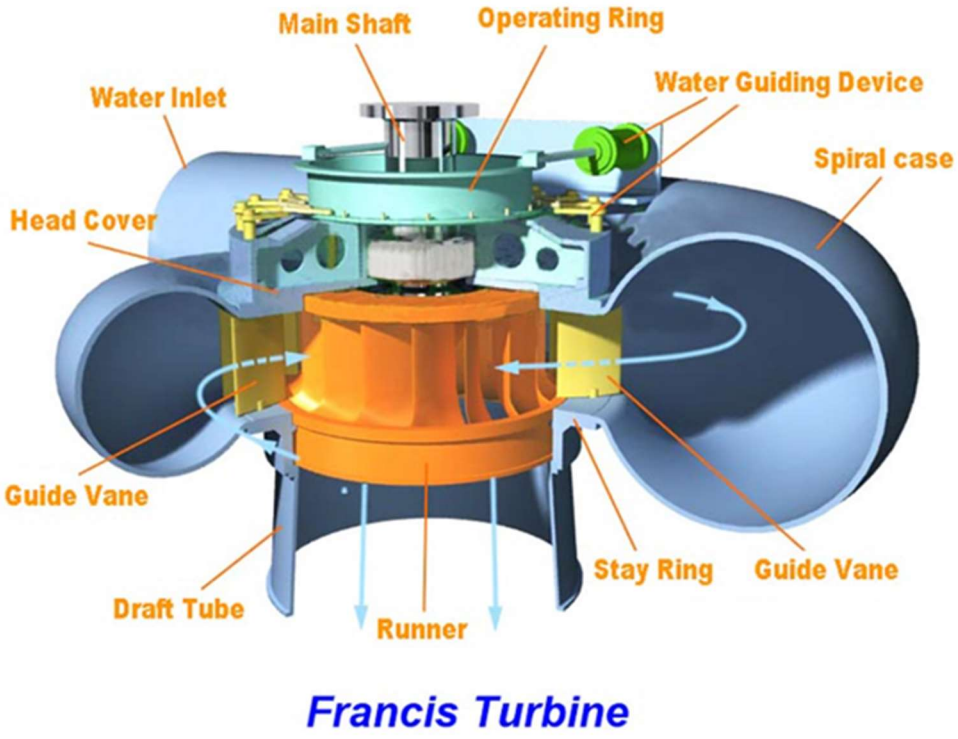


Figur 6 Målekjede (hentet fra undervisning: Vedlikehold og driftssikkerhet)

3.6 Tilstandsovervåkning Francis-turbin

En vertikal Francis-turbin med generator, utstyres vanligvis med sensorer for å overvåke tilstanden til turbinen og generatoren. I tillegg er det visuelle inspeksjonsrunder som gjøres av selskapet sine operatører.

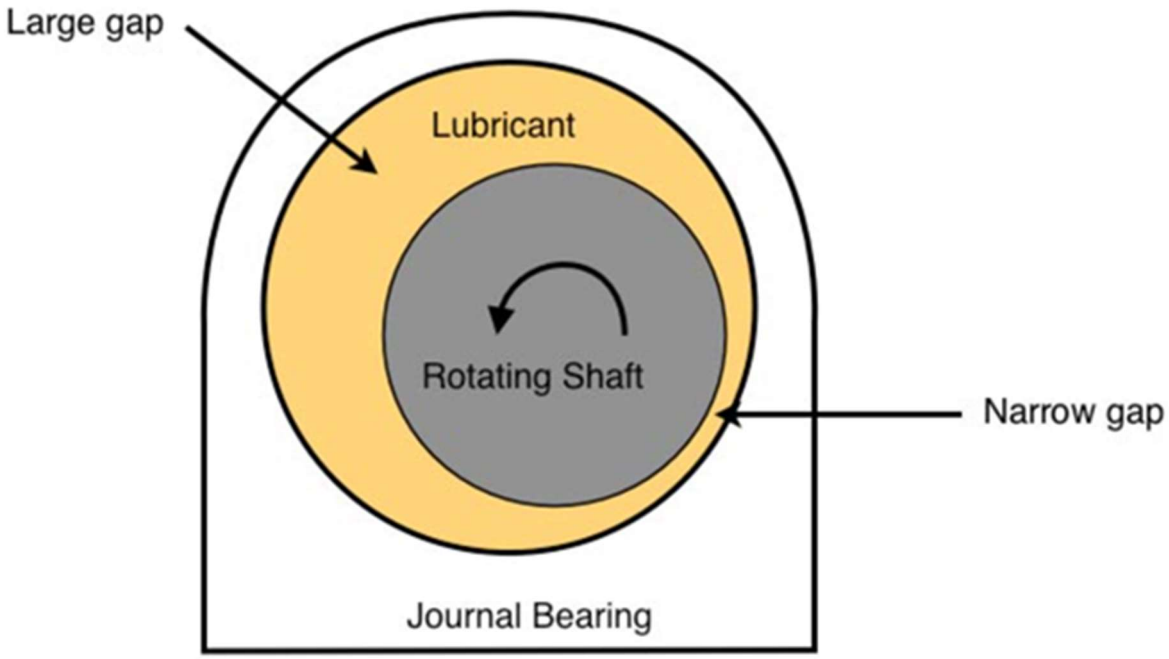
I vår oppgave, fokuseres det på turbinen, og tilstandskontroll på turbinens styrelager.



Figur 7 Francis-turbin (theconstructor, 2014)

3.6.1 Temperaturmåling

Temperaturmåling brukes i tilstandskontroll av turbinlager, hvor temperaturmålinger sammenlignes med et temperaturnivågrunnlag, utarbeidet av tidligere temperaturmålinger. Avvik fra normaltemperatur i kjølevann og lagerolje kan gi en indikator på feilutvikling, og større endringer i lagertemperatur, kan gi en indikator på at situasjonen er kritisk. (EBL, 1994)
Endringer i oljetemperatur, påvirker egenskapene til oljen. Endringer i smøreegenskapene til olja i glidelageret, kan føre til feilutvikling, siden glidelageret er avhengig av en oljefilm for å fungere korrekt. (EBL, 1994)

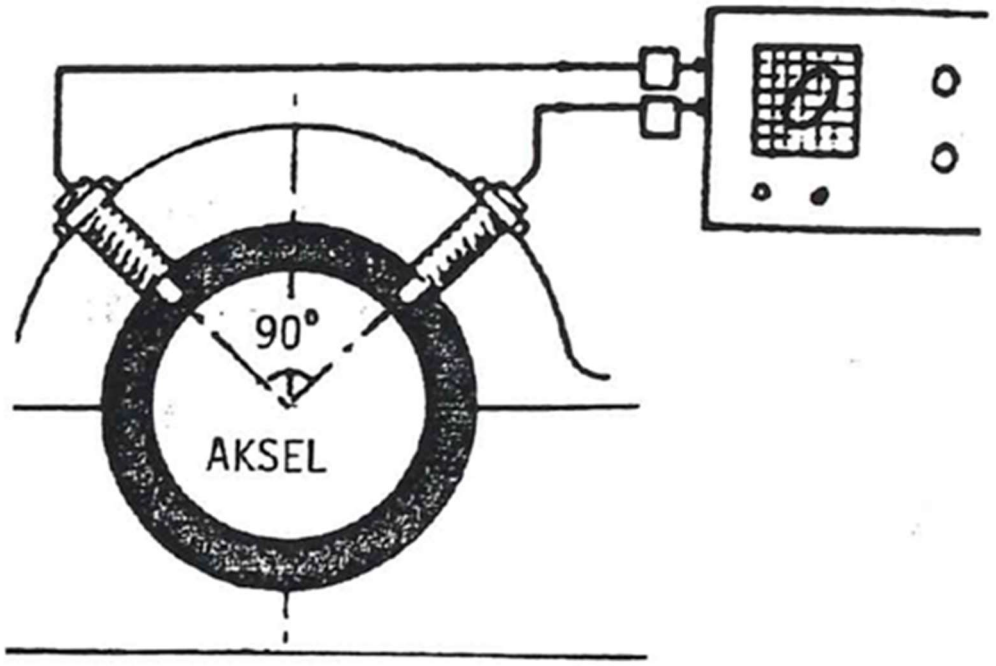


Figur 8 Glidelager (xyobalancer, 2011)

Glidelageret av en roterende aksling, et lagerhus, og en oljefilm mellom lagerhuset og akslingen.

3.6.2 Vibrasjonssmålig

På en Francis-turbin overvåkes vibrasjonsnivået vanligvis med nærhetssensorer, som fungerer som vern. Om vibrasjonsnivået overstiger et gitt nivå, slår vernfunksjonen inn, og stanser hele aggregatet. For radielle målinger, benyttes det vanligvis to nærhetssensorer montert 90 grader i forhold til hverandre. (Bye, 2009)



Figur 9 Radiell vibrasjonsmåling (Bye, 2009)

For vibrasjonsmåling på lagerhuset på Francis-turbiner, er det vanlig å benytte et akselerometer. Piezo-elektriske akselerometer er oftest brukt til vibrasjonsmåling på Francis-turbiner. MEMS-akselerometer blir stadig oftere brukt ellers i industrien, da de blir stadig billigere, og bedre egnet for områder hvor det tidligere kun ble brukt piezo-elektriske akseleremoter (Looney, 2014). Et piezo-elektrisk akselerometer består av et krystall som gir fra seg ei elektrisk ladning når det utsettes for en ytre påkjenning. Gruppen bruker et MEMS-akselerometer, som benytter endringen i kapasitans under bevegelse for å få ut et signal (ref. 5.3.3). Plasseringen til sensoren er viktig for å gjennomføre målinger. Målepunktet må monteres på massivt gods, og så nært lageret som mulig. (Bye, 2009)

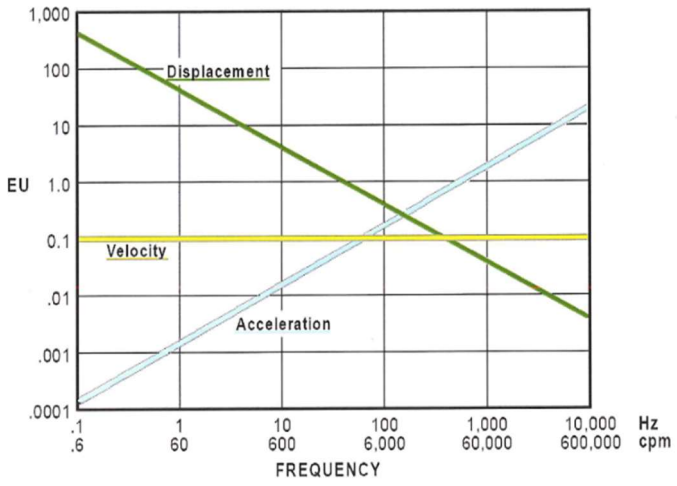
4 Vibrasjon og signalbehandling

Som vi har sett i det tidligere kapittelet er vibrasjonsovervåking på en Francis-turbin et viktig middel for å overvåke tilstanden til et vannkraftverk.
Maskineri med roterende deler vibrerer under drift. Vibrasjonene i maskineriet kan gi en pekepinne på eventuell svikt i systemet. Vibrasjonene kan også advare om maskineriet nærmer seg svikt, før det inntreffer. (Mobius Institute, 2016)

4.1 Intro vibrasjon

Vibrasjon kan defineres som hurtige, periodiske og elastiske svingninger i stive legemer (Store norske leksikon). Svingninger kan visualiseres som en masse festet til en fjær, som svinger opp og ned. Vibrasjon er da svingninger rundt et nullpunkt, hvor nullpunktet er posisjonen til massen, når det ikke er noe bevegelse. Ved måling av vibrasjoner, kartlegges bevegelsen rundt dette nullpunktet, i forhold til tid. Forflytningen av målepunktet i forhold til nullpunktet gir oss amplituden, mens antall fulle svingningssykluser per sekund gir oss frekvensen, Hz. (Mobius Institute, 2016)

Vi kan måle vibrasjonen ved enten forskyvning, hastighet eller akselerasjon.
Forskyvning måles ved å kartlegge hvor langt en masse flytter seg fra nullpunktet. Måling av forskyvning benyttes helst når vi vil måle vibrasjoner i det lave frekvensområdet, 0-100 Hz. Ved lave frekvenser kan akselerasjonen være lav, men forflytningen høy. Forskyvningen er proporsjonal med belastningen. Hastighet er den vanligste vibrasjonsenheten for vibrasjonsanalyse, og er best egnet av de tre for et bredt frekvensspekter. Det er egnet for måling av frekvenser mellom 2 og 2000 Hz, og er proporsjonal med utmattingen. Akselerasjon er mest sensitiv ved høye frekvenser, og brukes derfor som måleenhet for de høye frekvensene fra 150 Hz og oppover.

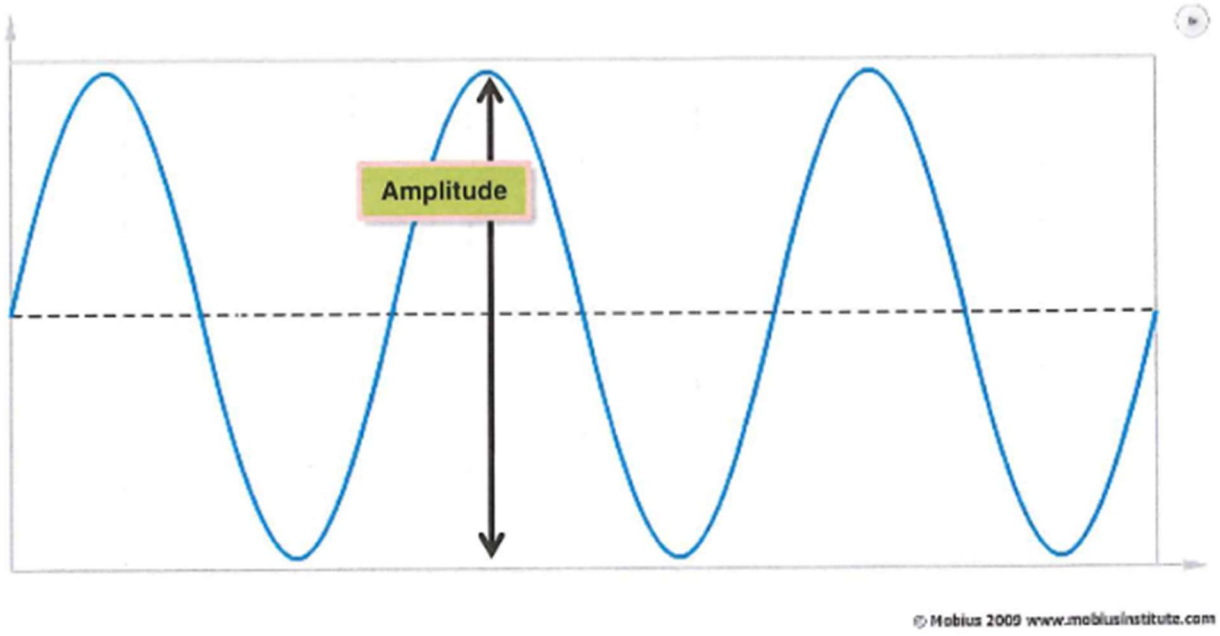


Figur 10 Forholdet mellom hastighet, akselerasjon og forskyvning (Mobius Institute, 2016)

Vibrasjon kan måles med enten nærhetssensorer, eller med akselerometer. En nærhetssensor måler avstanden til akslingen, som vil variere gjennom rotasjonssyklusen når den vibrerer.
Et akselerometer registrerer kreftene som sensorelementet utsettes for, og gir fra seg data som kan brukes til å danne et bilde av vibrasjonen. Dataene samles i tidsbildet, og kan senere gjennomgå en Fast Fourier-transformasjon (FFT), hvor tidsbildet går over til frekvensbilde, for å kunne identifisere individuelle frekvenser. Disse individuelle frekvensene, kan si oss noe om mulige feilutviklinger i individuelle element i maskineriet. (Mobius Institute, 2016)

4.1.1 Amplitude

Amplitude er avstanden fra topp- eller bunnpunktet til en bølge, til likevektstilstanden. Likevektstilstanden, også kalt normaltilstanden, er linjen bølgen beveger seg om. Når en overflate vibrerer, representerer amplituden avstanden overflaten går fra normaltilstanden. Man kan tenke seg ei vekt på ei fjær, hvor maksimalavstanden vekten går fra sin normaltilstand, er amplituden.



Figur 11 Amplitude (Mobius Institute, 2016)

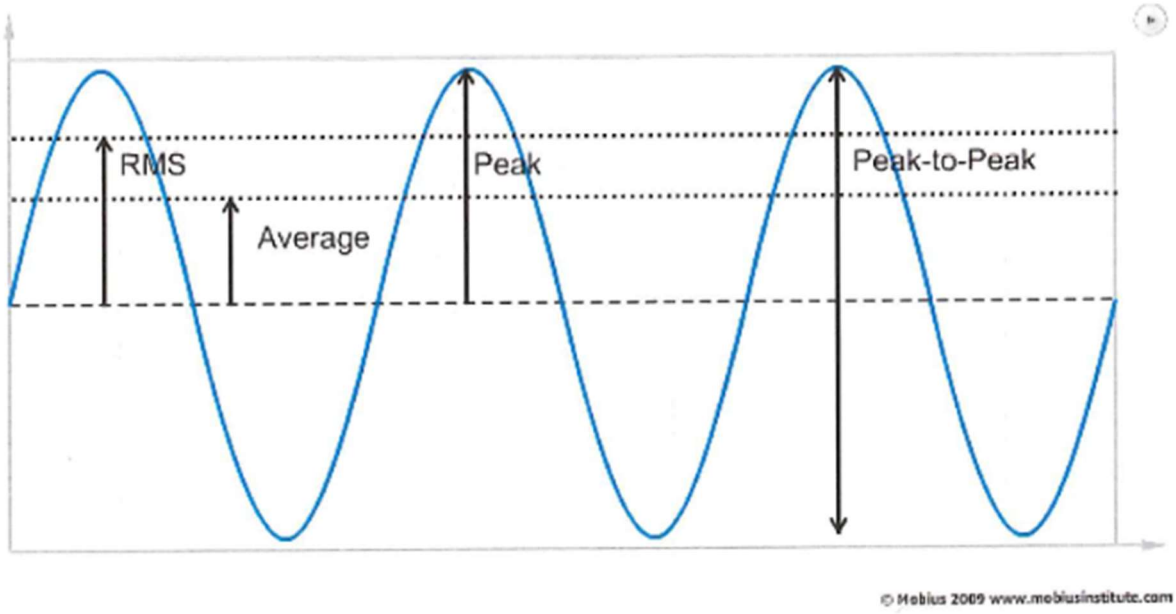
For å beskrive vibrasjonsamplitude nærmere, bruker vi følgende uttrykk;

- Topp til topp-amplitude
- Topp-amplitude
- Root Mean Square (RMS)

Topp til topp-amplitude er avstanden mellom det høyeste og laveste utslaget til bølgen. Topp-amplituden er den største avstanden fra normaltilstanden til det største utslaget til bølgen.

Root Mean Square er et uttrykk for den gjennomsnittlige energien i vibrasjonen. Når vi ser på amplitude, kan vi være mer interessert i den gjennomsnittlige energien, enn individuelle topp- og bunnpunkt. RMS er nyttig, da den gir et bilde på hvor mye energi som er i vibrasjonen.
For en sinusbølge har vi følgende uttrykk for RMS:

$$RMS = \frac{Topp}{\sqrt{2}} = 0{,}707 \times Topp$$



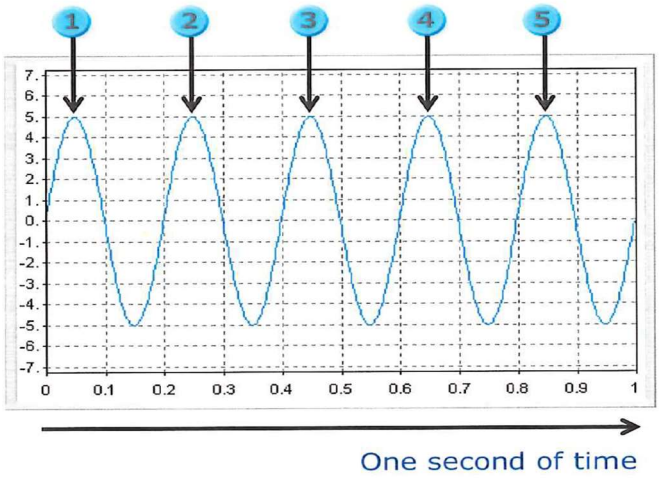
Figur 12 Måleenheter (Mobius Institute, 2016)

Dette gjelder kun for symmetriske sinusbølger. Reell vibrasjon i maskineri har sjeldent formen til symmetriske sinusbølger, siden vibrasjonen påvirkes av mange deler som påvirker hverandre, støy og løse deler. (Mobius Institute, 2016)

4.1.2 Periode og frekvens

Mens amplitude beskriver den maksimale bevegelsen til bølgen, sier den ikke noe om hvor ofte bevegelsen hender. For å beskrive det, trenger vi frekvens.

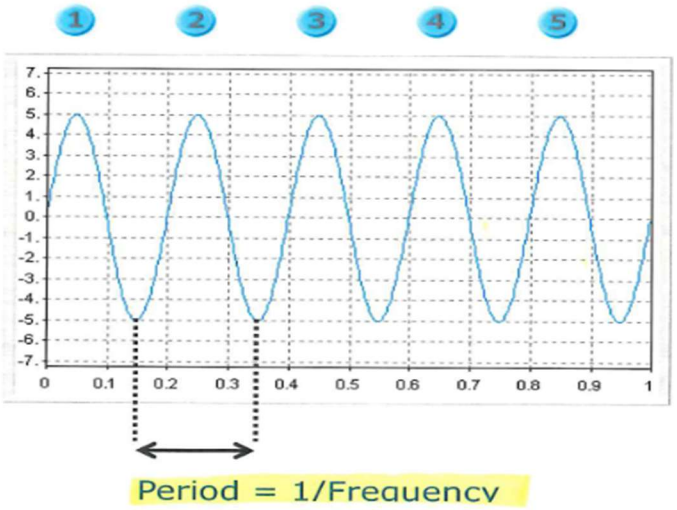
Frekvens beskriver hvor ofte en hendelse skjer innenfor et gitt tidsrom. I vibrasjonsanalyse ser vi etter antall bølgesykluser i sekundet, som beskrives som Hertz.



Figur 13 Periode og frekvens (Mobius Institute, 2016)

Her har vi fem fullstendige sykluser på et sekund. Frekvensen er altså 5 Hz.

Periode er tiden det tar for en bølgeform å gjennomføre en hel syklus. Fra nullpunkt, til topp, til nullpunkt, til bunnpunkt og tilbake til nullpunkt er da en periode. Topp til neste topp, eller bunn til neste bunn er også en periode. Når frekvensen øker, synker perioden.

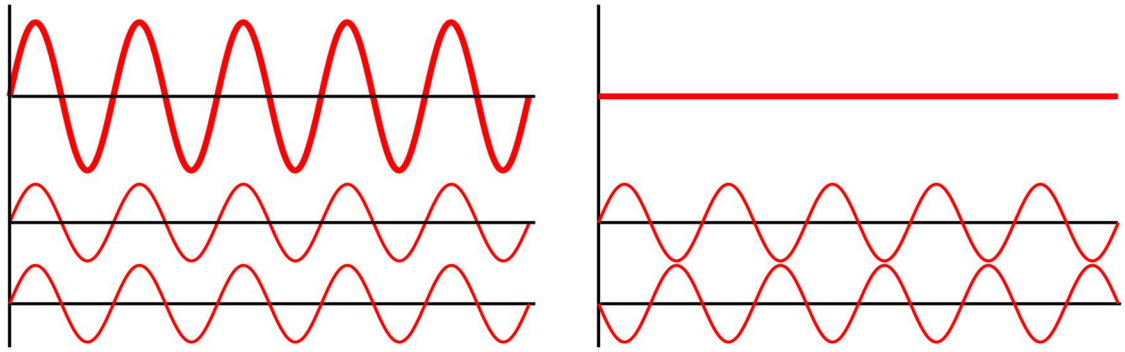


Figur 14 Periode og frekvens (Mobius Institute, 2016)

Frekvensen er fortsatt 5 Hz, og perioden blir 0,2 sekund. (Mobius Institute, 2016)

4.1.3 Bølgefase

En fase er en posisjon på en bølgeformsyklus i et gitt tidspunkt. Når man sammenligner to bølgeformer, kan faseforskjeller uttrykkes ved en vinkel mellom -180 og 180.
Ved sinusbølger med forskjellig frekvens, men samme amplitude, vil forsterke og kansellere hverandre, ettersom de går inn og ut av fase. (NIST, 2019)

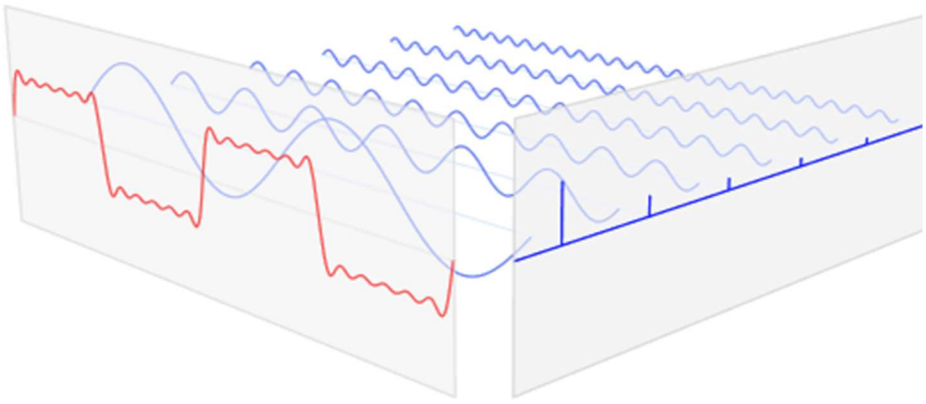


Figur 15 Bølgefase (Wikipedia, Wave interference)

Over har vi to sett med bølger med samme amplitude og samme frekvens. Til venstre er de i fase, og forsterker hverandre. Til høyre er den ene faseforskjøvet 180 grader i forhold til den andre, og de kansellerer hverandre.

4.1.4 Tidsbilde og frekvensbilde

For å kunne beskrive vibrasjon, må vi skille mellom tidsbilde og frekvensbilde. I et tidsbilde, er tiden den ene variabelen. Men det forteller oss lite om de individuelle frekvensene som inngår i et vibrasjonsmønster. For å kunne lettere se det, trenger vi å omforme til et frekvensbilde, hvor tiden er bytta ut med frekvens som variabel.



Figur 16 Tidsbilde (Rød graf) og frekvensbilde (blå graf), (Wikipedia, Frequency domain)

Ved å identifisere de individuelle frekvensene, kan vi knytte dem sammen med element i maskineriet. Ved å foreta FFT, vil det være mulig å få en indikasjon på hvor feilutviklingen opptrer. (Mobius Institute, 2016)

4.2 Fire huskeregler for vibrasjon

For å forstå hva vibrasjonsmønsteret betyr for maskineri, er det fire sentrale tema som kan fortelle oss noe om det;

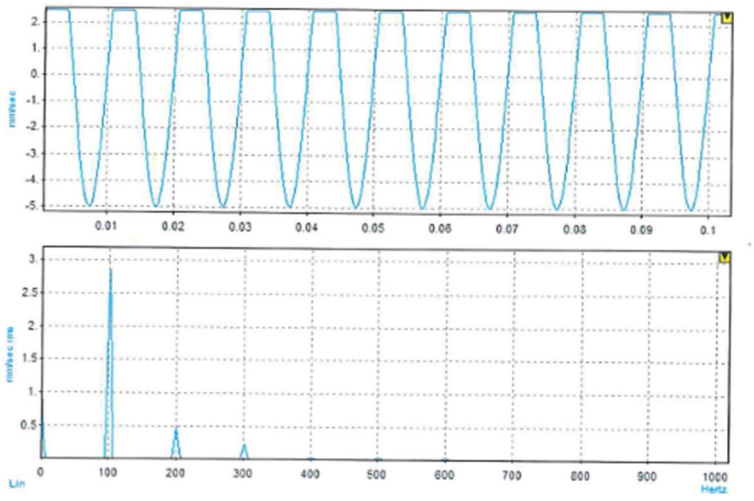
- Sinusbølger
- Harmoniske svingninger
- Støy
- Amplitudemodulasjon og sidebånd

4.2.1 Sinusbølger

Rene enkelte sinusbølger lager en enkelt topp i frekvensbildet. Når flere vibrasjonskilder legges til, vil frekvensbildet endre seg, basert på de forskjellige amplitudene og frekvensene til vibrasjonskildene. Om to sinusbølgeformer med forskjellig amplitude og forskjellig frekvens interagerer, vil den kombinerte amplituden ikke treffe null. Med samme amplitude, vil de gå inn og ut av fase med hverandre, og ha punkter med en amplitude på null. Med samme amplitude og samme frekvens, avhenger det av faseforskjellen om vi får forsterking, utligning, eller noe imellom. (Mobius Institute, 2016)

4.2.2 Harmoniske svingninger:

En repeterende bølgeform som ikke er en ren sinusbølge, skaper harmoniske svingninger. Harmoniske svingninger er gjentagende og avtagende topper i frekvensbildet med lik avstand mellom toppene. Den første toppen er fundamentalfrekvensen, mens de påfølgende toppene er de neste leddene i den harmoniske serien. Bølgeformer som skaper harmoniske svingninger kommer ofte fra løst maskineri, feilstilling eller feil med hjullager.



Figur 17 Harmoniske svingninger som følge av kappede bølger (Mobius Institute, 2016)

Kappede bølger oppstår når en ren sinusbevegelse blir fysisk begrenset. For eksempel om vibrasjonen har frihet til å bevege seg i en retning, men blir stoppet før den får fullført bevegelsen i den andre retningen. Jo mer kappet bølgen er, jo sterkere blir den harmoniske amplituden.

Repeterende slag: når det er ekstrem løshet i ei maskin, vil det kunne oppstå repeterende slag, som skaper harmoniske svingninger. Ekstreme tilfeller kan føre til under-harmoniske svingninger av ½, eller ⅓ -orden.

Selv om topper observeres rundt de forskjellige harmoniske ordenene, er det ikke sikkert at det er flere forskjellige kilder som står for toppene. Det trenger heller ikke være faktiske kilder til vibrasjon. En kilde kan være forvrenging gjennom Fast Fourier-transformasjonen som skaper ikke-reelle harmoniske svingninger i frekvensspektrumet. (Mobius Institute, 2016)

4.2.3 Støy

Støy skapes av enten urelaterte hendelser, som slag, eller tilfeldig vibrasjon som ikke er repeterende. Støy kan indikere at det er en feil et sted i maskinen, men det er vanskelig å bruke vibrasjonsspektrumet for å påvise feil, om feilene fører til støy. For eksempel ved kavitasjon, vil kavitasjonsboblene generere tilfeldige vibrasjoner, som ikke er repeterende, og dermed vanskelig å påvise i et vibrasjonsspektrum. (Mobius Institute, 2016)

4.2.4 Amplitudemodulasjon og sidebånd

Amplitudemodulasjon er når en frekvens blir modulert av en annen frekvens, så amplituden til signalet økes og senkes repeterende. Frekvensen som periodisk varierer, kalles bærefrekvensen. Frekvensen som modulerer bærefrekvensen kalles modulasjonsfrekvensen. Rundt bærefrekvensen, med en avstand tilsvarende modulasjonsfrekvensen, vil det oppstå sidebånd. Dette oppstår som parvise topper rundt bærefrekvensen. (Mobius Institute, 2016)

4.3 Signalbehandling:

Et akselerometer tar et analogt signal, og sender det videre til Analog Digital converter-en, som gjør det om til et digitalt signal for videre databehandling. Sensoren er en transduser som mottar et pådrag fra et fysisk system, og produserer et signal basert på pådraget.

Når vibrasjonssignaler behandles, er det flere aspekter som er viktige:

- Filtrering av signal
- Integrasjon
- Windowing
- Averaging, redusere støy

4.3.1 Filtrering av signal

For vibrasjonsanalyse, er det tre typer filtrering som står sentralt;

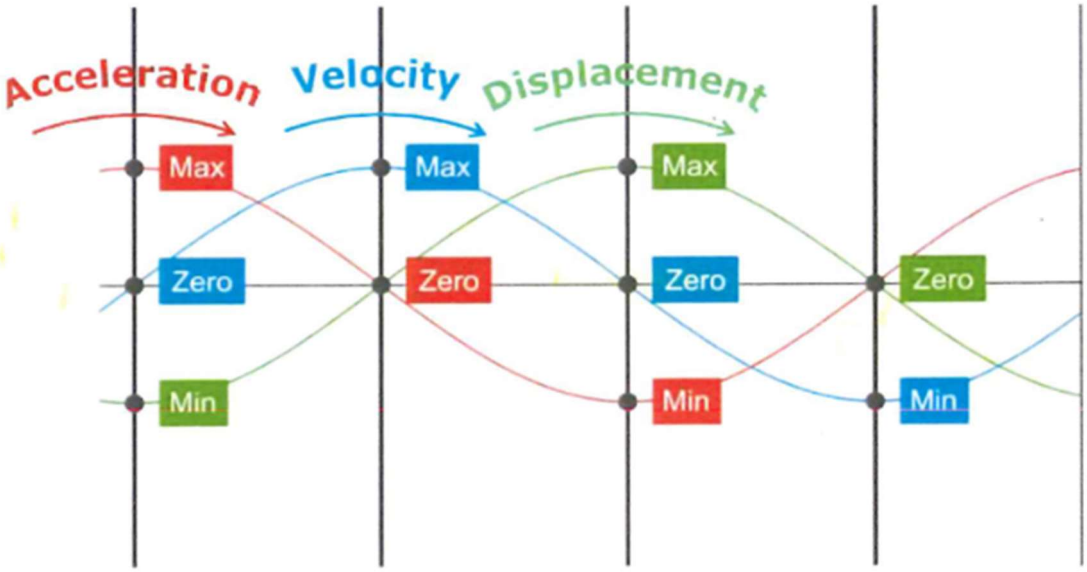
- Lavpassfilter
- Båndpassfilter
- Høypassfilter

Hvor man velger ut et frekvensintervall som er relevant for det en ønsker å måle.

Lavpassfilter filtrerer ut de høye frekvensene, og lar de lave frekvensene slippe gjennom. Dette brukes ofte for å fjerne aliasing.
Båndpassfilter filtrerer ut høye og lave frekvenser, og lar frekvensene i midten være igjen. Båndpassfilter brukes når man vil fange opp en spesifikk frekvens, men med minst mulig støy.
Høypassfilter filtrerer ut de lavere frekvensene. Brukes ofte for å detektere utmatting i hjullager. (Mobius Institute, 2016)

4.3.2 Integrering av signal

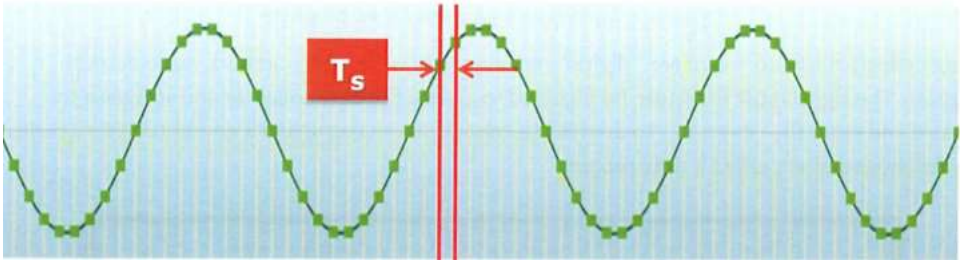
Når dataene er samlet inn ved bruk av et akselerometer, vil vi omforme det til hastighet eller forflytning. Fra akselerasjon til hastighet, får vi en faseforskyvning på 90 grader. Så når akselerasjonen krysser sin likevektslinje, vil hastigheten nå sin topp. Videre får vi en faseforskyvning på 90 grader når vi integrerer fra hastighet til forflytning. Forflytningen når sine topp- og bunnpunkter, når hastigheten krysser sin likevektslinje.



Figur 18 Enheter og fase (Mobius Institute, 2016)

Sampling, eller punktprøving er måling av punkter. Ved vibrasjonsanalyse, er det mange punktmålinger i sekundet. Antall målinger over et gitt tidsintervall kalles samplefrekvensen (F_S), som er antall målinger per sekund. Sampleperioden (T_S) og samplefrekvensen forholder seg til hverandre som perioden og frekvensen forholder seg til hverandre.

$$T_S = \frac{1}{F_S} \quad F_S = \frac{1}{T_S}$$

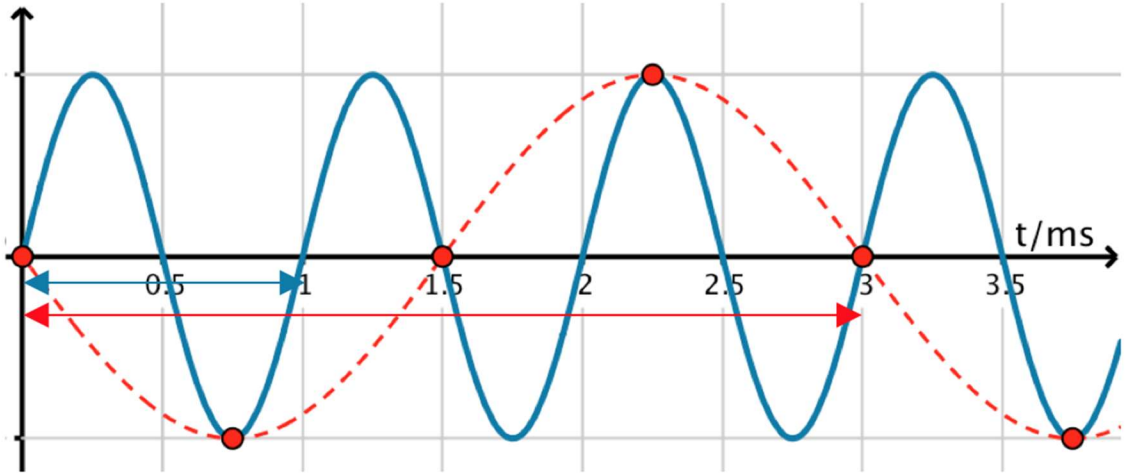


Figur 19 Sample periode (Mobius Institute, 2016)

Siden målingene kun måler punkter, er det viktig å få med mange nok målinger til å kunne korrekt beskrive kurven til vibrasjonen.

Om sampling-frekvensen er for lav i forhold til frekvensen til det vi ønsker å måle, ender vi opp med et feilaktig bilde av bølgeformen. For eksempel om sampling-frekvensen er den samme som frekvensen på det vi ønsker å måle, vil vi ende opp med et feilaktig bilde av en ikke-eksisterende bølge. Det feilaktige bildet kalles aliasing, eller foldingsfeil.

Figur 20 viser et eksempel på aliasing. Det reelle signalet er representert i blått. De røde punktene viser samplepunktene, mens den røde stiplete linjen viser signalet som bli oppfattet av systemet. (Mobius Institute, 2016)



Figur 20 Aliasing (Wikipedia, aliasing)

4.3.2.1 Nyquist-teoremet:

For å unngå aliasing, må samplefrekvensen (F_S) være høyere enn frekvensen på det vi ønsker å hente inn (F_{MAX}). Nyquist-teoremet konstaterer at sampling-frekvensen må være mer enn det dobbelte av frekvensen til bølgeformen vi ønsker å måle;

$$F_S > 2 \times F_{MAX}$$

I realiteten brukes minimum 2,56 ganger den maksimale frekvensen til det vi skal måle, når vi bruker digitale sensorer;

$$F_S > 2{,}56 \times F_{MAX}$$

4.3.2.2 Fast Fourier-transformasjon

En fast Fourier-transformasjon omgjør dataene fra tidsbildet, til et frekvensbilde. N antall verdier fra tidsbølgeformen omgjøres til $\frac{N}{2{,}56}$ antall verdier for frekvens, amplitude og fase.

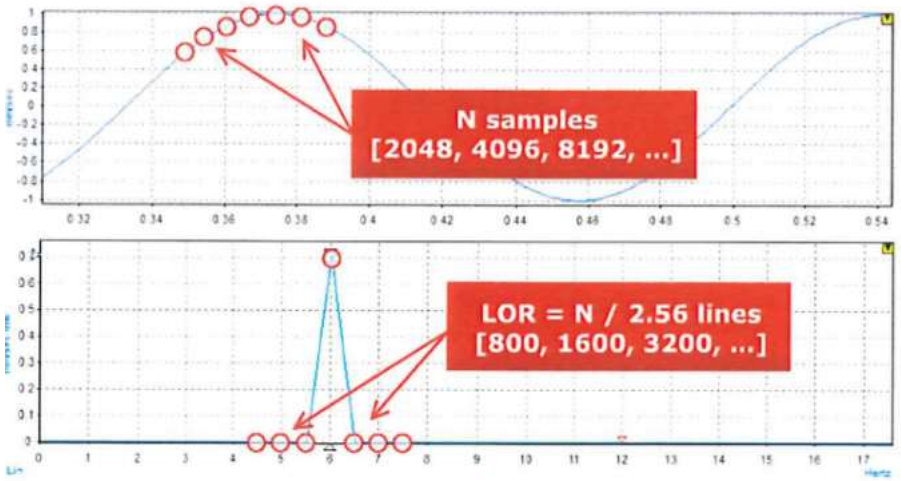
N er antall målepunkter. Så for å gjenskape et signal på 500 Hz, trenger vi $500\ Hz \times 2{,}56 = 1280\ Hz$ samplingrate. (Mobius Institute, 2016)

4.3.2.3 Oppløsning

Antall punktprøver i tidsbølgeformen avgjør hvor mange verdier vi får i spektrumet. Så for 1024 antall punkter, vil vi få ut $\frac{1024}{2{,}56} = 400$. Dette kalles LOR (Lines Of Resolution). Oppløsninga i et frekvensspekter vil kunne kalkuleres ut fra LOR og det maksimale frekvensområdet (F_{MAX}).

$$Oppløsning = \frac{F_{MAX}}{LOR}$$

Å skille forskjellige frekvenser fra hverandre er avhengig av oppløsningen. Så for en F_{MAX} på 1000 Hz, og med 400 LOR, vil vi få en oppløsning på $\frac{1000\ Hz}{400\ LOR} = 2{,}5\ Hz$. Da vil vi ikke kunne skille mellom forskjellige diskrete frekvenser som er mindre enn 2,5 Hz ifra hverandre. Dette kan løses ved å øke antall LOR. (Mobius Institute, 2016)



Figur 21 Tidsbilde og frekvensbilde N og LOR (Mobius Institute)

4.3.2.4 Samplingtid

En høyere oppløsning vil bedre skille mellom frekvenser som er nære hverandre. Men det vil føre til en lengre samplingtid (T). Tiden som kreves for å gjøre målingen, tilsvarer antall nødvendige punktmålinger delt på samplingraten. Vi kan også bruke LOR/F_{MAX}. Høyere oppløsning fører til flere LOR, som igjen fører til lengre samplingtid.

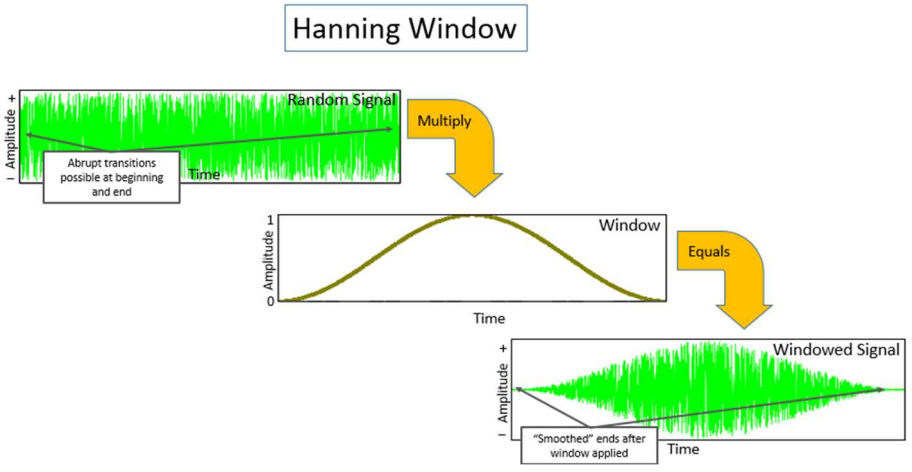
$$T = \frac{LOR}{F_{MAX}}$$

4.3.3 Windowing:

FFT avhenger av at vi deler opp dataene i blokker, hvor signalet starter og slutter rundt nullamplituden. Hvis det ikke deles opp hvor start- og sluttpunktet ender rundt nullamplituden, vil det se ut som et støt etter FFT. For å unngå dette, omgjør man amplituden til endene til det originale signalet til null, for så å gjennomføre FFT. Det finnes flere window-funksjoner for å gjøre dette, og vi bruker Hann-funksjonen. Window-funksjoner forvrenger signal-data ved at både amplituden og energien til et signal blir redusert. (Mobius Institute, 2016)

4.3.3.1 Hanning window

Hann-funksjonen er vanligvis brukt ved vibrasjonsmåling av roterende maskineri, hvor det er ventet støy. Hann-funksjonen ganges med signalet før en FFT for å minimere spekterlekkasjen. Hann-funksjonen endrer amplituden og energien i signalet. Der er derfor nødvendig å korrigere amplituden etter en FFT. Faktor for amplitudekorreksjon er 2, og for energikorreksjon 1,63. Det vil si at for å få det korrekte amplitudenivået i en FFT der det er benyttet en Hann-funksjon, må man multiplisere amplituden med 2. For å få korrekt energinivå, må man multiplisere med 1,63. Det er dermed mulig å korrigere det windowede signalet for amplitude eller energi, så det er likt original signalet. Men ved å benytte Hanning window kan man få opptil 15% amplitudefeil. (Siemens, 2017)



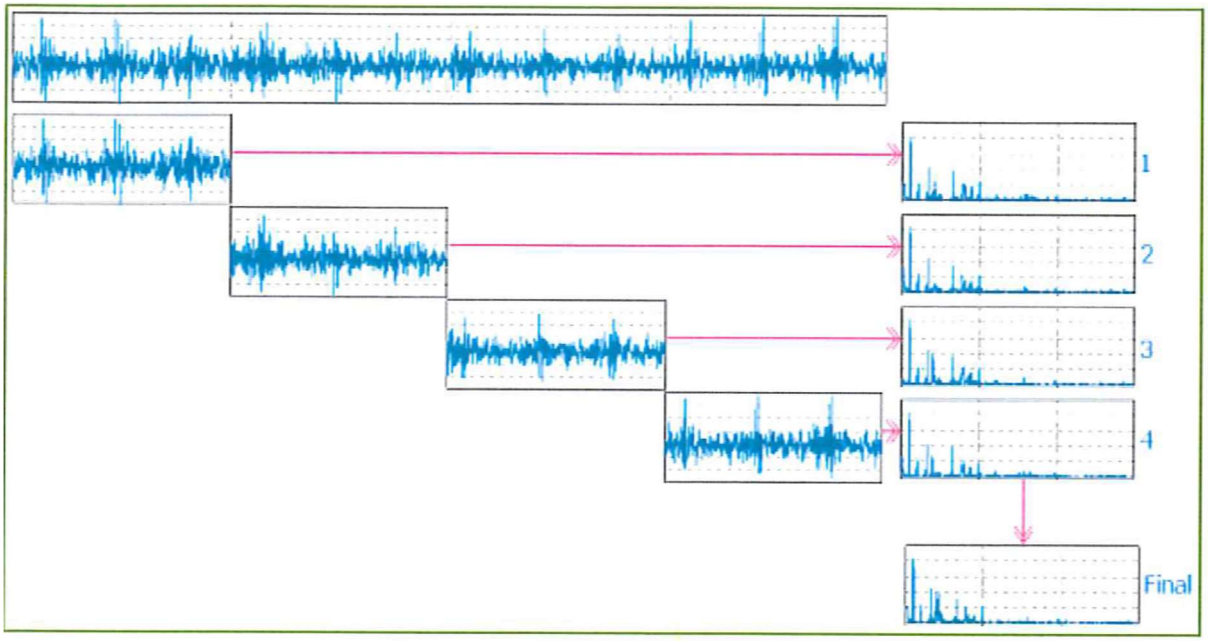
Figur 22 Hanning Window (Siemens, 2017)

4.3.4 Redusering av støy

Også kalt averaging. Siden det alltid vil være støy, og tilfeldige vibrasjoner, er vi avhengig av å ikke stole for mye på en enkelt måling for å kunne si noe om frekvensbildet. En metode å redusere støy på er å dele opp målingen i mange blokker, og benytte gjennomsnittet av blokkene. Det utføres en FFT på hver blokk. Så benyttes gjennomsnittet av FFT-ene for å visualisere et frekvensbilde. Vi har flere metoder å velge ut et gjennomsnitt på, Lineær, RMS, Overlap, negative og Peak hold. Vi har benyttet Lineær averaging, siden den er enklest å gjennomføre i Python. (Mobius Institute, 2016)

4.3.4.1 Lineær averaging

Signalet deles opp i blokker med lik lengde. Summen av blokktidene tilsvarer tiden av det originale signalet. Gjennomsnittet av blokkene brukes som det visualiserte bildet, som vist i figur 23.



Figur 23 Lineær averaging (Mobius Institute)

5. Resultatmål 1: Ståstedsanalyse

Dette kapitelet vil presentere resultatmål 1: Utarbeid en ståstedsanalyse som inneholder:

- Identifisere målepunkter på TEK sin turbin ved Brattset for måling av temperatur, trykk og vibrasjon med Raspberry Pi – identifiser måleområde for disse punktene.
- Identifiser Raspberry Pi sine kapasiteter og begrensninger.
- Identifiser sensorer egnet for måling i de identifiserte målepunktene. Sensorene må være kompatible med Raspberry Pi.

Ståstedsanalysen vil legge grunnlag for hvilken av de 3 måleparameterene trykk, temp og vibrasjon som skal videreføres i oppgaven.

5.1 Identifikasjon av målepunkter på TEK sin turbin ved Brattset for måling av temperatur, trykk og vibrasjon

For å identifisere egnede målepunkt ved Brattset sin vertikale Francis-turbin har man valgt en stegvis tilnærming.

1. Skaffe drift-teknisk informasjon om vertikale Francis-turbiner.
2. Befaring av vertikale Francis-turbinen på Brattset.
3. Benytte standard til å identifisere målområdet.

5.1.1. EBL KOMPETANSE: Håndboken om Francis-turbin

På 1990 tallet var det et generasjonsskifte i norsk vannkraftindustri. Verdifull kompetanse forsvant med folkene som sluttet eller ble pensjonister. EBL (Energibedriftens landsforening) bestemte seg derfor å sette i gang et prosjekt for å bevare den oppbygde erfaringen i egne håndbøker. I dag går EBL under navnet Energi Norge. Disse håndbøkene oppdateres kontinuerlig og gir en detaljert beskrivelse innenfor komponenter, skadetyper og målemetoder av vannkraftverk. Håndbøkene er delt opp etter hvordan vannkraftverket er montert.

På Brattset kraftverk er det montert 2 vertikale Francis-turbiner. Francis-turbiner monteres enten vertikalt eller horisontalt. Fallhøyde, vannføring og valg av stasjonsarrangement er faktorer som bestemmer hvilken akseretning som velges. Tabellen nedenfor viser enkelte komponenter hvor *håndboken om francisturbin* anbefaler temperatur, trykk og vibrasjonsmåling. (EBL, 1994)

Komponent	**Skadetype**	**Konsekvens**	**Prøvemetode for påvisning**
Løpehjulet	Sprekker	Skovelbrudd	Vibrasjonsmåling
Turbinlager	Slitasje på lagerflatene Løst lagermetall	Økt akselklaring Økt lagertemperatur	Vibrasjonsmåling Temperaturmåling av olje
Ledeapparat	Overbelastet reguleringsmekanisme	Brudd i bruddlenker Stor innbyrdes ledeskovlklaring	Trykkmåling på servomotor
Sugerøret	Utmatting Plater spekker	Redusert virkningsgrad	Trykkmåling

Tabell 5 Komponenter med mulig skadetype fra EBL

5.1.2 Besøk på Brattset og intervju med samarbeidspartner

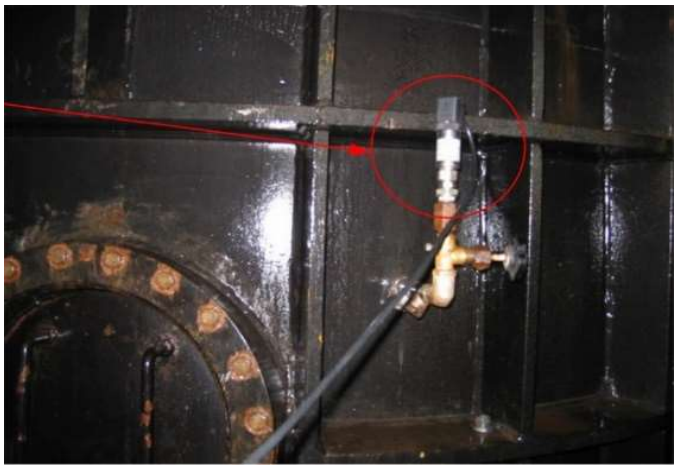
01.02.2019 gjennomførte gruppen et besøk på driftssentralen til TEK på Berkåk og Brattset kraftverk. Under besøket identifiserte gruppen mulige målepunkt på turbinen ved visuell inspeksjon og intervju med samarbeidspartner.

Spørsmål	**Svar**
Har dere noe overvåking av temperatur, trykk eller vibrasjon på turbinen på nåværende tidspunkt på Brattset og hvor isåfall?	Har overvåkning på turbinlagrene i Brattset fra før. -Oljenivå. -Oljetemperatur. -Lagersegmenttemperatur. -Vibrasjonsvern.
Hvilken temperatur ligger oljen på?	Har hatt litt høye verdier, og ligger nå på rundt 65C.

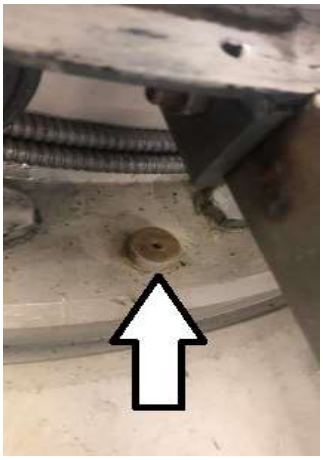
Tabell 6 Intervju med Oddvar Bjerkås

Resultat fra visuell inspeksjon av turbinen:

- Ingen uttak på sugerørkonus for trykkmåling. (Figur 24 viser hvordan et uttak på sugerørkonusen ville sett ut)
- Punkt for å skru fast vibrasjonssensor på flensen til turbinlager.

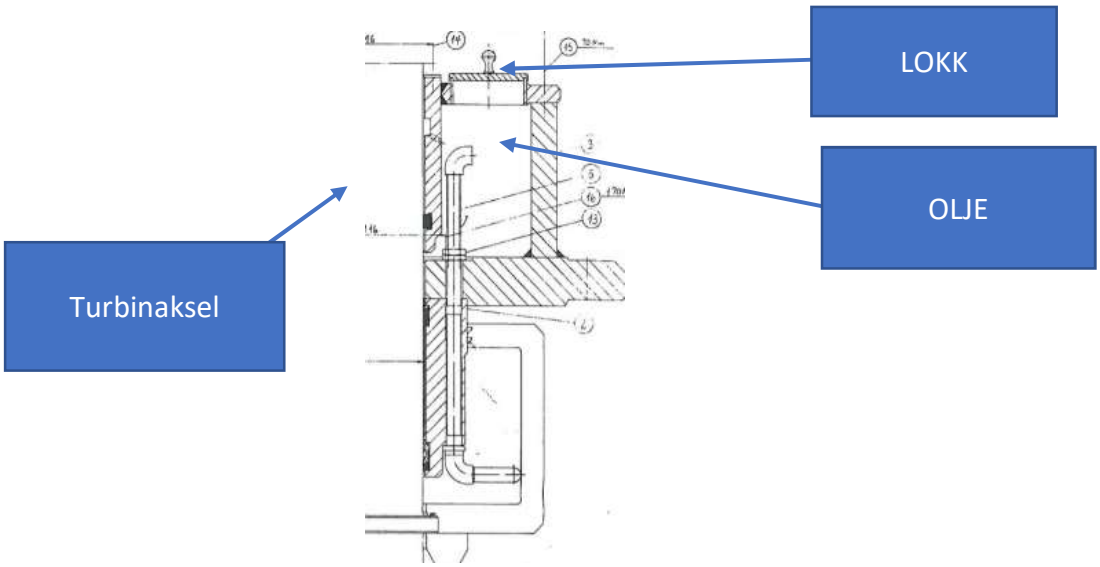


Figur 24 Utakk for trykkmåling på sugerørkonus (Bjørndal, 2007)



Figur 25 Plassering til vibrasjonssensor på turbinlagret ved Brattset

- Enkel tilgang til temperaturmåling av oljen i turbinlager. Ved å åpne lokket, kan man enkelt slippe temperatursensoren ned i oljen.



Figur 26 Maskintegning fra Brattset med tilgang på temperaturmåling

5.1.3 Identifikasjon og konklusjon på valg av målepunkter.

Konklusjonen fra besøket på Brattset viser at følgende målepunkt er tilgjengelige for temperatur og vibrasjon:

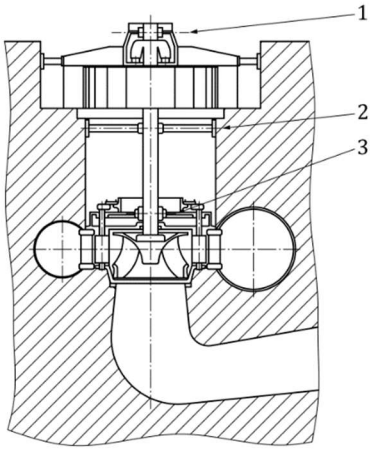
- Temperaturmåling av oljen i turbinlagret.
  - Forventet målområde rundt 65 °C
- Vibrasjonsmåling av lagerhusvibrasjoner hvor sensoren monteres på flensen til turbinlagret.

Trykkmåling vil ikke inngå i videre analyse. Hovedgrunnen til det er mangel på uttak for trykkmåling på sugerørkonus. I tillegg begrenses systemet til to sensorer på grunn av tilgjengelig tid.

5.1.4 ISO 20816 - 5

ISO 20816-5 er utviklet for vertikale og horisontale maskiner med turbiner av typen Francis, Pelton, Kaplan, Pumpe og Rør. Den retter seg inn mot anlegg med en typisk rotasjonshastighet mellom 60 rpm og 1000 rpm. (Brattset ligger på 600 rpm). Standarden gir retningslinjer for akseptabel vibrasjon på lageret, bærelageret og lagerhuset i hydrauliske kraft og pumpeanlegg når de opererer innenfor normale driftsforhold. Standarden beskriver også hvilke målepunkt og retningen som anbefales brukes for vibrasjonsmålingen.

Hvilke målepunkt standarden anbefaler bestemmes fra anleggets utforming. Anlegget på Brattset er utformet hvor alle lagerhusene er støttet opp mot stasjonsfundamentet. Teknisk tegning av anlegget på Brattset finnes i vedlegg 1. Siden anlegget er utformet slik, benytter man figur 3 gruppe 3 S5 i standarden. Den indikerer 3 punkt for plassering av vibrasjonsmålingen. Dette støtter opp valget om å plassere vibrasjonssensoren på flensen til turbinlageret (Punkt 3 på figur (27)).



Figur 27 ISO 20816 figur 3 gruppe 3

Ved å benytte et målepunkt som standarden anbefaler, kan man se hvilken vibrasjon i turbinlageret som er akseptabel.

Målområdet til vibrasjonssensoren bestemmes ut fra aksjonsgrensene til standarden. Aksjonsgrensen blir dypere beskrevet i kap. 5.3.2 Signalamplitude.

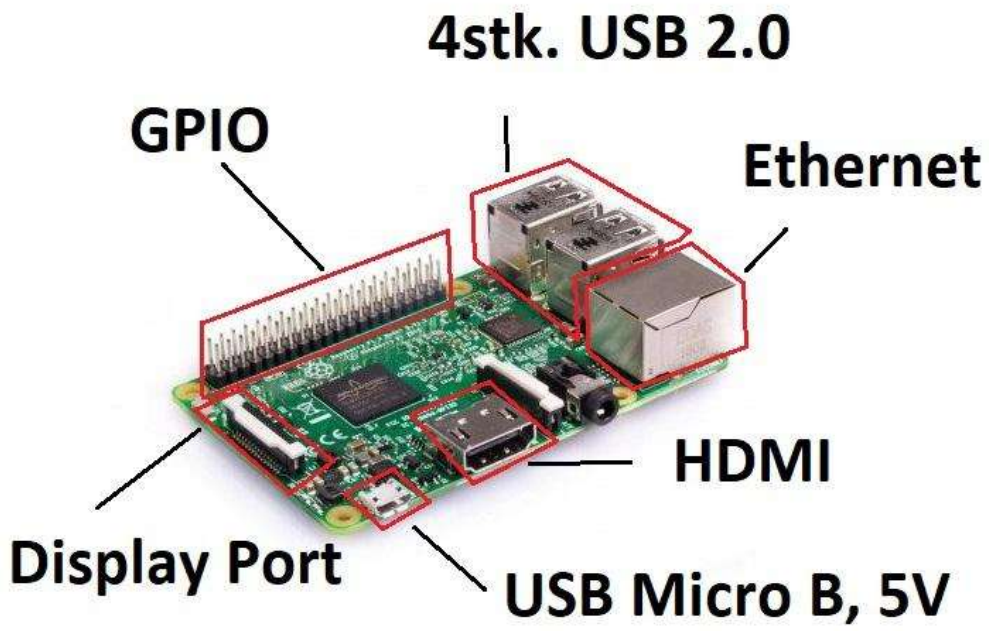
Standarden anbefaler også retningen (aksiell og radiell) vibrasjonssensorer bør ha på målepunktet. Hvordan dette er håndtert er, blir drøftet i kap. 5.3.3.

5.2 Identifisere Raspberry Pi sine kapasiteter og begrensinger

Raspberry Pi er en liten og kompakt ett-korts datamaskin på størrelse med et kredittkort. Den ble utviklet som en læringsplattform for bruk i data og elektronikk relaterte prosjekter. Raspberry Pi har derfor et stort open source utviklersamfunn, som gjør at det er stor tilgang på informasjon om hvordan forskjellige programmering- og elektronikk-problemstillinger kan løses (What Is a Raspberry Pi, 2015). I tillegg har Raspberry Pi en lav kostnad på 399.- og benytter et open source operativsystem basert på Linux (What Is a Raspberry Pi, 2015). Dette gjør at Raspberry Pi kan i utgangspunktet være en god plattform for utvikling av en prototype som logger tilstandsparameter som vibrasjon og temperatur. Vedlegg 2 viser Raspberry Pi sitt datablad.

5.2.1 Tilkoblingsmuligheter

Raspberry Pi er meget brukervennlig fordi den har grunnleggende tilkoblingsmuligheter som en ordiner laptop. Man kan enkelt koble til keyboard og mus via 4 stykk USB 2.0-kontakter. Skjerm kan kobles til via en HDMI-kontakt eller man kan benytte en «plug and play» LCD skjerm som kobles til displayporten (Richardson & Wallace, 2016).
Det er også mulig å kjøre Raspberry Pi «headless». Det vil si å bruke pi-en uten fysisk tilkoblet skjerm. Å kjøre Raspberry Pi-en headless kan gjøres ved å benytte en VNC (virtual network computing) server. Det krever at Raspberry Pi-en er tilkoblet internett. Det er både en ethernettport og WIFI på Raspberry Pi-en (Richardson & Wallace, 2016).



Figur 28 Raspberry Pi tilkoblingsmuligheter

5.2.2 GPIO

En av funksjonalitetene som gjør Raspberry Pi unik sammenlignet med andre microkontrollere er kombinasjonen av et Linux-operativsystem og 40 stk «General purpose input output» (GPIO) pinner. Kombinasjonen av et filsystem og integrert input og output gjør at det kreves mindre arbeid med programvare og mindre behov for integrering av hardware, for å kunne lese sensorer og logge avlesningen i en fil. Ved å bruke andre typer microkontrollere for å lese av en sensor, er det også behov for å programmere og integrere et eget lagringssystem for å kunne lagre dataen i en fil. På en annen side er det ikke mulig å lese analoge sensorer med Raspberry Pi sitt integrerte GPIO system. Skal det brukes analoge sensorer kreves det en «analog to digital converter» (ADC) (Richardson & Wallace, 2016).

5.2.3 Kommunikasjonsprotokoller

Raspberry Pi har 4 integrerte kommunikasjonsprotokoller. 1-wire bus, I^2C, SPI, og UART. Disse kommunikasjonsprotokollene har faste pinner på GPIO-en. Gruppen valgte å se på 1-Wire på grunn av brukervennligheten med temperatursensor. I^2C ble valgt til videre undersøkelse på grunn av tilgjengelighet av kodeeksempler, og tilsynelatende har de aller fleste digitale akselerometer I^2C grensesnitt.

5.2.3.1 1-Wire Bus

1-wire bus er en kommunikasjonsprotokoll som er designet av Dallas Semiconductors Corp og benytter seg av kun en ledning til å sende data. 1-Wire ble designet for bruk med enheter som sender små datamengder som temperatursensorer. I tillegg til en dataledning kreves det en jordledning. Sensorer som drives via 1-wire kan benytte dataledningen i kombinasjon med en kondensator som spenningskilde. Men for å sikre tilstrekkelig spenningstilførsel, kan det benyttes en egen spenningsleder til sensoren. Da blir det totalt 3 ledere fra sensoren til GPIO-en (Gay, 2018). Raspberry Pi har ferdiginstallerte drivere for å bruke temperatursensorer som DS18B20 med 1-Wire Bus. Det reduserer vanskelighetsgraden til å sette sammen et system for å måle temperatur. I tillegg benytter disse driverne seg av et adressesystem som tillater enkel parallell-kobling av flere temperatursensorer om det skal være behov for å måle med flere sensorer samtidig (Gay, 2018).

5.2.3.2 I2C

I^2C ble utviklet av Phillips i 1982. Denne kommunikasjons-protokollen ble designet for kommunikasjon innad på kretskortet og ved lave hastigheter. I^2C benytter seg av to dataledninger og to ledere til +3,3v og jord. Med totalt 4 ledere er det en fordel med tanke på kostnad av kabel (Gay, 2018). Populære bruksområder til I^2C-protokollen inkluderer kommunikasjon med ADC-er og data-logging fra sensorer (S. Parab, et al., 2008). Original hastighet med I^2C var 100kbit/s. Men i dag er det 4 hastighetsstandarder som benyttes (Gay, 2018):

- Standard mode: 100kbit/s
- Fast mode: 400kbit/s
- Fast Mode Pluss: 1Mbit/s
- High speed:3,4Mbit/s

I databladet til prosessoren (BCM2835) som benyttes i Raspberry Pi er det oppgitt at BCM2853 kjører I^2C protokollen på 400kbit/s. Men på en annen side oppgir databladet at BCM2853 skal være kompatibel med Phillips I^2C v.2.1 Januar 2000. Phillips I^2C v.2.1 støtter High speed mode på 3,4Mbit/s. Det er derfor uklart hva som er makshastigheten til I^2C bussen på en Raspberry Pi 3B+. Men det er mulig å endre I^2C hastigheten til Raspberry Pi ved hjelp av oppstarts-filen. Det er derfor mulig å forsøke å sett opp hastigheten om det skulle vise seg at avlesningen fra akselerometeret går for sakte.

I^2C protokollen var utviklet med tanke på kommunikasjon over korte avstander. Kapasitans i ledningen kan bli et problem over lengere distanser. Derfor begrenses praktisk kabel-lengde med bruk av I^2C til ca. 1m (Mosaic Industries, Inc., u.d.). 1m kabel-lengde skal være tilstrekkelig for vårt formål.

For å kommunisere med sensorer over I^2C kan det brukes SMBus modulen i Python. Dette er en enkel modul å bruke. Det benyttes en kommando(smbus.read_byte_data()) for å lese et register på sensoren og en kommando(smbus.write_byte_data()) for å skrive i et register på sensoren. Dette gjør det lettvint og oversiktlig å kommunisere med I^2C enheter. I tillegg er det mange tilgjengelige eksempler for å bruke SMBus i Python.

5.2.4 Driftsspenning

Raspberry Pi leveres med en spenningsadapter på 5V og 2.5A. Det skal være tilstrekkelig for å drifte Raspberry Pi med 7 tommers LCD skjerm, og sensorer som driftes på spenning under 5V. Skal det benyttes sensorer som krever driftsspenning på over 10V, eller som trekker mer enn 1.5A er det behov for ekstern spenningskilde (Richardson & Wallace, 2016).

5.2.5 Lagringskapasitet

Raspberry Pi benytter Micro SD minnebrikke som lagringsenhet. Maks størrelse på minnebrikke som Raspberry Pi-en aksepterer er 32GB. For å sjekke om dette er tilstrekkelig, ble det generert i Python, med kode fra github (Hohn, 2017), en csv. fil med fire koloner og ni millioner rader. Hvis Raspberry Pi-en logger med en samplerate på 3000Hz og logger tid med 3 vibrasjonsakser, tilsvarer det ni millioner rader og en sammenhengende logge-tid på 50 minutter. Den genererte csv.-filen fikk en størrelse på 249 MB (figur 29).

```
In [1]: runfile('C:/Users/Frode/Desktop/Skole/Python koder/csv test.py',
wdir='C:/Users/Frode/Desktop/Skole/Python koder')
Making 9000000 records

antall rader = 8999929
antall kolonner = 4
filstørrelse= 248.62927 MB
```

Figur 29 Fil størrelse på test

Det tyder på at en minne brikke på 32GB bør være mer en tilstrekkelig for punktlogging av flere sensorer. Hvis Raspberry Pi-en skal brukes i et system som logger data over lengre perioder, kan man benytte seg av WIFI eller Ethernet for å laste data opp på en egnet server.

5.2.6 Oppsummering av Raspberry Pi sine kapasiteter og begrensinger

Kapasiteter	**Begrensninger**
Raspberry Pi	
Brukervennlig med Linux OS	
Lett å bruke med Python	
40 stk GPIO	Leser kun digitale signal
USB, Keyboard, Mus, Ethernet, HDMI,	
WiFi	
5V USB drift	Sensorer må ha driftsspenning < 5V Behov for tilgang på stikkontakt
Lagring opptil 32GB	
I^2C	
Enkelt i bruk med Python SMBus	
Kun 4 ledere	
God tilgang på kodeeksempler	
Godt utvalg av digitale akselerometer med I^2C	Sensor kabel-lengde under 1 m Mulig hastighetsbegrensning
1-WIRE	
Godt egnet for temperatursensorer	
Ferdig installerte drivere for bruk med 18B20	
Egnet for bruk med flere sensorer	

Tabell 7 Raspberry Pi kapasiteter og begrensninger

5.3 Identifisere sensorer egnet for måling

5.3.1 Identifikasjon av temperatursensorer

Temperaturområde

Temperaturen i oljen på turbinlageret vil normalt ligge mellom 40-65°C (EBL, 1994) På Brattset har oljen ligget rundt 65°C. På bakgrunn av Brattset sin høye temperatur på turbinlageret, bør man velge et termometer som dekker forventet temperatur med god margin, 30-100 °C

Sensor-type

Det finnes generelt sett to metoder for å måle temperatur med en ekstern sensor og Raspberry Pi.

- Måle temperaturen med en analog sensor med tilhørende ADC
- Måle temperaturen med en digital sensor

Den mest brukte målemetoden av temperatur i et lager er mostandselementer av typen PT 100. (EBL, 1994) For å kunne bruke et analogt motstandselement av typen PT 100 blir det behov for å sette opp og programmere en ADC (Monk, 2016). For å sikre gjenførbarhet av oppgaven ønsker vi, om mulig, å benytte oss av komponenter som krever minst mulig eksterne kretser. I tillegg vil det være gunstig å benytte en sensor som enkelt lar seg programmere. Ved å velge en digital sensor som har innebygd ADC og 1-wire som kommunikasjonsprotokoll, økes sjansen for et vellykket prosjekt.

Innkapslingsbehov

Siden sensoren som skal brukes i denne oppgaven må kunne ligge i olje over tid, bør den ha en innkapsling som tåler belastningene. Leverandøren spesifiserte over telefon at DS18B20 skal tåle å ligge i olje over tid.

Spenningsbehov

Raspberry Pi har utganger på 3,3 volt og 5 volt. Ved å velge en sensor som kan drives av en spenning mindre enn 5 volt fjernes et eventuelt behov for en ekstern spenningskilde til sensoren. Man kan dermed koble sensoren direkte på Raspberry Pi sin GPIO.

5.3.2 Valg av temperatursensor

Som temperatursensor valgte gruppen det digitale temperatursensoren DS18B20. Denne sensoren dekker de parameterne beskrevet over. Tabellen viser sammenligning av parametere. Databladet til DS18B20 finnes i vedlegg 3.

Egenskaper	**Grenseverdier**	**DS18B20**
Temperaturområde	30-100°C	-55 - 125°C
Sensor-type	Digital	Digital
Innkapslingsbehov	Ligge i olje over tid	Spesifisert fra leverandør
Spenningsbehov	Max 5v	5v
Kommunikasjonsprotokoll	1-Wire	1-Wire

Tabell 8 Temperatursensor egenskaper og grenseverdier

Figur 30 DS18B20

5.3.3 Identifikasjon av vibrasjonssensor

	Brattset
Turtall	600rpm, 10Hz
Antall løpeskovler	14 helskovler
Antall ledeskovler	24

Tabell 9 Teknisk data Brattset

Tabell 9 viser relevante verdier som benyttes for å kunne velge passende vibrasjonssensor.

Signal-amplitude

For å kunne bestemme sensor spesifikasjoner er det nødvendig å finne forventet måleverdier. Forvente amplitude på vibrasjonen til et lagerhus på en vertikal Francis-turbin kan hentes fra ISO 20816-5, «Mechanical vibration — Measurement and evaluation of machine vibration part 5; machine sets in hydraulic power generating and pump-storage plants».

I tabell A.3 i ISO 20816-5 er det oppgitt to nivåer på vibrasjonsamplitude for lagerhuset til turbinlageret i vertikalt monterte Francis-turbiner. Disse amplitudenivåene er grenseverdier til to forskjellige aksjonsgrenser. Hver aksjonsgrense har et eget sett med tiltak som anbefales å gjennomføre dersom vibrasjonsnivået overstiger den tilhørende RMS-amplitudegrensen. Disse verdiene er satt på bakgrunn av statistikk fra tilsvarende og individuelle maskiner (International Organization for Standardization, 2018). Disse grenseverdiene bør være en god indikasjon på hvilket område det kan forventes at RMS- amplituden til vibrasjonssignalet befinner seg. Det bør derfor velges en sensor som dekker disse aksjonsgrensen med god margin.

Amplitudene fra tabell A.3 er 0,9 mm/s RMS og 1,4mm/s RMS. Verdiene er oppgitt i RMS velocity. For å kunne benytte disse til å bestemme måleområdet til et akselerometer må det regnes om til pk-pk akselerasjon. Fra ISO 20816-5 benyttes ligning 1 til å regne ut RMS-verdi av et signal. Der x_n er signal-bidrag og n er antall bidrag.

$$[1] RMS = \sqrt{\frac{x_1^2 + x_2^2 \ldots x_n^2}{n}}$$

Siden signal-bidragene kvadreres for å bli positiv blir den gjennomsnittlige øvre aksjonsgrensen ± 1,4mm/s, eller 2,8mm/s pk-pk. Den nedre aksjonsgrensen blir ±0,9mm/s eller 1,8 mm/s pk-pk.

For å uttrykke den gjennomsnittlige hastighetsverdien i akselerasjon, kan omregning gjøres ved formel 2 som er hentet fra ISO 14694:2003 (E) «Industrial fans — Specifications for balance quality and vibration levels», der ***a*** er akselerasjon i mm/s^2, $\boldsymbol{f_{rot}}$ er akselens rotasjons-frekvens og ***v*** er vibrasjons-amplituden i mm/s.

$$[2]\ a = 2\pi f_{rot} v$$

$$[3]\ a_{øvre} = 2\pi * 10Hz * 1{,}4mm/s = 87{,}96\ mm/s^2$$

$$[4] a_{nedre} = 2\pi * 10Hz * 0{,}9mm/s = 56{,}55mm/s^2$$

Den øvre aksjonsgrensen i akselerasjon [3] blir ± 87,96mm/s^2, og den nedre aksjonsgrensen [4] blir ±56,55 mm/s^2. Amplitudene kan også representeres i antall g. Hvor g er tyngdeakselerasjonen på 9,81 mm/s^2.

$$[5] a_{øvre\ i\ mg} = \frac{a_{øvre}}{g} = \frac{87{,}96mm/s^2}{9{,}81\ m/s^2} = 8{,}97mg$$

Sensoren som skal brukes må kunne måle amplitude større en 87,96mm/s^2 eller 8,97mg. Omregningen med formel [2] gjelder kun rene sinusbølger, siden ekte vibrasjonssignal sjeldent er rene sinussignal (Mobius Institute, 2016) blir dette bare en indikator på forventet akselerasjons-amplitude. I tillegg er det tatt utgangspunkt i gjennomsnittsverdier, så sensoren som skal brukes må kunne måle akselerasjonsamplituder godt over akselerasjonsverdiene regnet ut i [3] og [5]. Derfor multipliseres grenseverdien i [5] med 10 og rundes opp til 100mg. I tillegg er det anbefalt å kun bruke ca. 20% av måleområdet til akselerometer for å sikre at man får med seg uforutsette signal-amplituder (Hanley, 2016). Dette gir oss en grenseverdi på [6] 500mg

$$[6] Minimums\ sensor\ måleområde = \frac{\pm 100mg}{0{,}2} = \pm 500mg$$

Sensor-type

For å sikre at oppgaven blir gjennomførbar og følger oppsatte rammer, bør sensoren som skal brukes være billig, lett tilgjengelig, og relativt lett å bruke med Raspberry Pi. For at sensoren skal lett kunne brukes med Raspberry Pi, bør sensoren ha et digitalt grensesnitt. En sensor med digitalt ut-signal fjerner behovet for å integrere en egen ADC, med tilhørende filter og forsterker, i målekjeden. I tillegg til å ha et digitalt grensesnitt bør sensoren ha en lite komplisert kommunikasjonsprotokoll som I^2C. I^2C er en kommunikasjonsprotokoll som er enkel å programmere på grunn av lett tilgjengelige åpne kildekoder. Sensoren bør også ha driftsspenning under 5v. Raspberry Pi har utganger på 3,3 volt og 5 volt. Ved å velge en sensor som kan drives av en spenning mindre enn 5 volt fjernes et eventuelt behov for en ekstern spenningskilde til sensoren.
Det finnes 3 hovedtyper av akselerometer:

- Piezoelektriske
- Piezoresistive
- Kapasitive MEMS

Piezoresistive akselerometer har vanligvis lav sensitivitet, de er temperatursensitive og koster relativt mye. De er best egnet til bruk med høy akselerasjon, som kollisjons testing og lignende (Hanly, 2016) og passer da dårlig til vårt formål.

Piezoelektriske akselerometer er den mest brukte typen akselerometer til vibrasjonsmåling. De har god sensitivitet og lavt støynivå, men de krever en egen spenningskilde og har et analogt utsignal. De krever dataloggere som er dyre sammenlignet med microkontrollere som Raspberry Pi. Piezoelektriske akselerometer har en dempning ned mot 0Hz og måler ikke statisk akselerasjon. Det kan derfor i teorien være problematisk å måle signaler under 2 Hz. Men det finnes et godt utvalg av høysensitive piezoelektriske akselerometer med frekvens-respons helt ned til 0.1 Hz, så dette er ikke et problem i praksis. De er en del dyrere en MEMS-akselerometer og er ikke like enkle å kjøpe kommersielt (Hanly, 2016). På grunn av analogt utsignal, pris og tilgjengelighet er piezoelektriske akselerometer ikke egnet til vårt formål.

Siste typen akselerometer er kapasitive MEMS-akselerometer. Disse akselerometrene er små microchiper som kan monteres direkte på et kretskort (PCB), noe som gjør dem meget allsidige. En vanlig størrelse på en MEMS-akselerometer-chip er 6x6mm og veier under ett gram (Spence, 2017). MEMS-akselerometeret måler også statisk akselerasjon, og har ingen dempning av signalet ned til 0Hz (Hanley, 2016). Digitale MEMS-akselerometer er billige, kommersielt lett tilgjengelige og egnes godt til bruk med mikrokontrollere som Raspberry Pi og lignende. Dette gjør de godt egnet til utvikling av enkle og billige systemer. MEMS-akselerometer passer godt til vårt formål.

Båndbredde

Den båndbredden som er av interesse å måle defineres også i ISO 20816-5. I tabell 1 defineres minimumsfrekvens for Francis-turbiner ved ligning [7] og maksimumsfrekvens ved ligning [8]

$$[7]\ f_{min} = 0{,}1 * f_{rot} = 0{,}1 * 10Hz = 1Hz$$

$$[8]\ f_{max} = 3 * z_R * f_{rot} = 3 * 14 * 10Hz = 420Hz$$

Der f_{rot} er rotasjonsfrekvensen til akslingen i Hz og z_R er antall løpeskovler. Dette gir f_{min} =1Hz og f_{MAX} =420Hz. Ved valg av sensor økes f_{MAX} opp til 1000Hz på grunn av avrunding i lavpassfilter.

Samplefrekvens

Samplefrekvensen som skal benyttes må være høy nok for å få en representativ måling av vibrasjons-signalet. Ifølge Nyquist-Shannon teoremet skal samplefrekvensen være minimum 2,56 ganger større enn maksimal-frekvensen av interesse. ISO 20816-5 anbefaler en samplefrekvens på minimum 4 ganger maksimal-frekvensen av interesse. Dette gir en samplefrekvens på minimum 1680 Hz [9].

$$[9]\ Fs = 4 * f_{max} = 4 * 420Hz = 1680Hz$$

Oppløsning og Sensitivitet

I digitale MEMS-akselerometer er det en innebygd ADC. Oppgaven til en ADC er å konvertere det analoge vibrasjonssignalet om til et digitalt signal som kan behandles av datainnsamleren. Et digitalt signal består av et binært tall der hvert siffer kalles en bit. Antall bit bestemmer hvor mange verdier eller tall, som er tilgjengelig for å beskrive det analoge signalet. Ta for eksempel et akselerometer som måler over ±2g med en oppløsning på 12 bit. Dette akselerometeret vil ha 2^{12}=4096 verdier tilgjengelig for å beskrive et område på 4g. Det vil si at oppløsningen blir 4096/4g =1024 verdier per g. Det vil også si at oppløsningen definerer sensitiviteten til akselerometeret.

Vi ser videre på eksempelet med ±2g og 12 bit. Her blir sensitiviteten 4g/4096=0,98mg per linje. Det betyr at det må skje en endring i vibrasjonssignalet på mer enn 0,98mg for at det skal registreres av dataloggeren. Analoge signaler som har verdier mellom de tilgjengelige binære tallene blir altså avrundet til nærmeste digitale verdi. For å være sikker på at verdiene som måles ikke blir avrundet for mye i konverteringen fra analogt til digitalt må ADC-en ha tilstrekkelig oppløsning.
Vi setter grensen på sensitiviteten til 1% av differansen mellom aksjonsgrensene [10].

$$[10]\ Grenseverdi\ sensitivitet = (a_{øvre} - a_{nedre}) * 0{,}01 = (87{,}96mm/s^2 - 56{,}55mm/s^2) * 0{,}01 = 0{,}3141mm/s^2$$

Dette grensenivået bør være tilstrekkelig for å sikre en pålitelig påvisning av en RMS-amplitude over eller under de oppgitte aksjonsgrensene. I tillegg er differansen mellom aksjonsgrensen lavere enn den nedre aksjonsgrensen, så en grenseverdi på 0,3141mm/s^2 bør også være tilstrekkelig for å påvise om RMS-amplituden ligger under nedre aksjonsgrense.
For å bestemme oppløsningen som er nødvendig å ha i ADC-en for å oppnå vår grenseverdi, regner vi først om grenseverdien om til benevnelsen mg (tusendels tyngdeakselerasjon) [11].

$$[11] Grenseverdi\ i\ mg = \frac{Grenseverdi\ sensitivitet}{g} = \frac{0{,}3141mm/s^2}{9{,}81m/s^2} = 0{,}032mg.$$

Bestemmer så antall linjer dette tilsvarer over et område på ±2g [12].

$$[12] Antall\ linjer = \frac{måleområde}{grenseverdi} = \frac{4000mg}{0{,}03199mg} = 125039$$

Finner så hvor mange siffer, eller bits, som er nødvendig for å kunne skrive 125039 forskjellige linjer med et binært tall [13].

$$[13]\ Antall\ bits = 2^x = 125039 \rightarrow x * ln2 = ln125039 \rightarrow x = \frac{ln125039}{ln} = 16{,}93 \rightarrow 17bits$$

Oppløsningen til ADC-en i vår sensor bør være på minimum 17 bits ved ±2g for å oppnå en sensitivitet på mindre enn 0,03199mg. Denne grenseverdien tar utgangspunkt i en maskin med fast akselfrekvens og fastsatte aksjonsnivåer med tilhørende RMS-amplitude. Det er derfor ikke nødvendigvis tilstrekkelig med 17 bits eller mer i oppløsning for andre vibrasjonsmålingsapplikasjoner.

Støynivå

Støynivå i et akselerometer er amplituden på det signalet som akselerometeret produserer når det ikke blir utsatt for noe akselerasjon. Signal som har amplitude mindre enn dette støynivået blir ikke leselig i måledataene som hentes ut fra akselerometeret. I digitale MEMS-akselerometer med innebygd ADC oppgis støynivået som en støy-tetthet som er avhengig av båndbredden til akselerometeret. Støy-tettheten er oppgitt i enheten µg/√Hz. For å få amplituden på støynivået oppgitt i µg, multipliseres støy-tettheten med kvadratroten av båndbredden som måles (Hanley, 2016).

Vi har ikke funnet noe direkte krav til maksimalt støynivå. Men oppgitt støynivå på sensoren bør ligge så mye lavere enn forventet amplitude at det ikke er noe tvil om at målingene ikke påvirkes av støynivået. Derfor mener gruppen at amplituden på støynivåer ikke bør overstige 10mm/s^2. Med en båndbredde på 1000Hz blir maks tillatt støy-tettheten 32 µg/√Hz, om støynivået ikke skal overstige 10mm/s^2.

I tillegg kan man se i Norconsult sin målerapport fra Svorkmo agg.1 (Bjørndal, 2007) at Norconsult bruker IMI 629A11 akselerometer som har en støy-tetthet på 40 µg/√Hz (IMI sensors, 2017). En grenseverdi på 32 µg/√Hz bør da være tilstrekkelig.

Antall akser

Grenseverdiene som oppgis ISO 20816-5 gjelder absoluttvibrasjonen til lagerhuset. ISO 20816-5 anbefaler 2stk akselerometer plassert med 90grader mellom aksene og radielt på akslingen. For at oppgaven skal bli gjennomførbar velger gruppen å fokusere på ett akselerometer med minimum 2 akser, og benytte ett målepunkt for å måle absoluttvibrasjonen. Ved å benytte ett akselerometer kan et eventuelt måleresultat ha mindre troverdighet enn å benytte 2 stk. akselerometer montert 90 garder på hverandre, som beskrevet i ISO 20816-5. Gruppen fokuserer på ett akselerometer for å kunne se om konseptet er gjennomførbart. Om systemet fungerer tilstrekkelig, tillater I^2C bussen utvidelse av flere sensorer. I tillegg har Raspberry Pi 2 stk. I^2C innganger. Derfor kan det være mulig at det ikke krever mye arbeid å utvide til 2 sensorer om systemet fungerer med en sensor. Men på en annen side kan det bli en utfordring å konstruere softwaren til to sensorer slik at målingene er synkronisert.

5.3.4 Valg av vibrasjonssensor

Som vibrasjonssensor valgte gruppen MEMS-akselerometeret ADXL355. Dette akselerometeret dekker behovet som beskrevet over. Tabellen viser sammenligning av parametere. Databladet til ADXL355 finnes i vedlegg 4.

Egenskaper	**Grenseverdier**	**ADXL 355**
Måleområde	±500mg	±2,048g, ±4,096g, ±8g
Sensor-type	Kapasitive MEMS	Kapasitive MEMS
Båndbredde	1-1000Hz	0-1000Hz
Sampling-frekvens	1680Hz	4000Hz
Oppløsning	17 bit, ved ±2g	20 bit
Sensitivitet	0,03199mg	0,0039mg ved ±2,048g
Støytetthet	32 µg/√Hz	25 µg/√Hz
Antall akser	2	3
Spenningsnivå	Max 5V	3,3V

Tabell 10 Vibrasjonssensor egenskaper og grenseverdier

Figur 31 ADXL355

6 Resultatmål 2: Prototype

I dette kapittelet presenteres en oversikt over sammenkobling av Raspberry Pi med sensorer og tilhørende komponenter.

6.1 Hardware

Maskinvare(*hardware*) er de fysiske delene i et datasystem (Store Norske leksikon , 2018).

Komponentene gruppen benyttet til denne prototypen er listet under:

Komponenter	**Kommentar**
Raspberry Pi	Ettkortsdatamaskin
Touchscreen	Skjerm, interface til Raspberry
DS18B20	Temperatursensor
ADXL335	Vibrasjonssensor
Skjermet kabel	Kontakt mellom kabelkontakt og sensor
Kabelkontakter	Kontakt mellom Raspberry Pi og sensorkabel
Rustfritt sensorhus	Sensorhuset og innstøping av sensor
Pinner, jumper kabler	Overganger
Kabinett	3D-printing, SolidWorks, Cura

Tabell 11 Komponenter tilstandskontrollsystem

Raspberry Pi

Vi benytter oss av en microkontroller, som er en datamaskin på en liten chip med programmerbar prosessor som kan styre alt fra mobiler, fjernkontroller, tastaturer, datamus og lignende. Ved hjelp av egnet programvare samt en sensor som er kompatibel, kan Raspberry Pi 3B+ fungere som en datalogger.

Figur 32 Raspberry Pi

Touchskjerm

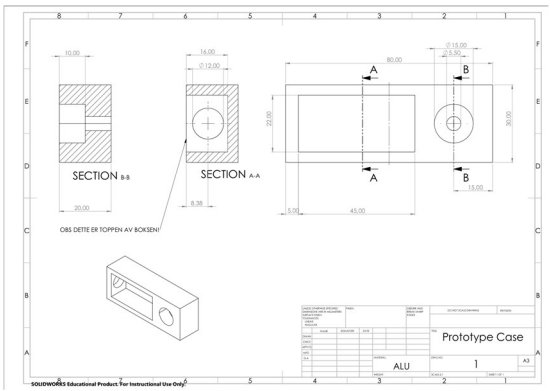
Vi benyttet oss av en touchskjerm på bakgrunn av at brukergrensesnittet og brukervennligheten ble bedre. Ved oppsett og under måling vil det være fordelaktig å ha både keyboard og mus, noe vi har benyttet under målinger i denne oppgaven. En annen faktor ved å benytte en touchskjerm er at vi slipper bruken av en ekstern datamaskin under måling. Databladet til touchscreen finnes i vedlegg 5.

Figur 33 Raspberry Pi touchscreen

Gruppen gikk til innkjøp av en lengre ribbon-kabel (hvit bred kabel), siden en for kort kabel fulgte med skjermen.

6.1.1 Sensorhus

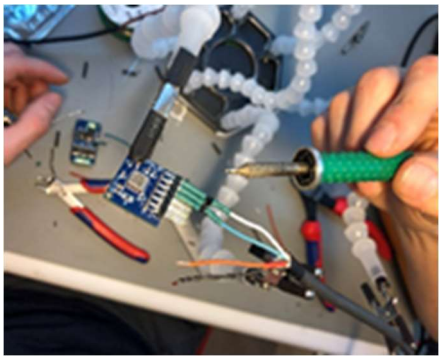
Sensorhuset til vibrasjonssensoren ble tegnet i SolidWorks, som er et egnet program for utforming av tekniske enheter. Etter ferdigstilling av 3D-modellen kontaktet gruppen Arild Sæther som har ansvaret for verkstedet til instituttet for maskinteknikk og produksjon på Valgrinda. På verkstedet freste de ut et sensorhus i rustfritt stål, som er det materialet som blir mest anvendt ved flere ulike sensorer innenfor industrisektoren. Når sensorhuset var ferdig frest, fikk vi testet passform med ADXL355(akselerometeret) plassert i sensorhuset.



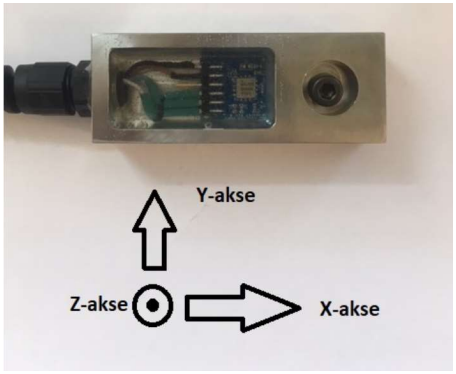
Figur 34 Iso tegning sensorhus

Figur 35 Sensorhus rustfritt stål

Etter at gruppen hadde testet at akselerometeret og sensorhuset passet overens, ble lederne i sensorkabelen loddet til pinnene på ADXL355(figur 36). I tillegg til lodding brukte vi også krympestrømpe ved hver lodding, for å beskytte mot uønsket kontakt og kortslutning.



Figur 36 Lodding av sensor



Figur 37 Akseretning ADXL355

Signalet fra et akselerometer kan lett påvirkes av støy (Bye, 2009). Gruppen valgte derfor å benytte en skjermet sensorkabel.

For å feste sensoren i sensorhuset ble innstøping valgt som metode. Materialet som skal benyttes til innstøpingen bør ha høy e-modul for å minimere demping. John Inge Edvardsen, overingeniør ved Materialteknologi, NTNU, antok at en epoxy kunne tilfredsstille våre behov og anbefalte oss å kontakte Struers. Struers er representert i over 50 land og har mer enn 140 års erfaring innen materialteknologi (Struers, 2019). Struers anbefalte Epofix til vårt formål. John Inge Edvardsen videreformidlet oss til avdelingsingeniør Berit Kramer, som ga oss tilgang til lab, Epofix og brukermanual.

Gjennomføring av innstøpning

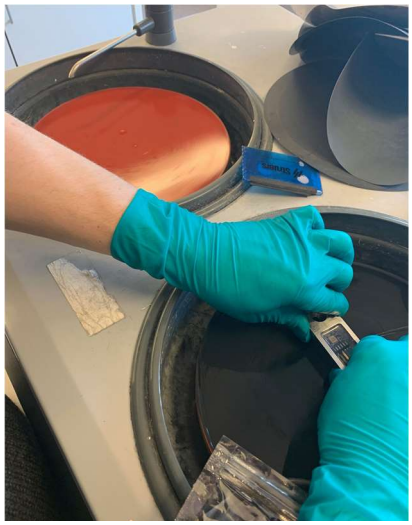
- Bland 25 g Epofix resin og 3 g Epofix hardener
- Kontinuerlig røring i 120 sekunder før man fyller tilstrekkelig med Epofix i utfresingen.

Figur 38 Resin og Hardner blandes

Figur 39 Vakuum beholder

Når vi hadde fylt utfresingen med Epofix, ble sensor-huset lagt i en vakuum-beholder i 5 min for å trekke ut alle eksisterende luftbobler i epoxyen. Etter at luften ble trukket ut av epoxyen ble sensor-huset satt til herding hvor epoxyen fikk herde i 8-12 timer.

Etter herdingen, ble sensorhuset våtslipt, for å få en finere finish og slipe bort overflødig epoxy. Under våtslipingen ble det slipt i flere omganger med sandpapir med forskjellig ruhet. De ulike sandpapirene vi benyttet var P2000 for å få vekk det groveste og P4000 for en finere finish.



Figur 40 Våtsliping av vibrasjonssensor hus

Figur 41 Ferdig laget vibrasjonssensor

6.1.3 Temperatursensor

Her vises en oversikt over komponenter samt sammensetting av temperatursensoren.

Figur 42 Temperatursensor medfølgende pullup-motstand, 2 stk kabelkontakter og jumperkabel

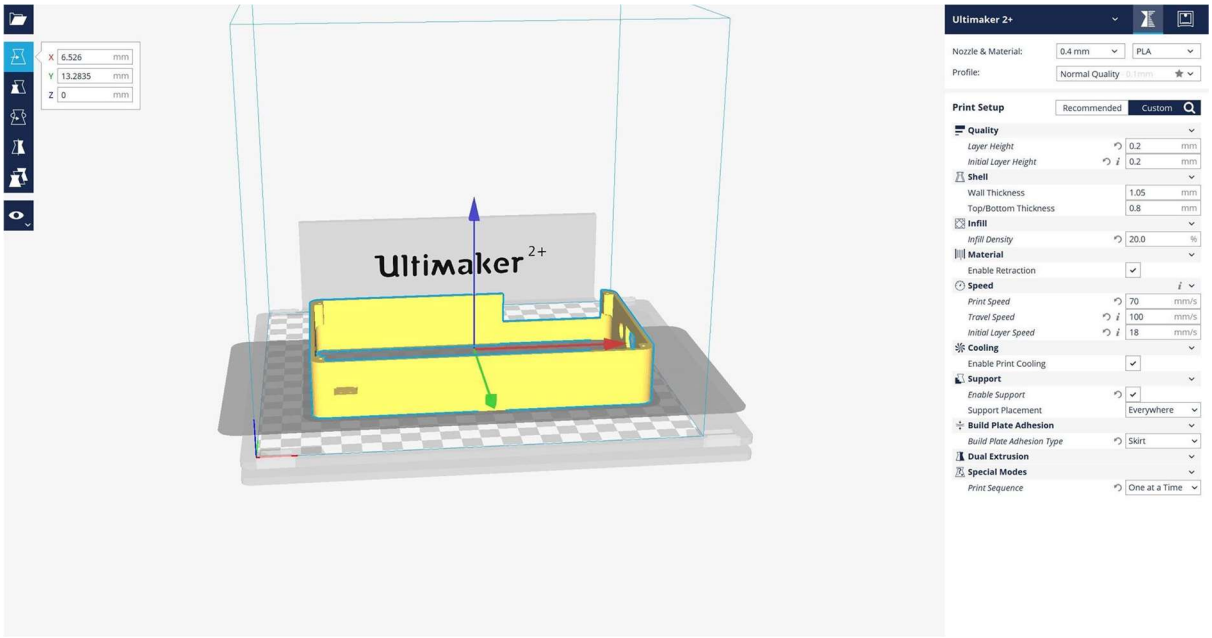
Det første som ble gjort var å dele opp samt avisolere kabel-enden og dermed føre de inn i kabelkontakten. Når ledningene og kabelkontakten er koblet sammen startet gruppen med å lage et passelig hull i rammen til kabelkontakten, som gjør det enkelt med på- og av-kobling av temperatursensoren.

Etter at kabelkontakten er festet, loddet vi jumperkabler til pinnene ved kabelkontakten som vist i figur 42. Jumperkablene er skrudd fast til medfølgende pull-up motstand. Oppgaven til pull-up-motstanden er å sikre et positivt høyt logisk nivå på 1-Wire-bussen.
Sistnevnte kabler er satt på GPIOen til Raspberry Pi med original kabelsko med krympestrømpe rundt som beskytter mot kortslutning.

6.1.2 Kabinett

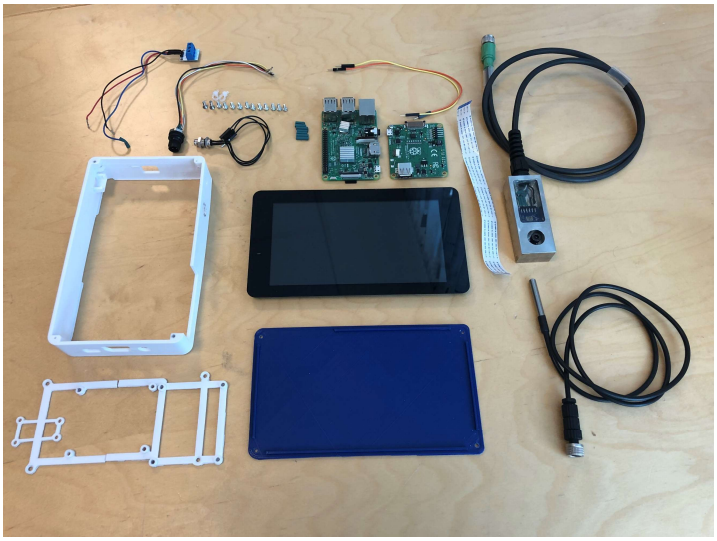
For å få tilgang til MakeNTNU sine 3D-printere, var 2/4 medlemmer av gruppen på kurs med MAKE NTNU for grunnleggende kunnskaper. Etter kurset fikk vi tilgang til 3D-printerene.
Programmet som ble brukt under designfasen av rammer, braketten og lokket er SolidWorks. Det er et egnet program, som fungerer godt til prototyping. Designet til kabinett var hentet fra en åpen kilde (Thingiverse, 2015). Iso tegning finnes i vedlegg 6

Modellen som var ferdig i SolidWorks ble overført til programmet Cura. I Cura blir 3D-modellen omformet til en "brukermanual" for printerne, slik at man får printet ut ønsket enhet. Cura gir mange muligheter i henhold til hastighet på printeren, tykkelse på lagene, om flaten skal varmes opp og temperatur til plasten.



Figur 43 CURA - med egendefinerte innstillinger

6.2 Sammensetting

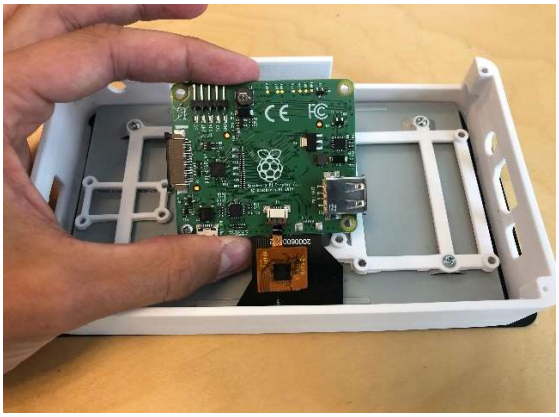


Figur 44 Deler for sammensetting

Steg 1: Rammen settes rundt touchskjermenn før braketten skrus til fast til baksiden av touchskjermen med bruk av 4 maskinskruer M3*5.

Steg 2 Monter display-kortet til Touchscreen.

Steg 1



Steg 2

Steg 3: Skru fast Raspberry Pi og display-kort på braketten som er festet til touchscreen med bruk av 8 maskinskruer M3*5.

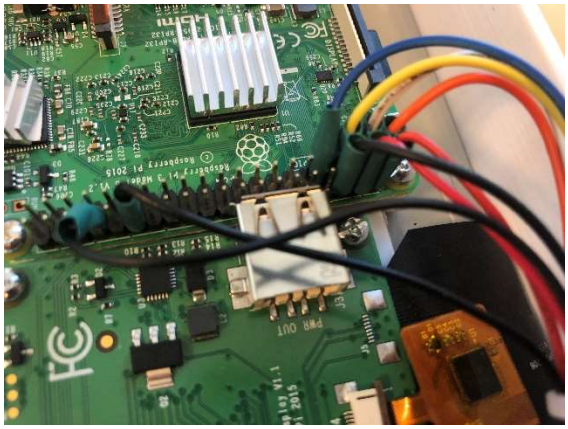
Steg 4: Fest kontakt vibrasjon 4-polet M12 og kontakt temperatur 3-polet M8 med bruk av mutter.

Steg 3

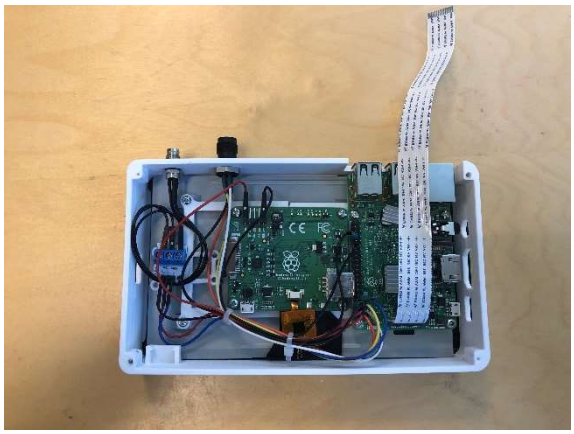
Steg 4

Steg 5: Plasser jumper-kabel på riktig GPIO, med bruk av krympestrømpe. Koblingsskjema finnes i kap. 6.2.1.

Steg 6: Koble Raspberry Pi opp mot touchscreen ved å montere ribbon cable mellom display-kort og Raspberry Pi



Steg 5



Steg 6

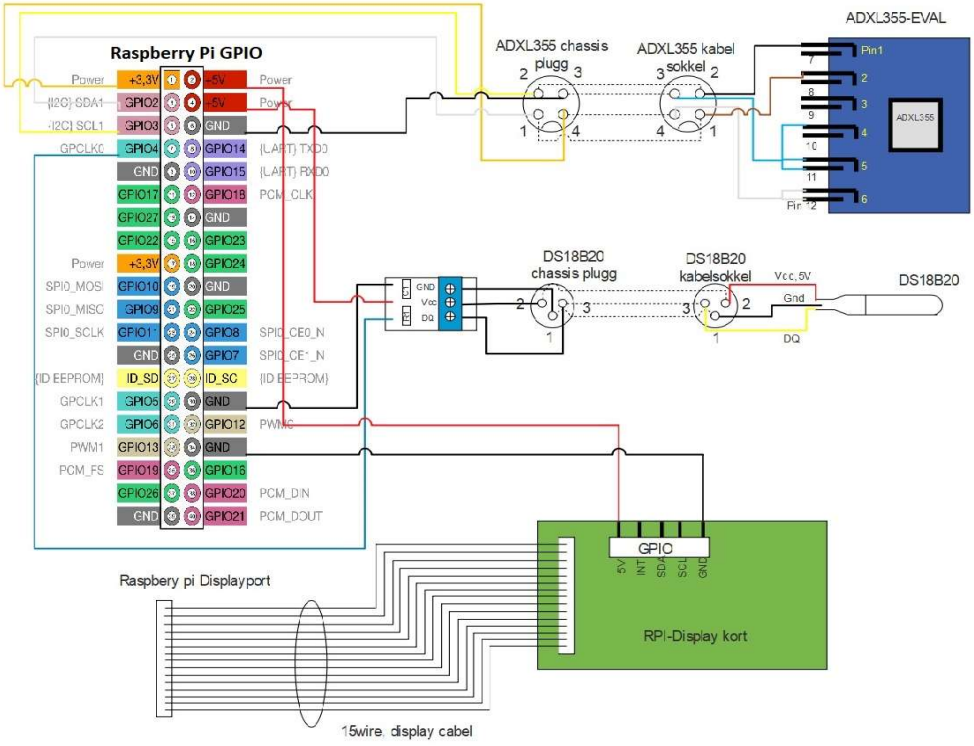
Steg 7: Skru fast deksel til kabinettet med bruk 4 maskinskruer M5*25.

Steg 8: Monter temperatursensor og vibrasjonssensor på tilhørende kontakter. Prototypen er ferdigstilt og brukermanual beskrives i kap.7.

Steg 7

Steg 8

6.2.1 Koblingsskjema



Figur 45 Koblingsskjema prototype

6.3 Kostnadsoversikt

	Antall	Produkt	Pris per stykk	Totalpris
	1	Raspberry Pi 3 Modell 3 b+ premium kit	399,00	399,00
	1	7` Touchscreen Display	999,00	999,00
	1	DS18B20	149,90	149,90
	1	ADXL355	267,05	267,05
	1	Ribbon Cable 200mm	18,90	18,90
	1	Kontakt temperatur 3-polet M8	68,30	68,30
	1	Kabelkontakt 3-polet M8	70,30	70,30
	1	Kontakt Vibrasjon 4-polet M12	44,25	44,25
	1	Sensorkabel M12	196,25	196,25
	1	Strekkavlastning M12 Polyamid	10,20	10,20
	8	Maskinskrue M3*5	0,30	2,40
	1	Maskinskrue M5*25	0,30	0,30
	4	Plateskrue 2,9*16	0,30	1,20
Kalkulert	145g	3D-printet kabinett	0,66kr/g	95,70
		Vibrasjonssensor hus		200
Total				2427,05.-

Tabell 12 Totale kostnader

Produkt	Kilde	Beregning	Pris
3D-printet Kabinett	(Japan Photo, 2019)	495kr/750g	0,66kr/g

Tabell 13 Kalkulerte kostnader

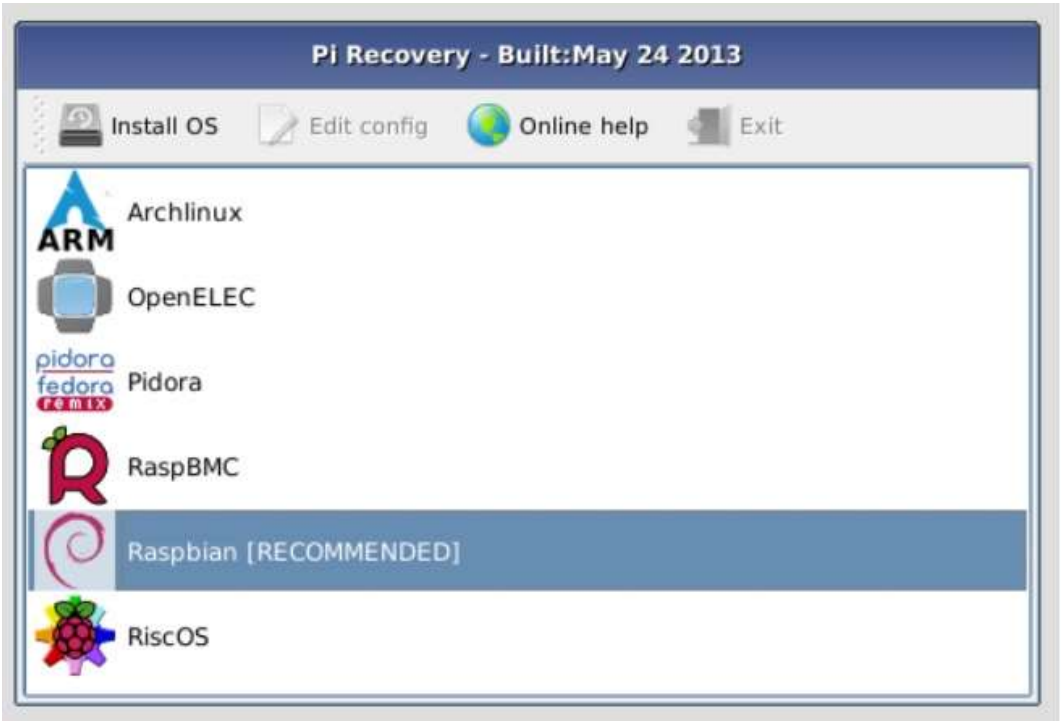
Kvitteringer finnes i vedlegg 7.

7 Resultatmål 3 Brukermanual

Dette kapittelet inneholder en brukermanual som systematisk beskriver hvordan prototypen, som er beskrevet i kap. 6 klargjøres for bruk, og hvordan den kan brukes til å logge temperatur og vibrasjon. Hensikten med dette kapittelet, er at det skal være mulig for andre studenter å benytte seg av manualen for å kjøre en tilsvarende prototype.

7.1 Oppsett Raspberry Pi

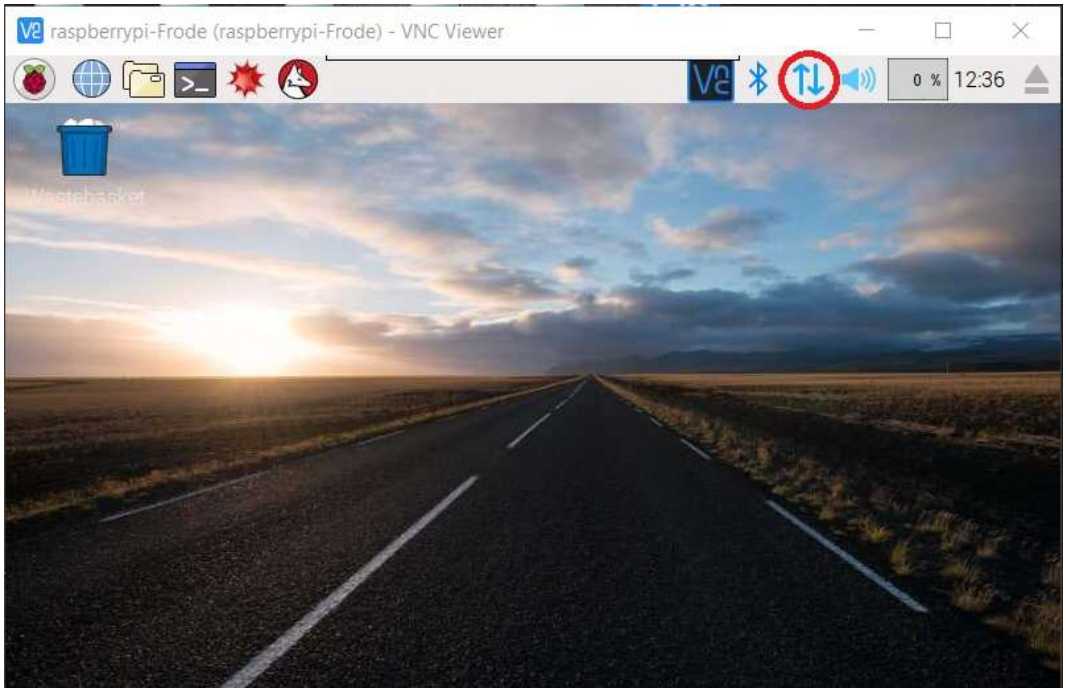
Det første steget er å installere operativsystemet Raspbian på Raspberry Pi. Om det allerede er installert Raspbian på Raspberry Pi-en, er dette ikke nødvendig.
Operativsystemet installeres fra et Micro SD-kort, som følger med når man kjøper en Raspberry Pi.
Før installeringen kan starte må følgende gjøres:
-koble til mus og tastatur i USB-utgangene på Raspberry Pi-en.
-koble til en skjerm, enten via HDMI eller en touch-skjerm som beskrevet i kap. 6.2
-sett i Micro SD-kortet i kortleseren på undersiden av Raspberry Pi-en.
-koble til strømforsyningen som følger med Raspberry Pi-en, til Micro USB-porten på Raspberry Pi-en.
Når strømmen kobles til starter Raspberry Pi-en og du får opp valg meny (**Feil! Fant ikke referansekilden.**) der man kan velge operativsystem. Velg **Raspbian.**



Figur 1

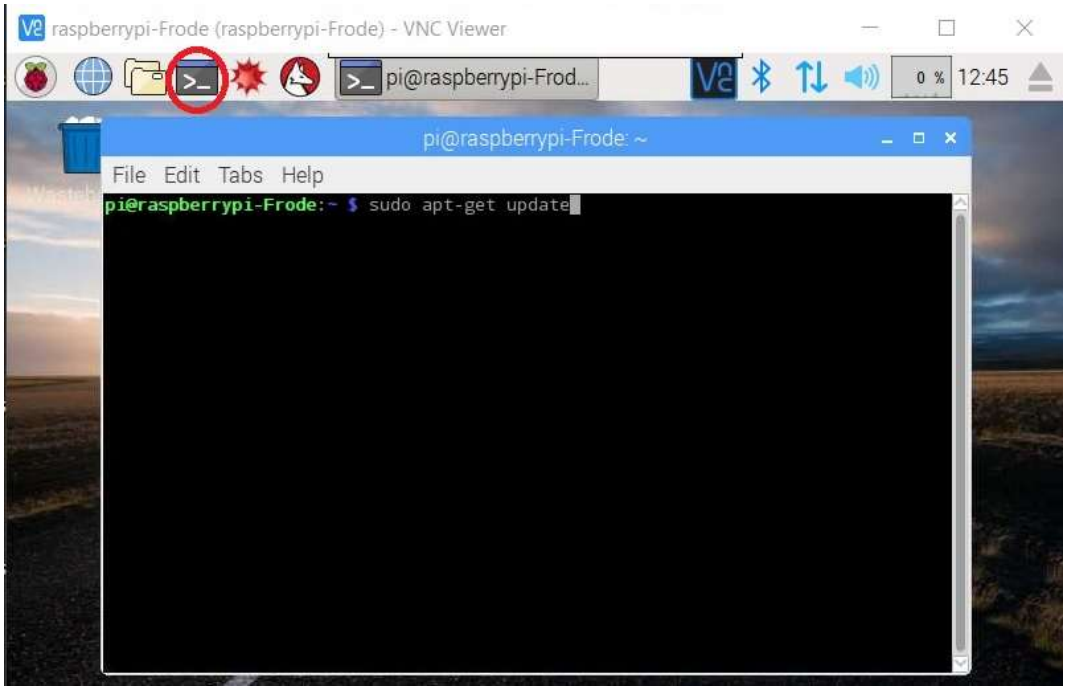
Når installasjonen er ferdig, logger du deg inn med brukernavn: pi og passord: raspberry.

Neste steg er å oppdatere operativsystemet på Raspberry Pi-en. For å kunne oppdatere må du være koblet til internett. Dette kan gjøres ved å koble til en Ethernet-kabel på Raspberry Pi-en, eller trykke på WIFI-symbolet oppe i høyere hjørne (Figur), velg så ønsket nettverk og koble til.



Figur 2

Deretter åpnes kommandovinduet ved å trykke på kommandovindu knappen oppe til venstre. I kommandovinduet skrives kommandoen «sudo apt-get update» (Figur). Trykk enter.



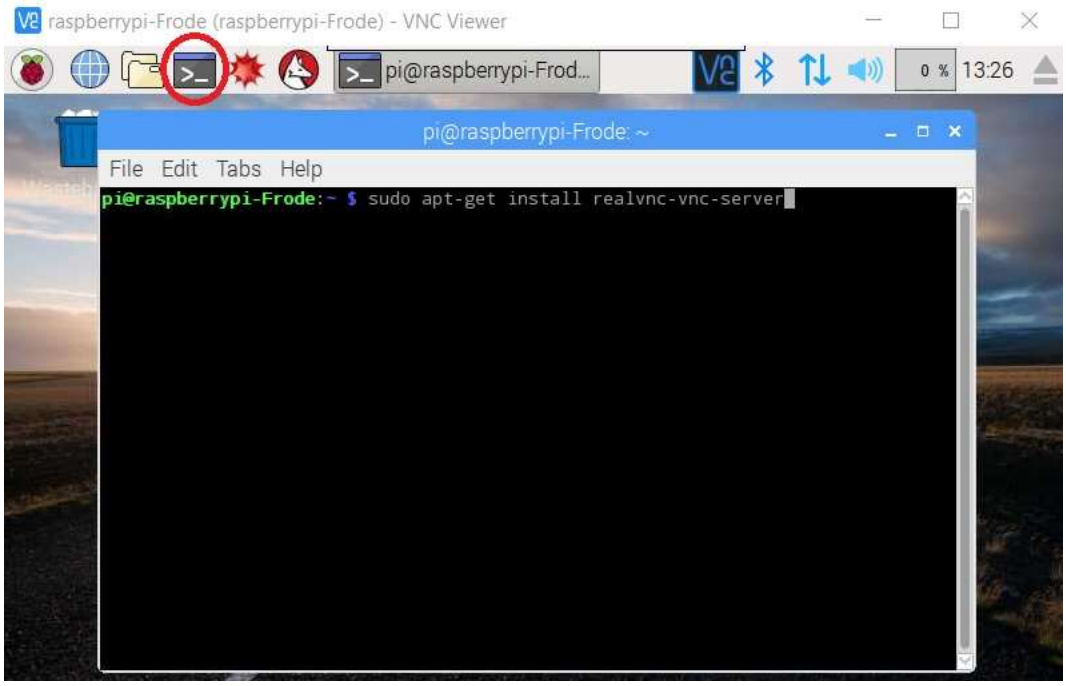
Figur 3

7.2 Oppsett av VNC

Under testing og sammenstilling av prototypen kan det være praktisk å kjøre Raspberry Pi-en fra en laptop eller datamaskin. Dette kalles «running headless». Når man kjører Raspberry Pi-en «headless» trenger man ikke å koble til en skjerm, men du bruker skjermen på laptopen din. Dette gjør det enklere å teste endringer. I tillegg er det praktisk for å flytte data til og fra Raspberry Pi-en. Det er flere metoder for å kjøre Raspberry Pi-en «headless». Her beskrives en metode som benytter VNC, Virtual Network Computing. Dette er en metode der det kjøres en VNC-server på Raspberry Pi-en. Når denne serveren er oppe og kjører, kan man via internett koble seg til Raspberry Pi-en fra hvilken som helst pc. Tilkoblingen er beskyttet med passord.

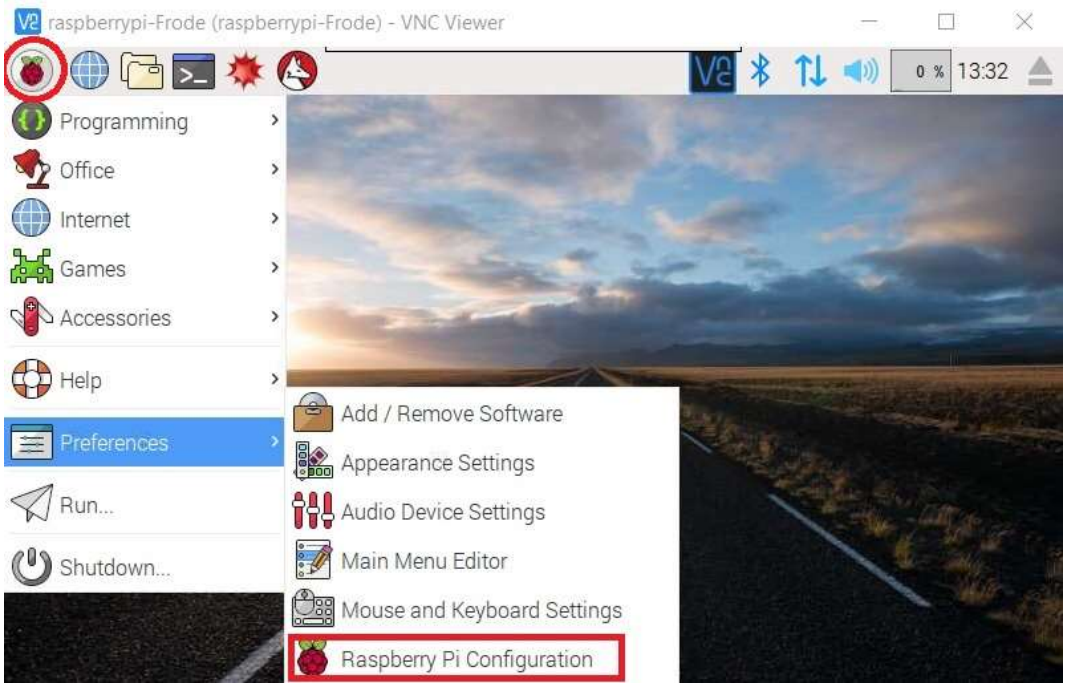
Oppsett av Real VNC-serveren krever at du har skjerm, tastatur, mus og nettverk koblet til Raspberry Pi-en.

Real VNC server er allerede installert på Raspberry Pi-en. Så første steg er å oppdatere Real VNC. Dette gjøres ved å åpne kommando-vinduet, skriv inn kommandoen «sudo apt-get install realvnc-vnc-server» og trykk enter (Figur).

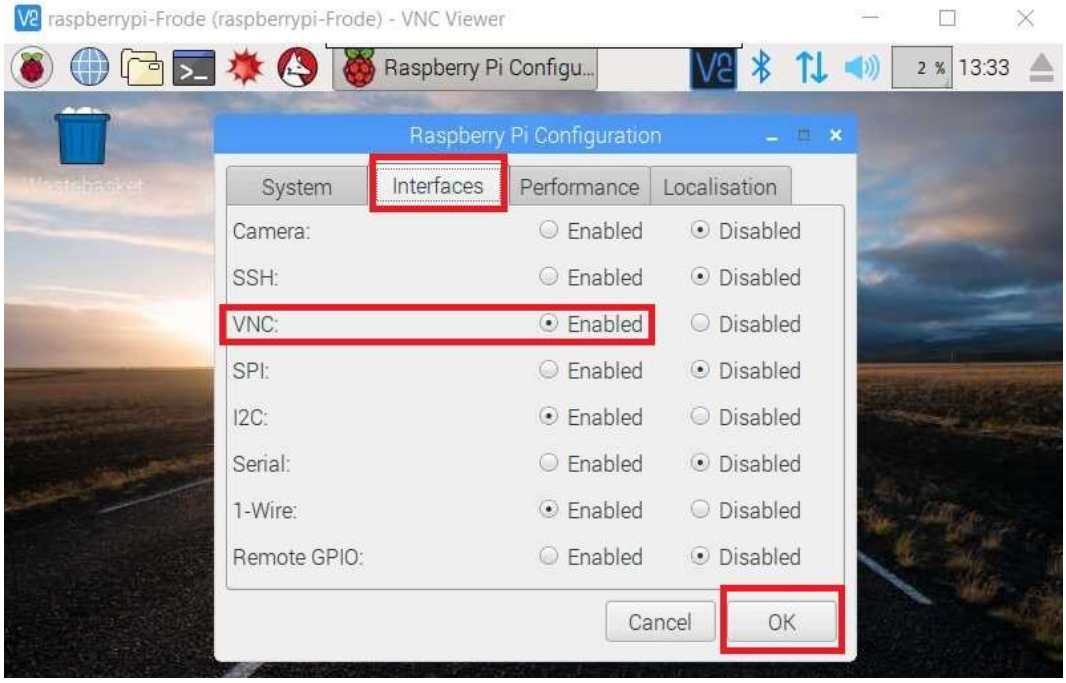


Figur 4

Nå må VNC Server aktiveres slik at den starter hver gang Raspberry Pi-en startes. Trykk på **Meny > Preferences > Raspberry Pi Configuration > Interface** og velg **VNC Enable** og trykk **OK** (Figur og Figur).



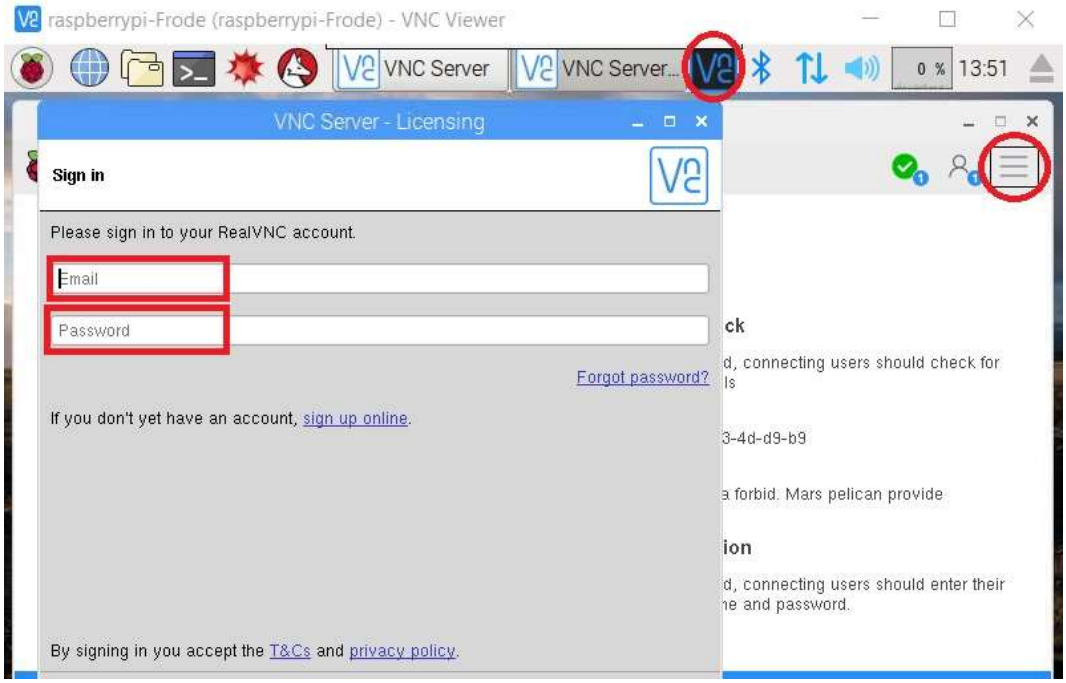
Figur 5



Figur 6

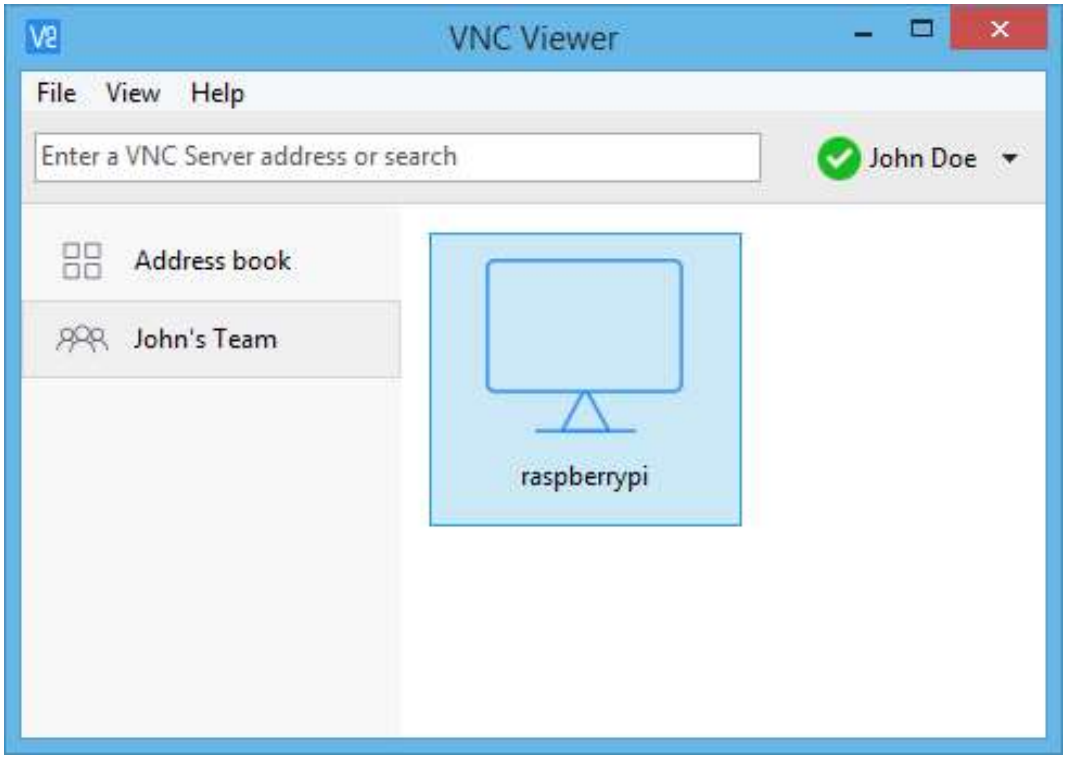
Neste steg er å lage seg en konto hos Real VNC, slik at serveren på Raspberry Pi-en kan linkes til en ønsket pc. Dette er gratis og gjøres fra en pc. Gå til www.realvnc.com og lag en konto med brukernavn og tilhørende passord.

Når du har laget deg en bruker, logger du deg inn på denne brukeren på VNC serveren på Raspberry Pi-en. Dette gjøres ved å trykke på **VNC > Meny > Licensing > Sign into your Real Vnc Account**. Skriv inn brukernavn og passord (Figur).



Figur 7

Last ned og installer Real VNC Viewer på den pc-en du ønsker å styre Raspberry Pi-en fra. Logg deg inn på brukerkontoen og Raspberry Pi-en er tilgjengelig ved å dobbeltklikke på skjermsymbolet (Figur). Første gangen du kobler til Raspberry Pi-en må du oppgi brukernavn og passord til Raspberry Pi-en, som standard er brukernavn «pi» og passord «raspberry».



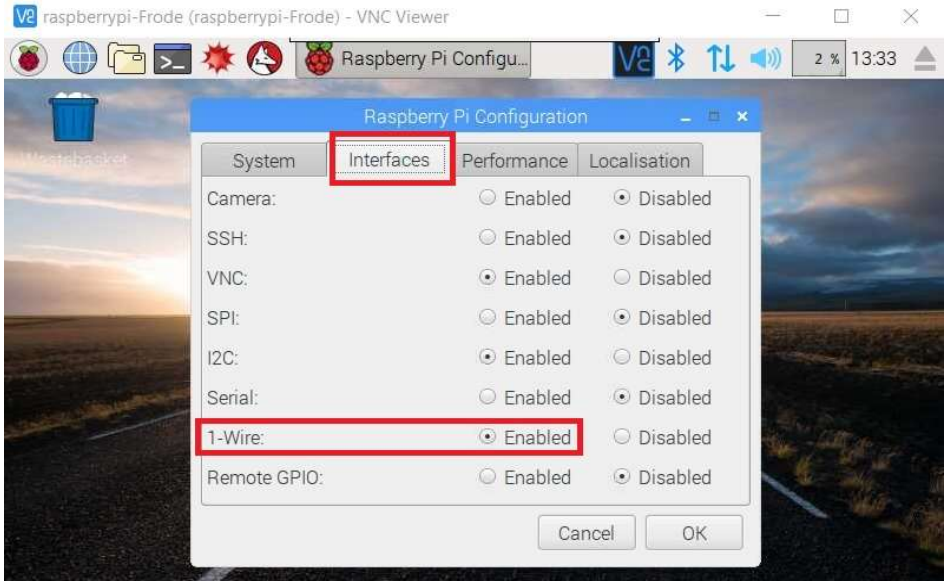
Figur 8

Nå kan man koble til Raspberry Pi-en fra en pc ved å bruke VNC Viewer og din Real VNC brukerkonto.

7.3 Temperaturmåling

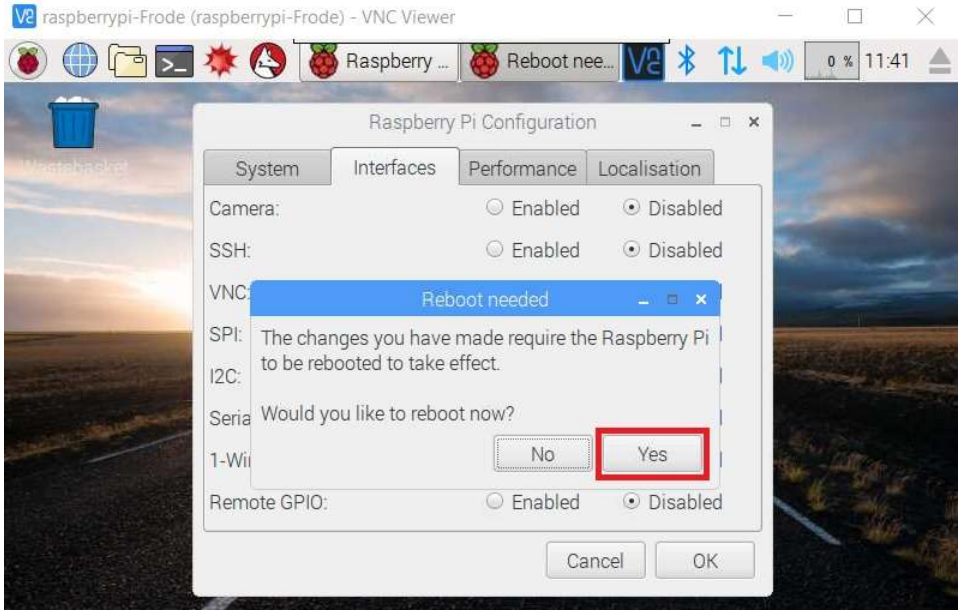
For å kunne ta temperaturmålinger med DS18B20 som beskrevet i kap. 8.1, må 1-wire buss aktiveres på Raspberry Pi-en. Dette kan gjøres ved å endre oppstartfilen config.txt, eller ved å gå inn på Raspberry Pi config. Her beskrives sistnevnte.

Trykk på **Meny > Preferences > Raspberry Pi Configuration > Interface** og velg **1-wire Enable** og trykk **OK** (Figur og Figur)



Figur 9

For at endringene skal ta effekt, kreves en omstart av Raspberry Pi-en. Det kommer automatisk opp et omstartsvalg, trykk **YES** (Figur).

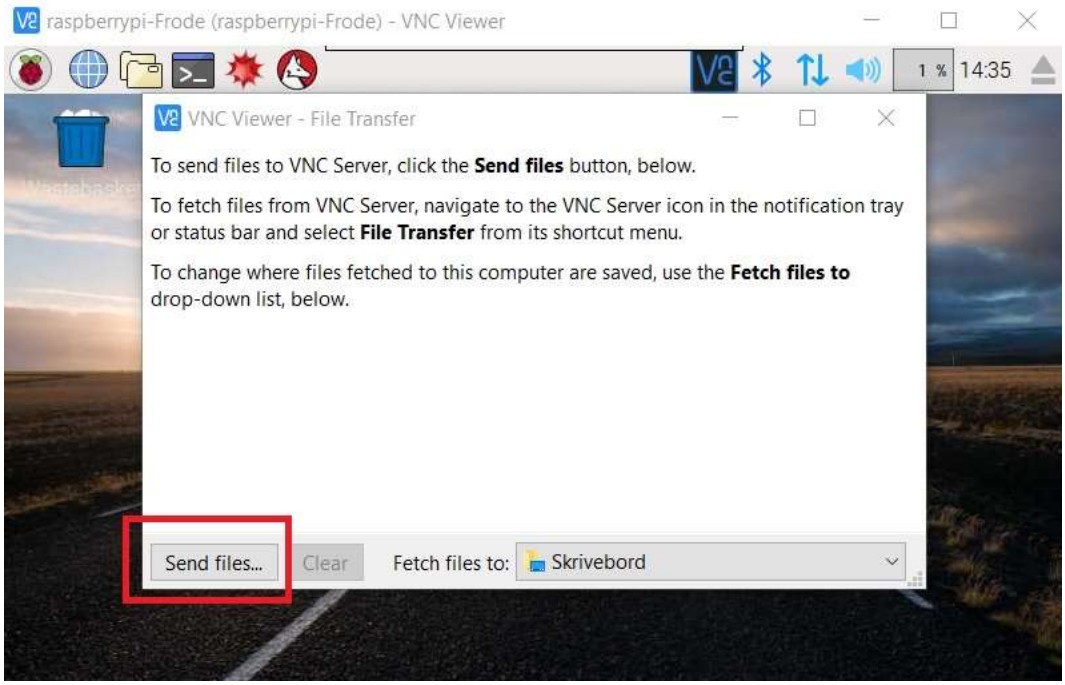


Figur 10

Neste steg er å overføre Python-koden, templogger.py (eget vedlegg), som logger temperatur. Dette gjøres via Real VNC. Koble til Raspberry Pi-en fra en pc med VNC Viewer. Når du er inne på skrivebordet til Raspberry Pi-en fører du musepekeren opp på midten slik at VNC Viewer-fanen kommer ned. Der trykker du på overføringssymbolet (Figur).

Figur 11

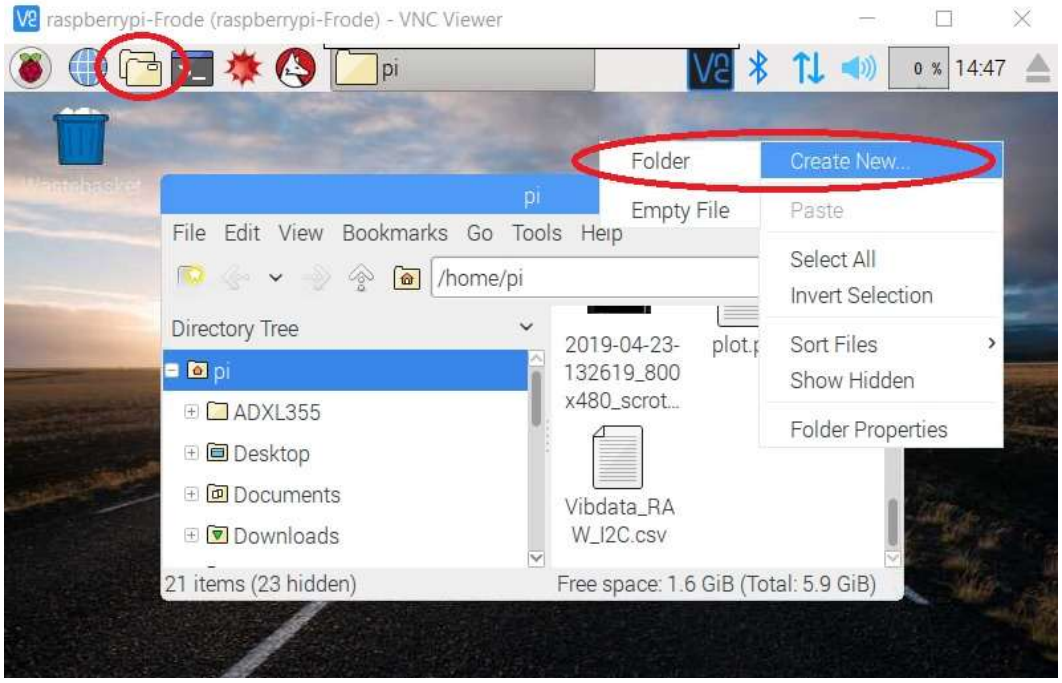
Da kommer det opp et overføringsvindu, trykk på **Send files…**(Figur).



Figur 12

Velg så templogger-filen fra pc-en din. Filen lagres på skrivebordet på Raspberry Pi-en. Det kan være lurt å lage en egen mappe på Raspberry Pi-en som du legger koden i. Temperaturloggen lagres på samme lokasjon som koden er lagret. Fil-systemet i Raspbian fungerer nesten likt som i Windows, så å lage en ny mappe fungere på nesten samme måte.

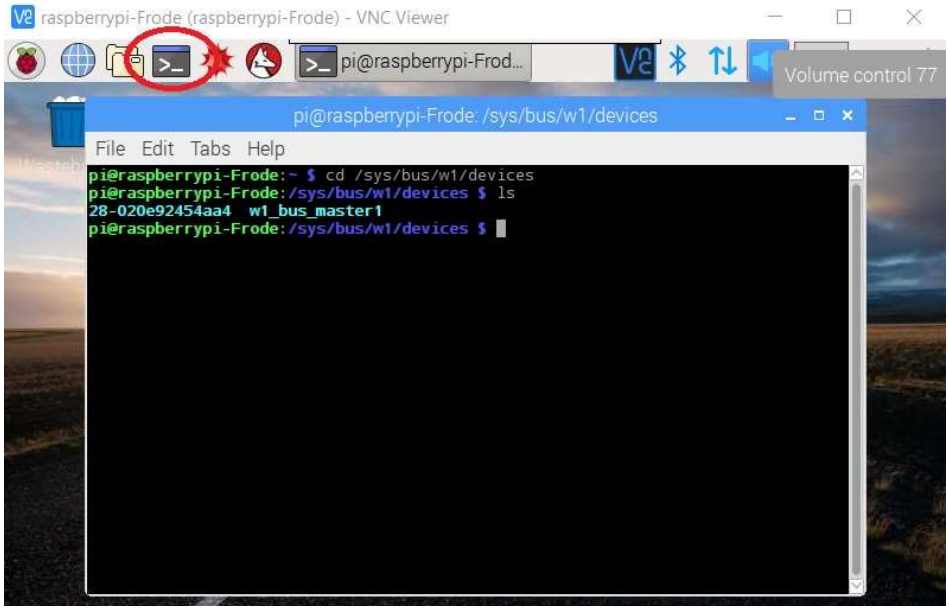
For å lage en mappe til kodefilen trykker du foldersymbolet oppe til venstre. Så navigerer deg til ønsket lokasjon og lager en ny mappe ved å høyre-klikke og velge **Create New.. > Folder** (Figur).



Figur 13

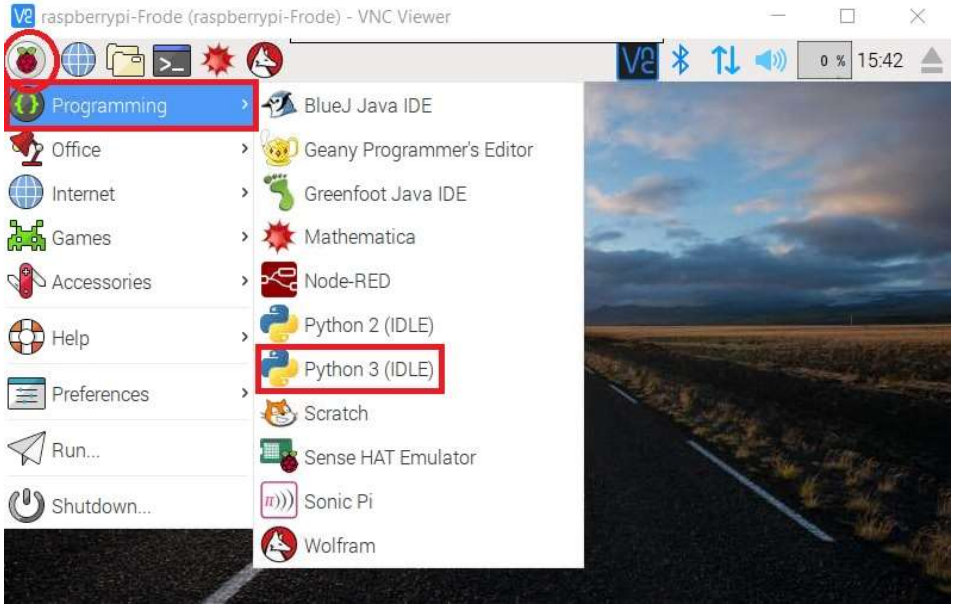
Navngi mappen og flytt templogger-filen hit. Det gjøres på samme måte som i Windows.

Skru av Raspberry Pi-en og koble til temperatursensoren DS18B20. Skru på Raspberry Pi-en igjen. Start med å sjekke at sensoren er koblet til og registreres. Åpne kommandovinduet (Figur) og skriv «**cd /sys/bus/w1/devices**», trykk enter. Skriv «**ls**» og trykk enter. Da skal serienummeret til sensoren, som begynner med 28-, vises på skjermen (Figur). Hvis sensoren ikke registreres, skru av Raspberry Pi-en og dobbeltsjekk at alle koblingene har kontakt og er koblet korrekt på GPIO-en.



Figur 14

Neste steg er å åpne et Python 3 IDLE (integrated development environment). Trykk **Meny > Programming >Python 3(IDLE)** (Figur).



Figur 15

Trykk **CTRL** + **O**, naviger deg frem til lokasjonen som templogger-filen ble lagret og åpne den. Før koden kjøres, må du velge loggetid og filnavn. I line 17 står variabelen for loggetid, «**log_period**». Bak denne skriver du inn ønsket loggetid i antall sekunder. I linje nr. 21 står variabelen for filnavn, «**file_name**». Her skrives ønsket filnavn mellom likhetstegnet og **{ :%Y_%m_%d}** (Figur).

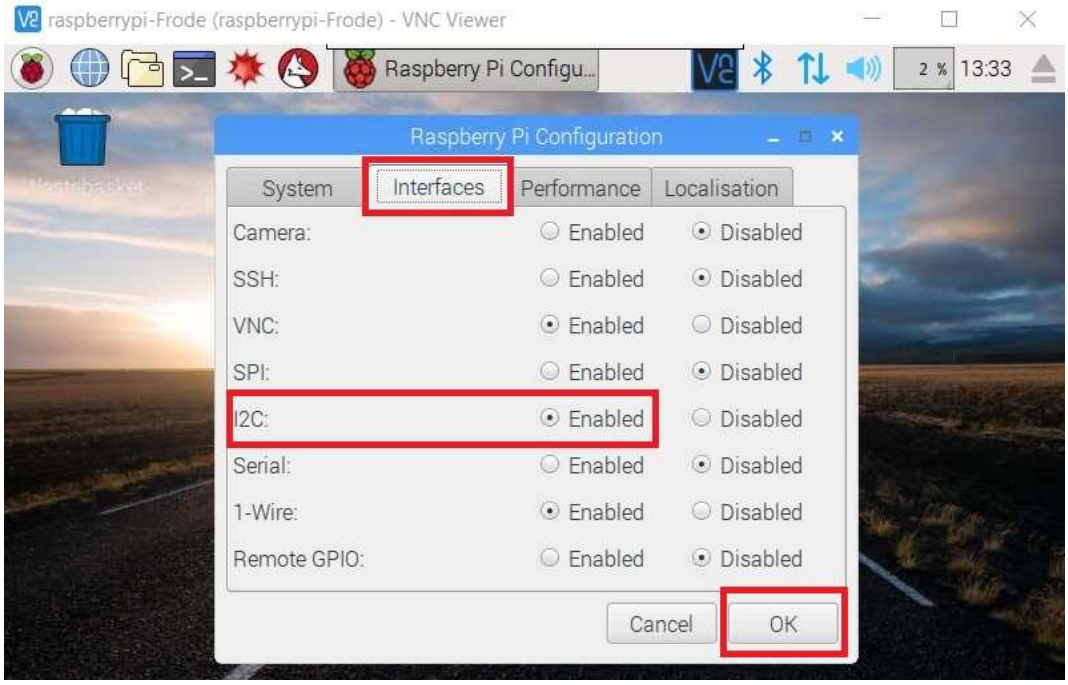
```
start= time.time()
log_period = 120    # seconds måletid

logging_folder = glob.glob('/home/pi/templogger*')[0]
dt = datetime.datetime.now()
file_name = "filnavn{:%Y_%m_%d}.csv".format(dt) #sett filnavn
logging_file = logging_folder + '/' + file_name
```

Figur 16

Når filnavn og loggetid er satt, trykk «**CTRL** + **S**» for å lagre og så trykk **F5** for å starte temperaturlogging. Loggefilen vil lagres som .csv i samme mappe som templogger-filen ligger.

7.4 Vibrasjonsmåling

Første steg for å kunne utføre vibrasjonsmålinger er å aktivere I^2C bussen på Raspberry Pi-en. Dette gjøres på samme måte som å aktivere 1-wire-bussen. Trykk **Meny > Preferences > Raspberry Pi Configuration > Interface** og velg **I^2C Enable** og trykk **OK** (Figur 6 og Figur).



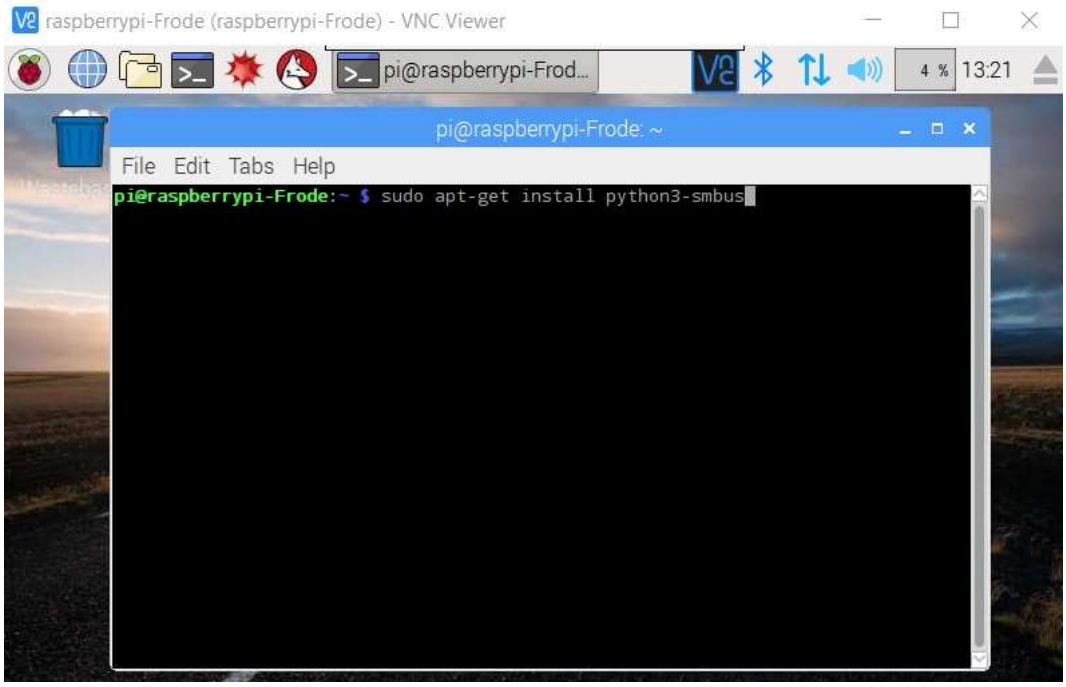
Figur 17

For at endringene skal ta effekt, kreves en omstart av Raspberry Pi-en. Det kommer automatisk opp et omstartsforslag, trykk **YES** (Figur 18).

Neste steg er å overføre Python-koden som kjører vibrasjonssensoren og loggingen, ADXL_355_I2C.py (eget vedlegg). Dette gjøres på samme måte som med templogger.py-filen. Etter at filen er lastet opp til skrivebordet på Raspberry Pi-en lagres den i en egen mappe på samme måte som templogger.py-filen.

Neste steg er å installere nødvendige moduler som benyttes av ADXL_355_I2C.py. Disse modulene er «SMBus», «Pandas» og Matplotlib». «SMBus» benyttes for å enkelt kommunisere med sensoren over I^{2}C-bussen. «Pandas» brukes for å lese .csv-filen for å lage en graf av målingen, og «Matplotlib» brukes for å plotte målingene.

For å installere modulene må du være koblet til internett. Åpne kommandovinduet og skriv «**sudo apt-get install python3-smbus**» trykk Enter. Modulen installeres, dette kan ta noe tid (Figur).

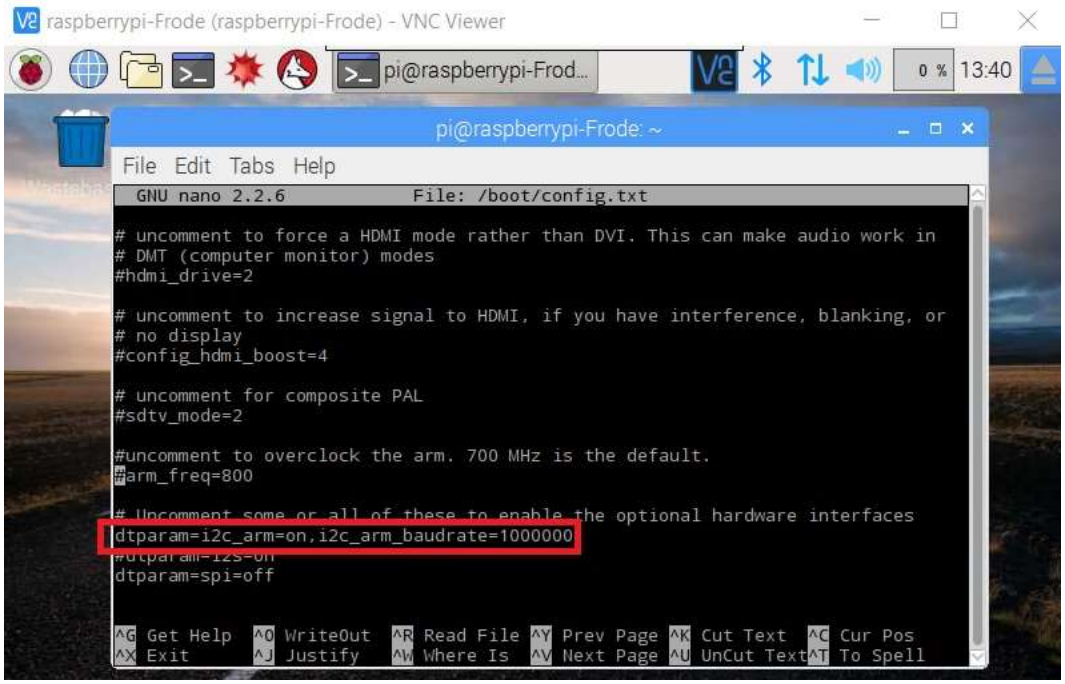


Figur 18

Fortsett på samme måte med alle kommandoene:

- **sudo apt-get install i2c-tools**
- **sudo apt-get install python3-pandas**
- **sudo apt-get install python3-matplotlib**

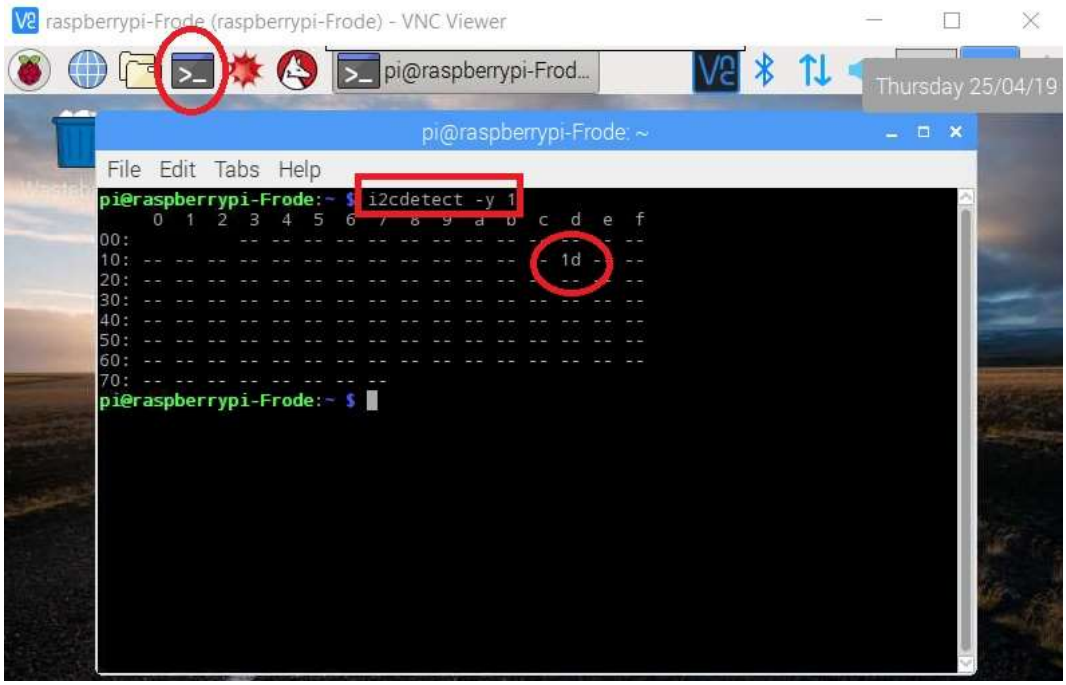
Neste steg er å sette I^{2}C bus-hastigheten opp til «Fast Mode-plus». Åpne kommandovinduet og skriv «**sudo nano /boot/config.txt**». Trykk Enter. Dette åpner oppstartsfilen. Bla deg nedover til du finner linjen «**dtparam=i2c_arm=on**». Bak denne skrives «**,i2c_arm_baudrate=1000000**» (Figur).



Figur 19

Trykk «**CRTL** + **X**» og **Y** og **Enter**. Restart Raspberry Pi-en for at endingene skal ha effekt. Dette kan gjøres ved å skrive «**sudo reboot**» og **Enter.**

Neste steg er å koble til sensoren. Skru av Raspberry Pi-en og koble til ADXL355-sensoren. Skru på Raspberry Pi-en. Start med å sjekke at sensoren har kontakt og fungerer. Åpne kommandovinduet og skriv «**i2cdetect -y 1**» trykk Enter. Da skal sensoren vises på adresse 0x1d (Figur).



Figur 20

Hvis sensoren ikke registreres eller registreres på en annen adresse, skru av Raspberry Pi-en og sjekk alle tilkoblinger.

Neste steg er å åpne et Python 3 «shell». Trykk **Meny > Programming >Python 3(IDLE)** (Figur). Trykk **CTRL** + **O**, naviger deg frem til lokasjonen som ADXL_355_I2C-filen ble lagret og åpne den. Før koden kjøres må du velge loggetid, filnavn og måleområde. I linje 9 står variabelen for loggetid, «**logge_tid»**. Skriv inn ønsket loggetid i sekunder. I linje 10 står variabelen for filnavn, «**filnavn»**. Her skriver du inn ønsket filnavn og lagringslokasjon (Figur).

```
logge_tid= 5   #i sekunder
filnavn = '/home/pi/ADXL355/Filnavn.csv' # sett filnavn og lagringslokalisjon
```

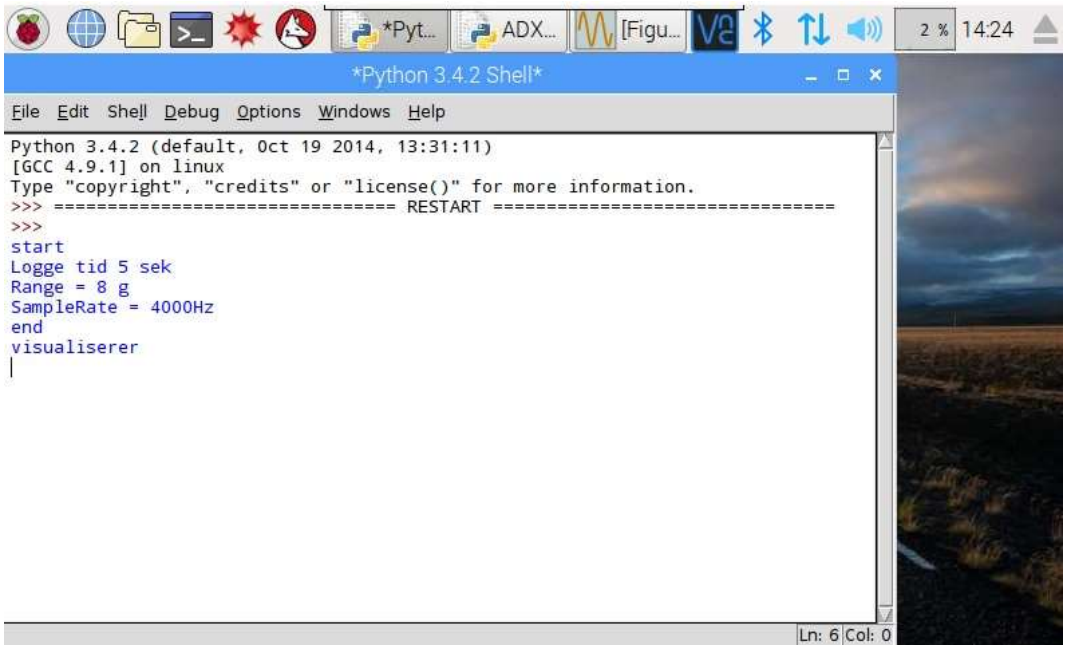
Figur 21

I linje 123 står variabelen for måleområde. Dette kan velges til ±2g, ±4g eller ±8g. Skriv inn ønsket måleområde innenfor parentesene. For +-2g skriv «**RANGE_2G**», for +-4g skriv «**RANGE_4G**», og for ±8g skriv «**RANGE_8G**» (Figur).

```
setRange(RANGE_2G) #RANGE_2G, RANGE_4G, RANGE_8G
```

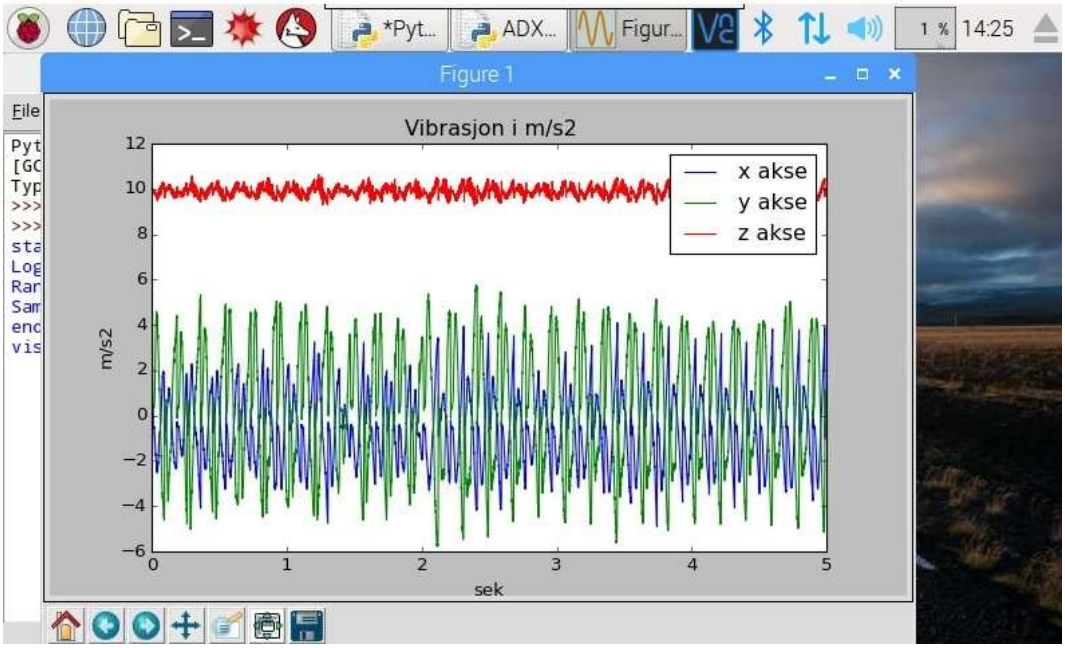
Figur 22

Når filnavn, loggetid, og måleområdet er satt trykk «**CTRL** + **S**» for å lagre og så trykk **F5** for å starte vibrasjonslogging. Logge-filen vil lagres som .csv på lokasjonen valgt i filnavn-variabelen. Når loggingen starter vises en oppsummering av valgte variabler (Figur).



Figur 23

Når vibrasjonsmålingen er utført, visualiseres signalet i en graf (Figur). Grafen gir en indikasjon på om loggingen har blitt utført korrekt. Grafen viser signalet for alle 3 aksene og signalet er representert i akselerasjon med enheten m/s^2.



Figur 24

8 Software-oversikt

Dette kapittelet vil beskrive flyten og oppbyggingen av softwaren som ble benyttet og utviklet for å lese og lagre data fra temperatur- og vibrasjonssensorene. I tillegg beskriver kapittelet den grunnleggende virkemåten til sensorene. Vi har benyttet Python 3 og «open source»-kilder for å sette sammen softwaren.

8.1 Temperatursensor- virkemåte

Temperatursensoren DS18b20 kommuniserer med Raspberry Pi ved hjelp av 1-wire bus. Det vil si en dataledning, spenningskilde og jord. Med 1-wire-protokollen vil Raspberry Pi-en operere som master og DS18b20 som slave. Masteren kontrollerer alltid bussen. Sensoren som er slave, vil kun sende signal etter at masteren har gitt klarsignal (Gay, 2018).
For å kjøre Raspberry Pi-en som master på 1-wire bussen benyttes to ferdiginstallerte drivere på Raspberry Pi-en. Den ene driveren heter W1-GPIO og den andre heter W1-term. Disse driverne leser rådata fra sensoren og definerer pinne 5 på GPIOen som datapinne (Gay, 2018). Dataen som leses fra sensoren lagres i en fil på Raspberry Pi-en. Dataene i filen oppdateres med en frekvens på ca. 1.2Hz. Python koden som benyttes må da bare aktivere driverne og leser rådata fra driverfilen. Det er ikke behov for å skrive en kode som behandler rådata i form av binære data.
Driverne lagrer avlesningene fra DS18b20 som en liste med to elementer, der element 0 er den øverste linjen i figur 1 og element 1 er den nederste linjen i figur 1.

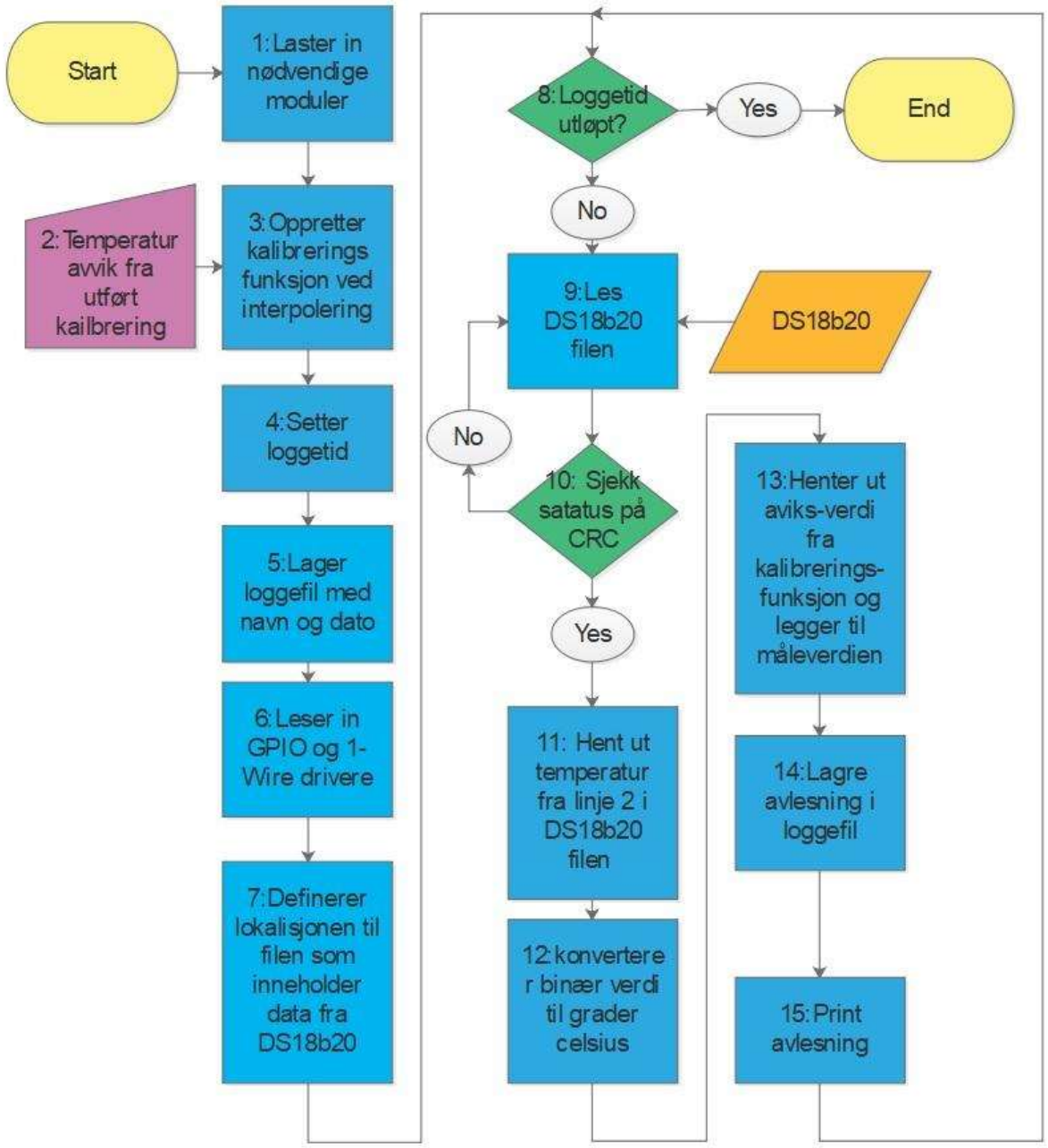
```
[0] ca 01 4b 46 7f ff 06 10 65 : crc=65 YES
[1] ca 01 4b 46 7f ff 06 10 65 t=28625
```

Figur 1, formatet på data som returneres fra w1-therm.

Temperaturen er oppgitt i tusendels celsius og står bak «**t=**» i linje 2 i figur 1. Driveren utfører også en CRC (cyclic redundancy check)-sjekk. CRC er en måte å sjekke om signalet har endret seg under overføring fra slave til master. Når det står «YES» i slutten av linje 1, betyr det at signalet er uforandret i overføringen fra slave til master. Dette er nyttig å vite for å forstå hvordan Python-koden henter ut temperatur fra driverfilen.

8.1.1 Programflyt

Figur 1 viser et flytskjema som beskriver hvilke operasjoner som gjøres i Python-koden for å hente ut temperatur data fra DS18b20, legge på kalibreringsavviket og lagre resultatet i en .csv-fil. Arbeidet for å skaffe kalibreringstallene beskrives i kap. 9.2.



Figur 2, temperatur, flytskjema

8.1.2 Beskrivelse av programmeringen

Som beskrevet i Resultatmål 3 skal vi her beskrive programmeringen. Vi har valgt å gå igjennom koden og beskrive betydningen til modulene, variablene og funksjonene i koden. Beskrivelsen av programmeringen ønskes, slik at en person uten annet enn grunnleggende programmeringskompetanse skal kunne følge de ulike stegene i programmeringen. Koden i sin helhet er vedlagt som eget vedlegg.

Som grunnlag har vi benyttet oss av en kode hentet fra «Raspberry Pi Cookbook» s.365 (Monk, 2016). Denne koden har så blitt tilpasset og utvidet til vårt behov.

Koden starter med innlasting av moduler, linje 1 til 5(figur3). Dette tilsvarer pkt.1 i figur2. En modul er et sett eller bibliotek med ferdigskrevede funksjoner.

```
1 import os, glob, datetime
2 import time
3 import sys
4 import numpy as np
5 from scipy import interpolate
```

Figur 3

os benyttes for å gjøre operasjoner i operativsystemet til Raspberry Pi-en. **globe** benyttes for å finne filer i filsystemet til Raspberry Pi-en, **datetime** benyttes for å finne dagens dato og klokkeslett. **Time** modulen benyttes til tidtaking. **Sys** benyttes til for å avslutte programmet. (Python Software Fundation, 2019)

Videre i linje 4 og 5 importeres **numpy** for å legge kalibreringsdata i matriser og **interpolate** fra **scipy** modulen for å lage kalibreringsfunksjonen. (Scipy.org, 2019)

I linje 7-12 legges verdier fra kalibrering inn og det produseres en kalibreringsfunksjon (Figur 3). Dette tilsvarer pkt.2 og pkt.3 i figur2.

```
 7 #kalibrerings verdier
 8 actual_temp=np.array([0.0,20.0,40.0,50.0,60.0,70.0,80.0,90.0,100.0])
 9 malt_differance=np.array([0.44,0.48,0.48,0.41,0.34,0.22,0.07,-0.14,-0.36])
10
11 #interpolert avviks funksjon
12 temp_interpolated=interpolate.interp1d(actual_temp,malt_differance,kind='cubic')
```

Figur 4

I linje 8 og 9 benyttes funksjonen **numpy.array** for å legge referanseverdiene fra kalibreringen (kap. 9.1) inn i variabelen **actual_temp** og **malt_differance**. **np.array** former verdiene som en 1-dimensjonal matrise.

I linje 12 (Figur 3) benyttes funksjonen **interpolate.interp1d** for lage en matematisk funksjon som returnerer et estimert avvik på bakgrunn av målt avvik. **Interp1d** returnerer det estimerte avviket ved å utføre en kubisk interpolasjon.

I linje 16 og 17 settes variabelen for logge-tid og start-tidspunkt for målingen defineres (Figur 4). Pkt.4 i figur 2

```
16 start= time.time()
17 log_period = 120# seconds måletid
```

Figur 5

For å definere start-tidspunkt for målingen benyttes **time.time** -funksjonen. **time.time()** returnerer antall sekunder som har gått siden 1. Januar 1970. Dette tallet benyttes som en startreferanse.

I linje 19-22 (figur 6) opprettes loggefilen som kalibrert måledata skal lagres i. Dette tilsvarer Pkt.5 i figur 2.

```
19 logging_folder = glob.glob('/home/pi/templogger*')[0]
20 dt = datetime.datetime.now()
21 file_name = "test1{:%Y_%m_%d}.csv".format(dt) #sett filnavn
22 logging_file = logging_folder + '/' + file_name
```

Figur 6

I linje 19 defineres lagringslokasjonen til loggefilen. I linje 20 leses dagens dato og klokkeslett fra Raspberry Pi-en. I linje 21 bestemmes navnet på loggefilen og dato legges inn i filnavnet. I linjen 22 legges fillokasjon og filnavn inn i en variabel.

I linje 24 til 29(figur7) åpnes driverne W1-gpio og W1-therm. I tillegg defineres navn og lokasjon på filen som inneholder «rå-data» fra sensoren. Dette er pkt.6 og pkt.7 i figur 2.

```
24 os.system('modprobe w1-gpio')
25 os.system('modprobe w1-therm')
26
27 base_dir = '/sys/bus/w1/devices/'
28 device_folder = glob.glob(base_dir + '28*')[0]
29 device_file = device_folder + '/w1_slave'
```

Figur 7

I linje 24 og 25 åpnes driverne w1-gpio og w1-therm med funksjonen **os.system()**. I linje 27 åpnes mappen som inneholder mappen til sensoren. I Linje 28 åpnes sensor-mappen og i linje 29 defineres hele adressen til filen som inneholder «rå-data» fra sensoren.

I linje 31 til 35(figur 8) står funksjonen **read_temp_raw()**, pkt.9 i figur 2. Denne funksjonen går inn på adressen hvor filen med rå-data ligger og leser innholdet. Innholdet som leses returneres i en variabel som kalles **lines**.

```
31 def read_temp_raw():
32     f = open(device_file, 'r')
33     lines = f.readlines()
34     f.close()
35     return lines
```

Figur 8

Linje 32 åpner sensor-filen som ble definert i linje 29. I linje 33 leses innholdet i sensor-filen og legger det inn i variabelen **lines**. **lines-**variabelen inneholder nå en liste i samme format som vist i figur 1. I linje 43 lukkes filen og i linje 35 returneres **lines**-variabelen.

I linje 37 til 48 (figur 9) står funksjonen **read_temp()**. Denne funksjonen står for pkt.10, 11, 12 og 13 i figur 2. **read_temp()** benytter **read_temp_raw()** for å lese innholdet i filen til DS18b20. Deretter kontrollerer den resultatet fra CRC-sjekken og finner tallet som representerer temperaturen i tusendels celsius. Regner så om fra tusendel celsius til celsius. Funksjonen benytter seg så av den avleste temperaturen for å hente ut kalibrering avviket, og legger til avviket på avlesningen og returnerer den kalibrert temperaturen.

```
37 def read_temp():
38     lines = read_temp_raw()
39     while lines[0].strip()[-3:] != 'YES':
40         time.sleep(0.2)
41         lines = read_temp_raw()
42     equals_pos = lines[1].find('t=')
43     if equals_pos != -1:
44         temp_string = lines[1][equals_pos+2:]
45         temp_c = float (temp_string) / 1000.0
46         temp_c_cal =round(temp_c+temp_interpolated(temp_c),2)
47 
48         return temp_c_cal
```

Figur 9

I linje 38 hentes data fra filen til DS18b20 ved hjelp av **read_temp_raw()** og legges inn i variabelen **lines** i formatet som i figur 1. I line 39 starter **while**-løkken som ser om CRC-sjekken er ok. Her hentes element 0 ut fra **Lines,** dvs. den øverste linjen i figur 2. Med **stip()**-funksjonen fjernes alle mellomrom fra linjen for så å velge ut de tre siste tegnene i linjen med **[-3:]**. Hvis disse er ulik (**!=**) **YES**, dvs. at CRC-sjekken ikke var vellykket, går løkken videre til linje 40. Her legges inn en pause på 0,2 sekund med **time.sleep()**. Så leses DS18b20 filen på nytt og linje 39 gjentas. Denne while-løkken gjentas helt til CRC-sjekken er ok. Da går programmet ned til linje 42.

I linje 42 hentes element 1 ut fra **lines** listen(linje 2 i figur1) og posisjonen i linjen som har **t=** som de to neste tegnene. Denne posisjonen legges inn i variabelen **equals_pos**. I linje 43 sjekker koden at det er mer enn ett tegn etter posisjonen **equals_pos**. I linje 44 hentes tallene, som tilsvarer temperaturen, ut fra element en(**[1]**) i **lines** listen med å lese fra **equals_pos** pluss to tegn. Temperatur-tallet legges inn i variabelen **temp_string**. I linje 45 deles temperatur-tallet, som er i tusendels celsius, på tusen for å få celsius og legges inn i **temp_c**-variabelen. I linje 46 kalibreres temperaturen ved å sette den avleste temperaturen inn i kalibreringsfunksjonen fra linje 12, **temp_interpolated**. Så adderes verdien fra kalibreringsfunksjonen til den avleste temperaturen. I linje 48 returneres den kalibrerte temperaturen.

I linje 50 til 56 (figur 10) står funksjonen **log_temp** som lagrer kalibrert temperatur i loggefilen. Dette tilsvarer pkt. 14 i figur 2.

```
50 def log_temp():
51     temp_c_cal = read_temp()
52     dt = datetime.datetime.now()
53     f = open(logging_file, 'a')
54     f.write('\n"{:%H:%M:%S}",'.format(dt))
55     f.write(str(temp_c_cal))
56     f.close()
```

Figur 10

I linje 51 legges kalibrert temperatur inn i variabelen **temp_c_cal** ved å bruke funksjonen fra linje 37(figur9) **read_temp**. I linje 52 leses klokkeslettet fra Raspberry Pi-en sin klokke med funksjonen **datetime.datetime.now** og legges i variabelen **dt**. I linje 53 åpnes loggefilen, som ble definert i linje 22 figur 6, med rettigheten «append» (**'a'**) og legges inn i variabelen **f**. I linje 54 benyttes **write**-funksjonen for å skrive i **f**. Det skrives først et linjeskift(**\n**) så tidspunktet i formatet time, minutt, sekund (**{:%H:%M:%S}**) som hentes fra **dt** med **format(dt)**. I linje 55 skrives den kalibrerte temperaturen inn, og i linje 56 lukkes filen **f** med **close()**-funksjonen.

I linje 61 til 63(figur 11) står hovedløkken som kjører programmet til loggetiden er utløpt. Dette tilsvarer pkt.8 i figur 2.

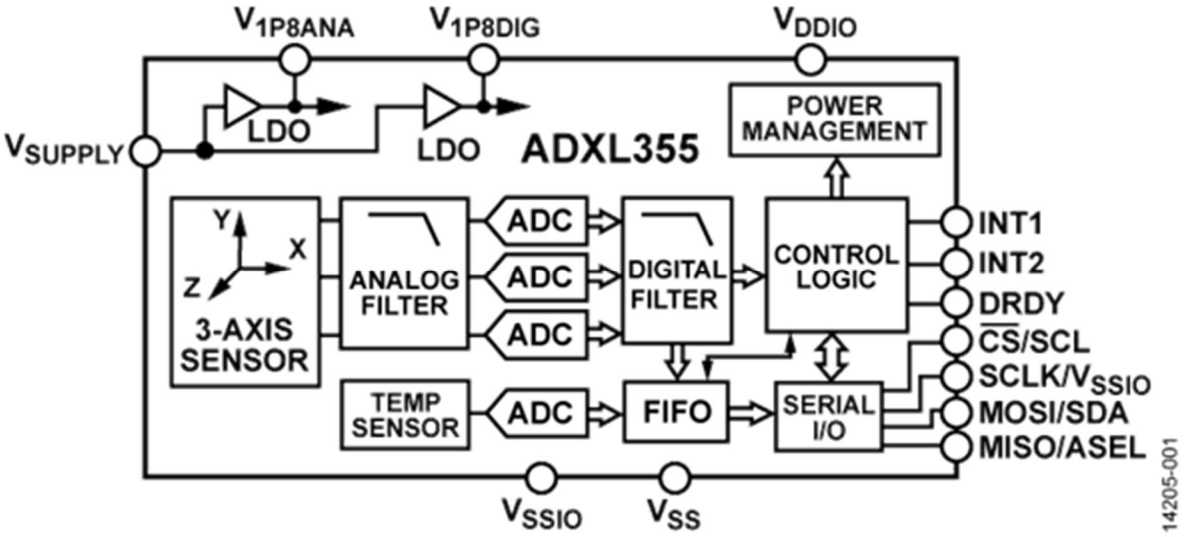
```
61 while time.time()<start+log_period:
62     log_temp()
63     print("temp C=", read_temp())
64
65
66 sys.exit()
```

Figur 11

I linje 61 undersøkes det om nåværende tidspunkt(**time.time()**) er mindre en start-tidspunktet (**start** i linje 16, figur 5) pluss loggetiden (**log_periode** i linje17, figur 5). Hvis dette er sant utføres linje 62 og 63 for så å gå opp i linje 61 igjen og sjekker om uttrykket fremdeles er sant. I linje 62 kjøres **log_temp()**-funksjonen fra linje 50, figur 10. I linje 63 skrives resultatet fra funksjonen **read_temp()**, som står i linje 37, figur 9, i konsollen. Når uttrykket i linje 61 ikke er sant går koden til linje 66 og avslutter koden med funksjonen **sys.exit()**.

8.2 Vibrasjonssensor virkemåte

Her beskrives en virkemåte som er relevant for å kunne følge gjennomgangen av Python-koden som ble utviklet for å lese akselerasjon fra ADXL355.
ADXL355 er et digitalt 3-akset kapasitive MEMS-akselerometer. Selve målingen av vibrasjonssignal er analogt. Etter at akselerometeret har målt vibrasjon på de tre aksene, benyttes et analogt anti-aliasing lavpassfilter på 1,5kHz. Deretter går signalet inn i de innebygde ADC-ene. ADXL355 har en egen kanal for hver av aksene for å minske støy og forvrenging av signalet. Etter digital konvertering av signalet, påføres signalet et programmerbart båndpassfilter. Så benyttes den innebygde kontroll-logikken for å gjøre avlesningen tilgjengelig på de serielle utgangene. Ved hjelp av de serielle inngangene og kontroll-logikken, programmeres innstillingene på båndpassfilteret, ODR (output data rate) og måleområdet. Figur12 viser et blokkdiagram av innsiden til ADXL355 chipen. (vedlegg 4)



Figur 12 ADXL355 Blokk Diagram

For å kunne hente ut data fra de serielle inngangene og utgangene, benyttet vi I^2C-kommunikasjonsprotokollen. I^2C bussen benytter seg av 2 datalinjer, SCL (serial clock) og SDA (serial data). I^2C-bussen ble designet for å ha en master og mulighet for flere slaver på samme bus. Derfor har alle slavene på bussen en egen 7-bits adresse (Gay, 2018). I dette tilfellet er det kun en slave, ADXL355, men det er allikevel nødvendig å benytte seg av sensorens egen adresse når det skal kommuniseres mellom Raspberry Pi og ADXL355.

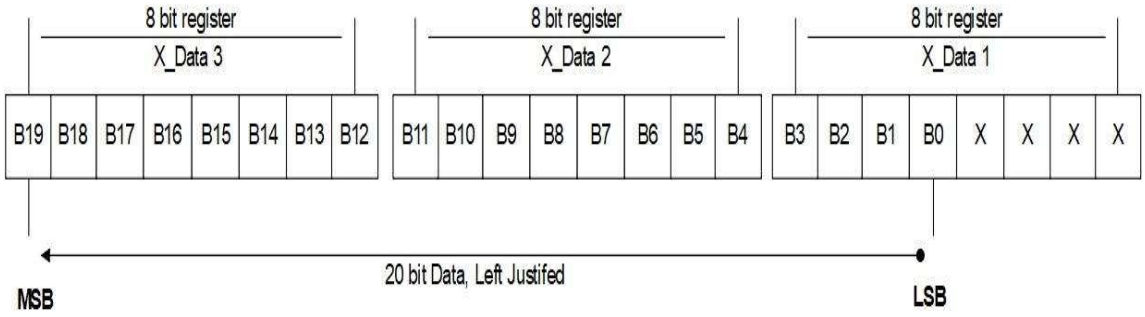
For å lese fra eller skrive til ADXL355 benyttes det et sett med register som ligger i kontroll-logikken til ADXL355. Hvert register er på 8 bit og har egen adresse på 8 bit. Noen register kan bare leses, som for eksempel registrene som inneholder akselerasjonsdata. Mens andre kan både leses og skrives ny informasjon inn i, som registeret til filtrene. Adressene til registrene står oppgitt i databladet til ADXL355(vedlegg 4).

For å kommunisere, sender masteren ut et unikt start-signal etterfulgt av adressen til slaven. Etter adressen til slaven, kommer adressen til det registeret som skal leses fra eller skrives inn i. Skal det leses tar slaven kontroll over bussen og sender informasjonen i det oppgitte registeret tilbake. Skal det skrives inn i det oppgitte registeret, kommer informasjonen som skal skrives etter register-adressen. (Gay, 2018)

For å lese og skrive i registeret til ADXL355 med Raspberry Pi, er det benyttet Python og **SMBus** modulen. **SMBus** gjør det enkelt å kommunisere. For å skrive til data til ADXL355 benyttes funksjonen **write_byte_data (**slave adresse, register adresse, data**)** fra **SMBus** modulen. For å lese, benyttes **read_byte_data(**slave adresse, register adresse), og innholdet returneres i den oppgitte register-adressen.

Slave-adresser, register-adresser og data overføres som binære tall. For å slippe å skrive verdier og adresse i Python som binære tall, oversetter Python automatisk heltall og hexadesimale tall til binære tall. Det vil si at en register-adresse som for eksempel er 1111 0000 binært kan skrives som 240(desimal) eller F0(heksadesimal). I praksis brukes heksadesimal. For at Python skal forstå at du skriver et heksadesimal tall, må tallet starte med **0x**, for eksempel **0xF0**. Om du allikevel ønsker å skrive et tall binært i Python, må du starte tallet med **0b,** for eksempel **0b11110000**.

Oppløsningen til ADXL355 er på 20 bit. For å kunne lagre informasjon på 20 bit ved hjelp av 8 bits register, bruker ADXL355 3stk 8 bits register per akse. Akselerasjonsdataene lagres i registrene i et format som kalles «left justified». Det vil si at det binære tallet som består av 20 bit legges mot venstre i registrene med MSB (most significant bit) til venstre(figur13).



Figur 13, Data formatert som Left justified

Måleområdet til akselerometeret er ±2g, ±4g eller ±8g. For å kunne representere negative akselerasjonsdata med binære tall, er vibrasjonsverdier formatert som «two's complement». Det vil si at når bit 19 er høy (har verdi =1), fungerer den som en negativ verdi som legges til verdien som står i bit0 til bit18. I figur14 vises et eksempel på hvordan «two's complement» vil fungere med 4 bit. Da vil MSB (most significant bit) representere verdien -8.

Biner verdi	Desimal verdi	Two's Complement
0000	0	0+0=0
0100	4	0+4=4
0111	7	0+7=7
1000	8	-8+0=-8
1001	9	-8+1=-7
1100	12	-8+4=-4
1111	15	-8+7=-1

Figur 14, Two's complement

8.2.1 Innstillinger

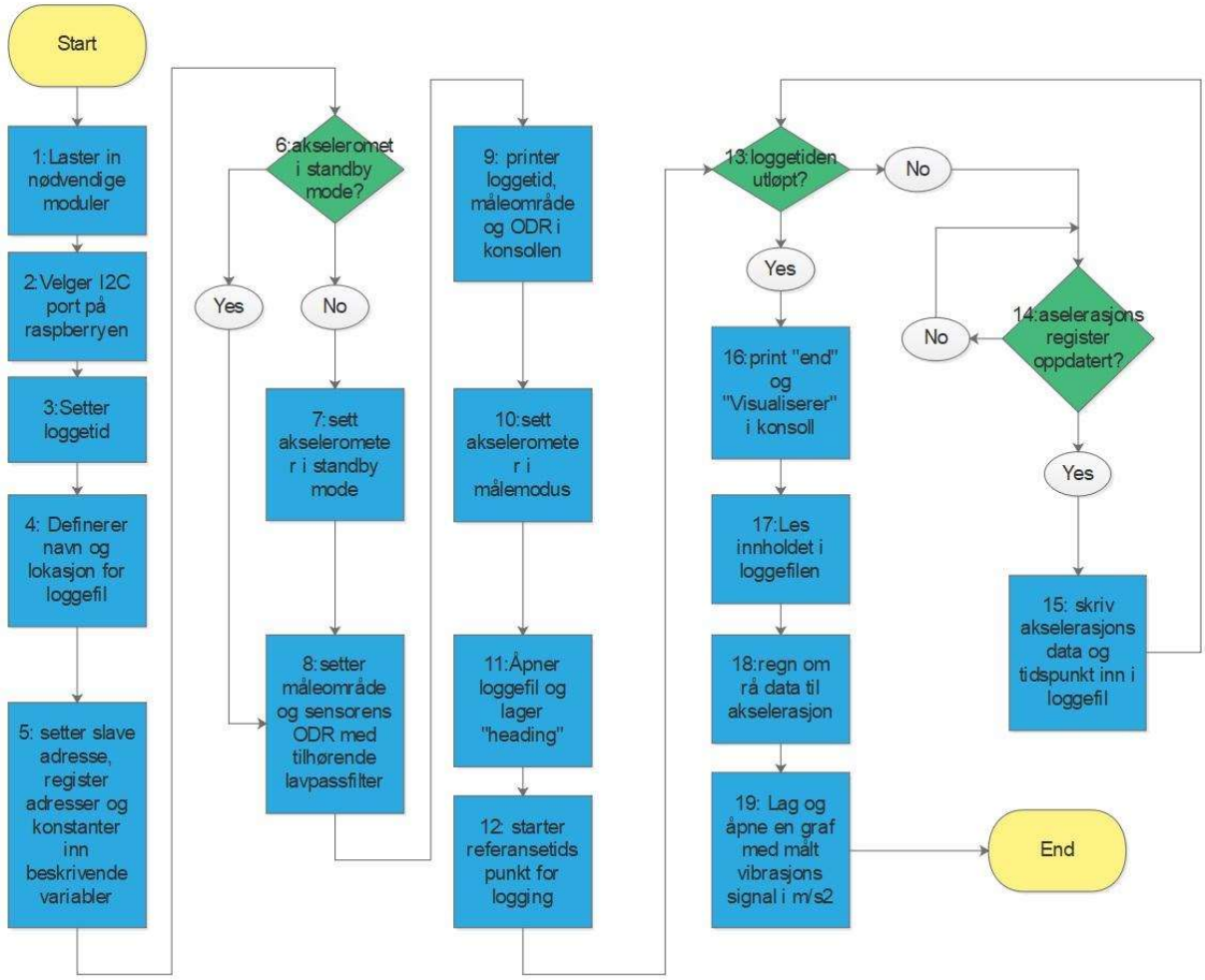
ADXL355 har forskjellige funksjoner og innstilinger som kan benyttes for å kontrollere sensoren. Her er det benyttet innstillinger for ODR (output data rate), lavpass-filter og måleområde. Det er også benyttet innebygde funksjoner for «data ready» og «power mode»
Innstillingen for ODR, lavpass-filter og høypassfilter skrives inn i filterregisteret med adressa **0x28**. Her er det bit 0 til 3 som justerer ODR og lavpassfilteret. ODR og lavpassfilter kan ikke velges individuelt. Så for å ha lavpassfilter på 1000Hz settes ODR automatisk til 4000Hz. I bit 4 til 6 settes høypassfilteret.

Innstillinger for måleområdet gjøres i RANGE-registeret med adresse **0x2c**. Her er det bit 0 og 1 som representerer måleområdet. Vi har ikke benyttet oss av de andre innstillingene i dette registeret, så for å ikke påvirke bit 2 til 7 benyttes en maskeringskonstant i koden (**RANGE_MASK** = **0x03** = 00000011).

Funksjonen «data ready» i ADXL355 indikerer når det er nye data tilgjengelig i akselerasjonsregistrene. Data ready-biten er settes høy når det er ny data tilgjengelig. Denne biten står i STATUS-registeret med adresse **0x04**. Data ready er bit 0 i registeret. Derfor benyttes maskeringsbyte på **0x01** for å kun hente ut data ready-biten.
Det samme gjelder for Power control-registeret med adresse **0x2D**. Her er det bit 0 som representerer standby mode av eller på.

8.2.2 Programflyt

Figur 15 viser ett flytskjema som beskriver hvilke operasjoner som gjøres i Python-koden for å logge vibrasjonsdata fra ADXL355 og lagre resultatet i en .csv-fil.



Figur 15

8.2.3 Beskrivelse av programmeringen

Som beskrevet i Resultatmål 3, skal vi her beskrive programmeringen. Vi har valgt å gå igjennom koden, og beskrive betydningen til modulene, variablene og funksjonene i koden. Beskrivelsen av programmeringen ønskes slik at en person uten annet enn grunnleggende programmeringskompetanse skal kunne følge de ulike stegene i programmeringen. Det antas at leseren kjenner til de logiske funksjonene AND og OR. Koden i sin helhet er vedlagt som eget vedlegg. Som grunnlag har gruppen benyttet seg av en kode hentet fra Github med open source-lisens (Radbourne, 2017). Denne koden har så blitt tilpasset og utvidet til vårt behov.
Koden starter med innlasting av nødvendige moduler i linje1 til 6 i figur 16. Dette tilsvarer pkt.1 i figur 15.

```
1 import time
2 import os
3 import sys
4 import pandas as pd
5 from matplotlib import pyplot as plt
6 import smbus
```

Figur 16

Time-modulen benyttes til tidtaking. **os** benyttes for å gjøre operasjoner i operativsystemet til Raspberry Pi-en. **Sys** benyttes til å avslutte programmet. (Python Software Fundation, 2019) **pandas** importeres som forkortelsen **pd** og fra **matplotlip** importeres **pyplot** som forkortelsen **plt** (Scipy.org, 2019). Disse brukes for å lese loggefilen etter fullført logging, for så å lage en graf i tidsbildet av det målte signalet.
I linje 7 til 10 i figur 17 utføres pkt.2 og pkt.3 i figur 15.

```
 7 bus=smbus.SMBus(1)
 8
 9 logge_tid= 2 #i sekunder
10 filnavn = '/home/pi/ADXL355/vifte_3.csv' # sett filnavn og lagringslokalisjon
```

Figur 17

I linje 7 settes **smbus.SMBus** til 1. Det vil si at **SMBus** skal benytte seg av Raspberry Pi sin I^2C port nr 1. Det er pinne 3 og 5 på GPIO-en. Definisjonen på hvilken I^2C-port som skal brukes, settes inn i variabelen **bus**. I linje 9 settes loggetiden i antall sekund, og legges inn i variabelen **logge_tid**. I Linje 10 defineres lokasjonen og navnet til loggefilen, og legges inn i variabelen **filnavn.**

I linje 12 til 42 flyttes slave-adresse, register-adresser og innstillingsverdier inn i beskrivende variabelt navn. (figur 18) Dette tilsvarer pkt.5 i figur 15.

```
12 #ADXL355 Adresser og konstanter
13 i2c_addr =0x1d                          #adressen til ADXL355
14 POWER_CTL=0x2d
15 POWER_CTL_OFF        = 0x01
16 POWER_CTL_ON         = ~POWER_CTL_OFF
17
18 RANGE                = 0x2C             #måle område register
19 RANGE_MASK           = 0x03
20 RANGE_2G             = 0b01
21 RANGE_4G             = 0b10
22 RANGE_8G             = 0b11
23
24 LOWPASS_FILTER       = 0x28      #filter og ODR register
25 LOWPASS_FILTER_MASK  = 0x0F
26 ODR_4000=0b000
27
28 XDATA3               = 0x08      #akselerasjons data register
29 XDATA2               = 0x09
30 XDATA1               = 0x0A
31 YDATA3               = 0x0B
32 YDATA2               = 0x0C
33 YDATA1               = 0x0D
34 ZDATA3               = 0x0E
35 ZDATA2               = 0x0F
36 ZDATA1               = 0x10
37
38 AXIS_START           = XDATA3
39 AXIS_LENGTH          = 9
40
41 STATUS                = 0x04      #status register
42 STATUS_MASK_DATARDY  = 0x01
```

Figur 18

I linje 46 til 58 i figur 19 står funksjonene som leser «power control»-registeret (**isRunning)**, setter akselerometeret i standby-modus(**end**) og setter akselerometeret i målemodus(**begin**). Disse funksjonene benyttes i pkt.6, pkt.7 og pkt.10 i figur 15.

```
46 def isRunning():
47         powerCtl = bus.read_byte_data(i2c_addr, POWER_CTL)
48         return (powerCtl & POWER_CTL_OFF) != POWER_CTL_OFF
49
50 def begin():
51         powerCtl = bus.read_byte_data(i2c_addr, POWER_CTL)
52         if (powerCtl & POWER_CTL_OFF) == POWER_CTL_OFF:
53             bus.write_byte_data(i2c_addr, POWER_CTL, (powerCtl & POWER_CTL_ON))
54
55 def end():
56     powerCtl = bus.read_byte_data(i2c_addr, POWER_CTL)
57     if (powerCtl & POWER_CTL_OFF) != POWER_CTL_OFF:
58         bus.write_byte_data(i2c_addr, POWER_CTL, powerCtl | POWER_CTL_OFF)
```

Figur 19

I linje 47 leses power control-registeret og innholdet legges inn i **powerCTL-**variabelen. I linje 48 utføres det en logisk AND funksjon (**&**) mellom bitene i **powerCTL-**variabelen og **POWER_CTL_OFF**-variabelen. Utfallet her blir enten 1(0x01) eller 0(0x00) siden **POWER_CTL_OFF** = 0x01**.** Utfallet returneres som **true** (1) eller **false** (0).

begin funksjonen i linje 50, setter adxl355 i målemodus. I linje 51 leses power control-registeret og innholdet legges inn i **powerCTL-**variabelen. I linje 52 utføres det en logisk AND funksjon (**&**) mellom bitene i **powerCTL-**variabelen og **POWER_CTL_OFF-**variabelen. Så undersøkes det om utfallet her er identisk med **POWER_CTL_OFF-**variabelen. Hvis det er tilfelle, settes akselerometeret i målemodus i linje53. Verdien som skrives inn i power control, er resultatet av en AND funksjon mellom **powerCTL** og **POWER_CTL_ON**. Da endres kun LSB i power control-registeret.

I **end-**funksjonen i linje 55 utføres på nesten samme måte som **begin**. Forskjellen er at i linje 57 må utfallet av AND-operasjonen mellom **powerCTL** og **POWER_CTL_OFF** være ulik fra **POWER_CTL_OFF** for at linje 58 skal utføres. I linje 58 benyttes det en OR-funksjon (**|**) mellom **powerCTL** og **POWER_CTL_OFF** for å kun endre på LSB i power control-registeret.

I linje 60 til 73 i figur 20 står funksjonen **getRange** og **setRange**. **getRange** leser hvilket måleområde som ADXL355 er innstilt på og **setRange** stiller ADXL355 in på ønsket måleområde. **getRange** benyttes i pkt.18 og **setRange** benyttes i pkt.8 i figur15.

```
60 def getRange():
61     return (bus.read_byte_data(i2c_addr, RANGE)) & RANGE_MASK
62
63 def setRange(newRange): #newRange =RANGE_2G, RANGE_4G or RANGE_8G
64     if type(newRange) is not int:
65         raise ValueError('newRange must be an integer')
66
67     if newRange < RANGE_2G or newRange > RANGE_8G:
68         raise ValueError('newRange is out of range')
69
70     else:
71         Range = bus.read_byte_data(i2c_addr, RANGE)
72         Range = (Range & ~RANGE_MASK) | newRange
73         bus.write_byte_data(i2c_addr, RANGE, Range)
```

Figur 20

I linje 61 under funksjonen **getRange()** leses data fra range-registeret til ADXL355 og det utføres en AND-operasjon med **RANGE_MASK**. I dette tilfellet returnerer AND-operasjonen en byte med bit [2:7] lik 0 og bit [0:1] med korresponderende binær verdi for måleområdet som ADXL355 står innstilt på. Den sistnevnte byten returneres til variabelen som kaller på **getRange()**-funksjonen.

I linje63 står **setRange(**newRange**)**-funksjonen. I linje 64 undersøkes det om innputten i funksjonen er et heltall. Hvis innputten ikke er et heltall avsluttes koden med feilmeldingen «newRange must be an integer» I linje 67 undersøkes det om innputten er en av de korresponderende binære verdiene til de mulige måleområdene. Hvis ikke, avsluttes koden med feilmeldingen «newRange is out of range» Hvis ingen av feilmeldingene aktiveres, utføres linje 71 til 73. I linje 71 leses range-registeret til ADXL355 og legges inn i **range** variabelen. I linje 72 utføres det en AND-operasjon mellom innholdet i registeret og den inverse (**~**) verdien av **RANGE_MASK**. Dette gir en byte med bit 2 til 7 identisk med bit 2 til 7 i range-registeret og bit 0 til 1 blir 0. Så utføres det en OR-funksjon mellom den nye byten og byten som setter måleområdet. Siden byten som setter måleområdet er 0 i bit 2 til 7 endres kun bit 0 og 1 i range-registeret.

I linje 75 til 82 i figur 21 står funksjonene **setLowpassFilter** og **getLowpassFilter**. Disse funksjonene benyttes for å lese filter-registeret og sette ODR med tilhørende lavpassfilter. Disse funksjonene benyttes i pkt.8 og 9 i figur 15.

```
75 def setLowpassFilter(newLowpassFilter): #ODR_4000 = 0b000
76         if type(newLowpassFilter) is not int:
77             raise ValueError('newLowpassFilter must be an integer')
78         lowpassFilter = newLowpassFilter
79         bus.write_byte_data(i2c_addr, LOWPASS_FILTER, lowpassFilter)
80 
81 def getLowpassFilter():
82         return (bus.read_byte_data(i2c_addr, LOWPASS_FILTER)) & LOWPASS_FILTER_MASK
```

Figur 21

setLowpassFilter i linje 75 begynner på samme måte som **setRange** i linje 63 med å sjekke om innputten i funksjonen er et heltall. I linje 78 og 79 skrives verdien for 4000Hz ODR og 1000Hz lavpassfilter rett inn i registeret. Her er det ikke laget en oversiktlig metode i koden for å velge alle mulige innstilinger. Det er fordi gruppen anså det som usannsynlig å ha en ODR på 2000Hz med tilhørende lavpassfilter på 500 Hz i forbindelse med tilstandsovervåkning. Men om man ønsker å sette ODR og lavpassfilter på andre innstillinger, kan man hente register-verdien fra ADXL355-databladet og sette inn som input.

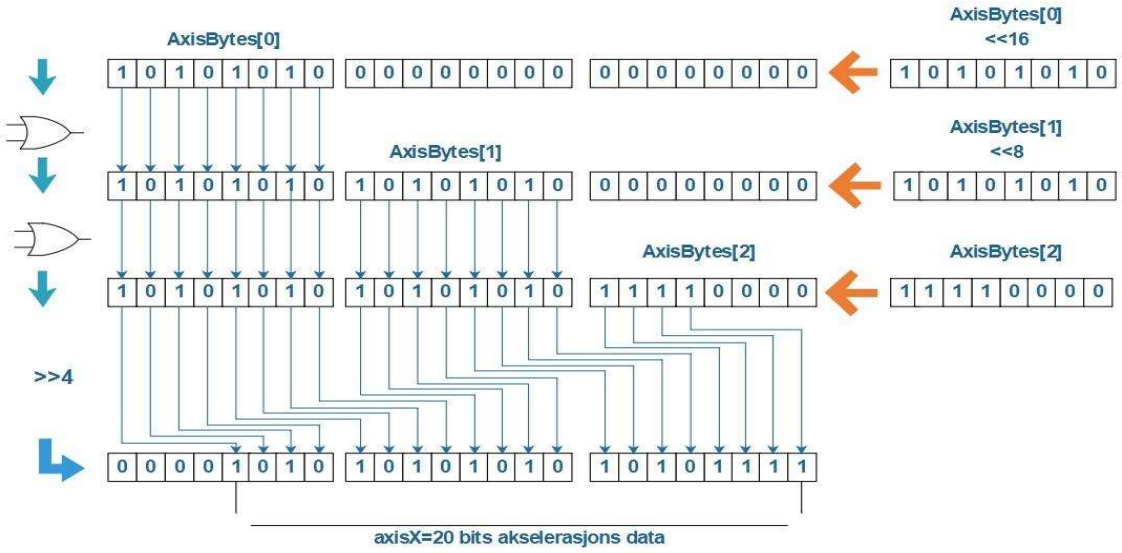
I linje 81 leses innstillingen på ODR og Lavpassfilteret. Dette gjøres ved å gjennomføre en AND-funksjon mellom innholdet i registeret og **LOWPAS_FILTER_MASK**. Da returneres en byte men bit 4 til 7 lik 0 og bit 0 til 3 med ODR og lavpassfilter-innstillingen.

I linje 84 til 100 i figur 22 står den mest sentrale funksjonen i koden. Funksjonen **getAxes()** leser akselerasjonsdataene og returnerer dataene i et bibliotek med labels x, y, og z for de korresponderende aksene. **getAxes()** benyttes i pkt.15 i figur 15.

```
84 def getAxes():
85         axisBytes = bus.read_i2c_block_data(i2c_addr, AXIS_START, AXIS_LENGTH)
86 
87         axisX = (axisBytes[0] << 16 | axisBytes[1] << 8 | axisBytes[2]) >> 4
88         axisY = (axisBytes[3] << 16 | axisBytes[4] << 8 | axisBytes[5]) >> 4
89         axisZ = (axisBytes[6] << 16 | axisBytes[7] << 8 | axisBytes[8]) >> 4
90 
91         if(axisX & (1 << 20 - 1)):
92             axisX = axisX - (1 << 20)
93 
94         if(axisY & (1 << 20 - 1)):
95             axisY = axisY - (1 << 20)
96 
97         if(axisZ & (1 << 20 - 1)):
98             axisZ = axisZ - (1 << 20)
99 
100        return {'x': axisX, 'y': axisY, 'z': axisZ}
```

Figur 22

I linje 85 uføres det en blokk-lesing av register i ADXL355. Det vil si at det første registeret som leses, har adressen som representeres av variabelen **AXIS_START**. Så leses registeradresser i stigende rekkefølge. Antall register som skal leses defineres av variabelen **AXIS_LENGTH**. Her leses det totalt 9 register, 3 for hver akse. Disse 9 bytene legges som en liste inn i variabelen **axisBytes**. I linje 87 til 89 legges dataene inn i variabler for hver akse. Byten med de høyeste verdiene forskyves 16 bit (**<<16**) til venstre, den mellomste byten forskyves 8bit (**<<8)** til venstre. Så utføres det en OR-funksjon (**|**) mellom dataene for å slå de sammen til et datasett på 3 byte. Deretter flyttes hele datasettet 4 bit til høyre for å fjerne de 4 ugyldige bittene. Disse operasjonene er illustrert i figur 23.



Figur 23 Prinsipp for sammenslåing av vibrasjonsdata

I linje 91 til 98 i figur 22 utføres «two's complement» på akselerasjonsdataene, en gang for hver akse. Hvis(**if**) en AND-funksjon (**&**) mellom akselerasjon dataen(**axisX**) og et 20 bits tall med MSB på 1 og resterende bits lik null (**1<<20-1**) er ulik 0(**true**) er verdien i **axisX** en negativ akselerasjonsverdi. Og da utføres det «two's complement» på dataen og den nye negative verdien flyttes inn i **axisX**.

I linje 102 til 116 i figur24 står funksjonene **getStatus()** og **lsb_g()**. **getStatus()** benyttes ipkt.14 i figur 15. Den leser status-registeret og returnerer bit 0, som angir om det er ny akselerasjonsdata tilgjengelig. **lsb_g()** undersøker hvilket måleområde som ADXL355 er innstilt på og returnerer måleområder som variabelen **g_range** og returnerer antall linjer per g i variabelen **lsb_g**. **g_range** benyttes i pkt.9 i figur 15 og **lsb_g** benyttes i pkt.18 i figur15.

```
102 def getStatus():
103         status = bus.read_byte_data(i2c_addr, STATUS)
104         return status & STATUS_MASK_DATARDY
105 
106 def lsb_g():
107     if getRange() == 1:
108         lsb_g=256000
109         g_range=2
110     elif getRange() == 2:
111         lsb_g=128000
112         g_range=4
113     elif getRange() == 3:
114         lsb_g=64000
115         g_range=8
116     return lsb_g, g_range
```

Figur 24

I linje 103 leses status-registeret og i linje 104 utføres en AND-funksjon mellom innholdet i status-registeret og variabelen **STATUS_MASK_DATARDY**. Det gjør at det er kun bit 0 i status-registeret som returneres.

I linje 107 undersøkes det ved hjelp av **getRange()**-funksjonen fra linje 60 i figur 20, om måleområdet er ±2g. Hvis måleområdet er ±2g returneres tilhørende verdier for **g_range** og **lsb_g**. Det samme gjøres i linje 110 og 113, men her sjekkes det for måleområde på ±4g og ±8g.

I linje 118 til 134 i figur 25 utføres pkt.6 til 10 i figur 15.

```
118 if isRunning:
119     end()
120 
121 setRange(RANGE_2G)
122 
123 setLowpassFilter(ODR_4000)
124 
125 print("Logge tid",logge_tid,"sek")
126 print("Range =",lsb_g()[1])
127 
128 if getLowpassFilter() == 0:
129     print("SampleRate = 4000Hz")
130 else:
131     print("low SampleRate")
132 
133 
134 begin()
```

Figur 25

I linje 118 sjekkes det om ADXL355 er i standby modus med funksjonen **isRunning** fra linje46 figur19. Hvis ikke settes ADXL355 i standby-modus med **end()** fra linje 55 i figur 19. I linje121 settes måleområdet, i dette tilfellet, til ±2g med **setRange()** fra linje 63 i figur 20. I linje 123 settes ODR til 4000Hz og lavpassfilteret til 1000Hz med **setLowpassFilter()** fra linje 75 i figur 21. I linje 125 skrives loggetiden i konsollen. I linje126 skrives måleområdet i konsollen fra **lsb_g()**, element **[1]**. I linje 128 til 131 skrives ODRen i konsollen. Hvis ODR er under 4000Hz skrives «low Samplerate». I linje 134 settes ADXL355 i målemodus med funksjonen **begin()** fra linje 50 i figur 19.

I linje 136 til 156 i figur 26 utføres pkt.11 til 16 i figur 15.

```
136 # Åpner fil og lager heading(Time,Accel_x_mm/s2,Accel_y.......) om den ikke alerede finnes.
137 file = open(filnavn,"a")
138 if os.stat(filnavn).st_size == 0:
139     file.write("Time,Accel_X_RAW,Accel_Y_RAW,Accel_Z_RAW\n")
140
141 start = time.time()
142 # løkke som leser tid og vibrasjons data og skriver det inn i fila. for så å lukke koden når loggetiden er utløpt.
143 while time.time()<start+logge_tid:
144     if getStatus()== 1:
145         now=round(time.time()-start,6)
146         axes = getAxes()
147         x_accel_value=axes['x']
148         y_accel_value=axes['y']
149         z_accel_value=axes['z']
150         file.write(str(now)+","+str(x_accel_value)+","+str(y_accel_value)+","+str(z_accel_value)+"\n")
151         file.flush()
152
153 file.close()
154 end()
155 print("end")
156 print("visualiserer")
```

Figur 26

I linje 137 åpnes loggefilen. I linje 138 undersøkes det om loggefilen har en størrelse på 0 byte med **os.stat.st_size**. Hvis det er tilfelle, skrives(**write.)** det en «heading» i loggefilen med navn på kolonnene. I linje 141 settes ett referanse-tidspunkt for starten av loggingen. Dette gjøres med **time.time()**, som er antall sekund siden 1.jan 1970. I linje 143 starter løkken som logger akselerasjonsdata helt til loggetiden er utløpt. Her undersøkes det om det tidspunktet som linja leses, er mindre enn start-tidspunktet pluss loggetiden. Så undersøkes det i linje144 om det har kommet ny data i akselerasjonsregistrene. Når både line 143 og 144 er **true**, utføres linje 145 til 151.

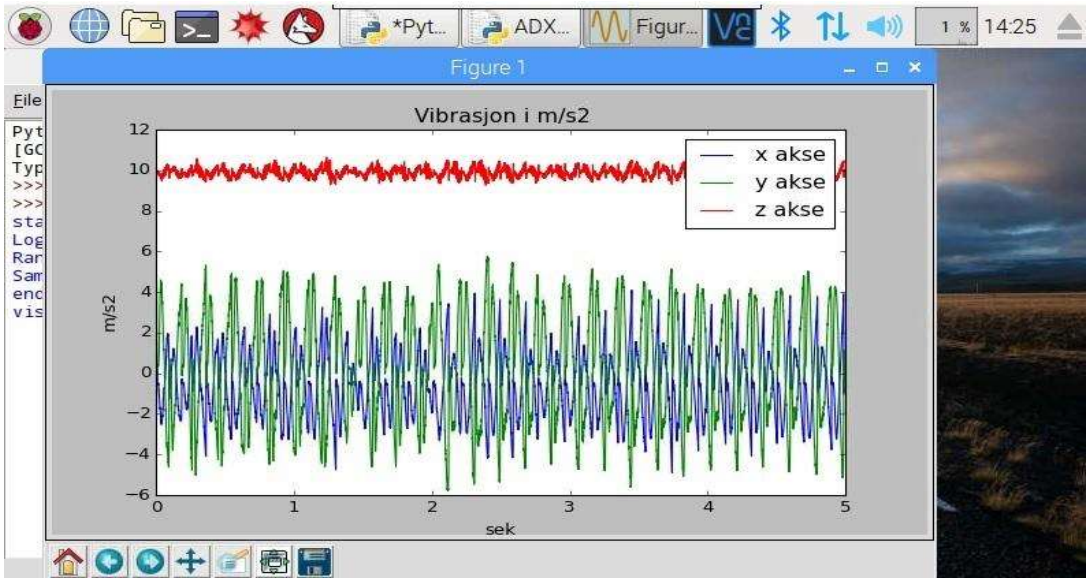
I linje 145 legges tiden som har gått siden start-tidspunktet, i antall sekund med 6 desimaler, inn i variabelen **now**. Deretter hentes akselerasjonsverdier med **getAxes()** fra linje 84 i figur 22 og legges inn i **axes**. I linje 147 til 149 legges akselerasjonsdata fra de 3 individuelle aksene inn i variabler merket med beskrivende navn. I linje 150 skrives (**write**) tidspunktet for målingen (**now**), og data fra de tre aksene inn i loggefilen. Her brukes **str** for å skrive tiden og akselerasjonsdataene inn i filen som desimaltall. Når utsagnet i linje 143 ikke lenger er sant går koden videre til linje 153 og lukker loggefilen med **close()**. I linje 154 settes ADXL355 i standbymodus. I 155 og 156 skrives «end» og «visualiserer» i konsollen for å fortelle at målingen er utført og at programmet begynner å lage en graf av det målte signalet.

I linje 157 til linje 169 i figur 27 visualiseres den målte vibrasjonen i tidsbilde. Dette tilsvarer pkt.17 til 19 i figur 15.

```
157 vib_data = pd.read_csv(filnavn)
158 plt.figure(1)
159 plt.plot(vib_data.Time,vib_data.Accel_X_RAW/lsb_g()[0]*9.81)
160 plt.plot(vib_data.Time,vib_data.Accel_Y_RAW/lsb_g()[0]*9.81)
161 plt.plot(vib_data.Time,vib_data.Accel_Z_RAW/lsb_g()[0]*9.81)
162 plt.legend(["x akse","y akse","z akse"])
163 plt.title("Vibrasjon i m/s2")
164 plt.xlabel("sek")
165 plt.ylabel("mm/s2")
166 plt.show()
167
168
169 sys.exit()
```

Figur 27

I linje 157 leses loggefilen med **pd.read_csv** og legges i variabelen **vib_data**. I 158 åpnes en figur som skal inneholde grafen. I linje 159 til 161 benyttes **plt.plot(**x-akse,y-akse) for å plotte signalet. Som x-akse, benyttes tids-kolonnen fra måledataen. Men for å få en y-akse som representeres i akselerasjon(m/s^2) må måledataen omformes. Måledataen er nå i området 524288(2^{19}) til -524288(-2^{19}). For å få dette representert i g, deles måledataen på oppløsningen (antall linjer per g), som hentes fra **lsb_g** i linje 126 figur 24. Deretter ganges verdiene, som nå er i g, med antall m/s^2 i en g (9,81m/s^2). Dataen i m/s^2 plottes som y-akse. Dette gjennomføres for alle vibrasjonsakser og plottes i samme graf. I 162 merkes de individuelle vibrasjonsaksene med egen farge og tittel. I 163 lages tittelen på grafen og i 164 og 165 merkes aksene til grafen med korrekt enhet. I 166 åpnes grafen på skjermen til Raspberry Pi-en. I linje 169 avsluttes koden. Figur 28 viser et eksempel på en graf fra en måling.



Figur 28

9 Resultatmål 4: Verifisering

I dette kapittelet skal prototypen verifiseres på bakgrunn av gitte krav til funksjonalitet og ytelse. Vi vil presentere fremgangsmetode og resultat av verifiseringen til temperatur- og vibrasjonssensoren. Med verifisering legger man til grunn for å bekrefte; ved å undersøke at noe er riktig.

9.1 Kalibrering Temperatursensor DS18B20

Den mest brukte metoden for å verifisere en temperatursensor er gjennom kalibrering.

«Kalibrering er en justering av et instrument mot en normal, mot et mer nøyaktig instrument eller mot et referansemateriale. Hensikten er å finne ut om instrumentet avviker fra korrekt verdi.» (Store Norske leksikon , 2018)

For å kunne kalibrere temperatursensoren ble det opprettet kontakt med Instituttet for energi og prosessteknikk ved NTNU. Ingeniør Aleksander Mosand stilte varmeteknisk laboratorium ved Gløshaugen til disposisjon.

9.1.1 Utstyr

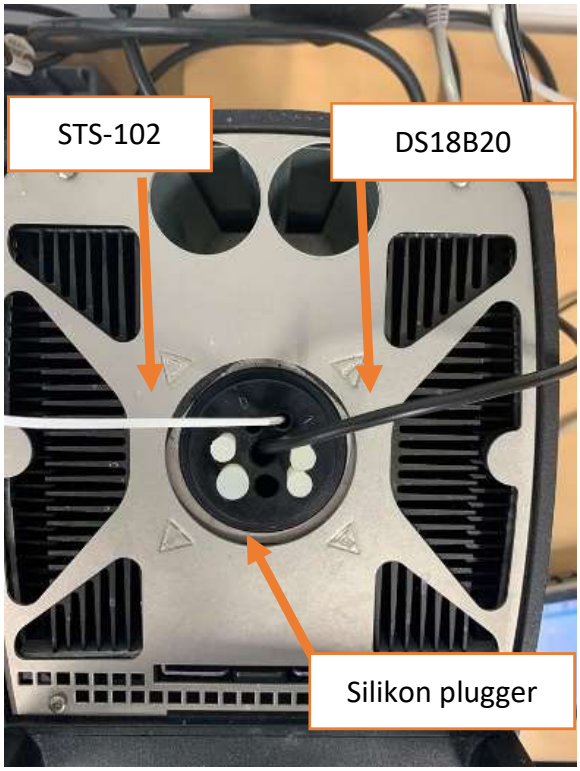
Apparatet som ble benyttet til kalibreringen var RTC-157 og en intern referansesensor STS-102, fra det danske firmaet Ametek Jofra. RTC-157 kan kalibrere innenfor et temperaturområde på -45 til +155 °C.
Den laveste stabilitetstoleransen til kalibreringsapparatet er oppgitt til ±0,005°C, mens nøyaktigheten til den interne referansesensoren er på ±0,1°C. Databladet til RTC-157 finnes i vedlegg 8.

Figur 46 RTC-157

9.1.2 Fremgangsmetode

Før gjennomføring

RTC-157 har 6 insert-hull på toppen av maskinen. Disse har forskjellig størrelser, tilpasset størrelsen på sensoren som skal kalibres. Temperatursensoren (DS18B20) slippes ned i det største hullet(12,7mm), mens den interne referansesensoren STS-102 benytter et mindre. De resterende hullene dekkes med silikonplugger.



Figur 47 Insert-hull til RTC-157

Stabilitetstoleransen ble satt til ±0,05°C og nøyaktigheten på den interne referansesensoren lik ±0,1°C. Det vil si at den dårligste kalibreringen man kan oppleve, vil gi en feilmargin på ±0,105 grader.

Før målingen kan begynne velges referanseverdier. Referanseverdier er de spesifikke temperaturene som skal kalibreres. Referanseområdet ble valgt fra 0C til 100C. Fra 0°C til 40°C, gjennomføres det tre kalibreringer (0°C,20°C,40°C). Fra 40°C-100°C velges et intervall på 10°C, dette siden temperaturen i oljereservoaret er forventet å ligge rundt 65°C.

Gjennomføring

RTC-157 starter med å kjøle seg ned til den laveste referanseverdien på 0°C. Når temperaturen når 0°C, stabiliseres RTC-157. Kravet for en stabil temperatur er at referansesensoren STS-102 skal ligge innenfor ±0,05°C på referanse temperaturen i 10 minutter. En gul firkant indikerer at stabiliseringen er under prosess. Når stabiliseringen er gjennomført skifter firkanten farge til grønn, dette indikerer at målingen kan starte. Etter fullført stabiliseringsprosess, logget gruppen 70 etterfølgende målinger over en periode på 2min med temperatursensoren DS18b20. Gjennomsnittet av de 70 målingene ble benyttet som den målte temperaturen. Siden DS18b20 er i kontakt med temperaturkilden under hele stabiliseringsprosessen til RTC-157, som tar over 10 min, antar gruppen at det lang nok responstid for DS18b20.

Kalibrering ved referanseverdi 20°C.
Den grønne firkanten indikerer at stabiliseringen er ferdig. Målingen kan begynne.

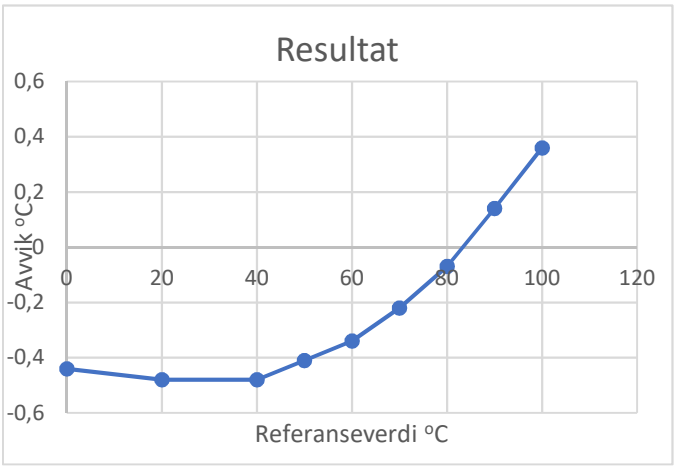
Figur 48 Stabilisering ved temperaturmåling

9.1.3 Resultat

Actual [°C]	Measured [°C]	Difference [°C]
0,00	-0,44	0,44
20,00	19,52	0,48
40,00	39,52	0,48
50,00	49,59	0,41
60,00	59,66	0,34
70,00	69,78	0,22
80,00	79,93	0,07
90,00	90,14	-0,14
100,00	100,36	-0,36

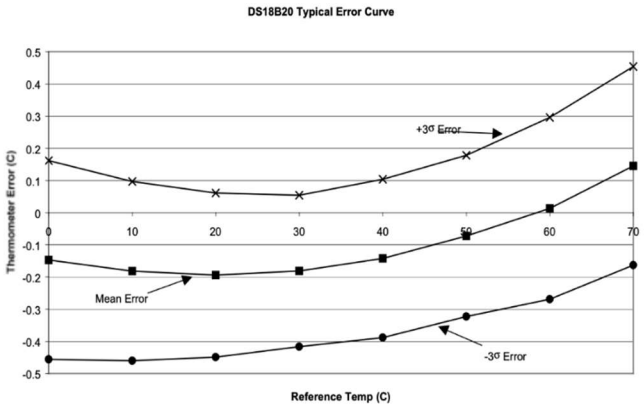
Tabell 14 Resultat temperaturkalibrering

Resultatene viser at DS18B20 har størst differanse ved 20°C og 40°C, hvor den måler 0,48°C for lavt. Rundt 80 til 90 °C er differansen til sensoren tilnærmet 0 °C. Dokumentet fra kalibreringen finnes i vedlegg 9.



Figur 49 Resultat temperaturmåling

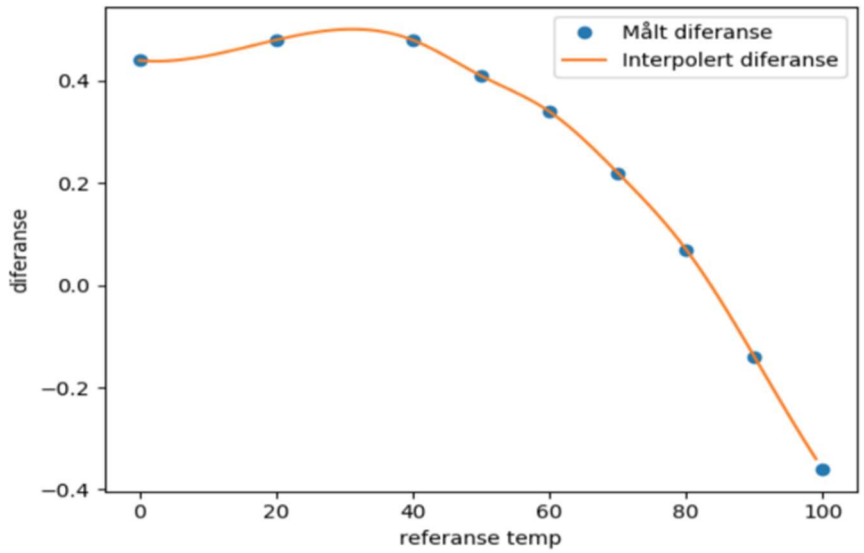
Resultatene fra kalibreringen (figur 49) viser et ulineært avvik. Dette ser ut til å stemme med den typiske feilkurven til DS18b20, vist i figur 50.



Figur 50 Datablad DS18B20, typisk feil kurve

8.1.4 Kubisk interpolasjon

For å måle mer nøyaktige verdier med DS18B20 i fremtiden legges differansen som ble målt, inn i systemet (Raspberry Pi). Verdier mellom referanseverdiene, tilnærmes ved å benytte en kubisk interpolasjonsfunksjon. Avviksfunksjonen, sammen med de målte avvikene, vises i figur 51.



Figur 51 Kubisk interpolasjon

Implementeringen av denne funksjonen i Python-koden beskrives i kap. 8.1.2.

9.1.5 Konklusjon

Det er viktig å presisere at kalibreringen av temperatursensoren er en sekundærkalibrering. Kalibreringen ble gjennomført mot et presisjonsinstrument som igjen fra tid til annen kalibreres mot en normal. Kalibringen mot normalen ble gjennomført av DANAK, som er det danske akkrediteringsfond (DEN DANSKE AKKREDITERINGSFOND, 2019). Gruppen kan dermed dokumentere sporbarheten til kalibreringsinstrumentet.

Men selv om instrumentet er dokumentert sporbar, betyr ikke nødvendigvis at det er mer nøyaktig. Det er garantien som ligger i sertifiseringen som er det viktige. «Danske akkrediteringsfond» følger internasjonale regler for kalibrering, og garantien på utførelsen er lik som det norske Justervesenet. Gruppa fikk etter kalibreringen var gjennomført, oppgitt at på kalibreringssertifikatet (vedlegg 10) til presisjonsinstrumentet var utgått på dato. Siden kalibreringssertifikatet ikke er gyldig lengre, kan man ikke bekrefte nøyaktigheten. Det er verdt å nevne er at kalibringen forteller om temperatursensoren sin tilstand den 4/4/19, og ikke gir noen garanti hva som kan skje i ettertid.

9.2 Sammenligning Vibrasjonssensor ADXL355

For å verifisere vibrasjonssensoren ADXL355 valgte man å sammenligne mot en antatt troverdig/kjent kilde. Intern veileder stilte vibrasjonsrigg og en håndholdt datainnsamler til disposisjon.

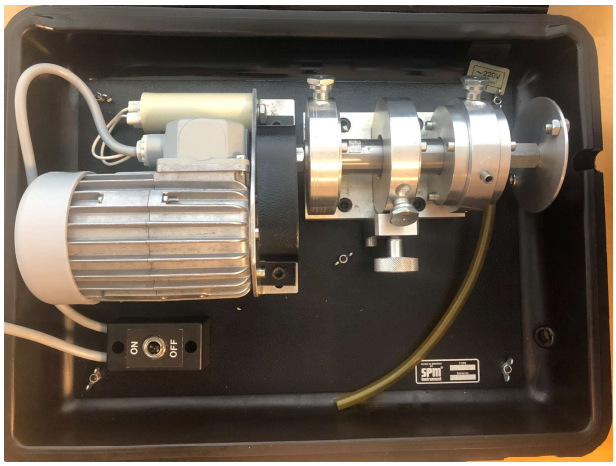
Sammenligningen gikk ut på å teste:

- Programmering og integrasjon er gjennomført riktig slik at RMS-verdi samsvarer.
- Programmeringen og signalbehandling er gjennomført riktig slik at frekvensbilde samsvarer.

9.2.1 Utstyr

Vibrasjonsrigg DEM-30

Vibrasjonsriggen DEM-30 av SPM-instrument består av 3 lager og en aksel koblet opp mot en motor. Motoren roterer med hastighet på 1300RPM og drives av elektrisitet(220V/50Hz). Vibrasjonsriggen er plassert i en lett håndterbar håndkoffert, med en vekt på 7,5kg. Databladet finnes i vedlegg 11.



Figur 52 Vibrasjonsrigg DEM-30

Leonova Diamond og SDL144

Leonova Diamond er en multi-funksjonell håndholdt datainnsamler for overvåkning og diagnostisering av maskiners tilstand. Funksjoner Leonova Diamond har:

- Vibrasjonsanalyse
- Shock pulse-overvåking
- Hastighetsmåling (RPM-måling)
- Temperaturmåling

Figur 53 Leonova Diamond

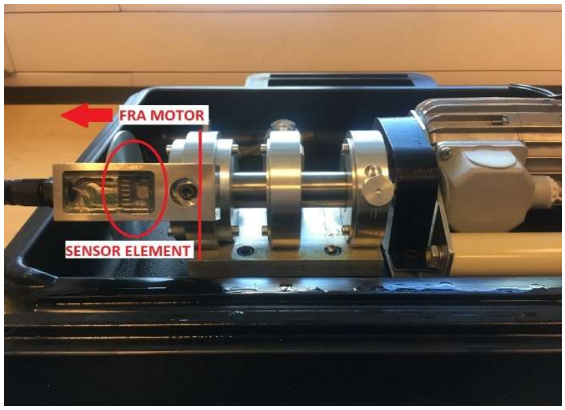
Under gjennomføringen av vibrasjonsmålingen ble et mobilt piezo-elektrisk akselerometer av typen SLD144S-M8 brukt. Akselerometret er kalibrert etter ISO-16063 -21 og har en frekvensgrense mellom 2 til 10000Hz. SLD144S består av en isolert kapsling i rustfritt syrefast stål som kan monteres med magnet eller skrues fast. Databladet til SLD144S finnes i vedlegg 12.

Figur 54 Akselerometer SLD-144S

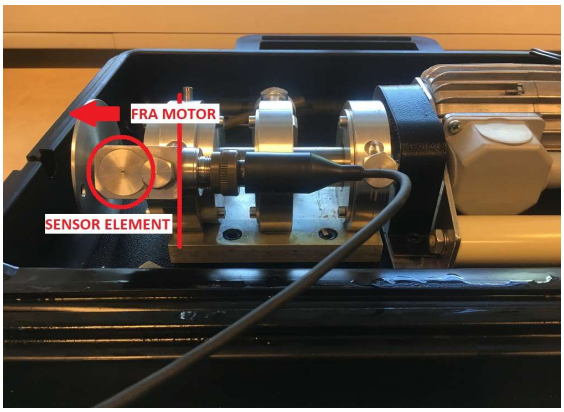
9.2.2 Fremgangsmetode

Sensorene ble plassert på det ytterste lageret på DEM-30. Begge sensorene ble skrudd fast i målepunktet. Den mest pålitelige metoden for montering er med skrueforbindelser (Mobius Institute, 2016). Akse-retningen til SLD-144s ble plassert radielt på senterlinjen til akslingen. Ved montering av ADXL355, var det z-aksen som ble radiell på senterlinjen til akslingen. Gruppen benyttet derfor kun data fra z-aksen til ADXL355 for sammenligning. For å undersøke om sensororientering hadde effekt på resultatet ble sensorene plassert i to retninger. Henholdsvis med sensorelementet mot motoren og fra motoren slik som vist i figurene nedenfor.

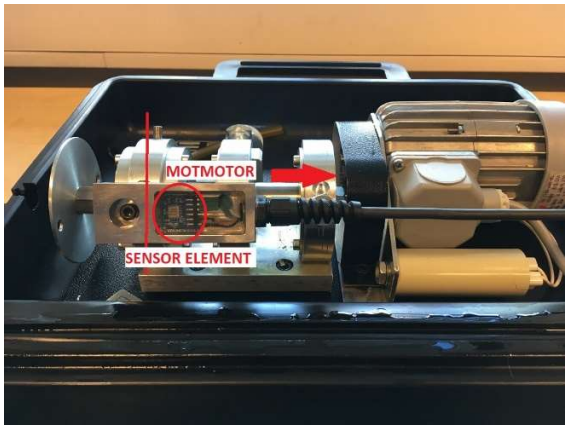
Det ble utført 5 individuelle målinger med hver sensor i hver retning.



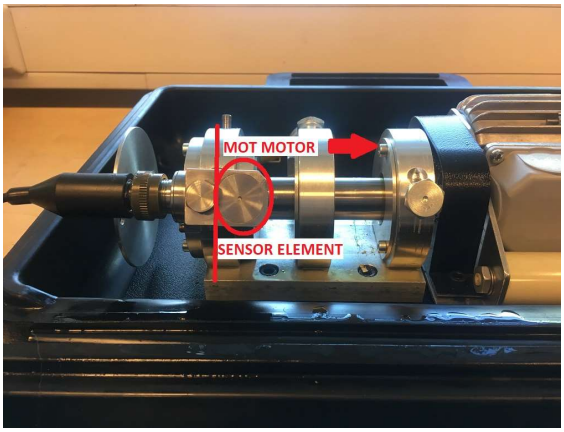
Figur 55 ADXL355 i orientering FRA MOTOR



Figur 56 SLD-144 i orientering FRA MOTOR



Figur 57 ADXL355 i orientering MOT MOTOR



Figur 58 SLD-144 i orientering MOT MOTOR

Innstillinger Leonova Diamond

Tabellen viser innstilingene som var benyttet på Leonova Diamond under målingene

	Valgte innstillinger
Laveste frekvens	2 Hz
Høyeste frekvens (F_{max})	1000Hz
Time signal unit	Akselerasjon m/s^2
Spectrum unit	Akselerasjon m/s^2
Window	Hanning
FFT Type	Amplitude Spectrum, Pk-amplitude
RPM	1300
LOR	1600 lines
Filter	Ingen
Avarage type	FFT linear
Avarage count	6
Måleperiode	9,6s
Oppløsning	0,625Hz

Tabell 15 Innstillinger Leonova Diamond

Ved å velge de innstillingene på Leonova Diamond som krever færrest steg for å sette tilsvarende innstillinger i signal-behandlingen til ADXL355, reduseres muligheten for følgefeil og feil-behandling av signalet.

FFT typen på Leonova Diamond ble satt til peak-amplitude. Dette ble gjort fordi peak-amplitude er FFT-typen som krever færrest operasjoner i en signal-behandling ved bruk av Python. Enheten i frekvensbildet på Leonova Diamond ble satt til akselerasjon. Da er det ikke nødvendig å benytte numerisk integrasjon på signalet fra ADXL355, for å få et likt sammenligningsgrunnlag av frekvensbildene.

Window typen ble satt til Hanning. Det er fordi det er den mest brukte windowing-typen og den gir minst amplitude-feil (Simens , 2019).
Average-type ble satt til lineær. Det ble gjort fordi det er den enkleste average-metoden å implementere i signal-behandlingen på signalet fra ADXL355.

Antall blokker (average count) ble satt til 6stk. 4-6 blokker er det mest vanlige antall blokker.
I signal-behandlingen utført på signalet fra ADXL355, benyttet gruppen forholdet for F_{max} og LOR som vist i ligning [4] og [5]. Der F_s er samplerate, Bn er antall målinger i en averaging-blokk, F_{max} er maks frekvensens i frekvensbildet og LOR er «lines of resolution» i frekvensbildet. (Mobius Institute, 2016)

$$[4]\ F_{max} = \frac{F_s}{2{,}56}$$

$$[5]\ LOR = \frac{Bn}{2{,}56}$$

F_s til Leonova Diamond er forskjellig fra F_s til ADXL355. Derfor vil FFT fra ADXL355 og FFT fra Leonova Diamond ha forskjellig F_{max} og LOR. Så for å få et likt sammenligningsgrunnlag må oppløsningen i begge frekvensbildene være lik. Oppløsningen til Leonova Diamond hentes fra forholdet i ligning [6], der $SPEC_{res}$ er oppløsningen i frekvensbildet.

$$[6] SPEC_{res} = \frac{F_{MAX}}{LOR} = \frac{1000Hz}{1600} = 0.625Hz$$

Innstillinger ADXL355

For å få et mest mulig likt sammenlignings grunnlag, ble måletiden til ADXL355 sensoren tilpasset for å oppnå samme $SPEC_{res}$ som på Leonova Diamond. Måletiden må være lang nok for å få totalt 7 averaging-blokker. Det ble benyttet totalt 7 blokker, der den første ble forkastet, fordi gruppen tidligere hadde observert at de første målepunktene som adxl355 returnerte var irregulære. For å justere blokktiden justeres antall målinger i en blokk. Forholdet mellom blokktid og Bn vises i ligning [7], Bt er blokktid og F_S er samplefrekvensen til prototypen, på 3715Hz.

$$[7] Bt = \frac{Bn}{F_s}$$

For å finne en Bn som gir en $SPEC_{res}$ lik 0,625 benyttes forholdet i ligning [8].

$$[8]\ SPEC_{res} = \frac{F_{MAX}}{LOR} = \frac{F_S}{2{,}56} * \frac{2{,}56}{Bn} = \frac{F_S}{Bn} \rightarrow \boldsymbol{Bn} = \frac{F_S}{SPEC_{res}} = \frac{3715}{0{,}625} = 5944$$

Måletiden regnes ut i [9].

$$[9] Måletid = 7 * Bt = 7 * \frac{Bn}{F_S} = 7 * \frac{5944}{3715} = 11{,}2s$$

F_{MAX} og LOR regnes ut i [10] og [11]

$$[10]\ F_{max} = \frac{F_s}{2{,}56} = \frac{3715}{2{,}56} = 1451Hz$$

$$[11]\ LOR = \frac{Bn}{2{,}56} = \frac{5944}{2{,}56} = 2322$$

Her er F_{MAX} er kun en teoretisk F_{MAX}, som benyttes i signalbehandlingen for å få lik oppløsning. Den reelle F_{MAX} begrenses til under 1000Hz pga. lavpassfilter på ADXL355.

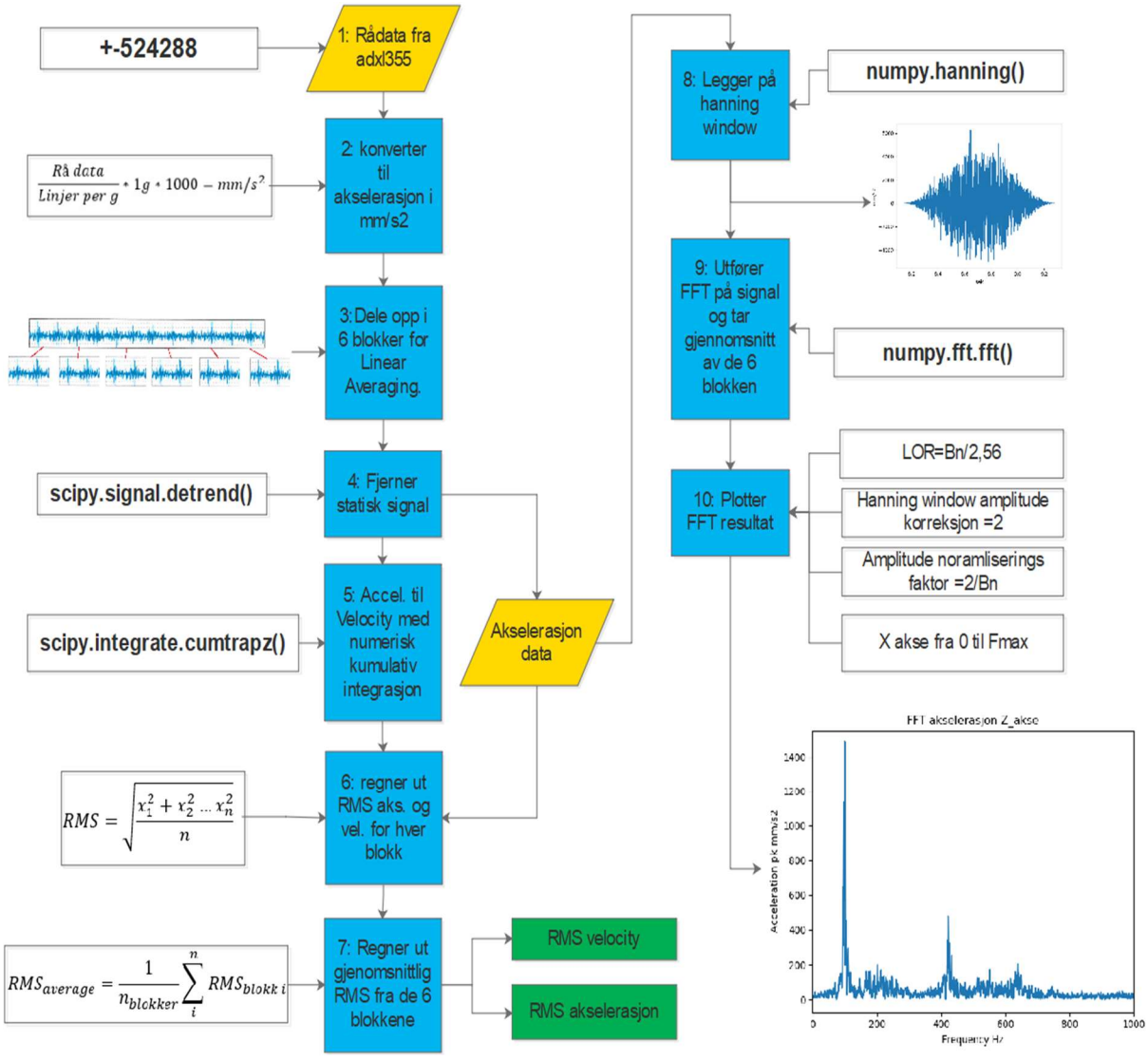
Tabell med verdier benyttet i signalbehandling

	Verdier benyttet
Antall målinger i en blokk, Bn	5944
Sample-frekvens, F_s	3715Hz
Laveste frekvens	0 Hz
Høyeste frekvens (F_{max})	1451
Time signal unit	Akselerasjon m/s^2
Spectrum unit	Akselerasjon m/s^2
Window	Hanning
FFT Type	Amplitude Spectrum, Pk-amplitude
RPM	1300
LOR	2322 lines
Filter	Analogt lavpassfilter =1500Hz Digitalt lavpassfilter = 1000Hz
Avarage type	FFT linear
Avarage count	6
Måleperiode	11,2s
Oppløsning	0,625Hz

Tabell 16 Innstillinger ADXL355

Signalbehandling

For å utføre signalbehandlingen, benyttet gruppen Python. Koden ligger vedlagt i sin helhet i eget vedlegg. Figur 59 viser framgangsmetoden som ble benyttet for å hente ut RMS_v, RMS_a og frekvensbildet med pk akselerasjonsamplitude.



Figur 59 Signalbehandlingsprosedyre

9.2.3 Resultater

Tabellene under viser sammenligningen av RMS verdier hentet fra målingene. Når sensorene var orientert fra motor, returnerete Leonova Diamond og ADXL355 nesten like RMS-akselerasjonsverdier med en forskjell på 1,6%. Men i samme orientering målte ADXL355 en gjenomsnittlig RMS velocity som var 18,2% høyere enn gjennomsnitlig RMS velocity fra Leonova Diamond.

Når sensorene var orientert mot motor, målte ADXL355 en gjennomsnittlig RMS-akselerasjon som var 46,5% høyere en gjennomsnittlig RMS-akselerasjonsverdi fra Leonova Diamond. I samme orientering returnerte ADXL355 en gjennomsnittlig RMS velocity verdi på 100,2% høyere enn fra Leonova Diamond.

Målingene fra orientering fra motor tyder på at ADXL355 måler korrekte akselerasjonsverdier, men at det er integreringen fra akselerasjon til velocity som utføres på feil måte. Målingen med senor-orientering motmotor tyder på at ADXL355 ikke måler korrekt akselerasjon, og at integreringen fra akselerasjon til velocity ikke er utført korrekt. Årsaken til feil målt akselerasjon i orienteringen mot motor, kan skyldes støy fra motor til signalkabelen til ADXL355.

Leonova Diamond	Sensor mot motor		Sensor fra motor	
	RMS v mm/s	RMS a m/s2	RMS v mm/s	RMS a m/s2
	2,71	1,85	2,46	2,08
	2,69	1,87	2,43	2,06
	2,65	1,81	2,39	2,05
	2,57	1,79	2,38	2,04
	2,57	1,79	2,35	2,03
Gjennomsnitt	2,64	1,82	2,40	2,05

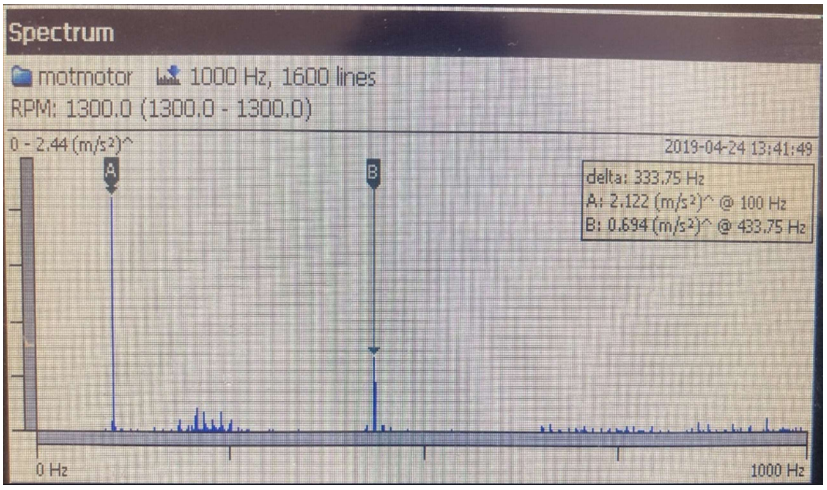
ADXL355	Sensor mot motor		Sensor fra motor	
	RMS v mm/s	RMS a m/s2	RMS v mm/s	RMS a m/s2
	5,27	2,66	3,23	2,07
	5,84	2,61	2,83	2,04
	5,41	2,61	2,93	2,03
	4,71	2,6	3,75	2,03
	5,28	2,67	2,84	2,02
Gjennomsnitt	5,30	2,63	3,12	2,04
Prosentvis økning	100,2 %	46,5 %	18,2 %	-1,6 %

Tabell 17 Resultater RMS verdier

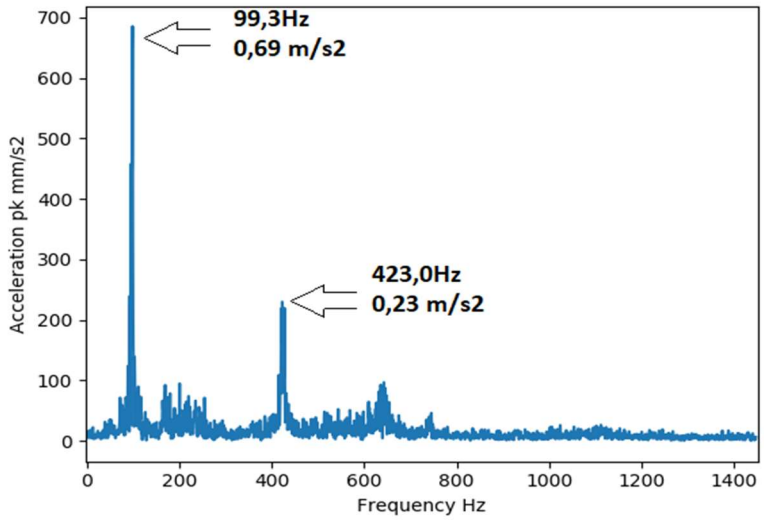
Frekvensbilde

Figur 60 viser frekvensspekteret fra en måling utført med Leonova Diamond og SDL-144s orientert mot motor. Det er to tydelige frekvenser som vises i frekvensbildet. Den første, markert med **A** i figur 56, har en frekvens på 100Hz og amplitude på 2,122m/s^2. Den andre tydelige frekvensen, markert med **B** i figur 55, har en frekvens på 433,75Hz og amplitude på0,694m/s^2. Frekvensbildet til ADXL355 (figur 55), har 2 tydelige frekvenser med henholdsvis frekvens på 99,3Hz og 423,0Hz.

Frekvensene fra Leonova Diamond og ADXL355 er ikke helt like, men i nærheten av hverandre. Amplitudene på frekvensene fra ADXL355, som var på 0,69m/s^2 på 99,3Hz og 0,23m/s^2 på 423,0Hz, er mye lavere en amplitudene fra Leonova Diamond. Dette kan tyde på at signalet fra ADXL355 gir en representativ måling, men at det skjer en feil i amplitude-korreksjonen for FFT-en i signal-behandlingen.

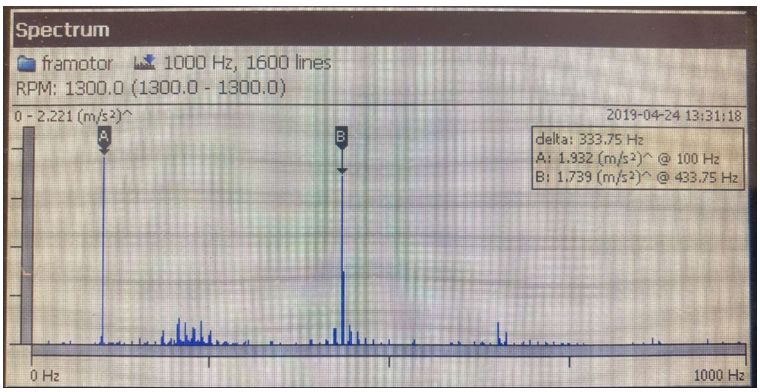


Figur 60 FFT fra Leonova Diamond, sensororientering MOT MOTOR

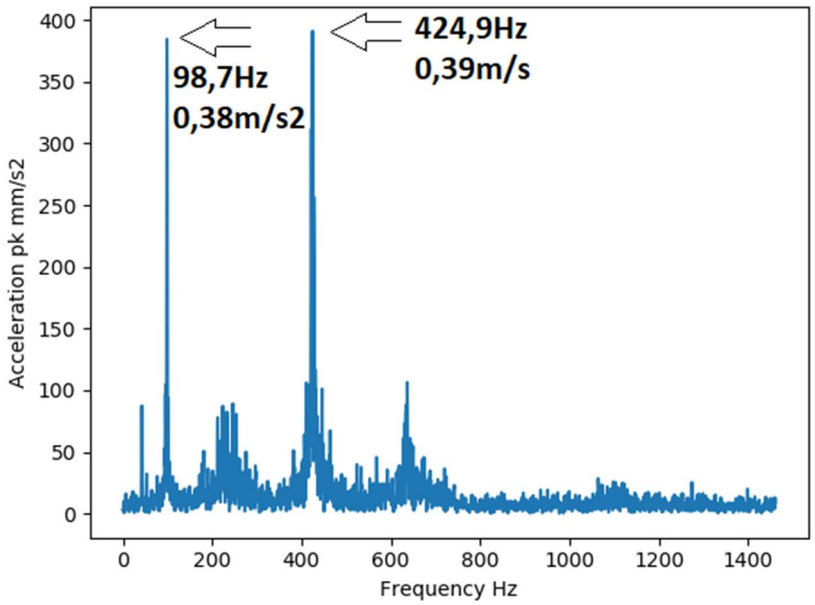


Figur 61 FFT fra ADXL355, sensororientering MOT MOTOR

Figur 62 og 63 viser frekvensbildene tatt med sensorene orientert fra motoren. Også her vises 2 tydelige frekvenser på 100Hz og 433,75Hz i frekvensbildet fra Leonova Diamond. Frekvensbildet til ADXL355 viser to frekvenser på 98,7Hz og 424,9Hz, som igjen ligger litt under frekvensene fra Leonova Diamond. Amplitudene er også her helt forskjellige. Leonova viser en amplitude på 1,932m/s^2 for 100Hz signalet mens ADXL355 viser 0,38 m/s^2. På 433,75Hz signalet viser Leonova en amplitude på 1,739m/s^2, mens ADXL355 viser 0,39m/s^2. Siden ADXL355 plukker opp to frekvenser som ligner på frekvensene som Leonova Diamond viser, kan resultatene tyde på feil og manglende signal-behandling utført på signalet fra ADXL355.



Figur 62 FFT fra Leonova Diamond, sensororientering FRA MOTOR



Figur 63 FFT fra ADXL355, sensororientering FRA MOTOR

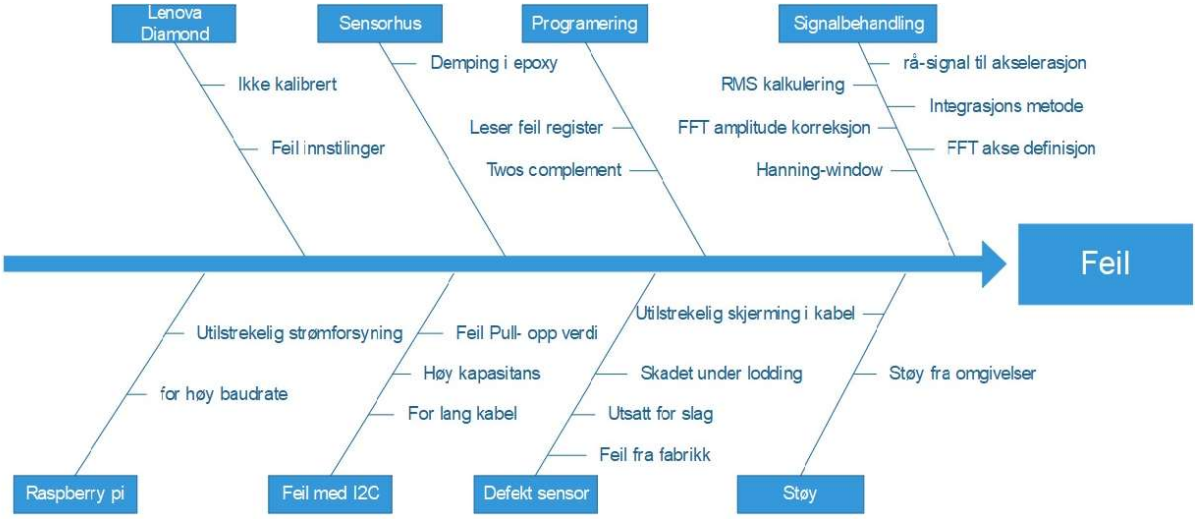
9.2.4 Konklusjon

Sammenligningen av Leonova Diamond med ADXL355 viser avvik mellom de to målekjedene. Avviket er så stort at det tyder på feil i målekjeden til ADXL355. Resultatene fra RMS sammenligningen tyder på at integrasjonen av signalet ikke er utført korrekt. I tillegg viser resultatet at ADXL355 er orienterings-sensitiv i vårt oppsett, siden den måler rett akselerasjon i en orientering men feil i en annen orientering.

Sammenligningen av FFT viser at ADXL355 plukker opp ett frekvensbilde som ligner på frekvensbildet til Leonova Diamond. Men grove feil i amplituden og mindre feil i frekvensen tyder på feil signal-behandling. Selv om feilen i frekvensene er relativt små i forhold til amplitude-feilen, er frekvensfeilen så stor at de ikke kan benyttes til en troverdig frekvensanalyse.

Resultatene viser at systemet med ADXL355, Raspberry Pi og vår signalbehandling, ikke bør benyttes til å samle inn troverdige vibrasjonsdata for en tilstandskontroll. Denne sammenligningen er basert på at Leonova Diamond representerer reelle målinger. Men Leonova Diamond som ble brukt har ikke fulgt anbefalt kalibreringsintervall fra leverandør.

9.2.5 Mulig feilkilder



Figur 64 Mulige feil kilder

10 Diskusjon

I dette kapittelet vil resultatene drøftes. Ulike synspunkter, årsaker og faktorer settes opp mot hverandre og ved enkelte punkter vil man presentere ting som kunne blitt gjort annerledes.

10.1 Resultatmål 1: Ståstedsanalysen

Målepunkt og måleområde

Målepunktene for vibrasjon og temperatur, ble valgt på bakgrunn av EBLs håndbok og befaring på Brattset. I tillegg ble målepunkt for vibrasjon sjekket opp mot ISO 20816-5. Målepunktene for temperatur og vibrasjon antas derfor godt egnet.

Anbefalte grenseverdier til vibrasjonsamplitude ble hentet fra ISO 20816. Men siden verdiene ble omgjort fra RMS velocity til pk-akselerasjon, er det usikkerhet rundt fremgangsmetoden som er beskrevet i ståstedsanalysen (kap.5.3.3). Vibrasjonssensoren som ble valgt til tilstandskontrollsystemet, dekket forventede amplituder og frekvenser av interesse. Sensoren ble valgt slik at forventet amplitude er under 20% av amplitudeområdet til sensoren. Frekvensområdet av interesse defineres i ISO 20816. Derfor tror gruppen at vibrasjonssensoren vil håndtere forventet måleområde uten å filtrere ut signaler av interesse.

Måleområdet for temperatur var dokumentert i EBL sine håndbøker. I tillegg fikk gruppen under befaringen på Brattset oppgitt temperaturen i turbinlageroljen på daværende tidspunkt. Gruppen antar derfor, at valgt temperatursensor vil håndtere forventet måleområde.

Kapasiteter og begrensinger Raspberry Pi.

For å identifisere kapasiteter og begrensninger til Raspberry Pi, ble det benyttet kilder og bøker som Raspberry Pi Cook Book (Monk, 2016) og Advanced Raspberry Pi (Gay, 2018). Slike kilder gir god informasjon om begreper og elementer gruppen ønsket å avdekke med Raspberry Pi. Men på en annen side tar slike kilder ikke høyde for våre spesifikke ytelseskrav til komponentene som skal benyttes sammen med Raspberry Pi. Det kan medføre at begrensinger som gjelder vår målekjede ikke avdekkes. Med mulig ukjente begrensinger, kan prototypen som er basert på Raspberry Pi, operere i grenseområdet for pålitelig funksjonalitet. Men utgangspunktet lå til rette for at oppgaven var gjennomførbar.

For å forsikre seg om at relevante begrensinger ble avdekket i ståstedsanalysen, burde gruppen i tillegg til å lese relevante kilder, forsøkt å finne personer med spesifikk erfaring for vår problemstilling.

10.2 Resultatmål 2 og 3: Prototype med manual

Komponenter

Komponentene til prototypen ble bevisst kjøpt billig, for å holde kostnadene nede. Vårt system hadde en totalkostnad på 2427.-NOK, mens lignende system koster rundt $1599 (Digiducer., 2019). Hvordan dette har påvirket prototypens funksjonalitet og ytelse kan det være delte meninger om. Prototypens pålitelighet og evne til å håndtere omgivelser, vil kunne være svekket med å benytte seg av billige komponenter. En utprøving i felt er nødvendig for å verifisere funksjonsevnen til prototypen. På en annen side viser resultatet fra kalibreringen av temperatursensoren at det er fullt mulig å designe og konstruere en funksjonell prototype av billige komponenter.

Kabinett

Produksjonsmetoden (3D-printing) av boksen kan være begrensende for å tilfredsstille industrielle krav, i form av støv, fukt og støtskader. 3D-printing er fortsatt i en tidlig fase, og mangel på faglig materiell bidrar til å svekke avgjørelser knyttet til konstruksjonen. Men det er viktig å presisere at dette er en prototype. 3D-printing gir kort vei mellom ide og visualisert løsning, samt at det er lav kostnad, og enkelt å gjøre justeringer. Gruppen legger større vekt på å få frem pålitelige målinger, enn å lage en prototype som tåler mekaniske påvirkninger fra operasjonsmiljøet. Den beste prototypen er en som på den enkleste og mest effektive måten gjør mulighetene og begrensningen til en designidé synlig og målbar. (Stolterman, et al., 2008).

Programmering

Et meget sentralt element i prosessen med å lage tilstandskontrollsystemet, er programmeringen i Python som kontrollerer sensorene, og logger data. Ingen på gruppen hadde noen form for programmeringserfaring fra før av. Kvaliteten på programmeringen kan dermed diskuteres. Komplekse feil av sporadisk karakter, kan være vanskelig å oppdage med liten erfaring, og slike feil kan forbli uoppdaget.

Programmeringsspråket som gruppen benyttet var Python. Dette ble valgt fordi det er et lett språk å lære. I tillegg er Python «open source», så mange eksempler og koder er tilgjengelig for allmenheten. Det ble ikke undersøkt om andre programmeringsspråk var bedre egnet for dette formålet.

Men på en annen side har man verifisert sensorene i ettertid, og fått antydninger på hva som er gjennomført riktig, og hvor feilene ligger. Basert på erfaringene burde gruppen tatt et programmeringskurs og kvalitetssikret programmeringen ved å knytte bedre kontakt med personer innenfor programmeringsfagfeltet.

Brukermanual

Brukermanualen til tilstandskontrollsystemet ble utarbeidet på et erfaringsbasert grunnlag. Alle stegene i brukermanualen er gjennomført selv, og med ønsket effekt. Brukermanualen er utformet med tanke på operasjoner som gruppen har utført. Den tar ikke for seg en dyptgående feilsøking. Det er ingen andre enn gruppemedlemmene som har sett manualen. Det er mulig manualen kan være mangelfull for en person som ikke har sett systemet som manualen beskriver.

For å finne uoppdagede mangler i manualen, kunne gruppen fått en medstudent til å utføre målinger, kun på bakgrunn av brukermanualen. Dette kunne avdekket manglende informasjon i manualen. Dog var dette ikke krevd i oppgaven, og det var heller ikke tid til å gjennomføre det.

10.3 Resultatmål 4: Verifisering

Kalibrering temperatursensor

I lys av resultatene fra kalibreringen, kan man påstå at metoden for gjennomføringen var pålitelig. Man har dokumentert sporbarheten til temperatursensoren, men nøyaktigheten er usikker grunnet at kalibreringssertifikatet til RTC-157 har gått ut på dato. Ser man på gjennomføringen av kalibreringen i ettertid, burde vi etterspurt gyldigheten til kalibreringssertifikatet før gjennomføringen.

Gruppen valgte å justere avviket mellom temperatursensor og RTC-157, ved bruk av kubisk interpolasjon i Python. Man kan stille seg spørsmål om behovet for å justere avviket. Håndboken til EBL gir karaktersetting for endringen av registrert temperatur. En temperaturøkning på 1-3^{o}C, gir kriteriene: mindre avvik/ingen forandringer. Siden kalibreringen viste det største avviket på 0,48^{o}C, ville det kanskje ikke vært nødvendig å justere avviket. Ved å genere nye funksjoner som muligens ikke er nødvendig inn i systemet, øker man muligheten for nye feilkilder.

Sammenligning av vibrasjonssensor

Signaler fra akselerometer er generelt sett utsatt for å bli påvirket av støy. Derfor bør det tas hensyn til blant annet plassering av kabelen til akselerometeret. (Bye, 2009). Figur 57 viser tydelig at kabelen fra ADXL355 ligger inntil motoren som driver akslingen. Kabelplasseringen kan påvirke resultatet av sammenligningen. Men det ble utført målinger med 2 forskjellige sensororienteringer. I sensorretningen «fra motor» ligger kabelen til ADXL355 ikke i nærområdet til motoren. For å få et bedre sammenligningsgrunnlag burde gruppen valgt en bedre kabelplassering.

Under sammensetningen av sensorhuset og ADXL355, ble det ikke utført tilstrekkelige tiltak for å sikre at z-aksen til sensoren, som ble benyttet i sammenligningen, sto normalt på undersiden av sensorhuset. Men testmålinger som ble utført etter innstøpning av sensoren, viste et statisk signal på 9,8 m/s^2 når sensoren lå på et flatt underlag. Det tydet på at z-aksen stod normalt på undersiden av sensorhuset. Siden disse testmålingene ikke ble dokumentert, er det usikkerhet om aksene til sensorene som ble sammenlignet, pekte i nøyaktig samme retning. Disse testmålingene ble ikke dokumentert fordi systemet var designet for å måle absoluttvibrasjon, og ikke målinger med kun en akse. Likevel burde det blitt benyttet en bedre sammenstillingsmetode av sensoren for å sikre at aksene til ADXL355 stod normalt på de tilhørende flatene i sensorhuset.

Får å sikre et godt sammenligningsgrunnlag, ble de matematiske prinsippene beskrevet i kap. 4.1 og kap. 9.2.2, som kan ha påvirkning på sammenligningsgrunnlaget, studert. Det ble benyttet antatt gode kilder som Mobius Institute og Simens Knowledge Base for å forstå de matematiske prinsippene. I tillegg ble det referert med professor Antoine Rauzy og professor Dag Roar Hjelme for hvordan disse prinsippene kunne bli implementert i signalbehandlingen til ADXL355. Men signalbehandlingsmetoden som ble benyttet i sammenligningen var ikke kontrollert av andre enn gruppemedlemmene, på grunn av manglende tid. Det er derfor usikkert hvorvidt implementeringen av prinsippene er utført på korrekt måte i signalbehandlingen. Resultatet av sammenligningen indikerer dette. Gruppen burde fått en eller flere personer med signalbehandlingskompetanse, til å verifisere at signalbehandlingen ble utført korrekt.

11 Konklusjon

Formålet med denne oppgaven var å undersøke muligheten for at en gruppe uten noe dyptgående kunnskaper eller erfaring innen elektronikk og programmering, kan sette sammen et **lavkost** tilstandskontroll**-system**. Et slikt system har til hensikt å detektere relativ endring i tilstand. For å undersøke dette ble det satt opp en målekjede for tilstandskontroll på et turbinlager på en Francis-turbin. Tilstandsparametere som skulle måles var vibrasjon og temperatur.

Målekjeden for temperatur ble satt sammen og kalibrert. Resultatet fra kalibreringen viser at systemet logger temperatur med akseptabel feilmargin. Målekjeden og signalbehandlingen for temperaturmålingen antas derfor å være korrekt utført. Men systemet er ikke verifisert under reelle driftsforhold, og det er usikkert om systemet vil reagere på støv, fukt og slag.

Målekjeden for vibrasjon ble satt sammen og sammenlignet med en kjent kilde, men med mindre suksess. Tilstandskontrollsystemet logger akselerasjon, i form av heltall mellom ±524288, og lagrer dataene i en csv.-loggefil. Dette viser at det er mulig å benytte rimelige og kommersielt tilgjengelige komponenter, for å sette sammen et system som logger akselerasjon. Gruppen lyktes ikke med å sette opp en komplett målekjede, slik at avleste verdier samsvarte med sammenligningskilden Leonova Diamond. Dette kan skyldes flere faktorer, men feil i signalbehandlingen som er utført i Python, er antatt som hovedfeilkilden. I tillegg er det muligheter for at kommunikasjonen mellom sensor og datalogger opererer på kanten av tilfredsstillende krav.

Konseptet er gjennomførbart. For å få en pålitelig målekjede kreves det en gjennomgang av alle ledd, fra sensor til signalbehandlingen i Python. For å kontrollere leddet mellom sensor og datalogger, kan man benytte en kommunikasjonsprotokoll som ikke er like utsatt for støy og kabelbegrensninger. Dette kan for eksempel gjøres ved å sette en microkontroller inn i sensorhuset. Da kan det benyttes I^2C over den korte avstanden mellom microkontroller og sensor innad i sensorhuset. Data fra sensor og microkontroller kan da sendes til Raspberry Pi over en bedre egnet kommunikasjonsprotokoll, som UART eller USB Serial. Ved å i tillegg utbedre signalbehandlingen, kan et slikt system være et reelt alternativ til vibrasjon- og temperatur-tilstandsovervåkning.

Slike lavkost tilstandssystem kan bli reelle løsninger med tanke på utviklingen innenfor elektronikk. Sannsynligvis, kan det i nær framtid dukke opp sensorteknologi som er billig, pålitelig, og enkel å integrere i et system slik at det kan bli benyttet i en CBM-applikasjon. MEMS-teknologien er et eksempel på dette. Aktører som Analog Devices har allerede utviklet komplette MEMS-sensorer med sensorhus og egnede kommunikasjonsprotokoller for implementering i et CBM-system. I og med at dataloggeren som er benyttet i vårt system også har innebygd WiFi, passer dette konseptet inn i Industri 4.0 og IoT-utviklingen.

11.1 Videre arbeid

Feilretting

Det første som bør jobbes videre med, er målekjeden til vibrasjon. Sammenligningen som gruppen utførte tyder på at signalbehandlingen ikke fungerer. Det kan være naturlig å utvikle en ny, eller forbedre den eksisterende signalbehandlingen, slik at alle utregninger av RMS-verdier og FFT blir utført korrekt. I tillegg kan kommunikasjonen mellom akselerometeret og Raspberry Pi forbedres. Dette kan for eksempel gjøres ved å sette inn en microchip i sensorhuset for å styre sensoren, og videreformidle signalet over UART eller USB Serial til Raspberry Pi for lagring.

Videreutvikling

Raspberry Pi har flere innganger og utganger enn det som er benyttet i denne oppgaven. Det er mulig å videreutvikle systemet til å bruke flere parametere enn temperatur og vibrasjon. Noen av parameterne som kan være aktuelle er;

- Ultralyd
- Trykk
- Balanseringsinstrument
- Strømninger
- Statistisk prosesskontroll
- Avstandsmålere
- Kraftmålere

Prototypen er et måleinstrument. Med utvikling av programvaren kan bruksområdet utvides. Siden Raspberry Pi leveres med WiFi, kan prototypen utvikles til et overvåkningssystem som logger og sender tilstandsdata kontinuerlig. Eventuelt kan tilstandsparametere måles periodisk for å begrense datamengden. I tillegg kan overvåkningsalgoritmer med alarm implementeres. Slik at systemet varsler for uønskede parameternivå.

Prototypen har ingen programvare for å forenkle bruken eller GUI (graphical user interface). Målinger utføres ved å kjøre Python-koder i et utviklermiljø (IDLE). Ved å videreutvikle programvaren som tillater brukeren å styre målinger og innstillinger på sensorene, kan brukervennligheten økes. I tillegg vil en godt utviklet GUI minske inntastingsfeil og utilsiktede endringer i programvaren.

Det kan være grunnlag for å videreutvikle kabinettet. Dette kan designes og produseres så det tåler støv, fukt, og slag bedre.

12 Referanseliste

Bjørndal, H., 2007. *Svorkmo kraftverk,* Sandvika: Norconsult.
Bye, P. I., 2009. *Vedlikehold og driftsikkerhet.* Trondheim : s.n.
Cooper, R., 2011. *winning at new products.* 5 red. s.l.:s.n.
Dalland, O., 2007. *Metode og oppgaveskriving for studenter.* 4 red. OSLO: Gyldendal Norske Forlag AS.
DEN DANSKE AKKREDITERINGSFOND, 2019. *DANAK.* [Internett]
Available at: http://portal.danak.dk/
[Funnet 20 April 2019].
Digiducer., I., 2019. *Digiducer.* [Internett]
Available at: https://digiducer.com/
[Funnet 24 Februar 2019].
EBL, 1994. *Tilstandskontroll av vannkraftverk.* 1 red. s.l.:ENFO.
estudie, 2019. *estudie.* [Internett]
Available at: https://estudie.no/fjerde-industrielle-revolusjon/
[Funnet mars 4 2019].
Gay, W., 2018. *Advanced Raspberry Pi.* Berkeley, CA: Apress.
Hanley, S., 2016. *Accelerometer Specifications: Deciphering an Accelerometer's Datasheet.* [Internett]
Available at: https://blog.mide.com/accelerometer-specifications-decoding-a-datasheet
[Funnet 15 Februar 2019].
Hanly, S., 2016. *Accelerometers: Taking the Guesswork out of Accelerometer Selection.* [Internett]
Available at: https://blog.mide.com/accelerometer-selection
[Funnet 15 Februar 2019].
Hohn, A., 2017. *makecsv.py,* s.l.: github.com.
Holmen, I. M. & Solvang, B. K., 1996. *Metodevalg og metodebruk.* 3. red. Otta: Engers Boktrykkeri A/S.
IMI sensors, 2017. *Industrial ICP Triaxial Accelerometers.* Depew: IMI sensors.
International Organization for Standardization, 2018. *ISO 20816 Mechanical vibration — Measuremen tand evaluation of machine vibration,* Geneva: International Organization for Standardization.
ISO 10816-3, 2009. *ISO 10816-3.* s.l.:International Organization for Standardization.
JCGM, 2008. *International vocabulary of metroly.* s.l.:s.n.
Larsen, A. K., 2017. *En enklere metode: Veiledning i samfunnsvitenskapelig forskningsmetode.* 2 red. s.l.:s.n.
Leseth, A. B. & Tellemann, S. M., 2014. *Hvordan lese kvantitativ forskning.* 1. red. Oslo: Cappelen Damm AS.
Looney, M., 2014. An Introduction to Mems Vibration Monetoring. *Analog Dialouge*, 6 Juni, p. 3.
Mobius Institute, 2016. *Data Acquisition.* s.l.:Mobiusinstitute.
Mobius Institute, 2016. *Signal Processing.* s.l.:Mobiusinstitute.
Mobius Institute, 2016. *Vibration Analysis and training manual.* s.l.:Mobiusinstitute.
Monk, S., 2016. *Raspberry Pi Cookbook.* 2 red. s.l.:O`Reilly Media.

Mosaic Industries, Inc., u.d. *Mosaic Documentation Web.* [Internett]
Available at: http://www.mosaic-industries.com/embedded-systems/sbc-single-board-computers/freescale-hcs12-9s12-c-language/instrument-control/i2c-bus-specifications
[Funnet 15 februar 2019].
Murphy, C., 2017. Choosing the Most Suitable MEMS Accelerometer for Your Application. *Analog Dialogue*, 10 Oktober.
NIST, 2019. *NIST.* [Internett]
Available at: https://www.nist.gov/time-and-frequency-services/p
[Funnet 2 April 2019].
Norsk standard, 2019. *Norsk standard.* [Internett]
Available at: https://www.standard.no/standardisering/ce-merking/
[Funnet 5 mars 2019].
NS-EN 13306, 2017. *Norsk Standard NS- EN 13306:2010.* s.l.:Den europeiske standardiseringsorganisajon.
NTi audio AG, u.d. *Fast Fourier-transformation.* [Internett]
Available at: https://www.nti-audio.com/en/support/know-how/fast-fourier-transform-fft
[Funnet april 2018].
NTNU-VIKO, 2019. *NTNU.* [Internett]
Available at: https://innsida.ntnu.no/wiki/-/wiki/Norsk/Finne+kilder#section-Finne+kilder-Hvordan+v%C3%A6re+kildekritisk?
[Funnet 4 April 2019].
Python Software Fundation, 2019. *Python 3.7.3 documentation.* [Internett]
Available at: https://docs.python.org/3/
[Funnet 29 April 2019].
Radbourne, M., 2017. *Minimal Python class to read data from an ADXL355 accelerometer,* s.l.: Github.Inc.
Richardson, M. & Wallace, S., 2016. *Getting Started With Raspberry Pi.* 3 red. San Francisco: Makermedia.
S. Parab, J. et al., 2008. *Practical Aspects of Embedded Sytem Design using Microcontrollers.* 1. red. Dordrecht: Springer Netherlands.
Scipy.org, 2019. *Scipy.org.* [Internett]
Available at: https://scipy.org/index.html
[Funnet 29 April 2019].
Siemens, 2017. *Siemens knowledge base.* [Internett]
Available at: https://community.plm.automation.siemens.com/t5/Testing-Knowledge-Base/Window-Types-Hanning-Flattop-Uniform-Tukey-and-Exponential/ta-p/445063
[Funnet 12 April 2019].
Simens , 2019. *Simens.* [Internett]
Available at: https://community.plm.automation.siemens.com/t5/Testing-Knowledge-Base/Window-Correction-Factors/ta-p/431775'
[Funnet 4 mars 2019].
Spence, E., 2017. What You Need to Know About MEMS Accelerometers for Condition Monitoring. *Analog Dialogue*.
Stolterman, . E., Lim, Y.-K. & Teneberg, J., 2008. *The Anatomy of Prototypes.* [Internett]
Available at:
http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.499.9367&rep=rep1&type=pdf
[Funnet 6 mars 2019].

Store Norske leksikon , 2018. *SNL.* [Internett]
Available at: https://snl.no/maskinvare
[Funnet April 4 2019].
Store Norske leksikon , 2018. *SNL.* [Internett]
Available at: https://snl.no/kalibrering
[Funnet 5 Mars 2019].
Struers, 2019. *Struers.* [Internett]
Available at: https://www.struers.com/en/About-Struers
[Funnet 2 mars 2019].
Thagaard, T., 2018. *Systematikk og innlevelse.* 5 red. Oslo: Fagbokforlaget.
theconstructor, 2014. *theconstructor.com.* [Internett]
Available at: https://theconstructor.org/practical-guide/francis-turbines-components-application/2900/
[Funnet 4 mai 2019].
Thingiverse, 2015. *Thingiverse.* [Internett]
Available at: https://www.thingiverse.com/thing:1082431
[Funnet 15 februar 2019].
Ulrich, K. & Eppinger, S., 2012. *Product design and development.* 5 red. New York: McGraw-hill.
Weele, A. v., 2014. *Porchasing and supply chain mangemant.* 6 red. s.l.:s.n.
What Is a Raspberry Pi. 2015. [Film] Regissert av Saladhouse. UK: Raspberri Pi Foundation.
xyobalancer, 2011. *xyobalancer.com.* [Internett]
Available at: http://www.xyobalancer.com/xyo-balancer-blog/journal_bearings_oil_whirl_and_oil_whip
[Funnet 5 mai 2019].