Rev 3.4

Nils Kr. Rossing og Christoffer Stausland

Lag en barometrisk høydemåler

Grunnkurs programmering Arduino

NTNU
Trondheim

Institutt for fysikk

Skolelaboratoriet
for matematikk, naturfag og teknologi

Januarr 2022

Lag en barometrisk høydemåler

Grunnkurs programmering Arduino

Nils Kr. Rossing Skolelaboratoriet og
Christoffer Stausland, NAROM

Lag en barometrisk høydemåler – Grunnkurs programmering Arduino

Trondheim 2022

Layout og redigering: Nils Kr. Rossing, Skolelaboratoriet ved NTNU

Trykk: NTNU Grafisk senter

Tekst og bilder: Nils Kr. Rossing, Skolelaboratoriet ved NTNU

Kursholdere: Nils Kr. Rossing, Skolelaboratoriet
Christoffer Stausland, NAROM

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 43
http://www.ntnu.no/skolelab/

Rev 3.4 – 20.01.22

Forord

Det har vært registrert at deltakere på CanSat-kurs har svært varierende bakgrunn når det gjelder programmering og at det har vært et behov for en gjennomgang av det helt grunnleggende når det gjelder programmering generelt og programmering av Arduino spesielt. Dette ble bekreftet da vi åpnet for at deltagere på CanSat-kurset sept. 2019 fikk anledning til å melde seg på et forkurs i grunnleggende programmering av Arduino. Vi registrerte at så og si alle valgte å delta på forkurset. Dette viser også at det ikke synes å bety så mye at lærere bruker tre istedet for to dager på kurs ved oppstarten av skoleåret, dersom de kan få anledning til å føle seg tryggere i klasserommet. Vi erfarte det samme ved påmelding til kursene i september 2020.

I og med at det er to separate kurs åpner det også for at de som kun ønsker en innføring i grunnleggende programmering har anledning til det. Det er imidlertid å anbefale at man planlegger programmeringskurset i sammenheng med CanSat-kurset. Dette gir en relativt fragmentarisk opplæring i programmering en retning knyttet til en konkret anvendelse i forbindelse med CanSat. Et slikt prosjektorientert programmeringskurs ser vi som attraktivt i en skole hvor alle skal tilegne seg grunnleggende programmeringskunnskaper.

Utgave 3.0 av dette heftet er tilpasset bruk i Trondheim. Årsaken til at det er forskjell på hefter som er beregnet på Andøya og i Trondheim er valg av referansestasjon. På Andøya brukes Andenes lufthavn, mens i Trondheim brukes Vold forsøksgård. Normal arbeider to og to sammen. Under korona-epidemien har det vært behov for å jobbe enkeltvis, men å kommunisere på avstand. Det har derfor vært behov for flere sett hvilket har medført at innkjøp av flere komponenter. Siden BMP180 er på vei ut og BMP280 på vei inn, så har det vært behov for å skrive om deler av heftet. Dette heftet inneholder derfor begge variantene.

En takk til Odd-Einar Cedervall Nervik som har gått gjennom og kvalitetssikret heftet før kurset 2020. Han er også kursleder i september 2020.

Selv om heftet i utgangspunktet ble laget som et forkurs til CanSat-kurset så er det også så generelt at det godt kan brukes også som en grunnleggende innføring i programmering av Arduino. Det har derfor også vært brukt som en grunninnføring overfor yrkesfaglærere og andre.

Skolelaboratoriet ved NTNU
Januar 2022
Nils Kr. Rossing
Christoffer Stausland

Innhold

1 Innledning

Heftet er ikke ment som noen lærebok, men som et ressurshefte for den som ønsker å anvende Arduino som et undervisningsopplegg i faget Teknologi og forskningslære (ToF) i videregående skole eller i Teknologi i praksis (TiP) i ungdomsskolen. Men også som et arbeidsbok til kurset *Grunnkurs programmering Arduino,* et introduksjonskurs til CanSat kursene. Vi har derfor valgt ut øvelser som skal bygge opp om senere arbeid med CanSat, men som også gir mening som et frittstående kurs i grunnleggende programmering av Arduino.

1.1 Fagfornyelsen 2020

Fra og med Fagfornyelsen 2020 skal programmering og digital kompetanse inn i alle fag fra langt ned i grunnskolen og til videregående skole, men blir også stadig viktigere som verktøy i fag ved høgskoler og universitet. Årsaken til dette er sammentatt, men noen av argumentene er knyttet til følgende:

- *Vi lever i en verden hvor vi finner digitale hjelpemidler så og si over alt, og hvor de i stadig større grad overtar funksjoner som mennesker har utført. Flere og flere arbeidsplasser handler derfor om enten å utvikle, tilpasse eller bruke digitale maskiner og hjelpemidler. Vi trenger derfor en arbeidsstokk med en annen kompetanse enn tidligere.*
- *Noen har innsett at programmering oppøver en kompetanse innen problemløsning som også kan komme til nytte i andre fag. Det handler om en kunnskap som øver opp evnen til strukturere tankeprosesser som er spesielt nyttig ved problemløsning, noen kaller det algoritmisk tenkning.*
- *Behersker man programmering og mikrokontrollere gir dette et særdeles slagkraftig designverktøy i forbindelse med produktutvikling. Det samme gjelder også digitale verktøymaskiner som på grunn av prisnivået er blitt tilgjengelig for store brukergrupper. Her tenkes spesielt på 3D-printere, CNC-fresere, laserkuttere, vinylkuttere, symaskiner o.l. som alle styres av programvare.*

Også demokratiargumentet kan brukes i denne sammenhengen.

- *Skal vi leve i et demokrati hvor digitaliseringen blir mer og mer utbredt og hvor det i stadig større grad anvendes digitale metoder og teknologi, så må vi kunne forstå så pass av denne teknologien at vi er istand til å mene noe om viktige samfunnsspørsmål som involverer slike metoder og slik teknologi.*

I Fagfornyelsen 2020 beskrives teknologi som et viktig kjerneelement i *Naturfag*:

> *Elevene skal forstå, skape og bruke teknologi, inkludert programmering og modellering, i arbeid med naturfag. Gjennom å bruke og skape teknologi kan elevene kombinere erfaring og faglig kunnskap med å tenke kreativt og nyskapende. Elevene skal forstå teknologiske prinsipper og virkemåter og vurdere hvordan teknologi kan løse og skape utfordringer.*

Tilsvarende finnes under grunnleggende ferdigheter i *Kunst og håndverksfaget*:

> *Digitale ferdigheter i kunst og håndverk innebærer å kunne bruke ulike digitale verktøy, medier og ressurser som kilde til inspirasjon og i skapende arbeid. Kjennskap til regler om opphavsrett og personvern når man bruker egne eller andres bilder, filmer og skapende arbeid er vesentlig på alle trinn.*

I et av kjerneelementene i matematikkfaget legges det vekt på eksperimentelt arbeid og problemløsning:

> *Utforsking i matematikk handlar om at elevane leiter etter mønster, finn samanhengar og diskuterer seg fram til ei felles forståing. Elevane skal leggje meir vekt på strategiane og framgangsmåtane enn på løysingane. Utforsking i matematikk samsvarer med bruken av prøving og feiling innanfor tradisjonell samisk opplæring. Algoritmisk tenking er viktig i prosessen med å utvikle strategiar og framgangsmåtar for å løyse problem.*

I Teknologi og forskningslære er ikke programmering og digital kompetanse foreløpig nevnt. Det nærmeste vi kommer er:

> *... bruke sensorer og styringssystemer i forbindelse med forsøk og konstruksjoner.*
> (ToF 1 - "Den unge ingeniøren")

som er hentet fra kompetansemålene for faget ***Teknologi og forskningslære 1***. Programmering av mikrokontrollere kan med god samvittighet legges under dette punktet.

Når det gjelder valgfaget ***Teknologi i praksis*** i ungdomsskolen, så favner kompetansemålene så vidt at det ikke er noe problem å inkludere mikrokontrollere i et produkt som elevene skal utvikle på bakgrunn av en produktspesifikasjon. Kravet er at oppgavene som løses og produktene som utvikles settes inn i den sammenhengen som planen skisserer.

1.2 Foreløpig plan for kurset Arduino – Grunnleggende programmering

Det er mange måter å gjennomføre kurs på. Dette heftet legger opp til at deltagerne skal jobbe på egen hånd. De får en del tips under veis, men utfordres i stadig større grad til selv å finne løsninger ut fra de erfaringer de høster. De har selvfølgelig også muligheter til å spørre i tillegg til at de underveis vil få faglig påfyll gjennom korte innslag med sentral teori.

Program kursdag – Oppmøte Skolelaboratoriet, Realfagbygget NTNU, C2-101

09.00 – 09.30	Velkommen, praktisk og forventninger
09:30 – 10:15	**Økt 1**
	Bli kjent med Arduino og programverktøy
10:15 – 10:30	**Frukt- og kaffepause**
10:30 – 12:00	**Økt 2**
	Korte avbrekk med viktig kunnskap – f.eks. bruk av I^2C buss
	Første prosjekt: Fra blinkende lysdiode til rakettnedtelling (OLED-display)
12:00 – 12:30	**Lunsj, Realfagkantina**

12:30 – 13:30	**Økt 3** Korte avbrekk med viktig kunnskap – f.eks. AD-konverteren og omregning *Andre prosjekt:* Avlesning av analog porter, framstilling av et voltmeter og temperatursensor (termometer)
13:30 – 13:45	**Kaffepause**
14:00 – 15:15	**Økt 4** Korte avbrekk med viktig kunnskap – f.eks. omregning fra trykk til høyde *Tredje prosjekt:* Avlesning trykk og lag en høydemåler
15:15 – 16:00	**Oppsummering, refleksjon** og diskusjon om hvordan ta erfaringene inn i klasserommet

Forslag til kjøreplan:

0. **Informasjon og refleksjon:** Introduksjon med gjennomgang og diskusjon om forventninger

Økt 1 (45 min))

1. **Teori:** Kort intro om Arduino, konseptet og Atmel (NKR)
   Hva er en mikrokontroller? (NKR)
2. **Teori:** Kort gjennomgang av Sparkfun inventors kit med tilleggskomponenter (NKR)
3. **Praksis:** Oppkobling og installasjon av programvare (som egentlig bør være gjort før oppstart på kurset) Sjekk at kommunikasjonen mot Arduinokortet er i orden (NAROM)

Økt 2 (90 min)

4. **Teori:** Gjennomgang av strukturen i et typisk Arduino-program (narom)
   De viktigste kommandoene i IDE
5. **Praksis:** Koble opp blinklys
6. **Praksis:** Skriv intro-oppgave blinkende lys og overfør programmet og få lysdiode til å blinke (0,5 sek. på 0,5 sek. av). Ev. eksperimentere med blinklys
7. **Teori:** Skriv til monitor. Skriv «Av» og «På» (narom)
8. **Praksis:** Lag et uendelig forløp med å slå av og på
9. **Teori:** "For loop" og nedtelling (narom)
10. **Praksis:** Lag et forløp som teller ned fra 10 til 0 på monitoren
    Hva skal skje når den når 0?
11. **Teori:** Bruk av display I^2C-buss, installasjon av bibliotek, skriving til display (NKR)
12. **Praksis:** Lag en nedtelling fra 10 – 0 på displayet

Økt 3 (60 min) – Lag et voltmeter og et termometer

13. **Teori:** Analog avlesning, AD-konverter, digital omregning, potensiometer (NKR)

14. **Praksis:** Koble opp potensiometer, les av spenningen, reguler spenningen
15. **Praksis:** Skriv ut verdien på displayet
16. **Teori:** Temperatursensoren TMP36, ev. kort om NTC-motstanden (NKR)
17. **Praksis:** Lag et termometer, sjekk med glasstermometer

Økt 4 (75 min) – Lag en trykksensor

18. **Teori:** Kort om trykksensoren, Installasjon av biblioteket (narom)
19. **Praksis:** Koble opp BMP180, les av og skriv ut verdi på display
20. **Praksis:** Regn om til høyde ved å skrive inn formelen
21. **Praksis:** Mål høyden i trapperommet i Realfagbygget
22. **Refleksjon:** Oppsummering og evaluering, ble forventningene innfridd?

Overordnet læringsutbytte:

Målet med kurset er gi lærere en generell innføring i programmering av Arduino med tanke på å ta det i bruk i egen undervisning.

Læringsutbytte:

Deltagerne skal:

- … kunne koble opp Arduino med sensorer og aktuatorer
- … kunne utføre grunnleggende programmering av Arduino
- … kunne hente inn data fra sensorer og brytere og kunne styre aktuatorer
- … ha kunnskap om vanligste programstrukturer brukt med Arduino
- … kunne bruke de vanligste kommandoene
- … kunne bruke monitor for visning av data

Faglig innhold:

Kurset vil basere seg på noen utvalgte prosjekter som deltagerne arbeider med to og to:

- Arduino mikrokontroller, hva er det?
- Oppstart og bruk av programverktøy
- *(Arduino-kortet, koblingsbrett, elektroniske komponenter, jumpere, programeditoren, grunnleggende programstruktur, innskriving av program, kompilering, opplasting, …)*
- Første prosjekt: Fra blinkende lysdiode til trafikklys eller rakettnedtelling
- *(Lysdioder, oppkobling, digitale porter, styring av LED, ev. LED-display, sekvensiell tenkning …)*

- Andre prosjekt: Lag et voltmeter (display)
  (Avlesning av analoge porter, ADC, omregning, bruk av IDE monitor, opplasting av bibliotek, skrive til lite display, bruk av I2C-buss)
- Tredje prosjekt: Avlesning trykk og temperatur og lag en høydemåler
  (Avlesning av temp. sensor, omregninger, avlesning av trykksensor (BMP180), skriv til display, beregning av høyde, kalibrering ...)

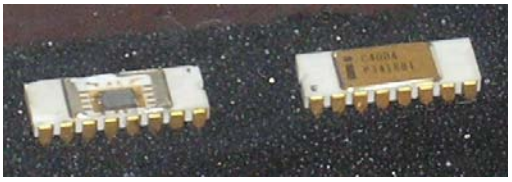
Vi ønsker at hver enkelt skal jobbe i sitt eget tempo, derfor må man regne med at man kanskje ikke rekker å komme gjennom alle tre prosjektene.

2 Bakgrunnen

2.1 Mikroprosessorer og mikrokontrollere

Vi er vel alle klar over at vi omgir oss med store mengder elektronikk både i skolen, på jobben, i bilen, hjemme, ja overalt hvor vi ferdes. De fleste av oss går rundt og bærer på svært avansert elektronikk og kommunikasjonsutstyr. Tenker vi etter så vil de fleste ha med seg en kalkulator som er en liten datamaskin, de vil ha en smarttelefon som ikke bare er ett kommunikasjonshjelpemiddel for sending av tekstmeldinger og telefonsamtaler, men i tillegg tilbyr stadig flere tilleggsfunksjoner som musikkanlegg, navigasjonsutstyr og kart, kamera for opptak av bilder og film, kalendere og arkivsystemer for å holde orden på hverdagen, spill og underholdning, bibliotek med både romaner og lærebøker, og utallige andre funksjoner. I tillegg går flere av oss rundt med bærbare PC-er eller nettbrett for større oppgaver, gjerne med nettilgang.

Figur 2.1 HTC Smartphone



Figur 2.2 Intel mikroprosessor Intel 4004 regnes for den første mikroprosessoren. Lansert 15. nov. 1971

Sentral i alt slikt utstyr er mikroprosessoren som er en miniatyrisert datamaskin som satt på spissen stort sett bare kan addere, sammenligne og flytte på binære tall. Dessuten kan den "Den lille multiplikasjonstabell" for binære tall som er 0 x 0 = 0 og 1 x 0 = 0 og 1 x 1 = 1. I tillegg må den kommunisere med omverdenen ved å omdanne kontinuerlige størrelser, som lyd, lys, temperatur, radiosignaler, gravitasjon osv. til binære tall.

Selv om mikroprosessorer i dag er blitt langt mer avanserte og raskere enn den første mikroprosessoren som Intel lanserte i 1971, så er grunnfunksjonen omtrent den samme.

En mikroprosessor er derfor en temmelig enfoldig og dum elektronisk "duppe dings", men den har noen svært nyttige egenskaper:

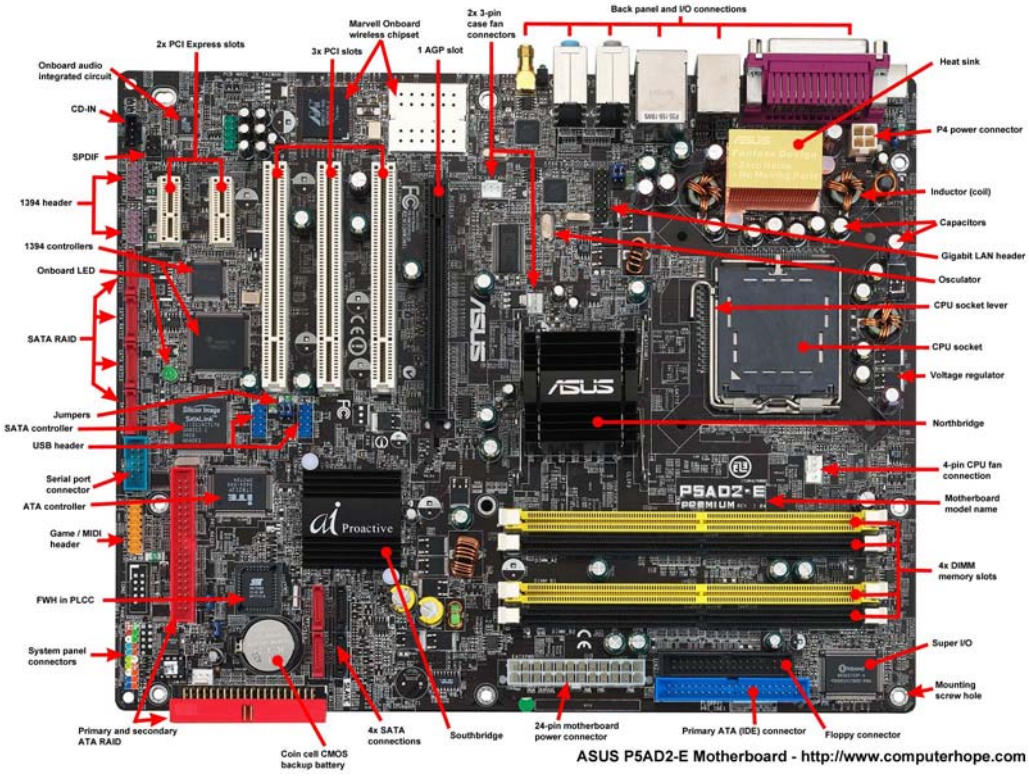
Den arbeider svært fort. I løpet av et sekund kan den utføre mange hundre millioner, ja noen ganger milliarder av små enkle oppgaver i sekundet . Setter man sammen mange nok små enkle oppgaver, kan man gjøre store komplekse oppgaver i løpet av en brøkdel av et sekund.

Den glemmer aldri. Når noe er lagret i hukommelsen til en mikroprosessor så går det ikke "i glemmeboka". Så lenge den virker som den skal så huskes det til "evig tid". Selv om "strømmen går" husker den det som er skrevet inn i hukommelsen.

Den går ikke lei. Dersom den er instruert til å utføre en regneoperasjon riktig, så gjør den det riktig hver gang uten å bomme en eneste gang. Dersom noe går galt så skyldes det som regel at de som har laget mikroprosessoren eller oppskriften (programmet) den følger, har gjort feil eller ikke tenkt på alt. Dessuten er en slik gjenstand sårbar slik at den kan bli ødelagt av gal bruk og uforsiktig bruk.

Ofte benyttes begrepene *mikroprosessor* og *mikrokontroller*. I dette heftet skal vi bare bruke mikrokontrollere. Men hva er egentlig forskjellen?

En mikroprosessor er en meget generell regne- og datamanipuleringsenhet som oftest er plasseres i et miljø med egne kretser for lagring av data og program på utsiden av mikroprosessoren. Likeså foregår all spesialisert kommunikasjon med omverdenen med egne spesialiserte kretser. Åpner du en datamaskin vil du gjerne se et stort kretskort (moderkort) som inneholder mange ulike kretser (se figuren under).

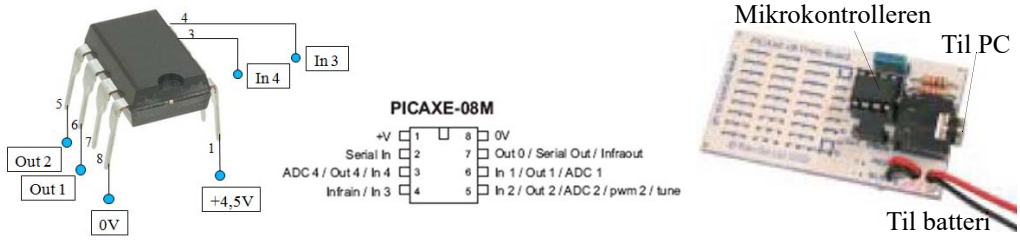


En mikrokontroller inneholder regne- og data-manipuleringsenheten, lager for program og data og det som skal til for å kommunisere med omverdenen, enten det er analoge spenninger eller digitale inn- og utganger, så finnes alt dette inne i den integrerte kretsen. Mikrokontrolleren er med andre ord en komplett datamaskin inkludert i én integrert krets. Prisen en må betale er at den ikke er så kraftig og generell som en mikroprosessor, men som oftest er den kraftig nok til vårt bruk. Den mangler dessuten muligheter for direkte å koble til PC-tastatur og skjerm. Skal vi kom-

munisere med en mikrokontroller er vi derfor som oftest avheng av å bruke en PC med tilpasset programvare for å programmere den. Når den først er programmert kan den frigjøres fra tastatur, mus, skjerm og PC.

Mens mikroprosessorer gjerne står i datamaskiner, finner vi mikrokontrollere i alt mulig annet utstyr som vaskemaskiner, biler, overvåkning og styringsystemer og annen små-elektronikk. Det er innen dette markedet at Atmel Norge gjorde seg bemerket.

Det finnes en rekke forskjellige familier med mikrokontrollere. Som et eksempel på hvor enkel en mikrokontroller kan være er det på bilde under vist en PIC-kontroller. Så og si en liten datamaskin med 8 bein (til venstre).



Denne vesle kretsen har i alt 5 porter (bein) som kan programmeres som digitale inn- eller utganger, med og uten puls-bredde-modulasjon, analoge innganger, eller innganger som kan fungere som mottakere for IR-kommunikasjon eller som berøringsbrytere. En av utgangene er forberedt for levere toner fra skalaen. Til høyre på figuren over er vist kretsen montert på et lite kretskort, med tilkoblingsplugg for batteri og programmering fra PC.

På 1980-tallet oppdaget man at det var mer effektivt å lage mikrokontrollere som hadde få og enkle kommandoer som kunne utføres svart raskt og effektivt, enn å ha mikrokontrollere med mange spesialiserte og ofte kompliserte kommandoer. Denne teknologien kalte man RISK (**R**educe **I**nstruction **S**et **C**omputer) i motsetning til CISK (**C**omlex **I**nstruction **S**et **C**omputer). Atmels mikrokontrollerfamilier er stort sett av RISK typen.

2.2 Arduino og AVR mikrokontrollere - litt historikk

Det utstyret som vi skal bruke i undervisningsoppleggene som vil bli omtalt her går under betegnelsen Arduino. La oss se litt på historien bak før vi går videre.

Arduino er et mikrokontrollerkonsept utviklet i den vesle italienske byen *Ivrea* på begynnelsen av dette årtusen. Hensikten var å lage et kontrollerkort som skulle gjøre det enklere og billigere for studenter å lære seg bruk av mikrokontrollere. Det første Arduino-produktet ble utviklet av grunnleggerne ***Massimo Banzi*** og ***David Cuartielles***. De oppkalte prosjektet etter *Arduin of Ivrea* som var den vesle byens historiske heltefigur. Navnet betyr *sterk venn* og burde passe godt for et kraftig kontrollerkort. Det som startet som et lokalt prosjekt for stu-

denter i 2005, hadde i 2010 spredd seg til hele verden. I februar 2011 hadde de solgt 300 000 eksemplarer. I dag er dette mangedoblet. Den tilhørende programvaren ble utviklet av studenten ***Hernando Barragán*** ved det lokale universitetet i Ivrea, som en "open source" kode.

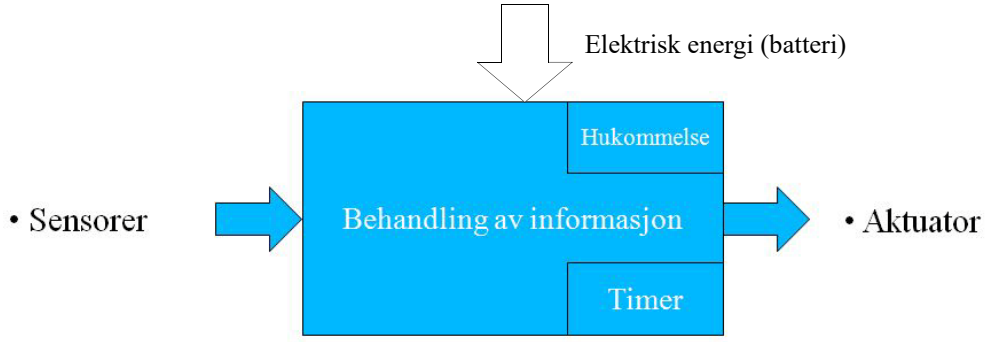
Figur 2.3 Alf Egil Bogen og Vegard Wollan

Kortet var bygget opp omkring ***AVR mikrokontrollere*** fra Atmel (hovedsakelig ATmega8, ATmega168, ATmega328, ATmega1280, og ATmega2560). Det tidligere kontrollkortet i CanSat anvendte ATmega168. Dette er en serie kontrollere som anvender RISK-arkitektur, en svært enkel, men meget effektiv arkitektur. Det er moro å vite at den første kontrolleren i denne serien ble utviklet av studentene ***Alf-Egil Bogen*** og ***Vegard Wollan*** ved NTH på begynnelsen av 1990-tallet. Etter endt studium tok de med seg konseptet inn i firmaet Nordic VLSI (nå NORDIC Semiconductor), hvor det ble videreutviklet. I 1995 gikk de ut av Nordic VLSI og ble snart kjøpt opp av Atmel. Firmaet er i dag kjøpt opp av Microchip Technology og heter Microchip Technology Norway og hadde i 2016 126 ansatte med en omsetning ca. 204 mill. Både Alf Egil og Vegard er gått over i andre virksomheter. De sier selv at AVR ikke har noen spesiell betydning, men det er allment akseptert at det opprinnelig sto for **A**lf (Egil Bogen) and **V**egard (Wollan) 's **R**isc processor.

La oss forsøke å sette mikrokontrolleren inn i en sammenheng. I det neste avsnittet skal vi lage et generelt rammeverk som vi kan sette vår elektronikk og mikrokontrollere inn i.

2.3 Generelt rammeverk

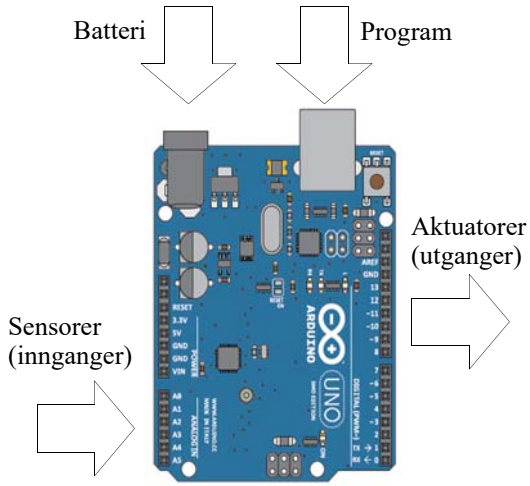
Elektronisk utstyr, nesten uansett hva det er, kan tilpasses følgende beskrivelse (se figuren under).



Utstyret inneholder følgende hoveddeler:

- ***Sensorer***
  Dette er elektroniske enheter som *innhenter* informasjon fra omgivelsene. Eksempler på slike kan være temperatur-, lys-, fuktighets-, bevegelses- og røyksensorer med flere. Det kan også være en enkel bryter av den typen vi bruker til å slå på lyset eller en mikrofon som registrerer lyd eller en uendelighet av andre sensorer.

- ***Aktuatorer***
  Dette er elektroniske enheter som *utfører en oppgave*. Dette kan være en motor som åpner en dør, en lyspære som gir lys, en høyttaler som gir lyd, et varmeelement som gir varme osv. Dvs. en enhet som utfører en aksjon.



- ***Databehandlingsenhet***
  Dette er en elektronisk enhet som kan være svært enkel eller uhyre komplisert og som samler inn informasjon fra omgivelsene ved hjelp av *sensorene*, for så å gi *aktuatorene* beskjed om handling. Sensorene er tilkoblet innganger, mens aktuatorene er tilkoblet utganger. I tillegg tilføres enheten *elektrisk energi* fra batteri eller strømnettet. Noen ganger inneholder enheten en "timer" som holder rede på tiden. Dersom arbeidsoppgavene til enheten er sammensatt må den også inneholde en *oppskrift* (program) som beskriver hvordan den skal behandle informasjonen fra sensorene.

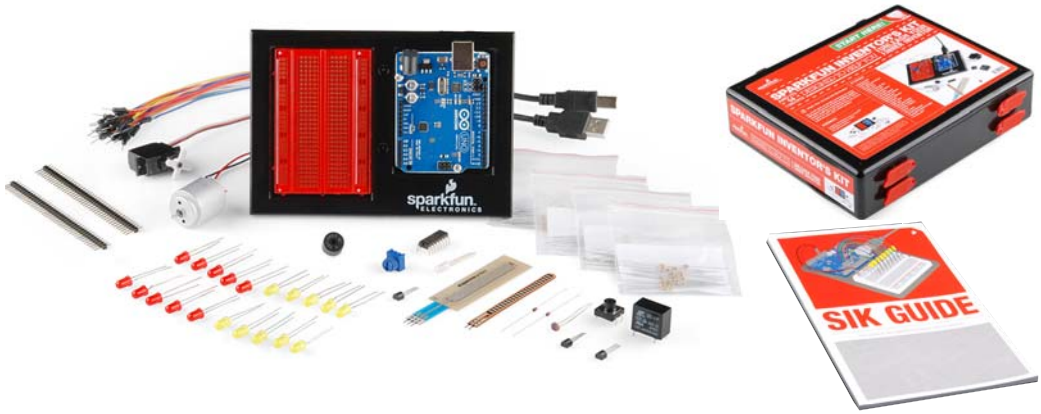
I neste kapittel skal vi se nærmere på *Sparkfuns Inventors kit* som er den grunnpakken vi har bygget våre undervisningsopplegg på.

2.4 Sparkfun Inventors kit: Grunnleggende byggeklosser

Avsnittet gir en oversikt over de elektroniske byggeklossene vi skal bruke i de følgende undervisningsoppleggene

2.4.1 Komponentoversikter

Grunnopplæringen er basert på Sparkfun's *Inventors kit 3.1*.



Med settet følger et relativt rikholdig utvalg av elektroniske byggeklosser. Her er ei liste over innholdet i Sparkfun Inventor's kit:

- 1 stk. Arduino UNO R3
- 1 stk. Plastholder for Arduino og koblingsbrett
- 1 stk. SIK Manual
- 1 stk. Oppbevaringsboks for kittet
- 1 stk Koblingsbrett
- 1 stk Skiftregister – 74HC595
- 2 stk. Transistorer – 2N2222
- 2 stk. Dioder – 1N4148
- 1 stk. DC Motor med ledninger (1,5–3V) – 201-A
- 1 stk. Liten Servo (0–160°) A0090 - 9 g
- 1 stk. Rele 5–12V maks 5A – JZC-11F
- 1 stk. Temperatur sensor – +10mV/K – 0,5 V ved 0 °C – TMP36GZ
- 1 stk. Bøyesensor, Spectra Symbol 25 kOhm v/flat sensor – FS-L-0055-253-ST
- 1 stk. Membran-potensiometer, 100–10kOhm – SP-L-0050-103-ST
- 1 stk. USB kabel, 6' vanlig A til B USB kabel

- 30 stk. Koblingsledninger, 7” Male/Male
- 1 stk. Fotomotstand, 8 kOhm (10 lux) – 1 MOhm (0 lux) topp ved 540 nm GL5528
- 1 stk. Tri-color LED, Intensitet (RGB): (800, 4000, 900) mcd YSL-R596CR3G4B5C-C10
- 10 stk. Rød lysdiode, 16–18 mA, maks 20 mA, Intensitet 150–200 mcd YSL-R531R3D-D2
- 10 stk. Gul lysdiode16–18 mA, maks 20 mA, Intensitet 40–100 mcd YSL-R341Y3D-D2
- 1 stk. Trimpot med ratt, 10 kOhm
- 1 stk. Magnetisk buzzer 100–10kHz 70–90 dB rel. 20µPa, maks 2048Hz, CEM1203(42)
- 1 stk. Trykkbryter, Big 12mm
- 20 stk. Motstand 330 Ohm 1/6 W
- 20 stk. Motstand 10 kOhm 1/6 W

Mer informasjon komponentene og datablader finnes på nettsidene til Sparkfun:
https://www.sparkfun.com/products/11227

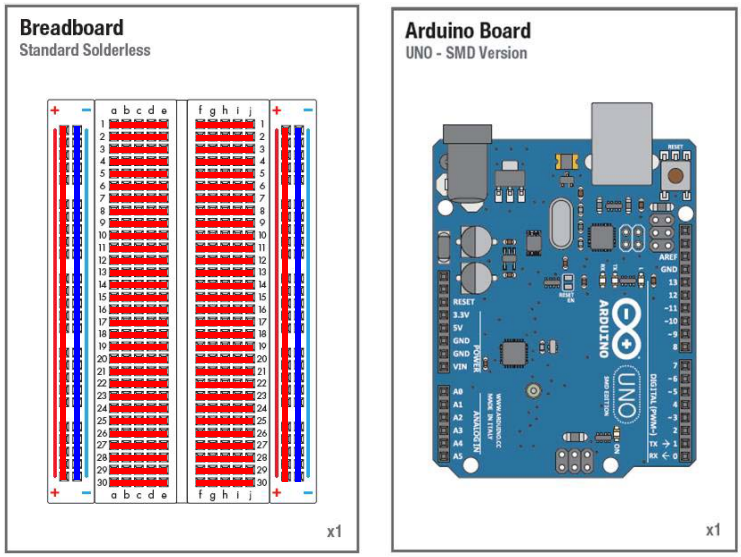
Supplement til Inventors kit

- 1 stk. Display, Adafruit SSD1306
- 1 stk. Trykksensor, BMP180

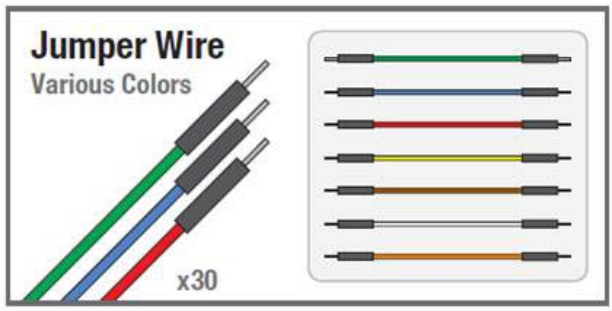
Begge kan kjøpes hos bl.a. www.kultogbillig.no

2.4.2 Koblingsbrettet

Normalt vil Arduino UNO og koblingsbrettet være montert på basisplata. Studer hvordan forbindelseslinjene i koblingsbrettet går.

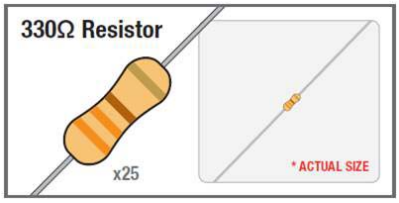


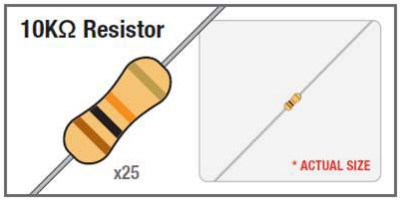
Strekene (røde) på koblingsbrettet viser hvilke koblingspunkter som er koblet sammen på undersiden. To komponentbein som er koblet til samme rad i to av hullene a – e eller f – j er elektrisk sammenkoblet. Komponentene plasseres på koblingsbrettet og forbindes med Arduino-en ved hjelp av jumperer.

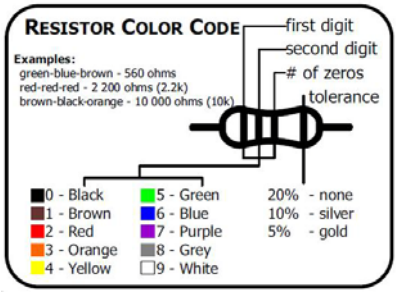


2.4.3 Motstander

Faste motstander







Beskrivelse:

Motstander kommer med mange ulike verdier. Her benyttes kun to: 330 Ohm (oransje, oransje, brun) og 10 kOhm (brun, sort, oransje). Fargekoden bestemmer verdien på motstandene. Det er viktig å velge riktig verdi.

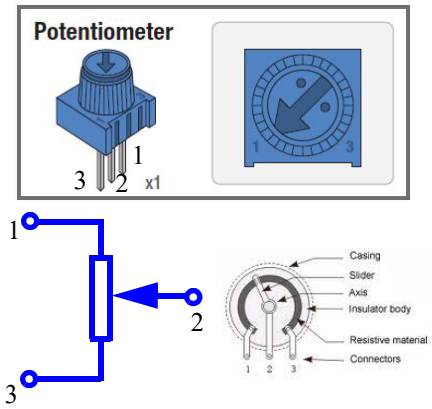
Bruksområder:

Motstander brukes til å begrense strømmen i en komponent, eller for å etablere et spenningsnivå slik som i spenningsdeleren. I følge Ohms lov vil spenningen over en motstand øke proporsjonalt med strømmen. Dette utnyttes når en ønsker å omdanne en varierende motstandsverdi som hos mange sensorer, til en varierende spenning. Det har ingen betydning hvilken vei motstanden kobles.

Bestem verdien:

Fargene på ringene bestemme verdien. Hver farge står for et av sifrene 0 til 9. Hold gullringen til høyre og les av fargene fra venstre mot høyre. Første og andre ring angir første og andre siffer i verdien. Tredje ring angir antall nuller.

Potensiometer



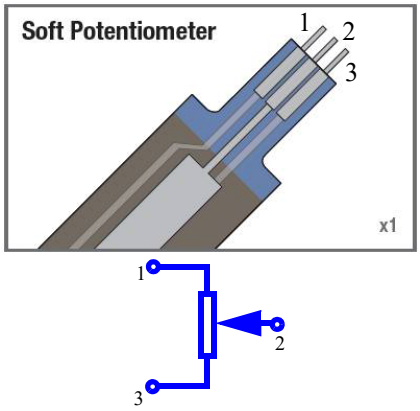
Beskrivelse:

Et potensiometer er en motstand med et variabelt uttak. Ved hjelp av et lite ratt kan uttaket flyttes langs motstanden. På denne måten kan en på pinne 2 ta ut en brøkdel av spenningen mellom pinne 1 og 3.

Bruksområder:

Potensiometeret kan etablere en variabel spenning mellom spenningen på ytterpunktene 1 og 3. Eller fungere som volumkontroll for et signal som sendes inn mellom pinne 1 og 3 og som tas ut mellom 2 og 3.

Trykkpotensiometer



Beskrivelse:

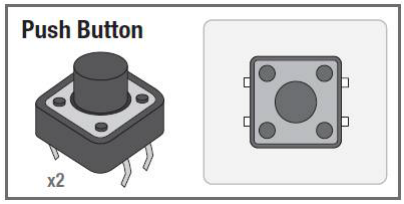
En trykkfølsom motstand fungerer på samme måte som et potensiometer. Istedet for å dreie på en knott, presser man på metallfilmen et sted mellom topp og bunn tilsvarende den spenningen man ønsker at potensiometeret skal gi ut.

Bruksområder:

Jeg har ikke sett slike trykkfølsomme potensiometer brukt andre steder enn i dette settet. Glidepotensiometer med en spak finner man f.eks. i miksebord.

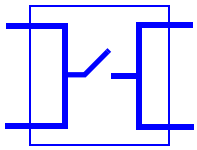
2.4.4 Sensorer

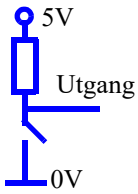
Bryter



Beskrivelse:

Bryterne har fire bein. De er forbundet med hverandre to og to som vist på tegningen under. Et trykk på knappen vil koble de to parene sammen. Forbindelsen opprettholdes så lenge knappen er trykket inn og brytes når den slippes.

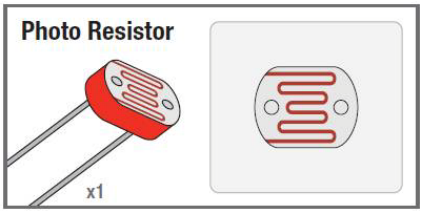




Bruksområder:

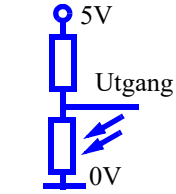
Brytere brukes til å gi en enkel på/av informasjon til kretsen, på samme måte som en lysbryter. For at kretsen skal forstå informasjonen må trykket omdannes til en spenning. Ved å koble bryteren mellom en motstand (10 kOhm) til 5 V og jord vil en på utgangen få en spenning som går fra 5 V til 0 V når bryteren trykkes inn.

Fotomotstand

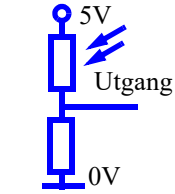


Beskrivelse:

En fotomotstand er en motstand hvor verdien bestemmes av intensiteten på lyset som treffer den. Jo mer den belyses, jo lavere motstandsverdi (mørke gir typisk 300 kOhm, sterkt lys 100 Ohm). Det betyr ingen ting hvilken vei den kobles inn i kretsen.



Høy spenning ut ved mørke



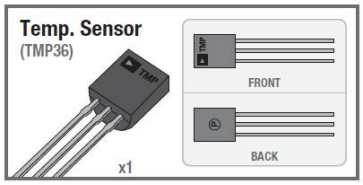
Høy spenning ut ved lys

Bruksområder:

Fotomotstander brukes der en ønsker å styre en funksjon ved hjelp av lysstyrken, som f.eks. tenning av lys når mørket faller på, telleapparater (en lysstråle brytes når noen går gjennom døra) o.l..

Den kobles gjerne i en spenningsdeler. Plasseringen bestemmes av funksjonen. Se figuren til venstre.

Temperatursensor

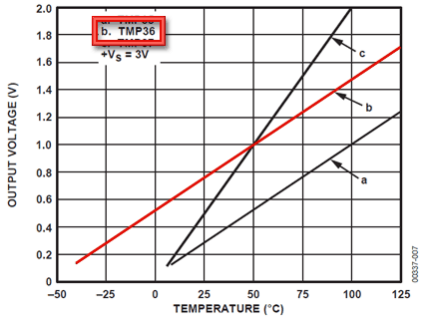


Beskrivelse:

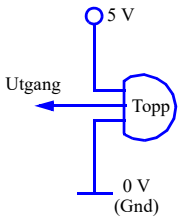
En temperatursensor (TMP36) av denne typen registrerer temperaturen og gir ut en spenning som er proporsjonal med temperaturen. OBS! Unngå å forbytte med transistorene!

Bruksområder:

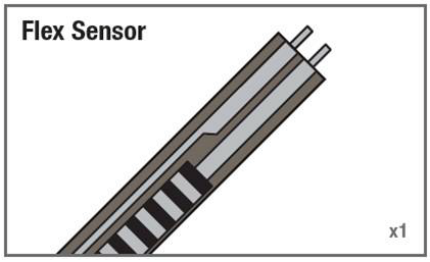
Temperatursensorer brukes i elektroniske termometre eller i termostater for å regulere en varmeovn. Den kan også brukes for å beskytte elektronikk, ved at strømmen brytes når temperaturen overskrider et maksimalt nivå. Hos TMP36 øker spenning med10 mV pr. grad C. Ved 0° C er spenningen 0,5 V.



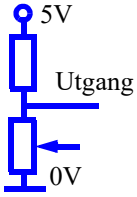
Spenning som funksjon av temperatur (TMP36 rød kurve)



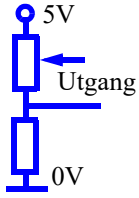
Bøyesensor



Zebrastripenen på motsatt side av trykket



Økende spenning på utgang



Avtagende spenning på utgang

Beskrivelse:

Ved å bøye sensoren vil motstanden endre seg. Bøyes den med sebrastripene på innsiden av bøyen faller motstanden til ca. 25 kOhm. Bøyes den med sebrastripene på utsiden av bøyen øker motstandsverdien til 65 kOhm. Verdiene avhenger av graden av bøyning.

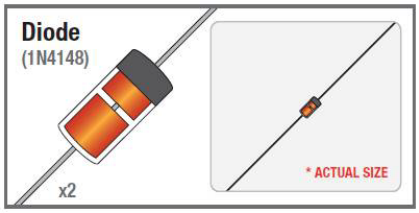
Bruksområder:

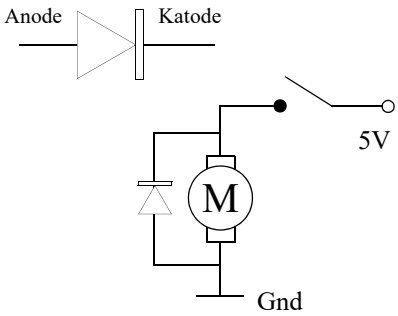
Lignende sensorer brukes i mange sammenhenger for å måle strekk eller sammentrykning i et materiale. Slik deformering kan f.eks. skyldes belastning eller nedbøyning. I denne sammenhengen går en slik sensor under betegnelsen *strekklapp*.

2.4.5 Halvledere

I denne sammenhengen er dette dioder og transistorer.

Diode





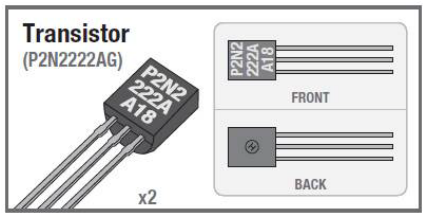
Beskrivelse:

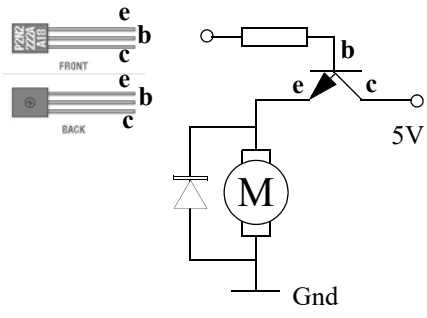
Dioden er en halvleder som kun leder strøm en vei, fra anode til katode, dvs. i pilens retning.

Bruksområder:

I denne sammenhengen skal vi benytte dioden som en "fly back" diode for å kortslutte strømmer som skyldes at strømmen i en spole, i en motor eller et rele skal ødelegge transistoren. Den må derfor montere slik at den normalt stenger for strømmen som normalt skal gå gjennom motoren. Når strømmen i motoren brytes, oppstår en motspenning som forsøker å hindre at motoren stopper. Denne motspenningen kan være så stor at den ødelegger transistoren. Dersom vi kobler en diode som vist på figuren til venstre, vil motspenningen kortsluttes gjennom dioden slik at den ikke når transistoren og ødelegger den.

Transistor





Beskrivelse:

Transistoren har tre terminaler (bein), Fra collector (c) til emitter (e) kan det gå en relativt stor strøm. Størrelsen på collectorstrømmen kan styres av spenningen mellom basen (b) og emitter (e). Når den blir stor nok begynner det å gå en strøm i transistoren.

Bruksområder:

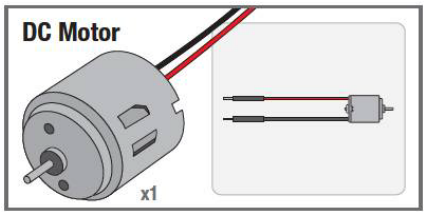
En transistor har gjerne to bruksområder. Som forsterker når basespenningen/strømmen varierer innen et lite område. Som bryter når basespenningen/strømmen er så stor at den slår transistoren på eller av.

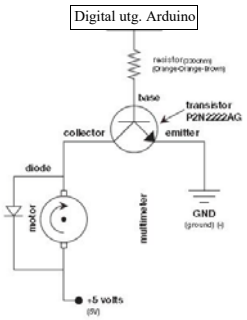
Transistorer som signalforsterker brukes i elektroniske forsterkere, radiosendere og mottakere, TV-er osv. I datamaskiner og i styringssystemer brukes den som bryter.

2.4.6 Aktuatorer

En aktuator er en komponent som kan utføre en handling, enten det er å skape mekanisk bevegelse (motorer, pumper, magneter, releer), gi lyd (øretelefoner, høyttaler, sirene) eller lys (LED, lyspærer) eller varme opp en gjenstand eller et rom (glødetråd).

Motor





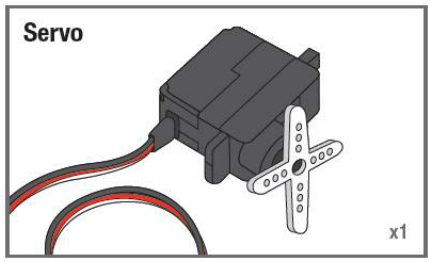
Beskrivelse:

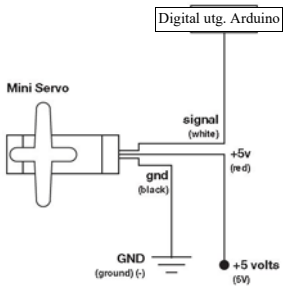
Akslingen begynner å rotere når motoren tilføres en spenning over 3 V. Motoren som følger med kitet kan brukes på spenninger opp til 40 V og strømmer på opp mot 200 mA. Siden Arduino'en ikke klarer å styre så store strømmer, benytter vi en transistorbryter som tåler strømmen. Legg merke til "fly back" dioden over motoren. Denne skal hindre overspenning på transistoren idet strømmen brytes for å stoppe motoren.

Bruksområder:

I dag brukes elektriske motorer til det meste der noe skal gå rundt eller bevege seg. Det være seg elektriske kjøkkenartikler, datadisker, leketøy som skal bevege seg, vaskemaskiner, pumper, vifter, vindusviskere og etterhvert elektriske biler.

Servo





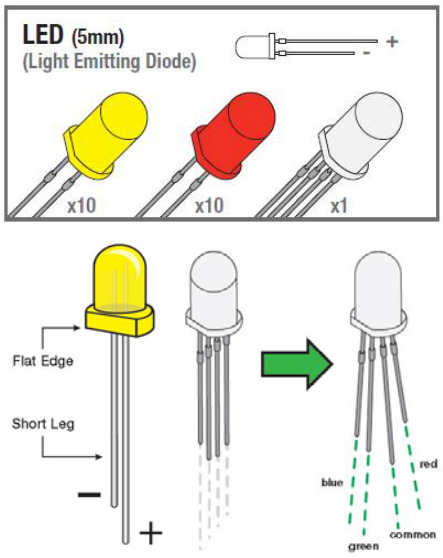
Beskrivelse:

En servo er en slags motor som kan dreie akslingen en bestemt vinkel eller rotere med en definert fart. Denne servoen kan på kommando fra mikrokontrolleren dreie akslingen en vinkel på fra 0 - 180°. Dreiningen skjer ved at servoen mottar pulser, hvor lengden av pulsen bestemmer dreievinkelen. En pulslengde på 1,5 ms gir en vinkel på 90°. Pulsene må gjentas med jevne mellomrom omtrent som man pulsbreddemodulerer et signal. Vi kobler derfor styreinngangen til servoen til en utgang som har denne funksjonen, f.eks. port 9.

Bruksområder:

Servoer brukes ofte i modellfly for å styre side- og høyderor og flaps. De er også en viktig komponent i mange roboter hvor som skal bevege en arm e.l.

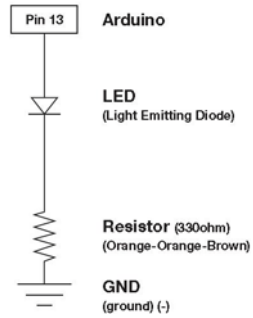
Lysdioder



Beskrivelse:

Som navnet tilsier leder lysdiodene strøm bare den ene veien, fra anoden til katoden. For at den skal lyse må den kobles opp rett vei. Når strømmen i dioden overstiger ca. 1,5–2 mA, begynner den å lyse svakt. Lysstyrken øker etter som strømmen øker. For å begrense strømmen, kobles den gjerne i serie med en motstand på 220–330 Ohm. Uten serie-motstand er det stor sannsynlighet for at dio-den går i stykker.

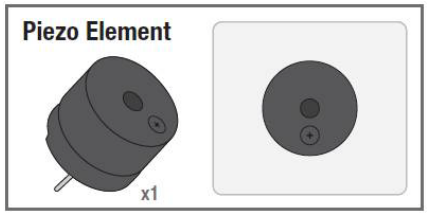
I settet er det vedlagt røde, grønne og gule lysdioder. I tillegg finnes en RGB-diode hvor en rød-, en grønn og en blå lysdioder montert i samme kapsel.

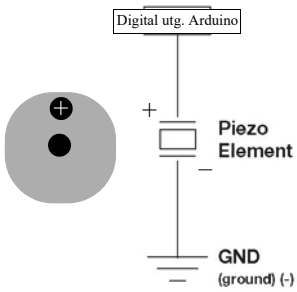


Bruksområder:

Lysdioder eller LED (Light Emitting Diode) brukes til signallamper, displayer og nå også i stor grad til belysning.

Buzzer





Beskrivelse:

Det piezo-elektriske elementet er et krystall som trekker seg sammen når det påføres en spenning og det høres et klikk. Når spenningen forsvinner, vil krystallet få tilbake sin opprinnelige form og det høres et nytt klikk. Ved å la spenningen variere fort kan man høre en lyd som i en høyttaler. For at den skal fungere rett, må + og – kobles rett.

Bruksområder:

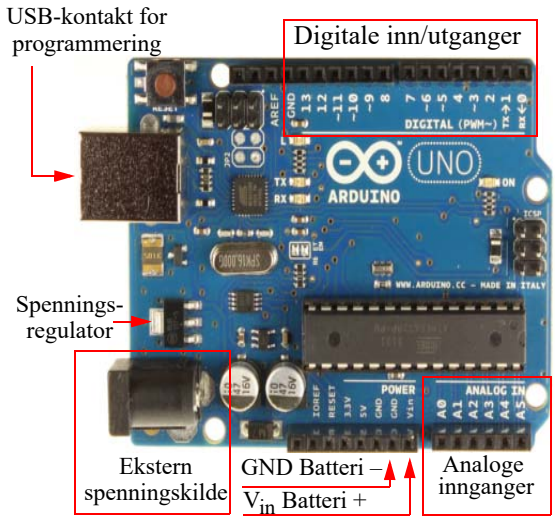
Piezo-elektriske elementer brukes for å lage lyd og toner med forskjellig frekvens. Den kan også brukes i alarmer som f.eks. røykvarslere. En *buzzer* består av et piezo-element og en enhet som lager en varierende spenning. Buzzeren trenger derfor bare en spenning for å gi en tone. Tonen er ofte fast.

2.4.7 Arduino – mikrokontrollerkortet

Arduino UNO-kortet er databehandlingsenheten med sine inn- og utganger for sensorer og aktuatorer. Den har også en intern timer og internt lager for program og data.

I tillegg har kortet en USB-inngang for å legge inn programvare og en plugg for batteri eller batteriadapter.

Det Arduino-kortet som følger med settet er Arduino UNO R3, som er ett av de mest populære mikrokontrollerkortene i Arduino-familien, og dermed leveres til en overkommelig pris (KultOgBillig.no pris kr. 129,- inkl. MVA + frakt eller ELFA pris kr. 207,5,- inkl. MVA)



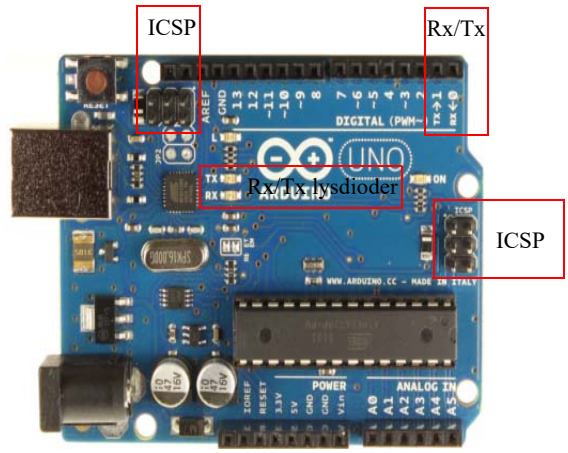
Kortet er bygget opp omkring Atmel mikrokontrolleren ATmega328P med en klokkefrekvens på 16 MHz og et flash lager på 32 kbyte, SRAM 2 kbyte og EEPROM 1 kbyte.

Kortet har dessuten følgende inn- og utganger:

- **Digitale I/O-porter**
  Kortet har 14 digitale inn/utporter (I/O-porter) som kan programmeres til enten å være en inn- eller en utgang. Seks av disse (3, 5, 6, 9, 10 og 11) kan *pulsbreddemoduleres* (pwm), disse er merket med ~ på kortet. Maksimal strøm på hver av I/O portene er 40 mA.
- **Analoge innganger**
  Kortet har 6 analoge innganger som kan tilføres spenninger fra 0 – 5V. Spenninger ut over 5V vil ødelegge mikrokontrolleren.
- **USB-kontakt** for direkte tilkobling av PC og for programmering av kortet og overføring. I tillegg kan data overføres mellom monitoren på PC-en og kortet. Under programmeringen tilføres kortet spenning fra USB-kontakten. Dersom denne belastes med mer enn 500 mA vil strømforsyningen bli brutt inntil strømtrekket reduseres under denne grensen. USB3 kan levere vesentlig høyere strøm.
- **Strømtilførsel**
  Tilkoblingspunkter for batterieliminator anbefalt spenning 7 – 12 V (grenseverdier 6 – 20 V). Batteri kan enten tilkobles eliminatorpluggen (2.1 mm + senter) eller via V_{in} (+) og GND (–). 5V utgangen lever spenning til f.eks. datainnsamlingskortet. Enkelte komponenter trenger lavere spenning som ev. kan leveres fra 3.3 V utgangen (f.eks. BMP180).
- **Reset**
  Kortet inneholder en RESET-knapp som resetter programmet og starter det på nytt.

Kantkontaktene er montert slik at ulike tilleggskort ("shield card") kan monteres rett ned på Arduino-kortet.

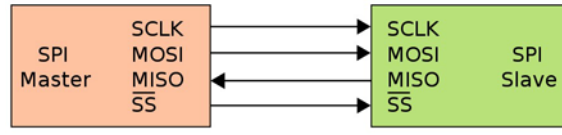
Kortet støtter ulik typer datakommunikasjon med omverdenen.

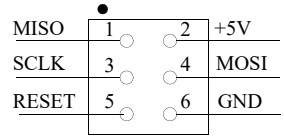


- **Rx/Tx**
  Kortet støtter UART seriell datakommuniaksjon via Rx og Tx portene (I/O-port 0 og 1). Det er også disse som benyttes for programmering av kortet. De er også tilkoblet to lysdioder som vil blinke når kortet kommuniserer med USB/PC. Tilsvarende vil skje når data overføres til programeditoren for monitorering av data på PC-skjermen.

- **I^{2}C-databus**
  I ^{2}C står for Inter IC-bus, og er ment å være akkurat det da den ble utviklet av *Philips Semiconductor* tidlig på 80-tallet. Bussen er svært enkel med sine to linjer (klokke og datalinje) i tillegg til spenning og jord. Videre har hver krets langs bussen en unik adresse som enten er satt av fabrikken eller kan settes med strap'er på brikken. Bussen er dessuten utstyrt med kollisjonsdeteksjon[1]. I starten var den definert med en hastighet på 100 kb/s. Senere, etter som en trengte raskere dataoverføring, er *Fast mode* – 400 kb/s og High speed – 3,4 Mb/s definert.

- **SPI-databus**
  Seriel Peripheral Interface buss (SPI-bus). Er en firelinjers databuss brukt for å overføre data til utstyr utenfor mikroprosessoren (periferutstyr). Databussen ble opprinnelig utviklet av Motorola. I en slik dataoverføring fungerer den ene modulen som master (sjefen), og den andre som slave (tjener). På figuren over er enheten til venstre master, og enheten til høyre slave. Master tar initiativet til dataoverføringen ved hjelp av SS-signallinjen, og sender sine data på linjen MOSI (Master Output Slave Input) linjen, mens slaven mottar på sin MOSI linje. Master mottar så svar på sin forespørsel til slaven som sender data tilbake på linjen MISO (Master Input Slave Output) som mottas av master på linjen MISO (Master Input Slave Output). SCLK er klokkesignalet som bestemmer takten til dataoverføringen. SS signalet (Slave Select) varsler slaven om at nå ønsker master kontakt. Dersom det er flere slaver i systemet trengs flere SS-linjer, en til hver slave[2].





1. For mer informasjon se: http://www.i2c-bus.org/

For den interesserte så kan kretsskjema for Arduino UNO hentes fra følgende nettadresse: http://arduino.cc/en/uploads/Main/Arduino_Uno_Rev3-schematic.pdf.

For mer informasjon om Arduino UNO R3 se: http://arduino.cc/en/Main/ArduinoBoardUno/

2. http://en.wikipedia.org/wiki/Serial_Peripheral_Interface_Bus

3 Programmering

Arduino programmeres normalt i en variant av C++, men kan også programmeres med blokkode, som f.eks. Blockuino, som er utviklet og driftes av ***Joachim Haagen Skeie*** ved Kodegenet AS. I dette kapittelet skal vi først installere programeditoren og vise hvordan den brukes. Dernest skal vi ganske kort foreslå en måte å strukturere CanSat-programmet på, for deretter å gi en oversikt over noen viktige og anvendelige kommandoer.

3.1 Installasjon av programvare

Det er en programpakke som skal installeres: Arduino programeditor, IDE.

3.1.1 Arduino programeditor, IDE

Nedlasting av programvare

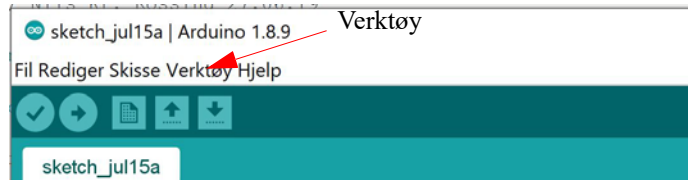
Arduino programeditor og kompilator hentes fra:

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

Aktuell versjon er 1.8.9, og filen som har navnet *arduino-1.8.9-windows* er pakket som en zip-fil og er på ca. 86 Mbyte. En tilsvarende fil er tilgjengelig for Mac fra samme nettsted.

Installasjon av programvaren:

1. Klikk på fila **arduino-1.8.9.zip**
2. For å pakke ut fila trenger du programmet WinRAR som kan hentes fra: http://www.rar-lab.com/download.htm eller Winzip som kan hentes fra: www.winzip.com/downwz.htm
3. Velg *Extract to* fra menylinjen øverst og velg f.eks. C:/Programfiler og trykk OK. Programfilene legges da i en egen underkatalog (arduino-1.8.9) i katalogen *Programfiler*.
4. Programmet startes ved å klikke på programikonet:  .
5. Koble USB-kabelen til ønsket port.
6. Klikk på *Verktøy (Tools)* på menylinjen og velg *Kort*. Her velges hvilken variant i Arduino-familien du ønsker å jobbe med. I dette tilfellet velg: *Arduino/Genuino UNO*.



7. 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 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 avslørt. Det er ikke nødvendigvis alltid der feilen befinner seg.

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

Kort oversikt over Arduino-editoren

Man finner følgende kommandoikoner på den grafiske menylinjen:

 Kompiler og verifiser at koden er riktig

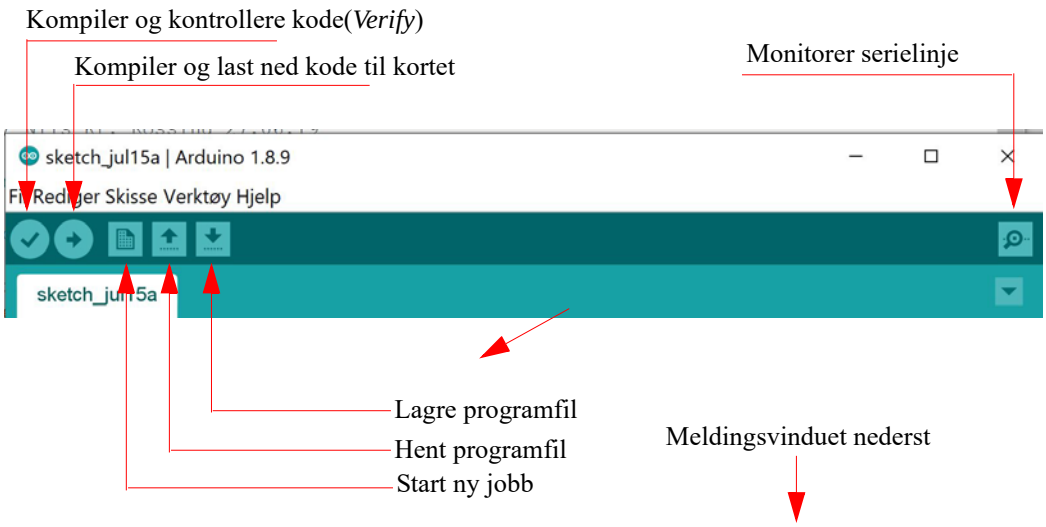
 Kompiler og last ned programmet til kontrollenheten

 Hent nytt "arbeidsark"

 Hent en eksisterende programfil

 Lagre programfil

 Monitorer data sendt tilbake fra kontrollerkortet på serielinjen



Manglende kontakt med kortet

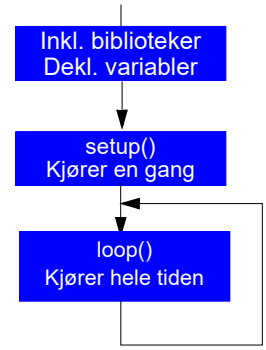
Det hender at en ikke oppnår umiddelbar kontakt med Arduino-kortet når en forsøker å laste ned et program. Feilmeldingen: avrdude: stk500_getsync(): not in sync: resp=0x00 i meldingsvinduet betyr at det ikke oppnås kontakt med kortet. Dette kan skyldes flere ting:

- Kabelen er ikke tilkoblet, eller ødelagt
- Feil port er valgt av programeditoren
  Endres ved å velge: *Verktøy* og *Port* fra menylinjen i editoren
- Feil type kontroller-kort valgt
  Endres ved å velge: *Verkstøy* og *Kort* fra menylinjen for så å velge rett kort, i vårt tilfelle *Arduino/Genuino UNO*.
- Manglende drivere, i så fall må driverne installeres manuelt

3.2 Programstruktur

Alle Arduino-programmer består av to hovedfunksjoner:

- **void setup()** som kjøres bare *en* gang hver gang programmet starter. Dette skjer etter at programmet er lastet opp, når man trykker på restartknappen eller åpner monitoren.
- **void loop() er en funksjon** som gjentas så lenge Arduino'en har spenning.
- **Biblioteker og deklarasjon** av globale variable plasseres gjerne helt i starten.
- Det er også vanlig å skrive ***egne funksjoner*** som gjerne legges på slutten av programmet, utenfor og etter loop-funksjonen(), men kan i prinsippet legges hvor som helst utenfor setup() og loop() funksjonene.

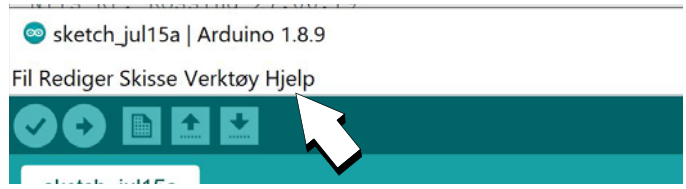


3.3 Viktige kommandoer

Referansemanualen til Arduino C++ for bruk ved programmering av Arduino-prosessorer finnes på følgende nettadresse:

http://arduino.cc/en/Reference/HomePage

Referanse manualen ligger også under *Hjelp* på menylinjen i programeditoren som vist på figuren til høyre.



3.3.1 Generelle kommandoer

Programstruktur

Som nevnt foran så består programmet hovedsakelig av to funksjoner omsluttet av *klammeparenteser*. I void setup()-funksjonen initieres mikrokontrolleren, mens selve programmet legges under void loop()-funksjonen som vist under.

```
void setup()
{
    <initiering>
}
void loop()
{
    <programkode>
}
```

Alle linjer må avsluttes med; (semikolon)

Initiering av dataoverføring til PC

Under uttestingen kan det være praktisk at data leses tilbake til terminalen. Datahastigheten settes opp i setup-funksjonen med kommandoen: Serial.begin(9600); her satt til 9 600 baud (ca. 9600 tegn pr. sekund) som vist under:

```
void setup()
{
    Serial.begin(9600);
}
```

Kommentarer:

Kommentarer kan skrives hvor som helst og begynner med

```
// Dette er en kommentar
```

Disse blir fjernet under kompilering og overføres ikke til mikrokontrolleren. Kommentarene er derfor kun en hjelp for den som skriver programmet, ev. andre som skal lese og forstå programmet senere.

Deklarasjon av lokale og globale variable:

I programspråket C må alle variable deklareres før de kan brukes. Deklarasjonene må inneholde *type* og *navn* på variabelen og gjøres gjerne i starten av programmet før setup() rutinen. Siden den deklareres utenfor funksjonene, så blir de ***globale***, dvs. at variablene kan brukes i alle funksjonene og beholder innholdet i variabelen.

Deklarering kan også gjøres *innenfor hver* funksjon. Slike variable gjelder da bare innenfor den funksjonen og kalles da ***lokale***. Under er vist deklarasjon av de vanligste typer variabler. I dette tilfelle vil de bare gjelde innen funksjonen loop():

```
void loop()
{
    Int a;              // deklarasjon av 16 bit heltallsvariabel (word)
    char b;             // deklarasjon av 8 bit karaktervariabel (byte)
    char c, d;          // deklarasjon av to 8 bits karaktervariabler (byte)
    float e;            // deklarasjon av variabelen e som et desimaltall f.eks. 1,65 (32 bit, dobbel word)
    unsigned long f;    // deklarasjon av 4 byts heltallsvariabel f (32 bit) uten fortegn
```

```
    boolean g;          // deklarasjon av en boolsk variabel g som kan ha verdiene 0 eller 1
    <dernest følger programkoden>
}
```

Skriv tilbake til PC skjerm:

Følgende kommandoer skriver en variabel eller en tekst tilbake på terminalvinduet i programeditoren.

```
Serial.print(a);            // Skriver variabelen a til en linje på skjermen,
                            // neste skrivekommando skriver på samme linje
Serial.println(a);          // Skriver variabelen a til en linje på skjermen og skifter linje,
                            // neste skrivekommando skrives derfor på ny linje
Serial.println("Hallo");    // Skriver teksten Hallo til en linje på skjermen,
                            // neste skrivekommando skriver på ny linje
```

Det er også mulig å kombinere tekst og variabler i samme printkommando:

```
Serial.println("Trykk:", a);   // Skriver teksten Trykk: til en linje på Arduino monitoren,
                               // etterfulgt av innholdet i variabelen a, og skifter deretter til ny linje
Serial.println(f, 2);          // Skriver desimalvariabelen f til terminal på PC med to desimaler
```

Under er vist et eksempel på utskrift av data brukt i en CanSat. På CanSat-kortet har man montert en SD-kort holder og radio, for enten å lagre innsamlede data på kort eller sende ut på radio. I så fall kan man anvende de samme kommandoene.

Dersom man ønsker å skrive rådata til monitoren, til en fil på SD-kortet eller til radioen, kan det være greit å skille de ulike variable fra hverandre. I tillegg kan det være lurt å legge inn en signatur i starten og et tall som angir tid fra oppstart. Som for eksempel:

```
Serial.print("Signatur; ");
Serial.print(millis());                    // Skriver ut antall millisekunder siden Arduino'en ble startet
Serial.print("; ");                        // Sett inn semikolon mellom hver verdi
Serial.print(temperatur_LM35_raadata);     // Skriv inn verdien i variabelen temperatur_LM35_raadata
Serial.print("; ");
Serial.print(temperatur_NTC_raadata);
Serial.print("; ");
Serial.print(trykk_raadata);
Serial.print("; ");
Serial.print(akselerator_x_raadata);
Serial.print("; ");
Serial.print(akselerator_y_raadata);
Serial.print("; ");
Serial.print(akselerator_z_raadata);
```

Definer digitale porter som inngang eller utgang:

En port er et fysisk tilkoblingspunkt på kretsen som kan kobles til eksterne kretselementer, sensorer eller aktuatorer f.eks. LED. Disse kan enten være analoge eller digitale, innganger eller utganger. Siden en strømkrets må være sluttet, vil alle porter forholde seg til jord på kortet (dvs. minus på batteriet).

Kontrolleren ATmega 328 (Arduino UNO) har en rekke *porter*, 14 digitale (0 – 13) og 6 analoge (0 – 5). De digitale portene kan både være innganger eller utganger og hver port må derfor defineres som en inn- eller utgang. Dette gjøres i setup()-funksjonen:

```
void setup()
{
    pinMode(8,OUTPUT);          // Definerer pinne 8 som utgang
    pinMode(7,INPUT);           // Definerer pinne 7 som inngang
    pinMode(6,INPUT_PULLUP);    // Definerer pinne 6 som inngang med pullup-motstand, dette er nyttig
                                //  for at inngangen ikke skal henge fritt (sveve)
                                //  når den ikke er tilkoblet noe
}
```

Dersom en inngang defineres med *pullup-motstand* betyr det at utgangen er koblet til pluss på batteriet ved en "stor" motstand internt i kretsen. Det betyr at dersom porten ikke er tilkoblet noe annet på utsiden så vil den ha verdi 1 (dvs. 5V).

Lese og skriv til en digital port:

Digitale porter som er definert som utganger, kan enten settes til høy eller lav spenning (5V eller 0V). Digitale porter som er definert som innganger, kan lese av om spenningen på porten er høy (5V) eller lav (0V). Dette gjøres med følgende kommandoer:

```
boolean bolsk;          // Definerer den boolske variabelen bolsk kan ha verdien 0 eller 1
int heltall;            // Definerer en heltals variabel heltall, kan meget vel brukes for 0 og 1
void loop()
{
digitalWrite(8, HIGH);  //Setter port 8 høy (5 V)
digitalWrite(8, LOW);   //Setter port 8 lav (0 V)
bolsk = digitalRead(7); // Leser den digitale verdien på port 7 og setter i variabelen bolsk
heltall = digitalRead(6); // Leser den digitale verdien på port 6 og setter i variabelen heltall
}
```

Det er imidlertid vanligere å bruke heltallsvariable for å holde digitale verdier lest fra innganger. Man sparer imidlertid litt lagerplass dersom man bruker boolske variabler.

Vent-kommando:

Dersom vi ønsker at programmet skal ta en pause kan vi skrive følgene:

```
delay(1000);                //Stopper programmet i 1000 msek (1 sek)
delayMicroseconds(100); //Stopper programmet i 100 µs, 0,1 msek (0,0001 sek)
```

Aritmetiske operasjoner

```
sum = a + b;            // Summen av a + b settes i variabelen sum
diff = a - b;           // Differansen av a – b settes i variabelen diff
prod = a * b;           // Produktet av a * b settes i variabelen prod
kvo = a / b;            // Koeffisienten av a / b settes i variabelen kvo
```

Variabelnavnene *sum, diff, prod* og *kvo* er bare valgt som eksempler.

Dersom man ønsker å endre innholdet i en av variablene, kan dette gjøres slik:

```
a = a + b;                  // Summen av a + b settes i variabelen a
```

Dette ser ikke lengre ut som en fornuftig "ligning" slik vi kjenner den fra matematikken. Her legges verdien i *a* til *b* før den så legges tilbake til *a*.

Les verdi fra en analog inngang og skriv til monitor (PC)

Syntaksen for lesing fra en analog inngang kan skrives som:

```
<variabel> = analogRead(<analog port>);   //Den analoge porten kan ha verdiene 0 til 5 V hos UNO
```

Eksempel 1:

```
Int verdi;                          // Deklarerer variabelen verdi
verdi = analogRead(0);              //Digitale verdien fra analog inngang 0 leses inn i variabelen verdi
```

Eksempel 2:

```
void loop()
{
    int pressure;                   //Deklarerer pressure som en heltalls-variabel
    pressure = analogRead(1);       //Leser av trykksensoren på AD-kanal 1
    Serial.println(pressure);       //Skriv resultatet tilbake til monitoren (PC)
}
```

Legg merke til *Serial.print();* skriver *uten* påfølgende linjeskift, mens *Serial.println();* skriver *med* påfølgende linjeskift.

Eksempel 3:

```
void loop()
{
    int temperature;                //Deklarerer temperatur som heltallsvariabel
    temperature = analogRead(5);    //Leser av temperatursensoren på AD-kanal 5
    Serial.println(temperature,2);  //Skriv resultatet tilbake til Arduino monitoren (PC) med 2 desimaler
}
```

De analoge portene er på innsiden tilkoblet en analog til digital omvandler (AD-konverter) som gjør om spenningen 0 – 5V til en digitalverdi på 0 – 1023 (0000000000 – 1111111111). Dvs. AD-konverteren har 10 bits nøyaktighet.

For-loop – For å frambringe mange like operasjoner (sløyfer)

Ofte kan det være aktuelt å gjenta den samme operasjonen mange ganger etter hverandre, kanskje med ganske små forandringer mellom gjentagelsene. Da er det på sin plass å bruke en *for-loop*. Denne skrives slik:

```
for(int x = 0; x < 100; x++)
{
// Her skrives koden som skal gjentas
}
```

x er en heltallsvariabel (int x) som brukes som teller. Denne starter på verdien 0 (int x = 0;) økes med 1 (x++) for hver runde i loopen og stopper ikke før den når opp til 100 (x < 100;). De ulike utrykkene skilles med semikolon.

Eksempel:

```
for(int x = 0; x < 100; x++)
{
Serial.println(i);
}
```

Denne skriver ut tallene 0 – 99 til monitoren, ett tall for hver linje.

If-setning – For å kunne gjøre "veivalg" i programmet

Noen ganger ønsker vi å gjøre forskjellige ting på bakgrunn av ulike betingelser, da kan vi bruke *if-setninger*. Igjen tar vi et konkret eksempel:

```
if (i < 10)
{
   // Gjør dette dersom innholdet i variabelen i < 10
}
else if (i == 10)
{
  // Gjør dette dersom innholdet i variabelen i = 10
}
else
{
// Gjør dette dersom noe annet er tilfelle i dette tilfellet i > 10
}
```

Avhengig av verdien til variabelen *i* utfører programmet ulike operasjoner. Parentesen etter if() skal inneholde en betingelse, denne kan være enkel, som her, eller ganske sammensatt. Man bruker da *logiske operatorer* for å undersøke om noe er større enn (>), mindre enn (<) eller lik (==). Legg merke til det doble likhetstegnet.

Man kan lage flere betingelser med ulik respons ved å bruke en eller flere *else if()* med nye betingelser. Men alt som faller utenfor de definerte betingelse kan samles opp i en *else*.

Legg merke til at linjen, *if (i < 10)* ikke avsluttes med semikolon men etterfølges av { }. Dvs. at if() er egentlig en funksjon. Heller ikke klammeparentesene etterfølges med ; (semikolon). Slik er språket definert.

3.4 Bruk av I^{2}C og biblioteker

I dette avsnittet skal se hvordan vi kan bruke I^{2}C buss til å overføre data fra sensorer utstyrt med denne typen kommunikasjonsgrensesnitt, i vårt tilfelle trykkmåleren BMP180 og displayet SSD1306. Til dette trenger vi som oftest spesiallagede biblioteker som må installeres dersom de ikke alt finnes blant bibliotekene som følger med Arduino editoren (IDE).

Den så kalte I^2C bussen består at to kommunikasjonslinjer som gjør det mulig å sende data etter hverandre (på serieform) mellom ulike kretser. Se side 49 for litt mer beskrivelse.

Standardbiblioteket for I^2C følger med Arduino programvaren, men krever at man inkluderer "header"-filen: "wire.h" øverst i koden ved å skrive:

```
include <wire.h>
```

Dette medfører at programmet får tilgang til en del funksjoner knyttet til kommunikasjon via I^2C-kommunikasjonslinjene.

For mange spesiallagde kretser finnes det spesielle pakker med funksjoner som gjør dem lettere å bruke. Slike biblioteker må gjerne hentes ned fra leverandøren eller andre og installeres i program-editoren. I neste avsnitt skal vi se hvordan vi kan laste ned og installere biblioteker.

3.4.1 Installasjon av pakkede biblioteker

Som eksempel bruker vi biblioteket som trengs for å lese data fra trykksensoren BMP180 som også inneholder en temperatursensor.

1. Last ned biblioteket i pakket form (*.zip). Denne pakken inneholder header-filer (*.h), biblioteksfunksjoner (*.cpp) og gjerne eksempler på bruk av sensoren. Er man heldig inneholder den også en beskrivelse av biblioteksfunksjonene. Man finner gjerne biblioteket på nett ved søke etter den aktuelle kretsen etterfulgt av ordet *library* om adressen ikke er kjent.

   Fra og med Arduino IDE versjon 1.5 kan man fra menylinjen velge:
   Sketch/Include library/Manage Libraries
   og skrive inn den aktuelle sensoren i søkefeltet. Man får da beskjed om biblioteket er installert eller forslag til hvor man kan hente det fra.

2. **Last ned biblioteket fra ekstern kilde:**
   Dersom biblioteksfilen for komponenten ikke finnes, kan man hente biblioteksfilene til BMP180 fra ulike steder, f.eks. fra SparkFun's hjemmeside:

   https://learn.sparkfun.com/tutorials/bmp180-barometric-pressure-sensor-hookup-/installing-the-arduino-library

   Vi har også lagt den ut under den blå hefteserien

3. **Installer biblioteket:**
   Dernest installeres biblioteket i Arduino-programvaren ved å velge:
   *Sketch/Include Library/Add *.ZIP Library* for så å velge den nedlastede zip-pakkede filen.
   Biblioteksfilene blir dermed lagt under *Sketch/include library* og eksempler under: *File/Eksempler*

4. **Inkluder headerfilen (*.h) i programmet**
   Det aktuelle biblioteket inkluderes i programmet ved å velge:
   Sketch/Include Library/<aktuelle biblioteket>
   Dermed er header-filen på plass i programmet.

3.4.2 I²C – Inter IC bus

Også I²C bussens bibliotek må lenkes inn i koden når den skal brukes. Bussen funger ved at en "master" styrer kommunikasjonen. Vanligvis er dette Arduino mikrokontrollerkortet som automatisk blir definert som master ved kommandoen *Wire.begin();* kommandoen i setup-funksjonen.

Biblioteket – Wire.h:

Biblioteket inneholder noen funksjoner som vi skal forsøke å beskrive her:

"Header file":

```
#include <Wire.h>
```

"Header" kommandoen knytter I²C-biblioteket til programmet slik at kompilatoren vet hvilket bibliotek funksjonene skal hentes fra.

```
Wire.begin();
```

Denne funksjonen definerer Arduino mikrokontrolleren som master for kommunikasjonen, dvs. at all kommunikasjon med sensorer og andre periferikretser som er koblet til I²C-bussen styres av Arduino'en. Siden Arduino'en er masteren, er det ikke nødvendig å definere en adresse (`Wire.begin(<adresse>);`) for denne. Ev. kan adressen være et tall mellom 0 – 127 som identifiserer masteren på bussen.

```
Wire.beginTransmission(<adresse>);
```

Med funksjonen *`Wire.beginTransmission(<adresse>);`* kan man starte en dataoverføring til en enhet på bussen med den spesifiserte adressen.

Vi skal senere se hvordan vi anvender dette i praksis.

4 Oppgavesamling

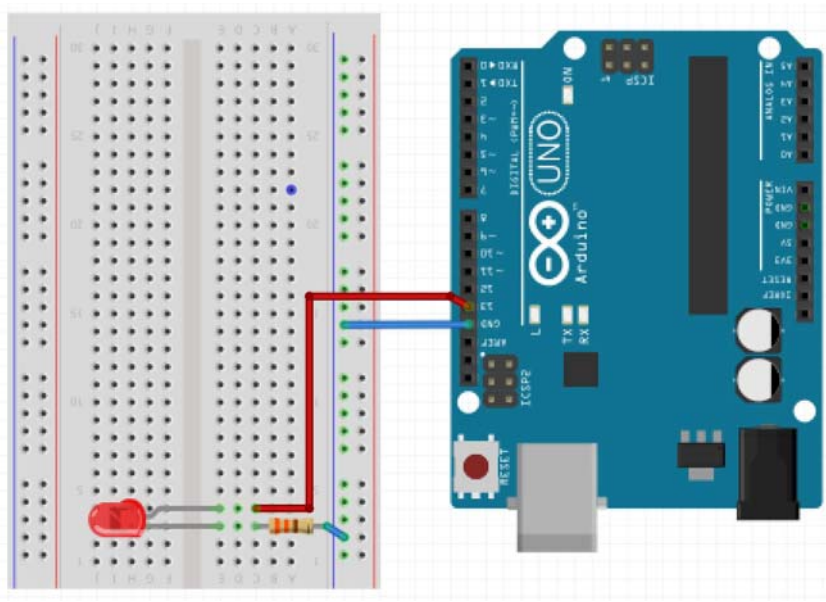
Gjennom tre øvinger skal vi gradvis bygge opp en kunnskap om Arduino-programmering. Oppgavene er valgt dels som en introduksjon og som en forberedelse til CanSat-kurs.

4.1 Øving 1 – Blinkende lysdiode, nedtelling og display

Dette er oppgaver for den som er helt ny med hensyn til programmering og vil egne seg som introduksjon for både lærere og elever.

Det skal bygges opp og programmeres en blinkende lysdiode.

Bygg følgende krets:



1. Skriv inn følgende programkode i programeditoren:

```
void setup                // Setup() funksjonen kjøres bare en gang ved oppstart
{
   pinMode(13, OUTPUT);   // Port 13 defineres som utgang
}

void loop()
{
   digitalWrite(13, HIGH);// Slå på LED
   delay(1000);           // Vent ett sekund
   digitalWrite(13, LOW); // Slå av LED
   delay(1000);           // Vent ett sekund
```

```
}
```

2. ***Last opp programmet og test at det fungerer.***

3. ***Studer koden***

```
//                          – Alt som står bak // er kommentarer
pinMode(13, OUTPUT);        – Port nr. 13 programmeres til å være en utgang
delay(1000);                – Programmet stopper i 1000 millisekunder (1 sek.)
digitalWrite(13, HIGH);     – Sett port 13 til 5V, slå på lyset
digitalWrite(13, LOW);      – Sett port 13 til 0V, slå av lyset
```

4. ***Øk blinktakten***

   Øk blinktakten til ett blink i sekundet

5. ***Skriv til monitoren***

   Monitoren er et vindu som viser informasjon som sendes fra programmet som kjører i Arduino'en og tilbake til PC'en. Monitoren åpner du etter at programmet er lastet over i Arduino'en. Når du åpner monitoren restartes programmet.

   Skrivekommandoer.

   For å kunne skrive informasjon tilbake til PC'en må det legges inn skrivekommandoer i programmet. Legg inn følgende i setup()-funksjonen. Skrives inn mellom klammeparentesene:

```
Serial.begin(9600);       // Setter kommunikasjonshastigheten til 9600 tegn i sekundet
```

   Du skal nå skrive "ON" til monitoren når lysdioden slås på og "OFF" når lysdioden slås av. Skrivekommandoen er slik:

```
Serial.println("ON");   // Skriver ON til monitoren
Serial.println("OFF"); // Skriver OFF til monitoren
```

   Kommandoene må legges inn i loop()-funksjonen idet lyset slås på og når det slås av.

   Kompiler og overfør programmet. Åpne monitoren ved å trykke på forstørrelsesglasset øverst i høyre hjørne av IDE-vinduet:

6. ***Tell ned på monitoren***

   Vi skal nå lage en variabel som teller ned fra 10 til 0.

   Alle variabler må defineres med *navn* og *type*. Type kan være heltall (int), desimaltall (float) eller bokstaver (char), i tillegg til en del andre. Navnet kan være hva som helst unntatt noen reserverte navn. Deklarering gjøres slik:

```
int i = 10;     // Deklarerer en heltallsvariabel med navnet "i" som settes til verdien 10
```

Global og Lokale variable. Deklarasjonen kan gjøres:

- ... før setup(). Den vil da være *global* og bevare verdien i alle funksjoner.
- ... i loop()-funksjonen. Da vil den være lokal og bare være tilgjengelig i denne funksjonen der den blir definert.

Endre verdien i variablene.

Vi kan endre verdien i en variabel ved å skrive inn en ny verdi:

```
i = 0;                  // Setter verdien til 0
i = i + 4;              // Øker innholdet i variabelen "i" med 4
i++;                    // Øker variabelen "i" med 1
i--;                    // Reduserer variabelen "i" med 1
```

Skriv nedtellingen til monitoren, *ned en verdi for hvert sekund.*

Bruk print-kommandoen for å skrive variabelen "i" til monitoren som vist under:

```
Serial.println(i); // Skriv ut verdien for "i"
```

7. Lag programmet og last opp

8. Hva skal skje når nedtellingen er slutt?

Legg merke til at programmet fortsetter å telle ned forbi null. *Vi vil at programmet når det kommer til null skal slutte nedtellingen og tenne lysdioden og være tent helt til programmet startes på nytt.*

For å vite når vi skal stoppe må vi bruke en *betingelsessetning* som sjekker verdien til variabelen "*i*". Når denne er null stopper programmet. Vi velger å bruke en *if-kommando.*

if-kommandoen skrives if (<Betingelse>) {<Programkode som skal utføres>}

Dersom betingelsen i parentesen ( ) er oppfylt, utføres programkoden mellom klammeparentesene. Om den ikke er oppfylt, går programmet videre uten å utføre programkoden mellom klammeparentesene.

if-kommandoen

```
if (i < 0)                  // Sjekk om i < 0, hvis ja utfør kommandoene i klammer
{
  digitalWrite(13,HIGH);// Slå på lysdioden
  while(1){}                // Stå her til programmet startes på nytt
}
```

while-kommandoen

While-kommandoen fungerer slik at det som står i klammeparentesen `{}` utføres dersom betingelsen i parentesen ( ) er oppfylt. Dersom vi setter betingelsen lik "1" så vil den alltid være oppfylt og while-kommandoen utføres til "evig tid".

Det kan være greit å slå av LED-lyset ved oppstart av programmet. Dette kan gjøres i setup()-funksjonen siden det bare skal gjøres ved oppstart.

9. *Skriv programmet og last opp*

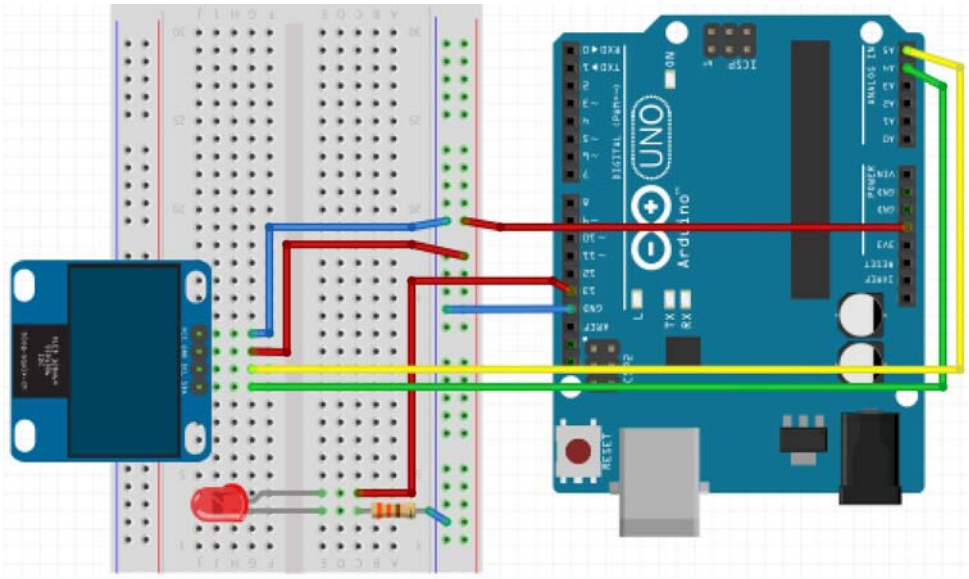
Skriv inn koden og test om den fungerer som ønsket.

10. Koble opp et eksternt display

Så langt har vi brukt monitoren på PC'en for å vise nedtellingen. Vi ønsker nå å vise nedtellingen på et lite display.

Koble Arduino-kortet fra PC'en før du kobler opp displayet.

Begynn med å montere displayet SSD1306 som vist på figuren. Displayet har fire pinner 128x64 punkter.



Kontroller en ekstra gang at oppkoblingen er riktig.

11. Installer biblioteker

For å kunne kommunisere med displayet (SSD1306), må vi installere to biblioteker. Biblioteker er en samling av kommandoer (funksjoner) som gjør at er det letter å bruke komponenter, i dette tilfellet et display. Bibliotekene er ofte komprimert i zip-filer og kan lastes ned fra:

https://www.ntnu.no/skolelab/bla-hefteserie

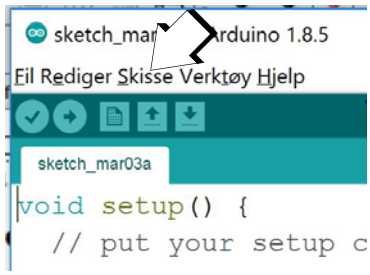
under fanen: *Grunnkurs programmering Arduino - (CanSat)*

Adafruit_SSD1306-master.zip
Adafruit-GFX-Library-master.zip

Lagre bibliotekene i nedlastingskatalogen eller et annet sted der du finner dem igjen. Programeditoren IDE vil automatisk pakke ut bibliotekene når de installeres.

Bibliotekene installeres på følgende måte:

Åpen Arduino-editoren



Velg: *Skisse*
Velg: *Inkluder bibliotek*
Velg: *Legg til .zip bibliotek* (øverst i nedtrekksmenyen)
Finn biblioteket i katalogen Nedlasting på PC'en (eller der du la det)
Velg: *Open*
... og biblioteket installeres

Gjenta for begge bibliotekene.

12. *Kommandoer for skriving til display*

- ***Inkluder bibliotekene i programmet***
  Øverst i programmet inkluderer man de to bibliotekene i tillegg til et par til. Dermed får programmet adgang til alle funksjonene som trengs for å skrive til displayet.

```
// Inkludering av biblioteker
#include <SPI.h>              // Inkluder bibliotek for kommunikasjon på serie linjer
#include <Wire.h>             // Inkluder bibliotek for kommunikasjon via I2C
#include <Adafruit_GFX.h>     // Inkluder bibliotek for å skrive til display
#include <Adafruit_SSD1306.h> // Inkluder bibliotek for å skrive til display
```

Disse legges inn helt i starten av programmet før deklarasjon av variabler og setup().

- ***Deklarasjon av displayet***

Dernest må vi deklarere, dvs. gi display et navn og en type, i prinsippet kan vi ha flere displayer med forskjellige navn og adresser.

```
Adafruit_SSD1306 display(4);   // Deklarasjon av enheten "display" av typen
                                  Adafruit_SSD1306
```

- ***Sett adressen til displayet***

Displayet har en adresse som programmet må vite om, denne er 3C (60) i heksadesimal kode og er normalt satt av fabrikken.

```
display.begin(SSD1306_SWITCHCAPVCC,0x3C);// Initialiser display med adresse 0x3C
```

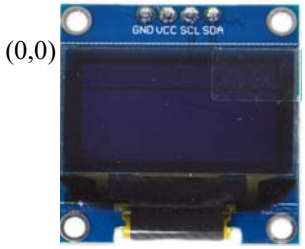
Adresselinjen skal plasseres i setup()-funksjonen.

- ***Posisjoner markør, tøm displayet og skriv ut tekst og variabler***

Så må vi tømme displayet før vi begynner å skrive, bestemme størrelsen på teksten og om vi skal skrive hvit tekst på svart bakgrunn eller omvendt. Disse kommandoene kan skrives inn i programmet der vi ønsker at utskriften skal skje, eller i setup()-funksjonen.

```
// Klargjør display for utskrift
display.clearDisplay();          // Slett informasjon på display
display.setTextSize(1);          // Sett størrelse på tekst (2 gir større tekst)
display.setTextColor(WHITE);     // Hvit tekst på mørk bakgrunn
```

Dernest må vi plassere markøren der vi ønsker at teksten skal begynne og skrive:



Koordinater for plassering av markør

```
// Skriver ut nedtellingen
display.setCursor(0,0);          // Plasser markør øverst til venstre (x, y)
display.print("Tekst: ");        // Skriv ut ordet "Tekst: "
display.println(i);              // Skriv ut variabelen i og skift linje (display.println)
```

Til slutt sendes det hele over til displayet og skrives ut:

```
display.display();               // Overfør informasjonen til displayet og vis
```

Dersom vi ønsker å skrive på en annen linje på displayet plasser markøren i ønsket posisjon før skriving begynner:

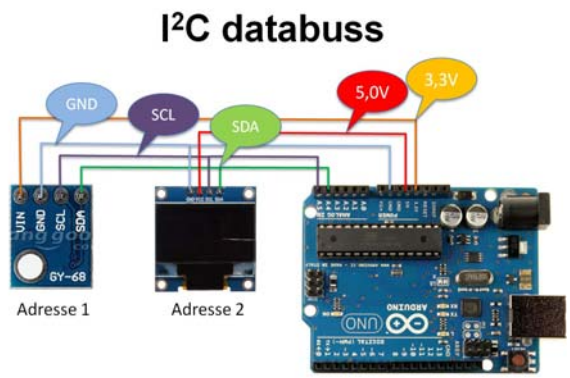
```
display.setCursor(0,10);         // Flytt markør til linje to
display.setCursor(0,20);         // Flytt markør til linje tre
```

Legg merke til at med skriftstørrelse 1 så er det greit å flytte markøren 10 bildepunkter ned for hvert linjeskift. Merk at koordinaten til markøren tar utgangspunkt øverste venstre hjørnet i karakterruta (origo).

13. Skriv programmet og test ut

Skriv nedtellingen til displayet og skriv "TENN" midt på displayet til slutt idet dioden tenner

14. IIC buss



Som omtalt i avsnitt 2.4.7 er dette en måte å kommunisere mellom integrerte kretser (Inter IC, I^2C) ved hjelp av to ledninger (SDA, SCL). Dersom hver enhet (IC) har forskjellig adresse, kan man adressere informasjonen slik at den kommer til riktig enhet.

På figuren til høyre ser vi at to enheter er koblet på de samme to kommunikasjonslinjene (SDA (data), SCL (klokke)). Dette er displayet vårt og en trykksensor (BMP180 også kalt GY-68). I tillegg er de forbundet til spenningskildene 5,0V og 3,3V og jord (GND). Det er også vanlig å koble en motstand mellom SDA og SCL og opp til 5V, men det går også ofte bra uten.

15. Lagre programmet

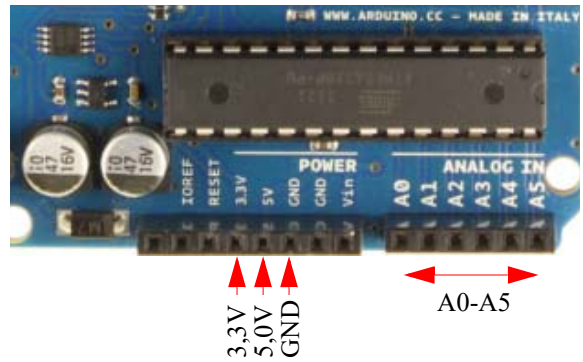
Husk å lager programmet før du går over til neste oppgave. Husk og bruk norsk tegnsett – f.eks. Oving 1.

4.2 Øving 2 – Voltmeter og termometer

I denne øvingen skal vi bruke displayet til å vise en spenning som vi måler. Deretter skal vi koble til en temperatursensor for å måle temperaturen i rommet. Vi skal så skrive både spenningen og temperaturen til displayet.

Vi kan la lysdioden være oppkoblet.

1. Analoge innganger – omregning fra digitale til desimale tall

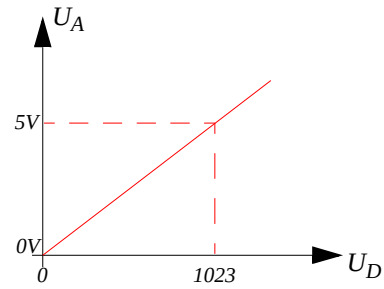


Arduino UNO har 6 analoge innganger. Ved hjelp av disse kan vi fra programmet lese av spenninger mellom jord (GND – 0V) og 5V, i trinn på noen mV. Prøver vi å lese av negative spenninger eller spenninger høyere enn 5V, kan Arduino'en ødelegges.

For at den avleste spenningen skal kunne brukes i programmet må den gjøres om til et tall. Siden Arduino'en er digital vil spenningen 0 – 5V konverteres til et heltall fra 0 – 1023, eller egentlig et binært tall fra 2^0–1 til 2^{10}–1 (10 bit).

Dersom vi ønsker å vise spenningen i Volt, må vi regne om fra et digitalt tall (U_D) til en analog spenning (U_A). Vi vet at $U_D = 0$ tilsvarer $U_A = 0{,}0$V, og at $U_D = 1023$ tilsvarer $U_A = 5{,}0$V som vist på figuren til høyre. Vi kan da bruke følgende formel for å regne om fra digitalt tall til spenning:

$$U_A = \frac{U_D}{1023} \cdot 5.0\text{V} \qquad (4.1)$$



2. Avlesning av spenninger

Velg "Ny" under "Fil" på meny-linja. Dere vil da få opp rammen for et nytt program:

```
void setup() {
  // put your setup code here, to run once:
}
void loop() {
  // put your main code here, to run repeatedly:
}
```

Dere kan godt beholde programmet fra Øving 1 i et nabovindu slik at dere kan kopiere kommandolinjer fra dette etter som dere trenger det.

- *Kopier følgende fra "Øving 1":*
  - *Bibliotekene (plasseres før setup()-funksjonen)*
  - *Deklarasjonen av displayet (plasseres før setup()-funksjonen)*
  - *Definisjonen av adressen til displayet (plasseres i setup()-funksjonen)*
  - *Klargjøringen av displayet (plasseres i loop()-funksjonen)*
  - *Skriv til displayet (plasseres i loop()-funksjonen)*
- *Deklarer to variable:*

```
int U_D;      // Variabel som holder digital avlest spenning
float U_A;    // Variabel som holder analogt avlest og omregnet spenning
```

Deklarer dem som globale variable, dvs. før setup()-funksjonen.

- *Avlesning av den analoge inngangen*

Følgende kommando leser av den analoge inngangen A0:

```
U_D = analogRead(A0);     // Avleser analog inngang A0
```

Vi leser av den digitale verdien og legger resultatet inn i variabelen *U_D*. Siden vi vil at denne skal leses av hele tiden legger vi den inn i loop()-funksjonen.

Bruk delay-kommandoen i loop()-funksjonen slik at programmet foretar måling og skriver ut fem ganger i sekundet.

- *Omregning fra digital verdi til analog verdi*

Vi legger inn omregningen fra den digitale verdien (0 – 1023) til en analog spenning (0 – 5V)

```
U_A = (5.0/1023)*U_D;      // Omregning fra digital til analog spenning
```

- *Skriv verdien til displayet*

Bruk display-kommandoene for å skrive ut den målte verdien til displayet. Det skal f.eks. stå: *Spenning: 3.3V* på en linje.

For å skrive ut et desimaltall med én desimal, bruker man følgende kommando:

```
display.print(U_A,1);      // Skriver ut spenningen med én desimal
```

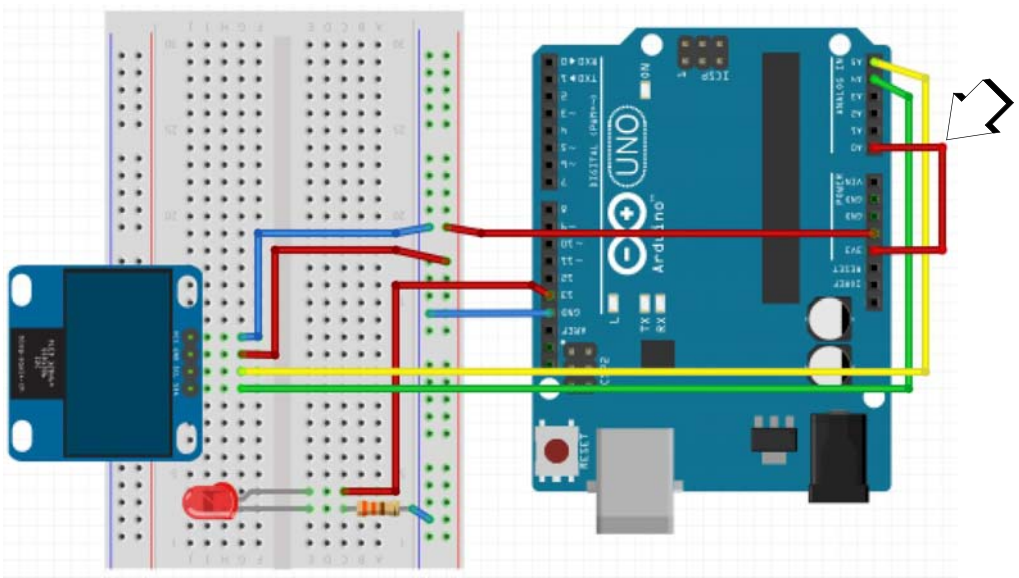
Prøv gjerne med to siffer også.

Teksten "Spenning: " og "V" må skrives på egne linjer. Husk at linjeskift utføres ved å legge "ln" til kommandoen:

```
display.println("V");      // Skriver ut en V
```

3. *Oppkobling*

Oppkoblingen er ganske enkel. Behold oppkoblingen fra Øving 1, men koble en jumper fra A0 og til 3,3 V som vist. Denne ledningen kan dere også flytte til 0,0V eller til 5,0V.



4. *Kompiler, last over og test programmet*

Kompiler, last over og test programmet.

- *Endre antall siffer til 2 og sjekke nøyaktigheten på spenningen 3,3V på arduino-kortet*

Sjekk også 5V og 0V

• *Sjekk spenningen på et 1,5 V batteri (ikke sjekk et 9 V batteri)*

Bruk en jumper fra GND og en fra A0 og sjekk spenningen på et 1,5 V batteri.
NB! **IKKE KOBLE BATTERIET FEIL VEI**, det er ikke sikkert at Arduino'en tåler det. Inngangene tåler heller ikke spenninger over 5V.

• *Måling av 9V batterier?*

Har dere et forslag til hvordan dere kan lage en batterisjekker som også kan måle 9V-batterier?

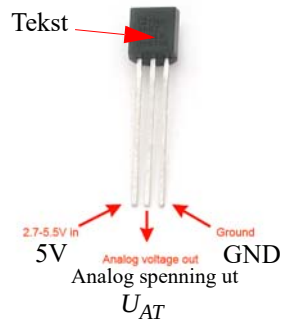
Tips: Bruk spenningsdeler.

Tegn koblingsskjema!

Hva må man passe på i programkoden for at displayet skal vise batteriets virkelige spenning?

5. *Omregning fra spenning til temperatur*

Sparkfun Inventor's kit inneholder en transistorlignende komponent som er en temperatursensor, TMP36GZ (bruk ev. et forstørrelsesglass for å lese teksten på komponenten). Denne skal kobles til 5V og 0V (GND). På senterpinnen kan en da ta ut en spenning som er proporsjonal med temperaturen i rommet i henhold til følgende sammenhenger:



- Spenningen er 0,5V ved 0°C
- Spenningen endrer seg 10mV pr. °C
- Spenningen øker med økende temperatur

Sammenhengen mellom temperatur og målt spenning er. Dette er en sammenheng vi finner i databladet for sensoren:

$$Temp = (U_{AT} - 0.5\ \text{V}) / 0{,}01\text{V} \tag{4.2}$$

Vi må også huske på at vi leser av den digitale verdien av spenningen slik at vi må regne om fra digital til analog spenning

$$U_{AT} = (5.0/1023)U_{DT} \tag{4.3}$$

U_{DT} er det tallet programmet får fra den analoge inngangen og som settes inn i ligning (4.3), som videre settes inn i ligning (4.2), som beregner temperaturen i °C.

Det kan være lurt å opprette følgende variabler:

```
long U_DT;
float U_AT;
float Temp;
```

float er desimaltall og *long* kan håndtere store heltall med fortegn.

Vi har valgt å definere U_DT som et stort heltall (long). *int* vil bare holde heltall med fortegn opp til $\pm 2^{15}$ hvilket vil si ± 32768. Som vi skal se så kan dette bli for lite dersom vi ønsker å gjøre en midling over flere målinger.

NB! Husk at desimaltall skrives med punkt (.) ikke komma (,) i ligningene i programmet.

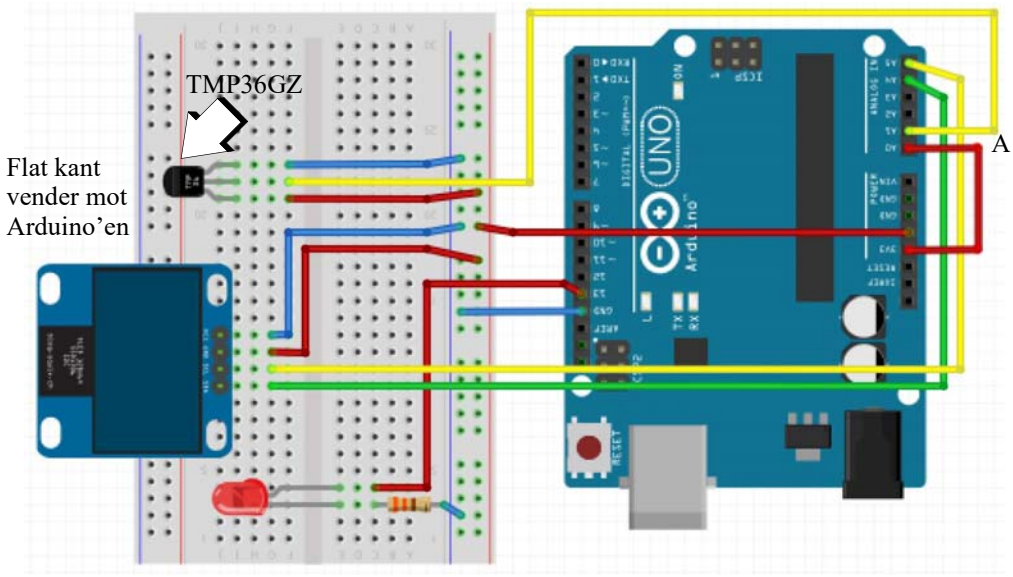
6. *Skriv programmet*

Skriv programmet under følgende forutsetninger:

- Bruk *A1* som analog inngang
- Skriv ut følgende på linje 2 på displayet: F.eks. *Temp: 23,3 C*
- Les av temperaturen 5 ganger i sekundet.

7. *Koble opp termometerkretsen*

Tegningen under viser hvordan kretsen kan kobles opp.



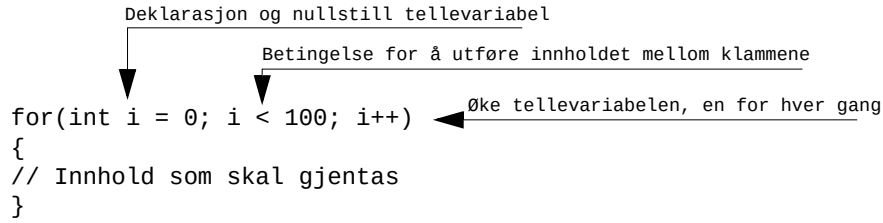
Pass på at temperatursensoren står riktig vei, med den flate siden mot Arduino'en.

Du vil antagelig se at temperaturen er svært ustabil. Dette skyldes sannsynligvis de lange ledningene som samler inn en del elektrisk støy fra det elektriske anlegget (spesielt lysstoffrør). Denne gjør at enkeltmålinger blir ustabile. Dersom vi midler over et større antall målinger, så vil tilfeldig støy kanselleres.

Dersom du har tid kan du gå til neste punkt hvor vi skal se nærmere på for-loopen for å kunne midle over mange målinger f.eks. 100.

8. Midling av måledata – bruk av for-loop.

Siden måling og addisjon av verdier går særdeles fort med en mikrokontroller vil vi knapt merke det dersom vi midler over f.eks. 100 målinger før vi går videre i programmet. Til det bruker vi en *for-loop*. Dvs. en løkke som gjentas et visst antall ganger. I vårt tilfelle må denne løkken inneholde avlesning og akkumulering av målte spenninger på inngang A1.



Mellom klammeparentesene gjøres en måling og verdien akkumuleres:

```
U_DT = U_DT + analogRead(A1);      // Avleser analog inngang A1
```

Dernest gjentas dette forløpet 100 ganger etter hverandre. Tellevariabelen økes med 1 (i++) for hver runde og stopper ikke før den er kommet til 99 (i < 100) som er den siste gangen målingen utføres.

Etter at alle 100 verdiene er akkumulert (*i* = 0 til 99), deler man på antallet målinger for å bestemme et gjennomsnitt:

```
U_DT = U_DT / 100;  // Midle ved å dele på antallet akkumulerte verdier
```

Man må dessuten huske på å sette U_DT lik null før man kjører en ny runde med målinger og akkumulering.

```
U_DT = 0;                  // Sett U_DT lik null slik at en ikke tar med seg verdier fra tidligere
```

Skriv programmet, kompiler, overfør til Arduino'en og test ut.

Ble temperaturmålingen mer stabil?

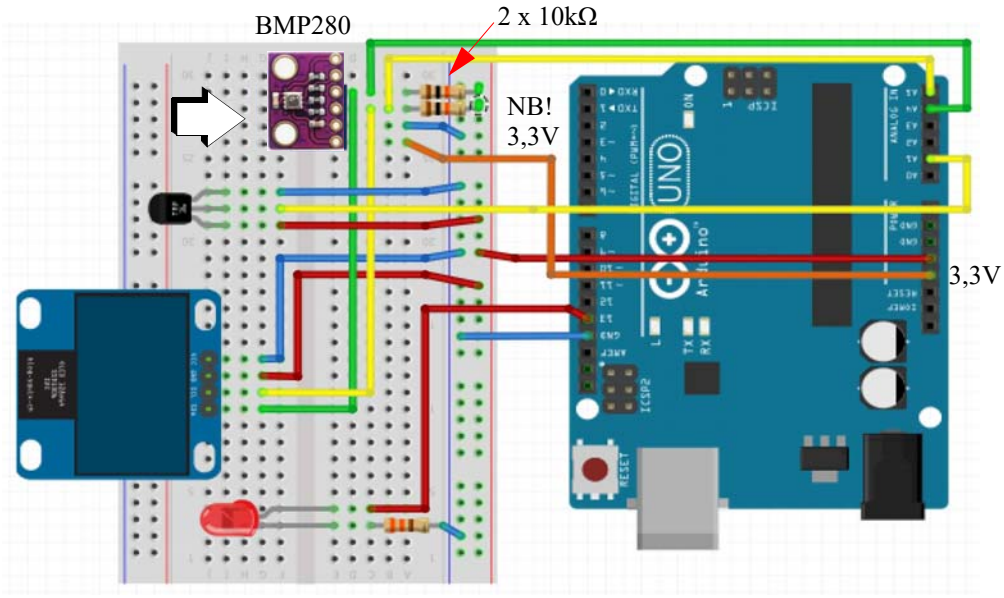
4.3 Øving 3 – Måling av lufttrykk, BMP180/BMP280

I denne arbeidsøkta skal vi videreføre byggeprosjektet nok en gang ved å legge på en trykkmåler. Vi har imidlertid to varianter av denne trykkmåleren BMP180 og BMP280 som er en nyere variant med omtrent like parametere og begge kan brukes i denne laboratorieoppgaven. Disse inneholder også en temperaturmåler. Dermed har vi to temperaturmålere som ev. kan sammenlignes.

NB! Dersom du skal bruke BMP280 så kan du fortsette her, skal du bruke BMP180 går du til vedlegg B.1, side 86.

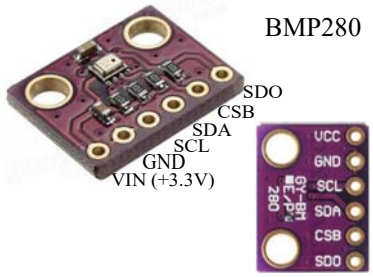
1. Montering av trykkmåler

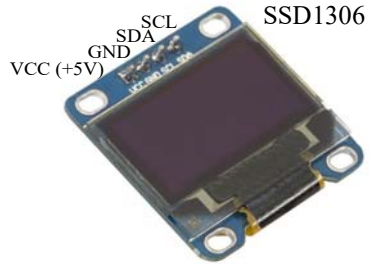
I^{2}C linjene (SDA, SCL) fra displayet skal nå kobles via SDA og SCL hos trykkmåleren. Siden de har hver sin adresse så skal dette gå bra. Det kan også være lurt å koble hver av linjene **til + 5V med to 10 kΩ motstander**. Vi kan fjerne ledningen fra A0. MERK at **BMP280 må kobles til 3,3 V** og IKKE til 5,0 V. Dette er viktig. GND kobles til felles jordpunkt (–).



Merk at navnet på beina til BMP280 står på undersiden av kretsen.

Figuren under viser nærbilder av BMP280 og displayet SSD1306 med tilhørende tilkoblingspunkter (bein).





Vi legger merke til at BMP280 har to ekstra tilkoblingspunkter som vi ikke bruker slik at den også kan kobles opp for SPI-kommunikasjon. Siden vi velger å bruke I^{2}C-bussen har vi valgt ikke å bruke disse to pinnene:

- CSB – Chip select (for SPI kommunikasjon)
- SDO – Seriell Data Out (for SPI kommunikasjon)

Vi skal nå installere biblioteket for BMP280.

2. Installasjon av biblioteket til BMP280

For å kunne kommunisere med trykkmåleren (BMP280) må vi igjen installere et bibliotek. Dette kan vi hente fra nettsiden:

https://www.arduino.cc/reference/en/libraries/bmx280mi/

Nederst på siden finner vi ulike utgaver av biblioteket. Last ned den nyeste versjonen:

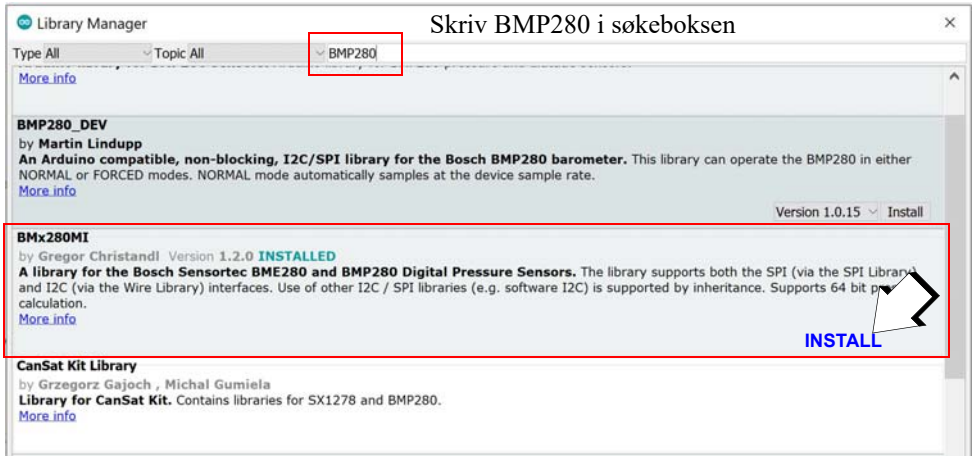
1.2.1 (latest)

Lagre biblioteket i nedlastingskatalogen som sist eller et annet sted der du finner det igjen.

Biblioteket installeres ved å åpen Arduino-editoren og ...

Velg: *Skisse*
Velg: *Inkluder bibliotek*
Velg: *Legg til .zip bibliotek* (øverst i nedtrekksmenyen)
Finn biblioteket i katalogen *Nedlasting* eller der du la det og ...
Velg: *Open*
... og biblioteket installeres.

Alternativt kan du gå til: *Tools –> Library Manager*. Skriv så BMP280 i søkevinduet og du får opp følgende vindu:



Velg BMx280MI og trykk INSTALL. Dermed blir biblioteket installert.

3. *Legg trykkmåling inn i programmet*

Vi har laget et lite testprogram for lesing av trykk og temperatur. Programmet kan lastes ned fra www.ntnu.no/skolelab/bla-hefteserie *Grunnkurs og videregående kurs Arduino/ESP32 – For yrkesfag elektro – Modul 2*. Programmet har fått navnet: *BMx280_I2C_Forenklet-1.ino*

Vi presenterer testprogrammet før vi gjennomgår og forklarer hver kommando:

```
// BMx280_I2C_Forenklet-1.ino
// Nils Kr. Rossing 22.08.20
// Programmet bygger på Gregor Christandl testprogram
// for avlesning av BMP280 via I2C-buss

#include <Wire.h>
#include <BMx280I2C.h>

#define I2C_ADDRESS 0x76

//Lag en instans bmx280 av typen BMx280I2C med I2C Address lik 0x76
BMx280I2C bmx280(I2C_ADDRESS);

void setup() {

 Serial.begin(9600); // Initialisering av seriekommunikasjon til monitor
 Wire.begin();       // Initialisering av I2C-bussen
 bmx280.begin();     // Initialisering av BMT280

 // Oversamplingsraten må settes før man kan gjennomføre målinger:
 bmx280.writeOversamplingPressure(BMx280MI::OSRS_P_x16);
 bmx280.writeOversamplingTemperature(BMx280MI::OSRS_T_x16);
 }

 void loop() {
 // Deklarerer to variabler for henholdsvis lufttrykk, P, og temperatur, T.
 float P, T;

 P = getPressure();   // Returnerer i mBar
 T = getTemperature();// Returnerer i grader C

 Serial.print("Lufttrykk: ");  // Skriv lufttrykk i mBar med to desimaler
 Serial.println(P,2);
 Serial.print("Temperatur: "); // Skriv temperatur i grader C med en desimal
 Serial.println(T,1);
```

```
delay(1000); // En måling pr. sekund
}

float getPressure() {

  if (bmx280.measure() == 0)
  {
   Serial.println("Kan ikke starte maaling før forrige maaling er avsluttet");
   return;
  }
  // Vent til måling er ferdig
  do
  {
    delay(100);
  } while (bmx280.hasValue() == 0);
  // Data kan ikke leses fra sensoren før funksjonen hasValue() varsler at data
  // er klare. Da kan data leses med funksjonene bmx280.getPressure()

  return bmx280.getPressure()/100; // Returner avlest lufttrykk i mBar
}

float getTemperature() {

  if (bmx280.measure() == 0)
  {
    Serial.println("Kan ikke starte temp. maaling før forrige maaling er
avsluttet");
    return;
  }
  // Vent til måling er ferdig
  do
  {
    delay(100);
  } while (bmx280.hasValue() == 0);
  // Data kan ikke leses fra sensoren før funksjonen hasValue() varsler at data
  // er klare. Da kan data leses med funksjonene bmx280.getTemperature()

  return bmx280.getTemperature();
}
```

Under har vi gjennomgått og forklart hver kommando i testprogrammet:

• *Inkluder biblioteket*

Biblioteket inkluderes med kommandoen:

```
#include <BMx280I2C.h>     // Inkluder biblioteket for måling av lufttrykk
```

Kommandoen legges øverst i programmet etter de andre bibliotekene.

• *Deklarasjon av trykksensoren*

Deklarasjonen av denne sensoren krever at vi legger inn I^2C adressen som er 0x76. Vi har valgt å definere denne adressen som en konstant:

```
#define I2C_ADDRESS 0x76
```

Så må trykksensoren som er av typen BMx280I2C, gis et navn i programmet.

```
BMx280I2C bmx280(I2C_ADDRESS);
```

Trykksensoren gis navnet "bmx280" av typen BMx280I2C. Denne deklarasjonen kan legges tidlig i programmet sammen med andre deklarasjoner.

• *Initialisering av trykk- og temperatursensoren*

Sensoren må så initialiseres ved at det sendes noe innledende informasjon til sensoren. Denne skal legges i setup()-funksjonen mellom klammeparentesene:

```
bmx280.begin();                    // Initier trykksensoren
```

BMP280 krever at vi setter en parameter i komponenten som handler om hvor ofte verdiene for trykk og temperatur punktprøves. Den interne punktprøvingsraten for trykk og temperatur settes med denne kommandoen:

```
// Oversamplingsraten må settes før man kan gjennomføre målinger:
bmx280.writeOversamplingPressure(BMx280MI::OSRS_P_x16);
bmx280.writeOversamplingTemperature(BMx280MI::OSRS_T_x16);
```

Disse kommandoene plasseres også i setup()-funksjonen

• *Deklarer variabler*

På dette tidspunktet nøyer vi oss med å deklarere en variabel for trykk (P) og temperatur (T). Vi velger å bruke *float* som gjør at vi kan ta vare på desimaltall. Vi velger også å deklarere dem som lokale variabler i loop()-funksjonen:

```
float P;                 // Deklarer variabelen for trykk
float T;                 // Deklarerer av variabel for temperatur
```

Denne deklarasjonen må gjerne legges i starten av loop()-funksjonen. Merk: At ved å legge deklarasjonen i loop()-funksjonen, så vil også variablene *P* og *T* deklareres og kanskje nullstilles for hver runde i loop()-funksjonen.

• *Les av trykk og temperatur*

Vi har valgt å definere en funksjon for avlesing av trykk (getPressure()) og for avlesing av temperatur (getTemperature()). Hver av disse funksjonene sjekker at måledata er tilgjengelig som beskrevet under:

For at vi trygt skal kunne lese trykket fra BMP280, så må det sjekkes at den er klar for å levere trykkmålinger. Dette gjøres med funksjonen:
`bmx280.measure()`
Denne funksjonen returnerer "1" dersom alt er klart for en ny måling og "0" dersom den fortsatt holder på med forrige måling. Dersom den ikke er klar til å gjøre målingen, så ønsker vi et varsel i monitoren. Alt dette gjøres med følgende if()-setning:

```
if (bmx280.measure() == 0)
 {
  Serial.println("Kan ikke starte maaling før forrige maaling er avsluttet");
  return;
 }
```

Dersom målingen ikke er klar så går den inn i en venteløkke `do { } while (<betingelse>)`. Programmet blir i løkken så lenge betingelsen er oppfylt. Funksjonen:
`bmx280.hasValue()`
returnerer "1" dersom den har fått en måleverdi og "0" dersom den ennå ikke er klar. Så lenge den returnerer "0" vil den derfor forbli i venteløkken til funksjonen returnerer "1" dvs. at måledata er klare. For hver runde venter den 100 ms. Alt dette gjøres i følgende venteløkke:

```
do
{
delay(100);
} while (bmx280.hasValue() == 0);
```

Når målingen er foretatt kan verdiene for henholdsvis lufttrykk (P) og temperatur (T) returneres med følgende kommandoer:

```
return bmx280.getPressure()/100; // Returnerer verdi for trykk i mBar
return bmx280.getTemperature(); // Returnerer verdi for temperatur i grad. C
```

Legg merke til at verdien for lufttrykk deles på 100 før den returneres. Dette skyldes at sensoren leverer verdien i Pascal (100 Pa = 1 mBar). De returnerte verdiene legges inn i de respektive variablene *P* og *T* i loop()-funksjonen:

```
P = getPressure();   // Returnerer i mBar
T = getTemperature();// Returnerer i grader C
```

og skrives til monitoren.

Kommandoen helt i slutten av loop()-funksjonen:
`delay(1000);`
avgjør hvor ofte det tas nye målinger. I dette tilfellet tar man en måling hvert sekund.

• *Bruk av funksjoner – en kort forklaring (Kjent stoff?– Hopp over)*

Før vi går videre la oss kort se hvordan en funksjon oppfører seg og hvordan den kan brukes.

En funksjon er en liten programbit som gjør en spesifikk jobb og som kan kalles opp når man ønsker det mens man er i *hovedprogrammet* (loop()-funksjonen).

I eksempelet til høyre ser vi hovedprogrammet og funksjonene `getPressure()` og `getTemperature()`, som er navnet vi har gitt funksjonene og som er av typen `float`, dvs. de returnerer desimalverdier.

Det funksjonen skal gjøre er avgrenset av {} og kalles funksjonens *kropp*. Vår funksjoner skal kommunisere med BMP280 og lese av trykket *P* eller temperaturen *T*.

Når hovedprogrammet kommer til kommandoen `P = getPressure();` vil den hoppe til funksjonen getPressure, lese av trykket målt av BMP280 og returnere verdien (`return P;`) til hovedprogrammet der verdien legges inn i variabelen P (`P = getPressure();`).

```
...
void loop()
{
...
// Innhold av loop()-funksjonen
...
P = getPressure() //Funksjonen kalles
T = getTemperature() //Funksjonen kalles
...
}

// Definer getPressure()-funksjonen
float getPressure()
{
float P;
...
// Innhold av getPressure()-funksjonen
...
return P; // Returnes trykkverdi
...
}

// Definer getTemperature()-funksjonen
float getTemperture()
{
float T; // Returnes temperatur
...
// Innhold av getTemperature()-funksjonen
...
return T;
...
}
```

På tilsvarende måte leses temperaturen og returneres på forespørsel.

Du skal nå inkludere det du har lært i programmet ditt. Du plukker da deler av testprogrammet og legger inn i programmet ditt.

• *Legg inn kommandoer i programmet ditt for å kunne lese av lufttrykket og temperatur*

Legg inn nødvendige kommandoer i starten av programmet, i setup()-funksjonen og i loop()-funksjonen, og de to funksjonene getPressure() og getTemperature() i slutten av programmet

• *Kall opp funksjonen getPressure() og les av lufttrykket*

Kaller opp funksjonen getPressure() fra loop()-funksjonen, som returnerer det avleste lufttrykket.

Dette gjøres med denne kommandoen:

```
P = getPressure();                    // Kall opp getPressure() og hent lufttrykket
```

... som legges inn i programmet etter temperaturmålingen.

Det målte lufttrykket ligger nå i variabelen *P* i millibar.

• *Skriv lufttrykket til displayet*

Vi velger nå å skrive ut trykket *P* på første linje på displayet istedet for spenningen. Videre ønsker vi å skrive ut trykket med 2 desimaler. Vi ønsker følgende utskrift på displayet:

```
Trykk: 985.23 mBar
```

Vi legger merke til at trykket skrives ut med to desimaler.

Lag programmet for å skrive ut trykket til displayet.

• *Kompiler, overfør og test programmet*

Kompiler og overfør programmet til Arduino'en. Du skal nå kunne lese av trykk og temperatur. Trykket fra BMP280 og temperaturen fra TMP36GZ.

Du kan nå løfte kretsen opp så langt kabelen rekker og se at lufttrykket endrer seg. Du vil også se at desimalene etter komma "flagrer" litt, dvs. at det er litt støy på målingene så en kan vurdere hvor mange siffer det er verdt å ta med.

4.4 Øving 4 – Omregning fra lufttrykk til høyde

I denne øvingen skal vi modulere endring i lufttrykk til endring av høyde.

1. Omregning fra trykk til høyde

Lufttrykket vil kunne endre seg med høyden over havet. Trykkmåleren er så følsom at vi kan se endringer selv om vi bare løfter den opp 1 meter.

• *Omregningsformelen fra trykk til høyde – en kort forklaring (Kjent stoff?– Hopp over)*

Sammenhengen mellom trykk, temperatur og høyde kan uttrykkes med formelen[3]:

$$h = \frac{T_1}{a}\left(\left(\frac{P}{p_1}\right)^{-\frac{aR}{g_0}} - 1\right) + h_1 \tag{4.4}$$

Hvor:

h	Beregnet høyde i meter
p_1	Trykk i mBar ved referansehøyden h_1
h_1	Referansehøyden i meter
T_1	Temperatur ved referansehøyden h_1 i K(elvin)
P	Målt trykk i mBar
a	Temperaturgradient, foreslått verdi -0,0065 K/m

3. Se lign. (5.2), side 74 for nærmere omtale.

g_0 Tyngdeakselerasjonen 9,81 m/s^2
R Den spesifikke gasskonstant 287,06 J/kg K

Dere skal nå legge inn formelen i programmet etter at trykk *P* er målt slik at dere kan bruke den i formelen.

• *Skriv inn variablene som trengs*

Før vi legger inn selve formelen kan det være greit å deklarere alle nødvendige variabler som vi trenger. Ikke alle er variabler, men vi velger her å definere dem som det.

```
// Deklarerer variable for høydemåling
float H;              // Beregnet høyde over havet
float H1 = 127;       // Referansehøyde i meter
float P1 = 992.80;    // Referansetrykk i mbar ved referansehøyde
float T1 = 17.2;      // Temperatur i grader C, husk at formelen krever grader Kelvin
float a  = 0.0065;    // Temperaturgradient i K/m
float R  = 287.06;    // Den spesifikke gasskonstanten i J/kg K
float g0 = 9.81;      // Tyngdeakselerasjonen i m/s2
```

Vi legger spesielt merke til *referansetrykk* og *referansehøyde*. Dette skal vi senere bruke til å kalibrere målingene slik at vi kan få absoluttverdier. Verdiene som ligger der er verdier fra da programmet ble laget. Selv om vi ikke har kalibrert målingene så kan vi fortsatt gjøre relative målinger.

Det er også lagt inn en temperatur i grader. Dette er *ute-temperaturen.* Det er derfor ikke alltid meningsfullt å bruke den temperaturen vi måler med instrument vårt dersom vi sitter innendørs.

Vi kommer tilbake til dette under kalibreringen.

• *Utfør omregningen og skriv til display*

Skriv inn formelen med de definerte variablene og verdiene.

```
// Beregning av høyden
H = ((T1 + 273.15) / a) * (pow((P / P1), (-(a * R) / g0)) - 1) + H1;
```

Studer uttrykket og sammenlign med formelen over.

Det er god pedagogikk å la elevene skrive det matematiske uttrykket på en form som passer i programmet. Her har vi valgt å oppgi syntaksen slik at vi kan konsentrere oss om høydemålinger og kalibrering.

Tips til programmeringen for å beregne høyden:

Aritmetiske funksjoner:

+	Addisjon
-	Subtraksjon
*	Multiplikasjon
/	Divisjon
pow(g,e);	Opphøye et grunntall (g) i en eksponent (e)
log(x);	Naturlig logaritme, definer argument og resultat som *double*

- Skriv programbiten som regner om skriv resultatet til displayet

Skriv ut høyden i "meter" til displayet på linje to og flytt temperaturmålingen til linje tre.

- *Test programmet og se at det fungerer som ønsket.*

Du kan nå løfte kretsen opp så langt kabelen rekker og se at høyden endrer seg. Du vil også se at desimalene etter komma "flagrer" litt. Naturlig nok vil variasjonen i trykkmålingen også gi seg utslag i støy på beregnet høyde.

2. Beregn gjennomsnittet for 100 målinger (kan hoppes over)

Om ønskelig kan man nå utføre en midling også av trykkmålingene slik som for temperaturmålingene.

Istedet for å lage en egen for-loop for trykkmålingene kan man legge dem inn i den allerede eksisterende for-loopen for midling av temperatur, dermed sparer man tid.

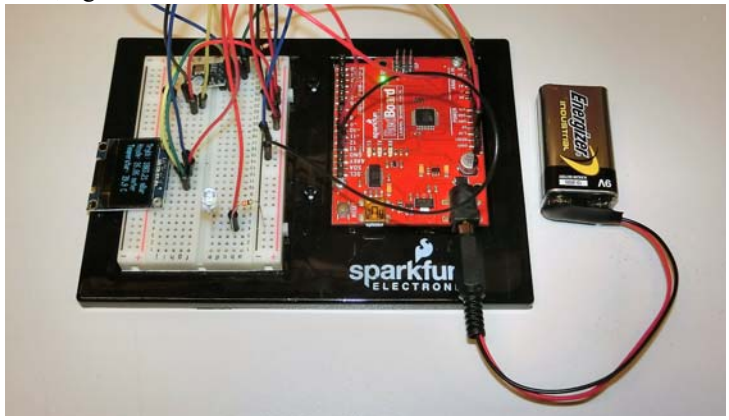
Dersom vi gjør denne øvelsen vil vi oppdage at tiden mellom hver måling forlenges betraktelig fra 0,2 sek til 3 – 4 sek ved midling av 100 målinger. Dette skyldes at å regne med desimaltall med dobbel lengde er en tidkrevende operasjon. Dessuten skal hver måleverdi hentes fra trykksensoren hvilket også tar tid.

3. Måleoppdrag 1 – Gjør relative målinger av høyde

Ta utgangpunkt i den bygningen dere befinner dere i (Realfagbygget) og mål relativ høyde for hver etasje, fra kjeller (U4) til toppetasje (5) så lavt og høyt dere kommer. Bestem den totale høyden av trapperommet. Sammenlign målingene med hva andre har fått. Det beste stedet å utføre slike målinger er i de sirkulære trappesjaktene som dere finner til høyre langs den lange korridoren langsetter bygget.

Her kan dere måle fra U4 – 5 (fra 4. underetasje til 5 etasje). Dere kan også prøve å ta heisen.

Sammenlign deres resultater målinger gjort med lasermåler. For å slippe å dra med hele PC'en kan vi drive kretsen med et 9V batteri. Vi bruker en batterikontakt og en plugg som passer til settet, og et 9V-batteri.



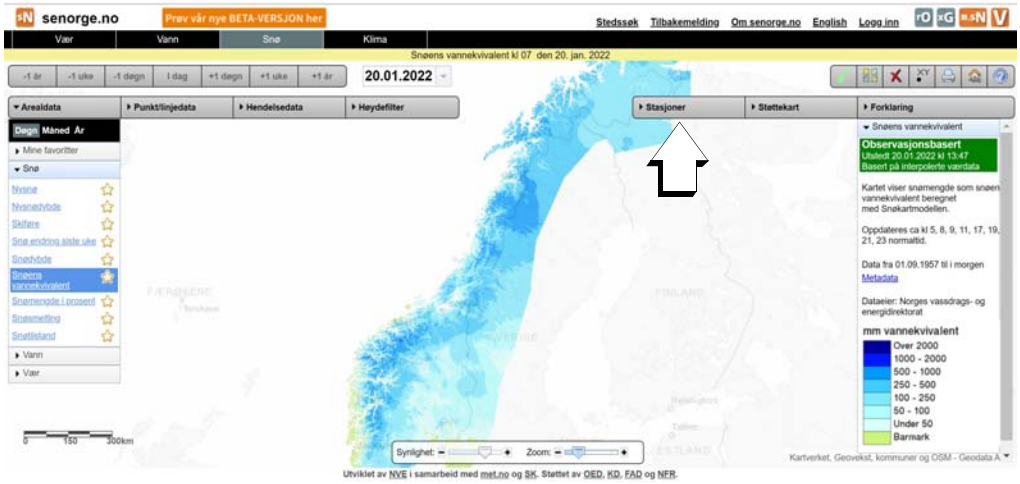
4. *Måleoppdrag 2 – Kalibrer høydemåleren slik at den måler absolutt høyde over havet*

Dersom vi ønsker å måle absolutt høyde over havet må vi kalibrere målingene. Dette kan vi gjøre ved å finne lufttrykket ved en nærliggende målestasjon (p_1) ved en kjent høyde (h_1). Her kan vi anta at luttrykksmålingene er lufttrykket på stedet ved den angitte høyden og ikke er omregnet til havnivå.

Høydemålingen kalibreres ved å sette verdiene for p_1 og h_1 inn i formelen vår:

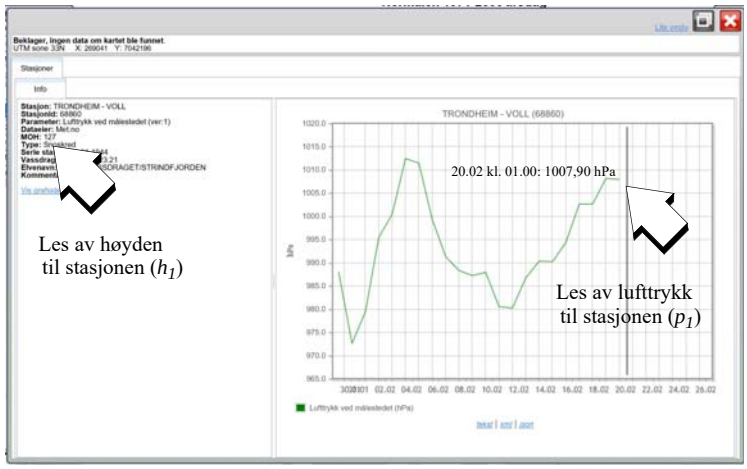
$$h = \frac{T_1}{a}\left(\left(\frac{P}{p_1}\right)^{-\frac{aR}{g_0}} - 1\right) + h_1 \tag{4.5}$$

For å finne en nærliggende målestasjon går vi inn i den offisielle portalen for målestasjoner i Norge og får opp følgende nettside:



Velg "Stasjoner" og kryss av for "Lufttrykk" og velg en målestasjon nær der du befinner deg og som registrerer lufttrykk. Den nærmeste stasjonen til Realfagbygget er **Voll forsøksgård på Moholt** ca. 2 km fra Realfagbygget.

Etter å ha valgt målestasjon får vi opp en graf som viser lufttrykket de siste 30 døgn. Målingene oppdateres en gang i timen så avvikene burde være til å leve med.



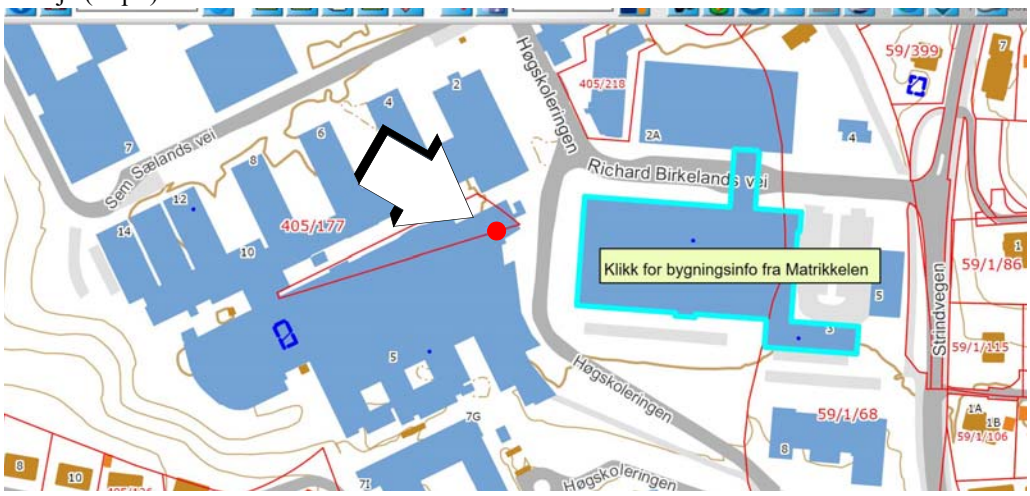
Før musa bort til punktet lengst til høyre på kurven og les av sist registrert verdi for lufttrykk. Les også av stasjonens høyde over havet (h_1). Disse verdiene settes så inn i programmet som p_1 og $h1$.

Les mer om kalibrering og hvordan du bruker programvaren i avsnitt 5.3, side 76.

5. ***Oppdrag 3 – Gjør absolutte målinger av høyden over havet i laboratoriet***

Gjør målinger av høyden over havet i laboratoriet ev. gå utenfor inngangen (svingdøra) på bakkenivå og gjør målinger. Sammenlign målinger utført av andre som befinner seg på samme sted og se på avvik. Hvordan kan vi ev. forklare avvik i absolutte målinger?

Kartet under er et utsnitt fra Gløshaugen. Skolelaboratoriet befinner seg nær prikken i 2. etasje (se pil).



Kartet er hentet fra:
https://kart5.nois.no/trondheim/Content/Main.asp?layout=trondheim&time=1550437879&vwr=asv

Gå inn i kartdatabasen og finn høyden over havet utenfor bygningen på bakkenivå, ev. estimer høyden på laboratoriet dere befinner dere i.

Hvilke avvik registrerer dere i forhold til høyden fra kartdata og den dere måler etter kalibrering?

6. *Oppdrag 4 – Mål høyden på et av høyhusene på Gløshaugen (om det er tid)*

Det høyeste punktet her på Gløshaugen er sannsynligvis toppen av en av høyblokkene. Ta med dere måleinstrumentet å finn ut følgende:

- *Mål absolutt høyde utvendig ved foten av høyblokka.*
- *Ta heisen opp i 13. etasje og finn den absolutte høyden der. Estimer høyden fra der dere klarer å komme til å måle og til toppen.*
- *Finn høyden av blokka fra foten til toppen.*
- *Sammenlign med andre som har gjort det samme.*

7. *Oppdrag 5 – Mål lufttrykket i klasserommet (etter eget ønske og tilgjengelig tid)*

I dette oppdraget er oppgaven å registrere lufttrykket i klasserommet når man åpner og lukker døra. For at dette skal være mulig må man gjøre følgende:

1. Sample trykket så raskt som mulig
2. Legge inn kommandoer som skriver data til monitoren
3. Sørge for å legge ut data slik at det lett lar seg kopiere inn i f.eks. Excel eller at annet analyseprogram. Skill gjerne verdiene med komma eller semikolon. Lag ny linje for hver måling.
4. Bruk Excel til å plotte trykkurven som funksjon av tiden
5. Undersøk om dere kan registrere trykkendringer når dere åpner og lukker døra raskt. Gå ev. inn i et naborom som er mindre og gjenta forsøket. Er forskjell på målingene gjort i et stor rom sammenlignet med et lite rom.

6. Oppdrag 6 – Foreslå andre oppgaver

Kom med forslag til andre oppgaver knyttet til måling av temperatur-, lufttrykk- og høydemålinger.

5 Måling av lufttrykk og beregning av høyden over havet

5.1 Måling av lufttrykk

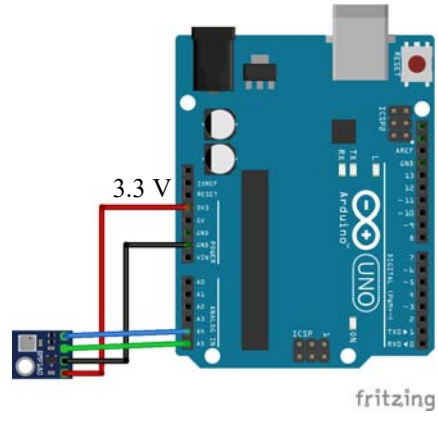
5.1.1 Måling av lufttrykk ved endring i resistans (piezo-resistivitet)

Den piezo-resistive effekten er forskjellig fra den piezo-elektriske effekten. Den piezo-resistive-effekten ble oppdaget av ***Lord Kelvin*** i 1856. Først i 1954 oppdaget C.G. Smith at germanium- og silisiumkrystaller hadde spesielt store variasjoner i ledningsevnen når de ble utsatt for mekanisk stress. Ledningsevnen til materialer er avhengig av mengden ladningsbærere i ledningsbåndet og hvor lett elektroner kan frigjøres fra valensbåndet. Dette er igjen avhengig av størrelsen på *båndgapet* mellom lednings- og valensbåndet i materialet. Når de nevnte materialene utsettes for stress, vil båndgapet endre seg og dermed også ledningsevnen.

5.1.2 Barometeret BMP180 (Bosch)

BMP180 er en trykksensor som går under betegnelsen *barometer*. Kretsen erstatter BMP085. Senere er også BMP280 kommet på markedet. Selve sensoren er basert på teknologi knyttet til piezo-resistivitet og levert av firmaet Robert Bosch. Kretsen leverer trykkdata via en digital I^2C buss. I tillegg til trykksensoren inneholder kretsen en temperatursensor. Her er noen nøkkeldata:

- Måleområde: 300 – 1100 hPa (millibar) – som typisk tilsvarer 9 000 til –500 m
- Oppløsning, trykk: 0,01 hPa (0,01 mbar)
- Oppløsning, høyde: 0,25 meter
- Temp. nøyaktighet (abs.) ± 1 °C (0 – 65 °C), ± 0,5 °C @ 25 °C
- Oppløsning i temp.: 0,1 °C
- Konverteringstid trykk: 7,5 ms (standard)
- Supplyspenning: 1,8 – 3,6 V
- Lavt effektforbruk: 5µA ved 1 måling pr. sek.
- Responstid: 7,5 ms (maks)
- Standby strøm: 5 µA
- Langtidsstabilitet: ± 1 hPa pr. 12 måneder.
- Absolutt nøyaktighet – 4,0 – + 2,0 hPa (mbar)



Kretsen leveres fra bl.a. Sparkfun og er montert på et kretskort for lettere å kunne kobles til f.eks. en Arduino (*"breakout board"*). Kretsen leveres også fra firmaet www.KultOgBillig.no og andre.

For å lese dataene via I^2C bussen brukes biblioteket: #include <wire.h> i tillegg til biblioteket som er laget til kretsen: `#include <SFE_BMP180.h>`

For mer informasjon se: https://www.sparkfun.com/tutorials/253. Her finner du også databladet og programvare for Arduino.

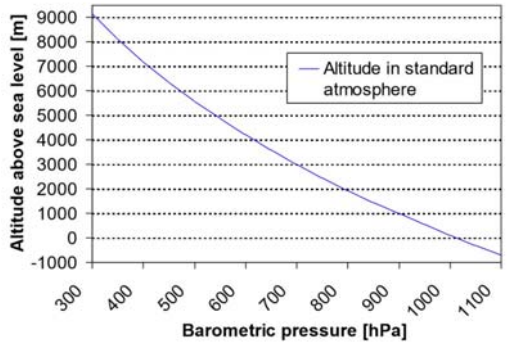
Biblioteket som følger med BMP180 beregner høyden på bakgrunn av trykkmålinger. Dvs. man ser bort fra temperaturgradienten som funksjon av høyden. Den beregnede høyden vil alltid være i forhold til høyden til stedet der referansetrykket er målt:

$$h = 44330 \cdot \left(1 - \left(\frac{p}{p_0} \right)^{\frac{1}{5{,}255}} \right) \tag{5.1}$$

Hvor:

h	Beregnet relativ høyde i forhold til referansehøyde ved p_0 (gjerne ved havnivå)
p	Målt trykk
p_0	Målt referansetrykk

Figuren under viser en typisk sammenheng mellom trykk og høyde.



For en mer generell behandling, se avsnitt 5.2, side 72.

Biblioteket til BMP180:

Biblioteket inneholder en rekke biblioteksfunksjoner som vil bli beskrevet her:

Inkludering av bibliotek og deklarasjon av trykksensor:

```
#include <SFE_BMP180.h>
SFE_BMP180 pressure;
```

Definer et *objekt* med et passende navn. Navnet velges av brukeren. Dersom man har flere like trykkmålere, velger man objekter med forskjellig navn for de to trykkmålerne. Her velger vi som eksempel: *pressure* som objektnavn.

```
pressure.begin ();
```

Initier kommunikasjon med objektet. Om kommunikasjonen med objektet lyktes, returneres 1 ellers 0 som gir mulighet for å generere en feilmelding.

status = pressure.startTemperature ();

Funksjonen initierer temperaturmåling. Dersom kommunikasjonen med objektet lyktes, returneres (til *status*) antall millisekunder enheten bør vente før temperaturen avleses. Om den mislykkes returneres verdien 0.

status = pressure.getTemperature (T);

Om status er forskjellig fra 0 så legges temperaturen i °C inn i variabelen T (argumentet til funksjonen).

status = pressure.startPressure(oversampling);

Argumentet *oversampling* angis med en verdi fra 0 til 3, og uttrykker graden av oversampling. En verdi på 3 vil gi større nøyaktighet, men tar lengre tid for å vente på svar. Variabelen *status* angir hvor lang tid en bør vente før en henter verdien for trykket. Returneres verdien 0 betyr det at det ikke er oppnådd kontakt med objektet.

status = pressure.getPressure(P,T);

Funksjonen *pressure.getPressure(P,T)* avleser både absolutt trykk (*P*) og temperatur (*T*). Trykket angis i millibar (mBar) og temperaturen i °C. Funksjonen returnerer status som angir om kontakt med sensoren er vellykket (returnerer delay før avlesning i ms) eller om kommunikasjonen mislykkes (returneres verdien 0).

p0 = pressure.sealevel(P,ALTITUDE);

Funksjonen *p0 = pressure.sealevel(P,ALTITUDE);* returnerer trykket ved havnivå (*p0*) på bakgrunn av kjent høyde over havet på målestedet (ALTITUDE). I forbindelse med værmelding er det vanlig å normalisere trykket til havnivå.

a = pressure.altitude(P,p0);

Biblioteket inneholder også en funksjon som beregner høyden over havet i meter (*a*) på bakgrunn av trykket målt på stedet i mbar (*P*) og trykket ved havnivået også i mBar.

Beskrivelsen av funksjonene er hentet fra eksempelet: SFE_BMP180_example av Mike Grusin, SparkFun Electronics 10/24/2013 V1.1.2 Updates for Arduino 1.6.4 5/2015 og ligger vedlagt biblioteket.

5.2 Måling av høyde basert på trykkmålinger

Det finnes flere ulike modeller for omregning fra trykk til høyde.

5.2.1 Vanlig brukt modell for omregning fra trykk til høyde i CanSat

Trykk måles normalt i Pascal hvor 1 Pa = 1 N/m^2.

Tidligere ble trykk målt i atmosfærer (atm), mmHg eller Bar.

En normalverdi for lufttrykket er:

1 atm = 760 mmHg = 1.01325 Bar = 1013.25 mBar = 101325 Pa = 1013.25 hPa

Vi legger merke til at h(ekto)Pa er det samme som m(illi)Bar og at 1 Bar er lik 100 000 Pa.

Lufttrykket er bestemt av tyngden til det "havet" av luft som vi befinner oss på bunnen av. Lufttrykket er derfor avhengig av mengden luft som til en hver tid befinner seg over hodet på oss. Vekta av luftmengden er avhengig av tyngdekraften, tykkelsen og tettheten til luftlaget, som igjen er avhengig av hvordan lufta forflytter seg og av temperaturen, dvs. værforholdene. Som vi ser er det mange faktorer å ta hensyn til. Likevel finnes det matematiske modeller som gjør at en kan gjøre rimelig nøyaktige høydemålinger på bakgrunn av trykkmålinger. Imidlertid er kalibrering særdeles viktig i denne sammenhengen

En regner normalt at trykket faller med 1 millibar pr. 8 meter, eller ca 12.5 millibar pr. 100 meter. Dette stemmer ikke så verst for de første 2000 meter, deretter minker trykket mindre for hver 1000 meter.

Målinger som refereres til på http://www.senorge.no/index.html angir lufttrykk på stedet, dvs. ikke korrigert mht. høyde over havet.

Tabellen under viser typiske verdier for sammenhengen mellom trykk, lufttetthet, temperatur og høyde over havet.

HoH	**Temperatur**	**Lufttrykk**	**Tetthet**	
(m)	*(C)*	*(hPa)*	*(kg/m^3)*	
0000	15.0	1013	1.2	
1000	8.5	900	1.1	
2000	2.0	800	1.0	(Galdhøpiggen)
3000	-4.5	700	0.91	
4000	-11.0	620	0.82	
5000	-17.5	540	0.74	
6000	-24.0	470	0.66	
7000	-30.5	410	0.59	
8000	-37.0	360	0.53	
9000	-43.5	310	0.47	(Mount Everest)
10000	-50.0	260	0.41	(Marsjhøyde rutefly)
11000	-56.5	230	0.36	
12000	-56.5	190	0.31	
13000	-56.5	170	0.27	
14000	-56.5	140	0.23	
15000	-56.5	120	0.19	
16000	-56.5	100	0.17	
17000	-56.5	90	0.14	
18000	-56.5	75	0.12	
19000	-56.5	65	0.10	
20000	-56.5	55	0.088	
21000	-55.5	47	0.075	
22000	-54.5	40	0.064	
23000	-53.5	34	0.054	
24000	-52.5	29	0.046	
25000	-51.5	25	0.039	
26000	-50.5	22	0.034	
27000	-49.5	18	0.029	
28000	-48.5	16	0.025	
29000	-47.5	14	0.021	

30000	-46.5	12	0.018
31000	-45.5	10	0.015
32000	-44.5	8.7	0.013
33000	-41.7	7.5	0.011
34000	-38.9	6.5	0.0096
35000	-36.1	5.6	0.0082

Omregningen fra trykk til høyde bør også ta hensyn til temperaturen, og temperaturen vil normalt forandre seg med høyden.

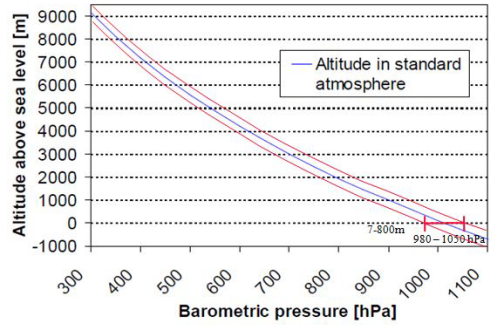
I programmer er det normalt lettere å forholde seg til en omregningsformel enn en tabell. Ulempen med en formel er at de mange parametrene kan gi stor usikkerhet i beregningene. Følgende formel er vanlig å bruke:

$$h = \frac{T_1}{a}\left(\left(\frac{p}{p_1}\right)^{-\frac{aR}{g_0}} - 1\right) + h_1 \qquad (5.2)$$

Hvor:

h	Beregnet høyde i meter
h_1	Starthøyde i meter
T	Temperatur i Kelvin
T_1	Starttemperatur i høyden h_1
a	Temperaturgradient, foreslått verdi -0,0065 K/m
p	Målt trykk i Pa
p_1	Trykk i Pa ved starthøyden
g_0	Tyngdeakselerasjonen 9,81 m/s^2
R	Den spesifikke gasskonstant 287,06 J/kg K

Denne formelen kan enten legges inn i datainnsamlingsenheten i f.eks. CanSat, men bedre i programvaren som behandler data. Har man rådataene fra sonden, har en større mulighet til etterbehandling enn om man bare har de omregnede dataene.



Diagrammet til høyre viser sammenhengen mellom trykk og høyde med økende høyde over havnivået. Normale variasjoner i luft-trykket ved havnivået kan være fra 980 hPa til 1050 hPa (millibar). Denne naturlige variasjonen kan derfor gi en absolutt endring i høydeberegningen på 7 – 800 meter dersom man ikke kalibrerer målingene.

Vi skjønner derfor at det er svært viktig med hyppig kalibrering dersom man skal kunne stole på høydemålerens absolutte høydeverdier.

5.2.2 Alternative beregningsmodeller

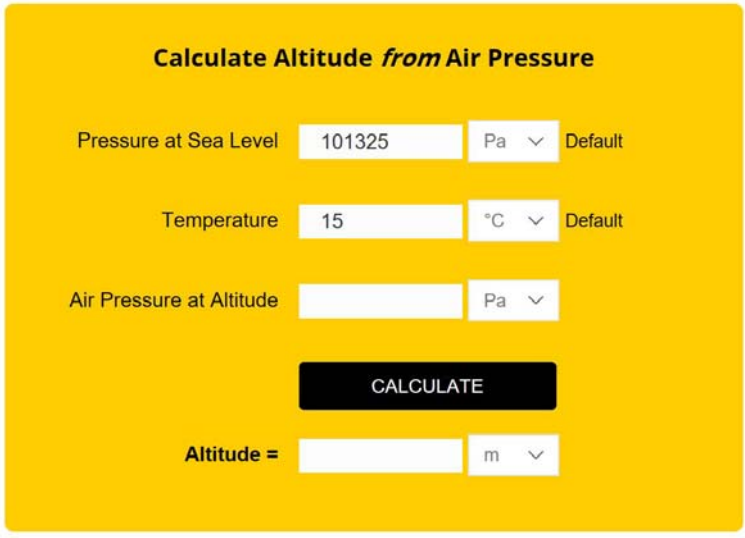
Det finnes imidlertid flere ulike modeller for sammenhengen mellom trykk og høyde. Her er en alternativ[4] modell hentet fra en teknisk note publisert av firmaet Vaisala som utvikler instrumenter for værobservasjoner og værstasjoner.

$$h = \left(\frac{R}{g}\right) T_m \ln\left(\frac{p_0}{p}\right) \tag{5.3}$$

Hvor:

h	Beregnet høyde over havet
R	Den spesifikke gasskonstant 287,06 J/kg K
g	Tyngdeakselerasjonen 9,81 m/s^2
T_m	Gjennomsnittlig lufttemperatur i det aktuelle måleområdet
p_0	Lufttrykk ved havnivået
p	Målt lufttrykk på det aktuelle stedet

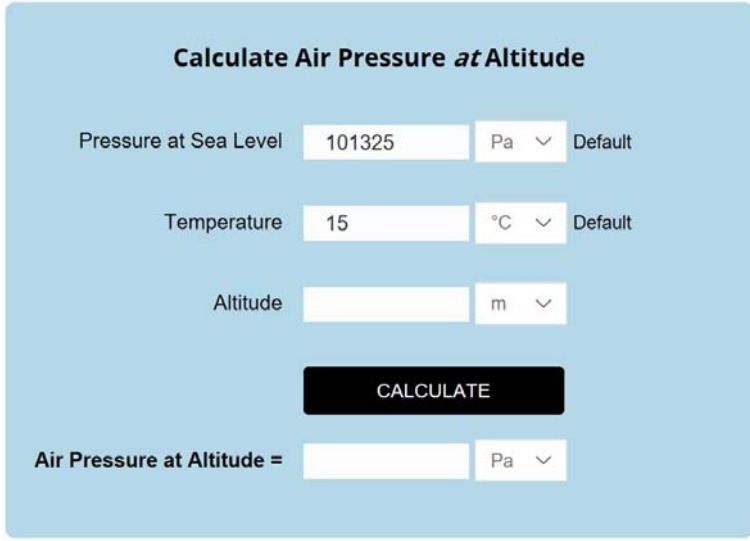
Det finnes også omregningskalkulatorer på nett. Firmaet MIDE Engeneering Solutions har en på nettsiden: https://www.mide.com/pages/air-pressure-at-altitude-calculator.



Her skriver man inn trykk og temperatur ved havnivå (ferskvare) i tillegg til målt trykk på den lokasjonen der man befinner seg, og man får et estimat av høyden på det stedet man er. Det er viktig at man bruker en referansehøyde ved havnivå som er i nærheten av stedet der man befinner seg.

4. http://www.vaisala.fi/Vaisala%20Documents/Measurement%20Theory/height_calculation.pdf

Tilsvarende kan man finne et estimat av trykket der man befinner seg dersom man kjenner trykket og temperaturen ved havnivået og hvor høyt man befinner seg.



5.3 Kalibrering av høydemåleren

En høydemåler basert på måling av lufttrykk kan kalibreres på flere ulike måter:

- Ved at man kjenner lufttrykket og høyden over havet til stedet der man starter målingen. I så fall kan man benytte lign. (5.2). Denne metoden er aktuell når man skal sende opp en sonde eller gå en tur i terrenget, og som gir absolutt høyde over havet.
- Ved at man nullstiller høydemåleren på bakkenivå, eller fra det stedet som man ønsker å måle høyden relativt til. Denne måten vil gi en relativ høydemåling i forhold til starthøyden for måleserien.
- Ved at man finner en offisiell målestasjon i nærheten, for eksempel en flyplass eller en havn som viser en oppdatert verdi av lufttrykket. Slike målestasjoner oppgir også deres høyde over havet, ev. at de referer målingene sine til havnivået. Det er imidlertid viktig å finne ut hvilken praksis de følger. Ulempen med flere slike målestasjoner er at de ikke viser kontinuerlige målinger, men oppdaterer målingene for eksempel hver time eller hvert døgn. Dessuten kan de være et stykke unna stedet der man befinner seg. En slik kalibrering burde imidlertid gi en absolutt høyde over havet med rimelig god nøyaktighet.

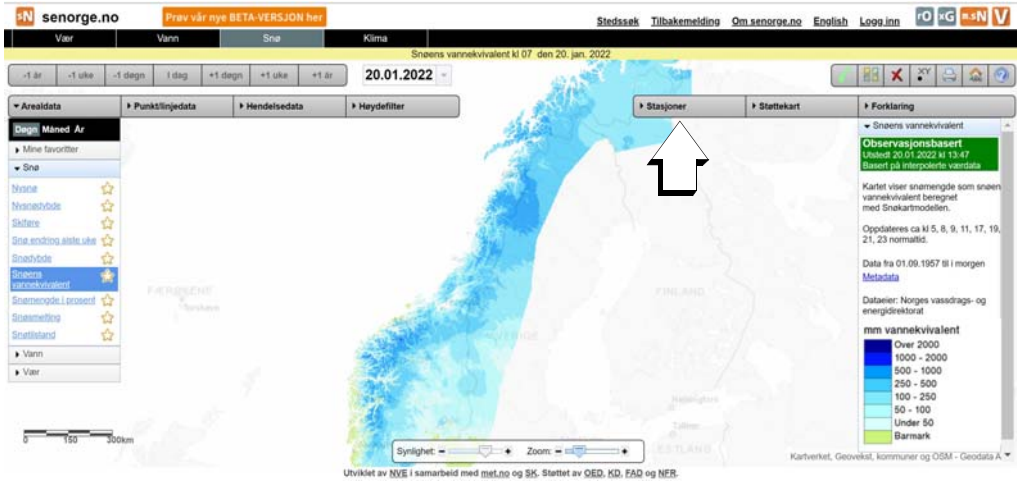
Uansett hvilken metode som benyttes så må man foreta relativt hyppige kalibreringer siden lufttrykket kan endre seg ganske raskt.

5.3.1 Eksempler på målestasjoner og bruken av dem

Her er noen eksempler på slike målestasjoner:

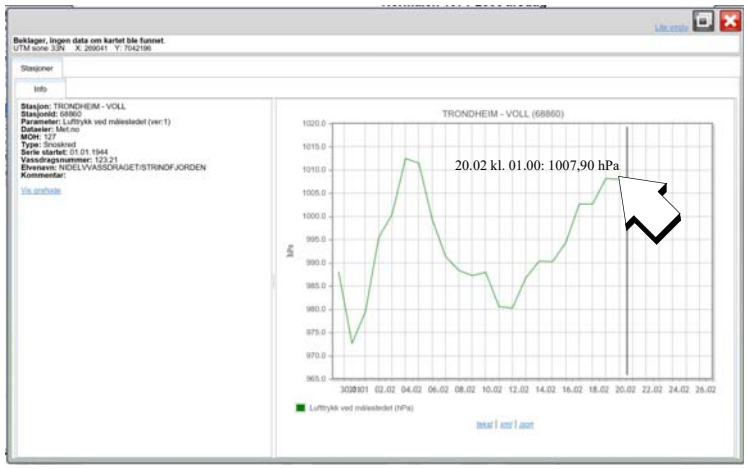
se.norge.no (http://www.senorge.no/index.html)

Dette er en åpen portal på Internett, som viser daglig oppdaterte kart over snø-, vær- og vannforhold og klima i Norge – og mye mer.



Velg "Stasjoner" og kryss av for "Lufttrykk" og velg en målestasjon nær der du befinner deg og som registrerer lufttrykk.

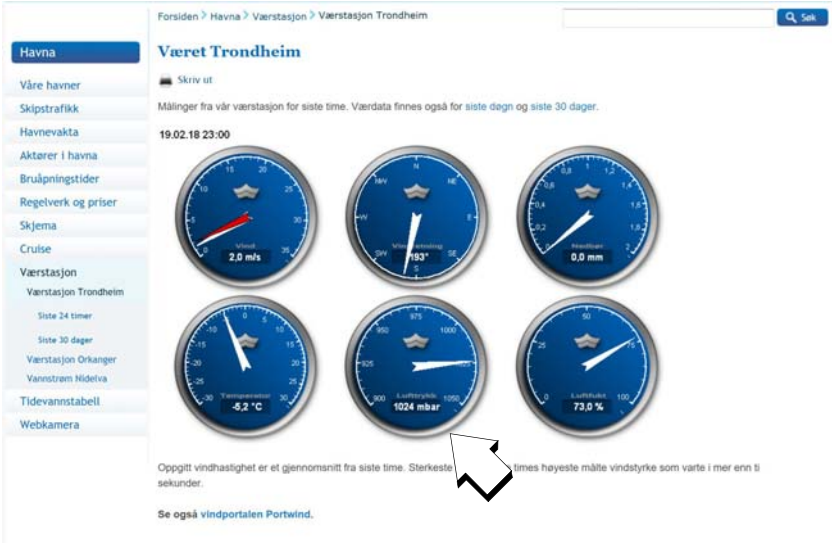
Etter å ha valgt målestasjon får man opp en graf som viser lufttrykket de siste 30 døgn. Målingene oppdateres en gang i timen så avvikene burde være til å leve med.



Værstasjoner

En kan også gå inn på oversikten over norske værstasjoner. Det finnes en uoffisiell oversikt på http://www.bjonnes.net/weatherstations/. Det er imidlertid noe varierende om disse stasjonene leverer informasjon om lufttrykk. Følgende https://portwind.no/ gir oversikt over vindretning og vindstyrke i et utvalg havner i Norge. Imidlertid oppgir de færreste lufttrykk.

Under er vist et eksempel fra Trondheim havn som også måler lufttrykket. Denne oppdateres hver time, men er relativt unøyaktig siden den oppgir målingen på nærmeste hele mbar.



5.4 Målinger utført med BMP180

Her skal vi vise to enkle forsøk utført med Arduino og BMP180.

5.4.1 Registrering av små endringer i lufttrykk med BMP180

For å illustrere hvor følsom BMP180 er så har vi gjort følgende enkle forsøk:

Vi har latt Arduino'en med BMP180 måle trykket i rommet og latt programmet skrive trykket til monitoren med følgende enkle programkode. Det er viktig at målingene gjøres så raskt som mulig. Her satt til ca. hvert 10 ms. Vi husker dessuten at responstiden til BMP180 er 7,5 ms maks:

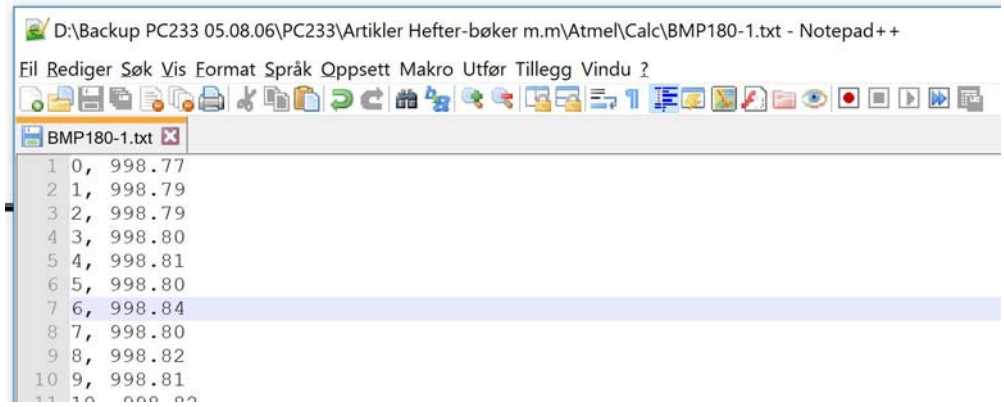
```
Serial.print(i);
  Serial.print(", ");
  Serial.println(P, 2);
  i = i+1;
  delay(10);
```

På denne måten vil vi få en lang rekke trykkmålinger skrevet ut i monitoren omtrent som utsnittet under:

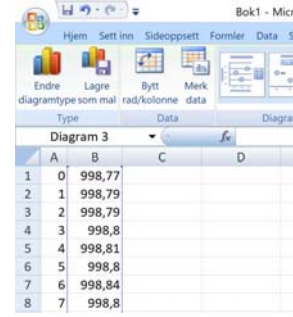
```
0, 998.77
1, 998.79
2, 998.79
3, 998.80
4, 998.81
5, 998.80
6, 998.84
...
```

I alt tok vi nær 500 målinger av trykket i løpet av ca. 30 sek. Til venstre for trykket er en telle-variabel som forteller oss nummeret til målingen. Vi må også passe på å legge inn et skilletegn mellom de to tallene på samme linje. Her har vi valgt komma, men et semikolon kan være bedre.

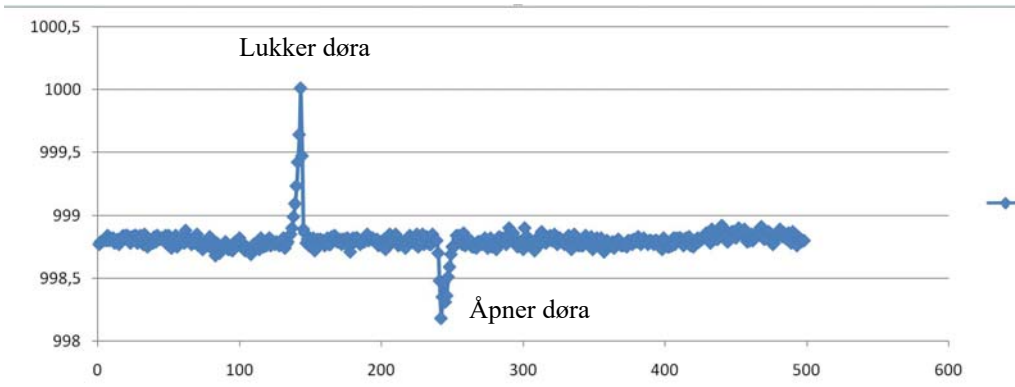
Vi kopierte tallrekken med programmet NotePad som vist på figuren under. Det fine med NotePad er at den kopierer data uten å trekke noe fra eller legge noe til.



Dernest lagret vi dataene i en tekst-fil som vi f.eks. kan importere til Excel. Dataene blir liggende som to kolonner som vist under. Her har vi sørget for at desimaltegnet er et komma, som er viktig for at Excel skal oppfatte verdiene som tall og ikke som tekst.



Dermed kan vi plotte dataene som funksjon av tiden (eller sample nummeret). På figuren under ser vi hvordan trykket endrer seg med tiden. Vi legger også merke til to topper, en som rager opp over gjennomsnittet og en som stikker ned under gjennomsnittet. Disse to avvikene skyldes at vi lukket døra til rommet for så å åpne den igjen. Ser vi nøye på grafen vil vi oppdage at utsvingene er relativt beskjedene men rager godt opp over støyen.



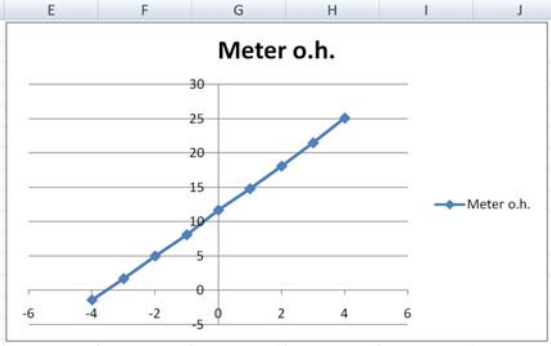
Det må innrømmes at forsøket ble gjort i et relativt lite rom og med kraftige bevegelser av døra.

5.4.2 Innendørs målinger av trykk i ulike etasjer

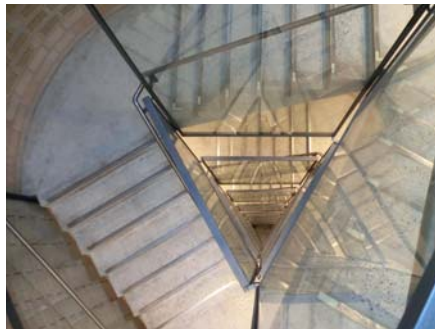
Disse målingene er gjort innendørs i Realfagbygget ved NTNU. Dette bygget har 9 etasjer hvorav fire er under bakken. Det er foretatt ni målinger ved samme høyde i hver av de ni etasjene. Disse er lagt inn i et regneark.

Figuren under viser resultatene av måleserien, dvs. sammenhengen mellom etasje og relativ høyde. Vi ser at den relative nøyaktigheten er rimelig god, men den absolutte er feil, da det kan se ut til at underetasje U4 ligger under havets overflate. Dette skyldes hovedsakelig at utrustningen ikke er godt nok kalibrert. For å få en mest mulig stabil måling har vi midlet lufttrykket over 100 enkeltmålinger. Det vil si at hver måling tar ca. 1 – 2 sekunder. Likevel ser vi at målingene varierer med 50 – 60 cm fra måling til måling ved samme høyde. Legg også merke til at vi har latt første etasje være etasje 0 slik at "x-aksen" blir riktig, dette problemet slipper man f.eks. i England.

Etasje markør	Meter o.h.	Etasje
-4	-1,4	U4
-3	1,7	U3
-2	5	U2
-1	8,1	U1
0	11,7	1
1	14,8	2
2	18,1	3
3	21,5	4
4	25,1	5



Bildet til høyre viser trappesjakta der målingene ble gjort. Det ble også foretatt målinger med laser. Denne målingen gikk fra målehøyden i øverste etasje og ned til gulvet i U4, og viste en total høyde på 31.2 meter. Dersom vi legger til høyden fra målepunktet og ned til gulvet i U4 i våre mølinger basert på trykk, får vi en total høyde på 26,3 m som er et betydelig avvik fra lasermålingen. Her er det rom for diskusjon og videre utforskning for å forklare avviket. Skyldes det lufttrykksmålingen eller lasermålingen? Kanskje kontrollmålingen burde ha vært gjort med målebånd.



6 Referanser

[1] Mer informasjon om Sparkfun Invention's kit:
https://www.sparkfun.com/products/11227

[2] Mer informasjon om I^2C-bussen:
http://www.i2c-bus.org/

[3] Mer informasjon om Serikommunikasjon på SPI-bussen:
http://en.wikipedia.org/wiki/Serial_Peripheral_Interface_Bus

[4] Koblingsskjema for Arduino UNO:
http://arduino.cc/en/uploads/Main/Arduino_Uno_Rev3-schematic.pdf.

[5] For mer informasjon om Arduino UNO R3 layout:
http://arduino.cc/en/Main/ArduinoBoardUno/

[6] For nedlasting av programeditoren IDE for Arduino:
http://arduino.cc/hu/Main/Software

[7] For nedlasting av Referansemanualen for Arduino C++
http://arduino.cc/en/Reference/HomePage

[8] For mer informasjon om BMP180 og bruk av biblioteket
https://learn.sparkfun.com/tutorials/bmp180-barometric-pressure-sensor-hookup-/installing-the-arduino-library

[9] For kjøp av BMP180
https://learn.sparkfun.com/tutorials/bmp180-barometric-pressure-sensor-hookup-/installing-the-arduino-library

[10] Skolelaboratoriets blå hefteserie:
https://www.ntnu.no/skolelab/bla-hefteserie

[11] Portal for norske målestasjoner
http://www.xgeo.no/index.html?p=klima

[12] Kommunekart, her Trondheim
https://kart5.nois.no/trondheim/Content/Main.asp?layout=trondheim&time=1550437879&vwr=asv

[13] Norske værstasjoner:
http://www.vaisala.fi/Vaisala%20Documents/Measurement%20Theory/height_calculation.pdf

[14] Omregningstabell for trykk og høyde
https://www.mide.com/pages/air-pressure-at-altitude-calculator

[15] Privat oversikt over værstasjoner:
http://www.bjonnes.net/weatherstations/

[16] Oversikt over vind i norske havner
https://portwind.no

Vedlegg A Komponentliste

Tabellen under inneholder noen av komponentene som er brukt og hvor de kan skaffes fra:

Typebetegnelse	Type komponent	Leverandør
V 4.1	Sparkfun Inventors kit V4.1 (100$)	Sparkfun USA
Adafruit SSD1306	0,96 Inch Display128 x 64 OLED LCD (kr. 59,-)	www.KultogBillig.no
BMP180	BOSCH Barometrisk lufttrykksmåler (kr. 59,-)	www.KultogBillig.no
BMP280	BOSCH Barometrisk lufttrykksmåler (kr. 31,-)	www.KultogBillig.no
Innkjøp av enkeltkomponenter istedet for Sparkfun Inventors kit		
Arduino UNO	Mikrokontrollerkort R3 REV3 m/USB kabel (128,-)	www.KultogBillig.no
1 stk. koblingsbrett	Koblingsbrett (ELFA nr: 300-91-149)	www.elfadistrelec.no
25 stk. motstand	10kOhm (ELFA nr.: 160-73-423)	www.elfadistrelec.no
25 stk. motstand	330 Ohm (ELFA nr.: 160-10-433)	www.elfadistrelec.no
10 stk. LED rød	Rød LED 5 mm (ELFA nr.: 301-00-026)	www.elfadistrelec.no
60 stk. jumpere	Breadboard Jump Wires (34,-)	www.KultogBillig.no
5 stk. TMP36	Temperatur sensor ELFA nr.: 301-29-188	www.elfadistrelec.no

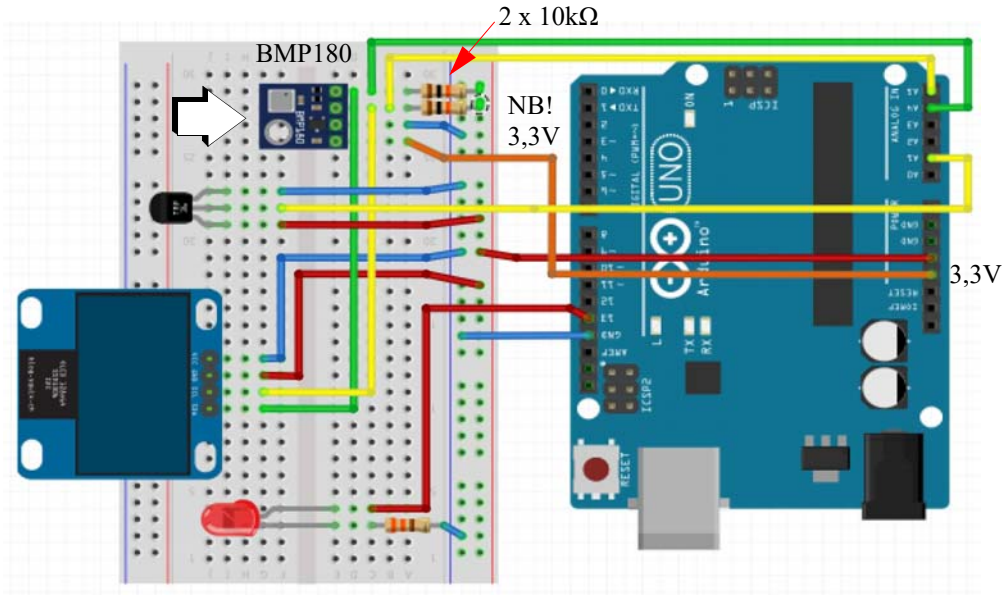
Vedlegg B Bruk av alternative komponenter

B.1 Øving 3 – Måling av lufttrykk og høyde, BMP180

I denne arbeidsøkta skal vi videreføre byggeprosjektet nok en gang ved å legge på en trykkmåler. I denne beskrivelsen velger vi å bruke BMP180 som er en eldre variant av BMP280 med omtrent like parametere. Begge kan brukes i denne laboratorieoppgaven og begge inneholder en temperatursensor. Dermed har vi to temperaturmålere som ev. kan sammenlignes.

1. Montering av trykkmåler

I^{2}C linjene (SDA, SCL) fra displayet skal nå kobles via SDA og SCL hos trykkmåleren. Siden de har hver sin adresse så skal dette gå bra. Det kan også være lurt å koble hver av linjene **til + 5V med to 10kΩ motstander**. Vi kan fjerne ledningen fra A0. MERK at **BMP180 må kobles til 3,3 V** og IKKE til 5,0 V. Dette er viktig. GND kobles til felles jordpunkt (–).



Merk at navnet på beina til BMP180 står på undersiden av kretsen.

Figuren under viser nærbilder av både BMP180 og displayet SSD1306 med tilhørende tilkoblingspunkter (bein).

2. Installasjon av biblioteket til BMP180

For å kunne kommunisere med trykkmåleren (BMP180) må vi igjen installere et bibliotek. Dette kan vi hente fra nettsiden til Skolelaboratoriet under nettadressen:

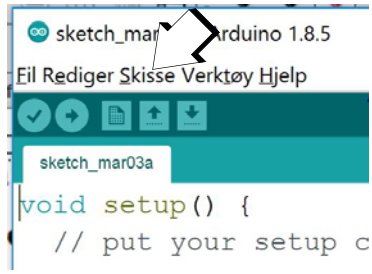
https://www.ntnu.no/skolelab/bla-hefteserie

under fanen: *Grunnkurs programmering Arduino - (CanSat)*

BMP180_Breakout_Arduino_Library-master.zip

Lagre biblioteket i nedlastingskatalogen som sist eller et annet sted der du finner det igjen.

Biblioteket installeres ved å åpen Arduino-editoren og ...



Velg: *Skisse*
Velg: *Inkluder bibliotek*
Velg: *Legg til .zip bibliotek* (øverst i nedtrekksmenyen)
Finn biblioteket i katalogen *Nedlasting* eller der du la det og ...
Velg: *Open*
... og biblioteket installeres.

3. Legg trykkmåling inn i programmet

For å lese av lufttrykket skal vi inkludere biblioteket for BMP180 i programmet. Det gjøre vi på samme måte som tidligere.

- *Inkluder biblioteket*

Biblioteket inkluderes med kommandoen:

```
#include <SFE_BMP180.h>     // Inkluder biblioteket for måling av lufttrykk
```

Kommandoen legges øverst i programmet etter de andre bibliotekene.

• *Deklarasjon av trykksensoren*

Så må trykksensoren som er av typen SFE_BMP180 gis et navn i programmet.

```
SFE_BMP180 pressure;        // Den gis navnet "pressure" og er av typen SFE_BMP180
```

Denne deklarasjonen kan legges tidlig i programmet sammen med andre deklarasjoner.

• *Initialisering av trykksensoren*

Sensoren må så initialiseres ved at det sendes noe innledende informasjon til sensoren. Denne skal legges i setup()-funksjonen mellom klammeparentesene:

```
pressure.begin();                   // Initier trykksensoren
```

Vi må anta at biblioteket inneholder adressen til BMP180 som gjerne er bestemt av fabrikken.

• *Deklarer variabler*

På dette tidspunktet nøyer vi oss med å deklarere en variabel for trykk. Vi velger å bruke *dobbel* som gjør at vi kan ta vare på store desimaltall (*dobbel* for desimaltall tilsvarer *long* for heltall). Vi velger også å deklarere den lokalt i loop()-funksjonen:

```
double p;                           // Deklarer variabelen for trykk
```

Denne deklarasjonen må gjerne legges i starten av loop()-funksjonen. Merk: At ved å legge deklarasjonen i loop()-funksjonen, så vil også variabelen *p* deklareres og kanskje nullstilles for hver runde i programmet.

• *Legg inn funksjon som leser av trykket*

For at vi trygt skal kunne lese trykket fra BMP180 så må det sjekkes at den er klar for å levere trykkmålinger. Dessuten må det korrigeres for temperatur. Alt dette gjøres i en egen funksjon som legges inn helt til slutt i programmet. Dette er funksjonen getPressure().

Vi vil ikke her gå nærmere inn på hvordan dette gjøres, men bare nevne at funksjonen sjekker status til BMP180, for så å lese av trykk og temperatur når alt er klart. Vi kopierer derfor denne funksjonen og legger den på slutten av programmet. Det skal gå greit å kopiere den fra en elektronisk utgave av dette heftet.

```
double getPressure()
{
  char status;
  double T,P,p0,a;                    // Deklarer lokale variable

  // For å foreta en trykkmåling så trengs temperaturen
  status = pressure.startTemperature(); // Forespørsel om temperaturmåling
  if (status != 0)                    // Om klar, gå videre
  {
    delay(status);                    // Vent til målingen er klar
    status = pressure.getTemperature(T);// Mål temperaturen
    if (status != 0)                  // Om målingen er vellykket gå videre
```

```
    {
      status = pressure.startPressure(3); // Forespørsel om trykkmåling

      // Forespørsel om trykkmåling, (3) høyest presisjon
      if (status != 0)                    // Om klar, gå videre
      {
        delay(status);                    // Vent til målingen er klar
        status = pressure.getPressure(P,T);// Foreta trykk og temperaturmåling
        if (status != 0)
        {
          return(P);
        }
        else Serial.println("Feil ved forespørsel av trykkmåling\n");
      }
      else Serial.println("Feil ved måling av trykk\n");
    }
    else Serial.println("Feil ved forespørsel av temperaturmåling\n");
  }
  else Serial.println("Feil ved måling av temperatur\n");
}
```

- *Bruk av funksjoner – en kort forklaring (Kjent stoff?– Hopp over)*

Før vi går videre la oss kort se hvordan en funksjon oppfører seg og hvordan den kan brukes.

En funksjon er en liten programbit som gjøre en spesifikk jobb og som kan kalles opp når man ønsker det mens man er i *hovedprogrammet* (loop()-funksjonen).

I eksempelet til høyre ser vi hovedprogrammet og funksjonen `getPressure()`, som er navnet vi har gitt funksjonen og som er av typen `double`.

Det funksjonen skal gjøre er avgrenset av {} og kalles funksjonens *kropp*. Vår funksjon skal kommunisere med BMP180 og lese av trykket *P*.

```
...
void loop()
{
...
// Innhold av loop()-funksjonen
...
p = getPressure() //Funksjonen kalles
...
}

double getPressure()
{
double P;
...
// Innhold av getPressure()-funksjonen
...
return(P);
...
}
```

Når hovedprogrammet kommer til kommandoen `p = getPressure();` vil den hoppe til funksjonen getPressure, lese trykket til BMP180 og returnere verdien (`return(P);`) til hovedprogrammet. Verdien legges inn i variabelen *p* (`p = getPressure();`).

- *Kall opp funksjonen getPressure() og les av lufttrykket*

Vi kaller opp funksjonen getPressure() fra loop()-funksjonen, som returnerer det avleste lufttrykket.

Dette gjøres med denne kommandoen:

```
p = getPressure();                    // Kall opp getPressure() og hent lufttrykket
```

... som legges inn i programmet etter temperaturmålingen.

Det målte lufttrykket ligger nå i variabelen *p* og er gitt i millibar.

- *Skriv lufttrykket til displayet*

Vi velger nå å skrive ut trykket *p* på første linje på displayet istedet for spenningen. Videre ønsker vi å skrive ut trykket med 2 desimaler. Vi ønsker følgende utskrift på displayet:

```
Trykk: 985.23 mBar
```

Vi legger merke til at trykket skrives ut med to desimaler.

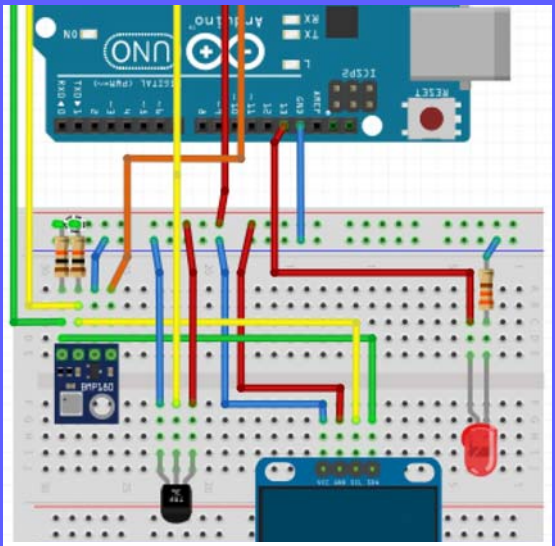
Lag programmet for å skrive ut trykket til displayet.

- *Kompiler, overfør og test programmet*

Kompiler og overfør programmet til Arduino'en. Du skal nå kunne lese av trykk og temperatur. Trykket fra BMP180 og temperaturen fra TMP36GZ.

Du kan nå løfte kretsen opp så langt kabelen rekker og se at lufttrykket endrer seg. Du vil også se at desimalene etter komma "flagrer" litt, dvs. at det er litt støy på målingene så en kan vurdere hvor mange siffer det er verdt å ta med.

Du kan nå returnere til ***Øving 4 i avsnitt 4.4, side 62***.



Heftet er en arbeidsbok knyttet til kurset: *Grunnkurs programmering, Arduino*, som først og fremst er ment som et forkurs til CanSat-kurset, men kan også fungere godt som et frittstående kurs i grunnleggende programmering av Arduino. Det har blant annet vært brukt som grunnkurs for yrkesfaglærere.

Kurset går fort over på anvendelse og fokuserer på temperatur- og trykkmålinger som er grunnleggende målinger i en CanSat. I tillegg introduseres deltagerne tidlig for utskrift til et lite OLED display som gjør utstyret mobilt slik at man kan komme igang med feltmålinger i nærområdet til laboratoriet. Ved å regne trykk om til ekvivalent høyde, får man også gjort målingene etterprøvbare, og gjør tilknytningen til CanSat åpenbar.

Denne måten å lære seg programmering på knytter programmeringsøvelsene tett opp til praktisk anvendelse og håndtering av virkelige måledata, som vi anser som viktig og som også er ett av grunnprinsippene for CanSat-prosjektet og i mange andre sammenhenger.

Nils Kr. Rossing
Dosent ved Skolelaboratoriet
E-post: nils.rossing@ntnu.no

Christoffer Stausland
Romfysiker NAROM
E-post: christoffer@narom.no

ISBN
Rev. 3.4 20.01.22

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