Rev 1.0

*Nils Kr. Rossing og Thor Inge Hansen*

# Elevhefte:
# Miljøovervåking med bøye
## Arduino MKR NB 1500


NTNU
Trondheim
Institutt for fysikk
Skolelaboratoriet
for matematikk, naturfag
og teknologi

Juli 2023

# Elevhefte:
# Miljøovervåking med bøye, Arduino MKR NB 1500

Nils Kr. Rossing,
Skolelaboratoriet NTNU, Institutt for fysikk
Thor Inge Hansen
Skien videregående skole
i samarbeid med Institutt for marin teknikk og
Institutt for lærerutdanning

**Elevhefte: Miljøovervåking med bøye, Arduino MKR NB 1500**

Trondheim 2023

| | |
|---|---|
| Layout og redigering: | Nils Kr. Rossing, Skolelaboratoriet ved NTNU |
| Trykk: | NTNU Grafisk senter |
| Tekst og bilder: | Nils Kr. Rossing, Skolelaboratoriet, NTNU,<br>Jussi Evertsen, Vitenskapsmuseet, NTNU |
| Forsidebilde: | Internett: Arduino MKR NB 1500 |
| Programmering: | Nils Kr. Rossing, NTNU<br>Thor Inge Hansen, Skien videregående skole |
| Kursholdere:<br>(Elektronikk/Progr.) | Nils Kr. Rossing<br>Thor Inge Hansen, Skien videregående skole<br>Johannes Ravn Munkvold (læringsassistent)<br>Skolelaboratoriet, Institutt for fysikk, NTNU |

Faglige spørsmål rettes til:
**Skolelaboratoriet for matematikk, naturfag og teknologi**
**Institutt for fysikk**
v/ Nils Kr. Rossing, nils.rossing@ntnu.no

Skolelaboratoriet ved NTNU
Realfagbygget,
Høgskoleringen 5,
7491 Trondheim

Telefon: 73 55 11 91
https://www.ntnu.no/skolelab/

Rev 1.0 – 30.07.23

# Forord

Prosjektet “Miljøovervåking med bøye” handler om å samle inn måledata fra kystnære strøk, i ferskvann eller elver i innlandet. Teknologien kan imidlertid brukes i mange ulike sammenhenger, i områder der man normalt ikke har tilgang til lokale nettverk slik at man kan overføre data via Wi-Fi. For å få til dette tar man i bruk ulike typer smalbånds mobildata som Telenor med flere tilbyr innen rammen av 4 og 5G.

Heftet gir en detaljert beskrivelse av oppbyggingen av elektronikk og et program for innsamling av data og sending av måleresultater til server.

Først bygger vi opp og programmerer for mikrokontrolleren Arduino MKR NB 1500 og tester denne ved å måle temperatur og GPS koordinater, for så å overføre disse til en server.

Dernest bygger vi opp en styringskrets som holder orden på måletidspunktene, slår av kretsen mellom målingene for å spare energi og gjør det enkelt å lade batteriet når bøya er i laboratoriet.

Skolelaboratoriet ved NTNU,
Juli 2023
Nils Kr. Rossing
Thor Inge Hansen

# Innhold

# 1 Innledning

Heftet er en veiledning av oppbygging og programmering av en sensornode med tanke på trådløs innhenting av data fra en bøye. Framgangsmåten er organisert i 10 deloppdrag som fører fram til en ferdig virkende node. Før man begynner på prosjektet bør man ha grunnleggende kunnskap om programmering av Arduino.

Siden sensornoden bruke NB IoT så har den et stort dekningsområde som er anslått til å dekke ca. 99,9% av befolkningen. Vi har valgt å bruke Arduino MKR NB 1500 og basere oss på SIM-kort fra Telenor. Inntil videre legges dataene ned på en server ved Institutt for marin teknikk og visualisering av dataene kan gjøres ved hjelp av Python skript eller Excel og Google Earth.

Vi har så langt ønsket å gjøre det så enkelt som mulig slik at man kan komme fort igang. Vi har derfor inntil videre valgt å legge dataene i CSV-filer på serveren som er lett å behandle med de nevnte programverktøyene. Det finnes imidlertid flere alternative skytjenester for lagring og visualisering av data, men ingen tilfredsstiller foreløpig de krav som kommuner og fylkeskommuner stiller mht. sikkerhet. Firmaet Company of Things arbeider imidlertid med et lovende konsept skreddersydd for bruk i skolen. De ser for seg at det så smått kan tas i bruk i løpet av 2024.

Dette heftet gir en oppskrift for hvordan komme igang, men på sikt er det dere elever som selv skal bygge opp bøya med elektronikk og utstyre den med relevante sensorer.

# 2 Prosjektet – Oppgavesamlingen

Vi skal først gi dere en oversikt over oppdraget dere skal løse gjennom prosjektet:

> **Oppdraget:** *Det skal lages et instrument for måling av miljødata, inntil videre bare vanntemperatur. Instrumentet skal utstyres med en GPS-mottaker slik at målingene kan tid- og stedfestes. Dataene skal legges på et minnekort (SD-kort) i tillegg til at de skal overføres til en server hvor måleserien skal legges i en CSV-fil. De lagrede dataene skal presenteres som tabeller og grafer, gjerne med bruk av Python-skript, Excel, Google Earth, eller annen passende programvare. Om ønskelig kan man også bygge opp et styringskort for styring av måletidspunkt og som slår av strømmen mellom målingene for å spare energi.*

**Oversikt over deloppdrag**

I løpet av 10 deloppdrag skal vi bygge opp og overføre data fra sensornoden vår til serveren og presentere dataene. Denne gangen skal vi konsentrere oss om måling av vanntemperatur og posisjon. Vi har delt opp prosjektet i følgende 10 deloppdrag:

> **Deloppdrag 1:** *Installer programeditor for Arduino og tilleggsmoduler for programmering av mikrokontrollerkortet MKR NB 1500.*
>
> **Deloppdrag 2:** *Koble opp hårdvaren, monter antennen og installer SIM-kortet.*
>
> **Deloppdrag 3:** *Installer nødvendige biblioteker og last opp og kompiler eksempelprogrammet. Gå gjennom koden og få oversikt over programmet.*

**Deloppdrag 4:** *Les av sensoren for måling av vanntemperatur og skriv resultatet ut til monitoren. Stikk sensoren ned i et glass med vann og undersøk om temperaturen er som forventet. Avlesningen av temperaturen legges inn i datastrengen i hovedprogrammet.*

**Deloppdrag 5:** *Monter MKR GPS-kortet og installer det tilhørende biblioteket. Bruk eksempelprogrammet og inkluder funksjonen for lesing av GPS-mottakeren. Kontroller at avleste data er som forventet.*

**Deloppdrag 6:** *Koble opp sensornoden til NB IoT-nettverket og overfør data til serveren. Gå inn på serveren og kontroller at lagrede data er som forventet.*

**Deloppdrag 7:** *Inkluder en #vakthund" som resetter programmet dersom det skulle stoppe opp.*

**Deloppdrag 8:** *Monter en SD-kort terminal, installer nødvendig bibliotek, opprett en fil og skriv måledataene til filen.*

**Deloppdrag 9:** *Presenter måleresultatene med passende programvare, f.eks. Excel eller et Python-script evt. Google Earth. Kommenter resultatene.*

**Deloppdrag 10:** *Lodd opp PLSC MKR-kortet og sjekk at det fungerer som tiltenkt.*

## 2.1 Deloppdrag 1: Installasjon av programvare

I dette deloppdraget skal vi installere programvaren som vi trenger for å bygge opp sensornoden vår.

**Deloppdrag 1:** *Installer programeditor for Arduino og tilleggsmoduler for programmering av mikrokontrollerkortet MKR NB 1500.*

### 2.1.1 *Installasjon av Arduino IDE*

Om du har installert Arduino IDE tidligere kan du hoppe over dette avsnittet og gå direkte til avsnitt 2.1.4 side 16.

**Nedlasting av programvare**

Arduino programeditor og kompilator hentes fra:

http://arduino.cc/hu/Main/Software

Versjonen som er brukt i denne sammenheng er 1.8.19. Filen som har navnet *arduino-1.8.19-windows,* er pakket som en zip-fil. En tilsvarende fil er tilgjengelig for Mac fra samme nettsted.

Det er også helt greit å bruke "Windows installer" versjonen som finnes på samme side. Det er imidlertid erfart noen uregelmessigheter ved bruk av Windows app-versjonen, så unngå gjerne den.

Det er også greit å installere den nye utgaven, versjon 2.1.0. Denne har en del fordeler, ved at den bl.a. gir økte muligheter til etterspore funksjonen til programmet (debuging) og for feilfinning.

**Installasjon av Arduino editoren:**

1. Klikk på *Windows installer* og følg anvisningen under installasjonen. For MAC-brukere klikk på *MAC OS X*

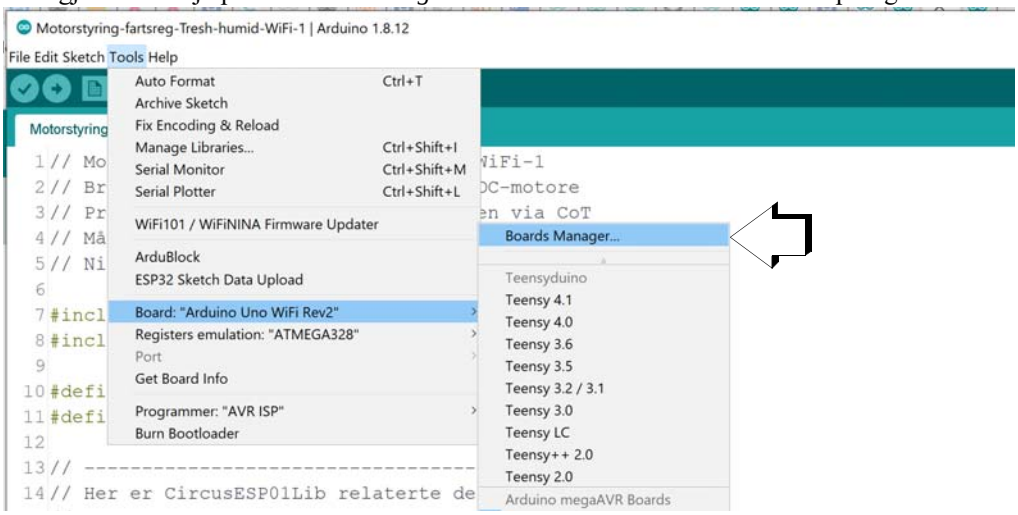
2. Programmet startes ved å klikke på programikonet:  .

Før vi går videre skal vi installere en egen programpakke for MKR NB 1500 mikrokontrollerkortet.

### 2.1.2 Installasjon av tilleggsmodul for håndtering av MKR NB 1500

Vi skal nå installere programpakken for å kunne programmere MKR NB 1500.

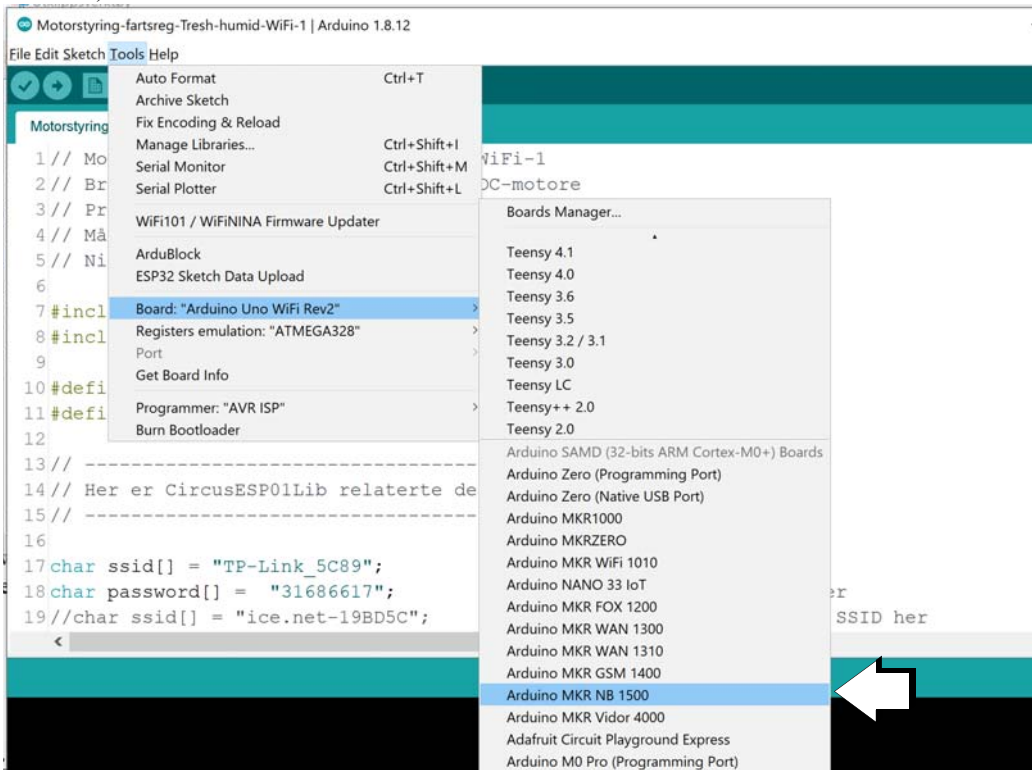
Dette gjør vi ved hjelp av *Boards Manager* som vi finner under *Tools* som vist på figuren under.




Velger vi *Board Manager* så kommer det opp et vindu med et søkefelt der vi kan skrive navnet på den familien av kort vi ønsker å installere: *Arduino SAMD Boards (32-bits ARM Cortex M0+)*. Etter noe tid kommer følgende alternativ opp:


Vi velger det kortet (se figuren over) som kommer opp og trykker INSTALL nederst til høyre. Programvaren for denne kort-familien består av 6 pakker og det tar litt tid å laste ned og installere dem.

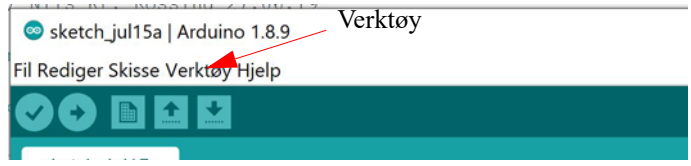
Når vi skal programmere kortet MKR NB 1500, må vi huske å velge dette kortet fra menyen av mulige kort, se figuren under. Vi velger Tools –> Boards –> Arduino SAMD(32-bits ARM Cortex-M0+) Boards –> Arduino MKR NB 1500.




La oss se litt på grunnleggende bruk av Arduino programeditoren (IDE).

### 2.1.3 Grunnleggende bruk av Arduino programeditoren

1. Koble USB-kabelen til ønsket USB-port på PC-en.
2. Klikk på *Verktøy (Tools)* på menylinjen og velg *Kort*. Her velges hvilken variant i Arduino-familien du ønsker å jobbe med.




   Etter at vi har installert tilleggsprogramvare for MKR NB 1500 (se avsnitt 2.1.4 på side 16), velges Arduino MKR NB 1500.
3. Klikk på *Verktøy* på menylinjen og velg *Port*. Sjekk at riktig port (Com?) er valgt. Normalt står det hvilken Arduino-variant porten er koblet til eller velg det høyeste Com-nummeret.

Programmet er nå klart til bruk og du kan skrive inn programlinjene. Når programmet er ferdig skrevet, skal det *kompileres*, dvs. overføres til en binærkode som mikrokontrolleren forstår. Dersom programmet inneholder ulovlige kommandoer eller skrivefeil, vil kompilatoren varsle om det og vise på hvilken linje feilen er oppdaget. Det er ikke nødvendigvis alltid der feilen befinner seg.

Dernest skal programmet lastes ned til mikrokontrollerens minne (Arduino-kortet). Dette gjøres ved å trykke på knappen  .

**Kort oversikt over Arduino-editoren**

Man finner følgende kommandoikoner på den grafiske menylinja:

 Kompiler og verifiser at koden er riktig – overfører ikke koden til mikrokontrolleren.

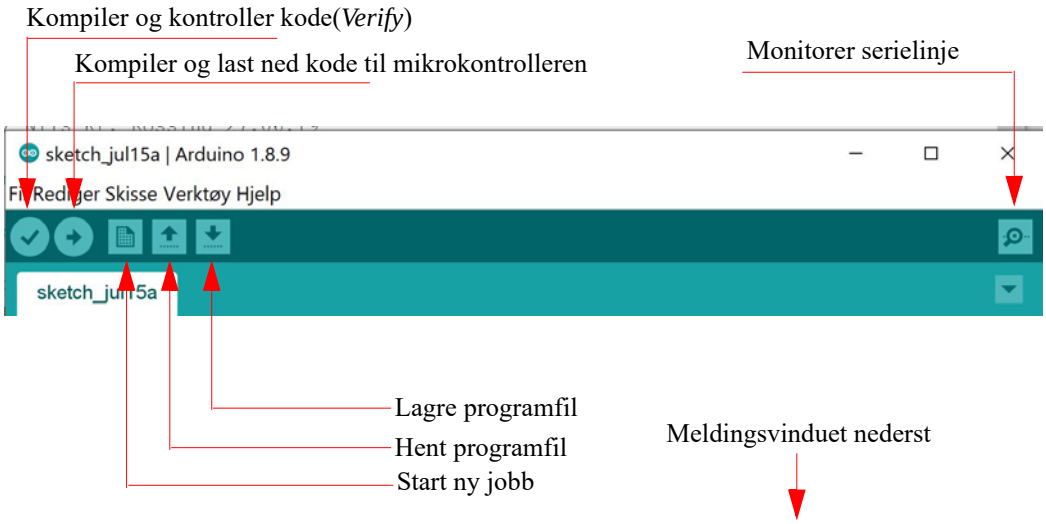
 Kompiler og last ned programmet til mikrokontrolleren.

 Hent nytt "arbeidsark", også kalt skisse ("sketch"), start ny jobb.

 Hent en eksisterende programfil.

 Lagre programfilen.

 Monitorer data sendt tilbake fra mikrokontrollerkortet på serielinjen.




**Manglende kontakt med kortet**

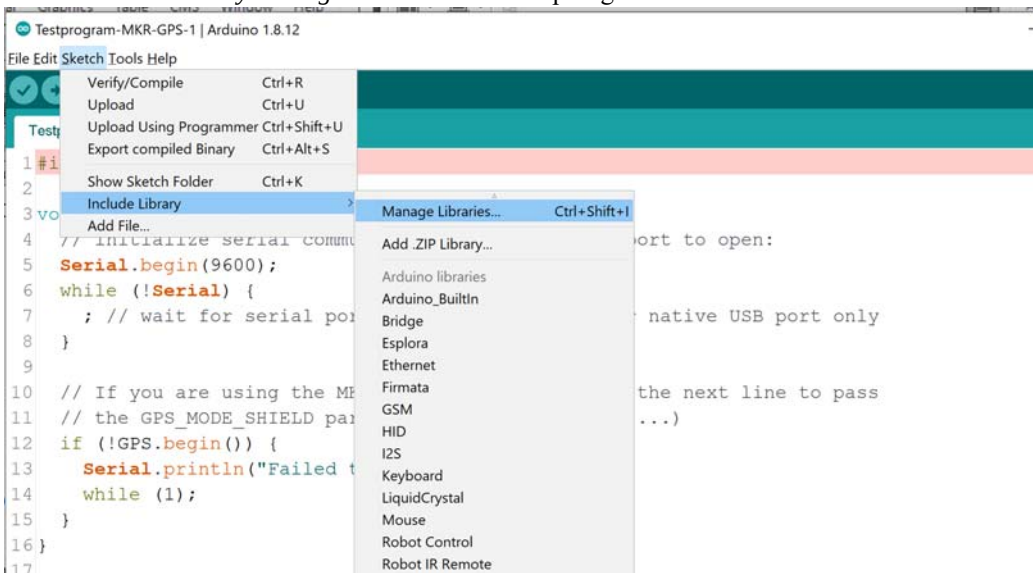
Det hender at en ikke oppnår umiddelbar kontakt med Arduino-kortet når en forsøker å laste ned et program. Feilmeldingen: *avrdude: stk500_getsync(): not in sync: resp=0x00* i meldingsvinduet betyr at det ikke oppnås kontakt med kortet. Dette kan skyldes flere ting:

- Kabelen er ikke tilkoblet, eller defekt.

- Feil port er valgt av programeditoren, som kan endres ved å velge: *Verktøy* og *Port* fra menylinjen i editoren.
- Feil type mikrokontroller-kort er valgt
  Endres ved å velge: *Verktøy* og *Kort* fra menylinjen, for så å velge rett kort, i vårt tilfelle *Arduino MKR NB 1500.* Dette er først mulig etter at ekstrapakken for MKR NB 1500 kortet er installert (se avsnitt 2.1.2 side 13).
- Manglende drivere, i så fall må driverne installeres manuelt.
- Eller rett og slett at programmet har hengt seg opp. I så fall kan man løse problemet med å lukke og starte programmet på nytt.
- Enkelte andre programmer kan ta kontroll over USB-porten, og slik blokkere for overføring til Arduino-kortet. Ett slikt program er CURA (Ultimaker)[1]. I så fall lukkes programmet som stenger for overføringen og man prøver å overføre Arduino-programmet på nytt.

### 2.1.4 Installasjon av bibliotek for å kunne programmere MKR NB 1500

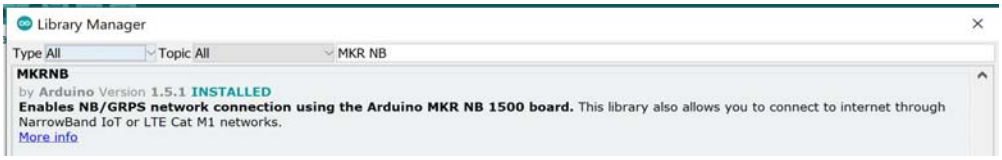
Her skal vi installere biblioteket som trengs for å programmere MKR NB 1500 og for å kunne kommunisere via NB IoT (4G). Vi bruker da *Library manager* som vi finner ved å gå inn i menyen *Sketch/Inklude Library/Manage Libraries* som vist på figuren under:




Vi får da opp følgende vindu hvor vi skriver inn *MKR NB* i søkefeltet og får opp et alternativ kalt:

1. Kan se ut som om dette problemet er fjernet på nyere utgaver av CURA.

MKRNB by Arduino Versjon 1.5.1 eller en nyere versjon.Vi velger å installere biblioteket.




Senere vil vi installere biblioteker for å håndtere ulike shield-kort som f.eks. MKR GPS og SD Proto kortet.

Du er nå klar til å bygge opp og programmere MKR NB 1500.

## 2.2 Deloppdrag 2: Oppkobling av hårdvare og montering av SIM-kort

I dette deloppdraget skal vi montere hårdvaren og SIM-kortet.

**Deloppdrag 2:** *Koble opp hårdvaren, monter antennen og installer SIM-kortet*

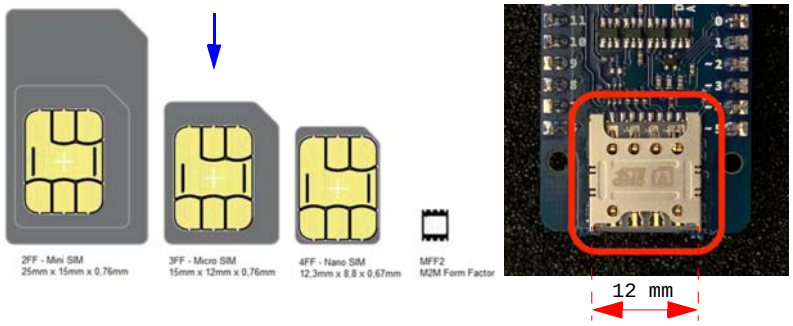
Til prosjektet vårt har vi valgt å bruke en Arduino MKR NB 1500 mikrokontroller. For å kunne overføre data via NB-IoT må den ha et SIM-kort kjøpt hos en teleoperatør.

### 2.2.1 Anskaffelse av SIM-kort

Før vi kan bli operative med å samle og overføre data, må vi skaffe et SIM-kort som passer til formålet. Vi har i dette tilfellet valgt å bruke Telenor som teleoperatør og *smalbåndsdataoverføring,* som passer til vårt formål med relativt små datamengder og med god dekning[2]. Denne kalles NB-IoT (Narrow Band Internet of Things).

SIM-kortene har forskjellig *formfaktor*, dvs. utformingen av kortet. Denne må passe til holderen hos MKR NB 1500. Sammenligner vi med holderen på MKR NB 1500 så ser vi at det er 3FF varianten, som har en bredde på 12 mm, som passer til vår teknologi.

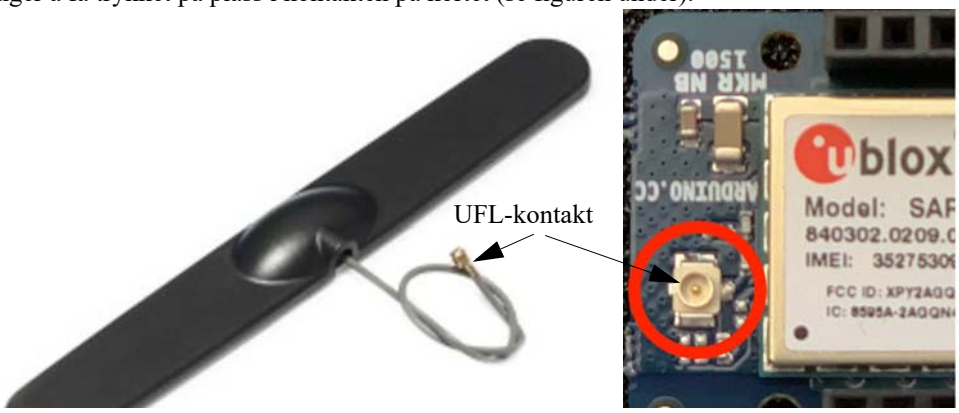
Spør læreren om å få et SIM-kort.



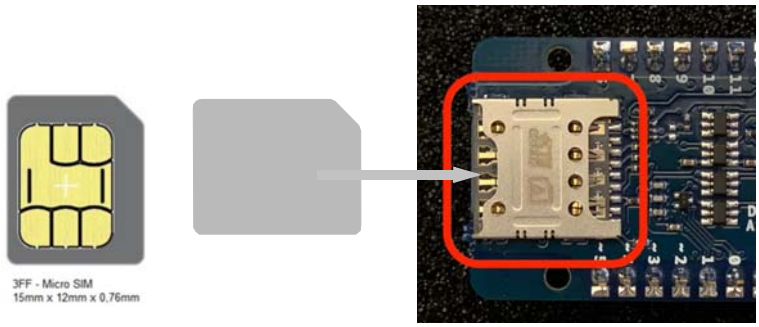

2. Det er flere leverandører av denne tjenesten. Com4 er en helnorsk mobiloperatør med hovedfokus på M2M- og IoT-kommunikasjon. Selskapet ble startet i slutten av 2011 og er en av få aktører som opererer med eget kjernenett for mobilproduksjon i Norge. https://www.com4.no/loesninger/m2miot-kommunikasjon/

### 2.2.2 Montering av mikrokontrollerkortet

Når vi har mottatt SIM-kortet og elektronikken, er vi klare til å montere antennen og SIM-kortet. Vi har valgt en liten dipolantenne med en UFL-kontakt som er særdeles liten og kan by på utfordringer å få trykket på plass i kontakten på kortet (se figuren under).




SIM-kortet skyves inn i kortholderen på undersiden av kretskortet. Det er så vidt antydet hvilken vei kortet skal skyves inn i holderen. Legg merke til at kontaktpunktene på kortet skal vende nedover, dvs. inn mot kretskortet.




## 2.3 Deloppdrag 3: Installer biblioteker og kjør testprogrammet

Vi skal nå installere en del biblioteker som vi vil ha nytte av når vi skal kjøre programmene våre. Dette er samlinger av funksjoner som vi blant annet kan bruke for å hente data fra sensorene.

> **Deloppdrag 3:** *Installer nødvendige biblioteker og last opp og kompiler eksempelprogrammet. Gå gjennom koden og få oversikt over programmet.*

### 2.3.1 *Hent og installer biblioteker og eksempelprogrammet:* `EVU-kurs23-Eksempelprogram.ino`

Vi skal i dette avsnittet laste opp og teste ut Eksempelprogrammet: `EVU-kurs23-Eksempelprogram.ino` Programmet kan utføre følgende funksjoner:

- Leser av temperatursensoren DS18B20 (trenger to biblioteker)

- Leser av GPS-mottakeren (trenger eget bibliotek)
- Bygger opp datastreng for overføring og lagring
- Skriver til SD-kort-terminal (trenger eget bibliotek)
- Sender over data via 4G til server (biblioteket skal være installert i avsnitt 2.1 på side 12)
- Holder styr på den programvare-baserte "vakthunden" som passer på å restarte programmet dersom det henger seg (trenger eget bibliotek).
- Legger mikrokontrolleren i dvale (trenger et eget bibliotek)

Men før vi kan laste opp og kompilere programmet, må vi installere bibliotekene vi trenger:

**Installer bibliotekene til temperatursensoren DS18B20 og entrådsbussen**

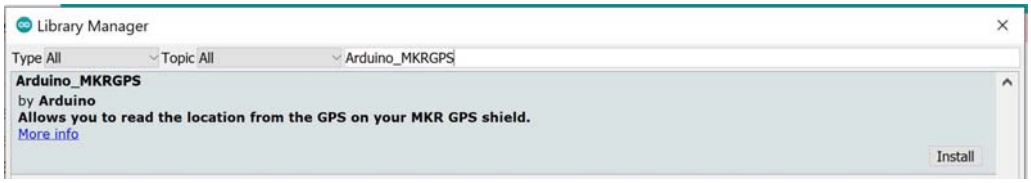
For å kunne bruke DS18B20 må vi installere to biblioteker: Ett for bruk av entrådbuss (oneWire.h) og en for avlesning av temperatursensoren (DS18B20.h).

**DS18B20:** Gå til: *Skisse* > *Inkluder bibliotek* > *Administrer bibliotek* og skriv DS18B20 i søkefeltet og velg biblioteket DS18B20 skrevet av ***Mathias Munk Hansen***:


**OneWire:** Gå til: *Skisse* > *Inkluder bibliotek* > *Administrer bibliotek* og skriv OneWire i søkefeltet og velg biblioteket OneWire skrevet av ***Paul Stoffregen***:


**Installer biblioteket til GPS-mottakeren**

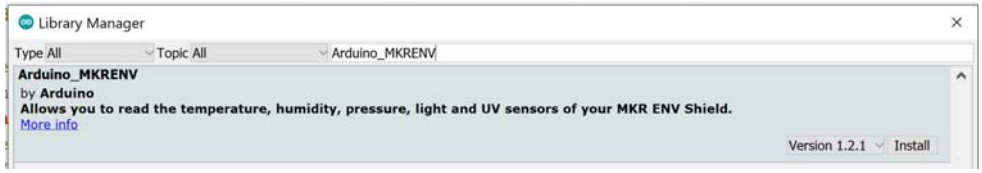
Gå til: *Skisse* > *Inkluder bibliotek* > *Administrer bibliotek* og skriv *Arduino_MKRGPS* i søkefeltet og velg å installer biblioteket som kommer opp som forslag.



**Installer biblioteket til SD-kort terminalen**

Dette ligger ved når vi installerer biblioteket til Arduino MKR ENV som også har en SD-kort terminal.

Gå til: *Skisse* > *Inkluder bibliotek* > *Administrer bibliotek* og skriv *Arduino_MKRENV* i søkefeltet og velg å installer biblioteket som kommer opp som forslag.




**Installer biblioteket for dvalefunksjoner**

For å kunne legge mikrokontrolleren i dvale så må vi installere biblioteket: ArduinoLowPower, Gå til: *Skisse* > *Inkluder bibliotek* > *Administrer bibliotek* og skriv *Arduino Low Power* i søkefeltet og velg å installer biblioteket som kommer opp som forslag.


**Installer biblioteket til programvare baserte "vakthunden" (WDTZero)**

Dette biblioteket må vi installere manuelt. Det gjør vi på følgende måte:

- En zip-utgave av biblioteket kan lastes ned fra denne siden:

  https://github.com/javos65/WDTZero/archive/refs/heads/master.zip

- Last ned zip-fila til et sted der du finner den igjen
- Gå til: *Skisse* > *Inkluder bibliotek* > *Legg til .ZIP Bibliotek* og finn zip biblioteket du nettopp lagret, og installer biblioteket

Eksempelkoder på bruk av "vakthunden" finnes bl.a. her:

https://github.com/javos65/WDTZero/blob/master/examples/WDTZero1/WDTZero1.ino

Så er vi klare til å laste opp programmet som vi skal bruke.

### 2.3.2 Last opp eksempelprogrammet

Last opp eksempelprogrammet: `EVU-kurs23-Eksempelprogram.ino` som er vedlagt i vedlegg C side 76. Programmet kan også lastes ned fra:

www.ntnu.no/skolelab/bla-hefteserie

Gå ned til overskriften: *Programmering av Tingenes internett (IoT)*
Åpne fanen: *EVU-kurs – Miljøovervåking med bøye*
Last ned og installer programmet: `EVU-kurs23-Eksempelprogram.ino`

### 2.3.3 Gå gjennom programmet og bli kjent med koden

Gå gjennom programmet og forsøk å få oversikt over de ulike delene. Bruk gjennomgangen i vedlegg A side 59, eller be om at læreren går gjennom programmet sammen med dere.

### 2.3.4 Sjekk Debug modus

Langt oppe i programmet finnes følgende programlinjer:

```
// Deklarasjon av konstanter:
#define DEBUG // Kommenter bort denne dersom Serial.print ikke skal inkluderes
//#define LED     // Kommenter bort denne dersom LED indikasjon ikke ønskes
```

Sjekk at #define DEBUG ikke er kommentert bort, mens //#define LED er kommentert bort. Dersom konstanten DEBUG er deklarert så vil vi kunne skrive ut kommentarer i monitoren. Det er bare aktuelt når kortet er koblet til PC'en med en kabel. Dersom dette ikke er tilfelle må vi kommentere bort denne linjen.

Alternativt kan vi bruke den innebygde lysdioden på kortet til å blinke for å vise hvor i programmet mikrokontrolleren er. Ønsker vi å bruke lysdioden så kommenterer vi bort //`#define DEBUG` og fjerner // foran `#define LED`.

### 2.3.5 Velg et unikt filnavn

På linje ca. 43 finner vi følgende kode:

```
char filename[]="<filnavn>.txt";// Velg navn på fil for lagring av data på serveren
```

Erstatt <filnavn> med et unikt filnavn, der du kan finne igjen dataene som lastes ned til serveren. Filen blir liggende sammen med andres filer. Husk at filnavnet skal ende på .txt, da er det lettere å åpne med en tekst-editor, for eksempel *Notepad++* som kan lastes ned gratis.

### 2.3.6 Sett opp loop()-funksjonen

Gå ned til loopen og velg inn funksjoner som er merket med rødt (se under), resten kan foreløpig være kommentert bort:

```
void loop() {

  //readGPSdata1();     // Inkluder GPS
  //readWaterTemp();    // Inkluder måling av temp. i vann
  //readBatVolt();      // Inkluder måling av batterispenning om den er implementert

  makeString();         // Bygge opp bufferet for overføring av data
  //sdPrint();          // Skriv data til SD-terminal
  //connectToGPRS();    // Koble til GPRS-nettverket
  //connectToServer();  // Koble opp mot server og overfør data
```

```
    //printData();         // Skriv data til monitoren
    printDataString();      // Skriv ut bufferet til monitoren
    //MyWatchDoggy.clear(); // Resetter vakthunden

    num++;                // Målingens nummer fra start
    //GoToSleep();        // Legg microkontrolleren og GPS'en i dvale
    delay(Pause);         // Ev. legg inn en pause i loopen
}
```

### 2.3.7 Sjekk at linjene med data skrives ut i monitoren

Kompiler og last programmet over til mikrokontrolleren. Åpne monitoren[3] og se at programmet skriver ut dummy-dataene som forventet. Det er den samme datastrengen som legges i fila på serveren og på SD-kortet. Det er funksjonen `printDataString();` som skriver ut datastrengen.

## 2.4 Deloppdrag 4: Monter sensoren for måling av vanntemperaturen

Vi skal bruke den vanntette temperatursensoren DS18B20 for måling av temperatur i vann. Det er ingen ting i veien for at vi også kan bruke denne for måling av temperatur i luft, men responsen på temperaturendringer vil være langsommere enn for vann.

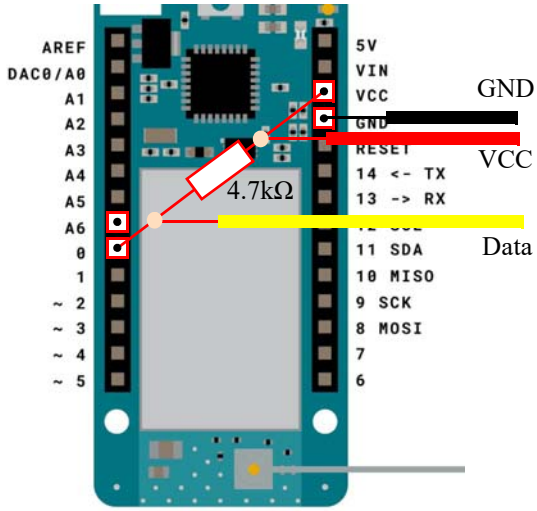
> **Deloppdrag 4:** *Les av sensoren for måling av vanntemperaturmåleren og skriv resultatet ut til monitoren. Stikk sensoren ned i et glass med vann og undersøk om temperaturen er som forventet. Avlesningen av temperaturen legges inn i datastrengen i hovedprogrammet og sendes over til serveren.*

### 2.4.1 Monter sensoren

Plugg temperatur-sensoren DS18B20 inn i MKR NB 1500 som vist på figuren under. Dette vil være en midlertidig montering for uttesting.

3. Monitoren åpnes ved å trykke på det vesle forstørrelsesglasset i øverste høyre hjørne.

Ved hjelp av en stiftlist kan vi lage oss en enkel tilkobling som vist på tegningen til høyre. Vi kobler oss da til stiftene som settes rett ned i hylsekontaktene på mikrokontrollerkortet eller shieldkortet om vi har montert et





slikt. Vi har plassert stiftene i D0 og A6, og VCC og GND, to på hver side av mikrokontrollerkortet. Signalet går til D0. Vi legger merke til at det skal være en motstand på 4,7kΩ mellom Data utgangen og VCC (3,3V). I tillegg skal jordledningen loddes til GND.

### 2.4.2 *Installer biblioteket*

Last ned og installer biblioteket for sensoren DS18B20 og biblioteket for bruk av entrådsbussen. Dette skal allerede være gjort under Deloppdrag 3 (se avsnitt 2.3.1 på side 18).

### 2.4.3 *Avlesning av temperatursensoren* (`EVU-kurs23-Eksempelprogram.ino`)

Bruk eksempelprogrammet for avlesning av sensoren DS18B20 ved å inkludere funksjonen (`readWaterTemp();`) i loop()-funksjonen og hent inn temperaturen fra sensoren, som vist på programutsnittet under. Kompiler og kjør programmet. Åpne monitoren og se at du får rimelige temperaturverdier.

```
void loop() {

  //readGPSdata1();     // Inkluder GPS
  readWaterTemp();      // Inkluder måling av temp. i vann
  //readBatVolt();      // Inkluder måling av batterispenning om den er implementert

  makeString();         // Bygge opp bufferet for overføring av data
  //sdPrint();          // Skriv data til SD-terminal
  //connectToGPRS();    // Koble til GPRS-nettverket
  //connectToServer();  // Koble opp mot server og overfør data

  printData();          // Skriv data til monitoren
  printDataString();    // Skriv ut bufferet til monitoren
  //MyWatchDoggy.clear(); // Resetter vakthunden
```

```
    num++;                // Målingens nummer fra start
    //GoToSleep();        // Legg microkontrolleren og GPS'en i dvale
    delay(Pause);         // Ev. legg inn en pause i loopen
}
```

Legg merke til hvor vanntemperaturen dukker opp i bufferet.

### 2.4.4 Mål en kjent temperatur i vann

Stikk sensoren i vann med ulike temperaturer og undersøk at den gir rimelige verdier. Kontroller målingene med et kalibrert termometer. Om nødvendig kalibrer sensoren.

## 2.5 Deloppdrag 6: Monter GPS-kortet og inkluder GPS data

Vi skal nå montere og hente inn data fra MKR GPS-kortet.

> **Deloppdrag 5:** *Monter MKR GPS-kortet og installer det tilhørende biblioteket. Bruk eksempelprogrammet og inkluder funksjonen for lesing av GPS-mottakeren. Kontroller at avleste data er som forventet*

### 2.5.1 Hent og installer MKR GPS-biblioteket

Hent ned og installer MKR GPS-biblioteket i Arduino editoren. Dette skal være gjort i Deloppdrag 3, se evt. avsnitt 2.3.1 side 18.

### 2.5.2 Monter GPS-kortet

Monter GPS-kortet ved hjelp av kabelen som kommuniserer med microkontrolleren via $I^2C$-bussen. Pass på å presse støpselet helt inn i kontakten. På bildet under er MKR GPS-kortet alt montert ved hjelp av kabel.

**NB!** *Det er viktig at GPS-mottakeren legges et sted der den har tilgang til et sett med GPS-satellitter, f.eks. utendørs eller nær et vindu med fri sikt til en større del av himmelen. Første gang den slås på kan det ta flere minutter før den låser til et akseptabelt antall satellitter. Ved bruk av batteri på GPS-kortet vil informasjonen lagres slik at det senere går fortere å låse til satellittene.*

Vi anbefaler derfor å montere batteriet et CR1216 knappebatteri.

### 2.5.3 *Inkluder funksjonen for avlesning av GPS* (`EVU-kurs23-Eksempelprogram.ino`)

Inkluder funksjonen `readGPSdata1();` i loop()-funksjonen. Den leser av GPS-mottakeren.

```
void loop() {

  readGPSdata1();       // Inkluder GPS
  readWaterTemp();      // Inkluder måling av temp. i vann
  //readBatVolt();      // Inkluder måling av baterispenning om den er implementert

  makeString();         // Bygge opp bufferet for overføring av data
  //sdPrint();          // Skriv data til SD-terminal
  //connectToGPRS();    // Koble til GPRS-nettverket
  //connectToServer();  // Koble opp mot server og overfør data

  printData();          // Skriv data til monitoren
  printDataString();      // Skriv ut bufferet til monitoren
  //MyWatchDoggy.clear(); // Resetter vakthunden

  num++;                // Målingens nummer fra start
  //GoToSleep();        // Legg microkontrolleren og GPS'en i dvale
  delay(Pause);         // Ev. legg inn en pause i loopen
}
```

Med biblioteket følger det også med eksempelkode som bl.a. inneholder et testprogram for lesing og utskrift av GPS-data til monitoren.

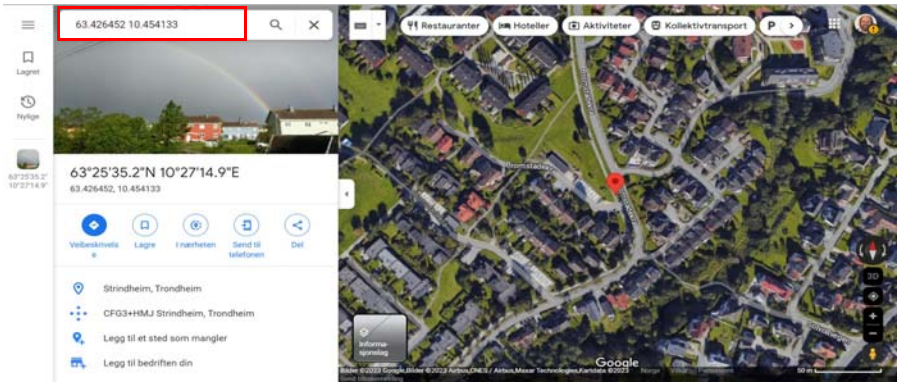
### 2.5.4 Kontroller resultatet

Åpne monitoren og studer om koordinatene som skrives ut og som legges i datastrengen ser rimelige ut.

### 2.5.5 Kontroller målt posisjon med Google maps

Klipp ut lengde- og breddegrad og lim det inn i Google Earth og kontroller om den målte posisjonen stemmer.

Når man starter https://www.google.no/maps så får man etter hvert opp vindu der man kan skrive inn bredde- og lengdegrad øverst til venstre i vinduet (rød ramme), og Google Earth vil "forflytte seg" til det angitte stedet på kartet.




Bredde- og lengdegrader legges inn som vist under:

63.426452,10.454133
Breddegrader[°],Lengdegrader[°]

Her er selvfølgelig rekkefølgen viktig.

## 2.6 Deloppdrag 6: Oppkobling til GPRS-nettet og nedlasting til serveren

I dette deloppdraget skal vi koble oss til NB-IoT-nettverket (4G) og sende databufferet til serveren.

> **Deloppdrag 6:** *Koble opp sensornoden til* NB-IoT-*nettverket og overfør data til serveren. Gå inn på serveren og kontroller at lagrede data er som forventet.*

Vi skal nå gjør forberedelser for overføring av data til serveren.

### 2.6.1 *Hent og installer bibliotek og eksempelprogrammet* (`EVU-kurs23-Eksempelprogram.ino`)

Bibliotekene for å bruke NB-IoT teknologien skal være installert og beskrevet i avsnitt 2.1.4 side 16.

### 2.6.2 Inkluder kode for oppkobling og nedlasting

Koden for oppkobling til GPRS-nettet finnes i funksjonen `connectToGPRS();` og nedlasting til serveren i funksjonen `connectToServer();`. Som det framgår så er mesteparten av funksjonene varsler om at alt går greit eller om at det oppstår feil.

Inkluder funksjonene `connectToGPRS(); og connectToServer();` i loop()-funksjonen. Det kan også være lurt å ***droppe avlesningen av GPS-mottakeren*** under denne uttestingen for å unngå å ha for mange usikre faktorer i spill samtidig.

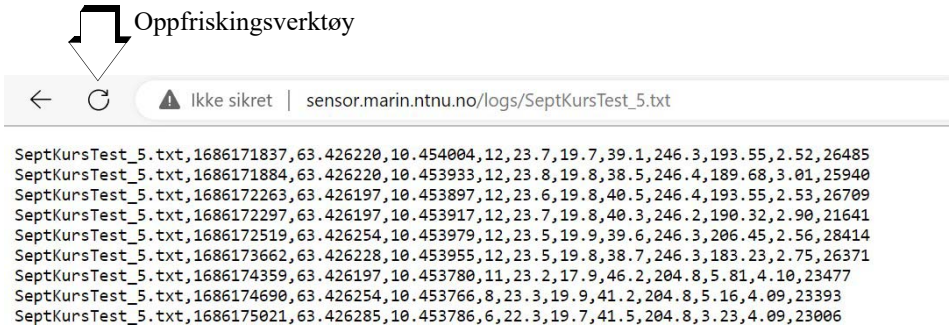
```
void loop() {

  readGPSdata1();       // Inkluder GPS
  readWaterTemp();      // Inkluder måling av temp. i vann
  //readBatVolt();      // Inkluder måling av baterispenning om den er implementert

  makeString();         // Bygge opp bufferet for overføring av data
  //sdPrint();          // Skriv data til SD-terminal
  connectToGPRS();      // Koble til GPRS-nettverket, om det ikke alt er gjort i setup
  connectToServer();    // Koble opp mot server og overfør data

  printData();          // Skriv data til monitoren
  printDataString();      // Skriv ut bufferet til monitoren
  //MyWatchDoggy.clear(); // Resetter vakthunden

  num++;                // Målingens nummer fra start
  //GoToSleep();        // Legg microkontrolleren og GPS'en i dvale
  delay(Pause);         // Ev. legg inn en pause i loopen
}
```

### 2.6.3 Kompiler og kjør eksempelprogrammet

Når programmet kjører vil det sende over tallverdiene som ligger i datastrengen (buffer) til serveren. For hver måling lages det en linje hvor måleverdiene er separert med et komma. Studer hvordan datastrengen er bygget opp i vedlegg A.1 side 69.

### 2.6.4 Sjekk data på serveren

Gå inn på følgende nettadresse: `https://sensor.marin.ntnu.no/logs/` og finn filen med navnet dere valgte, og sjekk at de forventede dataene ligger der. Bruk oppfriskningsverktøyet i fila for å laste inn nye innkomne data.




### 2.6.5 Evt. inkluder GPS-data

Dersom nedlastingen til serveren gikk greit, kan man inkludere avlesning av GPS-mottakeren. Vær oppmerksom på at det kan ta tid å få kontakt med GPS-satellittene, spesielt første gang.

## 2.7 Deloppdrag 7: Inkluder “vakthund”

I dette deloppdraget skal vi inkludere en “vakthunden” som sørger for å resette programmet dersom det stopper opp. Dette er den interne vakthunden som er realisert i programmet.

> **Deloppdrag 7:** *Inkluder en “vakthund” som resetter programmet dersom det skulle stoppe opp.*

### 2.7.1 Inkluder en programvarebasert “vakthund”

Inkluder funksjoner i programmet som styrer oppsett av “vakthunden”. Oppsettet av "vakthunden" gjøres i setup()-funksjonen. Under er det viset et utsnitt av setup()-funksjonen. Her fjerner vi kommentartegnene foran de to “vakthund” funksjonene (`MyWatchDoggy ...`) som vist under:

```
// Initialisering av "vakthund"
  MyWatchDoggy.attachShutdown(myshutdown); // Starter funksjonen myshutdown() ved reset
  MyWatchDoggy.setup(WDT_SOFTCYCLE1M);     // Setter intervalltiden for "vakthunden"
```

Her har vi valgt et “vakthund”-intervall på ca. 1 min. (64 sek). Dvs. at “vakthunden” resetter programmet dersom en syklus i programmet tar lengre tid enn 64 sek. Intervallet settes opp med konstanten `WDT_SOFTCYCLE1M`. Her kan man velge intervaller på inntil drøyt 16 min.

I tillegg må vi inkludere klarering av “vakthunden” i loop()-funksjonen (`MyWatchDoggy.clear();`). Dette gjør vi for å hindre at “vakthunden” resetter programmet så lenge programmet går som normalt og ikke stopper opp.

```
void loop() {

  readGPSdata1();       // Inkluder GPS
  readWaterTemp();      // Inkluder måling av temp. i vann
  //readBatVolt();      // Inkluder måling av baterispenning om den er implementert

  makeString();         // Bygge opp bufferet for overføring av data
  //sdPrint();          // Skriv data til SD-terminal
  connectToGPRS();      // Koble til GPRS-nettverket om det ikke alt er gjort i setup
  connectToServer();    // Koble opp mot server og overfør data

  printData();          // Skriv data til monitoren
  printDataString();    // Skriv ut bufferet til monitoren
  MyWatchDoggy.clear(); // Resetter vakthunden

  num++;                // Målingens nummer fra start
  //GoToSleep();        // Legg microkontrolleren og GPS'en i dvale
  delay(Pause);         // Ev. legg inn en pause i loopen
}
```

### 2.7.2 Sjekk at "vakthunden" fungerer

Kjør programmet og legg inn en pause i slutten av Loop()-funksjonen som er lenger enn intervallet "vakthunden" kan tillate, og se at den resetter og starter opp programmet på nytt. Dvs. at

verdien på pausen må settes til mer enn 64000 ms. Dette gjør vi i starten av programmet der vi deklarerer våre variabler.

```
long Pause                  = 64000; // En evt. pause i programloopen
```

Mot slutten av pausen vil vi høre at mikrokontrolleren kobler seg fra USB-porten og resetter programmet. Ved en slik resetting vil forbindelsen til monitoren brytes og ikke kobles opp igjen før vi lukker og starter den på nytt dersom vi bruker IDE versjon 1x, slik er det nødvendigvis ikke for IDE versjon 2x.

Vi har også erfart at mikrokontrolleren mister kontakten med GPRS-nettverket og forbindelsen til serveren under lengre pauser.

## 2.8 Deloppdrag 8: Skriv dataene til SD-kort

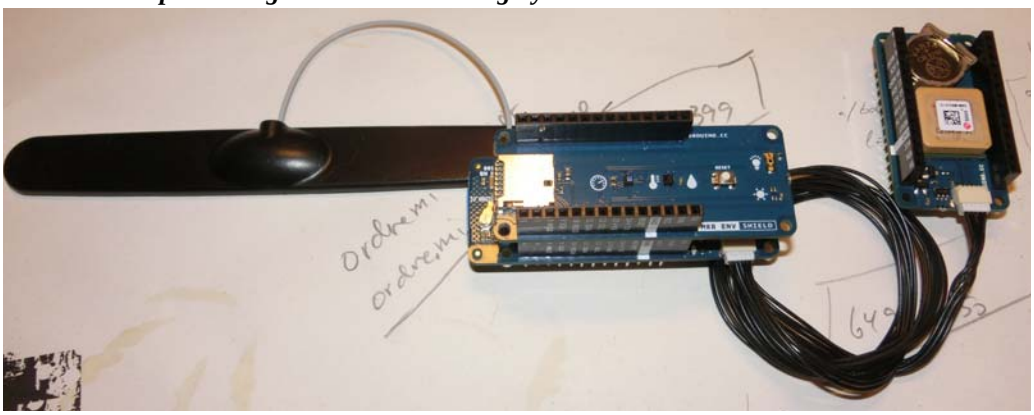
I dette deloppdraget skal vi skrive datastrengen til et SD-kort

> **Deloppdrag 8:** *Monter en SD-kort terminal, installer nødvendig bibliotek, opprett en fil og skriv måledataene til filen.*

### 2.8.1 Monter SD-kort terminalen

Den enkleste måten å inkludere en SD-kort terminal er å bruke et av shield-kortene som tilhører MKR-serien. En slik terminal finnes på følgende kort: MKR ENV (miljø-kortet), et MKR MEM kort som både inneholder mer lagerplass, en SD-kort terminal og plass for prototyping. Så finnes det et shield-kort kun med en SD-kortterminal og plass til prototyping (MKR SD Proto Shield), vi har valgt å bruke denne siste varianten her.

Alle disse kortene kan plasseres på toppen av mikrokontrollerkortet, som vist på figuren under, der miljøkortet er plassert på toppen av mikrokontrollerkortet. Ta det kortet som du har tilgjengelig og plasser det på toppen av mikrokontrollerkortet. Hylsekontaktene er merket med portnummer. ***Pass på at riktig bein kommer i riktig hylsekontakt.***



### 2.8.2 Sjekk filnavnet

I setup()-funksjonen opprettes fila. Her kan man sette opp filnavnet som skal brukes ved skriving til SD-kortet. En må gjerne gjenbruke filnavnet som ligger der (DataFile.txt) siden denne filen kun blir liggende på SD-kortet. Dersom man velger et eget navn, ***pass på at antall tegne i selve navnet ikke overskrider 8 tegn før .txt***. Blir det flere opprettes ikke fila.

```
// Lag en file med navnet DataFile.txt
  myFile = SD.open("DataFile.txt", FILE_WRITE);
  delay(1000);
  myFile.close();
  delay(100);
```

Husk å sett inn et SD-kort i terminalen.

### 2.8.3 Inkluder skriving til SD-kort

For å få skrevet dataene til SD-kortet kaller vi opp funksjonen `sdPrint();` i loop()-funksjonen som vist under:

```
void loop() {

  readGPSdata1();       // Inkluder GPS
  readWaterTemp();      // Inkluder måling av temp. i vann
  //readBatVolt();      // Inkluder måling av baterispenning om den er implementert

  makeString();         // Bygge opp bufferet for overføring av data
  sdPrint();            // Skriv data til SD-terminal
  connectToGPRS();      // Koble til GPRS-nettverket
  connectToServer();    // Koble opp mot server og overfør data

  printData();          // Skriv data til monitoren
  printDataString();    // Skriv ut bufferet til monitoren
  MyWatchDoggy.clear(); // Resetter vakthunden

  num++;                // Målingens nummer fra start
  //GoToSleep();        // Legg microkontrolleren og GPS'en i dvale
  delay(Pause);         // Ev. legg inn en pause i loopen
}
```

### 2.8.4 Lesing av fil på SD-kort

For å sjekke om dataene er kommet inn i fila på SD-kortet, tar vi ut SD-kortet og setter det inn i en tilsvarende leser i PC'en. Fila åpnes med en tekst-editor, som f.eks. Notepad++ eller lignende.

### 2.8.5 Inkluder GPS-data

Dersom GPS-data fortsatt er kommentert bort, så kan du jo prøve å inkludere denne igjen. Sørg imidlertid for at mottakeren ligger nær et vindu.

## 2.9 Deloppdrag 9: Presentasjon av måleresultatene

I dette deloppdraget skal vi presentere måleresultatene.

> **Deloppdrag 9:** *Presenter måleresultatene med passende programvare, f.eks. Excel eller et Python-script evt. Google Earth. Kommenter resultatene.*

### 2.9.1 Lag grafer av målinger

Presenter måledata som funksjon av tiden. Vis gjerne vanntemperatur som funksjon av tiden ved hjelp av Excel (avsnitt 3.3 side 41) eller ved Python skript (avsnitt 3.4 side 45).

### 2.9.2 Plott posisjonen i Google Earth

Hent fram kolonnene med lengde- og breddegrad og legg verdiene inn i Google Earth og vis på kartet hvor målingene er gjort, se avsnitt 3.5 side 54. Dersom det er gjort mens utstyret var i bevegelse kan man legge måleserien inn i kml-skriptet og plott ruta i Google Earth, se avsnitt 3.5.2 side 55.

### 2.9.3 Sjekk måledataene

Gå gjennom måledataene og sjekk at de er rimelig. Evt. beskriv avvik.

### 2.9.4 Plott eksempelfilen med Python eksempelprogram

På serveren ligger målefila SeptKursTest_5.txt (http://sensor.marin.ntnu.no/logs/SeptKursTest_5.txt). Denne fila kan leses og dataene kan plottes og dels analyseres med Python-scriptet i vedlegg B.1 side 72. For å kunne kjøre programmet må man installere programmet Spyder. Hvordan dette installeres og litt om bruken er vist i avsnitt 3.4.1 side 45.

Fila er skrevet spesielt for å lese den nevnte fila. For å lese andre filer med samme format og fra samme nettadresse må filnavnet endres.

## 2.10 Deloppdrag 10: Oppkobling av PLSC MKR-kretskort

I dette deloppdraget skal vi montere og lodde opp PLSC MKR-kortet (PLSC – Power Loop Sleep Control MKR). Dette er et kort som gjør det enklere å lade opp batteriene, opprettholde den ekstern "vakthund"-funksjonen, dvale-funksjonen og oppkonvertering av batterispenningen med mer.

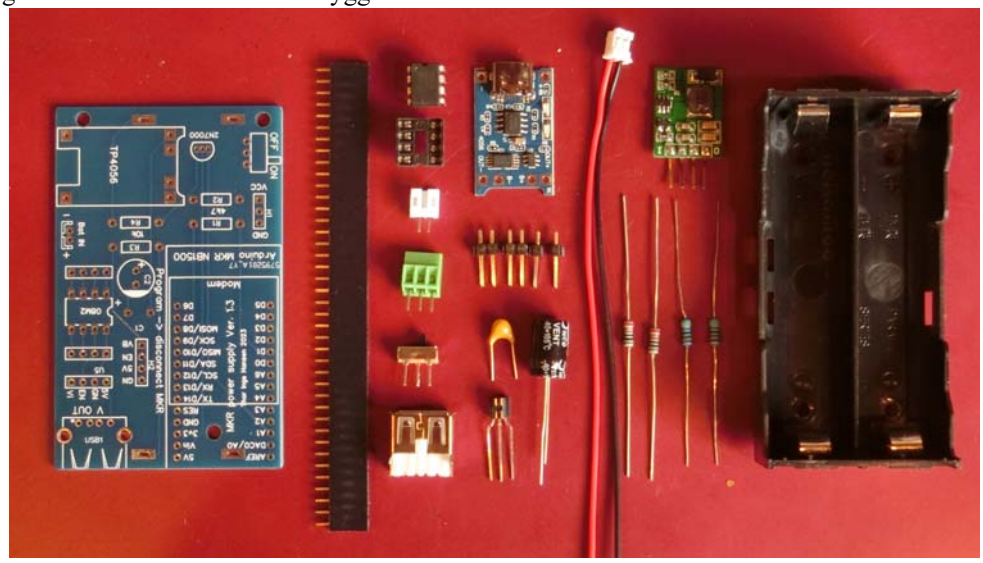
> **Deloppdrag 10:** *Lodd opp PLSC MKR-kortet og sjekk at det fungerer som tiltenkt.*

### 2.10.1 Monter og lodd opp kortet

Til dette trengs en normalt god loddebolt (30W) med en passende spiss. Spissen må ikke være for bred, men egne seg til denne typen arbeid.

I dette avsnittet skal vi beskrive montering og oppkobling av PLSC MKR-kortet. Kortet besørger lading av batteriet fra USB-C inngang, inneholder den eksterne "vakthunden" som holder rede på timingen for målingene, sørger for å slå av kretsene mellom hver måleperiode og hever spenningen fra batterispenningen på 3,7 V til 5,2 volt.

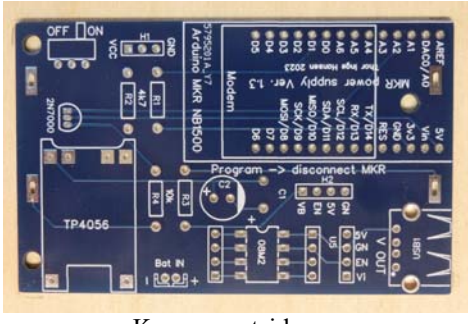
Figuren under viser innholdet i byggesettet:



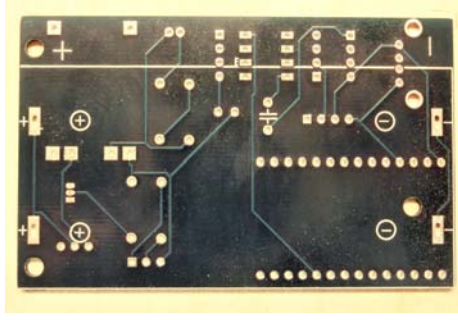
Følgende verktøy kreves for monteringen:

- Loddebolt 30W med liten spiss
- Avsug
- Loddetinn
- Sideavbiter

1. **Kretskortet**
   Kretskortet har en komponentside der de aller fleste komponentene skal plasseres (unntatt batteriholderen), og en loddeside der komponentene skal loddes til banene på kortet.




Komponentside

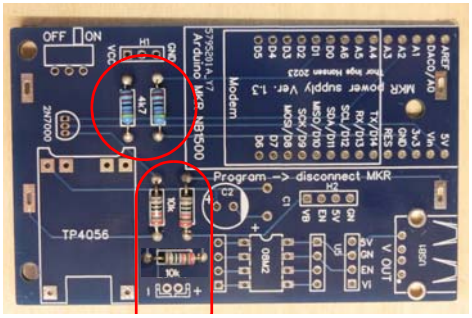


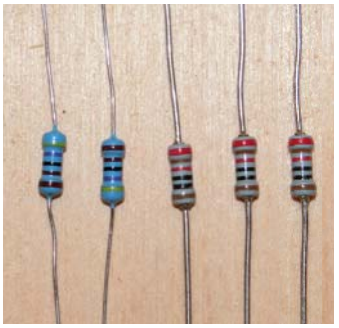
Loddeside

Kretskoret kan muligens avvike noe fra bildet over, da det stadig lage forbedrede utgaver.

2. **Motstandene**

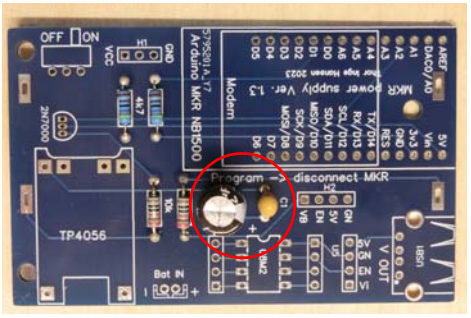
Motstandene er av to verdier:
R1 og R2 = 4,7kΩ (Gul, Fiolett, Rød, Gull (5%)) eller (Gul, Fiolett, Sort, Brun, Brun (1%))
R3, R4 og R5 = 10kΩ (Brun, Sort, Oransje, Gull (5%)) eller (Brun, Sort, Sort, Rød, Brun (1%)) Lodd på loddesida og klipp av ledninger.

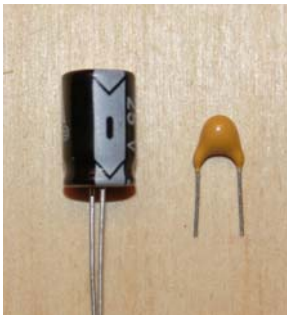




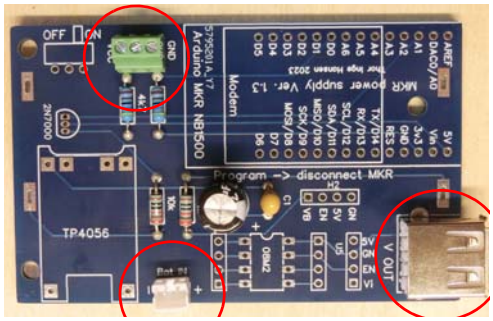
3. **Kondensatorene**

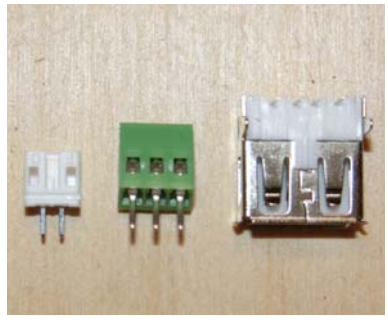
Kondensatorene er av to verdier:
C1 = 100nF (104)
C2 = 100µF (Husk at pinne merket – skal plassers i hull motsatt av +).
Lodd på loddesida og klipp av ledninger.



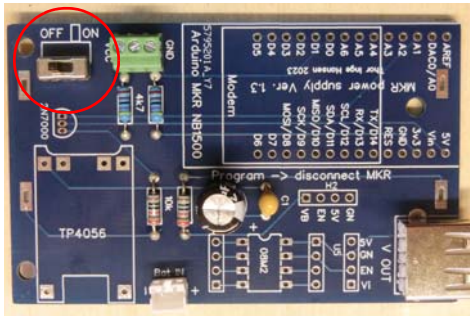


4. **Batterikontakt, Koblingsplint og USB-A kontakt hunn**
   Tenk gjennom hvilken vei det er hensiktsmessig å sette koblingsplinten
   Lodd på loddesida og klipp av ledninger.

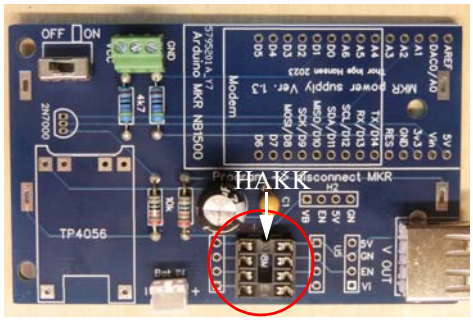




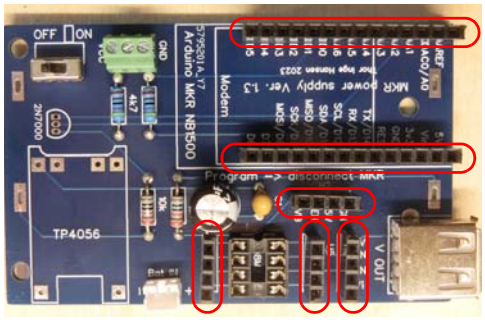
5. **Glidebryter**
   Plasser glidebryteren og lodd på loddesida. Det spiller ingen rolle hvilken vei bryteren settes.

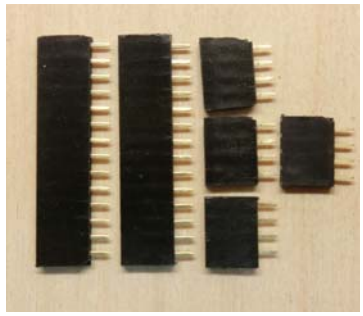


6. **8-pins sokkel**
   Husk å plasser hakket på rett side. Det skal vise hvilken vei 08M2+ mikrokontrolleren skal stå. 8pin IC-sokkelen har så tynne bein at de er lettere å brette flate på baksiden før lodding.
   Lodd på loddesida. Det er unødvendig å klippe ledningene på baksiden.

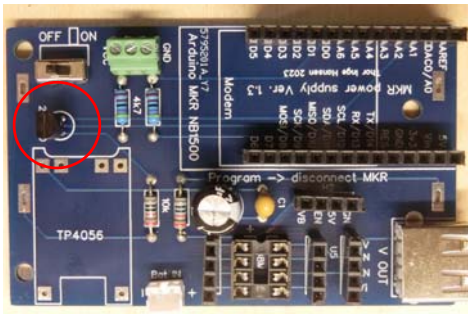


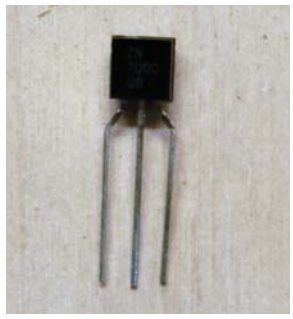
7. **Hylsekontakter**
   Del opp hylsekontakten i seks remser med følgende lengde:
   2 x 14 pins (til MKR NB 1500)
   4 x 4 pins (til booster, til 3 målepunkter a 4 pins)
   Lodd på loddesida og klipp av ledninger.



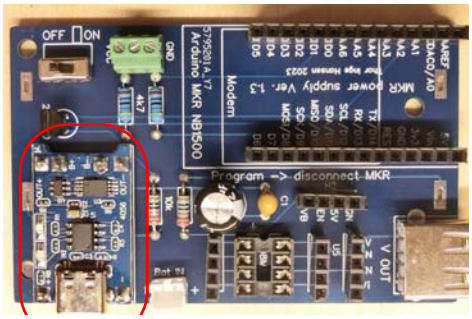


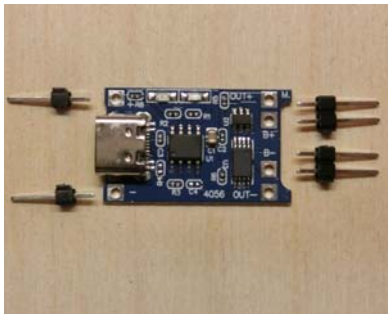
8. **MOSFET transistor**
   Husk flat side rett vei. Transistoren må gjerne stå 5 – 10 mm over kortet.
   Lodd på loddesida og klipp av ledninger.





9. **Ladekrets**
   Monter stiftlister på ladekrets og plasser på kortet.
   Lodd på loddesida og klipp av ledninger.



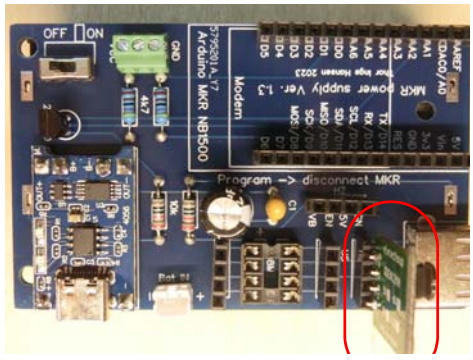


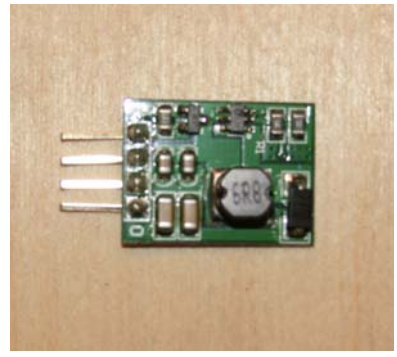
10. **Batteriholder**
    Snu kortet og plasser batteriholderen. Merk at batteriholderen har + og –, det samme har

kretskortet. Pass på at + og – kommer på rett plass. Lodd loddeørene på komponentsida. **NB!** Det anbefales imidlertid å koble opp 18650-holderen via batterikontakten slik at brettet kan funksjonstestes med denne som tilførsel før batteriholderen loddes på baksiden. Dette gjør at det er lettere å fikse loddefeil før batteriholderen monteres

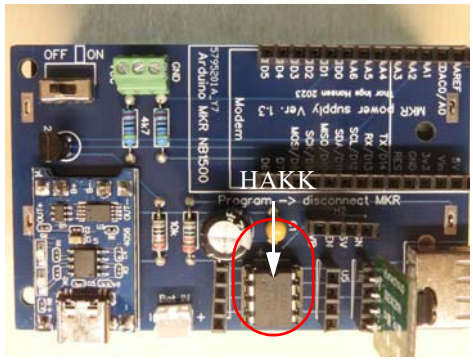


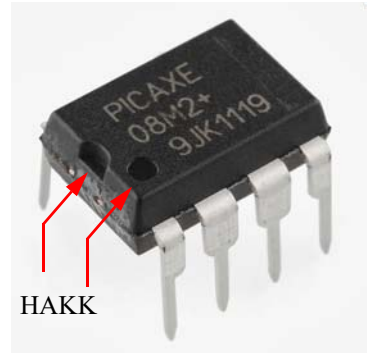
11. **Booster**
Plugg boosteren inn i hylselista som vist på bildet under. Pass på at den står riktig vei. Sørg for at Vi og Vo kommer på rett plass.



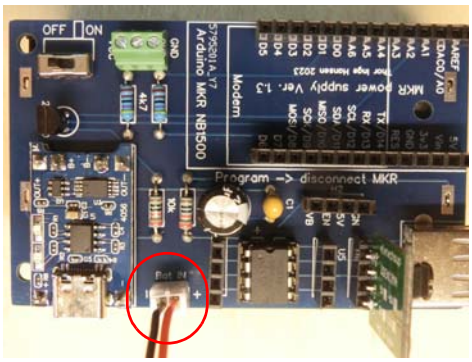



12. **Sett i "Vakthunden" 08M2+**
Mikrokontrolleren som utgjør den eksterne "vakthunden" er en IC med 8 pinner (08M2+).



13. **Sett i batteripluggen** om det er ønskelig med ekstern spenningskilde.
    Pass på at batteripluggen har rød ledning på + siden



Gå nøye over loddingene på kortet og sjekk om det er manglende loddinger, eller broer av loddetinn mellom punkter som ikke skal være koblet sammen.

### 2.10.2 Installer operativt program

Vi har kalt programmet *operativt* da det er et program vi har brukt for å samle inn data i felten. Programmet finnes i vedlegg D side 85, men hentes helst fra:

www.ntnu.no/skolelab/bla-hefteserie

Gå ned til overskriften: *Programmering av Tingenes internett (IoT)*
Åpne fanen: *EVU-kurs: Miljøovervåking med bøye*
Last ned og installer programmet: *Operativt_1.ino*

### 2.10.3 Legg inn personlig filnavn:

Vennligst legg inn et unikt filnavn for deres prosjekt. Dette gjøres ved å endre filnavnet vist på følgende programlinje:

```
char filename[] = "<unikt filnavn>.txt";
```

Bytt ut `<unikt filnavn>` med et unikt navn, f.eks. `VoldSkole_Erik`. Slik at det blir stående:

```
char filename[] = "VoldSkole_Erik.txt";
```

### 2.10.4 Kjør programmet og sjekk data

Kjør programmet over noe tid og sjekk at de ønskede dataene kommer inn på filen med det unike navnet.

### 2.10.5 Visualisering av data

Bruk Excel, Python eller Google Earth for å visualisere dataene.

# 3 Presentasjon og behandling av måledata

I dette kapittelet skal vi gi eksempler på hvordan vi henter inn data til Excel og Google Earth for visning av resultatene. I tillegg skal vi se hvordan vi kan bruke Python[4] til å lese, plotte og analysere data.

## 3.1 Skriving til og henting av data fra server

### 3.1.1 Skriving til fil på serveren

De innsamlede dataene fra hver deltaker eller gruppe legges i den samme katalogen på serveren. Hver deltager kan imidlertid opprette en unik fil for ulike måleserier om det er ønskelig.

Katalogen er bestemt av eieren av serveren, i dette tilfellet Inst. for marin teknikk. Filnavnet bestemmes av et av argumentene i bufferet som overfører datapakken. Vi har tidligere vist oppbyggingen av tekststrengen som legges inn i bufferet:

```
sprintf(buffer,"/cgi-bin/tof.cgi?Gruppe-1.txt,
,%d,%.2f,%.6f,%.6f,%.1f,%.1f,%.1f,%.1f,%.1f",
no, timeSec, longitude, latitude, temperature, w_temperature,
humidity, pressure, illuminance);
```

Første argument /cgi-bin/tof.cgi – cgi står for Common Gateway Interface og er et lite script (program) som web-servere bruker for å styre data på serveren. I vårt tilfelle vil scriptet ta tak i det neste argumentet etter "?", i dette eksempelet **`Gruppe-1.txt`** og opprette en fil med dette navnet, evt. skrive inn i en eksisterende fil med det samme navnet. Dernest legges dataene fortløpende inn i filen i henhold til ønsket format.

Det er derfor viktig at hver enkelt er seg bevisst hvilket filnavn som brukes slik at de finner igjen data sine.

### 3.1.2 Lesing av data fra fil på serveren

Gå til følgende webadresse:

```
http://sensor.marin.ntnu.no/logs/
```

Dataene finnes under fila:

```
http://sensor.marin.ntnu.no/logs/<filnavn>.txt
```

Dere må selv holde orden på filnavnene og sørge for at ingen bruker samme filnavn slik at dataene ikke blandes.

Dere kan åpne fila med et program som kan åpne tekst-filer, f.eks. Notepad++ eller lignende. Innholdet kan enten kopieres eller hele fila kan lagres for senere å hentes opp i f.eks. Excel eller med et Python-skript.

4. Python programmene er utviklet av Thor Inge Hansen ved Skien vgs.

## 3.2 Skrive data til fil

Følgende gir noen anbefalinger for organisering av data:

### 3.2.1 Lagre rådata

Dersom man ønsker å redusere sannsynligheten for å miste data pga. feil i programmeringen, vil det være fornuftig å sende over, eller lagre rådata. Behandlingen av dataene kan likevel lett gjøres i etterbehandlingen i f.eks. Excel, om det skulle være nødvendig. Overfører man beregnede data og man er så uheldig å regne feil i sensornoden før data lagres eller overføres til serveren, så kan dataene være tapt for alltid.

### 3.2.2 Tidsangivelse

Inkluder en teller og en tidsangivelse for når dataene ble samlet inn. En teller kan være grei for å se om man har mistet noen målinger under veis. En annen måte er å skrive inn tiden fra Arduino'en ble startet (evt. restartet), ved bruk av funksjonen:

```
millis();
```

som returnerer en verdi som er lik antallet millisekunder siden mikrokontrolleren ble startet/ restartet. En må imidlertid være klar over at funksjonen vil gå ut over sitt område etter ca. 50 dager, hvilket kan være for lite i noen tilfeller.

En annen mulighet er å bruke tidsangivelsen hentet fra GPS-mottakeren ("Epoch time") og legge den inn i fila. En nøyaktig tidsangivelse vil være essensiell dersom målinger skal brukes av meteorologisk institutt eller andre[5]. En fordel med å bruke epoketiden er at den er uavhengig om sensornoden blir slått av eller legges i dyp dvale. På den annen siden vil dette tidsstempelet gå tapt dersom man ikke oppnår kontakt med GPS-satellittene.

### 3.2.3 Skilletegn

Verdiene som skrives til fil skilles med et skilletegn. Her kan man i prinsippet bruke ulike tegn f.eks.: < > ; ; eller tab. Dersom man bruker "." og "," må man være klar over hva som brukes som desimalpunkt i den aktuelle sammenhengen. Imidlertid krever KML-fila med GPS-koordinatene at koordinatene skilles med komma, vi har derfor i denne sammenhengen valgt å bruke "," siden "." brukes som desimaltegn. Skal man bruke Python script for å behandle dataene så anbefales også ",".

### 3.2.4 Den endelige datafilen

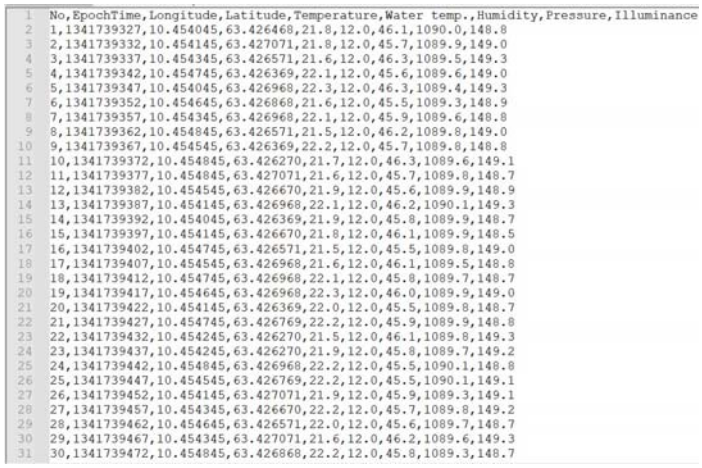
For å teste ut bruk av Excel kan man lage en testfil med simulerte data. I filen beskrevet under har vi kun brukt en teller og "," som skilletegn. I dette tilfellet går dette greit fordi vi vet at det tas én punktprøve hvert 5. sekund og at desimaltegnet er et punktum.

```
Serial.print(no);          // Målenummer
```

5. Dokumentasjon av MKR GPS biblioteket: https://www.arduino.cc/reference/en/libraries/arduino_mkrgps/

```
Serial.print (",");
Serial.print(epochTime);       // Epoketid GPS i sekunder siden 1.1.80
Serial.print (",");
Serial.print(longitude,6);     // Posisjonens lengdegrader
Serial.print (",");
Serial.print(latitude,6);      // Posisjonens breddegrader
Serial.print (",");
Serial.print(temperature,1);   // Målt lufttemperatur i grader C
Serial.print (",");
Serial.print(w_temperature,1); // Målt vanntemperatur i grader C
Serial.print (",");
Serial.print(humidity,1);      // Målt luftfuktighet i % rel. fuktighet
Serial.print (",");
Serial.print(pressure,1);      // Målt lufttrykk i mBar
Serial.print (",");
Serial.println(illuminance,1); // Målt lysintensitet ved overflata i lux
```

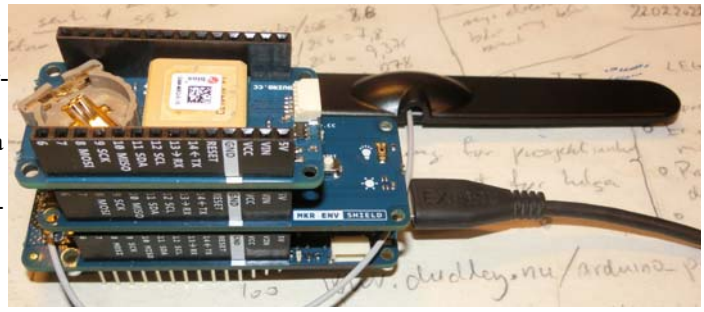
Figuren under viser den endelige filen vist ved hjelp av Notepad++ :



```
No,EpochTime,Longitude,Latitude,Temperature,Water temp.,Humidity,Pressure,Illuminance
1,1341739327,10.454045,63.426468,21.8,12.0,46.1,1090.0,148.8
2,1341739332,10.454145,63.427071,21.8,12.0,45.7,1089.9,149.0
3,1341739337,10.454345,63.426571,21.6,12.0,46.3,1089.5,149.3
4,1341739342,10.454745,63.426369,22.1,12.0,45.6,1089.6,149.0
5,1341739347,10.454045,63.426968,22.3,12.0,46.3,1089.4,149.3
6,1341739352,10.454645,63.426868,21.6,12.0,45.5,1089.3,148.9
7,1341739357,10.454345,63.426968,22.1,12.0,45.9,1089.6,148.8
8,1341739362,10.454845,63.426571,21.5,12.0,46.2,1089.8,149.0
9,1341739367,10.454545,63.426369,22.2,12.0,45.7,1089.8,148.8
10,1341739372,10.454845,63.426270,21.7,12.0,46.3,1089.6,149.1
11,1341739377,10.454845,63.427071,21.6,12.0,45.7,1089.8,148.7
12,1341739382,10.454545,63.426670,21.9,12.0,45.6,1089.9,148.9
13,1341739387,10.454145,63.426968,22.1,12.0,46.2,1090.1,149.3
14,1341739392,10.454045,63.426369,21.9,12.0,45.8,1089.9,148.7
15,1341739397,10.454145,63.426670,21.8,12.0,46.1,1089.9,148.5
16,1341739402,10.454745,63.426571,21.5,12.0,45.5,1089.8,149.0
17,1341739407,10.454545,63.426968,21.6,12.0,46.1,1089.5,148.8
18,1341739412,10.454745,63.426968,22.1,12.0,45.8,1089.7,148.7
19,1341739417,10.454645,63.426968,22.3,12.0,46.0,1089.9,149.0
20,1341739422,10.454145,63.426369,22.0,12.0,45.5,1089.8,148.7
21,1341739427,10.454745,63.426769,22.2,12.0,45.9,1089.9,148.8
22,1341739432,10.454245,63.426270,21.5,12.0,46.1,1089.8,149.3
23,1341739437,10.454245,63.426270,21.9,12.0,45.8,1089.7,149.2
24,1341739442,10.454845,63.426968,22.2,12.0,45.5,1090.1,148.8
25,1341739447,10.454545,63.426769,22.2,12.0,45.5,1090.1,149.1
26,1341739452,10.454145,63.427071,21.9,12.0,45.9,1089.3,149.1
27,1341739457,10.454345,63.426670,22.2,12.0,45.7,1089.8,149.2
28,1341739462,10.454645,63.426571,22.0,12.0,45.6,1089.7,148.7
29,1341739467,10.454345,63.427071,21.6,12.0,46.2,1089.6,149.3
30,1341739472,10.454845,63.426868,22.2,12.0,45.8,1089.3,148.7
```

Legg merke til at vi har fått med en tekst øverst. Dette er en tekst som er skrevet ut i setup()-funksjonen slik at den kun kommer med en gang. Dette er bare mulig dersom vi lar være å slå av spenningen på mikrokontrolleren mellom hver måling. Det er viktig at vi lagrer fila som en vanlig tekstfil eller CSV-fil (Comma Seperated Values).

## 3.3 Bruk av Excel for visualisering av data

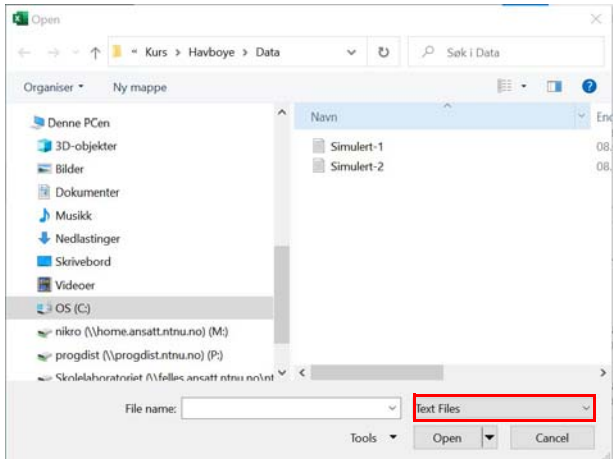


Et praktisk og vanlig verktøy for analyse av data er Excel. I dette avsnittet skal vi vise hvordan vi kan importere og tegne ut grafer av innsamlede data fra bøya. I dette eksempelet har vi følgende data: Måling nr., tidsangivelse (Epoketid), lengde- og breddegrad, luft- og vanntemperatur, luftfuktighet, lufttrykk og lysintensitet på overflata. Dataene for dette eksempelet er simulert.

### 3.3.1 Importer data fra en tekst-fil til Excel

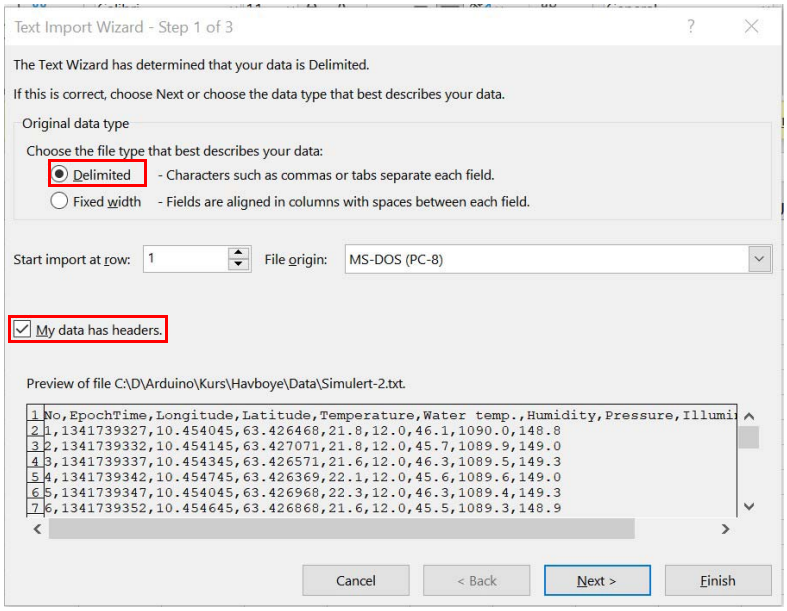
Importering av data i Excel kan gjøres på ulike måter. En måte som fungerer er å hente inn filen med "Open". Da vil man ledes gjennom følgende vinduer for at dataene skal bli formatert på ønsket måte.

Velg ønsket fil:

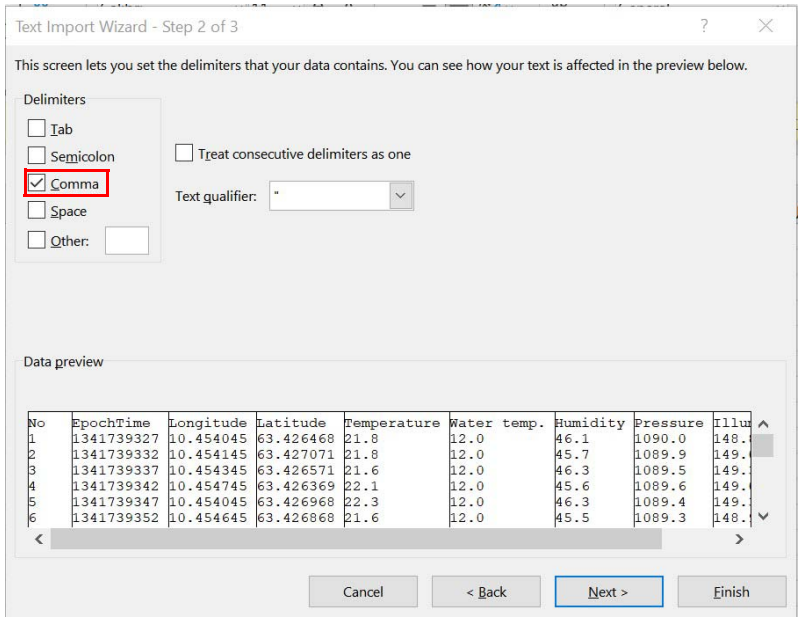



Siden det er en tekst-fil så velg denne typen format i innboksen nederst i høyre hjørne, dermed blir slike filer synlige i fil-oversikten.

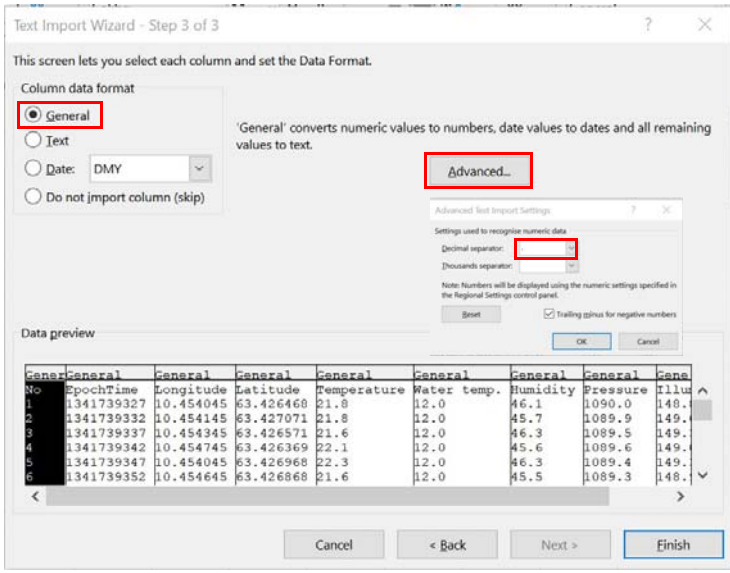
Velg “Data med skilletegn” (Delimited) og bestem fra hvilken rad du ønsker å starte å importere data. Likeså er det viktig å merke av om det brukes topptekst, som vi har gjort her. Trykk så “Neste”:




Velg hvilken type skilletegn som er benyttet. I vårt tilfelle har vi benyttet “komma”.



Tilslutt velges hvilken type data det gjelder, i vårt tilfelle er det standard tallformat (General). Dersom vi skal importere flyttall (med komma), velges desimalpunktet, "," eller "." ved å velge "Avansert". Det må vi gjøre i vårt tilfelle, ellers har dataene en tendens til å bli til datoer eller annen tekst.




Dermed skal dataene være importert i regnearket som vist på figuren under.

| | A | B | C | D | E | F | G | H | I | J |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | No | EpochTim | Longitude | Latitude | Temperati | Water tem | Humidity | Pressure | Illuminance | |
| 2 | 1 | 1,34E+09 | 10,45405 | 63,42647 | 21,8 | 12 | 46,1 | 1090 | 148,8 | |
| 3 | 2 | 1,34E+09 | 10,45415 | 63,42707 | 21,8 | 12 | 45,7 | 1089,9 | 149 | |
| 4 | 3 | 1,34E+09 | 10,45435 | 63,42657 | 21,6 | 12 | 46,3 | 1089,5 | 149,3 | |
| 5 | 4 | 1,34E+09 | 10,45475 | 63,42637 | 22,1 | 12 | 45,6 | 1089,6 | 149 | |
| 6 | 5 | 1,34E+09 | 10,45405 | 63,42697 | 22,3 | 12 | 46,3 | 1089,4 | 149,3 | |
| 7 | 6 | 1,34E+09 | 10,45465 | 63,42687 | 21,6 | 12 | 45,5 | 1089,3 | 148,9 | |
| 8 | 7 | 1,34E+09 | 10,45435 | 63,42697 | 22,1 | 12 | 45,9 | 1089,6 | 148,8 | |
| 9 | 8 | 1,34E+09 | 10,45485 | 63,42657 | 21,5 | 12 | 46,2 | 1089,8 | 149 | |
| 10 | 9 | 1,34E+09 | 10,45455 | 63,42637 | 22,2 | 12 | 45,7 | 1089,8 | 148,8 | |
| 11 | 10 | 1,34E+09 | 10,45485 | 63,42627 | 21,7 | 12 | 46,3 | 1089,6 | 149,1 | |
| 12 | 11 | 1,34E+09 | 10,45485 | 63,42707 | 21,6 | 12 | 45,7 | 1089,8 | 148,7 | |
| 13 | 12 | 1,34E+09 | 10,45455 | 63,42667 | 21,9 | 12 | 45,6 | 1089,9 | 148,9 | |
| 14 | 13 | 1,34E+09 | 10,45415 | 63,42697 | 22,1 | 12 | 46,2 | 1090,1 | 149,3 | |
| 15 | 14 | 1,34E+09 | 10,45405 | 63,42637 | 21,9 | 12 | 45,8 | 1089,9 | 148,7 | |
| 16 | 15 | 1,34E+09 | 10,45415 | 63,42667 | 21,8 | 12 | 46,1 | 1089,9 | 148,5 | |
| 17 | 16 | 1,34E+09 | 10,45475 | 63,42657 | 21,5 | 12 | 45,5 | 1089,8 | 149 | |
| 18 | 17 | 1,34E+09 | 10,45455 | 63,42697 | 21,6 | 12 | 46,1 | 1089,5 | 148,8 | |
| 19 | 18 | 1,34E+09 | 10,45475 | 63,42697 | 22,1 | 12 | 45,8 | 1089,7 | 148,7 | |
| 20 | 19 | 1,34E+09 | 10,45465 | 63,42697 | 22,3 | 12 | 46 | 1089,9 | 149 | |

Siden vi ikke trenger å gjøre beregninger på noen av dataene, så kan vi illustrere verdiene i hver kolonne som grafer om vi skulle ønske det.

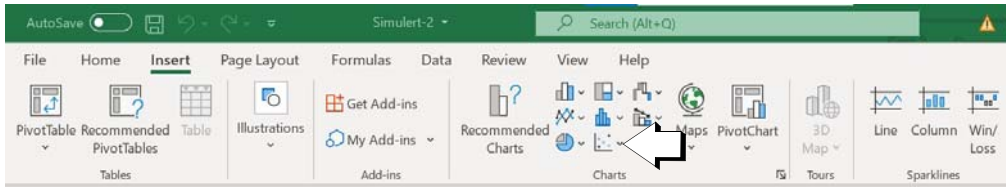
### 3.3.2 Lage grafer

Slik kan vi lage grafer:

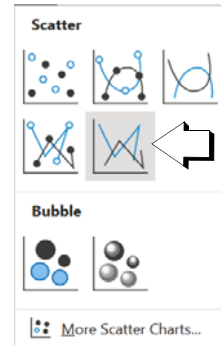
1. Merk de to kolonnene som skal representeres av en graf.
   Det kan være lurt å lage en overskrift på kolonnene om det ikke alt er gjort. Overskriften vil automatisk tilordnes seriene i grafen. Merk de kolonnene som skal utgjøre datasettet. I vårt tilfelle kan det være fornuftig å bruke Epoketiden langs x-aksen. Vi får da tegnet grafene som funksjon av tiden, f.eks. temperatur som funksjon av tiden.

| | A | B | C | D | E | F | G | H | I | J |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | No | EpochTim | ongitude | Latitude | Temperati | Vater ten | Humidity | Pressure | Illuminance | |
| 2 | 1 | 1,34E+09 | 10,45405 | 63,42647 | 21,8 | 12 | 46,1 | 1090 | 148,8 | |
| 3 | 2 | 1,34E+09 | 10,45415 | 63,42707 | 21,8 | 12 | 45,7 | 1089,9 | 149 | |
| 4 | 3 | 1,34E+09 | 10,45435 | 63,42657 | 21,6 | 12 | 46,3 | 1089,5 | 149,3 | |
| 5 | 4 | 1,34E+09 | 10,45475 | 63,42637 | 22,1 | 12 | 45,6 | 1089,6 | 149 | |
| 6 | 5 | 1,34E+09 | 10,45405 | 63,42697 | 22,3 | 12 | 46,3 | 1089,4 | 149,3 | |
| 7 | 6 | 1,34E+09 | 10,45465 | 63,42687 | 21,6 | 12 | 45,5 | 1089,3 | 148,9 | |
| 8 | 7 | 1,34E+09 | 10,45435 | 63,42697 | 22,1 | 12 | 45,9 | 1089,6 | 148,8 | |
| 9 | 8 | 1,34E+09 | 10,45485 | 63,42657 | 21,5 | 12 | 46,2 | 1089,8 | 149 | |
| 10 | 9 | 1,34E+09 | 10,45455 | 63,42637 | 22,2 | 12 | 45,7 | 1089,8 | 148,8 | |
| 11 | 10 | 1,34E+09 | 10,45485 | 63,42627 | 21,7 | 12 | 46,3 | 1089,6 | 149,1 | |
| 12 | 11 | 1,34E+09 | 10,45485 | 63,42707 | 21,6 | 12 | 45,7 | 1089,8 | 148,7 | |
| 13 | 12 | 1,34E+09 | 10,45455 | 63,42667 | 21,9 | 12 | 45,6 | 1089,9 | 148,9 | |
| 14 | 13 | 1,34E+09 | 10,45415 | 63,42697 | 22,1 | 12 | 46,2 | 1090,1 | 149,3 | |
| 15 | 14 | 1,34E+09 | 10,45405 | 63,42637 | 21,9 | 12 | 45,8 | 1089,9 | 148,7 | |
| 16 | 15 | 1,34E+09 | 10,45415 | 63,42667 | 21,8 | 12 | 46,1 | 1089,9 | 148,5 | |
| 17 | 16 | 1,34E+09 | 10,45475 | 63,42657 | 21,5 | 12 | 45,5 | 1089,8 | 149 | |
| 18 | 17 | 1,34E+09 | 10,45455 | 63,42697 | 21,6 | 12 | 46,1 | 1089,5 | 148,8 | |
| 19 | 18 | 1,34E+09 | 10,45475 | 63,42697 | 22,1 | 12 | 45,8 | 1089,7 | 148,7 | |
| 20 | 19 | 1,34E+09 | 10,45465 | 63,42697 | 22,3 | 12 | 46 | 1089,9 | 149 | |

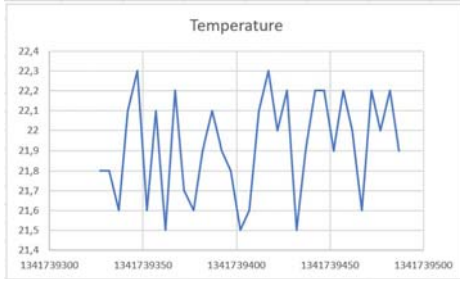
2. Velg type representasjon fra menyen "Sett inn" (Insert) og velg f.eks. punktdiagram med linjer mellom punktene i diagrammet:



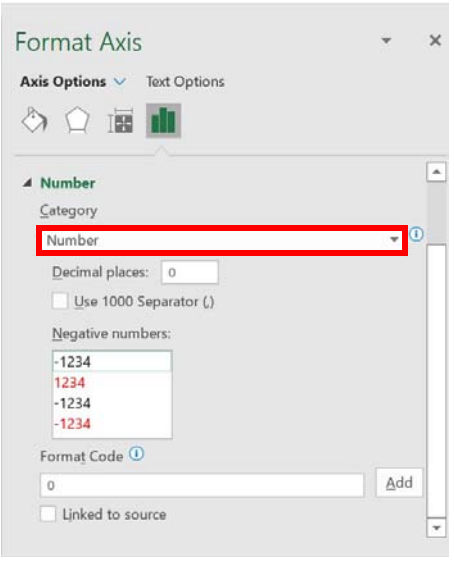

3. Velg linjediagram fra nedtrekksmenyen som vist på figuren under til høyre.



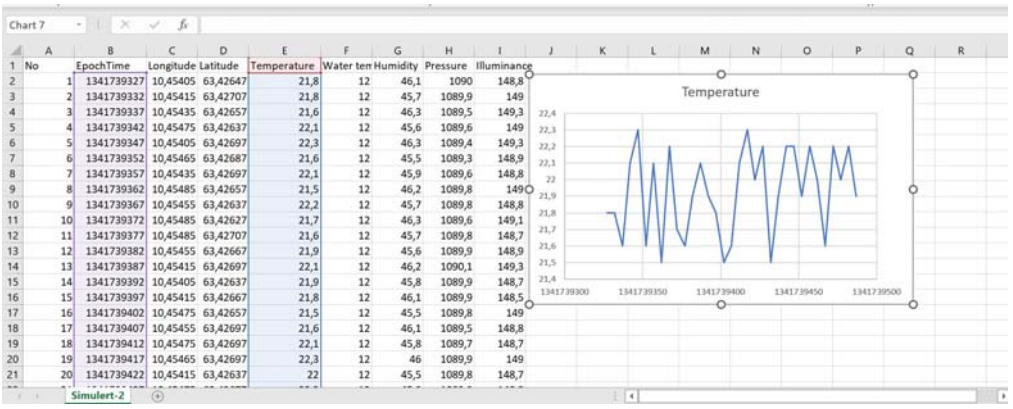

4. Når dette er gjort vil diagrammet tegnes ut som vist på figuren under.



5. Ønsker vi å endre på formatet på aksene, så klikker vi på den akse-benevningen vi ønsker å endre og vi får opp vinduet vist til høyre. Her kan man f.eks. endre på tallformatet til akse-benevningen, f.eks. fra vitenskapelig notasjon til vanlig tallnotasjon for tidsangivelsen langs x-aksen.




6. Vi kan dermed ende opp med en presentasjon som vist under:




## 3.4 Bruk av Python for å presentere og analysere dataene[6]

### 3.4.1 Installasjon og oppstart med Python

La oss først se hvordan vi etablerer programmeringsmiljøet med Python. Dette er blant annet hentet fra boka *Python for realfag* skrevet av: Hagen, Lysaker[7].

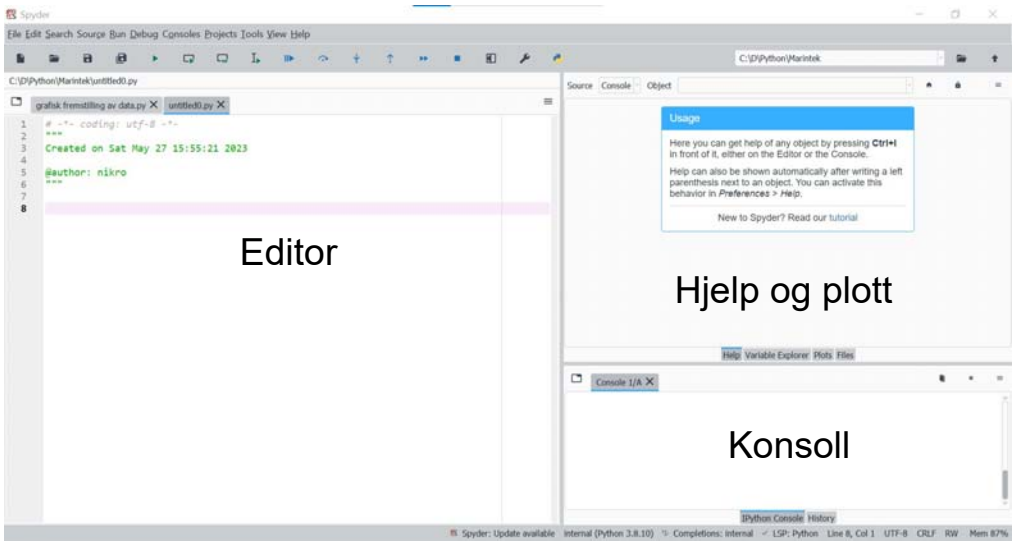
6. Dette avsnittet bygger på arbeidet til Thor Inge Hansen ved Skien videregående skole som har utviklet presentasjon og analyseverktøyet.
7. Finn Aakre Haugen, Markus Lysaker, *Python for realfag*, Fagbokforlaget 2020

For å installere Python-pakken går man til hjemmesiden til firmaet Anaconda og laster ned og installerer:

https://www.anaconda.com/download/

I tillegg til Spyder som er Python editoren, får man med flere bibliotekspakker og andre mer eller mindre nyttige programmer.

Når vi starter Spyder får vi opp en side som ligner på den som er vist i figuren under, med tre felter eller vinduer med ulike funksjoner: Editor, hjelp og plott, og et konsoll som viser responsen som programmet gir, inkludert advarsler og feilmeldinger.




I "Editoren" skriver vi inn kommandoene, evt. programkoden vi ønsker å kjøre. I vinduet "Hjelp og plott" kan vi få tips og hint, evt. oppsøke hjelpefunksjoner, vi kan vise innhold i variabler, vise plott og utskrifter eller oversikt over filer. I vinduet "Konsoll" får vi også svar fra programmet, feilmeldinger, advarsler ("warnings") og kommandohistorien.

I stedet for å liste opp alle mulighetene til Spyder så skal vi gjennomgå et aktuelt programeksempel, men først la oss se litt på et typisk fileformat for dataene våre slik de kan framstå etter at de er lagt ned på filen.

I dette eksempelet legger vi kretsen i dvale mellom hver måling.

### 3.4.2 Organisering av data i filen

La oss ta utgangspunkt i måledata organisert linje for linje i en file. Vi tenker oss at vi ønsker å gjøre målinger til gitte tidspunkter, f.eks. en gang hvert 15. minutt. Når vi først gjør en måling ønsker vi å måle flere ulike parametere samtidig. Det kan være et navn som identifiserer dataene, posisjon der målingene ble gjort bestemt ved hjelp av GPS, tidspunktet, temperatur, spenningen

på batteriet, med mer. Dataene legges etter hverandre på en linje med komme mellom. Når så neste måling skal gjøres, legges de på neste linje slik at det etter hvert dannes en tabell av data som vist i eksempelet under.

```
SeptKursTest_2.txt,1685052745,63.426197,10.453953,7,0.0,20.2,0.0,0.0,0.00,4.02,115088
SeptKursTest_2.txt,1685053136,63.426189,10.453951,3,0.0,20.1,0.0,0.0,0.00,4.01,72369
SeptKursTest_2.txt,1685053538,63.426369,10.454008,4,0.0,20.0,0.0,0.0,0.00,4.00,84543
SeptKursTest_2.txt,1685053955,63.426304,10.454157,7,0.0,19.9,0.0,0.0,0.00,4.05,100712
SeptKursTest_2.txt,1685054342,63.426220,10.453920,4,0.0,19.8,0.0,0.0,0.00,4.08,68519
```

I dette eksempelet slås mikrokontrollen og sensorene helt av mellom målingene, dermed kan vi ikke legge inn en tekstlinje øverst i fila ved programstart. Lengst til venstre ser vi filnavnet som identifiserer måleserien vår, deretter følger epoketiden, bredde- og lengdegrad og antall satellitter. Så følger to åpne felter som vi ennå ikke har tatt i bruk, deretter temperaturmålingen. Så kommer tre tomme verdier før vi møter batterispenningen. Siste verdi på hver linje er antall millisekunder det går fra programmet starter og til dataene er sendt og lagt ned på serveren (denne fila) og spenningen så slås av. Vi ser at sistnevnte variere en god del.

Komma er et greit skilletegn når vi skal bruke Python til å analysere dataene siden desimaltegnet er et punkt. Slike filer kalles CSV-filer (Comma Separated Values).

Det kan også være greit å legge en header over hver kolonne som forteller hvilke data det er snakk om. Det er ikke alltid så lett dersom vi starter det samme programmet om og om igjen for hver måling slik vi gjør.

La oss nå se hvordan vi kan hente utvalgte måledata inn i Python for presentasjon.

### 3.4.3 Lesing og presentasjon av data fra file[8]

La oss se på et grunnleggende eksempel. Eksempelet leser data fra filen vist i forrige avsnitt. Eksempelet burde være enkelt å modifisere og utvide. Programeksempelet finnes i sin helhet i vedlegg B.1 side 72.

Her går vi gjennom programmet del for del slik at det skal være mulig å forstå kommandolinjene og dermed være lettere å gjøre endringer og bygge ut programmet.

```
import numpy as np
import matplotlib.pyplot as plt
import datetime as dt
import pandas as pd
```

Først importeres diverse biblioteker som brukes i forbindelse med utregningene og presentasjonene mm. `numpy` er et bibliotek som håndterer arrayer av data, `matplotlib.pyplot` en modul av et plottebibliotek, `datetime` et bibliotek som i dette tilfellet konverterer epoketid til tid og dato og `pandas` er et kraftig bibliotek for dataanalyse.

8. Programmet er skrevet av Thor Inge Hansen ved Skien videregående skole

I Python er det dessuten vanlig å lage forkortelser av biblioteksnavnene slik at det skal være lettere å hente ut funksjoner fra bibliotekene. `numpy` forkortes f.eks. np (`numpy as np`) og `matplotlib.pyplot` forkortes plt (`matplotlib.pyplot as plt`) osv.

```
#%% Valg av tidsvindu for plotting av data
vintertid = False
```

De neste to linjene setter variabelen `vintertid` til verdien `False`, siden det når dette programmet ble brukt var sommertid. Vi skal ganske snart bruk denne variabelen i omregningen av *epoketid* til dato og klokkeslett. Programlinjer som begynner med # er kommentarer og vil ikke bli prosessert av programmet.

**Henting og strukturering av data fra file på serveren**

```
#%% import og laging av arrays

#Henter inn fila som en dataframe i pandas
data = pd.read_csv('http://sensor.marin.ntnu.no/logs/SeptKursTest_2.txt',
delimiter=',')
```

Måleverdiene hentes ut av datafila ved hjelp av biblioteksfunksjonen `pd.read_csv` fra pandas-biblioteket (pd.). Biblioteksfunksjonen har som vi ser, to argumenter. Det første er *nettstedet*: `http://sensor.marin.ntnu.no/logs/` i fila `SeptKursTest_2.txt`. og det andre argumentet angir skilletegnet mellom verdiene, i dette tilfellet er det ",". Verdiene legges inn i dataarrayet `data`.

```
epochTime = data["1685052745"].values
volt = data["4.02"].values
temp = data['20.2'].values
runTime = data['115088'].values/1000 #gjør ms om til s
```

Det neste som gjøres er å plukke ut verdiene i utvalgte kolonner og legge disse i egne array eller lister som har navnene `epochTime`, `volt`, `temp` og `runTime`. Disse henter ut samtlige verdier i de angitte kolonnene. Siden vi ikke har et kolonne-navn, bruker vi for enkelhets skyld den første dataverdien i hver kolonnen for å angi kolonnen. "epochTime" er tidspunktet dataene er hentet inn, her som antall sekunder etter 01.01.1980. "volt" er den målte batterispenningen, "temp" er den målte temperaturen (senere vanntemperaturen) og "runTime" er tiden det tar fra spenningen på mikrokontrolleren slås på til målingen er gjennomført, data er lagt ned på serveren og spenningen på mikrokontrolleren slås av. "runTime" er derfor nyttig for å beregne den totale på-tiden og således forholdet mellom på- og av-tiden gjennom måleserien.

**Omregning fra epoketid til dato og tid**

```
#%% Funksjon for å lage datetime-objeter fra epochTime

def epochToDatetime(x):
    liste=[]
    for i in range(len(x)):
        if vintertid:
            t=dt.datetime.utcfromtimestamp(x[i]) + dt.timedelta(hours=1)
```

```
            else:
                t = dt.datetime.utcfromtimestamp(x[i]) + dt.timedelta(hours=2)
            liste.append(t)
        return liste
```

Så defineres funksjonen `epochToDatetime(x)` med argumentet "x". "x" er her variabelen eller arrayet med epoketider som vi ønsker gjøre om til "år", "dato", "timer", "minutter" og "sekunder". I starten av funksjonen lager vi oss en åpen liste (liste), dvs. en liste med åpent antall elementer, som inneholder dato og tid. For å gjøre denne omregningen benyttes biblioteket `datetime` (dt.) og funksjonen `dt.datetime.utcfromtimestamp(x[i])`. Siden dette er UTC (Greenwich Mean Time) tid må vi legge til henholdsvis 1 eller 2 timer avhengig om det er sommer- eller vintertid, til dette bruker vi en if-setning og tester på variabelen vintertid (`if vintertid:`). Variabelen "t" holder enkeltverdier og legges til lista (`liste.append(t)`) etter som enkeltverdiene beregnes i for-loopen (`for i in range(len(x)):`). For å vite antall runder i for-loopen, finner vi lengden av variabelen x (`len(x)`). Tilslutt returneres den komplette lista fra funksjonen.

```
#%% Anvender funksjonen ovenfor

timelist = epochToDatetime(epochTime)
```

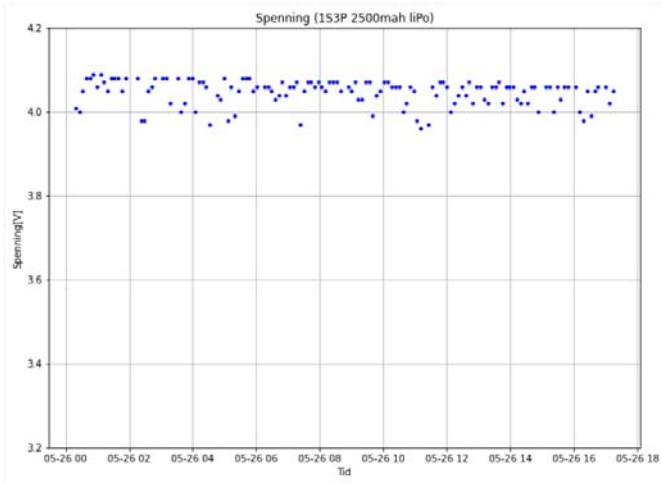
Vi anvender så den nylig definerte funksjonen på arrayet av epoketidsmålinger og får ut en liste over tidspunkter i en hensiktsmessig form.

**Plotting av data**

Vi er nå klare for plotting av dataene våre. Vi vil i første omgang lage fire plott.

Først plotter vi spenningsnivået som funksjon av tiden.

Her ser vi hvordan spenningen varierer med ca. 0,1 V fra måling til måling over et tidsintervall på ca. 18 timer den 26. mai 2023.




Programbiten under viser hvordan dette plottet framkommer:

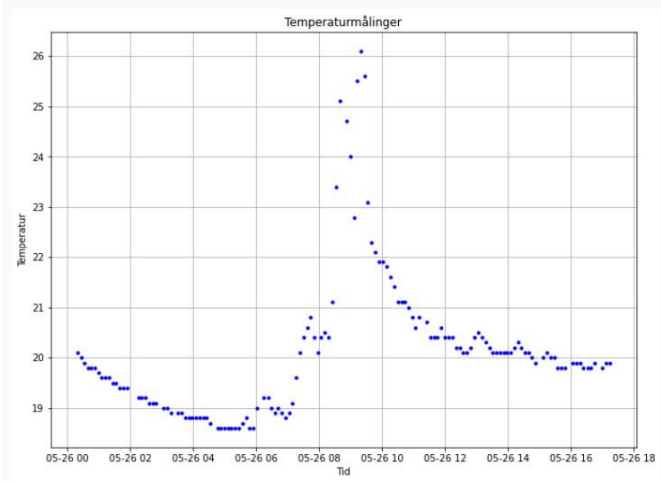
```
plt.figure(1,figsize=(11,8)) # Angir figurnummer og figur størrelse (cm)
plt.plot(timelist,volt,'b.') # Plotter spenning som funk. av tid med blå .
```

```
plt.ylabel('Spenning[V]') # Angir tekst på den vertikale aksen
plt.xlabel('Tid') # Angir tekst på den horisontale aksen
plt.title('Spenning (1S3P 2500mah liPo)') # Angir figuroverskrift
plt.ylim(3.2,4.2) # Angir skalaen på den vertikale aksen
plt.grid() # Angir at det skal tegnes opp et rutenett
```

I `plt.figure` vil normalt figuren bli angitt i inch, men dersom den generelle måleenheten i programmet er satt til cm er det denne som gjelder. I `plt.plot` vil måleverdiene angis med blå punkter `'b.'`.

Dernest plotter vi måleverdiene for temperatur som funksjon av tiden.

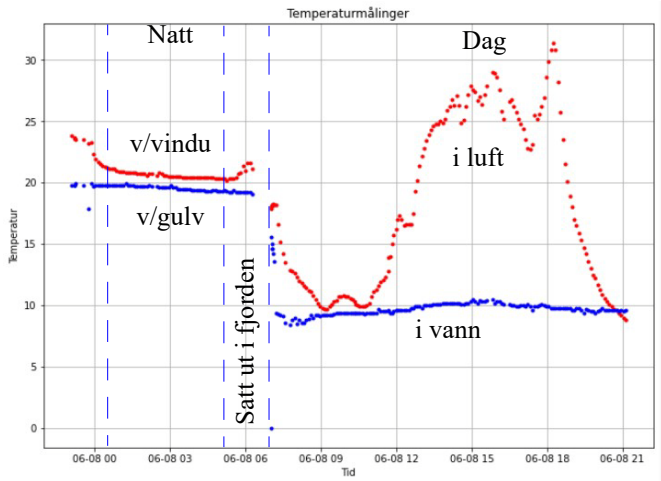
Her ser vi hvordan temperaturen endres over et tidsintervall på ca. 18 timer den 26. mai 2023. Sola skinner inn gjennom vinduet fra kl. 07:00 og fram til ca. kl. 09:00.




Her er det ingen overraskelser mht. programmeringen:

```
plt.figure(2,figsize=(11,8))
plt.plot(timelist,temp,'b.')
plt.xlabel('Tid')
plt.ylabel('Temperatur')
plt.title('Temperaturmålinger')
plt.grid()
```

Dersom vi ønsker å plotte flere kurver i samme diagram som vist i figuren til høyre, føyer vi til en ekstra linje med plt.plot().

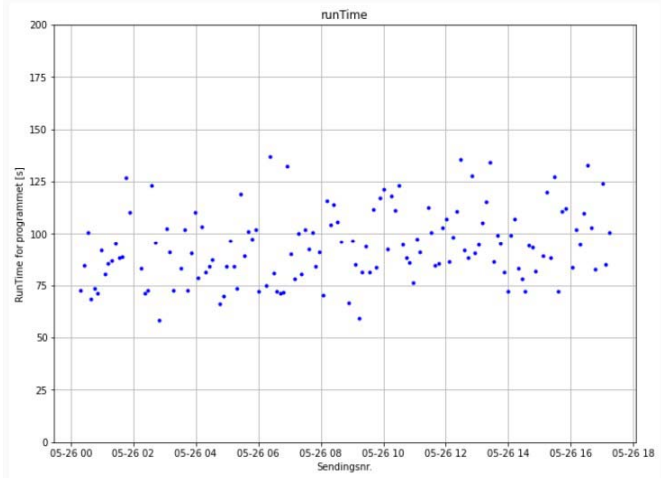



Her har vi plottet temperaturen i vann (blå kurve) og lufttemperaturen (rød kurve).

Vi legger merke til at de to sensorene ikke er helt samstemte der begge er innendørs. Den ene er plassert ved gulvet den andre foran vinduet.

Når de plasseres ut i fjorden så ser vi at temperaturen i vann er ganske stabil mens temperaturen i lufta i instrumentboksen, varierer mye.

Til slutt plotter vi *på*-tiden som funksjon av tiden.




Her ser vi hvordan på-tiden varierer over et tidsintervall på ca. 18 timer den 26. mai 2023. Vi ser at på-tiden varierer fra ca. 60 til 140 sekunder.

Her er heller ingen overraskelser mht. programmeringen:

```
plt.figure(3,figsize=(11,8))
plt.plot(timelist,runTime,'b.')
plt.ylim(0,200)
plt.title('runTime')
plt.xlabel('Sendingsnr.')
plt.ylabel('RunTime for programmet [s]')
plt.grid()
```

### 3.4.4 Analyse av data med Python[9]

I dette avsnittet skal vi se hvordan vi kan gjøre enkle beregninger på måledataene.

**Beregning av middelverdier**

I sin enkleste form er en middelverdi summen av alle de aktuelle verdiene delt på antallet. La oss se på et aktuelt eksempel, nemlig middelverdien av avstanden mellom målingene gjort i en måleserie. Vi tar da utgangspunkt i epoketiden nedtegnet for hver måling.

```
#%% Regning av gjennomsnittlig intervall

for i in range(len(epochTime)-1):
        interval[i]=int((epochTime[i+1]-epochTime[i]))

interval_mean=np.mean(interval[0:len(interval)-1])
```

Alle epoketidspunktene ligger lagret i arrayet `epochTime`. Funksjonen `len(epochTime)` bestemmer antall elementer i arrayet som utgjør én mer enn antall runder i for-loopen fordi antall intervaller er én mindre enn antall elementer. Dernest beregnes alle intervallene (`epochTime[i+1]-epochTime[i]`) som samles i arrayet: `interval`. Tilslutt beregnes middelverdien av alle elementene i arrayet ved hjelp av funksjonen `np.mean()`. Vi legger merke til at elementene i arrayet som middelverdien beregnes ut fra, er spesifisert som: `[0:len(interval)-1],` hvilket burde være unødvendig så lenge som alle verdiene er tatt med i beregningen.

*Kommentar:* Dersom målinger uteblir fra måleserien vil den beregnede middelverdien øke i forhold til det nominelle intervallet mellom målingene. Jo flere målinger som uteblir jo større blir avviket fra den nominelle verdien.

**Telling av avvikende målinger**

En måte å unngå problemet nevnt under kommentaren over, er å forsøke å finne antallet måleintervaller som overskrider en gitt verdi og dermed gir et bedre grunnlag for å beregne en nominell verdi.

```
#%% Finner antall ganger intervall mellom sending er over 150 s
count = 1

for i in range (len(interval)):
    if interval[i]>150:  #150sek = 2,5 min = feilsending
        count+=1
```

Her telles de gangene intervallet overskrider 150 sekunder (2,5 min). Denne grensen må selvfølgelig settes i henhold til måleintervallet satt i programmet.

*Kommentar:* Med kjennskap til antall manglende målinger vil man kunne korrigere middelverdien og komme nærmere det nominelle intervallet mellom målinger. Imidlertid vil ikke metoden korrigere for to påfølgende manglende målinger. For også å kunne korrigere for slike feil må man telle opp antallet av to påfølgende manglende meldinger, osv.

9. Dette er egentlig en gjennomgang av arbeidet gjort av Thor Inge Hansen ved Skien vgs

**Estimat av batteritid**

Et nyttig estimat er hvor lenge batteriene vil vare før målestasjonen kobler ned. Dette kan gjøres ut fra en måleserie som går over noe tid. Dersom vi studerer målingene som er vist på figuren på side 49, så vil vi se at spenningen faller jevnt over tid, i tillegg ser vi at spenningen har en del støy. Dersom vi antar en nedre grense for spenningen når den slutter å måle, kan vi ved hjelp av regresjon estimere på hvilket tidspunkt spenningen har nådd ned til denne grensespenningen.

```
#%% Lineær regresjon med beregning av total tid ned til min_volt
min_volt=3.2

epochLin = np.array(timeStamp).reshape((-1, 1))
model = LinearRegression().fit(epochLin, volt)

a = model.coef_
b = model.intercept_

volt_lin_reg=a*timeStamp+b

epoch_end=(min_volt-b)/a
duration=(int(epoch_end)-int(timeStamp[0]))/(3600*24)
```

Her gjøres en lineær regresjon ut fra innledende målinger gjort over ca. 18 timer. Nedre grense for spenningen der man regner med at utstyret slutter å virke er satt til 3,2 V (`min_volt`).

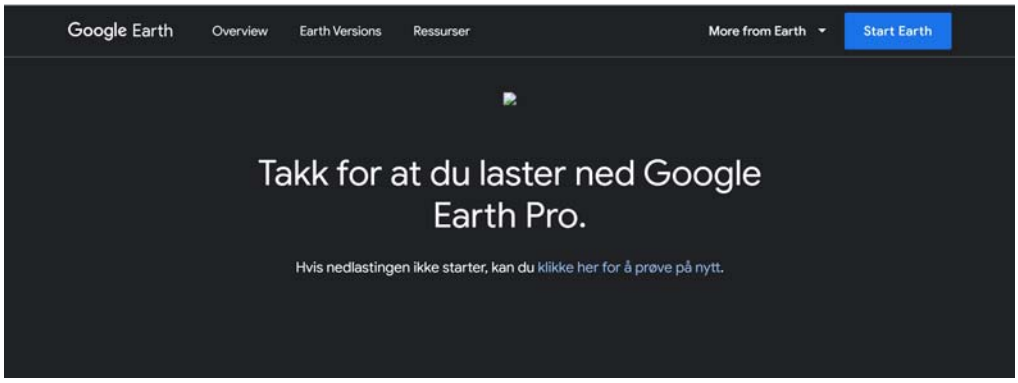
Det første man gjør er å linearisere måletidspunktene, dvs. gi dem lik avstand (`reshape((-1, 1))`). Dernest gjøres en lineær regresjon på alle måleverdiene av spenningen innen det tilgjengelige intervallet (`LinearRegression().fit(epochLin, volt)`). Dette resulterer i to parametere (a og b) som er henholdsvis stigningstallet og skjæringspunktet med y-aksen, som gjør det mulig å lage en lineær funksjon som estimerer forløpet av batterispenningen (`volt_lin_reg=a*timeStamp+b`), hvilket gjør det mulig å beregne tidspunktet når spenningskurven når ned til grensespenningen på min_volt (3,2V) som her er angitt som *duration*, som oppgis i dager.

*Kommentar:* Nå er det slett ikke sikkert at batterispenningen avtar som en lineær funksjon mellom full spenning og nedre grense på 3,2V.

## 3.5 Bruk av Google Earth for visning av posisjon fra GPS

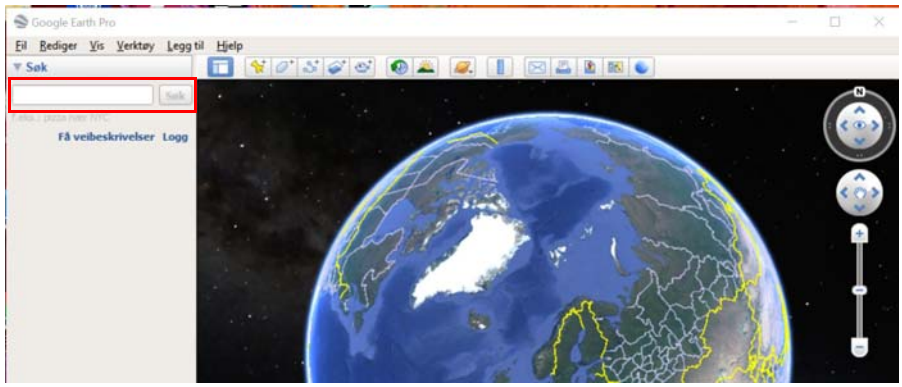
I dette avsnittet skal vi gi en oppskrift for hvordan vi kan vise en posisjon eller en trase i Google Earth.

Google Earth kan lastes ned og installeres fra følgende adresse: https://www.google.com/earth/versions/download-thank-you/




### 3.5.1 Plotting av en enkeltposisjon

Når man starter Google Earth får man opp følgende vindu.



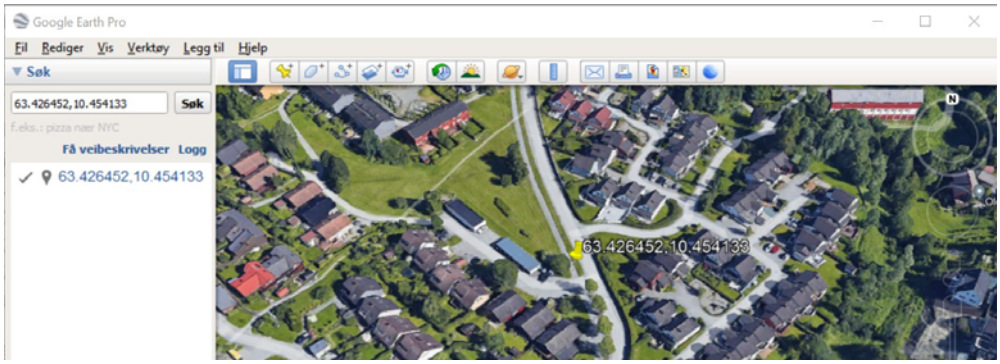

Ved å skrive inn bredde- og lengdegrader i innboksen øverst til venstre i vinduet, vil man kunne "forflytte seg" til det angitte stedet på kloden.

Lengde- og breddegrader legges inn som vist under:

63.426452,10.454133
Breddegrader[°],Lengdegrader[°]

Her er selvfølgelig rekkefølgen viktig.

Legg merke til at koordinatene minst bør ha med 4 siffer etter komma, gjerne 5 eller 6, høyde godtas ikke.




### 3.5.2 Plotting av en trase i Google Earth

Dersom man ønsker å plotte en trase i Google Earth kan man benytte et programmeringsspråk kalt "Keyhole Markup Language" (KML) som er utviklet for visualisering av to- og tredimensjonale strukturer knyttet til kartdata.

Siden språket er temmelig omfattende og vi kun trenger å bruke en beskjeden del av det, så benytter vi en ferdige programkode og klipper inn våre data for lengde-, breddegrad og høyde. Disse legges inn som en liste med data i kml-koden (se under), før kml-fila lagres med et ønsket navn.

Koordinater og høyde data legges inn som vist under:

-112.2550785337791,36.07954952145647,2357
Lengdegrader[°],Breddegrader [°],Høyde [m]

Legg merke til at rekkefølgen på koordinatene er omvendt av det vi la inn i innboksen på forsiden av Google Earth.

Programkode skrevet i kml (legg merke til at et sett med eksempelkoordinater og høyde er klippet inn – rød kode). Her er det lagt inn 11 posisjoner. Du må gjerne legge inn flere eller færre.

```
<?xml version="1.0" encoding="UTF-8"?>
<kml xmlns="http://www.opengis.net/kml/2.2">
  <Document>
    <name>Paths</name>
    <description>Examples of paths. Note that the tessellate tag is by default
      set to 0. If you want to create tessellated lines, they must be authored
      (or edited) directly in KML.</description>
    <Style id="yellowLineGreenPoly">
      <LineStyle>
        <color>7f00ffff</color>
        <width>4</width>
```

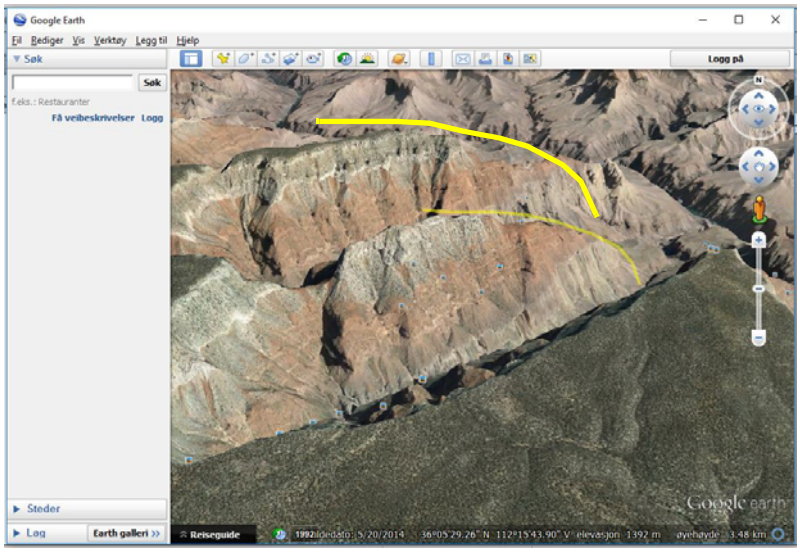
```
    </LineStyle>
    <PolyStyle>
      <color>7f00ff00</color>
    </PolyStyle>
  </Style>
  <Placemark>
    <name>Absolute Extruded</name>
    <description>Transparent green wall with yellow outlines</description>
    <styleUrl>#yellowLineGreenPoly</styleUrl>
    <LineString>
      <extrude>0</extrude>
      <tessellate>0</tessellate>
      <altitudeMode>absolute</altitudeMode>
      <coordinates>
        -112.2550785337791,36.07954952145647,2357
        -112.2549277039738,36.08117083492122,2357
        -112.2552505069063,36.08260761307279,2357
        -112.2564540158376,36.08395660588506,2357
        -112.2580238976449,36.08511401044813,2357
        -112.2595218489022,36.08584355239394,2357
        -112.2608216347552,36.08612634548589,2357
        -112.262073428656,36.08626019085147,2357
        -112.2633204928495,36.08621519860091,2357
        -112.2644963846444,36.08627897945274,2357
        -112.2656969554589,36.08649599090644,2357
      </coordinates>
    </LineString>
  </Placemark>
 </Document>
</kml>
```

De koordinatene som pr. i dag ligger i eksempelet, angir en helikopterflyvning over Grand Canyon i Colorado, USA.

Eksempeldataene byttes ut med de aktuelle dataene ***og kodefila lagres under et ønsket filnavn som ender med .kml.***

Filen hentes opp i Google Earth ved å dobbelklikke på filnavnet. Siden fila ender på .kml, så skal den automatisk bli gjenkjent av Google Earth, starte programmet og laste inn datafilen.

En vil da få tegnet inn traseen som angitt av lista med koordinater og vist på riktig sted på jorda. Eksempelet under viser traseen til helikopteret over Grand Canyon (gul linje).



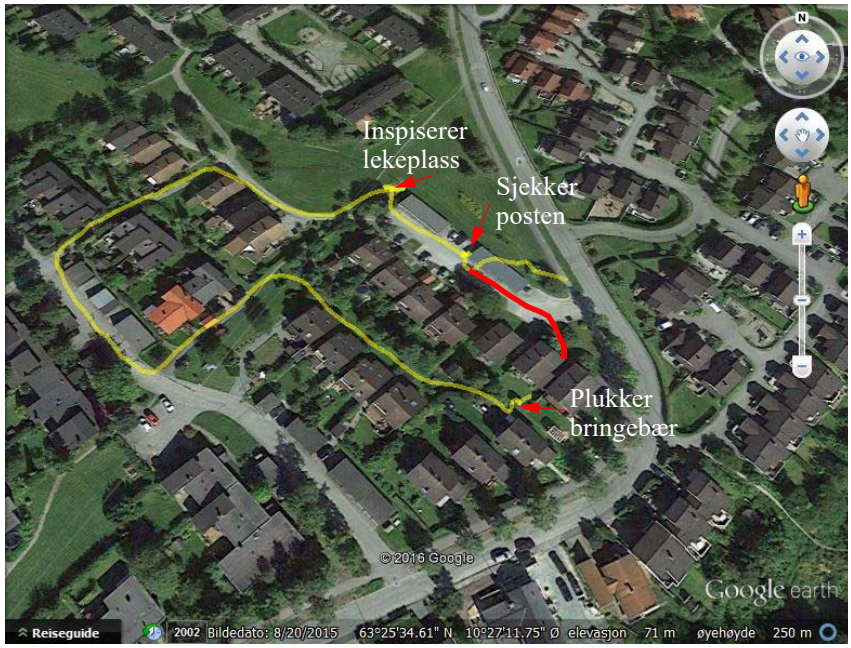

Dersom man ønsker å vise hvor man har gått en tur, så kan det være upraktisk å måtte vise absolutt høyde. Høydemålinger kan ha store avvik slik at en kan oppleve at traseen går mange meter over eller forsvinner under bakken, til tross for at man hadde begge beina på jorda. I slike situasjoner kan det være greit å bytte ut kommandoen:

```
<altitudeMode>absolute</altitudeMode>
```

med kommandoen:

```
<altitudeMode>clampToGround</altitudeMode>
```

så vil kurven følge bakken som vist på traseen på bildet under.




Vi legger imidlertid merke til at mottakeren har problemer i starten. I dette området er avviket stort før den tar seg inn. Den røde kurven angir den riktige ruta. Deretter er den svært nøyaktig. Posisjonsdata samples hvert 3. sekund på turen.

### 3.5.3 Editering av kml-fila

Hvilket program skal man så bruke for å legge inn koordinater og høyde? Et nyttig editeringsverktøy til dette formålet er Notepad++. Dette programmet er en avansert teksteditor som ikke gjør noen endringer med filformatet såfremt man ikke ønsker det. Notepad++ kan lastes ned fra:

https://notepad-plus-plus.org/downloads/

For å forenkle prosessen med å klippe inn data i kml-koden, er det praktisk å skrive koordinat- og høydedata i det formatet som programmet ønsker: Lengdegrad, breddegrad, høyde. Husk å bruke punktum som desimalpunkt og komma mellom hver verdi. ***NB! Det skal ikke være mellomrom etter kommaene.***

# Vedlegg A Gjennomgang av eksempelprogrammet

Vedlegget skal være en hjelp til å forstå koden til eksempelprogrammet slik at man er istand til å gjøre personlige justeringer.

### 3.5.4 Detaljert gjennomgang av programkoden

Vi skal nå gå gjennom eksempelprogrammet og forklare de ulike delene.

**Inkludering av biblioteker**

Følgende biblioteker trengs:

```
#include <DS18B20.h>        // Bibliotek for avlesning av temperatursensoren
#include <OneWire.h>        // Bibliotek som etablerer entrådsbuss til temp. sensoren
#include <Arduino_MKRGPS.h> // Bibliotek som leser av GPS
#include <MKRNB.h>          // Bibliotek som overførerdata til server via GPRS
#include <Arduino_MKRENV.h> // Bibliotek for lesing av miljøkortet ENV om det brukes
#include "ArduinoLowPower.h"// Bibliotek som legger MKR i dvale
#include <WDTZero.h>        // Bibliotek som håndterer vakthunden
#include <SD.h>             // Bibliotek for håndtering av SD-kort
#include <SPI.h>            // Bibilotek for håndtering av serie-bus
```

Bibliotekenes header-filer (.h) plasseres normalt i toppen av programmet. Dersom man ønsker å legge dataene inn på et SD-kort, så finner man en slik terminal både på miljø-kortet (MKR ENV) og på Prototyp-kortet (MKR SD Proto Shield). Bruker man miljø-kortet må man ta med biblioteket for dette kortet i tillegg til at man må initialisere det i setup()-funksjonen, se under setup()-funksjonen.

**Deklarasjon av objekter**

```
WDTZero MyWatchDoggy; // Deklarerer instansen til vakthunden MyWatchDoggy av typen WDTZero
NBClient client;      // Deklarerer instansen som opererer mot serveren
GPRS gprs;            // Deklarerer instansen som opprettet kontakt med GPRS-nettet
NB nbAccess;          // Deklarerer instansen som som gir tilgang til GPRS-nettet
DS18B20 ds(0);        // Deklarerer instansen som kommuniserer med temp. sensoren
File myFile;          // Deklarerer et objekt for filbehandling, kun ved lagring på SD-kort
```

*Objekter* er sentrale i C++ som er et objektorientert språk. Et objekt eller *instans* er en deklarasjon på linje med deklarasjon av variabler. Mens en variabel gjerne inneholder en verdi eller et array av verdier av samme type, så kan et objekt inneholde et pakke med ulike variabler av ulike typer. Ja, ikke bare variabler med også ulike *funksjoner*. Når vi deklarerer våre objekter gir vi et *navn* til en *pakke* av en bestemt type. F.eks. så deklarerer vi et objekt som *vi* har kalt MyWatchDoggy av typen WDTZero. For å nå de ulike elementene i objektet så bruker vi *vårt* pakkenavnet, punkt og navnet på variabelen eller funksjonen f.eks. *MyWatchDoggy.setup(WDT_SOFTCYCLE1M);*

**Deklarasjon av konstanter**

I motsetning til variabler som kan endre innhold mens programmet kjører, så må konstantene beholde sitt innhold gjennom hele programmet. De kan bare endres i selve programkoden.

```
#define DEBUG          // Kommenter bort denne dersom Serial.print ikke skal inkluderes
#define LED            // Kommenter bort denne dersom LED indikasjon ikke ønskes
```

Konstanter plasseres gjerne før setup()-funksjonen og vil derfor være globale, dvs. kunne brukes i alle funksjoner. Konstantene DEBUG og LED har spesielle funksjoner. Avhengig av om vi deklarerer hver av disse to så påvirker dette hva som kan vises i monitoren. Vi kommer snart tilbake til dette.

### Deklarasjon av variabler

Variabler kan endre verdi mens programmet kjører. Dersom de defineres utenfor alle funksjoner blir de globale og kan brukes i alle funksjoner. Vi har her valgt å gruppere dem i henhold til i hvilken sammenheng de brukes i:

```
// Temperatursensor
uint8_t address[] = {40, 250, 31, 218, 4, 0, 0, 52};
uint8_t selected;

// Oppkobling til server
char server[] = "sensor.marin.ntnu.no";  // Navnet på serveren der data legges
char path[] = "/cgi-bin/tof.cgi?";       // Angir cgi-kode for plassering i file
char filename[]="NTNU-1.txt";            // Navnet på fil for lagring av data
int port = 80;                           // Port 80 er default for HTTP
boolean connected = false;               // Status oppkobling til GPRS
char buffer[128];                        // Deklarsjon av buffer med tilhørende lengde

volatile unsigned long num = 0;    // Måling nummer
float w_temperature        = 0;    // Vanntemperatur
float BatVolt              = 0;    // Variabel for å holde batterispenningen
int Pause                  = 5000; // En evt. pause i programloopen
int SleepTime              = 5000; // Tid for dvale

float GPSlat               = 0;    // GPS breddegrad
float GPSlong              = 0;    // GPS lengdegrad
unsigned long epochTime    = 0;    // GPS tidsstempel fra 1.1.1980
int numSat                 = 0;    // GPS antall satellitter
int maxNoChecks            = 5000; // Antall runder for lesing av GPS-data
```

Disse deklarasjonene plasseres gjerne før setup()-funksjonen.

### Deklarasjon av porter

Her kan vi sette navn på de portene vi ønsker å bruke.

```
int BatVoltPin = A3;             // Analog port for tilkobling av batterispenning
const int SD_CS_PIN = 4;         // Port for styring av SD-kort terminal, fabrikkbestemt
```

Port 4 er fabrikk-koblet til SD-kort-terminalen på shield-kortene så denne kan ikke endres.

### Setup()-funksjonen

Som kjent kjøres denne funksjonen kun ved oppstart og inkluderer initiering av de ulike objektene og lignende:

```
void setup() {

  pinMode(LED_BUILTIN, OUTPUT);  // Port for innebygget LED, definert som utgang
  analogReadResolution(12);      // Angir oppløsning for AD-konverter

  #ifdef DEBUG
     Serial.begin(115200);       // Setter opp data hastighet til monitor
  #endif

  if (!GPS.begin())              // Initialisering av GPS
  {
   #ifdef DEBUG
     Serial.println("GPS initialisering mislyktes");
   #endif
  }
  SPI.begin();
  ENV.begin();

  if(!SD.begin(SD_CS_PIN))
  {
    Serial.println("Initialisering SD terminal mislyktes");
  }

  // Lag en file med navnet DataFile.txt
  myFile = SD.open("DataFile.txt", FILE_WRITE);
  delay(1000);
  myFile.close();
  delay(100);

  // Initialiserer dvale-funksjon
  LowPower.attachInterruptWakeup(RTC_ALARM_WAKEUP, dummy, CHANGE);

  // Initialisering av "vakthund"
  MyWatchDoggy.attachShutdown(myshutdown); // Starter funksjonen myshutdown() ved reset
  MyWatchDoggy.setup(WDT_SOFTCYCLE1M);     // Setter intervalltiden for "vakthunden"

  // Opprett tilkobling til 4G - GPRS-tjeneste
  connectToGPRS();

  delay(1000);
}
```

Som vi ser kalles det en rekke funksjoner som initialiserer eller setter parametere for de funksjonene vi senere skal bruke i programmet. Kommandoen `analogReadResolution(12);` setter f.eks. oppløsningen til AD-konverteren til 12 bit. Om denne utelates vil oppløsningen automatisk settes til 10 bit.

Ellers initialiseres GPS-mottakeren (`GPS.begin()`), SPI-bussen (`SPI.begin()`) som kommuniserer med SD-kortet, miljø-kortet (MKR ENV) dersom dette brukes som SD-kortterminal (`ENV.begin()`), selve SD-kortterminalen (`SD.begin(SD_CS_PIN)`), og det opprettes en fil hvor dataene skal legges dersom man lagrer dataene på SD-kort (`SD.open("DataFile.txt", FILE_WRITE);`).

Vi legger spesielt merke til følgende programlinjer som vi finner igjen flere steder i programmet:

```
#ifdef DEBUG
  Serial.println("GPS initialisering mislyktes");
#endif
```

Dette er en måte å gi beskjed til kompilatoren om utskrifter ala `Serial.println ...` ønskes tatt med i programmet eller ikke. Dersom konstanten `DEBUG` er definert under deklarasjon av konstanter, så blir programkoden mellom `#ifdef DEBUG ...` og `#endif` tatt med i programmet. Dersom `DEBUG` ikke er deklarert så utelates programlinja av kompilatoren. Dette har vi gjort da det er lite hensiktsmessig å forsøke å skrive til monitoren dersom bøya ligger i sjøen og dermed ikke kan kommunisere over USB-kabelen. Ja, det kan til og med se ut til at programmet har en tendens til å henge seg om det forsøker å skrive til monitoren uten at USB-kabelen er tilkoblet.

### Loop()-funksjonen

Som vi ser så inneholder loop()-funksjonen nesten bare kall av funksjoner, som enten ligger i et bibliotek eller som vi selv har definert under.

```
void loop() {

  //readGPSdata1();     // Inkluder GPS
  readWaterTemp();      // Inkluder måling av temp. i vann
  //readBatVolt();      // Inkluder måling av baterispenning om den er implementert

  makeString();         // Bygge opp bufferet for overføring av data
  sdPrint();            // Skriv data til SD-terminal
  connectToServer();    // Koble opp mot server og overfør data

  printData();          // Skriv data til monitoren
  printDataString();    // Skriv ut bufferet til monitoren
  MyWatchDoggy.clear(); // Resetter vakthunden

  num++;
  //GoToSleep();        // Legg microkontrolleren og GPS'en i dvale
  delay(Pause);         // Ev. legg inn en pause i loopen
}
```

På denne måten kan vi inkludere de funksjonene vi ønsker å teste ut og på den måten gradvis utvide funksjonaliteten til programmet. F.eks. har vi i dette tilfellet kommentert bort avlesning av batterispenningen (`//readBatVolt();`) og GPS-mottakeren (`//readGPSdata1();`), og foreløpig droppet å legge mikrokontrolleren i dvale (`//GoToSleep();`).

La oss se på de enkelte egendefinerte funksjonene:

### Egendefinerte funksjoner

#### *Funksjon for avlesning av vanntemperatur*

```
void readWaterTemp()
{
  w_temperature = ds.getTempC();
}
```

`ds.getTempC()` er en biblioteksfunksjon som leser av temperaturen. Verdien legges i variabelen `w_temperature.`

#### *Funksjon for avlesning av batterispenningen*

```
void readBatVolt()
{
  BatVolt = analogRead(ADC_BATTERY)*(4.25/4096);
}
```

Her leser vi av den analoge inngangen (ADC_BATTERY) og regner den om til en spenning. Inngangen er koblet til en intern spenningsdeler på mikrokontrollerkortet og vil bare fungere dersom vi har koblet til et batteri. Legg merke til at vi multipliserer den avleste verdien med 4.25. Årsaken er at forsøk har vist at denne verdien gir den riktigste verdien når vi sammenligner den internt målte verdien på pinne ADC_BATTERY, med målt spenning på batteriet med et voltmeter.

Vi kan også koble opp vår egen eksterne måling av spenningen på batteriet, i så fall må vi sørge for å bruke en spenningsdeler slik at vi unngår spenninger som overgår driftspenningen til mikrokontrolleren på 3,3V. Det kan være praktisk å bruke en spenningsdeler som halverer batterispenningen da et nyladet batteri kan ha en spenning på opp til 4,2V. Når målingen er utført så må vi multiplisere resultatet med to for å få riktig spenning som vist i programkoden under:

```
void readBatVolt()
{
  BatVolt = 2*analogRead(BatVoltPin)*3.3/4096;
}
```

Her har vi brukt en av de analoge inngangene hos mikrokontrolleren som vi har kalt `BatVoltPin`.

En analog-til-digitalkonverter (ADC) omformer spenningen til et digitalt tall. Med 12 bits ADC vil maksimalspenningen på 3,3V tilsvare tallet 4096. Vi finner da spenningen i Volt ved å dele måleverdien med 4096 og multiplisere med arbeidsspenningen på kortet, 3,3V.

### Funksjon for avlesning av GPS-mottaker

```
void readGPSdata1()
{
  int noChecks = 0;

  for (int i = 0; i < maxNoChecks; i++)
  {
```

```
    // Check if there is new GPS data available
    if (GPS.available())
    {
      // If there is, read GPS values
      GPSlat   = GPS.latitude(); // Avlesning av breddegrad
      GPSlong  = GPS.longitude();// Avlesning av lengdegrad
      epochTime  = GPS.getTime();// Avlesning av epoketid, ant. sek. siden 1.1.1980
      numSat = GPS.satellites(); // Kontakt med antall satellitter

      #ifdef DEBUG
        Serial.print("Antall GPS sjekk: ");
        Serial.println(noChecks);
      #endif

      break;
    }

    noChecks++;

    #ifdef DEBUG
     if (noChecks == maxNoChecks) Serial.println("Mislykket henting av posisjon fra GPS");
    #endif
  }
}
```

Legg merke til at avlesningen av GPS-mottakeren er lagt inn i en for()-loop med et maksimalt antall gjentakelser på `maxNoChecks`. Dette har vi gjort fordi det kan ta litt tid å etablere forbindelse med satellittene. Ved å legge inn et maksimalt antall forsøk på avlesning, unngår vi at programmet henger seg og forårsaker at "vakthunden" resetter mikrokontrolleren. I tillegg teller vi hvor mange runder den må gå for å få gyldige data, evt. varsler at maksimalt antall avlesninger er nådd slik at denne gangen er avlesningen mislykket. Deretter går programmet videre og forsøker på nytt i neste runde.

***Funksjon for oppbygging av databufferet***

```
void makeString()
{
   sprintf(buffer,"/cgi-bin/tof.cgi?%s,%u,%u,%.6f,%.6f,%i,%.1f,%.2f", file-
   name, num, epochTime, GPSlat, GPSlong, numSat, w_temperature, BatVolt);
}
```

Her bygges innholdet av bufferet opp som en tekststreng som består av måleverdier skilt med komma. *Det kan bemerkes at dersom vi planlegger å lese datafila med Python så kan det være greit å bruke komma*. For nærmere beskrivelse av oppbyggingen av bufferet se vedlegg A.1 side 69.

***Funksjon for utskrift av databufferet til monitoren***

```
void printDataString()
{
```

```
    #ifdef DEBUG
    Serial.println(buffer);
    Serial.println();
    #endif
  }
```

Vi legger merke til utskriften kun gjøres dersom DEBUG er definert. Utskriften blir lik den som skrives til datafila på serveren.

### *Funksjon for tilkobling til NB-IoT-nettet (4G GSM)*[10]

```
void connectToGPRS()
{
  while (!connected)
  {
   if ((nbAccess.begin("", true, true) == NB_READY) && (gprs.attachGPRS() == GPRS_READY))
    {
      connected = true;

      #ifdef DEBUG
        Serial.println("Vellykket tilkobling til GPRS");
      #endif

      #ifdef LED
        flash(1);
      #endif
    }
    else
    {
      #ifdef DEBUG
        Serial.println("Mislykket tilkobling til GPRS");
      #endif

      #ifdef LED
        flash(5);
      #endif

      break;
    }
  }
```

Essensen i denne funksjonen kan gjengis slik:

```
void connectToGPRS()
{
  while (!connected)
  {
   if ((nbAccess.begin("", true, true) == NB_READY) && (gprs.attachGPRS() == GPRS_READY))
```

10.https://github.com/arduino-libraries/MKRNB/blob/master/docs/api.md

```
    {
      connected = true;
    }
  {
}
```

resten er bare meldinger som forteller om det har lyktes å koble opp eller ei.

`nbAccess.begin(“”, true, true)` ber om tilgang til nettverket ved hjelp av SIM-kortets PIN-kode. Om slikt ikke brukes som hos oss, kan første argument stå åpent. Andre argument (true) handler om synkron (true) eller asynkron (false) modus, mens siste argument angir om radiomodemet skal restartes (true) eller ikke. Dersom den gis tilgang til nettet returnerer funksjonen NB_READY (alternativt kan en av følgende koder returneres: GPRS_READY, TRANSPARENT_CONNECTED, CONNECTING, IDLE og ERROR).

`gprs.attachGPRS()` ber om å bli koblet opp mot GPRS-nettverket. Oppkobling er gjennomført tilfredsstillende når funksjonen returnerer `GPRS_READY`.

Når begge disse er akseptert så settes variabelen `connected = true;` slik at man ved neste overføring av data, slipper å koble opp på nytt. Imidlertid kan det være nødvendig dersom mikrokontrolleren har ligget i dvale.

Dersom vi kjører med frakoblet USB-kabel og batteri, så vil meldingene til monitoren utebli. Vi har derfor laget en flash()-funksjon, hvor argumentet angir et antall blink.

```
#ifdef LED
  flash(1);
#endif
```

Ved at vi deklarerer konstanten LED så gir den innebygde LED’en på kortet fra seg ett eller flere blink som indikerer hvor i programmet den befinner seg, og om oppkoblingen var vellykket (1 blink) eller ikke (5 blink).

### *Funksjon for tilkobling til serveren ved Inst. marin teknikk*

Denne funksjonen foretar oppkobling og overføring av data til serveren.

```
void connectToServer()
{
  if (client.connect(server, port))
  {
    client.print("GET "); // Gjør et HTTP request:
    client.print(buffer);
    client.println(" HTTP/1.1");
    client.print("Host: ");
    client.println(server);
    client.println("Connection: close");
    client.println();

    #ifdef DEBUG
      Serial.print("Vellykket overføring til server av datasett nr.: ");
      Serial.println(num);
```

```
    #endif

    #ifdef LED
      flash(2);
    #endif
  }
  else
  {
    #ifdef DEBUG
      Serial.print("Mislykket overføring til server av datasett nr.: ");
      Serial.println(num);
    #endif

    #ifdef LED
      flash(10);
    #endif
  }
}
```

Essensen i denne funksjonen kan også gjengis i kortform, resten er bare meldinger om hvordan leveransen av dataene til serveren har gått:

```
void connectToServer()
{
  if (client.connect(server, port))
  {
    client.print("GET "); // Gjør et HTTP request:
    client.print(buffer);
    client.println(" HTTP/1.1");
    client.print("Host: ");
    client.println(server);
    client.println("Connection: close");
    client.println();
  }
}
```

`client.connect(server, port)` kobler opp mot serveren (`server`) med port nr. (`port`). Begge argumentene er gitt innhold under deklarasjonen av variabler:

```
char server[] = "sensor.marin.ntnu.no";  // Navnet på serveren der data legges
char path[] = "/cgi-bin/tof.cgi?";       // Angir cgi-kode for plassering i file
```

Dernest sendes en HTTP forespørsel til clienten (serveren): `GET <pathname> HTTP/1.1`. Denne er delt opp i tre print-kommandoer:

```
client.print("GET ");
client.print(buffer);
client.println(" HTTP/1.1");
```

der `buffer` også inneholder stien.

```
client.print("Host: ");
client.println(server);
client.println("Connection: close");
```

```
client.println();
```

Så angis server-navnet før forbindelsen lukkes, og kommandorekka avsluttes med en tom linje.

### *Funksjon for skriving til SD-kort*

I vårt eksempelprogram skrives bufferet til filen DataFile.txt på SD-kortet:

```
void sdPrint(){

  myFile = SD.open("DataFile.txt", FILE_WRITE);
  delay(1000);

  if (myFile)
   {
    myFile.println(buffer);
    myFile.close();
   }
}
```

Det er det samme bufferet som skrives til serveren.

### *Funksjon for å gi lysblink*

Som nevnt foran vil det være vanskelig å få skriftlige meldinger når mikrokontrolleren er frakoblet PC'en. Funksjonen flash() gir et antall lysblink i mikrokontrollerens innebygde lysdiode for å signalisere at den passerer ulike punkter i programmet. Når funksjonen kalles, vil man i argumentet kunne angi et antall blink.

```
void flash(int number)
{
  delay(1000);
  for(int i=0; i<number; i++)
  {
  digitalWrite(LED_BUILTIN, HIGH);
  delay(100);
  digitalWrite(LED_BUILTIN, LOW);
  delay(100);
  }
 delay(1000);
}
```

Vi har valgt å gi *ett* blink ved oppnådd kontakt med NB IoT-nettverket, *to* når det oppnås kontakt med serveren og *tre* når mikrokontrolleren legges i dvale. Det angis 5 og 10 blink ved mislykket oppkobling mot henholdsvis NB IoT-nettverket og serveren.

### *Funksjon som legger mikrokontrolleren i dvale*

Funksjonen `GoToSleep()` legger mikrokontrolleren i dvale for en viss tid (`SleepTime`). Man må regne med at radio forbindelsen brytes ved slik nedkobling.

```
void GoToSleep() // Legg mikrokontrolleren og GPS'en i dvale
```

```
{
  #ifdef DEBUG
    Serial.println("Går i dvale");
  #endif

  #ifdef LED
    flash(3);
  #endif

  GPS.standby();
  LowPower.deepSleep(SleepTime);
}
```

De to siste kommandoene `GSP.standby();` og `LowPower.deepSleep(SleepTime);` slår av GPS'en og legger mikrokontrolleren i dvale i tidsrommet `SleepTime`.

***Funksjon for retur fra dvale***

```
void dummy()
{
   GPS.wakeup();
   NVIC_SystemReset(); // Resett programmet
}
```

Idet mikrokontrolleren vekkes fra dvale, havner den inn i funksjonen `void dummy()`, der de første tiltakene etter oppvåkning kan gjøres. Her har vi valgt å vekke GPS'en (`GPS.wakeup();`) og resette programmet (`NVIC_SystemReset(); // Resett programmet`)[11].

***Funksjon for siste aksjon før "vakthunden" resetter programmet***

```
void myshutdown()
{
  #ifdef DEBUG
    Serial.println("Resetter MKR");
  #endif
}
```

Her nøyer vi oss med å varsle `("Resetter MKR")` om at mikrokontrolleren resettes av den interne "vakthunden".

## A.1 Bygg opp tekststreng for overføring av en måleserie

Vi skal nå bygge opp tekststrengen som beskriver formatet til en rad i datafilen som skal legges på serveren. Vi tar utgangspunkt tekststrengen i vårt eksempel:

11. https://devzone.nordicsemi.com/f/nordic-q-a/77006/what-does-nvic_systemreset-actually-do

```
sprintf(buffer,"/cgi-bin/tof.cgi?%s,%u,%u,%.6f,%.6f,%i,%.1f,%.2f", file-
name, num, epochTime, GPSlat, GPSlong, numSat, w_temperature, BatVolt);
```

La oss ta for oss hvert av leddene i eksempelet over:

`sprintf()` står for "string print format" og lager en tekststreng, satt sammen av tekst og tallverdier omgjort til tekst.

`buffer,` som er det første elementet i argumentet, angir variabelen der den endelige tekststrengen skal plasseres. Denne er tidligere deklarert som et buffer på inntil 128 karakterer (Byte), men som bør gjøres så kort som mulig. Selve tekststrengen er plassert i hermetegn: " " .

/`tof.cgi?` er et lite skript (program) som web-serveren bruker for å styre data på serveren ("cgi" står for *Common Gateway Interface*). I vårt tilfelle vil skriptet ta tak i det neste argumentet etter "?", som i dette eksempelet er "`%s,"`. Der %-tegnet kalles *formatspesifikasjonen* ("format specifier") og "s" *formatkarakteren* ("format character"). Lengre bak på linja ligger variabelen som skal anvende formatspesifikasjonen, i dette tilfellet strengen `filnavn`. Alternativt kunne vi ha skrevet filnavnet rett inn i tekststrengen.

Stedene i tekststrengen der vi ønsker å plassere tekst eller måleverdier, angis derfor med formatspesifikasjoner av ulike slag og som harmonerer med hver av variablene vi ønsker å legge inn. Skal vi legge til et desimaltall, så angis dette som f.eks. ".nf". Dvs. at ".6f" betyr at desimaltallet skal angis med 6 desimaler. "f" angir at det er snakk om et flyttall, "u" et heltall uten fortegn, "i" et heltall med fortegn, "d" et desimaltall og "s" at det handler om en tekststreng som vist under.

```
%s,%u,%u,%.6f,%.6f,%i,%.1f,%.2f
```

Etter hermetegnet kommer listen over variabler som skal settes inn i tekst-strengen på de angitte plassene. I vårt tilfelle er dette:

```
filename, num, epochTime, GPSlat, GPSlong, numSat, w_temperature, BatVolt
```

Man kan eventuelt sette inn rene tall (`67.5, 5.6`), eller tekst, men formatspesifikasjonen må stemme med typen til variabelen.

Vi har her separert leddene med ",". Grunnen er at det påstås at ved bruk av Python så vil lesingen bli enklere dersom vi bruker komma framfor f.eks. semikolon. Etter den første formatspesifikasjonen %s, skal det uansett være komma, siden %s på dette stedet indikerer filnavnet til fila der vi ønsker å legge dataene.

Deretter legges dataene fortløpende inn i tekststrengen i henhold til angitt format. En slik tekststreng utgjør en rad i fila. Siden dette blir en tekst-fil, bør vi føye til .txt. Dermed vil den bli lettere å åpne med en text-editor f.eks. notepad++.

Den ferdige tekststrengen legges i strengvariabelen: *buffer*. Deretter sendes innholdet av buffer til serveren med følgende kommando:

```
client.print(buffer);
```

Det er selvfølgelig ingen ting i veien for å bygge opp en lengre streng med flere variabler, en må bare passe på at man ikke går ut over den maksimale bufferlengden. 255 kan være en slik grense, men bør sjekkes[12].

12.https://www.sciencedirect.com/topics/computer-science/maximum-packet-size

Med tanke på at dataene skal kunne leses enkelt ved hjelp av Excel eller et Python-script er det lurt å droppe tekst mellom hver verdi. Strengen blir ikke så lett å lese, men egner seg godt for å ta inn i Excel eller Python.

# Vedlegg B Python-program for presentasjon av resultater

## B.1 Grunnleggende program for presentasjon av data[13]

Programmet viser hvordan Python kan brukes til å hente måleresultater fra en file, legge dataene inn i lister og plotte dem i grafiske diagrammer. Målefilen ligger på en server på Inst. for marin teknikk. I Python er innrykk viktig for at programmet skal fungere som tiltenkt, derfor er det viktig at dette blir riktig. Det beste er derfor å hente fila fra:

www.ntnu.no/skolelab/bla-hefteserie

Gå ned til overskriften: *Programmering av Tingenes internett (IoT)*
Åpne fanen: *EVU-kurs: Miljøovervåking med bøye*
Fila: *grafisk fremstilling av septtest_5.py*

```
import numpy as np
import numpy as np
import matplotlib.pyplot as plt
import datetime as dt
import pandas as pd

#%% Valg av tidsvindu for plotting av data

vintertid=False

#%% import og laging av arrays

#Henter inn fila som en dataframe i pandas
data = pd.read_csv('http://sensor.marin.ntnu.no/logs/SeptKursTest_5.txt',
delimiter=',')

# Tar ut de kolonnonene som jeg vil bruke fra data og lager egne arrays (numpy
lister) av dem data blir lagret som float
# Kolonnene hentes ut ved overskrift (første data i kolonnen).

epochTime = data["1686171837"].values
templuft = data['23.7'].values
tempvann = data['19.7'].values
tempfukt = data['39.1'].values
light = data['193.55'].values
volt = data["2.52"].values
runTime = data['26485'].values/1000 #gjør ms om til s
```

13. Programmet er skrevet av Thor Inge Hansen ved Skien vgs.

```
sumRunTime = sum(runTime)

#%% Funksjon for å lage datetime-objeter fra epochTime

def epochToDatetime(x):
    liste=[]
    for i in range(len(x)):
        if vintertid:
            t=dt.datetime.utcfromtimestamp(x[i]) + dt.timedelta(hours=1)
        else:
            t=dt.datetime.utcfromtimestamp(x[i]) + dt.timedelta(hours=2)
        liste.append(t)
    return liste

#%% Anvender funksjonen ovenfor

timelist=epochToDatetime(epochTime)

#%% Regning av gjennomsnittlig intervall

interval=np.zeros(len(timelist))

for i in range(len(epochTime)-1):
        interval[i]=int((epochTime[i+1]-epochTime[i]))

interval_mean=np.mean(interval[0:len(interval)-1])

#%% Finner antall ganger intervall mellom sending er over 150 s
count = 1

for i in range (len(interval)):
    if interval[i]>600:  #1200sek =20min = feilsending
        count+=1


#%% Plotfunk

plt.figure(1,figsize=(11,8))
plt.plot(timelist,volt,'b.')
plt.ylabel('Spenning[V]')
plt.xlabel('Tid')
```

```
plt.title('Spenning (1S3P 2500mah liPo)')
plt.ylim(3.2,4.2)
plt.grid()

plt.figure(2,figsize=(11,8))
plt.plot(timelist,light,'b.')
plt.ylabel('Lux')
plt.xlabel('Tid')
plt.title('Lysstyrke')
plt.ylim(0,800)
plt.grid()

plt.figure(3,figsize=(11,8))
plt.plot(timelist,templuft,'r.')
plt.plot(timelist,tempvann,'b.')
plt.xlabel('Tid')
plt.ylabel('Temperatur')
plt.title('Temperaturmålinger')
plt.grid()

plt.figure(4,figsize=(11,8))
plt.plot(timelist,tempfukt,'b.')
plt.xlabel('Tid')
plt.ylabel('Luftfuktighet i %')
plt.title('Relativ luftfuktighet')
plt.grid()

plt.figure(5,figsize=(11,8))
plt.plot(timelist,runTime,'b.')
plt.ylim(0,200)
plt.title('runTime')
plt.xlabel('Sendingsnr.')
plt.ylabel('RunTime for programmet [s]')
plt.grid()

print('Antall målinger så langt',len(volt),'#')
print('Antall manglende målinger',count,'#')
print('Prosentvis manglende målinger',round(100*count/len(volt),1),'%')
print('Gjennomsnittlig runtime er',round(np.mean(runTime),1),'s')
print('Gjennomsnittlig intervall er',round((np.mean(interval)/60),1),'min')
print('Sum runtime',round(sumRunTime/3600,1),'timer')
```

# Vedlegg C Eksempelprogram for EVU-kurset

Vedlegget inneholder det gjennomgående eksempelprogrammet som vi skal bruke gjennom hele kurset.

## C.1 Gjennomgående eksempelprogram (EVU-kurs23-Eksempelprogram.ino)

```
// EVU-kurs23-Eksempelprogram.ino
// Kode for uttesting av program for måling og overføring av måledata til server
// Testoppstillingen inneholder følgende:
//
// MKR NB 1500 m/GSM-antenne
// MKR GPS GPS-mottaker
// DS18B20 Temperatursensor for måling i vann
// SD-kort terminal for lagring av data
//
// Nils Kr. Rossing og Thor Inge Hansen 03.02.23


// Inkludering av biblioteker:
#include <DS18B20.h>        // Bibliotek for avlesning av temperatursensoren
#include <OneWire.h>        // Bibliotek som etablerer entrådsbuss til temp.
sensoren
#include <Arduino_MKRGPS.h> // Bibliotek som leser av GPS
#include <MKRNB.h>          // Bibliotek som overførerdata til server via GPRS
#include <Arduino_MKRENV.h> // Bibliotek for lesing av miljøkortet ENV om det
brukes
#include "ArduinoLowPower.h"// Bibliotek som legger MKR i dvale
#include <WDTZero.h>        // Bibliotek som håndterer vakthunden
#include <SD.h>             // Bibliotek for håndtering av SD-kort
#include <SPI.h>            // Bibilotek for håndtering av serie-bus

// Deklarasjon av instanser:
WDTZero MyWatchDoggy;   // Deklarerer instansen til vakthunden MyWatchDoggy av
typen WDTZero
NBClient client;        // Deklarerer instansen som opererer mot serveren
GPRS gprs;            // Deklarerer instansen som opprettet kontakt med GPRS-nettet
NB nbAccess;          // Deklarerer instansen som som gir tilgang til GPRS-nettet
DS18B20 ds(0);        // Deklarerer instansen som kommuniserer med temp. sensoren
File myFile;            // Deklarerer et objekt for filbehandling, kun ved bruk
ved lagring på SD-kort

// Deklarasjon av konstanter:
```

```
#define DEBUG           // Kommenter bort denne dersom Serial.print ikke skal
inkluderes
//#define LED           // Kommenter bort denne dersom LED indikasjon ikke ønskes

// Deklarasjon av variabler:
// Temperatursensor DS18B20
uint8_t address[] = {40, 250, 31, 218, 4, 0, 0, 52};
uint8_t selected;

// Oppkobling til server
char server[] = "sensor.marin.ntnu.no";  // Navnet på serveren der data legges
char path[] = "/cgi-bin/tof.cgi?";       // Angir cgi-kode for plassering i file
char filename[]="Lynkurs.txt";              // Navnet på fil for lagring av data
int port = 80;                           // Port 80 er default for HTTP
boolean connected = false;               // Status oppkobling til GPRS
char buffer[128];                     // Deklarsjon av buffer med tilhørende lengde

volatile unsigned long num = 0;    // Måling nummer
float w_temperature        = 0;    // Vanntemperatur
float BatVolt              = 0;    // Variabel for å holde batterispenningen
long Pause                 = 5000; // En ev. pause i programloopen
int SleepTime             = 5000; // Tid for dvale, for lengre tider enn 32 sek,
sett verdien rett inn i argumentet

float GPSlat               = 0;    // GPS breddegrad
float GPSlong              = 0;    // GPS lengdegrad
unsigned long epochTime    = 0;    // GPS tidsstempel fra 1.1.1980
int numSat                 = 0;    // GPS antall satellitter
int maxNoChecks            = 10000; // Antall runder for lesing av GPS-data
int noChecks               = 0;    // Antall nødvendige sjekk av GPS

// Deklarasjon av porter
int BatVoltPin = A3;            // Analog port for tilkobling av batterispenning
const int SD_CS_PIN = 4;         // Port for styring av SD-kort terminal,
fabrikkbestemt

void setup() {
 pinMode(LED_BUILTIN, OUTPUT); //Port for innebygget LED, definert som utgang

  analogReadResolution(12);      // Angir oppløsning for AD-konverter

  #ifdef DEBUG
```

```
    Serial.begin(115200);       // Setter opp data hastighet til monitor
  #endif

  if (!GPS.begin())             // Initialisering av GPS
  {
   #ifdef DEBUG
     Serial.println("GPS initialisering mislyktes");
   #endif
  }

  GPS.begin();              // Initialisering av GPS
  SPI.begin();

   if(!SD.begin(SD_CS_PIN))
  {
    #ifdef DEBUG
    Serial.println("Initialisering SD terminal mislyktes");
    Serial.println("Har du satt i SD-kort?");
    #endif
  }

  // Lag en file med navnet DataFile.txt
  myFile = SD.open("DataFile.txt", FILE_WRITE);
  delay(1000);
  myFile.close();
  delay(100);

  // Initialiserer dvale-funksjon
  LowPower.attachInterruptWakeup(RTC_ALARM_WAKEUP, dummy, CHANGE);

  // Initialisering av "vakthund"
  MyWatchDoggy.attachShutdown(myshutdown); // Starter funksjonen myshutdown() ved
reset
  //MyWatchDoggy.setup(WDT_SOFTCYCLE1M);     // Setter intervalltiden for
"vakthunden"

  // Opprett tilkobling til 4G - GPRS-tjeneste
  connectToGPRS();

  delay(1000);
}
```

```
void loop() {
  //readGPSdata1();     // Inkluder GPS
  readWaterTemp();      // Inkluder måling av temp. i vann
  //readBatVolt();     // Inkluder måling av baterispenning om den er implementert

  makeString();         // Bygge opp bufferet for overføring av data
  //sdPrint();          // Skriv data til SD-terminal
  //connectToGPRS();    // Koble til GPRS-nettverket
  connectToServer();    // Koble opp mot server og overfør data om den ikke alt
er oppkoblet

  printData();
  printDataString();    // Skriv ut bufferet
  //MyWatchDoggy.clear(); // Resetter vakthunden

  num++;                // Målingens nummer fra start
  //GoToSleep();        // Legg microkontrolleren og GPS'en i dvale
  delay(Pause);         // Ev. legg inn en pause i loopen
}

// Funksjonen leser av vanntemperaturen
void readWaterTemp()
{
  w_temperature = ds.getTempC();
}

// Funksjonen leser av batterispenningen
void readBatVolt(){
  BatVolt = analogRead(ADC_BATTERY)*(4.25/4096); //volts
}

// Funksjonen leser av GPS-data med bruk av en for-loop
void readGPSdata1()
{
  noChecks = 0;

  for (int i = 0; i < maxNoChecks; i++)
  {
    // Check if there is new GPS data available
    if (GPS.available())
    {
```

```
      // If there is, read GPS values
      GPSlat   = GPS.latitude();
      GPSlong  = GPS.longitude();
      epochTime  = GPS.getTime();
      numSat = GPS.satellites();

      break;
    }

    noChecks++;

    #ifdef DEBUG
      if (noChecks == maxNoChecks) Serial.println("Mislykket henting av posisjon
fra GPS");
    #endif

  }
}

// Funksjonen bygger opp datastrengen for overføring
void makeString()
{
   sprintf(buffer,"/cgi-bin/tof.cgi?%s,%u;%u;%.6f;%.6f;%i;%.1f;%.2f", filename,
num, epochTime, GPSlat, GPSlong, numSat, w_temperature, BatVolt);
}

// Funksjonen skriver ut bufferet til monitoren
void printDataString()
{
  #ifdef DEBUG
  Serial.println(buffer);
  Serial.println();
  #endif
}

// Funksjonen kobler opp til GPRS-nettverket (4G)
void connectToGPRS()
{
  while (!connected)
  {
    if ((nbAccess.begin("", true, true) == NB_READY) && (gprs.attachGPRS() ==
GPRS_READY))
```

```
    {
      connected = true;

      #ifdef DEBUG
        Serial.println("Vellykket tilkobling til GPRS");
      #endif

      #ifdef LED
        flash(1);
      #endif
    }
    else
    {
      #ifdef DEBUG
        Serial.println("Mislykket tilkobling til GPRS");
      #endif

      #ifdef LED
        flash(5);
      #endif

      break;
    }
  }
}

void sdPrint(){

  myFile = SD.open("DataFile.txt", FILE_WRITE);
  delay(1000);

  if (myFile)
   {
    myFile.println(buffer);
    myFile.close();
   }
}

// Funksjonen kobler opp til server og skriver databuffer til serveren
void connectToServer()
{
```

```
  if (client.connect(server, port))
  {
    client.print("GET "); // Gjør et HTTP request:
    client.print(buffer);
    client.println(" HTTP/1.1");
    client.print("Host: ");
    client.println(server);
    client.println("Connection: close");
    client.println();

    #ifdef DEBUG
      Serial.print("Vellykket overføring til server av datasett nr.: ");
      Serial.println(num);
    #endif

    #ifdef LED
      flash(2);
    #endif
  }
  else
  {
    #ifdef DEBUG
      Serial.print("Mislykket overføring til server av datasett nr.: ");
      Serial.println(num);
    #endif

    #ifdef LED
      flash(10);
    #endif
  }
}

// Funksjonen blinker et angitt antall ganger som en hjelp til debugging
void flash(int number)
{
  delay(1000);
  for(int i=0; i<number; i++)
  {
  digitalWrite(LED_BUILTIN, HIGH);
  delay(100);
  digitalWrite(LED_BUILTIN, LOW);
```

```
  delay(100);
  }
 delay(1000);
}

// Funksjonen legger kretsen i dvale
void GoToSleep() // Legg mikrokontrolleren og GPS'en i dvale
 {
   #ifdef DEBUG
     Serial.println("Går i dvale");
   #endif

   #ifdef LED
     flash(3);
   #endif

   GPS.standby();
   LowPower.deepSleep(SleepTime);
 }

// Returfunksjon etter endt dvale
void dummy()
{
   GPS.wakeup();
   NVIC_SystemReset(); // Resett programmet
}

void myshutdown()
{
  #ifdef DEBUG
    Serial.println("Resetter MKR");
  #endif
}

void printData(){
  #ifdef DEBUG
    Serial.print("GPS sjekk : "); Serial.println(noChecks);
    Serial.print("Breddegrad: "); Serial.println(GPSlat,6);
    Serial.print("Lengdegrad: "); Serial.println(GPSlong,6);
    Serial.print("Epoketid  : "); Serial.println(epochTime);
    Serial.print("Antall sat: "); Serial.println(numSat);
```

```
    Serial.print("Temp. vann: "); Serial.println(w_temperature);
    Serial.print("Batterisp.: "); Serial.println(BatVolt);
    Serial.print("Måling nr.: "); Serial.println(num);
    Serial.println();
  #endif
}
```

# Vedlegg D Operativt program med ekstern "vakthund"

Dette programmet er beregnet for å brukes dersom man bruker ekstern "vakthund" som holder styr på tidspunkter for måling. Hele programmet ligger setup()-funksjonen og vil derfor kjøre en gang hver gang "vakthunden" slår på spenningen. Dersom programmet skulle "henge seg" vil en intern programvare "vakthund" sende programmet til en endestopp-funksjon som terminerer programmet og gir beskjed til den eksterne "vakthunden" om at den kan slå av spenningen. I så fall mister man en måling[14].

```
// Programmet er skrevet av Thor Inge Hansen
// Programmet er modifisert av Nils Kr. Rossing
// 24.05.23 - Endret fil navn til SeptKursTest_1.txt
// 25.05.23 - Endret fil navn til SeptKursTest_2.txt - ;
// 27.05.23 - Endret fil navn til SeptKursTest_3.txt
// 29.05.23 - Montert ENV-kort og testet SD-kort
// 07.06.23 - Endret fil navn til SeptKursTest_5.txt



#include <DS18B20.h>
#include <OneWire.h>
#include <Arduino_MKRENV.h>
#include <SPI.h>
#include <SD.h>
#include <Arduino_MKRGPS.h>
#include <MKRNB.h>
#include <WDTZero.h>

WDTZero myWDT;
NBClient client;
GPRS gprs;
NB nbAccess;
DS18B20 ds(0);
File myFile;

//#define printState; //comment out to turn off all Serial print
```

14. Programmet er utviklet av Thor Inge Hansen ved Skien vgs. i samarbeid med Skolelaboratoriet ved NTNU

```
uint8_t address[] = { 40, 250, 31, 218, 4, 0, 0, 52 };
uint8_t selected;
char server[] = "sensor.marin.ntnu.no";
char path[] = "/cgi-bin/tof.cgi?";
char filename[] = "SeptKursTest_5.txt";
int port = 80;
bool connected = false;  // Status oppkobling
char buffer[128];

unsigned long pause = 1000 * 60 * 6;  //6 min
int shutOffPin = 1;                   // Port til shutOffPin
float w_temperature = 0;
float GPSlat = 0;
float GPSlong = 0;
unsigned long epochTime;
int numSat = 0;
float temperature = 0;
float humidity = 0;
float pressure = 0;
float illuminance = 0;
float battVolt = 0;

void setup() {
  pinMode(shutOffPin, OUTPUT);
  digitalWrite(shutOffPin, HIGH);
#ifdef printState
  Serial.begin(9600);
  while (!Serial) {}
#endif

  analogReadResolution(10);  // 10Bits

  myWDT.attachShutdown(shutDownFunc);  //when shutdown initialize
myshutdown()
  myWDT.setup(WDT_SOFTCYCLE2M);
```

```
  while (!GPS.begin()) {}  //vent til GPS

  readWaterTemp();

  readGPSdata();
  delay(500);
  readENVData();
  delay(1000);  //Delay for å prøve å få mer stabile spenningsverdier
  readBattVoltage();
  delay(1000);

  makeString();

  sdPrint();

#ifdef printState
  serialPrint();
#endif
  connectToGPRS();
  connectToServer();
  delay(1000);
#ifdef printState
  Serial.println(millis());
#endif
  digitalWrite(shutOffPin, LOW);  //Tells Picaxe to turn off power
}

void readWaterTemp() {
  w_temperature = ds.getTempC();
}

void readBattVoltage() {
  //Forutsetter lik deling av batterispenning mellom to like resistorer
og at A1 brukes
  battVolt = analogRead(A1) * 2 * 3.3 / 1024;
}
```

```
void readGPSdata() {

#ifdef printState
  Serial.println("Started readGPSdata");
#endif

  while (!GPS.available()) {}  // vent til GPS er tilgjengelig

  GPSlat = GPS.latitude();
  GPSlong = GPS.longitude();
  epochTime = GPS.getTime();
  numSat = GPS.satellites();
}

void readENVData() {
  ENV.begin();
  temperature = ENV.readTemperature();
  humidity = ENV.readHumidity();
  pressure = ENV.readPressure();
  illuminance = ENV.readIlluminance();
}

void makeString() {
  sprintf(buffer, "%s%s,%u,%.6f,%.6f,%i,%.1f,%.1f,%.1f,%.1f,%.2f,%.2f",
path, filename, epochTime, GPSlat, GPSlong, numSat, temperature, w_tem-
perature, humidity, pressure, illuminance, battVolt);
}

void connectToGPRS() {
#ifdef printState
  Serial.println("Started connectToGPRS()");
#endif

  while (!connected) {
   if ((nbAccess.begin("", true, true) == NB_READY) && (gprs.attachGPRS()
== GPRS_READY)) {
```

```
      connected = true;
#ifdef printState
      Serial.println("GPRS connected");
#endif
    } else {
      delay(1000);
#ifdef printState
      Serial.println("connected still False");
#endif
    }
  }
}
void connectToServer() {
#ifdef printState
  Serial.println("in connectToServer()");
#endif

  if (client.connect(server, port)) {
#ifdef printState
    Serial.println("Connected to Server");
#endif

    client.print("GET ");  // Gjør et HTTP request:
    client.print(buffer);
    client.print(',');
    client.print(millis() + 10271);  //Prints runtime + set delay + cal-
culated offset to show the runtime untill shuOffPin turns low
    client.println(" HTTP/1.1");
    client.print("Host: ");
    client.println(server);
    client.println("Connection: close");
    client.println();

#ifdef printState
    Serial.println("String uploaded to Server");
#endif
```

```
  } else {

#ifdef printState
    Serial.println("No connection to server");
#endif
  }
}
void sdPrint() {

  if (!SD.begin(4))
    while (1)
      ;  //STOPP OPP

  myFile = SD.open("Sept5.txt", FILE_WRITE);

  if (myFile) {
    myFile.println(buffer);
    myFile.close();
  }
}

void serialPrint() {
#ifdef printState
  Serial.println(buffer);
#endif
}

void shutDownFunc() {
  digitalWrite(shutOffPin, LOW);
  delay(5000);
}

void loop() {}
```

Prosjektet “Miljøovervåking med bøye” handler om å samle inn måledata fra kystnære strøk. Teknologien kan imidlertid brukes i mange ulike sammenhenger, i områder der man normalt ikke har tilgang til lokale nettverk slik at man kan overføre data via Wi-Fi. For å få til dette tar man i bruk ulike typer smalbånds mobildata som f.eks. NB IoT som Telenor med flere tilbyr.

Våren 2020 kom det en henvendelse fra Håvard Holm ved Institutt for marin teknikk som hadde en ide om å la elever bygge havgående bøyer med billige materialer og utstyre dem med elektronikk som kunne samle inn måledata fra havet og overføre disse til en server på land.

En kan tenke seg flere løsninger ved at man i større eller mindre grad bruker etablerte tjenester eller bygger opp tilbudet fra bunnen av og samler data til lokale servere. Vi har i dette tilfellet valgt å sende data til en lokal server ved Inst. for marin teknikk ved NTNU.

Hefte beskriver gangen fra ide og til man har etablert en forbindelse mellom målesensorer til data kan lastes ned fra serveren og analyseres ved hjelp av Excel, Python-script eller Google Earth. I tillegg beskrives aktuelle Arduino-komponenter og sensorer for måling i vann og en enkel bøye for uttesting.

***Nils Kr. Rossing***
Dosent emeritus ved Skolelaboratoriet
Institutt for fysikk, NTNU
E-post: nils.rossing@ntnu.no

***Thor Inge Hansen***
Adjunkt 1 ved Skien videregående skole
E-post: thor.inge.hansen@vtfk.no

ISBN
Rev. 1.0 - 30.07.23

Trykk: NTNU Grafisk senter

**NTNU**
**Trondheim**
**Institutt for fysikk**
**Skolelaboratoriet**
**for matematikk, naturfag og teknologi**

**Tlf. 73 55 11 43**
**https://www.ntnu.no/skolelab**