Bacheloroppgave

NTNU
Norges teknisk-naturvitenskapelige universitet
Fakultet for informasjonsteknologi og elektroteknikk
Institutt for elektroniske systemer

Aanensen, Eskild Jonatan Krumsvik; Berthelsen, Patric André; Henriksveen, Sigurd Ranheim; Kristiansen, Jacob

Prototype av sky-tilkoblet målesystem for EAQI, basert på nRF9160 SiP med energihøstingssystem

Prototype of a cloud-connected monitoring system for EAQI, based on the nRF9160 SiP with an energy harvesting system

Bacheloroppgave i Elektroingeniør (FTHINGEL)
Veileder: Arne Midjo

NTNU
Kunnskap for en bedre verden

Aanensen, Eskild Jonatan Krumsvik; Berthelsen, Patric André; Henriksveen, Sigurd Ranheim; Kristiansen, Jacob

Prototype av sky-tilkoblet målesystem for EAQI, basert på nRF9160 SiP med energihøstingssystem

Prototype of a cloud-connected monitoring system for EAQI, based on the nRF9160 SiP with an energy harvesting system

Bacheloroppgave i Elektroingeniør (FTHINGEL)
Veileder: Arne Midjo
Mai 2021

Norges teknisk-naturvitenskapelige universitet
Fakultet for informasjonsteknologi og elektroteknikk
Institutt for elektroniske systemer

NTNU
Kunnskap for en bedre verden

NTNU

Institutt for elektroniske systemer

Bacheloroppgave

Oppgavens tittel: Prototype av skytilkoblet målesystem for EAQI, basert på nRF9160 SiP med energihøstingssystem **Project title:** Prototype of a cloud-connected monitoring system for EAQI, based on the nRF9160 SiP with an energy harvesting system	**Gitt dato:** 15. Desember 2020 **Innleveringsdato:** 25. Mai 2021 **Gradering** [ ] åpent [ ] lukket [ ] åpnet fra _____ **Antall sider/bilag:** 97 sider
Gruppedeltakere: Jacob Kristiansen Patric André Berthelsen Sigurd Ranheim Henriksveen Eskild Jonatan Krumsvik Aanensen	**Veileder internt (navn/email):** Arne Midjo *arne.midjo@ntnu.no*
Studieretning: Elektronikk	**Prosjektnummer:** E2114
Oppdragsgiver: Nordic Semiconductor	**Kontaktpersoner hos Oppdragsgiver (navn/email):** Endre Rindalsholt *endre.rindalsholt@nordicsemi.no* Jon Helge Nistad *jon.helge.nistad@nordicsemi.no*

Sammendrag Denne rapporten handler om en luftkvalitets sensornode og skytjenesten som henter, lagrer og viser dataene til brukeren. Vi bruker LwM2M standarden for å kommunisere med de forskjellige nodene, som er basert på Thingy:91.

Abstract This thesis contains our report on an air quality measuring sensor node. And a cloud-service run by ourselves. It retrieves, stores, and shows the data to the user. We're using the LwM2M standard to communicate with the different nodes, which are based on Thingy:91.

stikkord: IoT, LwM2M, AQI, Thingy:91, LTE-M, Energihøsting, PV	Keywords: IoT, LwM2M, AQI, Thingy:91, LTE-M, Energy harvesting, PV

Sammendrag

Internet of Things (IoT) teknologi kan være et sentralt hjelpemiddel for å løse problemer som måling av luftkvalitet over urbane områder, spesielt med tanke på trådløse sensornettverk. I den anledning bestemte vi oss for å lage en målestasjon for luftkvalitet med et energihøsting-system, som kan fungere som en sensornode i et slikt nettverk. Vi skulle også verifisere den måledataen vi fikk. Vi fokuserte på å måle EAQI, som baserer seg på fem faktorer: PM_{10}, $PM_{2,5}$, NO_2, O_3 og SO_2. Systemet er basert på Thingy:91 fra Nordic Semiconductor, som er et utviklingskort bygd rundt mikrobrikken nRF9160. Energihøsting-systemet benytter seg av to solcellepanel på totalt 6W som går til en MPPT-kontroller, SPV1050. Denne er parallellkoblet med batteriet på Thingy:91, som er på 5,18Wh. Vi har også designet og 3D-printet en egen kapsling for prototypen. Vi benytter oss av LwM2M-client eksemplet fra Nordic, og har selv utviklet driverne for sensorene for luftkvalitet, *Sensirion SPS30* og *Spec Sensor DGSO3*. Skytjenesten er basert på en Intel NUC som maskinvare, Leshan som LwM2M implementasjon, PostgreSQL som database og Grafana som dashboard. I løpet av oppgaven støttet vi på flere problemer som førte til at noen av resultatmålene ikke ble oppfylt. Vi antar at systemet har mulighet til å opprettholde seg gjennom mesteparten av året, med unntak av mørketiden, men dette må bekreftes med videre testing. Basert på en rekke statistiske tester mellom måledataen som vi samlet inn, og med måledata fra NILU, så konkluderer vi med at vår måledata ikke er valid. Videre anbefaler vi å ikke bruke Thingy:91 til en slik oppgave, mest grunnet mangel på tilgjengelige kontakter. Her kunne vi ha økt størrelsen på batteriet, tidsintervallet mellom hver måling og valgt sensorer med passende forsyningsspenning. I forhold til protokoller og skytjeneste så er det kanskje bedre å bruke protokoller som CoAP/MQTT enn LwM2M, alt etter kompleksiteten på sensorsystemet. Det er også mulig å finne en annen løsning for hosting av servertjeneste, som for eksempel AWS eller Azure.

Abstract

Internet of Things (IoT) technology can be a key aid in solving problems such as measuring air quality over urban areas, especially with regards to wireless sensor networks. On that occasion, we decided to create an air quality measuring station with an energy harvesting system, which can act as a sensor node in such a network. We also had to verify the measurement data we received. We focused on measuring EAQI, which is based on five factors: PM_{10}, $PM_{2.5}$, NO_2, O_3 and SO_2. The system is based on Thingy:91 from Nordic Semiconductor, which is a development board built around the microchip nRF9160. The energy harvesting system uses two solar panels of a total of 6W which go into an MPPT controller, SPV1050. This is connected in parallel with the battery on the Thingy:91, which is around 5.18Wh. We have also designed and 3D-printed our own enclosure for the prototype. We used the LwM2M-client example from Nordic, and have developed the drivers for the air quality sensors: *Sensirion SPS30* and *Spec Sensor DGSO3*, ourselves. The cloud service is based on an Intel NUC as the hardware, Leshan as the LwM2M implementation, PostgreSQL as the database and Grafana as the dashboard. During the thesis, we encountered several problems that led to some of the performance targets not being met. We assume that the system has the ability to sustain itself through most of the year, with an exception of the longest winter nights, but this has to be confirmed with further testing. Based on a number of statistical tests on the data that we collected, and with the data from NILU, we concluded that our data is not valid. Further on we recommend to not use the Thingy:91 for such a task, mostly based on it's lack of available pins. Here we could have increased the size of the battery, increased the time interval between each measurement and we could have chosen sensors with the appropriate supply voltage. In regards to the protocols and cloud service, it might be better to use protocols such as CoAP/MQTT rather than LwM2M, depending on the complexity of the sensor system. It is also possible to find a different solutions for server hosting, such as AWS or Azure.

Annerkjennelser

Vi vil gi stor takk til Arne Midjo (NTNU), Endre Risdalsholt og Jon Helge Nistad fra Nordic Semiconductor som har veiledet oss gjennom prosjektet. De har gitt god tilbakemelding og hjulpet oss med å forme oppgaven til som den er i dag. Cuong Phu Le har gitt mye nyttig informasjon angående energihøstingsdelen av oppgaven. Vi vil også gi takk til Karina Ødegård fra SINTEF Norlab for veiledning og god info rundt temaet luftforurensning.

Innhold

Figurer

Tabeller

Kodelister

Akronymer

I^2C Inter Integrated Circuit. 4, 11, 12, 23–26, 42, 58

V_{EOC} End Of Charge. 8, 18

V_{OC} Open Circuit Voltage. 7

V_{UVP} Under Voltage Protection. 8, 18

W_P Peak Watt. 7, 17, 18

API Application Programming Interface. 27, 32

CAD Computer-Aided Design. 9

CoAP Constrained Application Protocol. i, ii, 13, 14, 33, 54, 55

CoAPS Constrained Application Protocol Secure. 33

CTS Clear to Send. 11

DK development kit. ix, 23, 30

DNS Domain Name System. 33, 54

DTLS Datagram Transport Layer Security. 14, 21, 31, 32, 55

EAQI European Air Quality Index. 4, 5, 37

EEA European Environment Agency. 4

FIFO First in-first out. 27

FOTA Firmware Over The Air. 3, 21, 31, 54, 55

GnuDIP Gnu Dynamic IP. 33

GPIO General-purpose input/output. 17, 26

HTTP Hypertext Transfer Protocol. 13, 14, 55

IMEI International Mobile Equipment Identity. 21, 32

Impact PLA High-grade Polyactic Acid. 41

IoT Internet of Things. i, ii, 1, 2, 12, 14, 58

IP Internet Protocol. 21, 32, 33, 54

JDBC Java Database Comunication. 15, 34, 43

JDK Java Development Kit. 31

LPWAN Low Power Wide Area Network. 12

LTE-M Long Term Evolution for Machines. 1, 4, 12, 33

LwM2M Lightweight Machine to Machine. i, ii, v, 14, 21, 25, 26, 28, 29, 31–33, 42–45, 53–56, 60

MPPT Maximum Power Point Tracking. i, ii, 3, 7, 17, 18, 21, 42, 45, 47, 53, 56

MQTT Message Queueing Telemetry Transport. i, ii, 13, 14, 54, 55

NB-IoT Narrow-Band Internet of Things. 1, 4, 12, 33

NILU Norsk institutt for luftforskning. i, ii, 6, 37, 39, 49, 54, 57, 60, 63

NUC Next Unit of Computing. i, ii, 31

OMA Open Mobile Alliance. 14

PETG Polyethylene Terephthalate Glycol-modified. 9, 41

PLA Polyactic Acid. 9, 41

PMIC Power Management IC. 56

ppb Parts per billion. 27, 28

PPK2 Power Profiler Kit II. 8, 20, 21, 43, 59

PSM Power Saving Mode. 12, 33

PV Photovoltaic. 7, 8, 18

PVGIS Photovoltaic Geographical Information System. 7, 17

PWM Pulsbreddemodulasjon. 4

RAT Requested Active Time. 13

RTOS Real Time Operating System. 10, 23

RTS Ready to Send. 11

RX Motta. 30

SIM Subscriber Identification Module. 33

SiP System-in-Package. 4

SPI Serial Peripheral Interface. 4

SQL Structured Query Language. 15, 33, 34

SSH Secure Shell. ix, 32

TAU Tracking Area Update. 12, 13

TCP Transmission Control Protocol. 14

TEG Thermoelectric Generator. 8

TX Sende. 30

UART Universal Asynchronous Receiver-Transmitter. 4, 11, 23, 27–29, 42, 52, 58

UDP User Datagram Protocol. 13, 14

URI Universal Resource Identifier. 14

VLAN Virtual Local Area Network. 32

Ordliste

bowden tube Rør med lav friksjon på innside, hvor filament går til varmeelement. 9

callback Funksjon som blir kalt av programvaren ved en hendelse. 29

CoAP enklere version av HTTP for Constrained Devices. 33

CSV Komma-separert filformat for tabelldata. 38

Daemon En bakgrunns prosess/service som kjører uten bruker kontroll. 31, 36, 54

Debian Populær stabil GNU/Linux distrubusjon for servere. 36

Devzone Nordic Semiconductor sitt eget nettforum for spørsmål rundt Nordic sine produkter. 59

dupont Dupont-konnektor er mye brukt i kabler til Arduino og lignende. 21

dyse Den smelta plastikken renner gjennom dysen, som da bestemmer bredden til hver plasikkstreng. 9

FDM Fused Deposition Modeling beskriver fabrikasjonsmetoden en 3D-printer bruker. FDM trykker og smelter en kontinuerlig streng av plast lagvis. 9

G-code G-code er det mest anvendte for Numerical control(NC), programmeringsspråk. G-code blir brukt i *computer-aided manufacturing* til å styre automatiserte maskinverktøy, slik som f.eks CNC. 9

Grafana Åpen-kildekode dashbord for visning av data. kobler seg enkelt til PostgreSQL databaser med timeseries.. i, ii, 32, 33, 36, 43, 56

hotend Varmeelement til 3D-printer, smelter filament. 9

Intel NUC Veldig små barebone PC'er fra Intel. 54

interrupt callback Funksjon som kalles når en forstyrrelse ("interrupt") oppdages. 27, 28

irradians Solens innstrålingstetthet på et objekt, målt i W/m^2. 19

Java Programmeringsspråk lagd for backend programmering. 54

Leshan laget av Eclipse i Java bygget på Californium. 14, 21, 31, 32, 34, 54

levelshifter Spenningsregulator. Brukes ofte for å sammenkoble to forskjellige enheter som bruker forskjellige kommunikasjonsprotokoller. 17, 26, 30

low-power standby Enheten går i søvnmodus og bruker lite strøm. 27

LwM2M IoT applikasjons Protokoll for standardiserte ressurser. 14

port forward Rute internett trafikk til riktig datamaskin. 33, 54

PostgreSQL et database språk, utvidelse av SQL. i, ii, 15, 33, 34, 36, 43

Python Script språk ment for backend programmering. 54

RTOS lett operativ system for enheter med lite minne. xiv

Slicer verktøy Slicer verktøy, et program som gjør om stl filer til gcode. Et program som Cura gjør dette og gir et visuelt bilde av det. 9

STL STereoLithography eller *Standard Triangle Language*, et filformat brukt i CAD som kun beskriver overflate geometrien til et tredimensjonelt objekt vha triangler. 9

Tec7 Lim-, tette- og fugemasse brukt i byggebransjen. 40

Thingy:91 enhet for WAN IoT. i, ii, ix, 3, 17, 18, 20–23, 26, 27, 29, 30, 32, 44–46, 52, 53, 56–58, 60

thread Funkjsonstråd: Seperat prosessflyt i et operativsystem. Ofte brukt i *RTOS*. 27

UART Hardware protokoll for asynkron seriell kommunikasjon. 27

Volatile Organic Compounds Organiske kjemikalier med høyt damptrykk ved romtemperatur. 17

warping Vridning under printing. Brukes til å beskrive at plastikk krymper når den kjøles ned etter smelting, bør unngås. 9

Zephyr linux basert RTOS for IoT enheter skrevet i C. 34

Zerotier En VLAN (Virtual Local Area Network) tjeneste. 32

Kapittel 1

Introduksjon

1.1 Bakgrunn

I løpet av høsten 2020 kontaktet vi Nordic Semiconductor med tanke om å skrive en oppgave hos dem. Etter et par møter, kom vi fram til et oppgaveforslag gitt av veilederne våre hos Nordic: Endre Rindalsholt, og Jon-Helge Nistad. Her fikk vi en oppgave rundt å bygge en værstasjon basert på mikrobrikken nRF9160. Denne skulle blant annet sende data til en skytjeneste, og den skulle også ha et energihøsting-system som gir enheten nok strøm ut sin levetid. Senere i forprosjektet la vi til innsamling av luftkvalitetsdata. Dette var for å øke kompleksiteten til oppgaven, og bygge en prototype som inneholder potensielt nyttig teknologi.

1.2 Motivasjon

Internet of Things (*IoT*) blir et stadig mere populært tema innen elektronikk og IT-bransjen. Spesielt innen områder som *smarthjem* og sensornettverk, der det stadig blir utviklet tjenester og protokoller for å koble sammen et økende antall komplekse enheter. Det er her nettverksttandarder som *LTE-M* og *NB-IoT* kommer inn. Disse standardene gjør det enklere for enheter å sende data til en skytjeneste ved hjelp av mobilnettet. Dette muliggjør blant annet større avstander mellom plasseringer av noder i et sensornettverk, og gir generelt en enhet muligheten til å koble seg til nettet hvor enn det er dekning for mobilt bredbånd.

I 2015 ble det utgitt en rapport fra *Institute for Transformative Technologies (ITT)* som legger fram 50 viktige vitenskapelige og teknologiske gjennombrudd som må skje for å oppnå FN sine bærekraftsmål [ITT 2019]. En av disse gjennombruddene omhandlet å bygge nettverk av billige distribuerte målesystem for å kartlegge luft- og vann-kvalitet. Her står *IoT* teknologi sentralt for å løse et slikt problem. Der et sensornettverk-system som er billig å bygge og implementere, og bruker lite energi, kan være med å kartlegge de mest utsatte områdene for luftkvalitet. Dette kan være med å legge grunnlaget for en videre prosess som kan øke livskvalitet i disse områdene.

1.3 Problemstilling

Problemstillingen til denne oppgaven, som beskrevet i forprosjektrapporten låter som følger:
"*Design og implementere en løsning for en værstasjon som kan måle værdata og luftkvalitet, basert på mikrobrikken fra Nordic Semiconductor, nRF9160, med et energihøsting-system og tilkoblet skytjeneste. Deretter verifiser måledataene dere får [Aanensen mfl. 2021].*"

Ut i fra dette så delte vi effektivt oppgaven i to deler:

1. "*Lag prototyp av målesystem/værstasjon*
   (9160 m. LTE-M/NB-IoT, energihøsting, sensorer, skytjeneste)
2. *Verifiser måledata*"

1.4 Resultatmål

I forhånd av prosjektet ble det utviklet et sett med resultatmål som er beskrevet i forprosjektrapporten. Prosjektmanualen sier at "*Resultatmålet beskriver hva som konkret skal foreligge som resultat når prosjektet er ferdig [NTNU 2021].*". Basert på dette kom vi fram til punktene under:

Resultatmål

Produktets kvaliteter

- Måle værdata, som temperatur, lufttrykk, fuktighet osv.
- Luftkvalitet
  - NO_2
  - $PM_{0,5-10}$
  - Ozon
- Energihøsting
  - Solpanel som lader batteri i *Thingy:91*, for lengre levetid
  - Ladekontroll, gjør om spenning fra solpanel til regulert ladespenning
- Skytjeneste
  - Serverinfrastruktur
  - Database
  - Dashboard
  - *Firmware Over The Air* (*FOTA*)

[Aanensen mfl. 2021]

1.5 Disposisjon

Rapporten er delt inn i fem deler: Teori, metode, resultat, diskusjon, og konklusjon. Det første er et teoretisk grunnlag som trengs for å forstå rapporten. I metodedelen går vi detalj rundt hvordan oppgaven ble utført, samt valgene vi tok i løpet av prosjektet. Resultatet beskriver den ferdig prototypen, og viser tester av *Thingy:91*, *MPPT* og verifiseringstester på måledataen. Diskusjonen forklarer hva som virket og hva som kunne blitt forbedret. Til slutt kommer konklusjonen som sier noe om utfallet av oppgaven.

Kapittel 2

Teori

2.1 nRF9160

nRF9160 fra Nordic Semiconductor er en todelt *System-in-Package* (*SiP*), med integrert *LTE-M/NB-IoT* modem, og en ARM Cortex-M33 applikasjonsprosessor som er bygd rundt lav effekt. Den har også utganger for grensesnitt og periferier som: *PWM*, *SPI*, *UART*, og I^2C. Den blir brukt i utviklingskort som nRF9160DK, og Thingy:91 [Nordic 2021a].

2.2 EAQI

European Air Quality Index (*EAQI*) er et verktøy for bedre forståelse rundt luftkvalitet i Europa. Indeksen ble utviklet av *European Environment Agency* (*EEA*).

2.2.1 Målinger

EAQI baserer seg på fem faktorer for luftkvalitet, som er:

PM_{10}	Partikler mindre enn 10µm
$PM_{2,5}$	Partikler mindre enn 2,5µm
NO_2	Nitrogendioksid-gass
O_3	Ozon-gass
SO_2	Svoveldioksid-gass

Partikkelmålingene ($PM_{2,5}$, og PM_{10}) bruker et 24 timers gjennomsnitt. Med 18 ut av 24 målinger vil målingen bli regnet ut til et gjennomsnitt og godkjent.
For gass (NO_2, 0_3, og $S0_2$) er måleverdier fra hver time brukt. Alle verdier i *EAQI* bruker enheten $\mu g/m^3$, altså mikrogram per kubikkmeter.

Krav

Avhengig av om det er en trafikk-, industri- eller bakgrunns-målestasjon, så er kravene for hva som må måles eller modelleres forskjellig for å få den offisielle indeksen kalkulert.

For industri og bakgrunnsstasjoner er det bare nødvendig å måle O_3, NO_2, og en type partikkelmåling ($PM_{2,5}$, eller PM_{10}). For trafikkmålestasjoner ser man på hvilken verdier som blir påvirket av veitrafikk. Her kreves det da målinger av partikler og NO_2. SO_2 blir ikke tatt med siden det ofte er stor variasjon i lokale omstendigheter. Dette kan derfor gi et feil bilde av luftkvaliteten. Ozon trengs ikke, siden den ikke blir påvirket av kjøretøy [EEA 2021].

Manglende data

Ved manglende data kan en fylle noen gap med bruk av forskjellige metoder, avhengig av hva en måler:

- For NO_2, $PM_{2,5}$, og PM_{10} bruker man differansen mellom målingene.
- For 0_3 så kan man bruke multiplikasjon av målingene.
- Det er ingen metode som brukes for SO_2, siden den ikke er nødvendig.

2.2.2 Helserisiko

Årsaken til at luftkvalietet blir målt er at høye verdier ofte fører til økt helserisiko, spesielt for de med pustevansker eller hjerteproblemer. Risikoen for $PM_{2,5}$, O_3, og NO_2 er basert på WHO sin rapport om helserisiko av luftforurensning i Europa som ble utgitt i 2013 [WHO 2013]. Verdiene for PM_{10} er satt som dobbelt så høye, i forhold til verdier for $PM_{2,5}$, etter retningslinjene fra WHO rundt luftkvalitet [WHO 2006]. Til slutt er SO_2 grensen satt under EUs "*Air Quality Directive*" [EUR-Lex 2008].

EAQI

EAQI baserer seg på verste verdi. Hvis en verdi er for høy mens resten er lave, vil det fortsatt vises som dårlig luftkvalitet. Siden påvirkning over lang tid er farligere, er det derfor strengere krav for årlige gjennomsnitt enn kortsiktige verdier. Høyere verdier enn de i *"extremely poor"* blir ikke tatt hensyn til siden de oftest er ansett som feilverdier. [EEA 2021].

Pollutant	Good	Fair	Moderate	Poor	Very poor	Extremely poor
$PM_{2,5}$	<10	<20	<25	<50	<75	<800
PM_{10}	<20	<40	<50	<100	<150	<1200
NO_2	<40	<90	<120	<230	<340	<1000
O_3	<50	<100	<130	<240	<380	<800
SO_2	<100	<200	<350	<500	<750	<1250

Tabell 2.1: Tålelige verdier tabell, målt i $\mu g/m^3$ [EEA 2021]

Sensitiv populasjon inkluderer personer med puste eller hjerte problemer.

AQ index	General population	Sensitive populations
Good	The air quality is good. Enjoy your usual outdoor activities.	The air quality is good. Enjoy your usual outdoor activities.
Fair	Enjoy your usual outdoor activities	Enjoy your usual outdoor activities
Moderate	Enjoy your usual outdoor activities	Consider reducing intense outdoor activities, if you experience symptoms.
Poor	Consider reducing intense activities outdoors, if you experience symptoms such as sore eyes, a cough or sore throat	Consider reducing physical activities, particularly outdoors, especially if you experience symptoms.
Very poor	Consider reducing intense activities outdoors, if you experience symptoms such as sore eyes, a cough or sore throat	Reduce physical activities, particularly outdoors, especially if you experience symptoms.
Extremely poor	Reduce physical activities outdoors.	Avoid physical activities outdoors.

Tabell 2.2: Helse meldinger

2.2.3 Kilder

I Norge er det flere kilder som kan ha en påvirkning på luftkvaliteten. *NILU* betrakter NO_2 og svevestøv ($PM_{2,5}$ og PM_{10}) som de viktigste forurensningskomponentene i norske byer [NILU 2021b]. Her er veitrafikk, eller eksos, regnet som primærkilden til blant annet NO_2. $PM_{2,5}$ kommer for det meste fra forbrenning (for eksempel: vedfyring, bileksos), mens de større partiklene (PM_{10}) kommer for det meste fra vei og dekkslitasje. De mindre partiklene kan transporteres over lengre avstander med luftmasse, som kan bidra betydelig til konsentrasjonen av $PM_{2,5}$.

2.3 Energihøsting

2.3.1 Solcellepanel

Solcellepanelene fra [Seeed-Studio 2017] er laget av monokrystalline celler, med tynt lag av resin for beskyttelse og vanntetting av cellene. Terminalene på panelet er loddet til kabler som er utstyrt med en 2mm JST kontakt.

- Effektivitet = 16%
- Spenning ved Peak Power = 5,5V
- Strøm ved Peak Power = 540mA
- *Open Circuit Voltage* (V_{OC}) = 8,2V

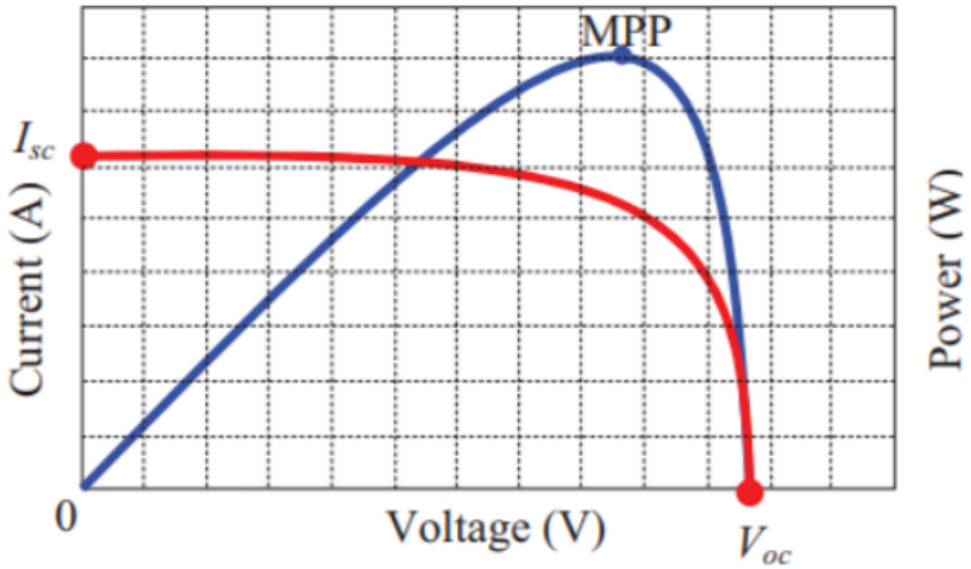
Solcelle, også kjent som *Photovoltaic* (*PV*) -panel, lar oss høste det solen stråler ut (*radians*). Det vi mottar kalles for *irradians* eller innstrålings-tetthet. *Irradians* måles i W/m^2 (Landsberg og Sands 2011). *PV* panel gjør om denne energien til en likestrøm (*DC*), som kan lagres på et batteri.

2.3.2 PVGIS

Photovoltaic Geographical Information System (*PVGIS*) er et nettbasert verktøy for kalkulering av hvor mye energi man får ut av et energihøsting-system. De bruker sol- og værdata samlet opp av satellitter over flere år, for å gi et estimat over energien man kan få over forskjellige geografiske områder. Kalkulatoren mates med info om batterikapasitet, *Peak Watt* (W_P) til solcellepanel, halvledertype til solcellepanel, og posisjonering av panel. Den gir også historisk data og estimert irradians ved forskjellige årstider [European-Commision 2021].

2.3.3 MPPT

Maximum Power Point Tracking (*MPPT*) går ut på å høste mest mulig energi ved riktig strøm-spenning forhold [STMicroelectronics 2018]. Et eksempel på dette er illustrert i figur 2.1. Her gir solcellepanelet ut en spenning V_{OC} på 8,2V ved åpen krets, og strøm lik null. Det gylne punktet for strøm-spenning forholdet kan være når solcellepanelet gir $100mA$ ved $3V$.



Figur 2.1: Maximum Power Point [Le 2020]

Spesifikasjoner - Evaluation board for SPV1050:

- First startup at $V_{in} = 500mV$
- Input voltage working range: $500mV \leq V_{in} \leq V_{EOC}$
- End of charge battery voltage: V_{EOC}= 4,25V
- Battery undervoltage protection: V_{UVP}= 3,7V

[STMicroelectronics 2014]

Evaluation board *STEVAL-ISV020V1* benytter mikrobrikken SPV1050 (STMicroelectronics 2014). Den er spesiallaget for å brukes med *PV*-panel, og kan også brukes med *Thermoelectric Generator* (*TEG*). Mikrobrikken benytter seg av MPPT algoritmen for å høste mest mulig energi. Inneholder to LDO-output på 1,8V og 3,3V, for eventuelle andre mikrobrikker og kretser. Evaluation board kutter lading til batteri ved *End Of Charge* (V_{EOC}) = 4,25V.

2.3.4 PPK2

Power Profiler Kit II (*PPK2*) er et produkt fra Nordic Semiconductor som brukes til strømmåling. Den kan logge opp mot 500 dager, og måle fra 200nA til 1A. Enheten brukes sammen med et tilhørende program kalt nRF Connect - Power Profiler. Den kan settes i *Source meter* og *Ampere meter*. Under *Source meter* brukes den også som en strømforsyning, hvor man kan justere spenningsnivået fra 0,8-5V. Ved *Ampere meter* funker den som en strømmåler. *PPK2* har også åtte digitale logikk porter. Se [Nordic 2021b] for mer info.

2.4 CAD

I prosjektet ble Autodesk Fusion 360 benyttet for å lage en *Computer-Aided Design* (*CAD*) modell rundt prototypen. Fusion 360 er et 3D modellering program som har flere verktøy for å utforme og sette sammen komplekse konstruksjoner. Når man designer i Fusion 360 så starter man i x-,y- og z-planet med en rektangulær figur. Deretter *extruder* man, eller drar ut fra dette planet. Man sitter da igjen med en kube. Samme gjelder for sirkel, der man drar denne ut slik at det blir en sylinder. Deretter bygger man videre med disse figurene på hverandre, til man sitter igjen med ønsket 3D-modell. Dette er grunnprinsippet for å designe i et *CAD* program som Fusion360. Når 3D-modellen er ferdig, så kan man eksportere modellen til en fil i *STL* format som et *Slicer verktøy* kan håndtere og lage *G-code*. *G-code* er kommandoene 3D-printere bruker for å lagvis smelte plastikk over hverandre. Slik at man til slutt sitter igjen med 3D-modell. *G-code* bestemmer også temperatur, hastighet, kjølig fra vifter, matehastighet og lignende.

2.4.1 3D-printer

Creality Ender-3 Pro er en lavkostnad *FDM* printer, som har en printflate på 22x22x25cm. Printeren er egnet for å printe *PLA* filament opp mot 230°C. Ved bruk av andre filament som krever høyere varme, anbefales det å oppgradere *hotend* og *bowden tube* [Polyalkemi 2019].

2.4.2 Filament

PLA PolyLactic Acid, er en bioplast brukt til 3D-printing, med smeltepunkt på rundt 180-210°C. Filamentet degraderes fortere når den blir utsatt for sol.
Egenskaper til *PLA* er at den relativt til andre har lavt smeltepunkt, lite *warping* og lite friksjon i *dyse* ved 3D-printing. *PLA* er det mest brukte filament innenfor 3D-printing, noe som gjør at den er høyt tilgjengelig.

Impact PLA er en sterkere variant av vanlig *PLA*, men som er nesten like printevennlig. Den har et høyere smeltepunkt på rundt 200-235°C, som gjør at den takler høyere belastninger etter print [Polyalkemi 2021a].

PETG *Polyethylene Terephthalate Glycol-modified* (*PETG*) er en termoplastisk copolyester. Den er mer fleksibel og tåler UV-lys bedre enn andre filament. *PETG* er glycol-modified, noe som gjør at den kan brukes som filament til 3D-printing. Der hvor *PLA* er sterk, så er *PETG* mer fleksibel og mindre sprø, som kan være en større fordel når man lager prototyper [Polyalkemi 2021b].

2.5 Firmware

2.5.1 RTOS

Real Time Operating System (*RTOS*) er en type operativsystem som er laget for sanntidsapplikasjoner [FreeRTOS 2021].

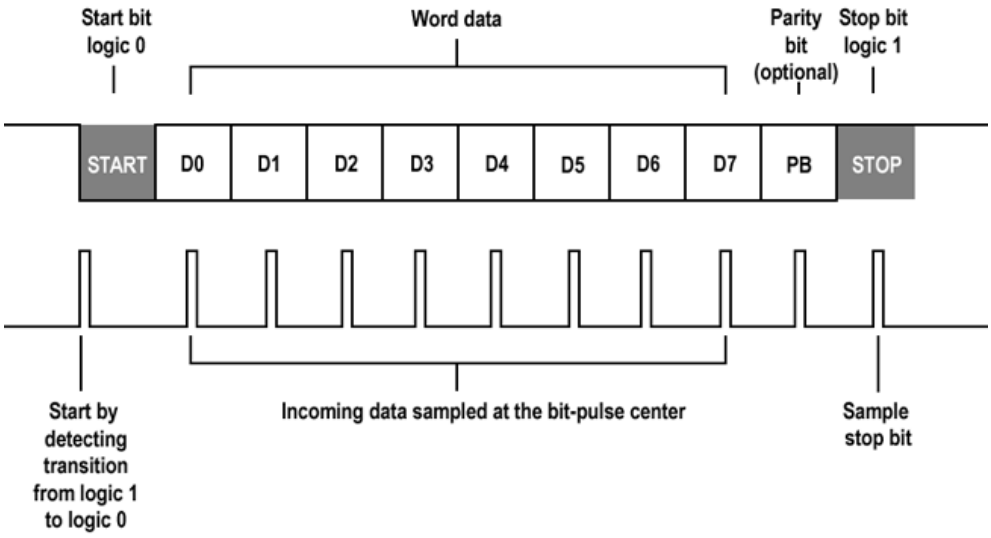
Et *RTOS* opererer på en måte som gjør at vi får en deterministisk oppførsel fra operativsystemet. Dette vil si at alle operasjoner er designet for å være forutsigbar i forhold til prosessortid. Typisk så skjer dette ved at hver prosesstråd får sin egen prioritet, som er bestemt av brukeren, der operativsystemet benytter denne prioriteten til å bestemme hvilken tråd som skal kjøres.

2.5.2 Zephyr RTOS

Zephyr Project er et samarbeidsprosjekt støttet av Linux Foundation, med flere kjente medlemmer som blant annet Facebook, Google, Intel og Nordic Semiconductor [Project 2020].

Zephyr *RTOS* stammer fra dette samarbeidsprosjektet, og er en av de mest populære åpen kildekode sanntidsoperativsystemene i verden, sammen med FreeRTOS.

2.5.3 UART



Figur 2.2: *A UART frame* (illustasjon) [Electricimp 2021]

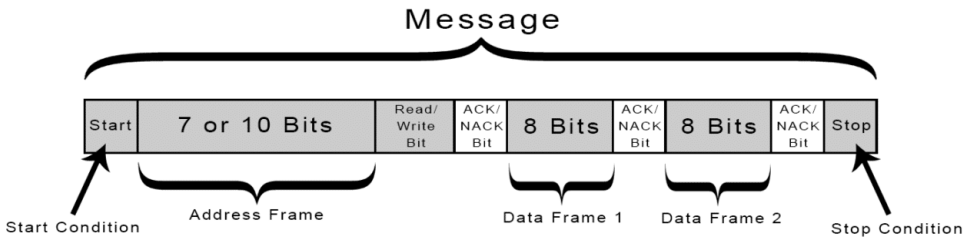
Universal Asynchronous Receiver-Transmitter (UART) er en hardware protokoll for asynkron seriell kommunikasjon.
Protokollen har konfigurerbart dataformat og hastighet. De elektriske signalene er håndtert med et driver system som ligger eksternt for *UART* systemet.

En *UART* pakke (eller "*frame*") består typisk av 10-11 bits, der pakken begynner med en start bit (vanligvis dratt lav "0"), følgt av 8-9 bits der siste bit typisk fungerer som paritet for feilsøking. Dette er etterfulgt av en stop bit (vanligvis dratt høy "1").

UART har i tillegg en metode for å kommunisere mellom trege og raske enheter uten tap, som kalles for "*flytkontroll*" ("*flow-control*").
Denne funksjonen er en del av RS232 standarden. Her legges det til to linjer på enheten, *Ready to Send* (*RTS*) *Clear to Send* (*CTS*). Disse to linjene gjør det mulig for hver enhet å gi beskjed til den andre om sin egen tilstand.

I forhold til konfigurerbarhet, så kan også hastigheten (*baudrate*), antall paritet bit, antall stopp bit, antall data bit og flytkontroll konfigureres på de tilkoblede enhetene.

2.5.4 I2C



Figur 2.3: *A I2C frame* [illustasjon] (Circuitbasics 2016)

I^2C (*«Inter-Integrated Circuit»*) er en protokoll for å kommunisere med flere enheter [NXP 2014]. Kommunikasjonen blir mye brukt for å koble sammen lavhastighets integrerte kretser til prosessorer og mikrokontrollere på kort avstand.

I^2C benytter seg av et master-slave system. Dette gjør at master-enheten har mulighet til å kontrollere en eller flere slave-enheter. Ved oppkobling benytter man seg av to forskjellige linjer, *SDA*, og *SCL*. *SDA* står for «serial data», dette er linjen der man sender eller mottar data på. SCL står for «serial clock». Denne linjen brukes for å sende klokkefrekvensen, slik at dataoverføringen kan bli synkronisert mellom de to enhetene.

Siden I^2C er en seriell kommunikasjon protokoll, blir data overført bit for bit. Klokkesignalet blir alltid kontrollert av master enheten. Når data skal sendes blir en adresse først sendt ut fra masteren. Adressen bestemmer hvilken enhet som man skal styres og er som regel 7 bit eller 10 bit. Deretter sendes det en bit som sier noe om data skal leses eller sendes. Til slutt sendes datarammen, som alltid sendes i 8-bits pakker med ACK/NACK ("*Acknowledged*", "*Not acknowledged*") bit mellom for å verifisere data.

2.6 Server

2.6.1 LTE-M

Long Term Evolution for Machines (*LTE-M*) er en *Low Power Wide Area Network* (*LPWAN*) mobilnett protokoll, designet for *IoT* enheter. Den har høyere hastighet enn *NB-IoT*, men passer også for batterisparing [Ligero 2018].

PSM

Batterilevetiden til enheter kan utvides med *Power Saving Mode* (*PSM*). *PSM* er en modus som gjør det mulig å skru av radioen, uten å måtte registrere seg på nytt når enheten kobler seg til nettet igjen. Dette gjør det mulig å ha enheter av i lange perioder, men raskt sende data når den våkner og skrus av, uten å måtte vente lenge og uforutsigbart for å registrere seg på nytt.

***Tracking Area Update* (*TAU*)** skjer normalt når enheten skifter fra en mobil basestasjon til en annen, eller når den bytter mellom 2G, 3G, og LTE [Do 2013]. Periodisk *TAU* er måten som er brukt i *PSM*. Det skjer ved faste tidspunkt satt av klienten, ved å spørre basestasjonen. Tiden blir satt ved å sende en byte med tiden enheten skal være i *PSM*. Formatet til byten er slik: Første tre bit er hvilke tidsinkrement som brukes, og siste fem er vanlig binær verdi i den størrelsen [Nordic 2021c].

000 = 10 minutter
001 = 1 time
010 = 10 timer
011 = 2 sekunder
100 = 30 sekunder
101 = 1 minutter
110 = 320 timer
111 = deaktivert

***Requested Active Time* (*RAT*)** er tiden enheten skal være påkoblet nettet for å sende data. *RAT* blir også satt av en byte, med samme format som *TAU*, men med litt andre koder [Nordic 2021c]:

000 = 2 sekunder
001 = 1 minutt
010 = 6 minutter
111 = deaktivert

2.6.2 CoAP

Constrained Application Protocol (*CoAP*) er en protokoll for enheter med lite minne, prosessorkraft, og nettverkhastighet. Den sender meldinger med *UDP*, eller SMS. *CoAP* er laget som en lett versjon av *HTTP*, med en liten header med størrelse på 4 byte. Siden den er lagd for å være lik *HTTP*, så deler den de samme kommandoene: GET, POST, PUT, og DELETE. Disse kan derfor lett oversettes til hverandre.

Protokollen benytter seg av et forespørsel/respons system for overføring av data. Dette er forskjellig fra blant annet *MQTT* som bruker publisering/abonnering for overføring. *CoAP* kan sende bekreftelse (*ACK*) eller ikke på *UDP*. Protokollen sender en ny melding hvis den ikke får et svar med bekreftelse.

<table>
<tr><td>Offsets</td><td>Octet</td><td colspan="8">0</td><td colspan="8">1</td></tr>
<tr><td>Octet</td><td>Bit</td><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td></tr>
<tr><td>2</td><td>16</td><td colspan="2">VER</td><td colspan="2">Type</td><td colspan="4">TokenLength</td><td colspan="8">Request/Response Code</td></tr>
<tr><td>4</td><td>32</td><td colspan="16">Message ID</td></tr>
<tr><td>6</td><td>48</td><td colspan="16" rowspan="4">Token (0 - 8 bytes)</td></tr>
<tr><td>8</td><td>64</td></tr>
<tr><td>10</td><td>80</td></tr>
<tr><td>12</td><td>96</td></tr>
<tr><td>14</td><td>112</td><td colspan="16" rowspan="2">Options (If Available)</td></tr>
<tr><td>16</td><td>128</td></tr>
<tr><td>18</td><td>144</td><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td><td colspan="8" rowspan="2">Payload (If Available)</td></tr>
<tr><td>20</td><td>160</td></tr>
</table>

Tabell 2.3: CoAP Header [Shelby, Hartke og Bormann 2014]

2.6.3 LwM2M

Open Mobile Alliance (*OMA*) sin protokoll, *Lightweight Machine to Machine* (*LwM2M*), er en *IoT* protokoll for server og klient. Den bruker *CoAP*/*MQTT*, og *TCP*/*UDP*. Den lager også standardiserte objekter og ressurser. Her kan man hente/skrive data, samt oppdatere og konfigurere innstillinger på enheten. Kommandoene som kan brukes er *read*, *write*, *delete*, *execute*, *start observe* og *stop observe*. Ved forskjellige instanser kan en ha flere av samme objekt [AVSystem 2019].

LwM2M bruker *Universal Resource Identifier* (*URI*) adresser. Formatet er likt, uavhengig av protokoll. Linjen under viser hvordan en slik adresse kan bli satt opp:
'protokoll'://'ip':'port'/#/clients/'endepunkt'/'objekt'/'instanse'/'ressurs'

Under ser vi et eksempel på hvordan adressen hadde sett ut ved bruk av *HTTP*:
http://leshan.eclipseprojects.io/#/clients/Thingy:91/

Under ser vi et eksempel på hvordan adressen hadde sett ut ved bruk av *CoAP*:
coap://leshan.eclipseprojects.io/api/clients/Thingy:91/3303/0/5700
Her har vi også lagt ved eksempel på objekt, instance og ressursverdi som kan benyttes.

2.6.4 Leshan

Leshan er en *LwM2M* server og klient som er utviklet av *Eclipse Foundation*. Programvaren er utviklet med programmeringsspråket *Java*, og er bygget på *Californium*, som er en *CoAP* implementasjon, som også er utviklet av *Eclipse Foundation*. Programvaren bruker *Apache Maven* for bygging og kontroll av avhengigheter i *Java* [Eclipse 2015].

Leshan baserer seg på en enkel objekt-basert ressurs modell. Den har operasjoner for å sette, lese, oppdatere, slette, og konfigurere ressursene. Man kan få oppdateringer og notifikasjoner når en ressurs endrer seg.

Leshan støtter flere filformater mellom serveren og klienten, deriblant: *TLV*, *JSON*, *TXT*, og *Opaque*. *Leshan* kan benytte seg av flere typer protokoller, deriblant *UDP*, og *SMS* transport. *Leshan* bruker også en type sikkerhet basert på *DTLS*. Den har også en *Kø modus*. Dette blir brukt å sette klienter i "*sovemodus*", der de ikke har konstant kommunikasjon med serveren [Eclipse 2020].

Noen av kjerneobjektene til *LwM2M*, som blir brukt av *Leshan* er: *LwM2M* sikkerhet, *LwM2M* server, aksess kontroll, enhet, tilkobling, overvåkning, firmware oppdatering, lokasjon, og tilkoblings statistikk.

2.6.5 PostgreSQL

PostgreSQL er en utvidelse av *SQL* språket, som etterkommer av *Ingress*, eller *Post Ingress Structured Query Language* (*SQL*). *PostgreSQL* legger til ekstra funksjoner, som innebygd grensesnitt for *JDBC* [PGDG 2021]. Den har en åpen kildekode og er en fritt objekt-relasjonelt databasesystem (ORDBMS). Dette betyr at hver rad i en tabell må ha en egen *nøkkel* som sier at det er en unik rad (innsettelse) og fungerer som hovedsorteringen av tabellen.
Timescale er en utvidelse til *PostgreSQL* som bruker timeseries, som baserer seg på tidspunkt og ikke id som *nøkkel*.

id	data
1	10.12
2	1.35
3	304.25

Tabell 2.4: enkel tabell

tidspunkt	data
2021-04-12 11:42:20.303927	10.12
2021-04-13 10:37:04.923033	1.35
2021-04-14 12:28:11.587236	304.25

Tabell 2.5: timeseries tabell

2.7 Verifisering

2.7.1 Pearsons korrelasjonskoeffisient

Pearons korrelasjonskoeffisient (r) brukes for å måle lineær assosiasjon mellom to variabler. Koeffisienten varierer mellom -1 og 1, der ekstremverdiene viser til enten en sterk negativ eller sterk positiv korrelasjon og 0 viser til ingen korrelasjon (Kirch 2008).

For å gi en bedre indikasjon på signifikansen på korrelasjonskoeffisienten, så kan man benytte seg av en tommelfingerregel [Krehbiel 2004]:

$$|r_{xy}| \geq \frac{2}{\sqrt{n}} \tag{2.1}$$

Her er n lik antall variabler som måles. Hvis høyresiden er større eller lik venstresiden, så kan man anta at det finnes et lineært forhold.

2.7.2 T-test

I følge [Kim 2015] så er en t-test en type statistisk test som er brukt for å sammenligne gjennomsnittet mellom to grupper. T-tester kan bli delt opp i to grupper: en uavhengig t-test, der de to gruppene som sammenlignes er uavhengig av hverandre, og en paret t-test, der de to gruppene er avhengig av hverandre.

For å designe en t-test for korrelasjonskoeffisienten, så kan man basere seg på formelen gitt av [Illowsky og Dean 2021]:

$$t = r * \sqrt{\frac{(n-2)}{(1-r^2)}} \tag{2.2}$$

Her er r korrelasjonskoeffisienten og n er antall variabler som skal sammenlignes. Basert på dette får man en t-verdi, der en enten kan finne en *kritisk* t-verdi, eller man kan finne den relevante *p-verdien*.

2.7.3 p-verdi

I en statistisk test så baserer man seg på to hypoteser:

- Nullhypotesen (H_0), som tilsier at det er ingen forskjell mellom to variabler som blir sammenlignet. Det er vanlig å motsi denne hypotesen.
- Alternativ hypotese (H_1 eller H_a). Motsier nullhypotesen. Er ofte den hypotesen som man prøver å bevise

[Illowsky og Dean 2021]

For å komme til en beslutning på hvorvidt man avkaster nullhypotesen i fordel for den alternative hypotesen så beregner man vanligvis en p-verdi. Gitt med et signifikansnivå (α) så kan man anslå en statistisk sannsynlighet for å få resultater som er minst like ekstreme som de resultatene som er observert. Det har vært vanlig å bruke et signifikansnivå på 5%, det vil si $\alpha = 0{,}05$, men denne verdien er i all hovedsak vilkårlig.

Kapittel 3

Metode

Denne delen av rapporten beskriver arbeidsprosessen rundt oppgaven, hvilke valg vi har valgt å ta, samt begrunnelse bak disse valgene.

3.1 Thingy:91

I utformingen av prosjektet kom vi fram til at *Thingy:91* hadde flere fordeler ovenfor nRF9160DK, og var derfor mer aktuell å bruke i den ferdigstilte prototypen.

Fordeler med *Thingy:91*:

- Batteri med ladesystem
- Intern gassensor (Bosch BME680) for måling av *Volatile Organic Compounds*, temperatur og fuktighet
- Mere kompakt og lettere enn nRF9160DK

Ulemper med *Thingy:91*:

- Lite tilgjengelige kontakter for tilkobling av sensor
- *GPIO* er sendt ut med lav spenning (1,8V), og trenger derfor en *levelshifter* for sensorene.

3.2 Energihøsting

Basert på strømbudsjettet vi lagde under forprosjektet, så trengte vi solcellepanel som hadde *Peak Watt* (W_P) på 4W. Disse ble bestilt ved en ekstern leverandør. Etterhvert som vi fikk testet koden vår på *Thingy:91* og samtidig logget med PPK, fant vi ut at strømtrekket var høyere enn antatt. Her måtte vi justere strømbudsjettet ved å endre måletid for sensorer og øke effekt fra solcellepanel. Vi benyttet oss av *PVGIS* til å finne nye parametre for systemet, som passet på at batteriet ikke ble tomt. Vi endte opp med 6W solcellepanel og en *MPPT* evaluation board til å forvalte energi fra solcellepanelene.

Det ble valgt å kjøpe to solcellepanel med *Peak Watt* (W_P)=3W som til sammen skulle gi oss 6W. Vi hadde ikke diskutert hvordan vi skulle koble sammen panelene, altså om vi skulle gå for serie- eller parallellkobling eller finne en annen løsning. Dette gjelder også for panelene på 4W. Etter at vi fikk bestillingene våre så kunne vi utføre noen tester. Det var viktig for vår forståelse av hvor mye strøm/spenning vi faktisk ville få. Det ble hintet tidlig av vår veileder at strømbudsjettet var mangelfullt med tanke på flere faktorer, deriblant hvor mye effekt solcellepanelene faktisk ville gi ved sol og skyfri himmel. Her erfarte vi er at den strømmen som er oppgitt ved *peak power* er vanskelig å oppnå, der det er mer en standardisering fra produsentene. Etter som vi fikk testet panelene våre i ulike solforhold, erfarte vi at ved å koble de i parallell ville det gi tilstrekkelig med spenning til at *MPPT*-en ville selvstarte med god margin, og at vi fikk mest mulig strøm fra panelene. I denne oppgaven har vi laget to prototyper, hvor den ene vil benytte seg av 4W oppsettet.

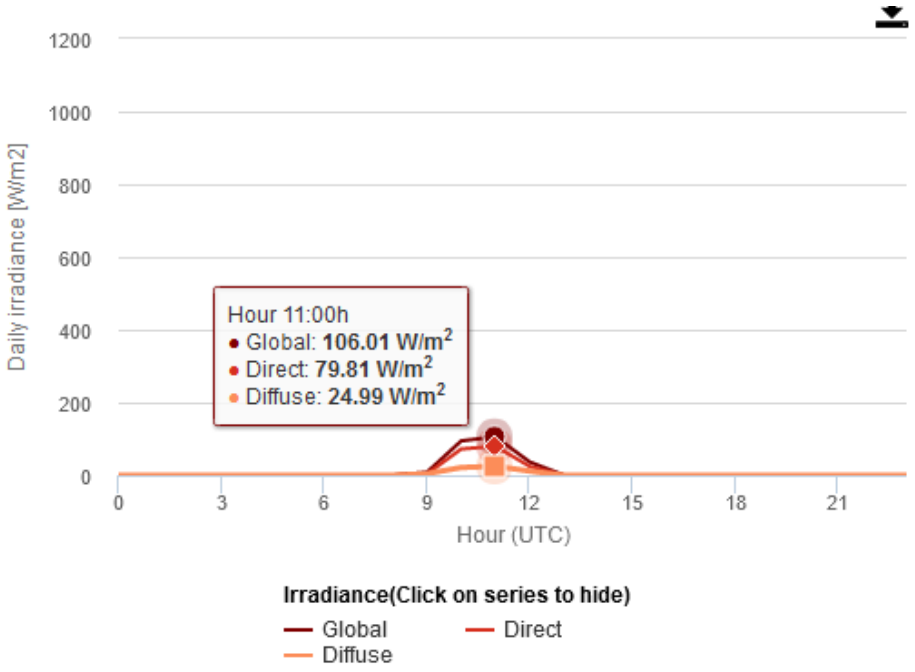
Fra forprosjektet så ble det bestemt å bestille en *buck/boost konverter*, som skulle brukes som en ladekontroller (Instruments 2019). Etter som vi tilegnet oss mere informasjon om energihøsting ved hjelp av *Photovoltaic* (*PV*)-panelene, så kunne vi slå fast at en mikrobrikke med *Maximum Power Point Tracking* (*MPPT*) ville gi oss en fin ladestrøm. Dette er i motsetning til en *buck/boost konverter* som bare vil gi ut den strømmen den får inn fra solcellepanel, som vi har erfart kan være opp mot 800mA. Dette er noe som kan være skadelig for batteriet, og kan redusere levetiden betraktelig. *Buck/boost konverter* blir nå brukt for å øke spenningen til 5V til partikkelsensoren.

3.2.1 MPPT

Vi valgte å bruke mikrobrikken SPV1050 for *MPPT* siden vår veileder fra Nordic hadde erfaring med denne, og Cuong Phu Le, vår foreleser i elektronikk og energihøsting, anbefalte SPV10xx-serien.

SPV1050 møter en del av kravene til energihøsting-systemet, og med noen endringer vil den være godt egnet for batteriet og systemet vårt. Ingen endringer til *MPPT* har blitt gjort i dette prosjektet. Her vil den starte lading av batteriet ved 3,7V, kalt for *Under Voltage Protection* (V_{UVP}), og den vil stanse lading når *End Of Charge* (V_{EOC}) når 4,25V. Batteriet er koblet i parallell til utgangen *BATT* i MPPT og *Thingy:91* sin originale kontakt for batteriet.

3.2.2 PVGIS



Figur 3.1: Desember,
(Generert ved bruk av `https://re.jrc.ec.europa.eu/pvg_tools/en/tools.html`)

Figur 3.1 viser *irradians* målt i desember. Her kan vi se hvor mye effekt vi kan forvente å få ved vårt referansepunkt, som er satt i Høgskoleparken, Gløshaugen. Vi gikk ut ifra denne årstiden i videre beregninger, siden det er da vi forventer å få minst ut av energihøstingen. Som grafen viser har vi et toppunkt for global *irradians* kl 11.00 på $106{,}01 W/m^2$ på et objekt som er 60° vinklet på horisontalplanet.

Vi har to 3W solcellepanel med dimensjoner 138x160mm hver, som gir et areal på $44160 mm^2$. Tar vi et gjennomsnitt av *irradians* på de fire timene det er dagslys, får vi:

$$\frac{(8{,}12 + 94{,}46 + 106{,}01 + 36{,}59) W/m^2}{4} = 61{,}29 W/m^2 \tag{3.1}$$

Fra dette finner vi effekt mottatt på solcellepanelet sitt areal i fire timer:

$$61{,}29 W/m^2 \cdot 0{,}04416 m^2 \cdot 4h = 10{,}79 Wh \tag{3.2}$$

Fra dette kan vi forvente å få $10{,}79 Wh$ per dag fra solcellepanelet i desember. Noe variasjon dag til dag vil det være grunnet snø og skyer, samt effektiviteten til solcellepanelene, men det gir oss et grovt overslag over det vi kan hente inn i den tidsperioden ved hjelp av energihøsting.

3.2.3 Strømbudsjett

I forprosjektet ble det utviklet et strømbudsjett som ga oss et teoretisk anslag for hvor mye de forskjellige delene av systemet ville trekke av strøm. Dette budsjettet kom ikke med i forprosjektrapporten, siden budsjettet da manglet en del beregninger som spilte en rolle senere for utviklingen av prototypen.

Budsjettet ligger som vedlegg her [A.4]. Den er delt opp i 5 deler:

- Grønn: Viser kapasitet til batteriet.
- Rød: Viser data rundt solpanel.
- Blå: Viser forskjellig teoretisk forbruk av sensorene.
- Oransje: Viser teoretisk forbruk av Thingy:91.
- Blå: Beregnet totalt forbruk.

Her er data rundt batteriet hentet herfra [Nordic 2019]. Data rundt solpanelet er basert på PVGIS-kalkulatoren [3.2.2], der alt untatt *maks forbruk* kommer fra egne beregninger. Her er *maks forbruk* [*Wh/dag*] det maksimale forbruket systemet kan ha, før den ikke klarer å forsynes av energihøsting-systemet, regnet over ett år.

Dataen rundt strømforbruk av sensorene er beregnet med formelen:

$$\frac{(I_A[mA] \cdot t_s[s] + I_s[mA] \cdot 300[s]) \cdot U \cdot 24}{1000 \cdot 3600} \tag{3.3}$$

Her er I_A strømmen som sensoren forbruker i *aktiv* modus (når den henter inn måledata), I_s er strømmen som trekkes i *standby* modus (dvalemodus), t_s er samplingstiden i sekund og U er spenningen til sensoren. Formelen beregner hvor mye strøm som trekkes både når sensoren tar en måling (i løpet av måletiden) og når den er i *standby* modus (5 min). Basert på denne formelen får vi ut et forbruk i *Wh/dag* for hver sensor.

nRF Connect - Power Profiler & PPK2

For å kartlegge strømforbruk og energihøsting har vi benyttet oss av PPKII/PPK2 fra Nordic Semiconductor. Vi så på potensialet til program og *PPK2* for å logge resultater. Ved bruk av *Power Profiler Kit II* (*PPK2*) til å logge nRF9160DK, så var det bare å følge instruksjon fra Nordic sin nettside (Nordic 2021b). Vi brukte dette til å sjekke strømforbruk til nRF9160 brikken. Deretter flashet vi vår egen programvare til *Thingy:91*, slik at vi kan sjekke strømforbruket til enheten. Ved oppkobling til *Thingy:91* måtte vi fjerne en loddebru på pad *SB3*, for så å koble *PPK2* til *P1*, samt *IN* og *OUT* kontakter til nRF9160. Siden vi skulle gjøre det her flere ganger, og vi krevde at nRF9160 fikk strøm når vi ikke utførte målinger, så valgte vi å lodde til to headerpinner til disse kontaktpunktene. Ved vanlig bruk så satte vi en jumper til disse headerpinnene, slik at *Thingy:91* fungerte som normalt.

Ved oppkobling til *PPK2* er det viktig at man kobler den i serie, og i riktig vei. Med mange forskjellige iterasjoner og målepunkter kan det være forvirrende å vite hva som er V_{IN} & V_{OUT}. Vi valgte å implementere en målesløyfe med *dupont*-konnektor til batteriet, slik at vi kunne koble direkte til *PPK2*. Ved test av strømtrekk fra systemet har vi utført testene innendørs, slik at vi ikke får noe strøm fra solcellepanelene. Dette er fordi vi ville ha disse målingene adskilt, da vi minimerer mulige feilkilder til strømtrekket.

Oppkobling til tester

- Solcellepanel - Panelene er koblet i parallell, deretter går det positiv og negativ leder til *MPPT*. Vi kobler oss på positiv leder, V_{IN} som kommer fra panel, deretter går V_{OUT} videre til *MPPT*. Vi får nå se hvor mye strøm som kommer fra solcellepanel til *MPPT*.
- MPPT - V_{IN} fra *PPK2* blir koblet til BATT terminal i *MPPT*, videre blir V_{OUT} koblet til batteriet. Vi ser nå ladestrømmen *MPPT* gir til batteriet.
- Batteri - Denne testen viser hvor mye strøm systemet trekker. Kobler positiv leder fra batteri til V_{IN}, videre går V_{OUT} til batterikontakt til *Thingy:91*.

3.3 Firmware

Denne delen av rapporten handler om hvilke programvarer vi har valgt å bruke på enheten for å oppnå de målene vi har satt.

3.3.1 LwM2M-client

Som hovedprogram valgte vi å bruke *LwM2M-client* samplet fra Nordic Semiconductor. Dette på grunnlag av at programmet allerede støttet mye av den funksjonaliteten vi var ute etter, deriblant servertilkobling med Leshan, sikker tilkobling med *DTLS* og *FOTA*, selv om den siste funksjonen ble valgt bort under oppgaven.

3.3.2 Konfigurasjon

I løpet av oppgaven så måtte hovedprogrammet konfigureres slik at den klarte å oppfylle de målene som vi satt tidligere i oppgaven. Deriblant så måtte vi legge inn *IP*-adressen til *Leshan* serveren i *prj.conf*, i tillegg til *endpoint* (navnet) til enheten, som matcher *IMEI*-adressen notert på selve kretskortet. Det måtte også legges til en sikkerhetsnøkkel i ***src\config.h***, som er med på å øke sikkerheten mellom serveren og klienten/enheten. Det ble også lagt til filer for kompileringen av programvare til *Thingy:91*, deriblant ***child_image/spm*** og ***configuration/pm_static.yml***. Disse filene er med på å beholde riktig partisjonering av minne for *Thingy:91*. Filene er hentet ut fra *Asset Tracker v2* samplet fra Nordic Semiconductor.

Det ble også oppdaget ved slutten av oppgaven hvordan vi kunne konfigurere *PSM* og *Active Mode* [2.6]. Disse endringene utføres i ***overlay-queue.conf***. Her kan man konfigurere *TAU*-verdien, som bestemmer *queue-mode* intervallet, og *RAT*, som bestemmer hvor lenge enheten er i *Active-mode*.

3.3.3 Installering og bruk av firmware

Resten av denne delen av rapporten er dedikert til hvordan leseren kan installere og benytte seg av programvaren som vi har utviklet i løpet av denne oppgaven.

Git/GitHub

For å laste ned programvaren, så må du først installere Git på maskinen din (Chacon og Straub 2021). Etter dette må du bruke kodeliste 3.1 for å laste ned programvaren til din lokale maskin.

Kodeliste 3.1: Kommando for installering av Git repo

```
git clone https://github.com/patricab/nrf9160-aqi.git
```

Dette vil installere prosjektet i mappen *nrf9160-aqi* på din lokale datamaskin.

Zephyr/Nordic Connect SDK

For å få kompilert og installert prosjektet på en *Thingy:91* enhet, så må du først installere noen verktøy. Disse kan enten lastes ned for Windows/Mac [Semiconductor 2021b], eller for Linux [Semiconductor 2021a]

I tillegg så må du installere *nRF Command Line Tools* programvaren, som er nødvendig for programmering og debugging av *Thingy:91* enheten (Semiconductor 2021c).

Etter du har installert prosjektet på din lokale datamaskin, så må du kompilere programvaren for å få en fil som du kan programmere inn på enheten. For dette må du navigere deg fram til mappen som du lastet ned: *nrf9160-aqi*, og bruke kodeliste 3.2 for kompilering til *Thingy:91*. Hvis du har Windows 10 installert på din lokale maskin, så anbefaler vi å bruke kommandolinjen som kommer med nRF Connect SDK. Denne kan du enten finne i *nRF Connect for Desktop* applikasjonen under **Toolchain Manager > *Nedlastet NCS versjon* > Open bash**, eller på ditt lokale filsystem under **ncs*Nedlastet NCS versjon*\toolchain\git-bash.exe**

Kodeliste 3.2: Kommando for kompilering av programvare for *Thingy:91*

```
west build -b thingy91_nrf9160ns -- -DOVERLAY_CONFIG=overlay-queue.conf
```

Kodeliste 3.3: Kompilering kommando av programvare for *development kit* (*DK*)

```
west build -b nrf9160dk_nrf9160ns -- -DOVERLAY_CONFIG=overlay-queue.conf
```

For å programmere enheten, så må du benytte deg av *nRF Connect Programmer* som er en del av *nRF Connect SDK*. For programmering anbefaler vi å følge guiden fra Nordic [Semiconductor 2021d].

NB!: Filene som skal programmeres på enheten ligger i:
nrf9160-aqi\build\zephyr\app_signed.hex.

3.4 Sensorer

Denne delen av rapporten handler om hvordan vi gikk fram med å utvikle driverene til de forskjellige sensorene som ble brukt i oppgaven. Her legger vi også fram de fysiske løsningene vi brukte for oppkobling, og annet arbeid vi måtte gjøre for å få sensorene til å kommunisere med *Thingy:91* enheten.

3.4.1 Sensirion SPS30

Dette kapittelet vil gjennomgå hvordan driveren til SPS30 sensoren ble skrevet og hvordan den ble satt opp. I databladet [Sensirion 2020] sto det at sensoren hadde to ulike muligheter for kommunikasjon, *UART* og I^2C. Vi valgte å bruke I^2C, siden det var begrenset med kontakter for *UART* på Thingy-en. I^2C har også muligheten til å koble til flere sensorer hvis det kreves.

Biblioteker

Siden Nordic bruker Zephyr som *RTOS*, begynte vi med å se på eksempler på drivere fra Zephyr sine egne biblioteker. Zephyr har mange ulike biblioteker med eksempler med I^2C drivere. Dette ble brukt til å forstå hvordan koden delvis skulle bygges opp. Eksemplene ble hentet fra zephyr/samples [Zephyr 2021c]. Driveren for en annen partikkelsensor PMS7003 ga oss mye info hvordan strukturen til sensorverdiene skulle være.

Kommandoer og dataformat

I databladet til SPS30 viser de hvilke ulike kommandoer man må kjøre gjennom for å kommunisere med sensoren. Sensoren har også et bestemt dataformat. Data blir sendt i 2-byte pakker, etter dette kommer det en checksum av pakkene som brukes for å godkjenne data.

Address Pointer	Command Name	Transfer Type	Parameter length including CRC [bytes]	Response length including CRC [bytes]	Command execution time	min. required Firmware
0x0010	Start Measurement	Set Pointer & Write Data	3	-	< 20 ms	V1.0
0x0104	Stop Measurement	Set Pointer	-	-	< 20 ms	V1.0
0x0202	Read Data-Ready Flag	Set Pointer & Read Data	-	3	-	V1.0
0x0300	Read Measured Values	Set Pointer & Read Data	-	float: 60 integer: 30	-	V1.0
0x1001	Sleep	Set Pointer	-	-	< 5 ms	V2.0
0x1103	Wake-up	Set Pointer	-	-	< 5 ms	V2.0
0x5607	Start Fan Cleaning	Set Pointer	-	-	< 5 ms	V1.0
0x8004	Read/Write Auto Cleaning Interval	Set Pointer & Read/Write Data	read: - write: 6	read: 6 write: -	read: < 5 ms write: < 20 ms	V1.0
					read: - write: < 20ms	V2.2
0xD002	Read Product Type	Set Pointer & Read Data	-	12		
0xD033	Read Serial Number	Set Pointer & Read Data	-	max. 48	-	V1.0
0xD100	Read Version	Set Pointer & Read Data	-	3	-	V1.0
0xD206	Read Device Status Register	Set Pointer & Read Data	-	6	-	V2.2
0xD210	Clear Device Status Register	Set Pointer	-	-	< 5 ms	V2.0
0xD304	Reset	Set Pointer	-	-	< 100 ms	V1.0

Figur 3.2: SPS30 Kommandoliste [Sensirion 2020]

Interne funksjoner

Vi satt opp de tre ulike kommandoene: *Set Pointer*, *Set Pointer & Write Data* og *Set Pointer & Read Data*, som interne funksjoner. Dette gjorde at koden ble mer oversiktlig, og siden vi bruker flere ulike kommandoer så slipper vi å skrive lang kode for hver kommando. Ved å bruke Zephyr sitt biblioteket zephyr/include/i2c.h [Zephyr 2021b] kunne vi lett sende I^2C meldinger uten å prøve å emulere I^2C funksjonalitet i koden.

set_pointer består av tre bytes, og sender først hvilken I^2C-adresse den skal bli sendt til. Deretter kommer det to bytes som beskriver hvilken kommando-addresse som skal kjøres gjennom. De to andre fungerer som den første, men enten tar å leser en byte etterpå, eller skriver til slaven med et argument.

set_pointer_read kjører først gjennom *set_pointer* funksjonen. Deretter skrives det til I^2C adressen der slaven sender data tilbake, som blir lest inn i funksjonen.

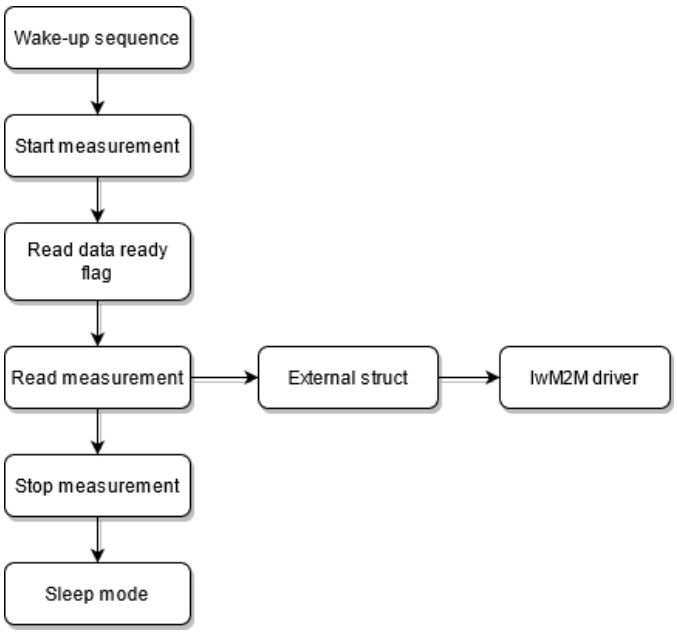
check er en intern funksjon som regner ut en checksum av argumentene som den tar inn. Den tar inn 2 bytes, og godkjenner dataen man får inn eller sender ut.

Tilgjengelige funksjoner

I driveren er det to funksjoner som kan brukes globalt:

sps30_init brukes for å initialisere I^2C enheten ved å bruke en ekstern funksjon fra "I2C.h". Den eksterne funksjonen konfigurer enheten og gjør den klar for kommunisere ved I^2C.

sps30_particle_read er hovedfunksjonen i hele driveren. Det er denne som kjører i gang partikkelsensoren og lagrer sensorverdier. Først starter funksjonene med å kjøre gjennom en "*våkne*" sekvens som starter sensoren. Deretter kommer *start measurement* kommandoen. Denne starter vifta og sensorene begynner å måle partikler. Funksjonen bruker den interne funksjonen *set_pointer_write*, der vi sender et argument for valg av dataform. Vi valgte å bruke *Big Endian IEEE754 float* form for data, siden det var enklere å sende opp en *float* verdi i *LwM2M* driveren enn en *integer*.



Figur 3.3: Flytdiagram av funksjon sps30_particle_read

Testbenk

Vi brukte først en *Arduino* testbenk som ble hentet fra Sensirion sitt *Git repository* [Sensirion 2021]. Ved å teste med Arduino, kunne vi kjøre gjennom enkel kode med et veldig modulært oppsett. Dette gjorde det mulig å justere og lett teste om begge sensorene funket som de skal. Før dette måtte vi koble til kontakter til sensorene. Ledningene ble krympet og gjort klare for lodding.

Etter begge sensorene ble testet og godkjent med *Arduino*, gikk vi over til å skrive en testbenk i Zephyr for nRF9160DK. Vi opprettet en ny branch i Git, og kalte den for *pms_test*. Deretter satt vi opp en enkel filstruktur for Zephyr med *CMakeLists.txt* og *Kconfig* filer. Dette gjorde at vi kunne tilpasse filstrukturen spesifikt for testbenk rundt nRF9160DK, uten å få andre problemer.

Oppsett til *Thingy:91*

Vi brukte to testpadder, *tp10* og *tp9*, for å kommunisere med *Thingy:91*, siden de var eneste kontaktene vi kunne lodde på som hadde I^2C tilkobling. Grunnet at sensoren krevde 3,3V (LVTTL) eller 5V (TTL), så måtte vi bruke en *levelshifter* siden *GPIO* kontaktene til *Thingy:91* bare ga ut 1,8V.

LwM2M

For å sende sensordata til skyen måtte vi lage en *LwM2M* driver (*lwm2m_pms*). Denne ble brukt for å kjøre gjennom de riktige funksjonene i sensordriveren, og for å sette riktige objekt typer. For sensorverdiene brukte vi ressursverdi *5700*, som er en *LwM2M* ressurs for sensorverdier. Sensoren ble også satt opp som en generell sensor, med objekttype *3303*, for at det skulle funke med gass- og VOC-sensoren.

3.4.2 Spec Sensor DGSO3

Hvis man ser på databladet til gassensoren som vi valgte å bruke [Spec 2017], så ser man at sensoren kun kan bruke *UART* til kommunikasjon. Basert på dette så måtte vi designe vår egen driver for å klare å kommunisere og hente ut data fra sensoren.

Biblioteker

For å gjøre prosessen så enkelt som mulig valgte vi å bruke Zephyr sine egne drivere for *UART* kommunikasjon. Disse er tilgjengelige ved hjelp av Zephyr sitt *API* [Zephyr 2021a]. I løpet av utviklingen så oppsto det tre metoder/funksjoner som vi kunne benytte oss av for å kommunisere med sensoren:

1. uart_poll: Henter ut karakterer enkeltvis ved *interrupt callback*
2. uart_fifo: Henter ut data fra *FIFO* køen. Kalles også ved *interrupt callback*
3. Asynkron : Bruker asynkrone funksjoner (*uart_tx*, *uart_rx*). Må helst legges i egen *thread*.

For å gjøre utviklingen enklere så utelukket vi metode 3, siden dette ville øke kompleksiteten til driveren ytterligere. Vi gikk for en blanding av metode 1 og 2, der vi sender ut kommandoer til sensoren ved hjelp av *uart_poll* og tar inn data ved *uart_fifo*.

Kommandoer og dataformat

På side 6 i databladet til gassensoren, så er det listet opp de forskjellige kommandoene som vi kan sende til sensoren. Vi bestemte oss for å velge fire kommandoer: *TRANSMIT* som sender tilbake en målestreng, *STANDBY* som setter sensoren i *low-power standby* modus, *ZERO* som nullstiller kalibreringen til 0 *ppb*, og *SPAN* som gir brukeren mulighet til å sette eget nullpunkt til kalibrering [Spec 2017].

Målestrengen som blir sendt tilbake av sensoren ved *TRANSMIT* kommandoen er spesielt formatert. En målestreng blir sendt tilbake til brukeren slik:
SN, PPB, T (°C), RH (%), ADC Raw, T Raw, RH Raw, Day, Hour, Minute, Second.

Her sendes først serienummeret til sensoren, etterfølgt av måleverdien i *ppb*, temperatur i Celsius, prosentverdi av fuktighet, råverdier for temperatur og fuktighet samt måletid i dag, timer, minutt, og sekund. Det vi fokuserte på var måleverdien i *ppb*, siden vi har andre sensorer som måler temperatur og fuktighet på *Thingy:91*, samt internt tidsstempel på når måleverdien ble tatt. Derimot så kan serienummeret brukes for å verifisere om sensoren fortsatt fungerer. Dette ble gjort ved en anledning, der vi fant ut at en av sensorene ikke funket, siden det manglet store deler av serienummeret, samt rar formatering på måleverdiene som vi fikk tilbake.

Interne funksjoner

I driveren er det et par funksjoner som ikke er tilgjengelig utenfor selve driveren, men som brukes internt til ulike formål:

uart_cb() er en *interrupt callback*. Dette blir konfigurert i *init_uart()*. I funksjonen så begynner driveren først å prosessere interrupts ved *uart_irq_update*. Deretter sjekker den om *UART* rx-bufferen har mottatt karakterer. Hvis det er tilfelle leser den inn karakterer ved *uart_fifo_read* inn i variabelen *rx_val*. Denne variabelen blir tømt hver gang funksjonen kjøres. Funksjonen sjekker også om variabelen som tas inn er en *newline* karakter (*\n*). Hvis det er tilfelle setter den flagget (boolsk) *data_received* som blir behandlet i funksjonen *uart_rx*. Hvis variabelen ikke er en *newline* karakter så legger den til variabelen i bufferen *rx_buff* og øker indeks *n* med 1.

uart_rx blir kalt av funksjonen *read_gas*. Denne funksjonen forbereder noen variabler, som for eksempel å nullstille indeks *n*, og sette flagget *data_received* til *false*. Funksjonen bruker også *uart_irq_rx_enable* og *uart_irq_rx_disable* som slår henholdsvis på og av interrupts for *UART* mottakeren. I mellomtiden så venter funksjonen på at *data_received* blir satt til *true*, som betyr at *interrupt callback* *uart_cb* har mottatt en målestreng.

Tilgjengelige funksjoner

Etterfølgt av disse har vi de funksjonene som er tilgjengelig utenfor driveren. Det er disse funksjonene som kalles eksternt av *LwM2M* driveren. De bruker ofte de funksjonene som er beskrevet over, og er der for å klargjøre og utføre kommandoene som skal sendes til sensoren, samt forbereder hvordan dataen skal mottas.

read_gas er en av funksjonene som blir brukt mest av eksterne programvarer. Funksjonen setter opp nødvendige buffere, sender ut kommandoer til sensoren, kaller *uart_rx*, filtrerer ut *ppb* basert på dataformatet og sender ut bufferen fra driveren.

Funksjonene *standby_gas*, *zero_gas*, og *set_gas* sender de kommandoene til sensoren som er nevnt i 3.4.2. I tillegg så legger funksjonen *set_gas* opp til å sende over både kommando og data til sensoren, som er verdien brukeren har lyst til å sette som nullpunkt.

Til slutt så har vi funksjonen *init_uart*, som gjør alt klart for kommunikasjon via *UART* til gassensoren. Her settes de nødvendige strukturene som både konfigurerer hvordan programvaren kommuniserer med enheten, og som konfigurer enhetens *UART* innstillinger. Disse blir da videre behandlet ved funksjoner som *uart_configure*, og det blir satt en *interrupt callback* ved funksjonen *uart_irq_callback_set*.

LwM2M

I tillegg til at vi måtte utvikle driverprogramvare for gassensoren, måtte vi også utvikle drivere for å kommunisere med serveren, ved hjelp av *LwM2M*. Derimot så var denne jobben betydeligere lettere, siden vi allerede hadde tilgang til eksempler av drivere i *lwm2m_client* programvaren fra Nordic.

Selve driveren består av 3 funksjoner: *lwm2m_init_gas*, *gas_read_cb*, og *read_val*. Disse funksjonene har forskjellige formål i forhold til oppkobling og sending av sensorverdier til serveren.

read_val er en funksjon som blir kalt av *gas_read_cb*. Denne funksjonen har som formål å bruke sensordriveren som vi har utviklet for å kommunisere med gassensoren. Funksjonen benytter seg av *read_gas* for å få verdier inn i variabelen *val*. Denne verdien blir satt inn i *float_val* strukturen som blir sendt ut av funksjonen til *gas_read_cb*. Denne strukturen består av to *int32_t* variabler: *val1*, og *val2*. Disse variablene representerer et desimaltall, der *val1* er heltallsdelen, og *val2* er fraksjonsdelen. For å gjøre det enklere å feilsøke koden så bestemte vi oss for å ignorere å dele opp verdien vi fikk fra sensoren i en heltallsdel og fraksjonsdel, og heller sette hele verdien inn i *val1*.

gas_read_cb er en *callback* funksjon. Den blir kalt av serveren når den vil hente inn verdier, representert med *LwM2M* ressursverdi *5700*. Funksjonen blir brukt til blant annet å sette timestamp for sensorverdien, lese sensorverdier med funksjonen *read_val*, og sende verdien videre til serveren som en *float32* verdi.

lwm2m_init_gas funksjonen gjør *UART*-enheten tilgjengelig, og konfigurerer driveren som vi har utviklet ved hjelp av henholdsvis: *device_get_binding*, og *init_uart*. Funksjonen gjør også alt klart i koden til oppkobling mot serveren. Den bestemmer for eksempel:

- Type objekt sensoren skal ha
- Forekomst sensoren skal ha
- Hvilken funksjon som skal kalles når serveren forespør ressurs *5700*
- Gjøre klart tidspunkt for sensorverdiene

Hardware/Thingy

For å få etablert fysiske koblinger mot gassensoren, så må vi referere til databladet til gassensoren [Spec 2017], og *Thingy:91* [Nordic 2019]. Vi kan se hvilke kontakter vi må koble oss til på gassensoren, hvis vi ser på tabellen på side 3 av databladet. Vi har komprimert denne tabellen til tabell 3.1. Her er $V+ = 3{,}3V$, som er driftsspenningen til gassensoren.

Kontakter	Funksjoner
2	RXD
3	TXD
6	GND
8	V+

Tabell 3.1: Pinout - DGSO3

For å få en kobling opp mot *Thingy:91*, så måtte vi derimot gjøre noen endringer. Siden vi hadde et begrenset antall kontakter tilgjengelig, prøvde vi å bruke kontakter som allerede er tatt i bruk av systemet. Først testet vi med TP34, og TP35, som blir brukt som SPI kobling av *&spi3* linjen på enheten, spesifikt *SCK*, og *MOSI*. Vi fikk ikke kontakt og måtte derfor bytte over til *DK*-et, der vi brukte kontaktene P0 og P1.

Et problem som oppstod i løpet av utviklingen av prototypen var at sensoren og *Thingy:91* opererte på to forskjellige spenningsnivå: *3,3V*, og *1,8V*. For å løse dette problemet så benyttet vi oss av en *levelshifter*, som hadde mulighet å øke og minke spenningen fra 1,8 til 3,3V, og fra 3,3 til 1,8V. Her deler da partikkelsensoren SPS30 og gassensoren samme *levelshifter*.

Her ble da kontakt 2, og 3 fra tabell 3.1, som er *RX*, og *TX* linjene fra sensoren, koblet opp til *levelshifter*en og videre til *Thingy:91*. Det ble også tilført en referansespenning på 3,3V og 1,8V til *levelshifter*en for å kalibrere spenningsreguleringen.

3.5 Skytjeneste

Denne delen av rapporten handler om serveren, databasen og dashbordet.

3.5.1 Server

For oppgaven valgte vi å bruke en egen server. Dette ga oss mere frihet rundt hvordan vi kunne sette opp programvaren for *Leshan* og *LwM2M*, og ga oss mere innsikt rundt hvordan å sette opp en egen server for et slikt formål.

Til maskinvare brukte vi en Intel *Next Unit of Computing (NUC)*, som er en liten formfaktor *barebones* PC (uten inkludert minne, harddrive, og operativsystem). Her installerte vi *Debian Linux 10.8*. Valget av *Debian Linux* kom på grunnlag av at operativsystemet er ofte brukt som en standard i servere, og er lagd for å være stabilt. Programmene ble kjørt som *Daemon*er.

Leshan

Vi valgte *LwM2M* med tanke på sikkerhet (*DTLS*), *FOTA*, og interoperatibelt. For server versjonen av *LwM2M* gikk vi for *Leshan*, siden det så ut til å være en god implementasjon av *LwM2M* basert på Java, som er et programmeringsspråk vi har vært borti før.

Vi startet med å bruke en demo for klient og server for testing. Så lagde vi en egen server for bedre forståelse for koden, og la til funksjonaliteten vi trengte. Her er *DTLS* et godt eksempel, siden vi måtte endre hvor lenge den prøvde å være i *kø-modus*, for at den ikke skulle avregistreres fra serveren. Vi byttet til å sette status til *offline* når den går til søvnmodus, og til *online* når den våkner [A.3]. Når det virket, la vi til funksjonene fra egen *server-til-server* demo. Dette var for å bruke den allerede fungerende webserveren og sikkerhetskonfigurasjonen.

Installasjon

- *Java Development Kit* (*JDK*)
- Maven
- ZeroTier One

Alle disse installeres som vanlige program. *JDK* og Maven trengte vi for *Leshan*, ZeroTier brukes for sikker tilkobling til serveren.

Kodeliste 3.4: Miljøvariabler (endre til dine versjoner)

```
pathman /au C:\Program Files\apache-maven-3.6.3\bin
pathman /au C:\Program Files\Java\jdk-15.0.2\bin
setx JAVA_HOME -m "C:/Program Files/Java/jdk-15.0.2"
```

Kodeliste 3.5: Start Leshan Server

```
cd nrf9160-aqi/leshan/aqi-server
mvn install
java -jar target/aqi-server-1.0-SNAPSHOT-jar-with-dependencies.jar
```

Test klient

For å teste om serveren funket, brukte vi en test klient (thingy-sim), basert på *Leshan* demo klient. Den genererte tilfeldige tall til alle sensorverdiene, for de riktige *LwM2M* ressursene.

Kodeliste 3.6: Start Leshan Klient

```
cd nrf9160-aqi/leshan/thingy-sim
mvn install
java -jar target/thingy-sim-1.0-SNAPSHOT-jar-with-dependencies.jar
```

På port :8080 er det ett enkelt web grensesnitt, basert på RESTful *API*. Dette er bare en kopi fra demo-serveren, med små endringer.

ZeroTier One

Zerotier One er et *Virtual Local Area Network* (*VLAN*) for sikker tilkobling til serveren. Alle som skal kunne koble til serveren må ha installert *Zerotier*. For å koble til nettverket må en starte *Zerotier* på PC-en, så klikke *join network* i *system tray* og skrive inn nettverks-ID-en. Du trenger ikke endre noen innstillinger. Så må den autentiseres av nettverkseieren. Bare nettverkseieren trenger å ha en *Zerotier* bruker. Deretter godkjennes *IP*-en, og du kan nå se *Grafana* dashbord og *Leshan* serveren. For å styre serveren brukte vi *Secure Shell* (*SSH*)

Kodeliste 3.7: *SSH* kobling til serveren

```
ssh -C bruker@zerotierserverip
passord
```

Thingy:91

For å koble *Thingy:91* til *Leshan* med *DTLS* så må man først legge til *IMEI* og sikkerhetsnøkellen (PSK) fra *Thingy:91* i sikkerhetsdelen av *Leshan*. Da kan *Thingy:91* koble seg til *Leshan LwM2M* serveren og bli registrert. Da sender den data som dato, *IP*, navn og en liste med tilgjengelige *LwM2M* objekter tilbake til serveren. Det blir også satt tidspunkt og det blir loggført i databasen at den er tilkoblet. Deretter går den i søvnmodus og sier ifra til serveren at den blir borte i 5 min. Dette blir også logget.

Når den skrur seg på igjen så kjører serveren en kommando som spør etter alle sensorverdier, som temperatur, fuktighet, gass og partikler. Disse blir så lagret i en *postgreSQL* database. Alt fra loggen blir vist i *Grafana* dashbordet. For å få sensorene til å virke med *LwM2M*, måtte vi skrive egne drivere (se *LwM2M* paragrafer i seksjon [3.4.1, 3.4.2]).

Nettverk

Vi brukte *LTE-M* siden iBASIS, selskapet som har *SIM*-kortene med Nordic, ikke støtter *Narrow-Band Internet of Things* (*NB-IoT*), selv om iBASIS bruker mobilnettverket til Telenor som støtter det [iBASIS 2021]. Telenor har også støtte for *Power Saving Mode* (*PSM*) som gjør at en ikke trenger å registrere seg på nettverket hver gang en går ut av *søvnmodus*. For at Thingy skal kunne koble seg på serveren må *CoAP* portene åpnes med *port forward*ing. *CoAP* porten :5683 og *CoAPS* porten :5684.

DNS

Vi koblet oss opp til serveren med en *Domain Name System* (*DNS*). Adressen *freedombox.rocks* ble satt til riktig *IP*. Dette var nyttig når vi kom i en situasjon der *IP*-en ble endret uventet. *DNS*-en vi brukte heter *GnuDIP*, som er en dynamisk *IP DNS* service.

3.5.2 Database

PostgreSQL

For lagring bruker vi en *PostgreSQL* database med *Timescale* utvidelse. *SQL* er ett programmeringsspråk for databaser, og *Timescale* gjør at dataen bli lagret i timeseries, altså lagrer data etter tid og ikke ID. Vi valgte å bruke *PostgreSQL* siden den har timeseries utvidelse, som er nyttig når dataen skal vises i *Grafana* [Linuxize 2019, Timescale 2020].

Kodeliste 3.8: Lage PostgreSQL Database

```
echo "deb http://apt.postgresql.org/pub/repos/apt/ $(lsb_release -c -s)-pgdg main" |
sudo tee /etc/apt/sources.list.d/pgdg.list
wget --quiet -O - https://www.postgresql.org/media/keys/ACCC4CF8.asc |
sudo apt-key add -
sudo apt update
sudo apt install postgresql postgresql-contrib   # install postgresql and extensions
sudo -u postgres psql -c "SELECT version();"
# ‘lsb_release -c -s‘ should return the correct codename of your OS
sudo sh -c "echo ’deb https://packagecloud.io/timescale/timescaledb/debian/
‘lsb_release -c -s‘ main’ > /etc/apt/sources.list.d/timescaledb.list"
wget --quiet -O - https://packagecloud.io/timescale/timescaledb/gpgkey |
sudo apt-key add -                               # Add repository
sudo apt update
sudo apt install timescaledb-2-postgresql-13     # install appropriate package
sudo su - postgres                               # login as postgres
psql                                             # start database
sudo su - postgres -c "createuser airq"          # create user
sudo su - postgres -c "crearedb airqdb"          # create database
sudo -u postgres psql                            # enter database
GRANT ALL PRIVILEGES ON DATABASE airqdb TO airq; # grant privileges
```

JDBC

Java Database Comunication (*JDBC*) blir brukt til å snakke med databasen fra *Leshan*. For at det skulle funke måtte vi legge til en CLASSPATH til *PostgreSQL*. JDBC driver lokasjonen. Dette kan gjøres i *.profile* filen i hjemme-mappen med et tekstredigeringsprogram, for eksempel brukte vi *Vim*:

Kodeliste 3.9: .profile endringer

```
cd ~            # HOME directory
vim .profile    # edit .profile file

# legg til disse linjene i filen trykk ins(ert)
export CLASSPATH=/usr/share/java/*
export PATH=$PATH:$HOME/bin
alias adb=’psql -U airq -h localhost airqdb’ #lagre med Esc shift+z shift+z
```

Tabeller

For å sette inn i tabeller bruker du denne *SQL* koden:

INSERT INTO tabell(*kolonner å sette inn*) VALUES(*verdier å sette inn*) [kodeliste 3.10].

Vi brukte objekt *3300* for resten av sensorene siden den var allerede definert på *Zephyr* klienten.

Kodeliste 3.10: Observasjon database kode

```
DROP TABLE IF EXISTS observation CASCADE; -- Fjern tabell først for å oppdatere
CREATE TABLE observation (
    "time" TIMESTAMP WITH TIME ZONE DEFAULT CURRENT_TIMESTAMP NOT NULL,
    "timestamp" BIGINT,
    "endpoint" VARCHAR(64) NOT NULL,
    "object" INT NOT NULL,
    "instance" INT,
    "resource" INT DEFAULT 0,
    "value" TEXT
);

INSERT INTO observation(timestamp, endpoint, object, instance, resource, value)
VALUES(0, Thingy:91-Test, 3300, 0, 5700, 53.50);
```

time	age	endpoint	description	status
2021-04-23 10:57:07	01:02:09	gas-pab	Ozon sensor	2 (online)
2021-04-23 09:10:09	02:49:07	gas-test	Ozon sensor	1 (sleeping)
2021-04-22 12:28:24	23:30:52	Thingy-Sim	Leshan test	0 (offline)

Tabell 3.2: Enhet database tabell

time	ts	endpoint	object	inst	resource	value
2021-04-14 12:46:36+01	0	Thingy:91	3300	0	5700	53.50
2021-04-14 12:46:37+01	0	Thingy:91	3300	1	5700	0.96
2021-04-14 12:46:37+01	0	Thingy:91	3300	2	5700	114.00
2021-04-14 12:46:37+01	0	Thingy:91	3300	3	5700	281.00
2021-04-14 12:46:37+01	0	Thingy:91	3300	4	5700	129.00
2021-04-14 12:46:37+01	0	Thingy:91	3303	0	5700	13.60
2021-04-14 12:46:37+01	0	Thingy:91	3304	0	5700	39.70

Tabell 3.3: Observasjon database tabell

time	endpoint	ip	regid	message	error
2021-04-19 11:32:46	gas test	/62.93.143.64	v75FS1zeB1	Registered	
2021-04-19 11:34:53	gas test	/62.93.143.64	v75FS1zeB1	Deregistered	
2021-04-19 11:36:44	gas test	/62.93.143.72	q3iLU1u6sk	Registered	
2021-04-19 11:46:27	gas test	/62.93.143.72	q3iLU1u6sk	Deregistered	

Tabell 3.4: Log database tabell

For å gjøre tabellene mer leselige kan man lage "*Views*". Dette er tabeller som tar data fra hovedtabellene og setter de opp slik du vil ha de. Dette gjør det mulig å sortere etter sensortype, tidspunkt, eller enhet.

time	age	endpoint-size	value	unit
2021-04-27 13:44:31+01	21:04:26	nrf9160dk PM10	10220865	ug/m3
2021-04-27 13:44:21+01	21:04:36	nrf9160dk PM2,5	10944080	ug/m3
2021-04-27 13:44:08+01	21:04:49	nrf9160dk Typical PM	10706381	ug/m3

Tabell 3.5: Partikkel sensor view

3.5.3 Dashbord

Som dashbord valgte vi *Grafana* siden det er en av de mest populære programmene for dette, og den har åpen-kildekode. Den kan også ekspanderes med *plugins* for utvidet funksjonalitet.

Installasjon

Bash koden under viser hvordan man installerer og setter opp en *Grafana Daemon* service, på *Debian* linux [Grafana 2021].

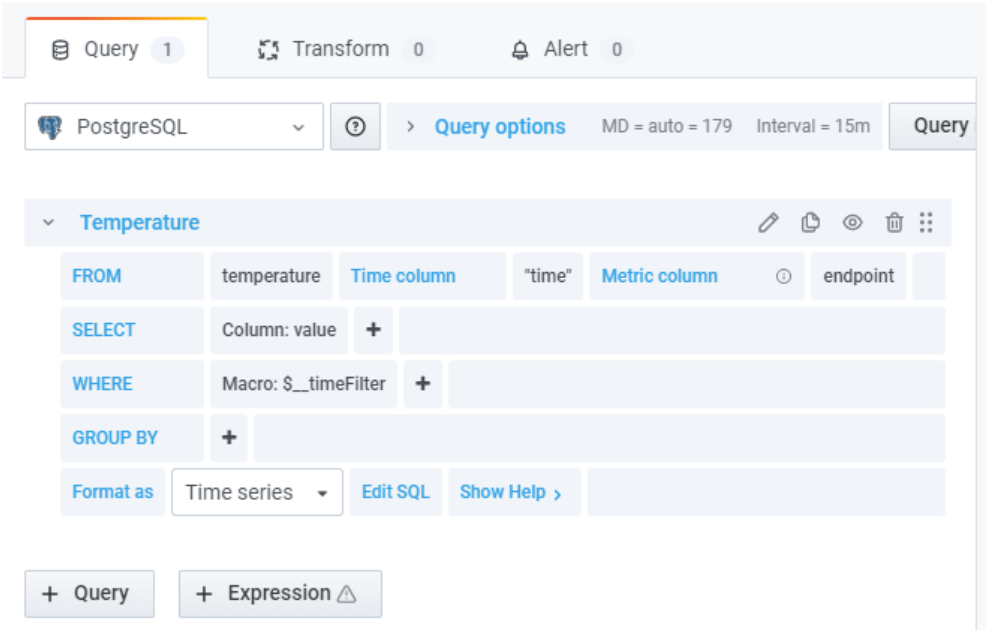
Kodeliste 3.11: Sette opp Grafana Server

```
sudo apt install -y apt-transport-https
sudo apt install -y software-properties-common wget
wget -q -O - https://packages.grafana.com/gpg.key |
sudo apt-key add -           # add grafana repository Open-Source Software version
echo "deb https://packages.grafana.com/oss/deb stable main" |
sudo tee -a /etc/apt/sources.list.d/grafana.list
sudo apt update                                          # update package list
sudo apt install grafana                                 # install grafana
sudo service grafana-server start                        # Start grafana server
sudo service grafana-server status                       # Check if it works
sudo update-rc.d grafana-server defaults                 # Set to start on boot
```

Nå kan man logge seg inn på localhost:3000 i en nettleser.
Der vil man få opp ett *Grafana login* vindu. Logg inn med disse verdiene.
username: admin
password: admin

Lage dashbord

Deretter kan man lage et dashbord, og sette opp paneler med grafer, metere, og tabeller. For dette kan man trykke *Create Dashboard* i venstre kolonne. *add panel* knappen i toppen og *add empty panel*. En kan velge mellom *Graph*, *Gauge*, eller *Table* i *Vizualization*. Her velger du *PostgreSQL* som datakilde. Vi brukte *query builder*, men skrev litt SQL for å endre verdiene til passende enheter (se Fig 3.4).



Figur 3.4: Grafan Query Builder

Oppsett

Etter vi hadde samlet inn data, så kunne vi fikse på noen småting for å få ett finere oppsett. Som blant annet å sette navn på panel til passende sensorverdi, legge til riktige enheter for Y-min/max i *Axes* panel og farge etter verdi med *thresholds* i *Field*-panel. Grenseverdiene er fra *EAQI*, men vi brukte verdiene for $PM_{2,5}$ for alle partikkelverdier, selv om PM_{10} verdiene ikke er like strenge [EEA 2021]. For VOC verdiene fulgte vi databladet til BME680 sensoren [Bosch 2017].

3.6 Verifisering av data

En del av problemstillingen til denne oppgaven [1.1] er å verifisere måledataen som vi får ut av systemet. Dette gjør vi for å få en sterkere antydning på om den dataen vi henter inn stemmer med virkeligheten, og gir et grunnlag på hvorvidt systemet faktisk fungerer optimalt.

For å komme til en slik konklusjon så må vi prøve å finne en statistisk korrelasjon mellom to datasett: Det vi samler inn ved egne sensorer, og et datasett fra en ekstern aktør. For dette benyttet vi oss av *Norsk institutt for luftforskning* (*NILU*) sine måledata for luftkvalitet [NILU 2021a]. Her legges det ut historiske data fra målestasjoner plassert rundt omkring i Norge. Vi benyttet oss av to målestasjoner lokalt i Trondheim: En ved Elgeseter, og en på Omkjøringsveien (plassert nært Moholt). Begge disse målestasjonene samler inn data rundt partikkelmengde ($PM_{2,5}$, PM10) og gasskonsentrasjon, spesifikt nitrogendioksid (NO_2).

Her har vi delt verifiseringen i to steg:

- Få en generell indikasjon på korrelasjonen mellom datasett ved hjelp av *Pearsons korrelasjonskoeffisient* og en tommelfingerregel for koeffisienten
- Gjør en T-test ved hjelp av *students T-distribusjon*, med en $\alpha = 0{,}05$

Her fungerer *Pearsons korrelasjonskoeffisient* som en generell indikator på om de datasettene vi sammenligner stemmer med hverandre. Etter dette gjør vi en T-test for å komme fram til signifikansen til korrelasjonen. Det vil si: Hvor sannsynlig er det at korrelasjonen, gitt av korrelasjonskoeffisienten, kommer av tilfeldighet. Her har vi en nullhypotese H_0: *Datasettene er ikke korrelert*, og en alternativ hypotese H_1: *Datasettene er korrelert*.

For dette ble det produsert et Python program som automatiserer denne prosessen [A.4].

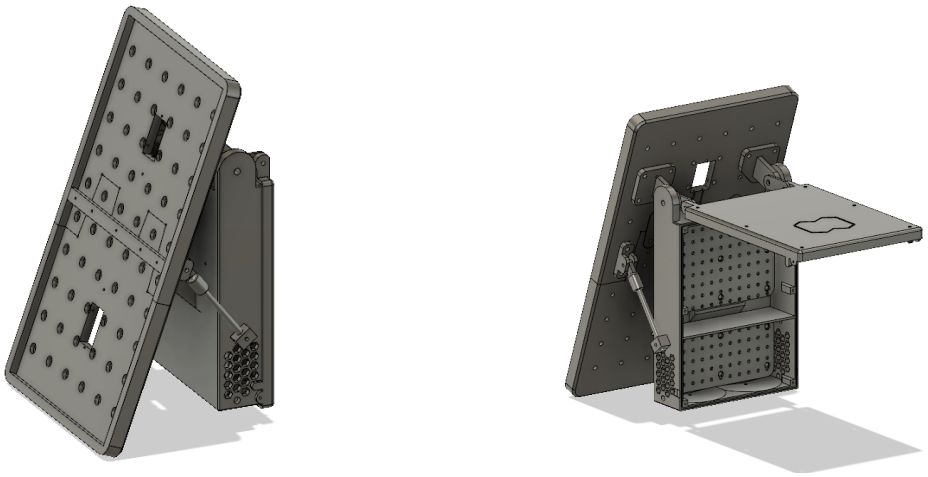
Her blir datasettene lest inn fra *CSV* format, og de kolonnene med relevant data ($PM_{2,5}$, PM_{10} og NO_2) blir hentet ut. Deretter sjekker programmet om kolonnene fra datasettene er av samme lengde, som er med å sikre at datasettene blir sammenlignet optimalt. Deretter benytter vi oss av funksjoner fra *SciPy*-biblioteket i Python for å beregne de variablene som trengs for å gjennomføre verifiseringen. Dette inkluderer *scipy.stats.pearsonr* som gir ut *Pearsons korrelasjonskoeffisient*, og en to-sidet p-verdi i et array. Vi velger å ignorere det siste elementet, siden vi selv beregner en p-verdi basert på T-testen. Det beregnes også en boolsk variabel, på grunnlag av tommelfingerregelen for korrelasjonskoeffisienten, gitt av formel 2.1

Deretter beregnes det en t-verdi basert på formel 2.2. Denne verdien blir satt inn i funksjonen *scipy.stat.t.sf*, som finner riktig p-verdi gitt absoluttverdien av den beregnede t-verdien, og antall grader av frihet. Det settes da en boolsk verdi på grunnlag av denne p-verdien. Hvis p-verdien er under 0,05 så forkaster vi nullhypotesen og setter variabelen til *True*. Hvis ikke så forkastes ikke nullhypotesen og variabelen settes til *False*. Alle disse verdiene skrives ut til terminalen.

Før disse verdiene blir behandlet av algoritmen som er beskrevet over så må de formateres riktig. I forhold til vårt eget datasett, så må vi skille ut sensorverdier basert på sensortype. Dette er fordi datasettet som ble generert kommer direkte fra databasen nevnt i 3.5.2. Her oppstår det en blanding av flere ulike sensorverdier og sensortyper fra flere ulike enheter. I programmet [A.4] henter vi ut tidsverdiene og bytter orientering på tabellen, siden tidsverdiene fra databasen er sortert fra nyeste verdi først. Deretter henter vi ut verdiene fra kolonne 4 basert på *instance*-nummeret, siden de verdiene vi er mest interessert i, i forhold til verifisering, benytter seg eksklusivt av objekt *3300*. Dette gjør at den eneste måten å skille ut

riktig type sensorverdi på, er å se på hvilket tilfelle ("*instance*") av et sensorobjekt som oppstår. Deretter snur vi tabellene, på samme grunnlag som for tidsverdiene, og fjerner alle nullverdier. I tillegg velger vi bare et utsnitt av elementer som har en tidsverdi som sammenfaller med tidsverdiene til datasettet fra *NILU*. Her er det et tidsintervall fra 14:00 til 21:00 (*UTC+02*), med 1 time intervall mellom hver verdi. For vårt eget datasett så valgte vi et utsnitt der tidsverdiene er innen 5 minutter av datasettet fra *NILU*.

3.7 CAD



(a) CAD modell framside

(b) CAD modell bakside med lokk hevet

I løpet av oppgaven ble det satt kriterier til prototypens design og funksjonalitet. Kriteriene er valgt på bakgrunn fra erfaring med montering av koblingsbokser og elektrisk utstyr utendørs som følger norske tetningskrav. Disse kriteriene skal dekke behovet for et produkt som skal stå ute i det nordiske klimaet, uten at det finnes noe formelt sertifikat for at det faktisk skal tåle det.

- Tett kammer for elektronikk
- Luftig kammer for sensorer
- Juster- og låsbart hengsel for PV-panel
- Enkel lukke/åpne lokk, samtidig tett
- Kabelinnføring
- Kabelhåndtering
- Festemuligheter for hele system til tre/stang/bygg og lignende
- Sluk/avløp/drenering for at oppsamlet kondens/vann skal renne ut av luftig og tett kammer.
- Modulært, utskifting av deler, parametriske hull til skruer og festeanordninger

Tett kammer for elektronikk blir løst ved å legge til en kanal mellom kontaktflatene til kapsling og lokk. I denne kanalen kan vi plassere en pakning/O-ring for å holde alt tett. Samt designe slik at vann ikke blir ledet inn til kammeret. Det gjelder spesielt ved kabelinnføring som går fra utsiden til innsiden. Derfor har vi et innhuk på baksiden av kapslingen, slik at kabler får en dryppkant, eller en u-sving som gjør at vannet ikke blir ledet inn til kammeret. Kabelinnføringer skal også tettes med en fugemasse lik *Tec7*.

For at sensorene skal måle realistiske verdier må de plasseres i et luftig kammer. Her har vi en del av kapslingen som har en rekke med heksagonale hull på begge sider som skal la de måle luftkvaliteten. De heksagonale hullene er nokså store. For å unngå at insekter kan komme inn i kammeret så kan man plassere en fin netting i disse hullene. Begge kammerene har en opphøyd, avtagbar monteringsplate for elektronikken. Plassen under er for å plassere kablene, og slik at eventuell fuktighet ikke renner gjennom elektronikken.

Når det kommer til en enkel åpne/lukke mekanisme av lokk, som også er tett, har vi gått for fire skruer som klemmer kapsling og lokk sammen. Skruene blir skrudd inn i kapsling som har gjengede skruestykker, eller messingstykker med gjenger. Under produksjon blir disse messingstykkene varmet opp og presset inn i tilhørende plastikkhull. Ved å bruke disse får vi en mer robust sammenkobling, og de blir ikke like fort slitt som plastikken som vi printer med. Lokket er også hengslet til kapslingen, der hengslet er avtagbart.

Vi hadde et ønske om å låse vinklingen til solcellepanelene. For dette designet vi et ledd fra kapsling til panelet. For å låse posisjon kan disse leddene strammes til, eller man bruke en stang fra panel til kapsling og låse posisjonen slik.
For å redusere eventuelt oppsamlet vann/fuktighet i begge kammerene, så designet vi med et fall i gulvet som leder til et hull ut av kapslingen. Med tanke på modularitet, så har vi prøvd å designe alle iterasjonene slik at deler kan byttes mellom hver modell. Med mer tanke til hver del, så unngår vi å printe flere deler som har samme egenskaper.

Vi har lagt med et overflødig antall av skruehull ved disse modellene, slik at vi fikk mulighet til å teste forskjellige oppsett og plassering av hengsler og kapsling. For skruehull så har vi fulgt størrelser som passer boltene M2, M3, M4, M5 og M6.

Egenskaper som ikke er implementert i designet er festemuligheter for prototype til stang/tre/bygning, samt solcellepanelene er utsatt for vær og vind. Her tenker vi mest på kantene og baksiden. Modellen kan stå oppreist for seg selv på et bord og lignende, noe som har vært tilstrekkelig for vår tid med testing.

3.7.1 Filament

Vi har sett på de forskjellige filamentene sine styrker og svakheter. Det har blitt sett på flere typer filament som vi kunne bruke. Vi hadde et ønske om et robust og værbestandig prototype, samtidig som vi er begrenset av å 3D-printe med en hobbyprinter. Det beste for oss ville vært å bruke *PETG* filament, som ville tålt tidens tann bedre og plastikken degraderes ikke like fort som *PLA*. Når man printer med *PETG* så må man gjøre en rekke testprint og kalibrering, samtidig som vi ikke kunne være sikre på at vår 3D-printer kunne lage modellene med lik kvalitet hver gang. For denne prototypen sa vi oss fornøyd med *Impact PLA*, som gir en sterkere modell enn ved å bruke vanlig *PLA*. Det har også blitt printet med en dyse på 0,8mm, der det er mer vanlig å bruke en dyse på 0,4mm. Dette er gjort for å printe fortere, og modellen blir sterkere siden den printer en *plastikkstreng* som er tykkere enn vanlig. Det blir også bedre sammensmelting av lagene.

Kapittel 4

Resultat

4.1 CAD og 3D-print

Arbeidet vi har gjort er å lage og designe en CAD modell for prototype. Denne lar oss feste solcellepanelene og endre vinklingen etter sola. Det er også et tett kammer for elektronikken, og et mer luftig/åpent kammer for måling av luftkvalitet. Prototypen har også egen kabelinnføring fra ytre til indre kammer, noe som reduserer sjansen for at vann kommer inn. I alt har vi en 3D-printet modell som har solcellepanel, kapsling for elektronikken med monteringsmuligheter og har et tett lokk.

4.2 Energihøsting

I den ferdige prototypen har vi da to 3W solcellepanel som energihøsting, og en *MPPT* kontroller som regulerer strøm og spenning på en effektivt måte fra solpanelene.

4.3 Drivere

4.3.1 Sensor

Vi har skrevet drivere for sensorene. En for gassensoren *DGSO3* som bruker *UART*, og en for partikkelsensoren *SPS30*, som bruker I^2C. Disse driverne får dataen ut fra sensorene og over til *LwM2M* driverne.

4.3.2 *LwM2M*

Vi skrev også drivere for *LwM2M*, som tar dataene fra sensordriverne og gjør de mottakelige for *LwM2M* serveren. Vi måtte skrive disse driverne for alle verdier vi ville sende til serveren, altså: temperatur, luftfuktighet, VOC gass, NO_2 gass og de tre partikkelverdiene. De tar verdiene, gjør de om til *float* verdier, og sender tidpunktet for målingen samtidig.

4.4 Skytjeneste

LwM2M serveren Leshan, klarte å hente data og lagre det i databasen, med *JDBC*. *PostgreSQL* databasen holdt dataen i *float* verdier, og hadde et eget tidspunkt for alle verdier. *Grafana* hentet verdiene fra databasen, og viste de fram på dashbordet, som vist i vedlegg [A]. I vedlegget er det også et systemdiagram som viser hvordan dataen blir overført fra steg til steg [A].

4.5 Måling av strøm

4.5.1 Optimalisering av kode og strømforbruk

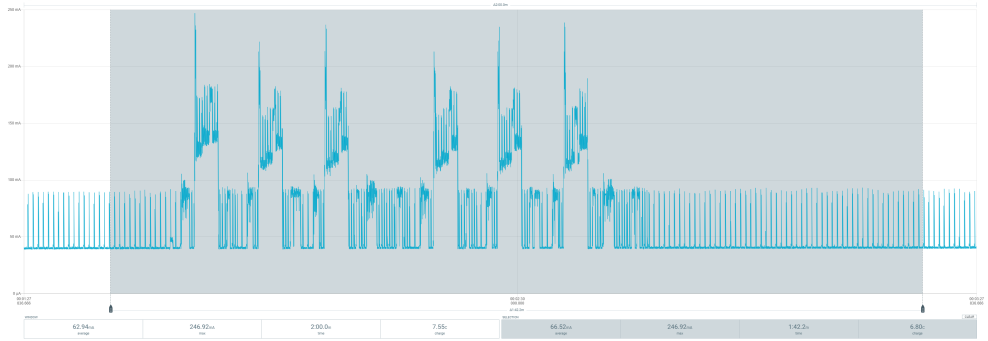
De følgende testene ble gjort etter presentasjonen av oppgaven den 12. mai 2021. Hensikten med disse testene var å få en oversikt over hva som hadde størst strømtrekk i systemet, og deretter optimalisere og justere strømforbruket. Testene ble utført den 13. og 14. mai.

All data er målt ved hjelp av Nordic Semiconductor *PPK2*, og behandlet ved hjelp av *Power Profiler App* fra nRF Connect, eller *matplotlib*.

Test 1 - Baseline

Gjennomsnittlig forbruk: 49,08*mA*
Maks forbruk: 246,92*mA*
Varighet: 11:55 min
Dato: 13.05.2021

Denne testen gir et utgangspunkt for hva forbruket på systemet er, med de samme innstillingene som vi hadde før presentasjonen.

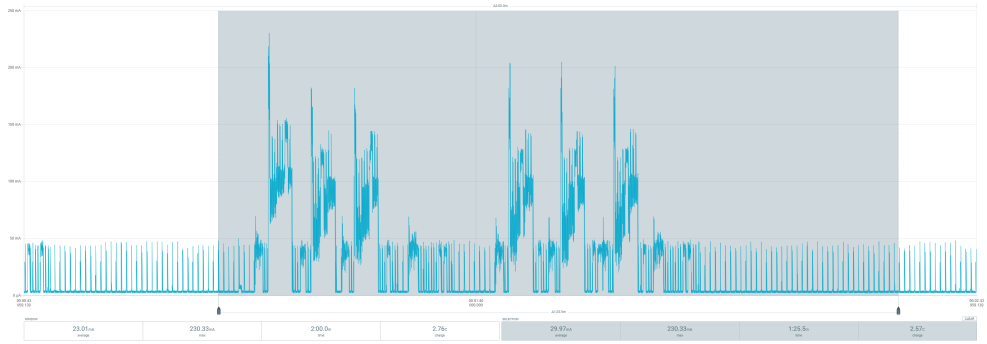


Figur 4.1: Baseline test

Test 2 - Optimalisering 1

Gjennomsnittlig forbruk: 6,46*mA*
Maks forbruk: 204,99*mA*
Varighet: 9:32 min
Dato: 13.05.2021

Denne testen viser den første optimaliseringen som vi gjorde på systemet, der vi skrudde av LED-lyset og systemlogg på *Thingy:91*, samt skrudde på *LwM2M queue-mode*.

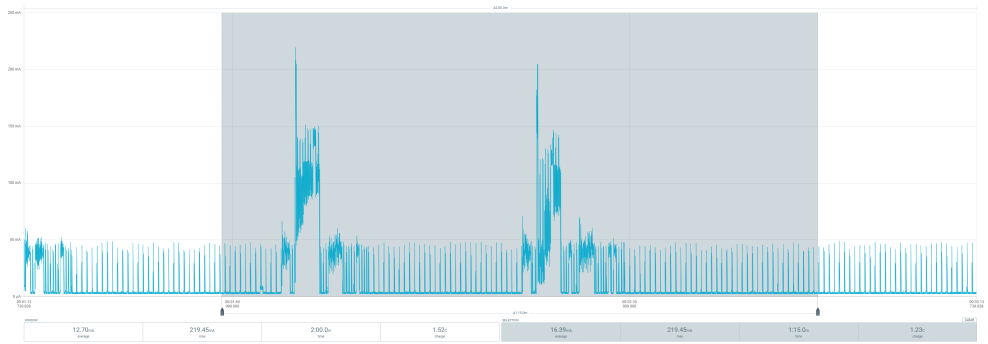


Figur 4.2: Optimaliseringsstest 1

Test 3 - Optimalisering 2

Gjennomsnittlig forbruk: 9,06*mA*
Maks forbruk: 223,89*mA*
Varighet: 18:37 min
Dato: 13.05.2021

Denne testen viser den andre optimaliseringen som vi gjorde på systemet, der vi endret *LwM2M*-driveren til partikkelsensoren, slik at den mellomlagrer dataen som den samler inn.

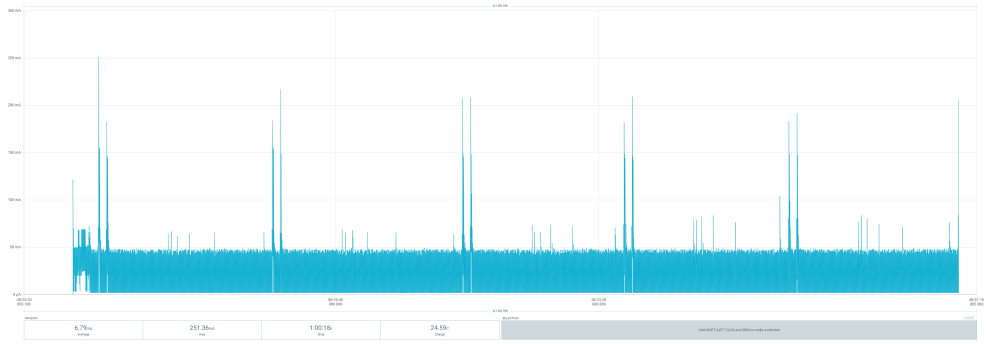


Figur 4.3: Optimaliseringsstest 2

Test 4 - Forbruk over lengre tid

Gjennomsnittlig forbruk: 6,79*mA*
Maks forbruk: 251,36*mA*
Varighet: 1:00:18 timer
Dato: 13.05.2021

Denne testen gir oss et utgangspunkt for hva forbruket på systemet er over lengre tid. Oppsettet er lik testen vist i 4.3

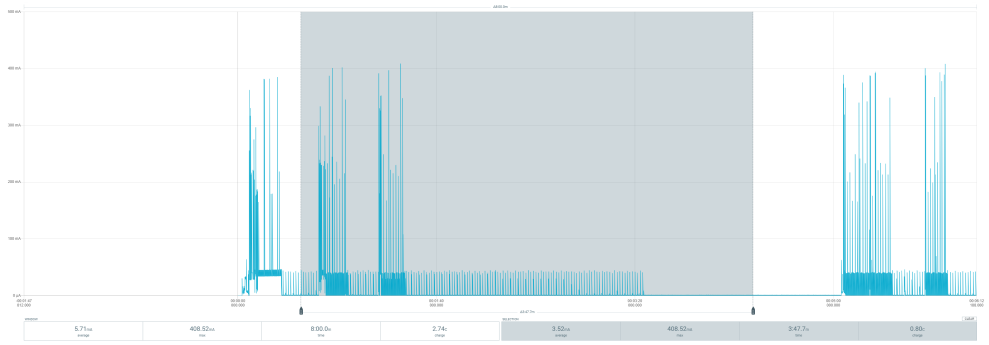


Figur 4.4: Systemforbruk over lengre tid

Test 5 - Feilsøking 1

Gjennomsnittlig forbruk: 3,71*mA*
Maks forbruk: 408,52*mA*
Varighet: 08:00 min
Dato: 14.05.2021

Denne testen viser den første runden med feilsøking vi utførte på hele systemet. Hensikten med denne testen var å prøve å finne ut hva som trakk mest strøm av systemet. Her skrudde vi av driveren for både partikkelsensoren og den interne sensoren på *Thingy:91*, der det effektivt er kun selve *LwM2M*-client applikasjonen som kjører på enheten, uten ekstra programvare. I tillegg så er alle ekstrakomponenter som *MPPT* og partikkelsensoren koblet av *Thingy:91*. Under testen var også *queue-mode* skrudd på.

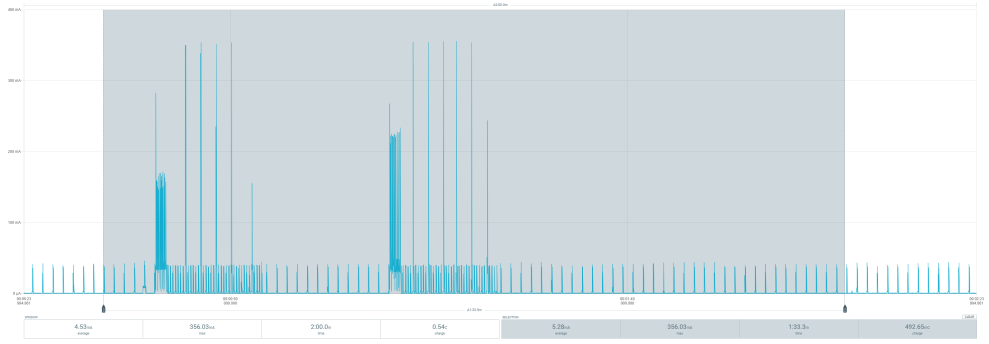


Figur 4.5: Feilsøkingstest 1

Test 6 - Feilsøking 2

Gjennomsnittlig forbruk: 1,9*mA*
Maks forbruk: 358,61*mA*
Varighet: 03:17 min
Dato: 14.05.2021

Denne testen bygger videre på testen illustrert i figur 4.5. Her ser vi bare på strømforbruket til nRF9160 brikken på *Thingy:91*.

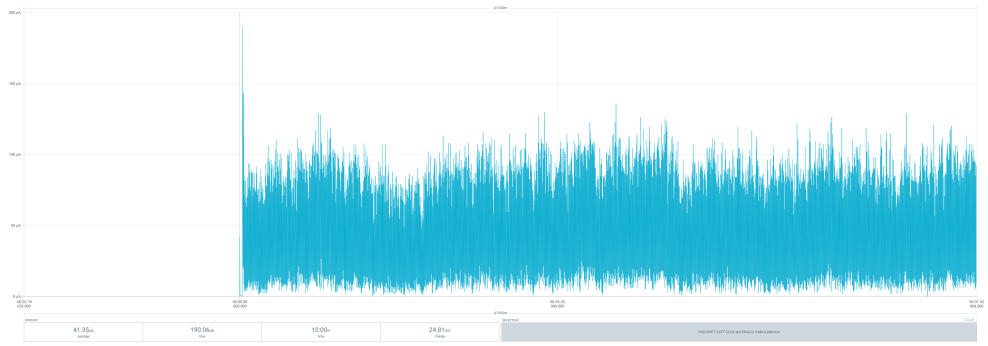


Figur 4.6: Feilsøkingstest 2

Test 7 - Feilsøking 3

Gjennomsnittlig forbruk: 41,35μA
Maks forbruk: 190,06μA
Varighet: 10:00 min
Dato: 14.05.2021

Denne testen bygger videre på testene illustrert i figur 4.5 og 4.6. Her ser vi på strømforbruket på nRF9160 brikken, men på nRF9160DK.



Figur 4.7: Feilsøkingstest 3

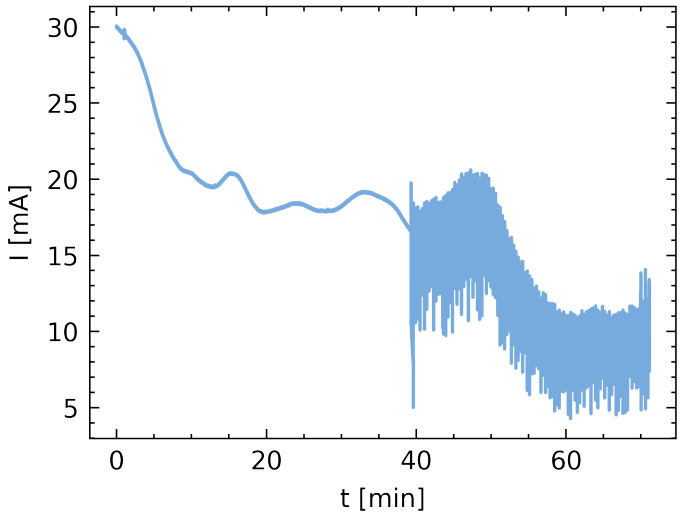
4.5.2 Test av systemforbruk og energihøsting

Hensikten med test av forbruk og energihøsting er å sjekke hvor mye strøm vi får inn, og ut av systemet. Alle målingene ble gjort ut ifra *MPPT*-en.

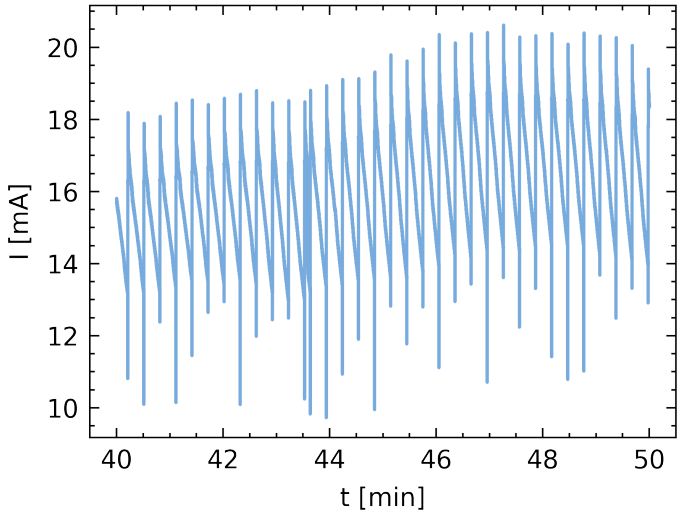
Energihøsting - Test 1 - MPPT

Forhold: Overskyet
Varighet: 1:11:05 timer
Dato: 13.05.2021 kl. 18:10

Figur 4.8 viser strøm fra *MPPT* på en overskyet dag. Spenningen ut ifra *MPPT* ble målt til å være konstant 3,9V. Over denne perioden lå gjennomsnittlig strøm på 16,71mA. I starten av testen lå strømmen på 29mA, deretter falt den ned mot 14mA senere utover kvelden. Etter ca. 40 min begynte det å oppstå sagtannbølger med tanke på strømdataen. På figur 4.9 får man et mer detaljert bilde av dette.



Figur 4.8: Test av MPPT/solpanel, overskyet dag

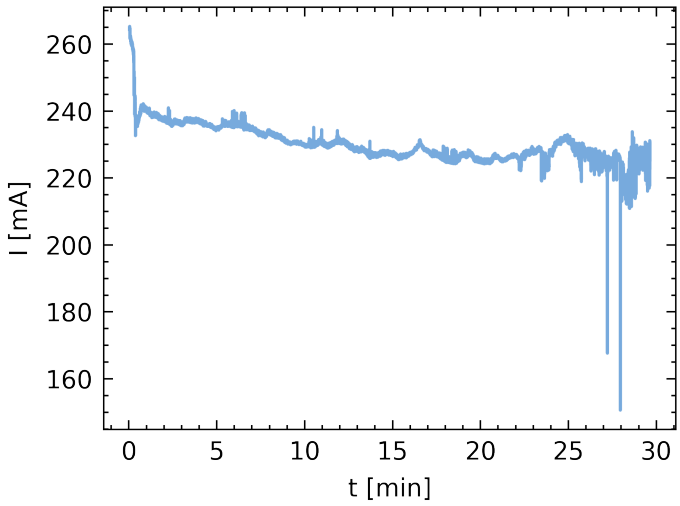


Figur 4.9: Sagtannbølger

Energihøsting - Test 2 - MPPT

Forhold: Sol
Varighet: 0:29:35 t

I denne testen målte vi strømmen på en solfylt dag med 60° vinkling på solpanelet. Figur 4.10 viser den målte strømmen over ca. 30 min. Den hadde et gjennomsnitt på 230*mA*.

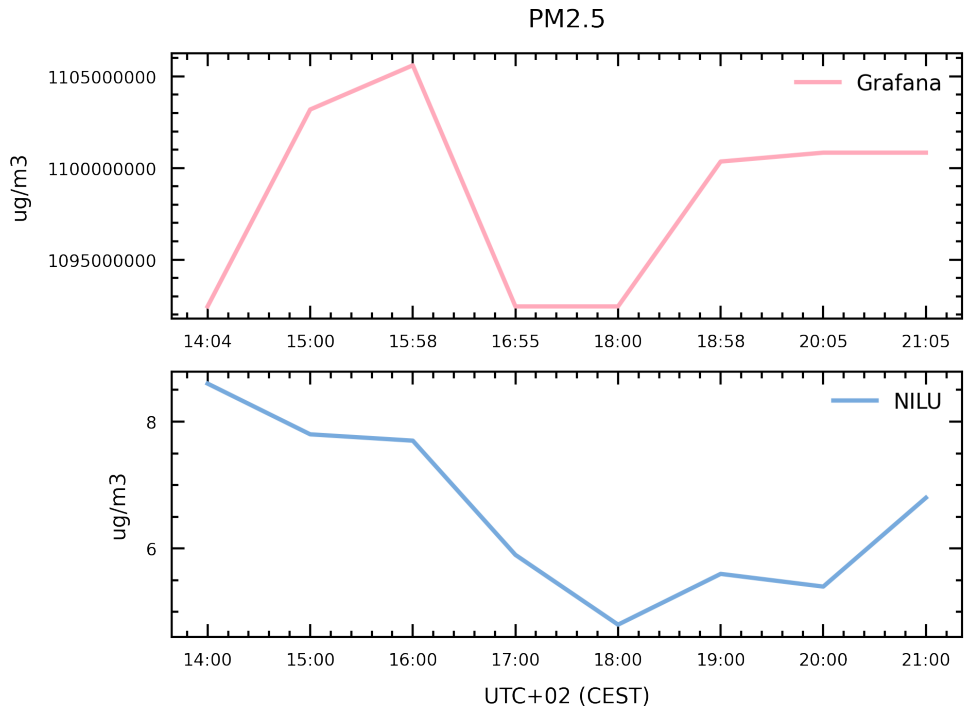


Figur 4.10: Test av MPPT/Solpanel, Solfylt dag

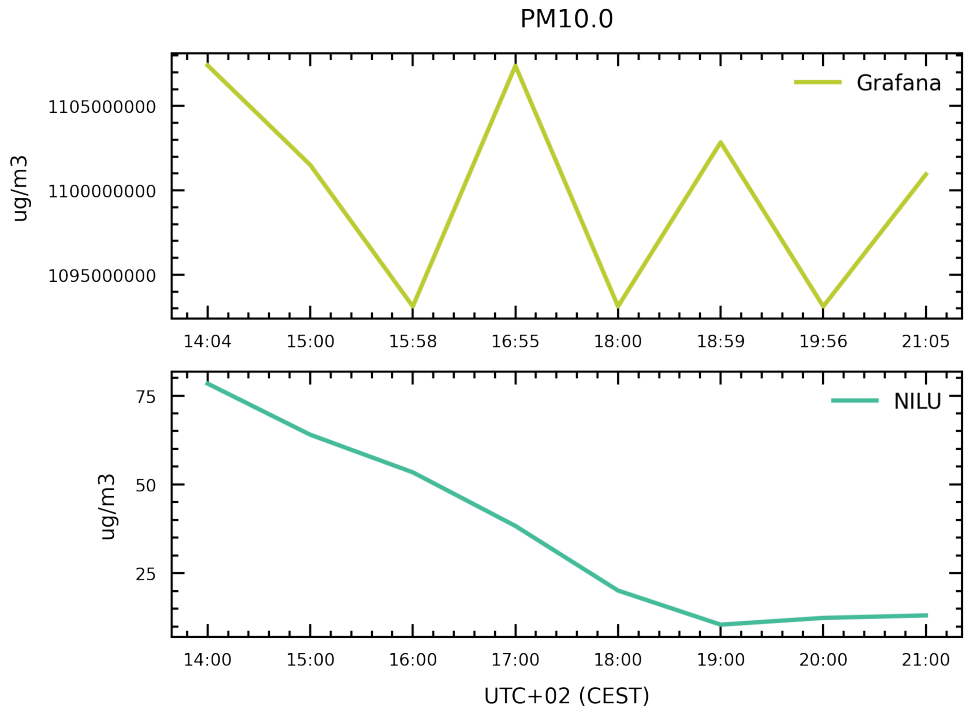
4.6 Verifisering av data

Figur 4.11 og 4.12 viser plottet data av henholdsvis $PM_{2,5}$ og PM_{10}, der vi sammenligner datasettene fra Grafana og *NILU*. Her kan det merkes at dataen fra Grafana er i størrelseorden $1 * 10^9$. Tid vises også som *UTC+02*, der tidsverdiene på Grafana er innen 5 minutter av tidsverdiene på datasettet fra *NILU*, som forklart i [3.6].

Kodeliste 4.1 viser utskriften av to verifikasjonstester av dataen plottet i figur 4.11 og 4.12. Første test av $PM_{2,5}$ har en *r-verdi* tilnærmet lik 0,115. Den har beregnet en *t-verdi* lik 0 og en *p-verdi* lik 0,5, som fører til at test 1, som bruker en tommelfingerregel for *Pearsons korrelasjonskoeffisient* (2.1), feiler. Det samme skjer med test 2, som sjekker om *p-verdien* er under $\alpha = 0{,}5$ (det vil si $p < 0{,}05$). Andre test av PM_{10} gir en *r-verdi* på 0,24, med en *t-verdi* lik 0 og en *p-verdi* lik 0,5. Her feiler også test 1 og test 2.



Figur 4.11: Grafana vs NILU data for $PM_{2,5}$



Figur 4.12: Grafana vs NILU data for PM_{10}

Kodeliste 4.1: Verifiseringstester, Grafana vs NILU data

```
Test - PM2,5
r value:  0,11542000102540292
t value:  0,0
p value:  0,5
Test1 (Pearson RoT):  False
Test2 (T-test):  False

Test - PM10
r value:  0,24058729406838428
t value:  0,0
p value:  0,5
Test1 (Pearson RoT):  False
Test2 (T-test):  False
```

Kapittel 5

Diskusjon

5.1 Resultatmål

I forprosjektet ble det fastsatt ulike resultatmål som vi skulle gjennomføre. I løpet av prosjektet oppsto det ulike problemer, som førte til at noen av resultatmålene ikke ble helt ferdigstilt. Dette kapittelet vil diskutere hvilke mål som ble gjennomført.

5.1.1 Målingsverdier

Et av resultatmålene var at prototypen skulle kunne måle gassverdier som O_3 og NO_2, samt partikler i lufta. Dette har blitt delvis nådd.

Gassensoren funket med nRF9160DK og vi fikk lagret gassverdiene i skyen. Siden *Thingy:91* hadde *UART* linjer som ikke funket med gassensoren, ble den ikke integrert i den ferdige prototypen. Vi hentet derfor data ut fra nRF9160DK, noe som førte til at den ikke ble tatt med i strømbudsjettet til prototypen. Vi hadde en gassensor som målte begge verdier (NO_2 og O_3), der disse ikke kan skilles fra hverandre. Dette gjør det vanskelig å klare å skille ut hva vi faktisk måler.

Partikkelsensoren funket som forventet og ga verdier opp til skyen. Hvorvidt verdiene er legitime tas opp i kapittelet for validering [5.1.5]. Sensoren ble også integrert i den ferdige prototypen, og resultatmålet antas som gjennomført.

5.1.2 Energihøsting

En av resultatmålene som vi fokuserte på i løpet av oppgaven var energihøsting. Spesifikt, så forsøkte vi å bygge prototypen slik at energihøsting-systemet forsynte nok strøm ut hele levetiden til systemet. For å oppfylle dette så må vi sammenligne strømforbruket til systemet, kontra hvor mye strøm som blir forsynt. Figur 4.1 til 4.4 viser endringene av forbruket etter en runde med finjustering og optimalisering av systemet. Her har vi blant annet skrudd av LED-lyset på *Thingy:91*,

samt forbedret *LwM2M*-driveren, som forklart i [5.2.3]. Figur 4.4 viser også en lengre test av systemet, med samme oppsett som figur 4.3. Her ender vi opp med et forbruk på rundt 6-7mA. Basert på et antall mindre tester, så antar vi at dette forbruket forholder seg relativt konstant og benytter dette som det statiske forbruket til systemet.

Etter disse testene ble det også oppdaget et generelt høyere forbruk enn det vi forventet. Som illustrert med figur 4.5, 4.6, og 4.7, ble det gjort en runde med feilsøking på hele systemet. Her koblet vi av alle eksterne komponenter for å prøve å isolere hvor det uventede strømforbruket kom fra. Det ble også brukt samme programvare som på testene forklart over, men på disse testene ble også de eksterne sensorene skrudd av i *prj.conf*. Her kan vi observere et gjennomsnittlig forbruk på 3,71mA. Måler vi bare fra nRF9160 så kan vi observere et gjennomsnittlig forbruk på 1,9mA, som er betraktelig bedre, men fortsatt høyere enn det vi forventet. Måler vi det samme forbruket på nRF9160DK, så kan vi observere et forbruk på 41,35μA. Dette kan bety at *Thingy:91*, som et system, påvirker det totale forbruket betraktelig mer enn det vi først antok. Siden vi hadde problemer med å videre feilsøke systemet, så aksepterte vi forbruket slik det var. Men, det kan være relevant å videre finne ut hvor dette forbruket ligger, for å optimalisere systemet for mere krevende *low-power* løsninger i fremtiden.

På bakgrunn av dette så kan vi se på hvor mye strøm som blir forsynt til systemet. For dette kan vi se på testene illustrert av figur 4.8, og 4.10. Disse testene gir oss en pekepinne på hvor mye strøm vi får inn fra energihøsting-systemet ved ulike forhold. Her får vi da en gjennomsnittlig strøm på 16,71mA ved overskyet vær, og en gjennomsnittlig strøm på 230mA ved solfylt vær. Begge verdiene er målt fra *MPPT*. Som forklart i [3.2.2] så forventer vi en del variasjoner rundt hvor mye effekt som blir mottatt av energihøsting-systemet, og derav hvor mye strøm som blir forsynt inn i systemet. Eksakt hvor mye variasjonen ligger på er vanskelig å si, men det kan antas at variasjonsområdet kan ha et nedre grense som er langt under forbruket på systemet. Det samme gjelder også ved kveldstid, men her har vi et mere redusert tidsrom der systemet går i underskudd. Her kommer da batterikapasiteten inn i spill. Gitt det statiske forbruket vi har, gir en nedre utladningtid på $1400mAh/7mA = 200$ timer, som gir en betraktelig stor buffer fram til neste oppladningssyklus ved dagtid. Vi antar derfor at dette ikke blir det største problemet til selve systemet.

Det blir derimot vanskelig å kommme med konkrete tall på hvor lenge systemet vil vare, bare basert på de testene vi har utført hittil. Her hadde det vært foretrukket å ha et større datasett, der vi ser på forbruket over et helt år. Dette datasettet kan gi et bedre innblikk på hvor stor variasjonen vil være, samt utladningshastigheten til systemet ved lav forsyningsstrøm. Ladekontroller som vi kalte det i starten, som nå er en SPV1050 *MPPT*, gjør energihøstingen mer effektiv. Den varierende energien fra solcellepanel blir nå en kontrollert ladespenning og strøm.

5.1.3 Skytjeneste

Server Infrastruktur

Intel NUC var mer enn bra nok for det vi brukte den til. *port forward*ing av *CoAP* portene gikk ganske smertefritt. Vi opplevde ett problem når strømmen gikk og *IP*-en til serveren ble byttet, men en rask endring på *DNS*-en vi brukte fikset det.

5.1.4 Leshan

Leshan var ikke første valget. Vi byttet mellom flere protokoller, før vi endte med *LwM2M*. Først brukte vi *MQTT* i *Python*. Deretter gikk vi over til *coapthon*, som er en *Python* implementasjon av *CoAP*. Til slutt gikk vi over til *Leshan*, som er en *Java* implementasjon av *LwM2M*. Grunnen var at denne støttet *FOTA*, noe vi tenkte på å implementere i starten av oppgaven, hvis vi hadde nok tid igjen. *FOTA* hadde vært fint hvis vi ville oppdatere mange flere sensornoder fordelt utover et stort areal.

ZeroTier One funket bra, og ga oss sikker tilgang, uten at vi trenge å bekymre oss om *port forward*ing. Dette gjør at vi har eksklusiv tilgang til serveren.

PostgreSQL databasen funket bra, og var enkel å jobbe med, etter at vi lærte å bruke SQL.

Grafana dashbord funket utmerket, og var veldig greit å jobbe med. Av og til opplevde vi at denne ikke ville fungere. Dette ble løst ved å restarte en *Daemon* på serveren.

FOTA

Valgte å ikke implementere *Firmware Over The Air* (*FOTA*), etter anbefaling fra Nordic med tanke på dårlig tid. Dette var heller en ettertanke, hvis vi hadde god tid igjen på slutten av oppgaven.

5.1.5 Verifisering

Den siste delen av problemstillingen til oppgaven handlet om å verifisere den måledataen vi får ut fra systemet, for å gi et slags grunnlag på hvorvidt systemet faktisk fungerer optimalt. Her sammenlignet vi et datasett som vi samlet inn selv, og et datasett fra *Norsk institutt for luftforskning* (*NILU*). Disse datasettene var fra samme tidsperiode, og på relativt samme lokasjon i Trondheim. I valideringstesten brukte vi bare $PM_{2,5}$, og PM_{10}, siden *NILU* desverre ikke hadde data tilgjengelig for NO_2 i det gitte tidsrommet. Hvis vi sammenligner resultatene gitt av figurene 4.11, og 4.12 så ser vi en ganske klar forskjell mellom de to datasettene, spesielt på

PM_{10} der dataen varierer kraftig. Vi kan også se at dataen for $PM_{2,5}$ er gitt i størrelsesorden $1 \cdot 10^9$. Dette kan være grunnet av både dårlig kalibrering, samt måten *LwM2M*-driveren behandler de verdiene som partikkelsensorene samler inn. Her blir ikke verdiene riktig delt opp i forhold til heltall og desimaltall, siden verdiene som kommer fra sensoren blir sendt som en 32-bits *float* verdi. Dette kan påvirke størrelsesordenen til verdiene.

Vi kan også se på verifiseringstesten som ble utført på de to datasettene, gitt av kodeliste 4.1. Her ser vi at både datasettene for $PM_{2,5}$ og PM_{10} feilet verifikasjonstestene, og at *r-verdien* ligger på ca. 0,1, der en verdi på 1 viser til full (positiv) korrelasjon og en verdi på 0 viser til ingen korrelasjon. Vi antar derfor at datasettene ikke er korrelert, og dataen som vi henter inn er da ikke gyldig.

5.2 Forbedringer

5.2.1 Protokoll

For vårt bruk kunne vi like godt ha brukt *MQTT* eller *CoAP* uten *LwM2M* for å sende og hente data. Vi har heller ikke brukt *MQTT* eller *CoAP* på nRF9160 som gjør det vanskelig å anbefale, men med den erfaringen vi har så vet vi at det er en mindre avansert protokoll. *MQTT* er veldig enkelt, men lite fleksibelt som gjør det mindre effektivt for noe annet enn å sende verdier ved forespørsel fra klienten. *CoAP* er veldig fleksibelt og er lett å oversette til *HTTP*.

LwM2M er mer komplisert, og er bedre egnet for styring (write, exec, delete) og oppdatering (*FOTA*). Den er også mer egnet for at serveren spør klienten om data, så henter klienten det, mot det vi har lyst på med målinger med jevne mellomrom i tid, selv om det går ann å bruke *"observe"* for å få klienten til å bestemme når data skal sendes. Det er litt trøbbel med *"queue mode"* som gjør at den ikke våkner og registrerer seg innen 5 min. Dette skjer etter 10 min etter at den har ingen "levetid" igjen, uten at vi vet hvorfor.

Vi brukte ikke de standardiserte ressursene for interoperabilitet med andre sensornoder, eller *Bootstrap*. Vi brukte *DTLS* for sikkerhet, men vi erfarte at dette ikke er nødvendig. En bedre løsning er å bruke objektet *3428* for målingsverdiene, som er designet for AQI med riktige mål og tidsperioder. Dette gjelder for målingsverdiene PM_{10}, $PM_{2,5}$, PM_1, CO, SO_2, O_3, NO_2, CO_2, NO og H_2S.

5.2.2 Server Hosting

I stedet for å hoste sin egen server, slik vi gjorde, er det mulig å bruke andre skytjenester. *LwM2M* server, dashbord, og database kan alle bli kjørt i skyen. Dette kunne bli dyrere for oss, siden vi hadde en server tilgjengelig som vi kunne bruke, og gratis alternativene er litt avgrenset i funksjonalitet. Men for andre kunne

dette være en bedre løsning. *Grafana* har en egen skytjeneste-løsning som heter "Grafana Cloud" som kan prøves gratis, og er enkel å sette opp. PostgreSQL kan hostes av alle store selskap Azure, AWS, Google Cloud, osv. Alle kan prøves gratis, og settes opp på kort tid. AVSystem kan hoste en *LwM2M* server som kan kobles opp mot Azure eller AWS.

Det er usikkert hvilken problemer som følger, siden vi ikke personlig har testet disse selv. De er derimot gode alternativ, hvis en trenger å skalere til en større operasjon.

5.2.3 Strømbudsjett

Mye av de problemene som vi har hatt i løpet av oppgaven kan komme fra strømbudsjettet [A.4] som ble utviklet. Som forklart i 3.2.3 så baserer budsjettet seg på beregningene gjort med 3.2.2. Dette gir en begrenset mengde med effekt tilgjengelig. Vi hadde sett for oss at *Thingy:91* skulle ha et statisk strømtrekk på $300\mu A$, men det ville vært med en svært strømgjerrig programvare, og uten eksterne sensorer tilkoblet.

Vi kan også se på de begrensningene vi hadde i forhold til batteriet, som hadde en stor påvirkning på strømbudsjettet til oppgaven. Hadde vi sett på et større batteri så kunne vi klart oss med å sette et tilsvarende batteri vi allerede har, i parallell. Dette ville krevd mer undersøkelser rundt hvordan det skal løses i praksis. I teorien så kunne vi holdt oss med *Thingy:91* sin *PMIC*.

Vi endte opp med store solpanel for å sikre oss nok effekt. Vi kunne ha hatt enda større panel, men dette ville påvirke størrelsen på selve prototypen. Videre ville dette økt printetiden, og hadde gitt oss et for stort areal å utnytte til elektronikk og nødvendige komponenter.

Da vi søkte etter en *MPPT* kontroller, så fant vi en som passet systemet ganske bra. Den ville ikke være perfekt, men slik vi skjønte det så traff den mange av kriteriene til batteriet vårt. I følge [STMicroelectronics 2014] så ville den gi en ladestrøm fra $30\mu A \leq I_{MP} \leq 20\mu A$, men databladet refererte kun til hva slags solcellepanel den var best egnet til. Det vi faktisk får ut av *MPPT* kontrolleren ved strålende sol er betraktelig høyere, spesielt enn det *MPPT* skal gi ifølge databladene. Her sier databladet til evalueringsbrettet at man maks får $100mA$, der databladet til SPV1050 sier at den kun gir en ladestrøm på $70mA$. Vi ønsker å endre V_{UVP} i *MPPT* til 3,3V, men samtidig så har den ingen kontroll over batteriet slik det er nå. *Thingy:91* sin *PMIC* har kontroll over batteriet, så vi får ikke utnyttet alle funksjonene til *MPPT*-en, men får bare ladet opp batteriet til V_{EOC} til 4,25V.

5.2.4 Sensorer

I forhold til sensorene som vi brukte i oppgaven, så bruker vi nå heltall på dataformatet til partikkelsensoren. Dette fører til at vi ikke vet hvor verdiene etter komma starter. Her kan det lages en funksjon som sender opp riktig formatert verdi til skytjenesten. Disse verdiene endrer seg også mye mindre enn forventet. Dette kan skyldes at sensorene ikke er kalibrert riktig.

På bakgrunn av mengde effekt tilgjengelig måtte vi begrense måletiden til sensorene for å sikre lavt nok forbruk, og for å klare å oppnå resultatmålene nevnt i [1.4]. Vi benyttet oss av et målingsintervall på 5 min. Ved å endre dette mistenker vi at strømforbruket ville ha blitt redusert kraftig. Her kan for eksempel 1 time måleintervall være nødvendig. Dette sammenfaller også med målingsintervallene benyttet av målestasjonene til *NILU*.

Det var uvisst hvor mye *buck-boost* konverteren brukte av strøm. I tillegg befant det seg et LED lys på kretskortet til konverteren, som kunne bidra til et økt strømforbruk. Ved å benytte sensorer med forsyningsspenning på 3,3V ville vi unngått å bruke konverteren, og kunne heller koblet de direkte til *Thingy:91*.

Når vifta til partikkelsensoren sto på i for kort tid, så ble noe av dataen korrupt. De verdiene er karakterisert med at konsentrasjonen av $PM_{2,5}$, og PM_{10} blir satt til null, og at "*typisk størrelse*" ikke endret seg. Ved å øke tiden på vifta til 5 sekunder, fikk vi utelukket de korrupte verdiene. Dette medførte at partikkelsensoren brukte mer strøm enn først forventet. Derfor valgte vi å bruke kort forsinkelse og heller filtrere ut data i ettertid. Det kunne vært mer gunstig å heller fokusert på å få legitime målingsdata, og deretter prøve å optimalisere mest mulig rundt dette. Det vil si å fokusere mer på datakvaliteten enn strømbudsjettet.

Som diskutert i [5.1] så samstemmer ikke datasettet som vi har hentet inn med de fra *NILU*. En av grunnene til dette kan være kalibreringen til sensorene. I løpet av oppgaven ble det ikke utført noen slags form for kalibrering, der sensorene ble brukt med en gang vi fikk tak i de. Her kan det ha oppstått et statisk avvik mellom verdiene, som kan ha vært med å påvirke det endelige resultatet. Her kunne vi ha brukt mere tid på kalibrering, for å få mere nøyaktige måleverdier for verifisering.

Den situasjonen verden er i nå, i forhold til COVID-19, har også påvirket utviklingen av sensordriverne til oppgaven. Dette er grunnet lang leveringstid, der sensorene ble bestilt fra leverandører som blant annet Mouser og Digikey. Dette hadde en direkte påvirkning på utviklingstiden. Vi fikk ikke utført fysiske tester på sensorene i tide for integrering. Dette førte til at mye av integreringstiden ble forsinket, som hindret mye av mulighetene vi hadde til å etterprøve og forbedre designet vårt. Vi klarte derimot å utvikle fungerende drivere på tross av dette. Vi utviklet våre egne simuleringer av sensorene ved hjelp av *Arduino* mikrokontrollere.

5.2.5 Thingy:91

I løpet av oppgaven så brukte vi *Thingy:91* som en generell platform for datainnhenting og opplasting til skytjenesten. Det viste seg i etterkant at for vår bruk, så var ikke *Thingy:91* den mest optimale løsningen. Selv om enheten gir mye funksjoner som er veldig nyttig i en *IoT* sammenheng som innebygde sensorer og batteri, så er den ikke den beste løsningen for å prototype nye løsninger. Mest på grunn av et lite antall tilgjengelige kontakter, som gjør det vanskelig å koble til ekstra komponenter eller sensorer. Dette kom ekstra godt fram når vi skulle koble til de forskjellige sensorene [5.1]. Her var de fleste av de tilgjengelige kontaktene på nRF9160 allerede koblet til andre komponenter på enheten. Dette førte til at vi ble tvunget til å koble oss på testpunkt på kretskortet. Vi hadde også mulighet til å koble oss på en pinheader (*P6*), men denne var tilkoblet nRF52 brikken, som ville ha ført til at vi måtte gå igjennom nRF52 med *UART* for å kommunisere med sensorene. Dette hadde økt kompleksiteten, og derav feilmarginen, ytterligere.

5.3 Dokumentasjon

Utover oppgaven har vi brukt Nordic Semiconductor sin dokumentasjon for opplæring. Som bakgrunn har vi drevet med programmering av mikrokontrollere som Arduino, og Intel FPGA. Overgangen til større verktøy som egen *toolchain*, *devicetree*, *Zephyr* osv. har derfor vært et hakk vanskeligere. Derfor krevdes det at dokumentasjon var tilstrekkelig god. Formålet med dette kapittelet er derfor gi en tilbakemelding på Nordic sin dokumentasjon for videre forbedring.

5.3.1 nRF Connect

Når vi startet med hovedprosjektet var det vanskelig å finne fram til en guide. "*Getting started*"-guidene som ligger inne på infosenteret til Nordic tar for seg veldig spesifikke eksempler. Med mer variasjon ville det ha gitt oss mere hjelp. Når vi jobbet med sensordrivere fant vi fort ut at det ikke fantes mange eksempler på I^2C/*UART* kommunikasjon. For det meste brukte vi enhetstester fra *Zephyr* bibliotekene for å få et innblikk på hvordan vi kunne bruke bibliotekene for I^2C og *UART*.

5.3.2 Programmering av enhet

Programmering av Thingy:91 var veldig tungvindt, da man må programmere via nRF Connect Programmer i DFU Bootmode. Dette er en unødvendig tungvindt prosess, der man heller kunne ha programmert ved hjelp av kommandolinjen. Når PC-en går i søvn modus krasjer også all nRF Connect programvaren.

5.3.3 Power optimization

Det fantes også ikke noe særlig opplæring rundt lavspenning optimalisering. Mye av det vi optimaliserte fant vi fra ulike *Devzone* foruminnlegg. Vi fant en artikkel rundt dette på Nordic sin dokumentasjon [Nordic 2021d], men denne kunne ha vært mere utdypende.

5.3.4 Power Profiler

nRF Connect Power Profiler ble brukt til å visualisere den dataen vi samlet inn ved hjelp av *PPK2*. Da vi åpnet tidligere tester møtte vi på noen problemer. Slik som måten å bevege seg i grensesnittet ble komplisert, med snarveier som ga lite mening. Noen ganger hoppet man helt til slutten av grafen, og måtte derfor bevege seg helt til starten. Det var også vanskelig å få stilt inn Δt for å se på data over en periode lenger enn 2 minutt. Manuell innstilling av tidsperioden hadde vært mest gunstig.

Kapittel 6

Konklusjon

I løpet av oppgaven så har vi sett på om det er mulig å bruke *Thingy:91* fra Nordic Semiconductor som en målingsstasjon for luftkvalitet, ved å bruke teknologier som energihøsting og sensorer. I løpet av oppgaven har vi også støttet på flere problemer, som førte til at noen av resultatmålene som ble utviklet i begynnelsen av prosjektet, ikke ble oppfylt. Det fullstendige produktet ble en prototype som måler luftkvalitet, med unntak av måleverdier for O_3, og NO_2. Den har en løsning for energihøsting som skulle lade batteriet under hele dens levetid, men vi er usikker på om dette målet er fullt oppnådd. Ut ifra utregninger fra grafer av strømtrekk og energihøsting, så observerte vi at strømtrekket fra systemet var gjennomsnittlig mindre enn strømmen fra energihøstingen. Vi kan anta at systemet har mulighet til å opprettholde seg gjennom mesteparten av året, med unntak av mørketiden. For å bekrefte dette kreves det videre testing, helst gjennom hele året. Basert på statistiske tester mellom måledataen som vi samlet inn med måledata fra *NILU*, så konkluderer vi med at vår måledata ikke er gyldig.

Videre anbefaler vi å ikke bruke *Thingy:91* til en slik oppgave, mest grunnet mangler på tilgjengelige kontakter. Det er også mulig å bruke større batteri, for å øke mengden tilgjengelig strøm. Målingsintervallet kan også økes til hver time, som kan påvirke strømforbruket betraktelig. Ved å bruke sensorer med passende forsyningsspenning, er det mulig å redusere bruken av level-shifting/regulatorer. I forhold til protokoller og skytjeneste så er det kanskje bedre å bruke protokoller som *CoAP*/*MQTT* enn *LwM2M*, alt etter kompleksiteten på sensorsystemet. En annen løsning for hosting av servertjeneste (som for eks. AWS/Azure) kan også være relevant.

Bibliografi

[AVS19] AVSystem. *LwM2M - Lightweight M2M standard - protocol and its benefits*. AVSystem. 17. jan. 2019. URL: https://www.avsystem.com/blog/lightweight-m2m-lwm2m-overview/ (sjekket 06.05.2021).

[Bos17] Bosch. *BME680. Low power gas, pressure, temperature humidity sensor*. Jul. 2017. URL: https://cdn-shop.adafruit.com/product-files/3660/BME680.pdf (sjekket 12.02.2021).

[Cir16] Circuitbasics. *i2cillu*. Jan. 2016. URL: https://www.circuitbasics.com/wp-content/uploads/2016/01/Introduction-to-I2C-Message-Frame-and-Bit-2.png (sjekket 03.04.2021).

[CS21] Scott Chacon og Ben Straub. *Pro Git*. 2. utg. Apress, 28. apr. 2021, s. 18–21. URL: https://git-scm.com/book/en/v2/Getting-Started-Installing-Git (sjekket 25.05.2021).

[Do13] Dr. Michelle M. Do. *LTE: Tracking Area (TA) and Tracking Area Update (TAU)*. 30. aug. 2013. URL: https://www.netmanias.com/en/post/blog/5930/lte-tau/lte-tracking-area-ta-and-tracking-area-update-tau (sjekket 18.05.2021).

[Ecl15] Eclipse. *OMA Lightweight M2M server and client in Java*. Eclipse Foundation. 2015. URL: https://www.eclipse.org/leshan/ (sjekket 03.03.2021).

[Ecl20] Eclipse. *LWM2M Supported features*. Eclipse Foundation. 24. aug. 2020. URL: https://github.com/eclipse/leshan/wiki/LWM2M-Supported-features (sjekket 12.03.2021).

[EEA21] EEA. *The European Air Quality Index*. European Environment Agency. 11. mai 2021. URL: https://www.eea.europa.eu/themes/air/air-quality-index (sjekket 12.05.2021).

[Ele21] Electricimp. *uartillu*. 2021. URL: https://developer.electricimp.com/sites/default/files/attachments/images/uart/uart3.png (sjekket 28.03.2021).

[EUR08] EUR-Lex. *Directive 2008/50/EC of the European Parliament and of the Council of 21 May 2008 on ambient air quality and cleaner air for Europe*. 21. mai 2008. URL: https://eur-lex.europa.eu/legal-content/en/ALL/?uri=CELEX%5C%3A32008L0050 (sjekket 13.05.2021).

[Eur21] European-Commision. *Photovoltaic Geographical Information System (PVGIS)*. 24. mar. 2021. URL: https://ec.europa.eu/jrc/en/pvgis (sjekket 30.04.2021).

[Fre21] FreeRTOS. *What is an RTOS?* 2021. URL: https://www.freertos.org/about-RTOS.html (sjekket 24.05.2021).

[Gra21] Grafana. *Install on Debian or Ubuntu*. Grafana Labs. 2021. URL: https://grafana.com/docs/grafana/latest/installation/debian/ (sjekket 01.03.2021).

[iBA21] iBASIS. *IoT NETWORK COVERAGE*. iBASIS Inc. 2021. URL: https://ibasis.com/solutions/iot-connectivity/network-coverage/.

[ID21] Barbara Illowsky og Susan Dean. *Testing the Significance of the Correlation Coefficient*. [Online; accessed 2021-05-09]. 10. mar. 2021. URL: https://stats.libretexts.org/@go/page/800 (sjekket 09.05.2021).

[Ins19] Texas Instruments. *TPS63070 Buck/boost converter*. 2019. URL: https://www.ti.com/general/docs/suppproductinfo.tsp?distId=10&gotoUrl=http%253A%252F%252Fwww.ti.com%252Flit%252Fgpn%252Ftps63070 (sjekket 25.05.2021).

[ITT19] ITT. *Network of low-cost distributed monitoring sensors to measure and map air and water quality*. Institute for Transformative Technologies. 2019. URL: https://50breakthroughs.org/3/overview/bt/ (sjekket 23.05.2021).

[Kim15] Tae Kyun Kim. «T test as a parametric statistic». I: *Korean Journal of Anesthesiology* 68.6 (2015), s. 540. DOI: 10.4097/kjae.2015.68.6.540. URL: https://doi.org/10.4097/kjae.2015.68.6.540.

[Kir08] Wilhelm Kirch. «Pearson's Correlation Coefficient». I: *Encyclopedia of Public Health*. Springer Netherlands, 2008, s. 1090–1091. DOI: 10.1007/978-1-4020-5614-7_2569. URL: https://doi.org/10.1007/978-1-4020-5614-7_2569.

[Kre04] Timothy C. Krehbiel. «Correlation Coefficient Rule of Thumb». I: *Decision Sciences Journal of Innovative Education* 2.1 (jan. 2004), s. 97–100. DOI: 10.1111/j.0011-7315.2004.00025.x. URL: https://doi.org/10.1111/j.0011-7315.2004.00025.x.

[Le20] Cuong Phu Le. *Energiforvaltning og Teknologier for Energihøsting*. Forelesning. 5. okt. 2020.

[Lig18] Raquel Ligero. *Differences between Nb-IoT and LTE-M*. accent systems. 3. mai 2018. URL: https://accent-systems.com/blog/differences-nb-iot-lte-m/?v=c2f3f489a005 (sjekket 18.05.2021).

[Lin19] Linuxize. *How to Install PostgreSQL on Debian 10*. 25. okt. 2019. URL: https://linuxize.com/post/how-to-install-postgresql-on-debian-10/ (sjekket 01.03.2021).

[LS11] Joe Landsberg og Peter Sands. «Weather and Energy Balance». I: *Physiological Ecology of Forest Production*. Elsevier, 2011, s. 13–48. DOI: 10.1016/b978-0-12-374460-9.00002-0. URL: https://doi.org/10.1016/b978-0-12-374460-9.00002-0 (sjekket 24.05.2021).

[NIL21a] NILU. *Måledata for luftkvalitet*. *Norsk institutt for luftforskning* (*NILU*). 2021. URL: https://luftkvalitet.nilu.no/ (sjekket 16.05.2021).

[NIL21b] NILU. *Urban Luftkvalitet*. *Norsk institutt for luftforskning* (*NILU*). 27. jan. 2021. URL: https://www.nilu.no/forskning/urban-luftkvalitet/ (sjekket 23.05.2021).

[Nor19] Nordic. *Nordic Thingy:91 User Guide*. v1.0. Nordic Semiconductor. 22. aug. 2019. URL: https://infocenter.nordicsemi.com/pdf/Thingy91_UG_v1.0.pdf (sjekket 09.05.2021).

[Nor21a] Nordic. *nRF9160. Low power SiP with integrated LTE-M/NB-IoT modem and GPS*. Nordic Semiconductor. 2021. URL: https://www.nordicsemi.com/Products/Low-power-cellular-IoT/nRF9160 (sjekket 23.05.2021).

[Nor21b] Nordic. *Power Profiler Kit II*. Nordic Semiconductor. 26. feb. 2021. URL: https://infocenter.nordicsemi.com/index.jsp?topic=%2Fug_ppk2%2FUG%2Fppk%2FPPK_user_guide_Intro.html (sjekket 24.05.2021).

[Nor21c] Nordic. *Power saving mode setting +CPSMS*. Nordic Semiconductor. 7. mai 2021. URL: https://infocenter.nordicsemi.com/index.jsp?topic=%2Fref_at_commands%2FREF%2Fat_commands%2Fnw_service%2Fcpsms.html (sjekket 18.05.2021).

[Nor21d] Nordic. *poweropt*. Nordic Semiconductor. 24. mai 2021. URL: https://developer.nordicsemi.com/nRF_Connect_SDK/doc/latest/nrf/app_power_opt.html (sjekket 24.05.2021).

[NTN21] NTNU. *Prosjektmanual. For emnet TELE3001/3021/3031 Bacheloroppgave elektro*. 11. jan. 2021. (Sjekket 10.02.2021).

[NXP14] NXP. «UM10204 I2C-bus specification and user manual». en. I: (4. apr. 2014), s. 64. URL: https://www.nxp.com/docs/en/user-guide/UM10204.pdf.

[PGD21] PGDG. *PostgreSQL: The World's Most Advanced Open Source Relational Database*. The PostgreSQL Global Development Group. 2021. URL: https://www.postgresql.org/ (sjekket 01.03.2021).

[Pol19] Polyalkemi. *CE3Pro*. 2019. URL: https://polyalkemi.no/creality-ender-3-pro/ (sjekket 24.05.2021).

[Pol21a] Polyalkemi. *Egenskaper Impact PLA*. 2021. URL: https://polyalkemi.no/fiberlogy-impact-pla-graphite/ (sjekket 24.05.2021).

[Pol21b] Polyalkemi. *Egenskaper PETG*. 2021. URL: https://polyalkemi.no/fiberlogy-petg-transparent/ (sjekket 24.05.2021).

[Pro20] Zephyr Project. *About the Zephyr Project*. 2020. URL: https://www.zephyrproject.org/learn-about/ (sjekket 24.05.2021).

[See17] Seeed-Studio. *Small Solar panels*. 2017. URL: https://www.seeedstudio.com/3W-Solar-Panel-138X160.html (sjekket 23.05.2021).

[Sem21a] Nordic Semiconductor. *Installing the nRF Connect SDK manually*. 24. mai 2021. URL: https://developer.nordicsemi.com/nRF_Connect_SDK/doc/latest/nrf/gs_installing.html (sjekket 25.05.2021).

[Sem21b] Nordic Semiconductor. *Installing the nRF Connect SDK through nRF Connect for Desktop*. 24. mai 2021. URL: https://developer.nordicsemi.com/nRF_Connect_SDK/doc/latest/nrf/gs_assistant.html (sjekket 25.05.2021).

[Sem21c] Nordic Semiconductor. *nRF Command Line Tools*. 20. apr. 2021. URL: https://www.nordicsemi.com/Software-and-tools/Development-Tools/nRF-Command-Line-Tools/Download (sjekket 25.05.2021).

[Sem21d] Nordic Semiconductor. *Programming applications on Nordic Thingy:91*. 10. mai 2021. URL: https://infocenter.nordicsemi.com/index.jsp?topic=%2Fug_nc_programmer%2FUG%2Fnrf_connect_programmer%2Fncp_programming_appln_thingy91.html (sjekket 25.05.2021).

[Sen20] Sensirion. *Datasheet SPS30 - Particulate Matter Sensor for Air Quality Monitoring and Control*. Sensirion. Mar. 2020. URL: https://www.sensirion.com/fileadmin/user_upload/customers/sensirion/Dokumente/9.6_Particulate_Matter/Datasheets/Sensirion_PM_Sensors_Datasheet_SPS30.pdf (sjekket 24.05.2021).

[Sen21] Sensirion. *zephyr/include/i2c.h*. Mai 2021. URL: https://github.com/Sensirion/arduino-sps (sjekket 24.05.2021).

[SHB14] Z. Shelby, K. Hartke og C. Bormann. *The Constrained Application Protocol (CoAP)*. Internet Engineering Task Force (IETF). Jun. 2014. URL: https://tools.ietf.org/html/rfc7252.

[Spe17] Spec. *Digital Gas Sensor – Ozone*. DGS-O3 968-042. Spec Sensors. Aug. 2017. URL: https://www.spec-sensors.com/wp-content/uploads/2017/01/DGS-O3-968-042_9-6-17.pdf (sjekket 09.05.2021).

[STM14] STMicroelectronics. *Datablad-STEVAL-ISV019V1*. Versjon Rev 3. Nov. 2014. URL: https://www.elfadistrelec.no/Web/Downloads/_t/ds/STEVAL-ISV019V1_eng_tds.pdf (sjekket 23.05.2021).

[STM18] STMicroelectronics. *Datablad, SPV1050*. Versjon Rev 5. Mai 2018. URL: https://www.st.com/resource/en/datasheet/spv1050.pdf (sjekket 23.05.2021).

[Tim20] Timescale. *apt Installation (Debian)*. 2020. URL: https://docs.timescale.com/latest/getting-started/installation/debian/installation-apt-debian (sjekket 01.03.2021).

[WHO06] WHO. *Air quality guidelines. Global update 2005. Particulate matter, ozone, nitrogen dioxide and sulfur dioxide*. 2006. URL: https://www.euro.who.int/en/health-topics/environment-and-health/air-quality/publications/pre2009/air-quality-guidelines.-global-update-2005.-particulate-matter,-ozone,-nitrogen-dioxide-and-sulfur-dioxide (sjekket 13.05.2021).

[WHO13] WHO. *Health risks of air pollution in Europe – HRAPIE project. Recommendations for concentration–response functions for cost–benefit analysis of particulate matter, ozone and nitrogen dioxide*. report. 2013. URL: https://www.euro.who.int/en/health-topics/environment-and-health/air-quality/publications/2013/health-risks-of-air-pollution-in-europe-hrapie-project.-recommendations-for-concentrationresponse-functions-for-costbenefit-analysis-of-particulate-matter,-ozone-and-nitrogen-dioxide (sjekket 13.05.2021).

[Zep21a] Zephyr. *UART*. Zephyr Project. 3. mai 2021. URL: https://docs.zephyrproject.org/latest/reference/peripherals/uart.html#group__uart__interface_1gad04073b1b8e3de13b43ae1194561377 (sjekket 09.05.2021).

[Zep21b] Zephyr. *zephyr/include/i2c.h*. Zephyr Project. Mai 2021. URL: https://github.com/zephyrproject-rtos/zephyr/blob/main/include/drivers/i2c.h (sjekket 24.05.2021).

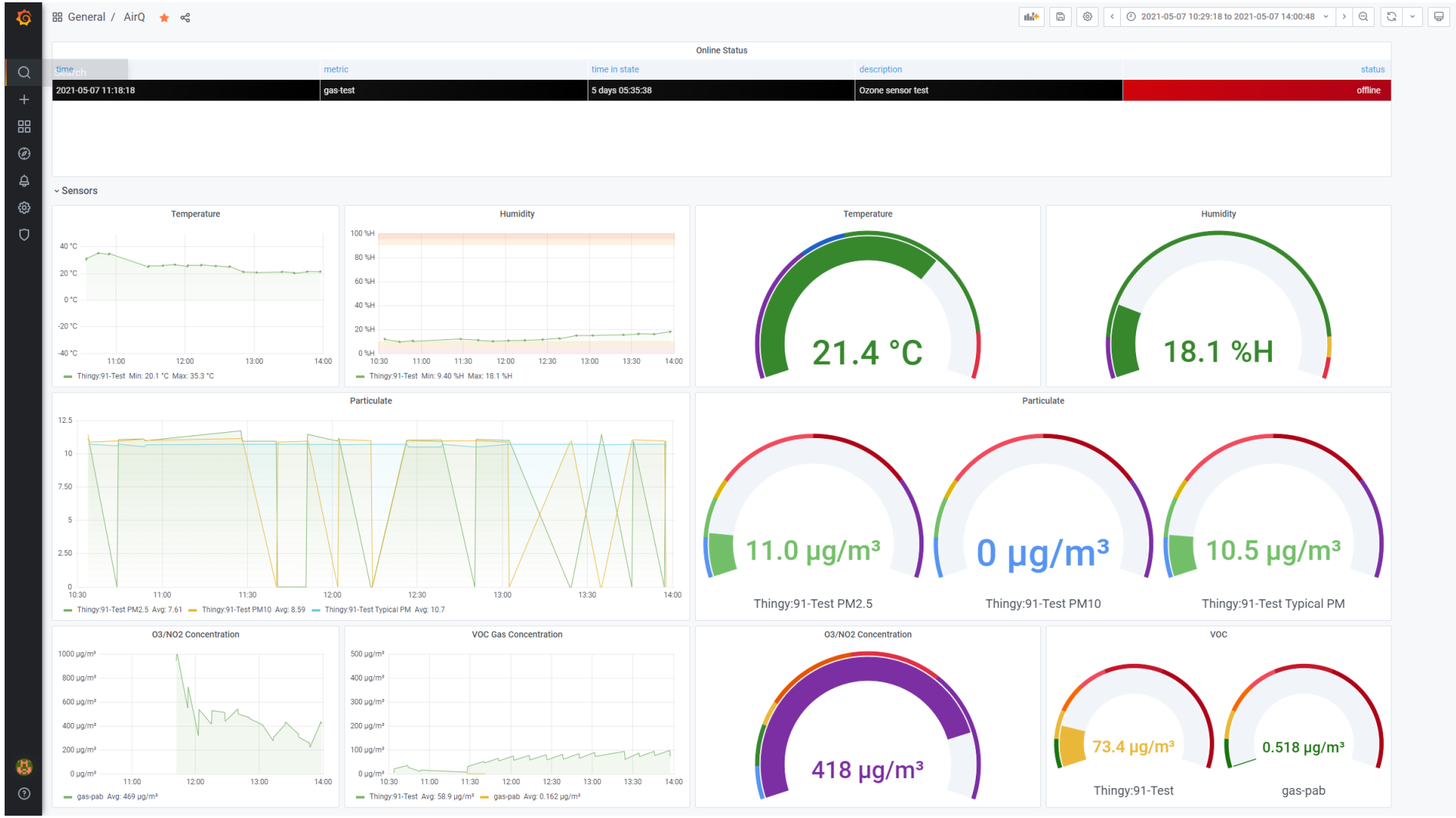
[Zep21c] Zephyr. *zephyr/samples*. Zephyr Project. Mai 2021. URL: https://github.com/zephyrproject-rtos/zephyr/tree/main/samples (sjekket 24.05.2021).

[Aan+21] E. J. K. Aanensen mfl. *Rapport Forprosjekt Bachelor 2021*. NTNU, 1. mar. 2021.

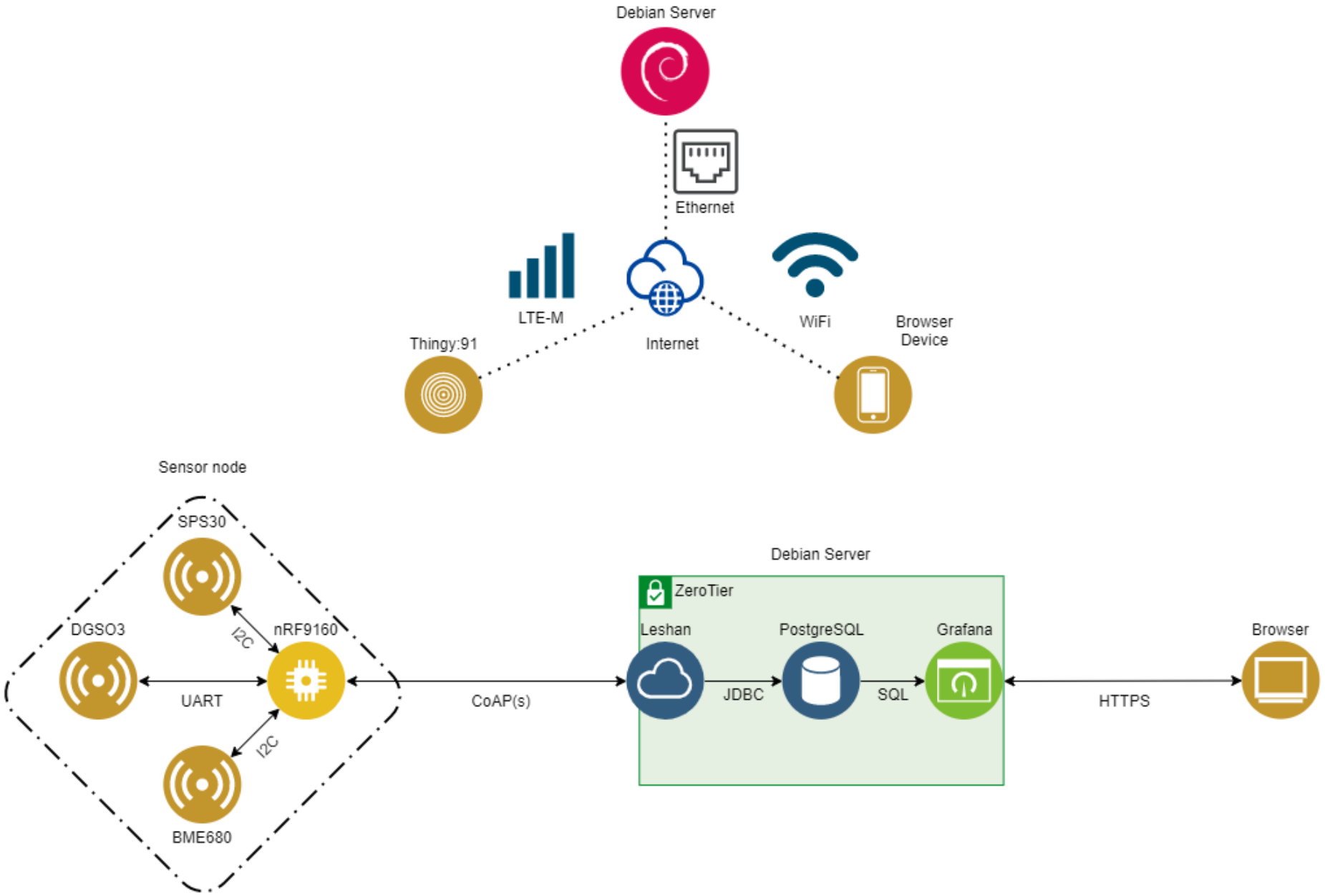
Vedlegg A

Appendiks

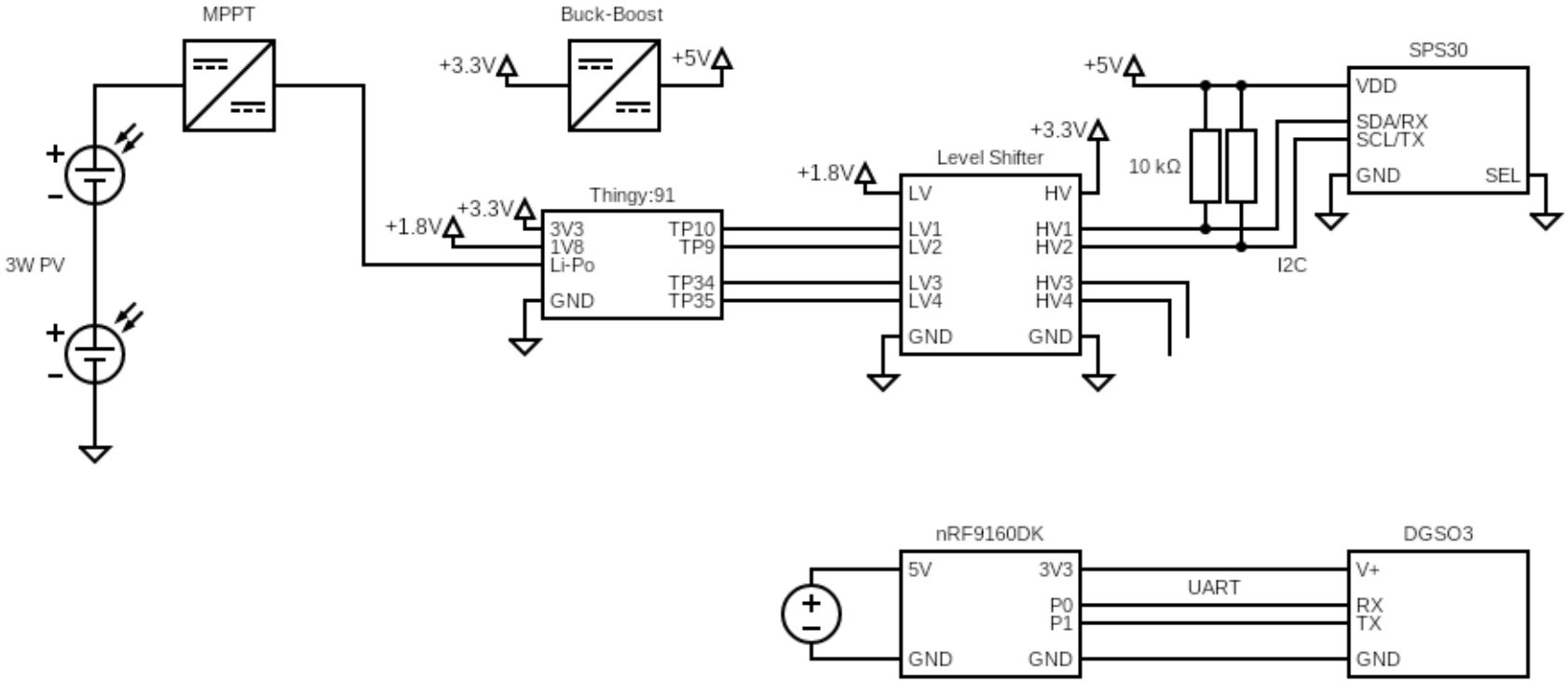
Figur A.1: Grafana Dashbord



Figur A.2: Systemdiagram



Figur A.3: Kretsdiagram



NTNU

Institutt for elektroniske systemer
Institutt for elkraftteknikk
Institutt for teknisk kybernetikk

Oppgaveforslag bacheloroppgave elektroingeniør i Trondheim, vårsemester 2021

Navn bedrift: **Nordic Semiconductor ASA**		**Kontaktperson: Jon Helge Nistad** **Epost: jon.helge.nistad@nordicsemi.no** **Telefon/mobil:99610537**		
Tittel på oppgave: **Sky-tilkoblet værstasjon med energihøsting basert på nRF9160**				
Hvilke studieretninger passer oppgaven for (kryss av for alle aktuelle retninger):	Automatisering x	Elektronikk x	Elkraftteknikk	Instrumentering
Er oppgaven reservert for noen bestemte studenter? I så fall skriv navnene på studentene til høyre.		Patric Andre Berthelsen Sigurd Ranheim Henriksveen Jacob Kristiansen		

Kort beskrivelse av oppgaven med problemstilling.

Konstruere en værstasjon som bruker brikken nRF9160 som hovedenhet i systemet.
Værstasjonen skal ha energihøsting og høste nok energi til å drifte seg selv i hele systemets forventede levetid.
Værstasjonen skal være koblet til mobilnett med LTE-M og/eller NB-IOT og kommunisere måledata til skyen.

Dette er en oppgave som vil kreve ferdigheter innen fagfelt som systemdesign, hardware, firmware og software.

Batteri	Supply voltage [V]	[mAh]	[Wh]		
Kapasitet	3,7	1400	5,18		
Solpanel	**Max spenning [V]**	**Dimensjonering**	**Maks strøm [mA]**	**peak [W]**	**Maks forbruk [Wh/dag]**
	5,5	1323,94 %	510	3,00	0,4
Sensorer	**Voltage [V]**	**Standby current [mA]**	**Active current [mA]**	**Måletid [s]**	**Forbruk [Wh/dag]**
SPS30	5	0,038	55	3	5,88E-03
DGS-O3 968-042	3,3	3,00E-03	4,24	1	1,13E-04
BMP680	1,8	0,00015	0,09	1	1,62E-06
BH1749NUC	3,3	0,002	0,22	1	1,80E-05
Thingy:91 / 9160	Voltage [V]	Standby current [mA]	Active current [mA]	Måletid [s]	Forbruk [Wh/dag]
Thingy:91	5	7,00E-03	1	10	2,42E-02
Totalt		Standby current [mA]	Active current [mA]	Måletid [s]	Forbruk [Wh/dag]
		0,05015	60,55	30	0,03021274

Figur A.4: Strømbudsjett

PROTOTYPE AV SKY-TILKOBLET MÅLESYSTEM FOR EAQI, BASERT PÅ NRF9160 SIP MED ENERGIHØSTINGSSYSTEM

Berthelsen, Patric André; Kristiansen, Jacob;
Henriksveen, Sigurd Ranheim; Aanensen, Eskild Jonatan Krumsvik

Institutt for elektroniske systemer, Norges teknisk-naturvitenskapelige universitet (NTNU)

Introduksjon

I løpet av høsten 2020 kontaktet vi Nordic Semiconductor med tanke om å skrive en oppgave hos dem. Etter et par møter, kom vi fram til et oppgaveforslag gitt av veilederne våre hos Nordic: Endre Rindalsholt, og Jon-Helge Nistad. Her fikk vi en oppgave rundt å bygge en værstasjon basert på mikrobrikken nRF9160. Denne skulle blant annet sende data til en skytjeneste, og den skulle også ha et energihøsting-system som gir enheten nok strøm ut sin levetid. Senere i forprosjektet la vi til innsamling av luftkvalitetsdata. Dette var for å øke kompleksiteten til oppgaven, og bygge en prototype som inneholder potensielt nyttig teknologi.

Teori

nRF9160 fra Nordic Semiconductor er en todelt System-in-Package, med integrert LTE-M/NB-IoT modem, og en ARM Cortex-M33 applikasjonsprosessor som er bygd rundt lav effekt. Den har også utganger for grensesnitt og periferier som: PWM, SPI, UART, og I2C. Den blir brukt i utviklingskort som nRF9160DK, og Thingy:91 [1].

EAQI er et verktøy for bedre forståelse rundt luftkvalitet i Europa. Indeksen ble utviklet av EEA. Den baserer seg på fem faktorer for luftkvalitet, som er: PM_{10}, $PM_{2,5}$, NO_2, O_3, og SO_2. Grunnen en måler luftkvalitet er siden det kan være en helserisiko, når verdiene blir for høye. Spesielt for de med pustevansker eller hjerteproblemer. Figuren under viser hvor høyt de forskjellige verdiene kan gå på en dag, før det skaper problemer. Gjennomsnittsverdier for ett år er strengere siden lengre utsettelse, spesielt for partikler, er mer skadelig. Disse er ikke tatt med.

Pollutant	Good	Fair	Moderate	Poor	Very poor	Extremely poor
$PM_{2,5}$	<10	<20	<25	<50	<75	<800
PM_{10}	<20	<40	<50	<100	<150	<1200
NO_2	<40	<90	<120	<230	<340	<1000
O_3	<50	<100	<130	<240	<380	<800
SO_2	<100	<200	<350	<500	<750	<1250

Fig. 1: Tålelige dags verdier tabell [$\mu g/m^3$] [2].

LwM2M er en måte å hente data fra klienter fra en server, og er utviklet for Internet of Things (IoT). Den baserer seg på standard objekter med forskjellige ressurser som holder verdier. Disse kan endres med beskjeder som blir sendt med CoAP/MQTT/HTTP.

Energihøsting Solcellene gjør om solenergien til likestrøm, som vi kan lagre i et batteri. MPPT går mellom solcellene og batteriet. Det går ut på å høste mest mulig energi ved riktig strøm-spenning forhold. PVGIS er nettsiden vi brukte for kalkulering av hvor mye energi man får ut av et energihøstingsystem. Dette ga oss et anslag på hvor store panel vi trengte, for vårt estimerte strømforbruk [3].

Resultat

En CAD modell er designet for prototypen. Denne modellen lar oss feste solcellepanel, vinkle panelene etter sola, ha tett kammer, ha mere luftig kammer for måling og kabelinnføring fra utsiden til innsiden som ikke leder vann inn. Resultatet fra dette arbeidet er at vi har en 3D-printa modell som har solcellepanel, kapsling for elektronikken med monteringsmuligheter og har et tett lokk.

Sensorene vi valgte å brukte er: Spec Sensors DGSO3, en sensor for måling av NO_2/O_3 gass, Sensirion SPS30, en sensor for måling av $PM_{2,5}, og PM_{10}$ partikler og BME680, en sensor for måling av temperatur, luftfuktighet, trykk, og VOC gass. BME680 var allerede integrert i Thingy:91 [4] [5].

LwM2M serveren Leshan klarte å hente data og lagre det i databasen, med JDBC. PostgreSQL databasen holdt dataen i float verdier, og hadde tidspunkt for alle verdier. Grafana hentet verdiene fra databasen, og viste de som illustrert i figur 2.

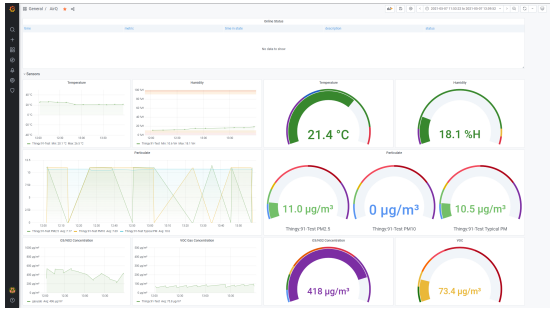
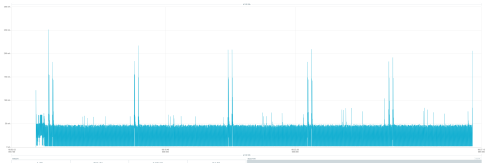


Fig. 2: Grafana Dashbord

Energihøsting systemet vi bruker er to stykk 3W solcellepanel og en MPPT kontroller, SPV1050.

Strømforbruket som vist i figur 3, viser forbruket til den ferdigstilte prototypen over en time. Strømforbruket holdt seg gjennomsnittlig på 6,77 mA. Hver topp viser når systemet våkner, når sensorene starter og deretter når data blir sendt opp til skytjenesten.



Diskusjon og Konklusjon

Fig. 4: Bilde av ferdig prototype

I prosjektet vårt har vi sett på om det går an å bruke Nordic sin Thingy:91 som en målingsstasjon ved å bruke energihøsting og sensorer. Det fullstendige produktet ble en prototype som måler luftkvalitet med unntak måleverdier som O3, og NO2. Den har energihøsting som skulle lade batteriet under hele dens levetid, men dette målet har ikke blitt oppnådd. Ut ifra målingene vi tok kan prototypen opprettholdes i store deler av året, men ikke i mørketiden. Med videre strømoptimalisering vil det vært mulig å opprettholde strømbudsjettet for en hel årstid.

Anerkjennelser

Vi vil gi stor takk til Arne Midjo (NTNU), Endre Risdalsholt og Jon Helge Nistad fra Nordic Semiconductor som har veiledet oss gjennom prosjektet. De har gitt god tilbakemelding og hjulpet oss med å forme oppgaven. Cuong Phu Le har gitt mye nyttig informasjon angående energihøstingen. Vi vil også gi takk til Karina Ødegård fra SINTEF Norlab for veiledning og god info rundt temaet luftforurensning.

Kilder

[1] Nordic. *nRF9160. Low power SiP with integrated LTE-M/NB-IoT modem and GPS*. Nordic Semiconductor. 2021. URL: https://www.nordicsemi.com/Products/Low-power-cellular-IoT/nRF9160 (visited on 05/23/2021).

[2] *About the European Air Quality Index*. European Environment Agency. URL: https://airindex.eea.europa.eu/Map/AQI/Viewer/#current (visited on 05/15/2021).

[3] PVGIS. *Photovoltaic Geographical Information System (PVGIS)*. Mar. 24, 2021. URL: https://ec.europa.eu/jrc/en/pvgis (visited on 04/30/2021).

[4] *Digital Gas Sensor – Ozone*. DGS-O3 968-042. Spec Sensors. Aug. 2017. URL: https://www.spec-sensors.com/wp-content/uploads/2017/01/DGS-O3-968-042_9-6-17.pdf (visited on 05/09/2021).

[5] Sensirion. *Datasheet SPS30 - Particulate Matter Sensor for Air Quality Monitoring and Control*. Sensirion. Mar. 2020. URL: https://www.sensirion.com/fileadmin/user_upload/customers/sensirion/Dokumente/9.6_Particulate_Matter/Datasheets/Sensirion_

Figur A.5: Poster

Kodeliste A.1: Leshan registrering kode

```
lwServer.getRegistrationService().addListener(new RegistrationListener() {
   public void registered(Registration registration, Registration previousReg,
                     Collection<Observation> previousObsersations) {
      System.out.println("new device: " + registration);
      timer = new Timer();
      timer.schedule(new ReadTimerTask(lwServer, registration), 60000);
      airqdb.setState(registration, 2);
      airqdb.log(registration.getEndpoint(), registration.getAddress().toString(),
                                             registration.getId(), "Registered");
      Date date = new Date(System.currentTimeMillis());
      LwM2mResource timestamp  = LwM2mSingleResource.newDateResource(13, date);
      LwM2mResource offset   = LwM2mSingleResource.newStringResource(14, "+00:00");
      LwM2mResource timezone = LwM2mSingleResource.newStringResource(15, "UTC");
      try {
         lwServer.send(registration,
                  new WriteRequest(Mode.UPDATE, 3, 0, timestamp, offset, timezone));
      } catch (InterruptedException e) {
         e.printStackTrace();
         airqdb.logError(registration.getEndpoint(),
                         registration.getAddress().toString(),
                         registration.getId(), e.toString());
      }
   }

   public void updated(RegistrationUpdate update,
                  Registration updatedReg, Registration previousReg) {
      System.out.println("device is still here: " + updatedReg.getEndpoint());
      timer.schedule(new ReadTimerTask(lwServer, updatedReg), 60000);
   }

   public void unregistered(Registration registration,
                            Collection<Observation> observations,
                            boolean expired, Registration newReg) {
      System.out.println("device left: "+ registration.getEndpoint()+ observations);
      airqdb.setState(registration, 0);
      timer.cancel();
      airqdb.log(registration.getEndpoint(),
                 registration.getAddress().toString(),
                 registration.getId(), "Deregistered");
   }
});
```

Kodeliste A.2: Leshan JDBC kode

```
package aqi;

import java.sql.*;
import java.time.*;
import java.util.*;
import java.nio.ByteBuffer;

import org.eclipse.leshan.core.observation.*;
import org.eclipse.leshan.core.node.*;
import org.eclipse.leshan.core.response.*;
import org.eclipse.leshan.core.request.*;
import org.eclipse.leshan.server.registration.*;
import org.eclipse.leshan.core.model.ResourceModel.Type;
import org.eclipse.leshan.server.californium.LeshanServer;
```

```
public class Database {
    private static String user;
    private static String pass;

    public Database(String user, String pass) {
        this.user = user;
        this.pass = pass;
    }

    // JDBC driver name and database URL
    static final String JDBC_DRIVER = "org.postgresql.Driver";
    static final String DB_URL = "jdbc:postgresql://localhost:5432/airqdb";

    private static Object getValue(ObserveResponse response) {
        Type type = ((LwM2mSingleResource)response.getContent()).getType();
        Object value = ((LwM2mSingleResource)response.getContent()).getValue();
        System.out.println("" + type + value.getClass());
        switch (type) {
            case OPAQUE:
                byte[] bytes = (byte []) value;
                if (bytes.length == 4)
                    return ByteBuffer.wrap(bytes).getFloat();
                else if (bytes.length == 8)
                    return ByteBuffer.wrap(bytes).getDouble();
                else if (bytes.length == 2 || bytes.length == 1)
                    return ByteBuffer.wrap(bytes).getInt();
                else
                    return null;
            default:
                break;
        }
        return value;
    }

    public static void setState(Registration reg, int status) {
        Statement stmt = null;
        Connection conn = null;
        String endpoint = reg.getEndpoint();
        System.out.println("Connecting to database...");

        String sql = "INSERT INTO devices(endpoint, status) " +
                     "VALUES('" + endpoint + "', " + status + ") " +
                     "ON CONFLICT (endpoint)" +
                     "DO UPDATE SET " +
                     "status = EXCLUDED.status," +
                     "time = now();";
        System.out.println(sql);

        try
        {
            Class.forName("org.postgresql.Driver");
            conn = DriverManager.getConnection(DB_URL, user, pass);
            System.out.println("INSERT data...");
            stmt = conn.createStatement();
            stmt.executeUpdate(sql);
            System.out.println("INSERT data successfully! ");
            stmt.close();
        }
```

```
        catch(SQLException se)
        {
            se.printStackTrace(); //Handle errors for JDBC
        }
        catch(Exception e)
        {
            e.printStackTrace(); //Handle errors for Class.forName
        }
    }

    private static String getTimestamp(ReadResponse response) {

        System.out.println("getTimestamp() "+response.getContent().toString());

        return (String) ((
            (LwM2mSingleResource)response.getContent()).getValue());
    }

    public static void observation(ReadResponse response, String endpoint,
                                LeshanServer server, Registration registration,
                                int obj, int inst, int res)
    {
        //Value
        LwM2mSingleResource resource = (
            (LwM2mSingleResource) response.getContent());

        String timestamp = "0";

        try {
            ReadResponse timeresponse = server.send(registration,
             new ReadRequest(ContentFormat.TEXT, obj,inst,5518));
            timestamp = getTimestamp(timeresponse);
        } catch (Exception e) {
            e.printStackTrace();
            System.out.println("Error read timestamp" + obj + inst);
        }

        double val = (double) resource.getValue();

        Statement stmt = null;
        Connection conn = null;
        System.out.println("Connecting to database...");
        String sql = "INSERT INTO observation(timestamp, endpoint, object,"
                   + "instance, resource, value) "
                   + "VALUES(" + timestamp + ", '" + endpoint + "', "
                   + obj + ", " + inst + ", " + res + ", " + val + ");";
        System.out.println(sql);
        try
        {
            Class.forName("org.postgresql.Driver");
            conn = DriverManager.getConnection(DB_URL, user, pass);
            System.out.println("INSERT data...");
            stmt = conn.createStatement();
            stmt.executeUpdate(sql);
            System.out.println("INSERT data successfully! " + timestamp);
            stmt.close();
        }
        catch(SQLException se)
        {
            se.printStackTrace(); //Handle errors for JDBC
```

```
        }
        catch(Exception e)
        {
            e.printStackTrace(); //Handle errors for Class.forName
        }
    }

    public static void log(String endpoint, String ip,
                           String regId, String message)
    {
        Statement stmt = null;
        Connection conn = null;
        System.out.println("Connecting to database...");
        String sql = "INSERT INTO log(endpoint, ip, regid, message) "
                   + "VALUES('" + endpoint + "', '" + ip + "', '" + regId
                   + "', '" + message + "');";
        System.out.println(sql);
        try
        {
            Class.forName("org.postgresql.Driver");
            conn = DriverManager.getConnection(DB_URL, user, pass);
            System.out.println("INSERT log...");
            stmt = conn.createStatement();
            stmt.executeUpdate(sql);
            System.out.println("Logged successfully!");
            stmt.close();
        }
        catch(SQLException se)
        {
            se.printStackTrace(); //Handle errors for JDBC
        }
        catch(Exception e)
        {
            e.printStackTrace(); //Handle errors for Class.forName
        }
    }

    public static void logError(String endpoint, String ip,
                                String regId, String error)
    {
        Statement stmt = null;
        Connection conn = null;
        System.out.println("Connecting to database...");
        String sql = "INSERT INTO log(endpoint, ip, regid, error) "
                   + "VALUES('" + endpoint + "', '" + ip + "', '" + regId
                   + "', '" + error + "');";
        try
        {
            Class.forName("org.postgresql.Driver");
            conn = DriverManager.getConnection(DB_URL, user, pass);
            System.out.println("INSERT error into log...");
            stmt = conn.createStatement();
            stmt.executeUpdate(sql);
            System.out.println("Logged error successfully!");
            stmt.close();
        }
        catch(SQLException se)
        {
            se.printStackTrace(); //Handle errors for JDBC
        }
```

```
        catch(Exception e)
        {
            e.printStackTrace(); //Handle errors for Class.forName
        }
    }
}
```

Kodeliste A.3: Leshan sensor leser kode

```
package aqi;

import java.util.*;
import java.util.concurrent.TimeUnit;
import java.time.*;
import java.nio.ByteBuffer;

import org.eclipse.leshan.core.util.Hex;
import org.eclipse.leshan.core.util.StringUtils;
import org.eclipse.leshan.server.californium.LeshanServer;
import org.eclipse.leshan.core.observation.*;
import org.eclipse.leshan.core.request.*;
import org.eclipse.leshan.core.response.*;
import org.eclipse.leshan.core.node.*;
import org.eclipse.leshan.server.registration.*;
import org.eclipse.leshan.core.model.ResourceModel.Type;
import org.eclipse.leshan.core.node.codec.CodecException;

import aqi.Database;

public class ReadTimerTask extends TimerTask {
    LeshanServer server;
    Registration registration;

    public static Database airqdb = new Database("airq", "airqpass");

    public ReadTimerTask(LeshanServer srv, Registration reg) {
        server = srv;
        registration = reg;
    }

    protected Object getValue(ReadResponse response) {
        Type type = ((LwM2mSingleResource) response.getContent()).getType();
        Object value = ((LwM2mSingleResource) response.getContent()).getValue();
        System.out.println("" + type + value.getClass());
        switch (type) {
            case OPAQUE:
                byte[] bytes = (byte []) value;
                if (bytes.length == 4)
                    return ByteBuffer.wrap(bytes).getFloat();
                else if (bytes.length == 8)
                    return ByteBuffer.wrap(bytes).getDouble();
                else
                    return null;
            default:
                break;
        }
        return value;
    }

    public void sendReq(int id1, int id2, int id3) {
```

```
        try {
            ReadResponse response = server.send(registration,
                                    new ReadRequest(id1, id2, id3));
            if (response.isSuccess()) {
                System.out.println("Value:"
                    + response.getContent().toString());
                airqdb.observation(response, registration.getEndpoint(), server,
                                   registration, id1, id2, id3);
            } else {
                System.out.println("Failed to read:" + response.getCode() + " "
                                    + response.getErrorMessage());
            }
        } catch (InterruptedException e) {
            e.printStackTrace();
            airqdb.logError(registration.getEndpoint(),
                            registration.getAddress().toString(),
                            registration.getId(), e.toString());
        }
    }

    @Override
    public void run() {
        try
        {
            sendReq(3300, 0, 5700); // Gas sensor
            Thread.sleep(500);
            sendReq(3300, 1, 5700); // Concentration sensor
            Thread.sleep(500);
            sendReq(3300, 2, 5700); // Particulate sensor 2,5PM
            Thread.sleep(500);
            sendReq(3300, 3, 5700); // Particulate sensor 10PM
            Thread.sleep(500);
            sendReq(3300, 4, 5700); // Particulate sensor Typisk PM
            Thread.sleep(500);
            sendReq(3303, 0, 5700); // Temperature sensor
            Thread.sleep(500);
            sendReq(3304, 0, 5700); // Humidity sensor
        }
        catch(InterruptedException ex)
        {
            Thread.currentThread().interrupt();
        }
        System.out.println("Read all sensor data");
    }
}
```

Kodeliste A.4: Statistisk test - Python

```
# %%
# Libraries
import math, sys
import numpy as np
import pandas as pd
import scipy.stats as st
from matplotlib import pyplot as plt
plt.style.use(['science','no-latex', 'ieee', 'light'])

# Class function for validating sensor data
# (For CSV format reference, see "test.csv")
#
```

```
# Input: file1 - csv file with sensor data
#        file2 - csv file with sensor data
# Output: r - Pearsons Correlation Coefficient: Tells us how correlated the data is
#         p - p value: Used to test the null hypothesis,
#                      which says that the data is not correlated
#         valid - Bool flag: True when p value < 0.05,
#                            which rejects null hypothesis (data is correlated)
#                            False when p value > 0.05,
#                            which accepts null hypothesis (data not correlected)
class Valid:
    def __init__(self, val1, val2):

        # Check if lenght of arrays match
        if (len(val1) != len(val2)):
            sys.exit("Length of arrays must be the same")

        # Set class 'r', 't' and 'p' values, as well as validity tests
        # print(st.pearsonr(val, val))
        length = len(val1)
        self.r = st.pearsonr(val1, val2)[0]
        # First test - Check pearson correlation rule of thumb
        self.valid1 = abs(self.r) >= 2/math.sqrt(length)
        # Get T value from student T-distrubution of pearson correlation
        self.t = r*math.sqrt((length - 2)/(1 - r**2))
        # Lookup p value based on t value
        self.p = st.t.sf(abs(self.t), df=2)

        # Check if p value is less than alpha level => rejects null hypothesis
        if (self.p < 0.05):
            self.valid2 = True
        # Check if p value is greater than alpha level => accept null hypothesis
        else:
            self.valid2 = False

# %%
# Prep Grafana data

# Read data and prepare arrays
data = pd.read_csv("ozon.csv", header=0, parse_dates=True)
t = data.iloc[:, 0].to_numpy()
t = np.flip(t)
pm2_5 = np.zeros(len(t))
pm10 = np.zeros(len(t))
pm_typ = np.zeros(len(t))
voc = np.zeros(len(t))
o3 = np.zeros(len(t))
r = data.iloc[:, 4].to_numpy()

# Extract elemnts by sensor type
x = data.iloc[:, 6].to_numpy()
voc = x[r==0]
o3 = x[r==1]
pm2_5 = x[r==2]
pm10 = x[r==3]
pm_typ = x[r==4]

# Flip arrays
o3 = np.flip(o3)
pm2_5 = np.flip(pm2_5)
pm10 = np.flip(pm10)
```

```
# Remove null values
t1 = t[r==1]
i = 0
for x in o3:
    if ((x == 0) or ((t1[i] == t1[i-1]) and i != 0)):
        o3 = np.delete(o3, i)
        t1 = np.delete(t1, i)
    else:
        i = i + 1

t2 = t[r==2]
i = 0
for x in pm2_5:
    if ((x == 0) or ((t2[i] == t2[i-1]) and i != 0)):
        pm2_5 = np.delete(pm2_5, i)
        t2 = np.delete(t2, i)
    else:
        i = i + 1

t3 = t[r==3]
i = 0
for x in pm10:
    if ((x == 0) or ((t3[i] == t3[i-1]) and i != 0)):
        pm10 = np.delete(pm10, i)
        t3 = np.delete(t3, i)
    else:
        i = i + 1

# Extract times that match with NILU data
pm2_5 = np.take(pm2_5, [0, 7, 17, 22, 28, 34, 39, 46])
pm10 = np.take(pm10, [0, 8, 16, 23, 31, 41, 48, 54])
t2 = np.take(t2, [0, 7, 17, 22, 28, 34, 39, 46])
t3 = np.take(t3, [0, 8, 16, 23, 31, 41, 48, 54])

# %%
# Prep NILU
data = pd.read_csv("nilu.csv", header=3, parse_dates=True)
nt = data.iloc[:, 0].to_numpy()

# Extract data from 14:00 to 21:00
nt = nt[0:8]
t2 = t2[0:8]
t3 = t3[0:8]

npm2_5 = data.iloc[:, 2]
npm2_5 = npm2_5[0:8]

npm10 = data.iloc[:, 5]
npm10 = npm10[0:8]

# %%
r = 0
t = Valid(pm2_5, npm2_5)
print("Test - PM2.5")
print("r value: ", t.r)
print("t value: ", t.t)
print("p value: ", t.p)
print("Test1 (Pearson RoT): ", t.valid1)
print("Test2 (T-test): ", t.valid2)
```

```
print("\n")

t = Valid(pm10, npm10)
print("Test - PM10")
print("r value: ", t.r)
print("t value: ", t.t)
print("p value: ", t.p)
print("Test1 (Pearson RoT): ", t.valid1)
print("Test2 (T-test): ", t.valid2)
print("\n")

# %%
plt.subplot(2,1,1)
plt.title("PM2.5", fontsize=6)
plt.gcf().axes[0].yaxis.get_major_formatter().set_scientific(False)
plt.plot(t2, pm2_5, '#ffaabb')
plt.xticks(fontsize=4)
plt.yticks(fontsize=4)
plt.ylabel("ug/m3", fontsize=5)
plt.legend(["Grafana"], fontsize=5)

plt.subplot(2,1,2)
plt.gcf().axes[1].yaxis.get_major_formatter().set_scientific(False)
plt.plot(nt, npm2_5, '#77aadd')
plt.xticks(fontsize=4)
plt.yticks(fontsize=4)
plt.ylabel("ug/m3", fontsize=5)
plt.xlabel("UTC+02 (CEST)", fontsize=5)
plt.legend(["NILU"], fontsize=5)

plt.savefig("test_pm2_5.png")
plt.show()

plt.subplot(2,1,1)
plt.title("PM10.0", fontsize=6)
plt.gcf().axes[0].yaxis.get_major_formatter().set_scientific(False)
plt.plot(t3, pm10, '#bbcc33')
plt.xticks(fontsize=4)
plt.yticks(fontsize=4)
plt.ylabel("ug/m3", fontsize=5)
plt.legend(["Grafana"], fontsize=5)

plt.subplot(2,1,2)
plt.gcf().axes[1].yaxis.get_major_formatter().set_scientific(False)
plt.plot(nt, npm10, '#44bb99')
plt.xticks(fontsize=4)
plt.yticks(fontsize=4)
plt.ylabel("ug/m3", fontsize=5)
plt.xlabel("UTC+02 (CEST)", fontsize=5)
plt.legend(["NILU"], fontsize=5)

plt.savefig("test_pm10.png")
plt.show()
```

Tabell A.1: Sensors Database View

Tid	Klient	Sensor	Verdi	Enhet
2021-05-07 12:58:30+01	gas-pab	NO2/O3 Gas	220.00	ppb
2021-05-07 12:58:00+01	gas-pab	NO2/O3 Gas	230.00	ppb
2021-05-07 12:57:49+01	Thingy:91-Test	Humidity	18.12	%RH
2021-05-07 12:57:49+01	Thingy:91-Test	Temperature	21.39	°C
2021-05-07 12:57:48+01	Thingy:91-Test	PM Typ	10.49	µg/m³
2021-05-07 12:57:43+01	Thingy:91-Test	PM10	0.00	µg/m³
2021-05-07 12:57:38+01	Thingy:91-Test	PM2.5	10.97	µg/m³
2021-05-07 12:57:32+01	Thingy:91-Test	VOC Gas	73.40	µg/m³
2021-05-07 12:57:19+01	Thingy:91-Test	Humidity	18.03	%RH
2021-05-07 12:57:19+01	Thingy:91-Test	Temperature	21.31	°C
2021-05-07 12:57:18+01	Thingy:91-Test	PM Typ	10.70	µg/m³
2021-05-07 12:57:13+01	Thingy:91-Test	PM10	10.95	µg/m³
2021-05-07 12:57:08+01	Thingy:91-Test	PM2.5	0.00	µg/m³
2021-05-07 12:57:02+01	Thingy:91-Test	VOC Gas	98.14	µg/m³
2021-05-07 12:47:43+01	gas-pab	NO2/O3 Gas	117.00	ppb
2021-05-07 12:47:13+01	gas-pab	NO2/O3 Gas	135.00	ppb
2021-05-07 12:46:22+01	Thingy:91-Test	Humidity	16.07	%RH
2021-05-07 12:46:21+01	Thingy:91-Test	Temperature	21.45	°C
2021-05-07 12:46:21+01	Thingy:91-Test	PM Typ	10.70	µg/m³
2021-05-07 12:46:16+01	Thingy:91-Test	PM10	11.04	µg/m³
2021-05-07 12:46:10+01	Thingy:91-Test	PM2.5	10.95	µg/m³
2021-05-07 12:46:05+01	Thingy:91-Test	VOC Gas	76.47	µg/m³
2021-05-07 12:45:52+01	Thingy:91-Test	Humidity	16.05	%RH
2021-05-07 12:45:51+01	Thingy:91-Test	Temperature	21.20	°C
2021-05-07 12:45:51+01	Thingy:91-Test	PM Typ	10.70	µg/m³
2021-05-07 12:45:46+01	Thingy:91-Test	PM10	11.00	µg/m³
2021-05-07 12:45:40+01	Thingy:91-Test	PM2.5	0.00	µg/m³
2021-05-07 12:45:35+01	Thingy:91-Test	VOC Gas	94.24	µg/m³
2021-05-07 12:37:06+01	gas-pab	NO2/O3 Gas	161.00	ppb
2021-05-07 12:36:37+01	gas-pab	NO2/O3 Gas	175.00	ppb
2021-05-07 12:35:10+01	Thingy:91-Test	Humidity	16.38	%RH
2021-05-07 12:35:10+01	Thingy:91-Test	Temperature	20.14	°C
2021-05-07 12:35:09+01	Thingy:91-Test	PM Typ	10.65	µg/m³
2021-05-07 12:35:04+01	Thingy:91-Test	PM10	0.00	µg/m³
2021-05-07 12:34:59+01	Thingy:91-Test	PM2.5	11.44	µg/m³
2021-05-07 12:34:54+01	Thingy:91-Test	VOC Gas	72.14	µg/m³
2021-05-07 12:34:41+01	Thingy:91-Test	Humidity	16.11	%RH
2021-05-07 12:34:40+01	Thingy:91-Test	Temperature	20.57	°C
2021-05-07 12:34:40+01	Thingy:91-Test	PM Typ	10.70	µg/m³
2021-05-07 12:34:35+01	Thingy:91-Test	PM10	0.00	µg/m³
2021-05-07 12:34:29+01	Thingy:91-Test	PM2.5	10.87	µg/m³
2021-05-07 12:34:24+01	Thingy:91-Test	VOC Gas	86.91	µg/m³
2021-05-07 12:24:34+01	gas-pab	NO2/O3 Gas	228.00	ppb
2021-05-07 12:24:28+01	Thingy:91-Test	Humidity	15.39	%RH

NTNU

Kunnskap for en bedre verden