Rev 2.5

Nils Kr. Rossing

Videregående opplæring Arduino: Værstasjon u/løsningsforslag

NTNU

Trondheim

Institutt for fysikk

Skolelaboratoriet
for matematikk, naturfag og teknologi

Mars 2022

Grunnopplæring Arduino: Værstasjon

En guidet videregående opplæring i Arduino-programmering

Nils Kr. Rossing

Grunnopplæring Arduino: Værstasjon
En guidet videregående opplæring i Arduino-programmering

Trondheim 2022

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

Trykk: NTNU Grafisk senter

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

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

Skolelaboratoriet ved NTNU
Realfagbygget,
Høgskoleringen 5,
7491 Trondheim

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

Rev 2.5 – 26.03.22

Forord

Heftet er ment for bruk i videregående opplæring i Arduino-programmering for lærere på elektro yrkesfag. Undervisningsopplegget forutsetter at man har grunnleggende kunnskaper om Arduino-programmering, se f.eks. *Grunnopplæring Arduino – Trafikklys*[1].

"Værstasjon" er valgt som tema da måling av måledata knyttet til været synes å være noe mange elever er opptatt av. En værstasjon kan dessuten bygges opp gradvis fra kun å registrere temperatur til å registrere alle mulige egenskaper ved været. Det være seg temperatur, lufttrykk, luftfuktighet, lysintensitet, UV-stråling, vindhastighet, vindretning, nedbør og snødybde med mer. Været er dessuten noe alle både har et forhold til og har tilgang til overalt i landet. Dessuten er været et kontinuerlig fenomen hvor man kan gjøre målinger over tid og sammenligne data fra en periode med en annen.

Målingene krever stor variasjon i måleteknikker fra temperaturmåling til måling av nedbør og snødybde. Noen av sensorene kan også lages fra bunnen av som f.eks. nedbørsmåling og måling av vindretning, f.eks. ved bruk av 3D-printing. Prosjekter knyttet til temaet kan derfor gjøres både små slik at de kan gjennomføres i løpet av et par timer, eller særdeles omfattende og strekke seg over uker og måneder. En ulempe er at sensorteknologien gjerne får en særdeles dominerende rolle, mens behovet for aktuatorer kan bli relativt beskjedent. I del to har vi derfor inkludert styring av rele og motorstyring.

Vi har valgt å bruke så enkel teknologi som mulig ved at vi tar utgangspunkt i Arduino UNO, da slike kretser gjerne finnes ved mange skoler. Vi ønsker likevel å kunne koble værstasjonen opp mot Internett slik at vi kan lese av værdata fra hvor vi måtte befinne oss. Vi har derfor valgt å bruke ESP01 for oppkobling til nettstedet Circus of Things som server for innsamling og visning av data.

Vi er kjent med lærebøkene "Elektroniske kretser og nettverk" (H. Venås og S. Vangsnes) for VG1 og "Elektroniske kretser og utstyr" (O.M. Horne og B.A. Selbekk) for VG2. Disse har dels basert seg på Arduino UNO og Arduno UNO Wi-Fi. I dette undervisningsopplegget har vi forsøkt å gjenbruke komponenter fra disse to bøkene. Dette gjelder Arduino UNO, TC74 og bruk av I^2C-buss, DHT11 og bruk av entrådsbuss, LDR, HC-SR04 (ultralydsensor), display, L298 (H-bro) og rele med galvanisk skille. Vi har imidlertid ikke benyttet Arduino UNO Wi-Fi, men kombinert Arduino UNO med ESP01. Dette gir oss mulighet til å anvende nettstedet Circus of Things og vinkle prosjektene mot Internet of Things (IoT). Det er heller ikke noe i veien for at den som ønsker det kan bruke en Arduino UNO Wi-Fi i prosjektene som et alternativ til Arduino UNO.

Vi har lagt større vekt på å få fram systemaspektene ved laboratoriearbeidet, samtidig som vi forsøker å beholde fordelene ved å arbeide med mindre moduler. Overbygningen værstasjon kan gi en retning for arbeidet dersom man ønsker det.

Vi håper derfor at arbeidsheftet kan være et supplement til de nevnte lærebøkene.

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

En spesiell takk til ***Johannes Ravn Munkvold*** som har lest gjennom, kommentert og rettet opp feil i heftet, og som også har deltatt som læringsassistent under den første gjennomføringen av undervisningsopplegget i forbindelse med lærerkurs våren 2022.

Skolelaboratoriet ved NTNU
Mars 2022
Nils Kr. Rossing

Innhold

1 Innledning

1.1 Valg av teknologi

Mikrokontrollerkortet Arduino, med alle sine varianter, gir rike muligheter til å koble til sensorer og lagringsenheter av mange slag. Arduino UNO er kanskje den mest brukte i familien av kontrollerkort, men det finnes også mange andre, f.eks. Arduino Nano som er en god kandidat dersom det er viktig med små dimensjoner, eller Arduino MEGA 2560 dersom man trenger et stort antall porter. En av fordelene med å bruke Arduino UNO er at det finnes et rikt utvalg av "shield"-kort, som lett kan plugges sammen med UNO'en og gi utvidet funksjonalitet (de fleste av disse kortene kan også plugges på MEGA-en).

For å koble opp Arduino på nett er flere ulike løsninger tilgjengelig, deriblant Arduino UNO Wi-Fi, Arduino + ESP01, Arduino MEGA 2560 Wi-Fi, RobotDyn UNO o.l.

Vi har tidligere med suksess brukt ulike varianter av ESP32 som har stor kapasitet og gir lett tilgang til lokale nettverk og dermed til Internett. ESP32 er også en teknologi mye brukt i forbindelse med Internet og Things (IoT). Vi har gjennom noen år hatt stor glede av å benytte nettstedet Circus of Things (CoT), utviklet og drevet av Jaume Miralles (Mallorca, Spania). Han tilbyr biblioteker og eksempler for flere ulike teknologier deriblant ESP32, Raspbarry Pi, Arduino + ESP01, men foreløpig ikke Arduino UNO Wi-Fi. Nettstedet tilbyr også verktøy for å lage konsoller og paneler for styring og overvåking av sensornoder.

Vi har vært svært usikre på hvilken teknologi vi skulle velge til dette undervisningsopplegget da det har vært mange hensyn å ta:

- Hva norske lærebøker bruker – Arduino UNO Wi-Fi
- Hva som er lett å få tak i på det norske markedet
- Hva som har en overkommelig pris
- Hva som gir lett tilgang til IoT nettressurser
- Hvilken teknologi vi som kurstilbydere behersker på kort varsel
- Valg av relevante faglige tema

På bakgrunn av disse kriteriene har vi gjort følgende foreløpige valg:

- **Teknologi:** Arduino UNO, da denne er lett tilgjengelig og mange skoler har den. Dessuten legger aktuelle lærebøker opp til bruk av UNO.
- **Nettilkobling:** Arduino UNO + ESP01, da denne løsningen lar seg kombinere, både med Arduino UNO og bruk av CoT. Dessuten er denne løsningen relativt billig, billigere enn Arduino UNO Wi-Fi.
- **Ulempe:** Bruk av Arduino UNO + ESP01 har lite dynamisk minne som gir begrensninger mht. bruk av CoT sammen med andre biblioteker. Årsaken er at bibliotekene til CoT tar betydelig plass i minnet. Dette kan ev. løses med å bytte ut Arduino UNO (2 kbyte) med Arduino UNO Wi-Fi som har vesentlig større dynamisk minne (8 kbyte).

Ved bruk av mikrokontrollere og sensorer kan mange fagområder knyttes sammen. Det er f.eks. lett å knytte sensorteknologi til fysikk og kjemi (ev. materialteknologi), programmering og matematikk for beregning og kalibrering av målte størrelser, og datalogging. Innsamlede data kan også eksporteres til filer for videre behandling med f.eks med Excel eller Python.

1.2 Valg av tema

I dette undervisningsopplegget har vi lagt vekt på en grunnleggende innføring som gradvis bygger opp funksjonaliteten til et lite system ved hjelp av enkle moduler som settes sammen til et endelig produkt med en definert funksjon, en værstasjon.

Å bruke en værstasjon som et undervisningsopplegg kan ha mange fordeler:

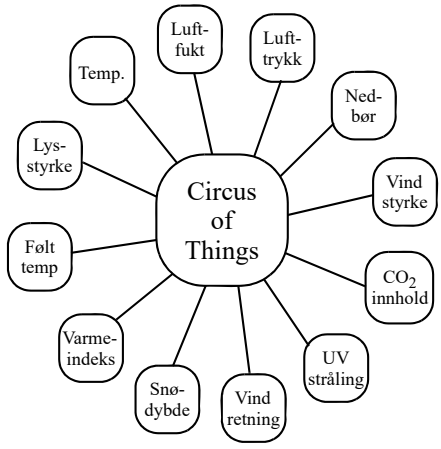
- **Lav terskel**
  Man kan begynne ganske enkelt ved kun å måle temperatur og lufttrykk og vise dette i monitoren. Til tross for enkelheten blir det et produkt med en bruksverdi.
- **Modulær oppbygging**
  Deretter kan man gradvis bygge opp et stadig mer avansert produkt ved å legge til nye funksjoner som hver for seg har en overkommelig kompleksitet.
- **Relevans**
  Alle har et forhold til været og kjenner til værstasjoner og noen av de funksjonene en slik har. Imidlertid er kanskje ikke alle kjent med mangfoldet i sensorer som kan knyttes til et slikt prosjekt.
- **Trådløse nettverk**
  En værstasjon har gjerne en ute- og en innedel. Dette gir grunnlag for bruk av trådløs kommunikasjon og tilkobling til Internett. Dessuten vil bruk av batteriløsninger og ev. lading med solcellepaneler være aktuelt for uteenheten.
- **Kreative løsninger og funksjoner**
  Mangfoldet av sensorer gir rike muligheter for allsidige utfordringer. Dette gjelder spesielt måling av vindstyrke, vindretning og ikke minst måling av nedbør i form av regn og snø. De to sistnevnte gir også rom for egne kreative løsninger.
- **Tverrfaglighet**
  Vær og værfenomener er i seg selv tverrfaglige og de berører ulike deler av livet vårt. Både store menger eller mangel på nedbør har store konsekvenser. Likeså betyr temperaturen mye å si for velvære, spesielt når den knyttes til vind (følt temperatur, forfrysning) og til luftfuktighet (varmeindeks, overoppheting).
- **Datalogging og statistikk**
  Siden værfenomenene utvikler seg over tid så vil innsamling og lagring av data over lengre perioder være aktuelt. Videre vil sammenligning og bruk av statistiske metoder være særdeles relevant, ikke minst med tanke på å oppdage langsiktige trender knyttet til klimaendringene.

1.3 Didaktisk tilnærming

Å bruke værstasjon i undervisningen er ikke spesielt originalt. Senest i januar 2022 beskriver et par universitetslærere ved universitetet i Yaounde [1], Kamerun en værstasjon for undervisningsformål ved bruk av Arduino UNO, ESP01, DHT11 og BMP180. De kobler stasjonen opp til Internett via en lokal router og sender data over til IoT nett-tjenesten ThingSpeak for å samle og vise data. Elektronikken bygges inn i en boks som gjør det mulig å plassere utstyret utendørs. De bruker så algoritmer hentet fra maskinlæring (Exponential smoothing technique) for glatting av målingene av temperatur, luftfuktighet og lufttrykk. I tillegg til å bygge og programmere stasjonen, analyserer studentene måledata som gir dem anledning til å oppdage sammenhenger mellom de tre målingene som i et tilfelle viser at en raskt stigende temperatur følger et raskt fall i luftfuktighet og lufttrykk. Artikkelen konkluderer med nytteverdien av å la studentene bygge, programmere og analysere data som en del av utdanningen, spesielt i fattige afrikanske land gir dette muligheter siden utstyret er billig, dessuten fremmer det bevissthet om vær og klima som er viktig også i Afrika.

Selv om tilgangen til avanserte værstasjoner er god i Norge så gir det et eget eierskap til prosjektet når elevene selv har bygget det opp fra grunnen av. Det har imidlertid vært kommentert at øvingsopplegg knyttet til programmering og elektronikk enten blir for små og fragmenterte til å ha en funksjonell nytteverdi, eller blir for store og komplekse dersom de skal ha en meningsfull funksjon.

I dette prosjektet ser vi for oss at små overkommelige elevprosjekter kan settes sammen til et større funksjonelt system. Istedet for at hver elevgruppe skal lage en værstasjon bestående av flere ulike sensorer, så kan den enkelte elevgruppen på to eller tre elever ta for seg en enkelt måling med tilhørende sensor som de utforsker i dybden, for så å teste, kalibrere og koble opp mot nett-tjenesten Circus of Things. Ved at hver gruppe tar for seg ulike sensorer tar man vare på mangfoldet. Ved å dele den enkelte gruppes måledata, kan man bygge opp paneler ved CoT som er langt mer mangfoldig enn det hver enkelt gruppe kan realisere.



Etter en felles oppstart kan hver enkelt gruppe spesialisere seg. Ett slikt opplegg gir også mulighet til differensiering siden noen sensorer kan være mer krevende å realisere enn andre. Mens måling av temperatur kan være relativt enkelt, så vil måling av vindretning være mer krevende. Ved at hver elevgruppe også presenterer sin kunnskap for de andre så vil hele elevgruppen få innsyn i hverandres arbeid. En annen fordel er at det stilles relativt beskjedne krav til den enkelte mikrokontroller siden den kun skal betjene én sensor i tillegg til biblioteket fra CoT.

Hver elevgruppe bør også utfordres til å bygge mikrokontrolleren med tilhørende sensor inn i en værbestandig boks slik at den kan plasseres utendørs for en periode. En utfordring som samtlige grupper vil møte er behovet for elektrisk energi som ev. kan knyttets til fornybare energikilder

(solceller, små vindturbiner, batteri), teknikker for energisparing (legge i dvale, vekking), rekkevidde (Wi-Fi) og problematikk knyttet til vær- og vannbestandighet.

Hver elev eller elevgruppe kan opprette en egen bruker hos CoT og disponerer dermed mer enn nok ressurser til å få overført og vist sine målinger. Når samtlige sensorer er koblet opp og målingene publiseres slik at alle kan nå hverandres målinger, kan de sette sammen et panel som er så kompleks de måtte ønske.

La oss se nærmere på hvilket oppdrag med tilhørende deloppdrag vi kan se for oss.

1.4 Kort beskrivelse av oppdragene knyttet til værstasjon

Oppdraget kan for eksempel formuleres slik:

> **Oppdraget:** *Velg ut de målingene du ønsker at værstasjonen skal inneholde. Følgende kan være aktuelle: Måling av lufttrykk, temperatur, luftfuktighet, lysintensitet, nedbør, vindstyrke, vindretning og snødybde. Deretter bygge, programmere og teste hver sensor individuelt før alle sammenstilles til en komplett værstasjon. I tillegg kan man etter eget ønske beregne følt temperatur og varmeindeks. Måleverdiene oversendes til nettstedet Circus of Things hvor du lager konsoller og paneler for visning av måledata. Innsamlede målinger kan også vises som funksjon av tiden. I tillegg skal det være mulig å fjernstyre releer og motorer som kan tenkes å inngå i et ventilasjonsanlegg.*

Vi kan betrakte værstasjonen som et lite system som vi gradvis bygger opp gjennom deloppdrag, slik at tilførselen av kunnskap er overkommelig for hvert del.

Under har vi ganske kort beskrevet aktuelle deloppdrag:

Del I – Sensorer og beregninger

Deloppdrag 1 – ***Temperatur og lufttrykk m/BMP280*** *(Avsnitt 6.1 side 69)*
Koble opp trykksensoren BMP280 og les av temperatur og lufttrykk. Skriv resultatene til monitoren med tekst foran verdien og benevning etter. Lag to funksjoner som leser av sensorene, skriver ut verdiene og returnerer henholdsvis lufttrykk og temperatur når de kalles fra void loop()-funksjonen..

Deloppdrag 2 – ***OLED-display m/SSD1306*** *(Avsnitt 6.2 side 73)*
Monter OLED-displayet SSD1306 og skriv ut temperatur og lufttrykk med navn på parametere og benevning. Temperatur og lufttrykk skal skrives ut under hverandre. Lag en funksjon som skriver temperatur og lufttrykk til OLED-displayet.

Deloppdrag 3 – ***Temperatur*** **m/TC74** *(Avsnitt 6.3.1 side 76)*
Monter sensoren TC74 og skriv programmet som leser av temperaturen. Skriv ut den avleste temperaturen til monitoren eller til et OLED-displayet. Legg avlesningen av temperaturen i en egen funksjon og la funksjonen returnere temperaturen.

Deloppdrag 4 – *Temperatur og luftfuktighet m/DHT11* *(Avsnitt 6.4.1 side 79)*
Koble opp og les av temperatur og luftfuktighet. Skriv resultatet til monitoren eller et OLED-display med beskrivende tekst (f.eks. Luftfukt.) og benevning (f.eks. mBar). Lag en funksjon som leser av temperatur og luftfuktighet, skriver ut resultatet og returnerer luftfuktighet.

Deloppdrag 5 – *Lysintensitet m/LDR* *(Avsnitt 6.5.2 side 84)*
Koble opp en fotomotstand (LDR) og en lysdiode (LED) og lag et program som er slik at lysdioden tenner når det blir mørkt og slukker når det blir lyst. Legg inn en terskelverdi for spenningen fra LDR-kretsen slik at omslaget fungerer når fotomotstanden dekkes til med et A4-ark. Skriv ut terskelverdien og målt spenning fra LDR-kretsen i monitoren. Legg målingen av belysningstyrken og utskrift i en funksjon og la funksjonen returnere en sann verdi dersom den detekterer mørke (TRUE) og en usann verdi dersom den detekterer lys (FALSE).

Deloppdrag 6 – *Lysintensitet m/BH1750* *(Avsnitt 6.5.3 side 86)*
Monter og lag et program som leser av belysningsstyrken i lux og skriver resultatet til monitoren eller et OLED-display. Gjør 100 målinger og finn middelverdien. Sammenlign måleresultatet med en kalibrert lysmåler. Legg avlesning, midling og skriving til monitor eller display, i en egen funksjon. Returner lysintensiteten i Lux.

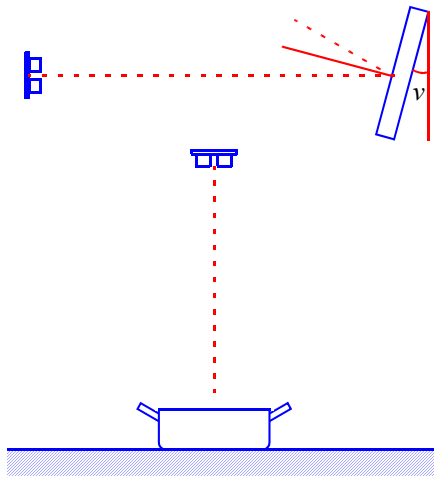
Deloppdrag 7 – Vindstyrke m/SEN-15901 (Avsnitt 6.6 side 91)
Monter og skriv et program som måler vindhastigheten (vindstyrken) i meter pr. sekund. Skriv ut resultatet i monitoren eller OLED-display med tekst og benevning. Legg målingen i en funksjon og returner vindhastigheten.

Deloppdrag 8 – Vindretning m/SEN-15901 (Avsnitt 6.7 side 95)
Koble vindretningsmåleren til mikrokontrolleren og skriv et program som registrerer riktig vindretning. La programmet angi vindretningen både som vinkel 0 – 360° og som navn på en av 16 vindretninger (f.eks. nord, nord nordøst, nordøst, øst sydøst ... osv.). Skriv resultatet til monitoren eller til OLED-displayet. Legg avlesningen og utskriften i en funksjon og la funksjonen returnere vinkelen i grader.

Deloppdrag 9 – Nedbør m/SEN-15901 (Avsnitt 6.8 side 101)
Koble nedbørsmåleren til mikrokontrolleren og skriv et program som registrerer nedbør i mm. Skriv resultatet til monitoren eller til OLED-displayet. Legg avlesningen og utskriften i en funksjon og la funksjonen returnere nedbør i millimeter. Kalibrer nedbørsmåleren med oppmålt mengde vann.

Deloppdrag 10 – Snødybde m/HC-SR04 (Avsnitt 6.9 side 104)

Plasser avstandsmåleren slik at den måler horisontalt langs bordet. Sett opp en vertikal hindring fra 20 til 50 cm fra sensoren. Skriv et program som måler avstanden mellom sensoren og hindringen og skriv resultatet ut i monitoren med riktig benevning. Bruk et målebånd og sjekk nøyaktigheten til målingene. Bruk hindringer av forskjellig materiale og undersøk hvordan dette påvirker målingene. Undersøk også hvordan målingene påvirkes av at hindringen avviker fra en rett vinkel (v) som vist på figuren over til høyre.



Plasser avstandsmåleren 40 – 50 cm over bordet og undersøk hvordan måleresultatet blir dersom hindringen er vann, våt eller tørr snø.

Beskriv en praktisk løsning for en sensor som skal plasseres utendørs gjennom vinteren.

Deloppdrag 11 – Følt temperatur (Avsnitt 6.10 side 109)

Monter temperaturmåler og vindmåler i samme oppsett og skriv et program som måler både vindhastighet og temperatur. Skriv ut resultatet av målingene i monitoren eller på displayet med riktig benevning.

Regn vindhastigheten om fra m/s til km/t og sett både temperaturmålingen og vindhastigheten inn i formelen for følt temperatur og beregn følt temperatur. Skriv ut resultatet i monitoren eller på displayet. Kontroller at følt temperatur endres med vindhastigheten.

Legg beregningen av følt temperatur i en egen funksjon med temperatur og vindhastighet som argumenter i kallet og med følt temperatur som retur.

Deloppdrag 12 – Beregning av varmeindeks (Avsnitt 6.11 side 116)

Koble opp DHT11 og lag et program som skriver ut temperatur, relativ fuktighet og varmeindeksen i °C. Varm opp sensoren med en hårføner eller lignende og undersøk om den beregnede indeksen stemmer med forventet verdi fra tabeller som viser varmeindeksen.

Del II – Styring og overvåking via nett (IoT)

Deloppdrag 13 – Mål temperatur og luftfuktighet
(Avsnitt 7.1 side 119)
Bygg opp og lag et program som leser temperatur og luftfuktighets fra sensoren DHT11. Skriv ut verdiene i monitoren.

Deloppdrag 14 – Koble opp ESP01
(Avsnitt 7.2.4 side 125)
Koble opp ESP01 på koblingsbrettet sammen med DHT11, dersom DHT11 er plassert på den foreslåtte plassen, skal det også være plass til ESP01. Gå over og sjekk at oppkoblingen er riktig, dette gjelder spesielt spenningene siden ESP01 skal ha 3,3V forsyningsspenning.

Deloppdrag 15 – Opprett konsoller og panel hos CoT
(Avsnitt 7.2.6 side 126)
Opprett en bruker ved Circus of Things og lag to konsoller, ett for temperatur og ett for luftfuktighet. Sammenstill de to konsollene i et panel. Registrer identifikasjonskoden for din bruker ("token") og kodenumrene for konsollene. Disse skal brukes når programmet hos sensornoden skal skrives.

Deloppdrag 16 – Oppkobling til Circus of Things
(Avsnitt 7.2.8 side 132)
Skriv et program som overfører temperatur og luftfuktighet til konsollene hos Circus of Things. Sjekk at data overføres tilfredsstillende og vises i konsollene hos CoT.

Deloppdrag 17 – Fjernstyring av rele fra Circus of Things
(Avsnitt 7.3 side 135)
Koble opp et rele, lag et konsoll for å slå av og på releet og skriv programmet som mottar kommandosignaler fra CoT og styrer releet ved sensornoden.

Deloppdrag 18 – Automatisk fjernstyring og bruk av terskelverdi
(Avsnitt 7.4 side 140)
Legg inn en tredje knapp, "AUTO" i trykknappkonsollet som er slik at når auto er trykket så er det temperaturen som styrer om ventilasjonsanlegget slås på eller av. Det skal også defineres et signal og lages et konsoll som gjør det mulig å sette terskelverdien for temperaturen for omslag.

Deloppdrag 19 – Styring av viftemotor med H-bro
(Avsnitt 7.5 side 143)
Monter og koble opp styrekretser for styring av en vifte (liten DC-motor). Skriv et program som slår av og på viften og endrer mellom inn- og utlufting. Dvs. vifta blåser lufta inn i 5 sekunder for så å blåse den ut i 5 sekunder.

Lag en funksjon for styring av viftemotoren der argumentet angir rotasjonsretningen.

Deloppdrag 20 – Styring av farten til viftemotor med PWM

(Avsnitt 7.6 side 146)

Lag et program som i tillegg til å snu retningen på motoren øker farten fra 0 til maks fart i løpet av de 5 første sekundene, for deretter å senke farten fra maks fart til 0 fart de neste 5 sekundene.

Deloppdrag 21 – Styring av farten til vifta fra CoT

(Avsnitt 7.7 side 148)

Lag et program som gjør det mulig å fjernstyre farten til motoren ved hjelp av en glidebryter hos Circus of Things. La glidebryteren gå fra –255 til +255. La fortegnet styre retningen til motoren, + fortegn blåser luft inn i rommet, – fortegn blåser luft ut av rommet.

Deloppdrag 22 – Sammenstill måling av luftfuktighet og styring av vifta

(Avsnitt 7.8 side 151)

Lag et program som leser av temperatur og luftfuktighet i rommet og viser resultatet av målingene i panelet på CoT. Sett terskelverdi for påslag av ventilasjonsvifta. Når luftfuktigheten i rommet overskrider terskelverdien for luftfuktighet, skal vifta blåse luft ut av rommet med en manuelt satt viftefart. Når derimot luftfuktigheten er under terskelverdien skal vifta blåse luft inn i rommet.

Deloppdrag 23 – Værstasjon – Velg en sensor til værstasjonen

(Avsnitt 7.9.1 side 156)

Velg ut en aktuell sensor som du ønsker at værstasjonen din skal inneholde. Gå til det aktuelle deloppdraget og koble opp og lag programmet. Koble sensoren opp mot CoT og lag et passende konsoll og et panel som passer for å presentere den målte størrelsen.

Deloppdrag 24 – Presentasjon av målinger over tid

(Avsnitt 7.10 side 157)

Lag et nytt panel som viser hvordan temperaturen endrer seg over tid. Legg til luftfuktighet eller lufttrykk i samme diagram.

Deloppdrag 25 – Inkluder målinger fra andre publiseringskilder

(Avsnitt 7.11 side 160)

Publiser signalene dine slik at andre får tilgang til dem. Lag et panel som viser ditt eller dine signaler og inkluder signaler fra andre.

2 Programmering med Arduino

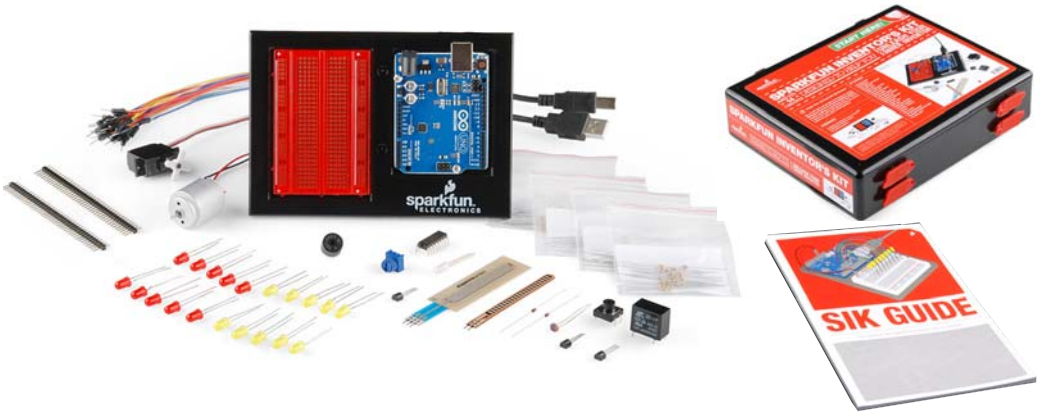
I dette kapittelet skal gi en oversikt over de komponentene vi skal bruke, beskrive hvordan vi installerer programvaren som skal til, hvordan vi bruker editoren, programstrukturen og de vanligste kommandoene i det programspråket vi skal bruke: Arduino C++.

2.1 Sparkfun Inventors kit: Grunnleggende byggeklosser

Avsnittet gir en oversikt over de elektroniske byggeklossene vi skal bruke i de følgende undervisningsoppleggene. Komponentene som er nødvendige for å bygge opp et trafikklys er også tilgjengelig i TinkerCAD.

2.1.1 Komponentoversikter

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



Med settet følger et relativt rikholdig utvalg av elektroniske komponenter. Her er ei liste over innholdet i Sparkfun Inventor's kit 3.1:

- 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

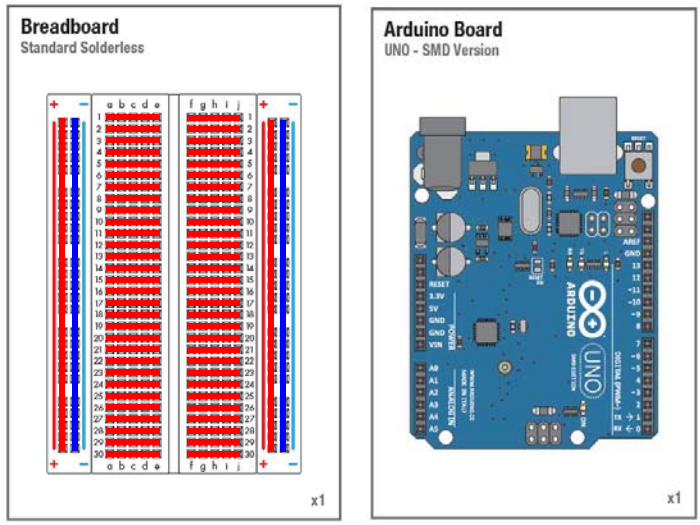
- 1 stk. Bøyesensor, Spectra Symbol 25 kOhm v/flat sensor – FS-L-0055-253-ST
- 1 stk. Membran-potensiometer, 100–10kOhm – SP-L-0050-103-ST
- 1 stk. USB kabel, 6’ vanlig A til B USB kabel
- 30 stk. Koblingsledninger, 7” Male/Male
- 1 stk. Fotomotstand, 8 kOhm (10 lux) – 1 MOhm (0 lux) topp ved 540 nm 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. Alfanumerisk LCD-display 2 x 16 karakterer
- 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 om komponentene og datablader finnes på nettsidene til Sparkfun: https://www.sparkfun.com/products/11227. Dette kitet er i dag erstattet av *SparkFun Inventor's Kit - v4.1* som er omtalt på nettsiden: https://www.sparkfun.com/products/15267. Dette kitet inkluderer også deler for å bygge en enkel robot.

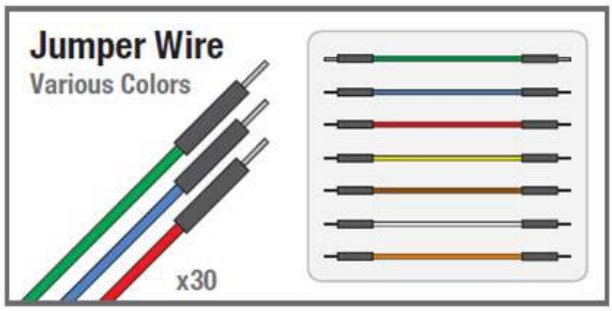
I dette opplegget skal vi bare bruke et fåtall av disse komponentene, men supplere med flere andre komponenter etter som det trengs. Disse vil bli behandlet etter som de tas i bruk.

2.1.2 Koblingsbrettet

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

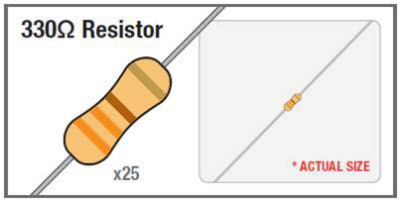


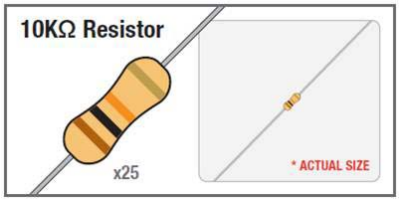
Strekene (røde) på koblingsbrettet viser hvilke koblingspunkter som er koblet sammen på undersiden. To komponentbein som er koblet til samme rad i to av hullene a – e eller f – j er elektrisk sammenkoblet. Komponentene plasseres på koblingsbrettet og forbindes med Arduino-en ved hjelp av jumperer.

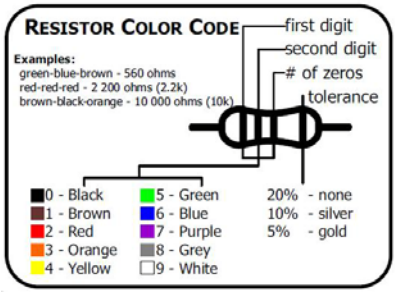


2.1.3 Motstander

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.

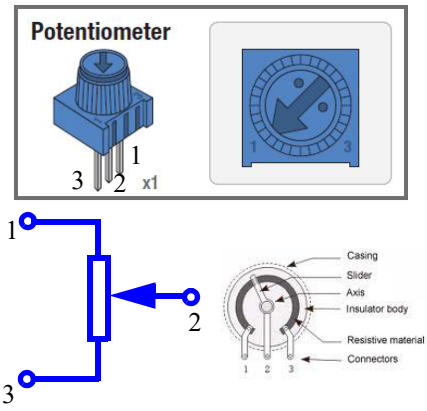
Bruksområder:

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



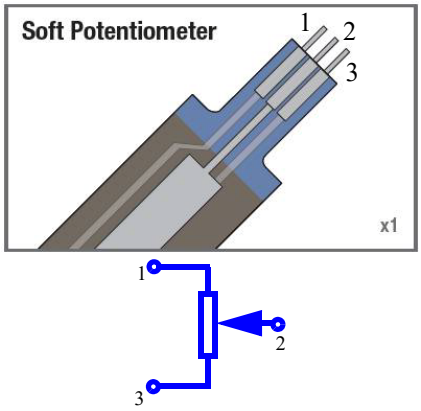
Beskrivelse:

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:

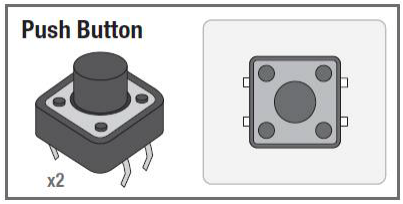
En trykkfølsom motstand fungerer på samme måte som et potensiometer. Istedet for å dreie på en knott, presser man på metallfilmen et sted mellom topp og bunn tilsvarende den spenningen man ønsker at potensiometeret skal gi ut.

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.

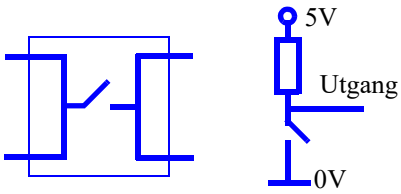
2.1.4 Sensorer

Bryter



Beskrivelse:

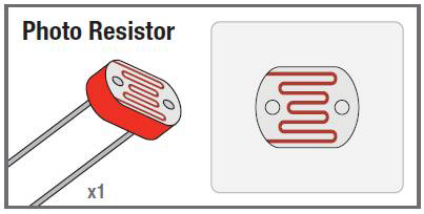
Bryterne har fire bein. De er forbundet med hverandre to og to som vist på tegningen under. Et trykk på knappen vil koble de to parene sammen. Forbindelsen opprettholdes så lenge knappen er trykket inn og brytes når den slippes.



Bruksområder:

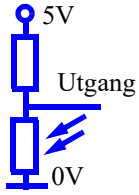
Brytere brukes til å gi en enkel på/av informasjon til kretsen, på samme måte som en lysbryter. For at kretsen skal forstå informasjonen må trykket omdannes til en spenning. Ved å koble bryteren mellom en motstand (10 kOhm) til 5 V og jord vil en på utgangen få en spenning som går fra 5 V til 0 V når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

Fotomotstand

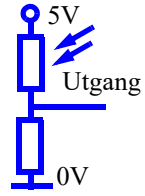


Beskrivelse:

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



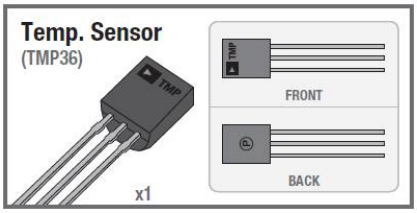
Høy spenning ut ved lys

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

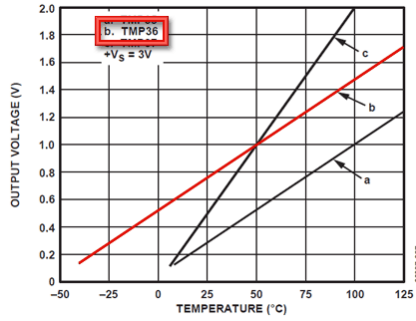


Beskrivelse:

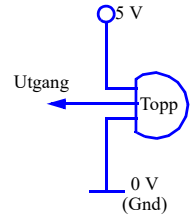
En temperatursensor (TMP36) av denne typen registrerer temperaturen og gir ut en spenning som er proporsjonal med temperaturen. OBS! Unngå å forbytte med transistorene!

Bruksområder:

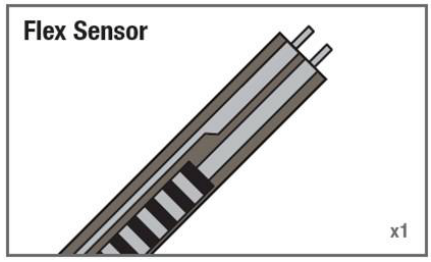
Temperatursensorer brukes i elektroniske termometre eller i termostater for å regulere en varmeovn. Den kan også brukes for å beskytte elektronikk, ved at strømmen brytes når temperaturen overskrider et maksimalt nivå. Hos TMP36 øker spenning med10 mV pr. grad C. Ved 0° C er spenningen 0,5 V.



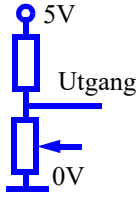
Spenning som funksjon av temperatur (TMP36 rød kurve)



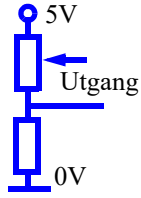
Bøyesensor



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.

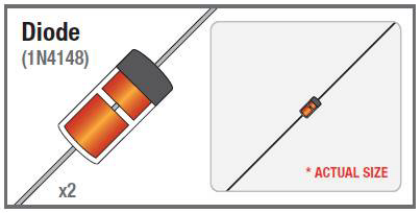
Bruksområder:

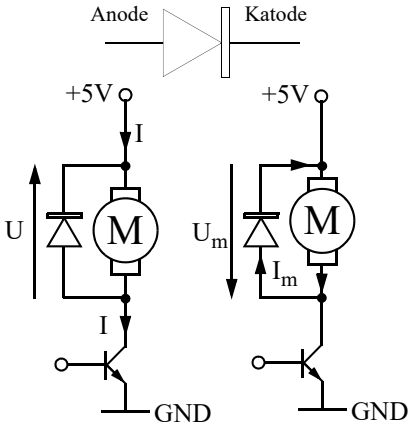
Lignende sensorer brukes i mange sammenhenger for å måle strekk eller sammentrykning i et materiale. Slik deformering kan f.eks. skyldes belastning eller nedbøyning. I denne sammenhengen går en slik sensor under betegnelsen *strekklapp*.

2.1.5 Halvledere

I denne sammenhengen er dette dioder og transistorer.

Diode





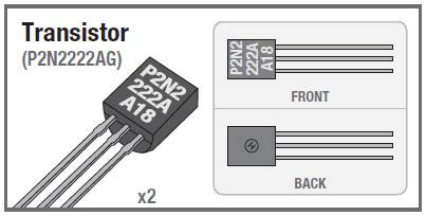
Beskrivelse:

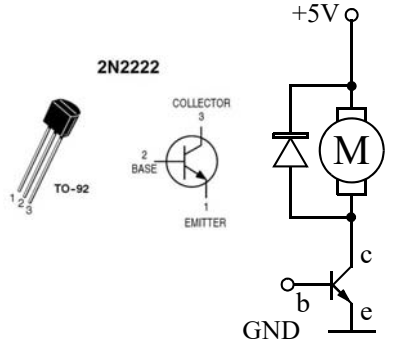
Dioden er en halvleder som kun leder strøm en vei, fra anode til katode, dvs. i pilens retning.

Bruksområder:

Dioder har et vidt bruksområde. En vanlig anvendelse er å bruke dioden som en såkalt "fly back" diode for å kortslutte strømmer som skyldes høye spenningstopper som oppstår når strømmen i en relespole, eller motor brytes brått. Dioden må derfor monteres slik at den stenger for strømmen (I) som normalt skal gå gjennom motoren eller spolen. Når strømmen i motoren brytes, oppstår en motspenning (U_m) som forsøker å hindre at motoren stopper. Denne motspenningen kan være så stor at den ødelegger transistoren som bryter strømmen. Dersom vi kobler en diode som vist på figuren til venstre, vil motspenningen (U_m) 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:

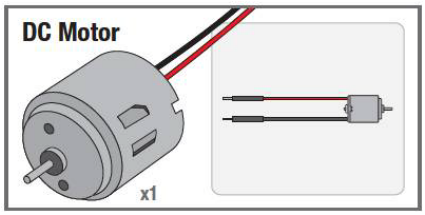
En transistor har gjerne to bruksområder. Som forsterker når basespenningen/strømmen varierer innen et lite område. Som bryter når basespenningen/strømmen er så stor at den slår transistoren på eller av.

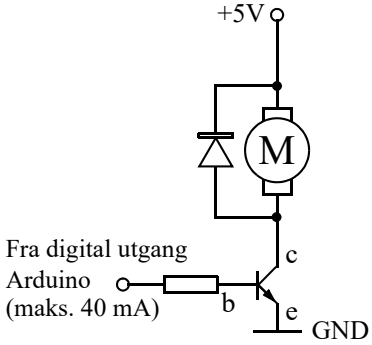
Transistorer som signalforsterker brukes i elektroniske forsterkere, radiosendere og mottakere, TV-er osv. I datamaskiner og i styringssystemer brukes den som bryter.

2.1.6 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





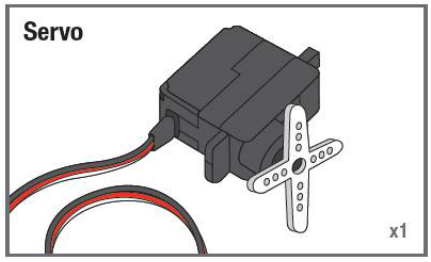
Beskrivelse:

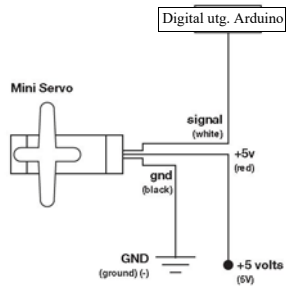
Akslingen begynner å rotere når motoren tilføres en spenning over 3 V. Motoren som følger med kitet kan brukes på spenninger opp til 40 V og strømmer på opp mot 200 mA. Siden Arduino'en ikke klarer å styre så store strømmer (absolutt maks. 40 mA), benytter vi en transistorbryter som tåler strømmen. Legg 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





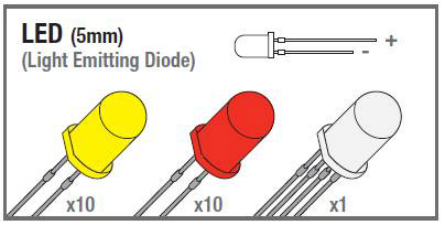
Beskrivelse:

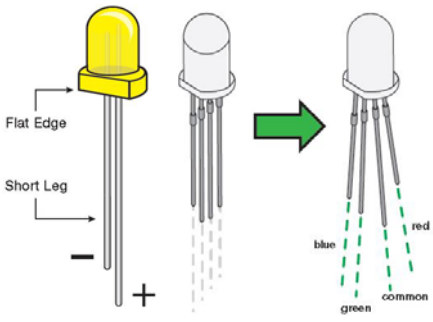
En servo er en slags motor som kan dreie akslingen en bestemt vinkel eller rotere med en definert fart. Denne servoen kan på kommando fra mikrokontrolleren dreie akslingen en vinkel på fra 0 – 180°. Dreiningen skjer ved at servoen mottar pulser, hvor lengden av pulsen bestemmer dreievinkelen. En pulslengde 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.

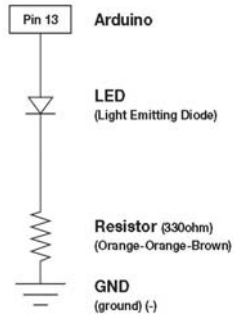
Lysdioder





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 seriemotstand dioden lett går i stykker. Typisk maks. strøm kan være fra 20 – 40 mA.

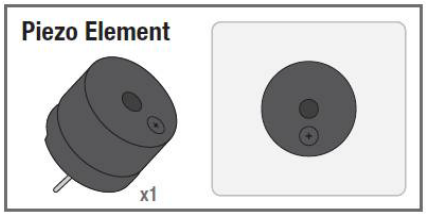


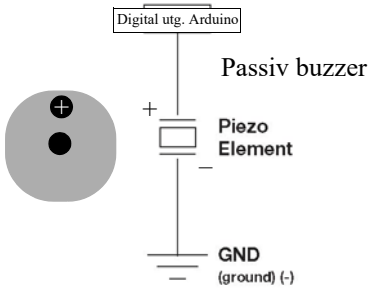
I settet er det vedlagt røde, grønne og gule lysdioder. I tillegg finnes en RGB-diode hvor en rød-, en grønn og en blå lysdioder montert i samme kapsel.

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:

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

Display (2 x 16)

Beskrivelse:

Dette er et relativt vanlig LCD display med et parallelt grensesnitt som krever 10 I/O porter. Det er derfor ikke så attraktivt å bruke etter at displayer med I^2C-buss kommunikasjon er blitt mer vanlig. Displayet har sine fordeler ved at karakterene er store og godt synlig. Displayet leveres også med bakgrunnsbelysning.

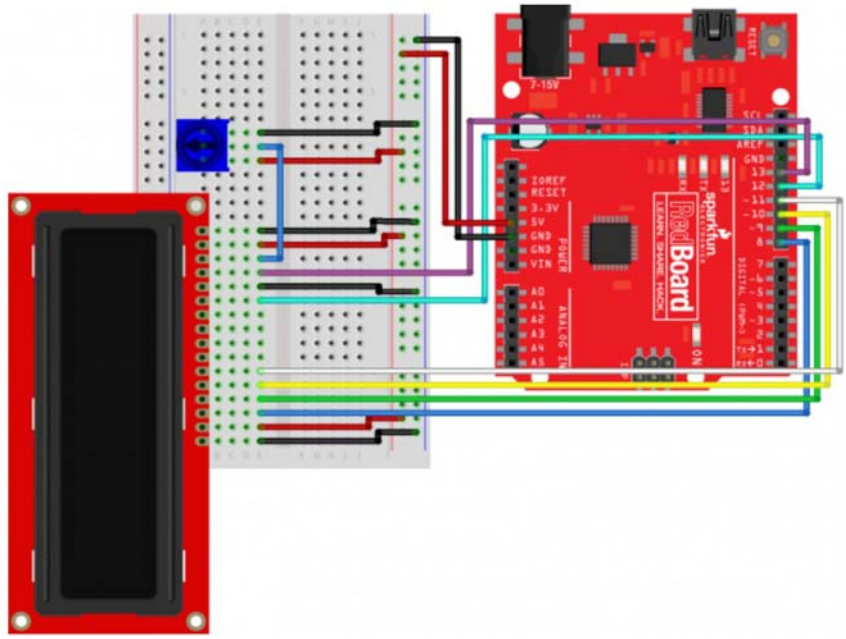
Bruksområder:

Displayet egner seg til å gir enkle beskjeder og måleresultater. Men anvendelsen blir litt begrenset siden det kun viser 2 linjer a 16 karakterer.

Pin Descriptions:
Pin 1 -- Ground
Pin 2 -- VDD Power for the LCD
Pin 3 -- Contrast Adjust
Pin 4 -- Register Select (RS)
Pin 5 -- Read / Write Select (R/W)
Pin 6 -- Enable

Pins 7-10 -- data lines d0 - d3 (not used)
Pins 11 - 14 -- data lines d4 - d7 (data transferred in 4-bits at a time)*

Pin 15 -- Backlight Power
Pin 16 -- Backlight GND (GND)

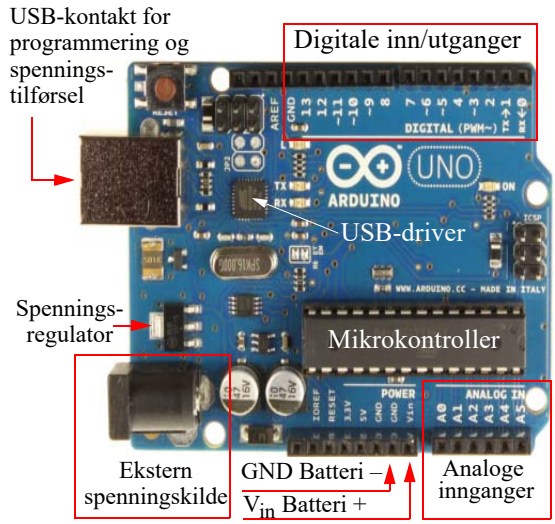


2.1.7 Arduino – mikrokontrollerkortet

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 ellers for kommunikasjon mellom PC-en og kortet. Det har også en plugg for batteri eller batteriadapter.

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. 211,5,- 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 (dynamisk minne) og EEPROM 1 kbyte.

Kortet har dessuten følgende inn- og utganger:

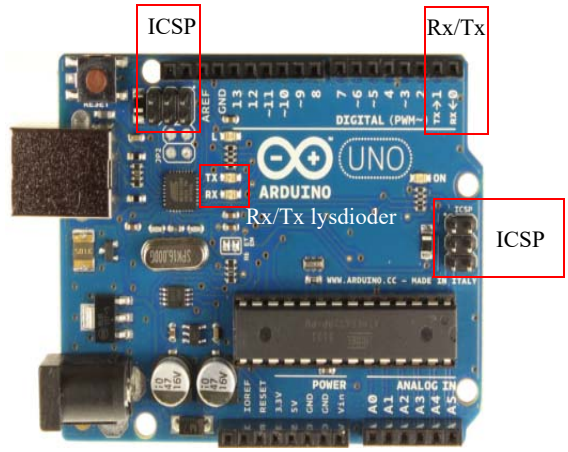
- **Digitale I/O-porter**
  Kortet har 14 digitale inn/utporter (I/O-porter) som kan programmeres til enten å være en inn- eller en utgang. Seks av disse (3, 5, 6, 9, 10 og 11) kan *pulsbreddemoduleres* (pwm), disse er merket med ~ på kortet. Maksimal strøm på hver av I/O portene er 40 mA.
- **Analoge innganger**
  Kortet har 6 analoge innganger som kan tilføres spenninger fra 0 – 5V. Spenninger ut over 5V vil ødelegge mikrokontrolleren. Disse kan også programmeres om slik at de kan brukes som digitale inn og utganger.
- **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 (–). Enkelte komponenter trenger lavere spenning som ev. kan leveres fra 3.3 V utgangen på kortet.
- **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: UART (Rx/Tx), I^{2}C-databuss og SPI-databuss.

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/

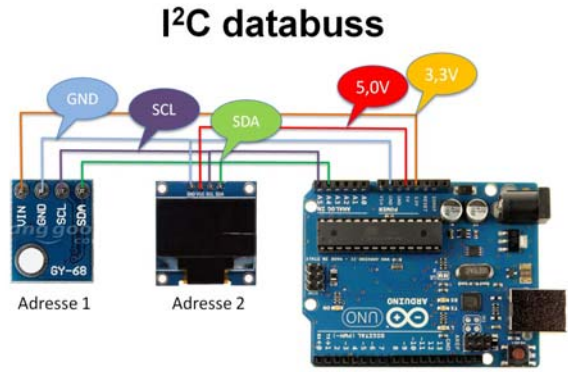


2.2 Kommunikasjon med eksterne enheter

I dette kapittelet skal vi se nærmere på ulike måter å kommunisere med eksterne enheter. La oss begynne med I^{2}C-bussen.

2.2.1 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 (SCL) og datalinje (SDA)) i tillegg til 5V (Vcc) og jord (GND).Videre er hver krets langs bussen adresserbar. Bussen er dessuten utstyrt med kollisjonsdeteksjon[2]. 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. En slik buss finnes også på Arduino og de fleste andre mikrokontrollere. Hos Arduino UNO deler bussen linjer med de analoge inngangene A4 – SDL og A5 – SCL.

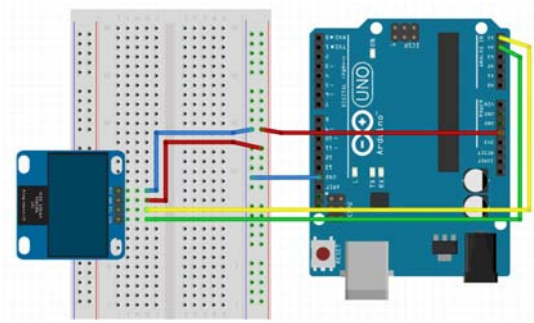


2. For mer informasjon se: http://www.i2c-bus.org/

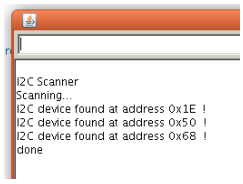
På figuren over til høyre ser vi at to enheter er koblet på de samme to kommunikasjonslinjene (SDA (data), SCL (klokke)). I dette eksempelet er det displayet SSD1306 og en trykksensor (BMP180 også kalt GY-68). I tillegg er de forbundet til spenningskildene 5,0V og 3,3V og jord (GND). Det er også vanlig å koble en motstand mellom SDA og SCL og opp til 5V, men det går også ofte bra uten når avstandene langs bussen er korte.

Enhetenes adresse

Som nevnt må hver enhet (komponent) langs bussen ha sin egen adresse fra 0 – 127, eller 7 bit[3]. Denne skrives ofte heksadesimalt og kan da uttrykkes som to tall hvor hvert av tallene er siffer fra 0 – 15 (dvs. at hvert siffer uttrykker 4 bit og benytter sifrene 0 – F i et såkalt 16-tallsystem). F.eks. vil vårt display gjerne ha adressen 3C (skrives 0x3C[4] som omregnet til desimale tall blir 3x16 + 12 = 60). Adressene til en komponent er ofte gitt ved produksjon med mulighet til å velge en eller flere nærliggende adresser. For vårt display kan vi alternativt velge 0x3D.



Dersom man er usikker på hva enhetens adresse er kan man benytte et programhjelpemiddel for scanning av adressen. Man kobler opp enheten eksempelvis som vist for vårt display, som vist på figuren til høyre, og installerer scanningprogrammet: *i2C_scanning* i Arduino'en. Programmet kan kopieres fra denne siden: https://playground.arduino.cc/Main/I2cScanner/ Programmet vil lete etter adressene til de enhetene som er koblet på bussen og vise resultatet i monitoren som vist på figuren under. Programmet er også vedlagt i vedlegg E.1 side 181.



Som vi ser av figuren kan flere enheter med ulike adresser være koblet til bussen, og programmet viser alle som det finner.

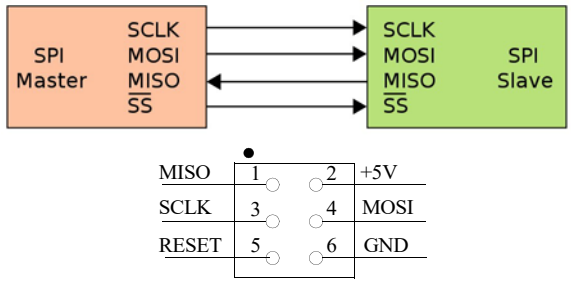
3. Det finnes også en 10 bits utgave der det teoretisk er mulig å adressere over 1000 enheter langs bussen dersom de har forskjellige adresser. Enheter med både 7 og 10 bits adresser kan kobles til samme bussen.
4. Det finnes flere varianter av notasjonen som brukes for å vise at det er snakk om heksadesimale tall. Eksempel vis kan man skrive 3Ch eller $3C_{16}$ eller som her 0x3C som gjerne er en notasjon som benyttes i ulike programmeringsspråk f.eks. i C eller C++ som er det språket Arduino ofte anvender. Samme notasjon brukes også i Python og i Java. Også \x eller %X kan i noen sammenhenger benyttes. https://en.wikipedia.org/wiki/Hexadecimal

Biblioteket

For å kunne kommunisere på I^{2}C-bussen må vi ta i bruk et eget bibliotek med kommandoer. Dette biblioteket trenger ikke installeres da det alt ligger som standard i Arduino-programvaren. Det eneste som er nødvendig er å inkludere biblioteket i programvaren vår. Dette gjørs ved å inkludere følgende kommandoer i starten av programmet vårt:

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

2.2.2 SPI-databus (eller ICSP):



Serial Peripheral Interface buss (SPI-bus). Er en firelinjes databuss brukt for å overføre data til utstyr utenfor mikroprosessoren (periferutstyr). Databusen 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 (Serial clock) 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[5]. Arduino UNO har to slike busser, en ICSP og en ICSP1 som også er koblet til D11 – D13.

2.2.3 Entråds-buss [10]

Dette er et kommunikasjonsformat som ble utviklet av Dallas Semiconductor Corp. for enkel lav-hastighetskommunikasjon mellom en *master* f.eks. Arduino, og en sensor eller en minnebrikke, *slaven*. Datahastigheten er normalt 16.3 kbps (kilo bit pr. sekund) som kalles “Standard speed”. Det finnes også en variant som opererer på høyere hastighet, denne går under betegnelsen “Overdrive”. De fleste komponentene som benytter denne type dataoverføring tillater begge hastighetene.

Bussen bruker halv-dupleks, dvs. at den kan både sende og motta data, men aldri begge deler samtidig[6]. Overføring av data må derfor avtales mellom master og slave. Slaven inneholder ofte ikke egen spenningsforsyning, men får tilført spenning fra masteren når det ikke skjer kommunikasjon, vi sier at bussen er i en “idle”-tilstand (hviletilstand). Under hviletilstanden trekkes ledningen høy av en motstand slik at spenningen på bussen blir liggende nær batterispenningen. At den betegnes

5. http://en.wikipedia.org/wiki/Serial_Peripheral_Interface_Bus
6. Når en kan sende og motta samtidig kalles sambandet for *full dupleks*.

som entråd ("1-wire") betyr i praksis at kommunikasjonen skjer over en leder og med tilhørende jord (GND). Et alternativ er at supplyspenning og jord overføres på to egne ledere, og at data overføres på en tredje ledning.

Ved innledningen til en dataoverføring så trekker master linjen lav i minimum 480µs før den slippes høy igjen. Deretter svarer slaven ved å trekke linjen lav. På denne måten signaliseres det at den er klar til å motta en melding samtidig som det skjer en synkronisering mellom master og slave. Synkroniseringen skjer ved at slaven er i stand til å sample det mottatte datasignalet til riktig tid.

Den etterfølgende dataoverføringen skjer i standardiserte tidsintervaller i henhold til den standardiserte datatakten (16.3kbps).

Fordelen med en slik dataoverføringer at den krever få linjer, og ingen egen linje for overføring av spenning til komponenten, selv om også det praktiseres i mange tilfeller.

Grunnen til at vi nevner denne bussen er at vi skal anvende den i noen tilfeller i forbindelse med styring av flerfarge lysdioder (RGB WS2811) og ganske populære fuktighets- og temperatursensorer (DHT11 og DHT22).

2.3 Kort innføring i bruk av Arduino editoren (IDE)

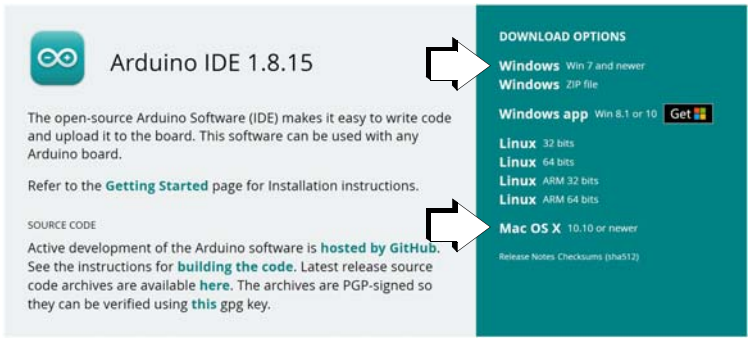
Dette er en meget kortfattet innføring i bruk av editoren til Arduino. Hopp over om dette er kjent.

1. **Installasjon av programvaren**

   Det er normalt ikke nødvendig å installere en ny utgave dersom en gammel er installert i maskinen fra før. Eldre versjoner vil sannsynligvis fungere godt.

   Programvaren hentes fra:

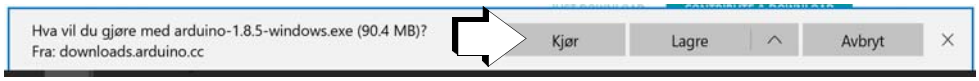
   https://www.arduino.cc/en/Main/Software



   Velg: *Windows installer* (For Mac – *Mac OS X*)

   Velg: *Just Download*

Velg: *Kjør*



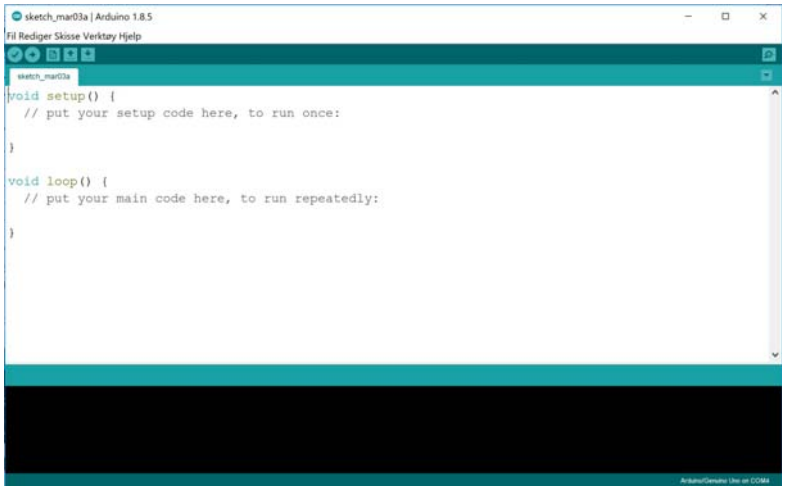
Velg: *Følg prosedyren i installasjonen*

Velg: *Å installere drivere*

2. Start programmet

Klikk på ikonet til Arduino og start programmet.

Du vil forhåpentlig se følgende programvindu åpne seg etter en liten stund.



3. Koble Arduino'en til en USB-kontakt

Se på menylinjen øverst til venstre.

Velg: *Verktøy.*
Velg: *Kort.*
Velg: *Arduino/Genuino UNO.*

Velg: *Verktøy.*
Velg*: Port.*
Velg: Det høyeste COM nummeret, eller velg porten merket med Arduino/Genuino UNO.

Det skal nå være opprettet kontakt mellom program-editoren (IDE) og Arduino-kortet.

4. Kort brukerveiledning for program-editoren

De viktigste kommandoene for å betjene bruken av editoren er oppsummert under:

Sjekk at koden er fri for skrivefeil (syntaksfeil)

Kompiler og send koden til kortet

Åpne en ny *skisse*[7] og start med blanke ark

Last opp en programskisse fra disken

Lagre programskisse på disken

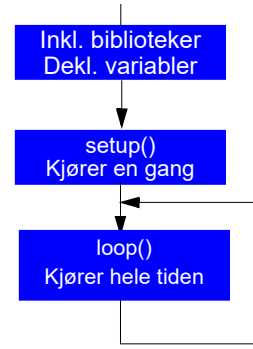
2.4 Programstruktur og bruk av funksjoner

2.4.1 Programstruktur

Alle Arduino-programmer består av to hovedfunksjoner:

7. Skisse er Arduinos navn på et program som skrives i editoren

- `void setup()` som kjøres bare *en* gang hver gang programmet starter. Dette skjer etter at programmet er lastet opp, når man trykker på restart-knappen eller åpner monitoren.
- `void loop()` er en funksjon som gjentas så lenge Arduino'en har spenning.
- **Biblioteker og deklarasjoner** av globale variabler plasseres gjerne helt i starten. ***Globale variabler*** kan brukes i alle funksjoner. ***Lokale variabler*** deklareres i den enkelte funksjon og kan kun brukes innenfor denne funksjonen.
- Det er også vanlig å skrive ***egne funksjoner*** som gjerne legges på slutten av programmet, utenfor og etter `void loop()`-funksjonen(), men kan i prinsippet legges hvor som helst utenfor `void setup()`- og `void loop()`-funksjonene.



De to hovedfunksjonene i Arduino-programmene omsluttet av *klammeparenteser*. I `void setup()`-funksjonen initierer mikrokontrolleren og ev. sensorer som er tilkoblet, mens selve programmet legges under `void loop()`-funksjonen, som vist på figuren under.

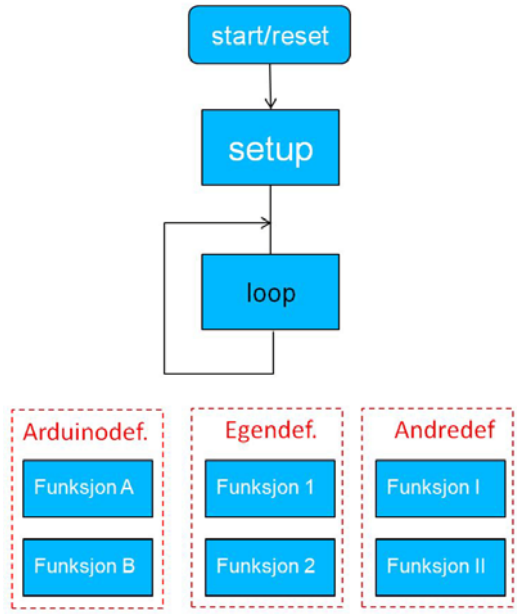
```
include <bibliotek.h> / Inkluderer spesielle biblioteker
// Deklarer globale variabler

void setup()
{
    // Koden i setup() kjører bare ved start
.... pinMode(<pin>, in/output);
}

void loop()
{
    // Deklarer lokale variable
    // Programlinjer

    funksjon1( ); // Kaller funksjon 1
...
}

void funksjon1()
{
    // Deklarer lokale variabler
    // kode
}
```



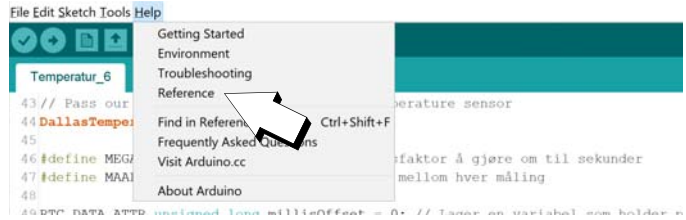
`setup()` og `loop()` er *navnet* til funksjonene, mens de to klammeparentesene {} omslutter *kroppen til funksjonen* hvor selve programmet ligger.

I tillegg til disse to faste funksjonene så bruker vi funksjoner fra Arduino's eget bibliotek (kalt Arduinodef. på figuren), funksjoner som andre har laget (Andredef.) og vi kan lage våre egne funksjoner (Egendef.). Funksjoner som andre har laget og som de ønsker å tilby fellesskapet, samles ofte i *biblioteker*, som kan lastes ned, installeres og gjøres tilgjengelig for programmerere. For nærmere beskrivelse av hvordan man lager og bruker egne funksjoner, se avsnitt 2.5.13 side 48.

2.5 Viktige kommandoer

Referansemanualen til Arduino C++ for bruk ved programmering av Arduino-kontrollerkort finnes på følgende nettadresse: http://arduino.cc/en/Reference/HomePage

Referansemanualen kan også nås via *Hjelp* på menylinjen i programeditoren som vist på figuren til høyre.



2.5.1 Initiering av dataoverføring til PC

Under uttestingen og visning av innhold av variabler kan det være praktisk at data leses tilbake til monitoren i Arduino-editoren. Datahastigheten settes opp i `void setup()`-funksjonen med kommandoen: Serial.begin(9600); her er datahastigheten satt til 9 600 baud[8] (her ca. 9600 *bit* pr. sekund) som vist under:

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

8. Baud rate - Angir hvor mange symboler som kan overføres pr. transmisjonsperiode. For et binært system vil baud-raten tilsvare bit pr. sek. For mer komplekse former for modulasjon (f.eks. QPSK) vil hver transmisjonperiode kunne overføre flere bit (2, 4 eller flere), dog på bekostning av økt følsomhet for støy.

I noen tilfeller kan det være hensiktsmessig å øke overføringshastigheten. 115 200 tegn pr. sekund er også vanlig å bruke. Pass på at mottakerhastigheten for monitoren er satt til samme hastighet. Dette gjøres nederst til høyre i monitoren som vist på figuren under.

Kommandoer for å skrive tilbake til PC, dvs. til monitoren i program-editoren:

Følgende kommandoer skriver en variabel eller en tekst tilbake til terminalvinduet (monitoren) 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 skjerm
                            // neste skrivekommando skriver på ny linje
```

Det er også mulig å kombinere tekst og variabler i samme print-kommando:

```
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
```

2.5.2 Kommentarer:

Kommentarer kan skrives hvor som helst og begynner med //

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

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

Ønsker man å lage kommentarer over flere linjer kan dette gjøres slik:

```
/*
Alle tre linjene
```

```
vil bli betraktet
som kommentarer
*/
```

Midlertidig å *kommentere bort* programlinjer kan også være et nyttig hjelpemiddel under feilsøking.

2.5.3 Bruk av variabler

Bruk av variabler er en praksis som gjør programmering særdeles slagkraftig.

> *Vi kan betrakte variabler som oppbevaringssteder ("skuffer") for tekst eller tallverdier med navn og type. Innholdet vil som oftest være ukjent når vi skriver programmet, men vi reserverer plass til verdiene. Navnene på variablene bør gjenspeile hva de representerer slik at det blir lettere og lese og forstå koden.*

La oss deklarere variablene "`tempForskjell`", "`tempStart`" og "`tempSlutt`" som float (desimaltall). Vi kommer da til den andre viktige egenskapen:

> *Vi kan behandle og utføre beregninger på variablene uten at verdiene til variablene er kjente.*

```
tempForskjell = tempSlutt – tempStart;
```

Som variabelnavnene indikerer, så ønsker vi å beregne temperaturforskjellen mellom start- og sluttidspunktet. Dette føres oss til den tredje viktige egenskapen:

> *Vi kan bruke variabler som lagringsplass for verdier over tid.*

Deklarasjon av lokale og globale variable

I programspråkene C og C++ må alle variabler deklareres før de kan brukes. Deklarasjonene må inneholde *type* og *navn* på variabelen og gjøres gjerne i starten av programmet før `void setup()`-funksjonen. Variabler deklarert utenfor funksjonene blir ***globale variabler***, dvs. at variablene kan brukes i alle funksjonene og beholder innholdet uavhengig av hvor de brukes.

Det er også verd å merke seg at ulike variabeltyper opptar forskjellig plass i minnet, se listen under. Dette er en grunn til ikke å bruke unødig plasskrevende typer nxår det strengt tatt ikke er nødvendig.

Deklarering av variabler kan også gjøres *innenfor hver* funksjon. Slike variabler gjelder da bare innenfor denne funksjonen og kalles ***lokale variabler***. Under er vist deklarasjon av de vanligste typer variabler. I dette tilfelle vil de bare gjelde innen funksjonen `void 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 bytes heltallsvariabel f (32 bit)
                   // uten fortegn
```

```
    boolean g;      // deklarasjon av en boolsk variabel g
                    // som kan ha verdiene 0 eller 1
    // Her følger resten av programkoden i funksjonen loop()-
}
```

Dersom vi deklarerer variabler i begynnelsen av loop()-funksjonen så vil de bli redeklarert for hver runde i loopen, hvilket betyr at vi må regne med at de vil miste den verdien de har.

2.5.4 Etablering og bruk av array[9]

Noen ganger har vi bruk for å lagre data i en tabell, eller å lage et oppslag. En slik tabell kalles gjerne et *array* i programmering. Et array kan være av en eller flere dimensjoner.

En-dimensjonalt array

Et en-dimensjonalt heltalls-array med navnet *tabell* (fritt valgt) med *n* elementer deklareres slik:

```
int tabell[n];
```

Vi kan også fylle det med verdier idet vi deklarerer det:

```
int tabell[n] {0, 1, 2, 3, 4, 5 ... , n-1};
```

Her har vi fylt arrayet med tallene 0 til n-1, det kunne selvfølgelig ha vært andre heltall. Elementene i arrayet nummereres fra 0 til n-1. Når man ønsker å lese ut verdien fra ett spesifikt element, f.eks. element nr. *i*, så kan man skrive dette slik:

```
int element_i = tabell[i];
```

To-dimensjonalt array

På samme måte kan man definere et to-dimensjonalt array med f.eks. desimaltall (float):

```
float tabell_2D[m][n];
```

Vi kan tenke på dette som en tabell med m rader og n kolonner, eller om vi vil, med n elementer i hver rad. Vi kan fylle tabellen med verdier slik:

```
float tabell_2D[m][n] {
                       {00, 01, 02, 03, 04, ... 0m-1},
                       {10, 11, 12, 13, 14, ... 1m-1},
                         ⋮
                       {n0, n1, n2, n3, n4, ... nm-1}
                      };
```

Når man ønsker å lese verdien fra ett spesifikt element, f.eks. element nr. i, j, så kan man skrive dette slik:

```
int element_ij = tabell[i][j];
```

9. https://www.tutorialspoint.com/arduino/arduino_multi_dimensional_arrays.htm

2.5.5 Pause-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)
```

Dette medfører imidlertid at heller ingenting annet kan gjøres i denne tiden. Har man ikke råd til å miste denne tiden, bør man finne andre løsninger, f.eks. bruk av `millis();` som vist i neste avsnitt.

2.5.6 Bruk av millis()

Dersom vi vil unngå at programmet stopper helt opp mens vi venter på at et tiden er inne for å utføre en handling, kan vi bruke `millis()` istedet for `delay()`. `millis()` angir til enhver tid tiden i millisekunder fra vi slo på strømmen eller resatte programmet og oppdateres kontinuerlig.

La oss anta at vi vil at programmet skal utføre en handling hvert 1000 ms (1 sekund) i tillegg til at programmet skal gjøre en del andre ting mellom disse tidspunktene. I så fall kan vi gjøre det slik:

```
long naaTidspunkt;


setup()
{
  naaTidspunkt = millis();
  ...
}

void loop()
{
  ...
  if(millis()- naaTidspunkt > 1000)
   {
    // Gjør det som skal gjøres hvert 1000 ms

    naaTidspunkt = millis(); // Sett nytt nåtidspunkt
   }
 // Resten av programmet
}
```

Dette er f.eks. aktuelt der man ønsker å telle og vise sekunder på et klokkedisplay hvert sekund.

Vi definerer en variabel `long naaTidspunkt`; Grunnen til at vi definerer variabelen som type long er for å unngå at variabelen skal tildeles verdier som er utenfor maksimal størrelse til variabelen. Bruker vi en `int` vil denne nå grensen for sitt området i løpet av bare ca. 32 sek. For bruk av `long` vil dette ta i underkant av 25 døgn. Bruker vi `unsigned long` vil det ta over 49 døgn.

I setup()-funksjonen setter vi `naaTidspunktet` lik `millis()`. I `void loop()`-funksjonen tester vi om det er gått 1000 ms siden vi tok vare på `naaTidspunktet` sist. Dette gjør vi ved å

trekke det sist oppdaterte `naatidspunktet` fra den løpende `millis()`. Dersom differansen er større enn 1000 ms, så utfører vi vår handling samtidig som vi oppdaterer `naaTidspunktet` til løpende `millis()`.

2.5.7 Aritmetiske og logiske 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
eksp = pow(g,e);// Et grunntall g, opphøyd i en eksponent e (g^e)
```

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

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

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

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

Setter vi inn tall, f.eks. a = 2 og b = 3 vil koden over gi at a = 5 etter at kodelinjen er utført som i og for seg gir mening.

I tillegg til de aritmetiske operasjonene har vi også tre logiske operatorer:

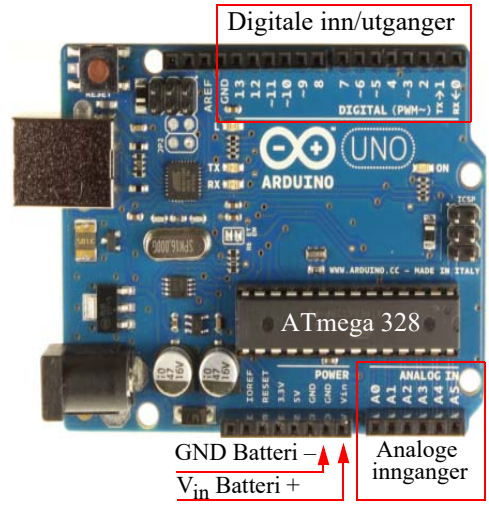
```
&&   // Uttrykker logisk "og", brukes når to betingelser må være sanne
||   // Uttrykker logisk "eller", brukes når en av to betingelser må
     // være sanne
!    // Negasjon av logisk verdi !sant er lik ikke sant
```

2.5.8 Digitale porter

Definer digitale porter som inn- eller utganger:

En port er en fysisk terminal 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 hos strømforsyningen eller batteriet).

Kontrolleren ATmega 328 (Arduino UNO) har en rekke *porter*, 14 digitale (0 – 13) og 6 analoge (0 – 5, som også kan brukes som digitale



I/O-porter). De digitale portene kan være innganger eller utganger. Hver port må derfor defineres som en inn- eller utgang. Dette gjøres gjerne i `void setup()`-funksjonen:

```
void setup()
{
   pinMode(8,OUTPUT);       // Definerer port (pinne) 8 som utgang
   pinMode(7,INPUT);        // Definerer port 7 som inngang
   pinMode(6,INPUT_PULLUP); // Definerer port 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 for Arduino og 3,3V for ESP32).

Les fra og skriv til en digital port:

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

```
boolean boolsk;          // Definerer den boolske variabelen boolsk kan ha
                         // verdien 0 (usann) eller 1 (sann)
int heltall;             // Definerer en heltallsvariabel heltall,
                         // kan meget vel også brukes for 0 og 1

void loop()
{
digitalWrite(8, HIGH);   // Setter port 8 høy (5V ev. 3,3V på ESP32)
digitalWrite(8, LOW);    // Setter port 8 lav (0V)
boolsk = digitalRead(7); // Leser den digitale verdien på port 7
                         // og setter den inn i variabelen bolsk
heltall = digitalRead(6);// Leser den digitale verdien på port 6
                         // og setter den inn i variabelen heltall
}
```

Det er imidlertid vanligere å bruke heltallsvariabler (krever 2 byte á 8 bit hver) for å holde digitale verdier lest fra innganger. Man sparer imidlertid litt lagerplass dersom man bruker boolske variabler (krever 1 byte lagerplass).

2.5.9 Analoge porter

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

Syntaksen for lesing fra en analog inngang er slik:

```
<variabel> = analogRead(<analog port>);// Den analoge porten kan ha verdier fra
                                       // 0 – 5V hos UNO ev. 0 – 3,3V hos ESP32
```

Eksempel 1:

```
Int verdi;             // Deklarerer variabelen verdi
verdi = analogRead(0);// Leser digital verdi fra analog inngang 0,
                       // verdien 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 at *Serial.print();* skriver *uten* påfølgende linjeskift, mens *Serial.println();* skriver ut *med* påfølgende linjeskift.

Eksempel 3:

```
void loop()
{
   int digitemp;               // Deklarerer digitemp som heltallsvariabel
   float anatemp;              // Deklarerer anatemp som float
   digitemp = analogRead(5);   // Leser av temperatursensoren på analog-kanal 5
   anatemp = digitemp * 500/1024;// Regner om til spenning og grader
   Serial.println(anatemp,2);  // Skriv resultatet tilbake til Arduino
                               // monitoren (PC) med 2 desimaler
}
```

De analoge portene er tilkoblet en analog til digital omvandler (AD-konverter) som gjør om spenningen 0 – 5V til en digital verdi fra 0 – 1023 (0000000000 – 1111111111). Dvs. AD-konverteren har 10 bits nøyaktighet. Tilsvarende kan ESP32 håndtere analoge spenninger fra 0 – 3,3V og har AD-konvertere med en oppløsning på 12 bit.

For å regne oss fra en digital avlesning med verdier fra 0 – 1023, til en spenning på 0 – 5V så bruker vi følgende omregning, hvor `digitemp` er den avleste digitale verdien fra sensoren:

```
digitemp * 5.0/1024;
```

For å regne oss om til en temperatur i grader Celcius multipliserer vi spenningen med 100:

```
anatemp = 100 * digitemp * 5.0/1024;
```

`anatemp` er en float (desimaltall), mens `digitemp` er den digitale avleste verdien. Vi multipliserer med 100 siden en vanlig brukt temperatursensor TMP36 da vil gi resultatet direkte i grader C.

Siden den digitale verdien kan være 0 – 1023 skulle man tro at det ville være riktig å bruke 1023 og ikke 1024 i nevneren på brøken i uttrykket over. Imidlertid er det slik at den målte toppverdien for AD-konverter 1023 tilsvarer en spenning på 4,995 V (og ikke 5,0V). dermed vil det være riktigere å skrive 4,995/1023. Dette er imidlertid det samme som 5,0/1024 som er lettere å huske.

2.5.10 Sløyfer – For- og While-loop

Noen ganger har man behov for å gjenta en operasjon flere ganger, da er en *for-loop* egnet. Andre ganger vil man ha behov for å gjenta en operasjon så lenge en betingelse er oppfylt, i så fall kan man benytte en *while-loop.*

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

For()-loop'en egner seg spesielt godt til å gjenta den samme operasjonen et bestemt antall ganger, kanskje med ganske små forandringer mellom gjentagelsene. En for()-loop kan skrives slik:

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

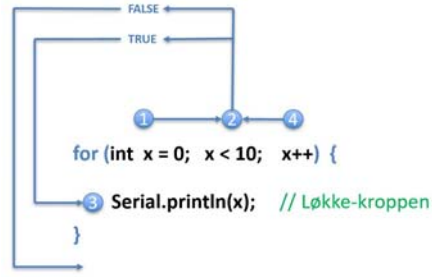
x er en heltallsvariabel (`int x`) som brukes som teller. Denne starter på verdien 0 (`int x = 0;`) økes med 1 (`x++`) for hver runde i loopen og stopper ikke før den når opp til 100 (`x < 100;`). De ulike utrykkene skilles med semikolon. Det som skal gjentas står innenfor klammeparentesene.

Eksempel:

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

Denne skriver ut tallene 0 – 99 til monitoren, ett tall for hver linje. Ønsker man å skrive ut tallene 0 – 100, kan man f.eks. skrive:

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



Figuren til høyre viser rekkefølgen av testen og inkrementering av løkkevariabelen[10].

Legg merke til at linjen, `for(int x = 0; x =< 100; x++)` ikke avsluttes med semikolon, men etterfølges av { }.

While-loop – For å få programmet til å vente i en sløyfe til betingelsen ikke lenger er oppfylt

Det kan for eksempel være tilfelle når man venter på svar fra en sensor. While()-loopen kan skrives slik:

```
while (<logisk betingelse>)
{
```

10.Figuren er hentet fra en av Arne Midjos presentasjoner, kan fås ved henvendelse til arne.midjo@ntnu.no.

```
// Her skrives koden som skal gjentas mens programmet venter
}
```

Legg merke til at linjen, `while (<logisk betingelse>)` ikke avsluttes med semikolon, men etterfølges av { }.

En while()-loop egner seg også for å hindre at en avlest bryter skal medføre et skred av avlesninger av bryteren.

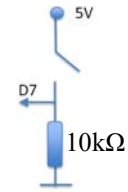
Vanligvis ønsker vi at endring i tilstanden til en bryter skal medføre kun én hending.

I figuren til høyre ser vi en bryter som, når den trykkes inn, forbinder port D7 til +5V (logisk "1"). Når den ikke er trykket inn ligger port D7 til jord via en motstand på 10kΩ. Siden strømmen ut av port D7 til jord er tilnærmet 0, vil spenningen på D7 også være tilnærmet 0V (logisk "0").

While-loop for å unngå mange avlesninger

```
int trykkBryterPin = 7;

void loop()
{
while(digitalRead(trykkBryterPin) == 1)
   {
    trykkBryterVerdi = 1;
   }
If(trykkBryterVerdi == 1)
  {
    // Gjør noe en gang når trykkBryterVerdi er lik 1
    trykkBryterVerdi = 0;
  }
}
```



I programmet ser vi at avlesningen av D7 (`trykkBryterPin`) gjøres inne i argumentet til `while()`-loopen. Så lenge avlesningen er lik "1", så skal programmet bli værende i loopen og gjentatte ganger sette `trykkBryterVerdi` til "1". Det er først når vi *slipper* bryteren at programmet forlater `while()`-loopen og går videre. Den påfølgende `if()`-setningen vil så sjekke om bryteren har vært trykket og utføre den ønskede handlingen. Siden vi ønsker at handlingen med å trykke , må vi huske på å nullstille variabelen `trykkBryterVerdi` før vi forlater `if()`-setningen.

2.5.11 `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 viser 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 bru-

ker da *aritmetiske operatorer* for å undersøke om noe er større enn (>), mindre enn (<) eller lik (==). Legg merke til det doble likhetstegnet. På tilsvarende måte kan en skrive større eller lik (>=) og mindre eller lik (<=).

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.

2.5.12 Switch/Case-kommandoen

Switch/Case kommandoen kan minne om `if()`-setningen, i og med at det er en kommando som styrer retningen til programmet på bakgrunn av verdien til en variabel <var>.

```
switch (var) {
    case 1:
      //Gjør noe når variabelen var er lik 1
      break;
    case 2:
      //Gjør noe når variabelen var er lik 2
      break;
    default:
      // Om ingenting stemmer med variabelens verdi,
      // gjør default. Default er valgfritt
    break;
  }
```

Her ser vi at det er verdien til variabelen <var> som bestemmer hvilket av alternativene (case) som velges. Såfremt verdien til <var> eksisterer i lista over alternativer (case), så utføres det som er knyttet til det aktuelle tilfellet. Så snart handlingen er utført, sørger kommandoen `break;` for at programmet forlater lista over alternativer. Dersom ingen av alternativene finner "match" med verdien til variabelen <var>, så

```
6  void loop() {
7    // read the sensor:
8    int sensorReading = analogRead(A0);
9    // map the sensor range to a range of four options:
10   int range = map(sensorReading, sensorMin, sensorMax, 0, 3);
11
12   // do something different depending on the range value:
13   switch (range) {
14     case 0:    // your hand is on the sensor
15       Serial.println("dark");
16       break;
17     case 1:    // your hand is close to the sensor
18       Serial.println("dim");
19       break;
20     case 2:    // your hand is a few inches from the sensor
21       Serial.println("medium");
22       break;
23     case 3:    // your hand is nowhere near the sensor
24       Serial.println("bright");
25       break;
26   }
27   delay(1);        // delay in between reads for stability
28 }
```

utføres `default`- alternativet.

Figuren over til høyre viser et eksempel på bruk av switch/case-kommandoen.

Her gjøres det en avlesning av en lyssensor i linje 8, verdien legges inn i variabelen `sensorReading`. En slik avlesning fra en analog inngang vil f.eks. ha verdier i området 0 – 1023. Dersom vi skulle ha en case for hver verdi, ville det bli et særdeles langt program. I stedet bruker vi en kommando som reskalerer området fra 0 – 1023 ned til et område på fra 0 – 4. Til denne reskaleringen bruker vi funksjonen `MAP()`:

```
range = map(sensorReading, sensorMin, sensorMAX, 0, 3);
```

Hvor `sensorMIN` og `sensorMAX` er henholdvis minste og største avlesning av lyssensoren. Funksjonen `MAP()` vil fordele verdiene mellom `sensorMIN` og `sensorMAX` på heltallsverdier mellom 0 og 3 som tilordnes variabelen `range`. `range` vil dermed få verdier fra 0 til 3 og vil dermed passe godt som variabel i switch/case- funksjonen som vist i figuren over.

Dersom man sløyfer `break;` under hvert tilfelle, vil programmet hoppe til det aktuelle tilfellet i henhold til variabelen, for deretter å gjennomføre alle etterfølgende tilfeller.

2.5.13 Definisjon av egne funksjoner

I figuren under til høyre ser vi `void setup()`-funksjonen og `void loop()`-funksjonen. Disse er ganske tomme i dette eksempelet, men vil normalt være fylt med kode.

Helt nederst har vi laget vår egen funksjon og kalt den `void funksjon1()`. Denne funksjonen er en bit av programmet som gjør en helt spesiell jobb. Det kan f.eks. være å få den grønne lysdioden i trafikklyset til å blinke et visst antall ganger.

Vi legger også merke til at vi finner funksjonsnavnet igjen under `void loop()`-funksjonen:

```
funksjon1();
```

Når programmet gjennomløper `void loop()`-funksjonen og kommer til `funksjon1()`, så hopper programmet ned til selve funksjonen nederst og utfører kommandoene i funksjonskroppen for så å hoppe tilbake til hovedprogrammet i `void loop()`.

Figuren til høyre viser et eksempel på en egendefinert funksjon. Funksjonen har vi kalt `blinkFemGanger()`. Det er viktig å gi funksjoner meningsbærende navn, som gjør at det er lettere å forstå hva programmet gjør når vi leser koden.

```
void setup()
  {
    // Gjøres en gang ved oppstart
  }

void loop()
  {
    // Gjentas hele tiden
    blinkFemGanger(); // kall funksjonen som lager blink
  }

void blinkFemGanger()
  {
    for (int i = 0; i < 5; i++)
      {
        digitalWrite(LEDpin, HIGH); delay(500);
        digitalWrite(LEDpin, LOW);  delay(500);
      }
  }
```

Definisjonen av funksjonen har vi lagt nederst i programmet. Her er den definert med et navn og et innhold som gjør nettopp det vi forventer – at det grønne lyset blinker fem ganger.

Bruk av argumenter

Hittil har parentesene etter funksjonsnavnet vært tomme. Her kan vi overføre parametere som påvirker hvordan programmet i funksjonen oppfører seg. Dersom vi ønsker at funksjonen skal blinke 6 istedet for 5 ganger, så kan vi lage en ny funksjon som nettopp gjør det, blinker 6 ganger.

En mer elegant måte å gjøre en funksjon mer generell på er å la det være åpent hvor mange blink funksjonen skal utføre. Dette kan vi f.eks. gjøre ved å gi funksjonen et *argument* som overfører det antallet blink vi ønsker at den skal utføre når den kalles.

```
void setup()
  {
    // Gjøres en gang ved oppstart
  }

void loop()
  {
    // Det som skal gjentas hele tiden
    antallBlink = 6;                // Heltallsvariabel
    blinkNGanger(antallBlink); // kall funksjonen som lager blink
  }

void blinkNGanger(int N)
  {
    for (int i = 0; i < N; i++)
      {
        digitalWrite(LEDpin, HIGH); delay(500);
        digitalWrite(LEDpin, LOW);  delay(500);
      }
  }
```

I `void loop()`-funksjonen setter vi inn det ønskede antall blink som argument når vi kaller funksjonen. Her bruker vi variabelen `antallBlink` for å overføre antallet. Vi legger merke til at navnet på variabelen i kallet og variabelen i funksjonsdefinisjonen kan være forskjellige, men det er vanlig at variabelens type er den samme, i dette tilfellet et heltall (`int`).

Funksjoner som returnerer en verdi

Noen ganger vil funksjoner utføre beregninger og vi er interessert i resultatet av beregningene. I det neste eksempelet ønsker vi å lage en funksjon som beregner volumet av en kule.

```
...
void loop()
  {
    // Det som skal gjentas hele tiden
    float volum;
    float radiusKule = 5.0;
    volum = volumKule(radiusKule);    // Kall funksjonen
    Serial.println(volum);
  }

float volumKule(float radius)
  {
    float kulevolum;
    kulevolum = (4/3) * 3.14 * pow(radius, 3);
    return kulevolum;
  }
```

Vi har kalt funksjonen `volumKule` med argumentet `radius` som er av typen `float` (desimaltall). Når vi kaller funksjonen har vi brukt argumentet `radiusKule`. Det betyr at i kallet så overføres verdien `radiusKule` til funksjonen via variabelen `radius`. Denne brukes så i beregningen. Kulevolumet returneres så med kommandoen `retur kulevolum;` som i sin tur legges i variabelen `volum` som så skrives ut til monitoren med kommandoen `Serial.println(volum);`. Legg merke til at funksjonens type må være det samme som typen til variabelen vi vil returnere, i vårt eksempel `float`.

Når vi tidligere har brukt typen `void` på funksjoner som f.eks. `void setup()` og `void loop()`. så betyr det at funksjonen ikke returnerer noen verdi. Void betyr "tom".

2.5.14 Bruk av interrupt

Interrupt brukes når man ønsker å bryte av programmet for å ta seg av en tilfeldig hendelse. Vanligvis er dette hendelser som kommer utenfra og på tilfeldige tidspunkter, som ikke kan forutsies og ikke kan vente, men må behandles umiddelbart. Hendelsene kan inntreffe ved at tilstanden på en port endrer seg, går fra høy til lav eller omvendt.

De ulike mikrokontroller-kortene i Arduino-familien kan håndtere et ulikt antall interrupt og er tilknyttet ulike porter. Tabellen under gir en oversikt over interrupt hos et par Arduino-produkter og ESP8266. Hos både ESP8266 og ESP32 så kan alle portene fungere som en interrupt innganger.

BOARD	INT0	INT1	INT2	INT3	INT4	INT5	...	INT15
UNO	Pin 2	Pin 3						
MEGA	Pin 2	Pin 3	Pin 21	Pin 20	Pin 19	Pin 18		
ESP8266	GPIO 0	GPIO 1	GPIO 2	GPIO 3	GPIO 4	GPIO 5	...	GPIO 15

Som det framgår av tabellen så har Arduino UNO to hårdvare interrupter som er knyttet til port 2 og 3, mens Arduino MEGA har seks interrupts som er knyttet til portene 2, 3, 21, 20, 19 og 18. Vi legger merke til at interruptets nummer (INT0, INT1, INT2, ...) som vi refererer til i programmet, er forskjellig fra portnummeret.

Interrupt service rutinen: Når et interrupt inntreffer vil alle mellomverdier lagres og programmet hopper til interrupt service rutinen (ISR). Denne rutinen er vanligvis svært kort som f.eks. å sette et flagg. Deretter går den tilbake til programmet, henter tilbake alle mellomverdier og fortsetter å kjøre programmet der det slapp. Ved å teste på interrupt-flagget kan man ferdigstille interruptet i ro og mak, med mindre det haster med å fulgt opp.

ISR begrensninger: Siden interrupt blir slått av når programmet hopper til ISR så vil ikke millis() oppdateres og delay()-funksjonen vil ikke fungere mens man er i interrupt service rutinen. Må man bruke delay så kan man bruke delayMicroseconds() som ikke berøres av at interruptet slås av. Det er heller ikke mulig å overføre verdier via argumenter til-, og heller ikke returnere verdier fra ISR.

Prioritet: Når det gjelder hardware interrupt så er det "første mann til mølla" som gjelder, dvs. hardware interrupt har ikke prioritet. Idet et interrupt inntreffer og programmet går til interrupt service rutinen, så blokkeres for andre interrupt, disse blir lagt på vent til det første interruptet er behandlet[11]. Når *det* er gjort håndteres ventende henvendelser.

Programvaremessig består interruptet av følgende kommandoer:

Deklarer porten som skal brukes som interrupt-inngang:

11. http://www.gammon.com.au/interrupts

```
const byte pin = 3;
```

Dernest tilordnes interruptet en navngitt service rutine (ISR) og en modus (mode):

```
attachInterrupt(digitalPinToInterrupt(pin), ISR, mode);
```

F.eks.

```
attachInterrupt(digitalPinToInterrupt(pin), wspeedIRQ, FALLING);
```

Denne legges gjerne i void setup()-funksjonen. `digitalPinToInterrupt(pin)` vil automatisk finne interrupt nummeret (INT0, INT1 osv) dersom man vet hvilken port som er benyttet. ISR er navnet på servise rutinen og mode angir hvordan hendingen som trigger interruptet skal være. Følgende muligheter finnes: På fallende flanke (FALLING), på stigende flanke (RISE), når et skifte inntreffer (CHANGE), når den er lav (LOW) eller når den er høy (HIGH).

Interrupt kan også kobles fra med følgende kommando:

```
detachInterrupt(digitalPinToInterrupt(pin))
```

Dernest lages en Interrupt Service Rutine (ISR). Denne må ha samme navn som angitt i tilordningen (ISR), men navnet bestemmer man selv. Rutinen eller funksjonen legges hvor som helst i programmet bare utenfor void setup() eller void loop(). Følgende er et enkelt eksempel:

```
void wspeedIRQ()
{
  windClicks++; // Øk antall pulser med 1
}
```

I dette eksempelet ser vi at det eneste som skjer i ISR er å øke et antall, praksis telle antall interrupt.

Vi kan også slå av og på interrupt med følgende kommandoer i programmet. Når vi gjør dette så utfører vi handlingen for samtlige interrupt.

```
noInterrupts();
```

Bruker vi noInterrupts(); så vil også dette påvirke enkelte funksjoner i programmet, som f.eks. at millis() vil ikke oppdateres og en del kommunikasjon vil heller ikke bli registrert.

```
interrupts();
```

Denne kommandoen gjennomretter alle interrupt.

Uansett ulemper så er bruken av interrupt et særdeles nyttig hjelpemiddel for å kunne håndtere tilfeldige eksterne kortvarige hendelser.

2.5.15 Installasjon av pakkede biblioteker

Som eksempel bruker vi biblioteket som trengs for å lese data fra trykksensoren BMP180 som også inneholder temperatursensor. Prosedyren er helt ekvivalent for f.eks. BMP280, men biblioteket hentes fra et annet nettsted.

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*

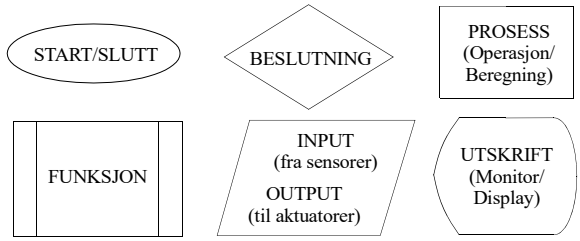
4. **Inkluder headerfilen (*.h) i programmet**
   Det aktuelle biblioteket inkluderes i programmet ved å velge:
   Sketch/Include Library/<aktuelle biblioteket>
   Dermed er header-filen på plass i programmet.

3 Bruk av flytdiagrammer[12]

Under presentasjonen av oppdragene senere i heftet vil vi anvende flytdiagrammer for å tydeliggjøre flyten i programmet. Dersom dere ønsker å gjøre litt ekstra ut av flytdiagrammer så kan dere bruke spesiell programvare for dette formålet.

Et *flytdiagram* eller flytskjema viser strukturen i et dataprogram og er et nyttig verktøy så vel for å planlegge et program, dokumentere programmet og feilsøke i programmet. Flytdiagrammet hjelper oss med å forstå hva som skjer, når og hvordan beslutninger tas og hvilke handlinger som utføres av mikrokontrolleren.

Bruk gjerne litt tid til å tenke gjennom strukturen i programmet, å lage et flytdiagram kan være et nyttig hjelpemiddel for strukturering av programmet. Følgende blokker er ofte nok for å tegne et tilstrekkelig detaljert flytdiagram. Som vi ser har blokkene ulik form avhengig av funksjon:

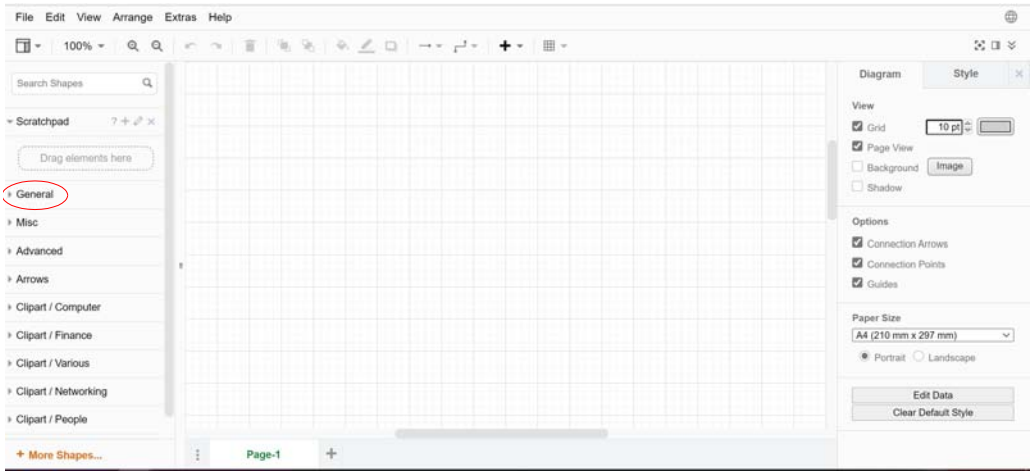


De ulike blokkene forbindes gjerne med linjer og piler for å indikere hvilken vei programmet beveger seg eller *flyter*. Ett slikt flytdiagram er ikke bare et nyttig hjelpemiddel under programmeringen, men også når man skal dokumentere programmet.

Vi skal bruke et gratisprogram som kan være til hjelp for å tegne gode flytdiagrammer:

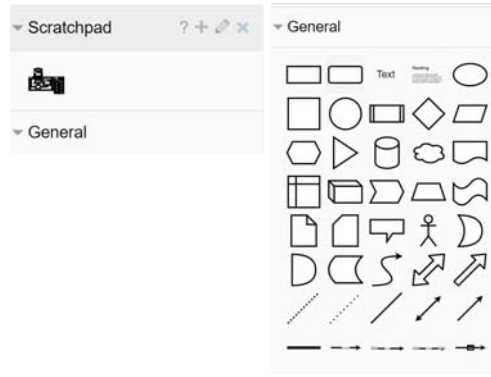
https://app.diagrams.net/

Når man trer inn i programmet blir man møtt med følgende grensesnitt:



12. Dette avsnittet bygger på det arbeidet Freddy Roger Moen utførte januar 2021 ved Levanger videregående skole – 2.0 Flytskjema.pdf

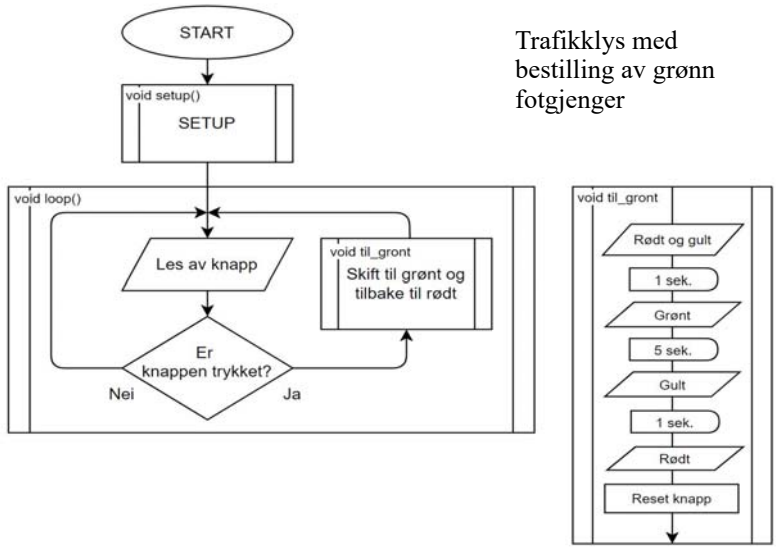
I menyen til venstre finner man flere samlinger av symboler, både for tegning av flytdiagrammer og annet. Velger man *General* så får man opp de vanligste symbolene brukt for tegning av flytdiagram som vist i figuren til høyre.



I menyfanen *Scratchpad* så kan man legge hele eller deler av flytdiagrammer slik at de kan brukes senere. Som det framgår av figur (A) til høyre så inneholder *Scratchpad* et lagret flytdiagram.

La oss se nærmere på et eksempel.

La oss ta utgangspunkt i flytdiagrammet under.



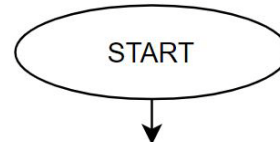
Trafikklys med bestilling av grønn fotgjenger

Vi har en fotgjengerovergang med trafikklys som inkluderer en knapp for bestilling av "grønn fotgjenger". Når ingen har trykket på knappen så er det grønt for bilene og rødt for fotgjengerne. Når knappen trykkes skifter det til grønt i 5 sek. for så å skifte tilbake til rødt igjen. Legg merke til at vi må resette *flagget* (Reset knapp) som leser av knappen, hvis vi har valgt å bruke et slikt flagg.

La oss se på de ulike symbolene:

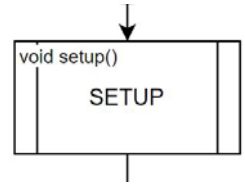
START og STOPP – Ellipsen

Ellipsen eller et rektangel med sirkelbue i begge endrer brukes for indikere at her starter eller slutter programmet. Siden programmer i Arduino gjerne går til strømmen tas, så har vi bare en *START*-ellipse. For programmer som har en slutt, bruker man en tilsvarende ellipse med teksten *STOPP*.



FUNKSJONER – Rektangel med sidekanter

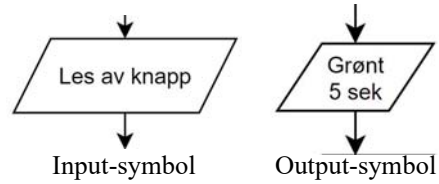
Noen ganger er det nødvendig at et program har en eller flere tilleggsfunksjoner. Da benyttes rektangler med dobbel kant på sidene. Disse funksjonene eller prosessene vil ofte ha egne flytdiagram. Slike funksjoner kalles noen ganger for *sub-rutiner* eller *interrupt rutiner*. I forbindelse med Arduino-programmering kalles de *funksjoner* og avgrenses av to klammeparenteser.



Vi vet at noe av det første mange Arduino-programmer gjør er å starte opp *void setup()*-funksjonen som utføres bare første gang programmet kjøres. Vi har valgt å sette navnet til funksjonen oppe i venstre hjørnet. *Void setup()*-funksjonen inneholder gjerne initiering av andre funksjoner som skal brukes i programmet, bestemme hvilke porter som skal være inn- og utganger og andre ting som skal være uforandret gjennom hele programmet.

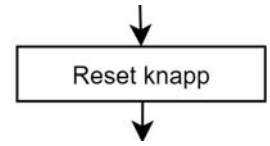
INPUT og OUTPUT – Parallellogrammet

Mikrokontrollere kommuniserer ofte med omverdenen. I flytdiagrammet er dette angitt med et parallellogram. Det er symbolet for I/O (Input/ Output) kommunikasjon. I eksempelet vårt leser mikrokontrolleren av fotgjengerknappen som er en input-kommunikasjon, eller også slår på en eller flere lysdioder, som er output-kommunikasjon.



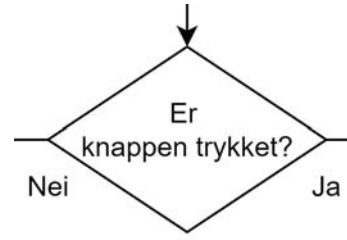
INTERN PROSESSERING – Rektangel

Rektangler beskriver interne operasjoner som ikke krever kommunikasjon med omverdenen. Dette kan f.eks. være matematiske beregninger eller tilbakestilling (resett) av flagg, som i vårt tilfelle, som må utføres før den går videre i programmet.



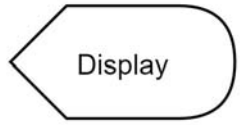
VEIVALG – Rombe

Å velge ulike veier i programmet på grunnlag av ulike *hendinger* (betingelser) er noe som gjør dataprogrammer til kraftige verktøy. Hvilke hendinger som kan inntreffe og hvilke betingelser som gjelder, må være kjent på forhånd. I eksempelet vårt er hendingen at fotgjengerknappen kan ha blitt trykket, og betingelsen er om den er trykket eller ikke. Dersom den ***ikke er trykket*** går den tilbake og tester om knappen er trykket. Dersom den ***er trykket*** så skifter trafikklyset til grønt i 5 sek. for deretter å gå tilbake til rødt igjen. Det er mulig å ha flere enn to hendinger eller betingelser.



UTSKRIFT/DISPLY – Avrundet rektangel med spiss

Ikke sjelden har vi bruk for å skrive ut resultatene av en datainnsamling, en måling eller beregning. Dette kan skje på ulike måter som for eksempel til et display. Da kan vi bruke symbolet vist på figuren til høyre. I programverktøyet vårt finner vi dette symbolet under menyfanen *Advanced*.



FORSINKELSE (DELAY) – Avrundet rektangel

I enkle programmer er det ikke uvanlig å legge inn forsinkelser, eller delay, i programmene. Dette kan være tilfelle i vårt trafikklys-eksempel hvor vi ønsker å beholde lysene på en viss tilstand.

4 Fritzing

4.1 Bruksområde

Fritzing er et tegneprogram for tegning av oppkoblinger på koblingsbrett. Programmet har et fyldig bibliotek med komponenter og kretskort fra bl.a. Arduino, Micro:bit, PIC og mange flere. Programmet egner seg derfor glimrende for dokumentasjon og til å lage oppgaver som krever anvisning for oppkoblinger. Vi har brukt Fritzing til alle oppkoblinger i dette heftet.

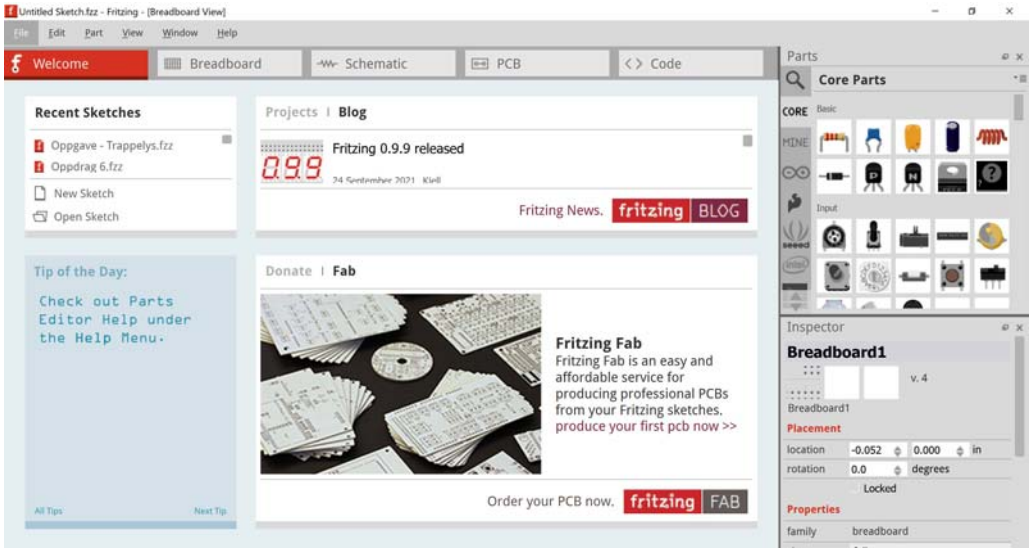
Når man har montert komponentene på koblingsbrettet, kan man konvertere tegningen til et koblingsskjema som viser komponentsymbolene, for deretter å legge ut som et kretskort. De seneste versjonene gir også mulighet for å skrive og utvikle programmer.

4.2 Installasjon

Programmet finnes i skrivende stund i versjon 0.9.9 (24. september 2021) som fritt kan lastes ned fra nettstedet: www.fritzing.org

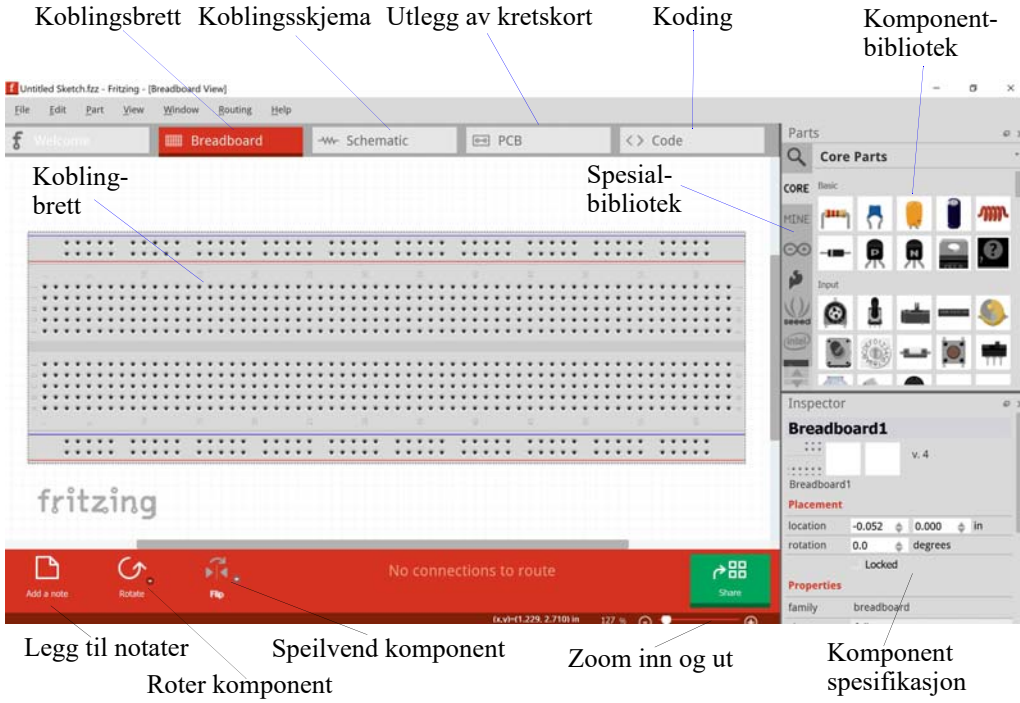
4.3 Introduksjon til bruk

Når man starter Fritzing kommer følgende introduksjonsvindu opp:



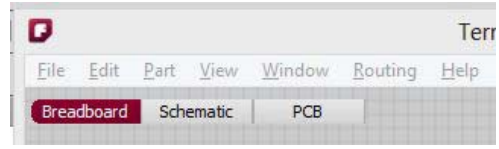
Deretter er det naturlig å velge *BreadBoard* som gir mulighet til å bygge opp en krets på et koblingsbrett som er et naturlig startpunkt for uttesting av kretser som ikke er spesielt kritiske mht. layout. Bruk av koblingsbrett er ikke nødvendigvis en god løsning for uttesting av kretser i RF-området (RF – Radio Frequency).

Velger man BreadBoard-fanen får man opp følgende skjermbilde:



Et standard koblingsbrett ligger klart til bruk i arbeidsfeltet til venstre. Til høyre finner vi *komponentbiblioteket*. Flere *spesialbiblioteker* kan hentes inn ved å trykke på ikonene til venstre for komponentbiblioteket. Ved hjelp av glidebryteren nederst kan en zoome inn og ut i arbeidsfeltet (hjulet på musa kan brukes på samme måte).

På menylinjen oppe til venstre finnes fire faner:

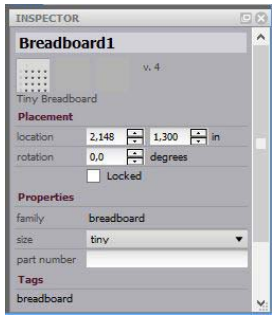


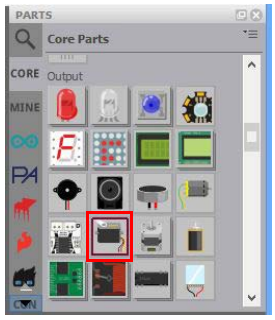
- ***Breadboard*** – Denne fanen gir deg mulighet til å konstruere kretser på koblingsbrett. Dette er et særdeles nyttig hjelpemiddel når du skal dokumentere dine konstruksjoner.
- ***Schematic*** – Ved å velge denne fanen overføres konstruksjon din til kretsskjema, Riktignok er programmet ganske primitivt slik at du selv må plassere og dreie komponentene slik at skjemaet ser greit ut.
- ***PCB*** – Ved å velge denne fanene overføres kretsskjemaet til "layout"-delen av programmet. Her kan du få lagt skjemaet ut som et kretskort og automatisk "rutet" kortet med kobberbaner. Enten på ensidig eller tosidige kretskort. Så kan en generere ulike filer som kan sendes inn for produksjon av kretskortet. Også her må en "hjelpe programmet" med å finne passende komponentplassering.

- ***Code*** – Her kan man skrive og teste ut kode for Arduino og PIC-prosessorer. Tanken er at man både kan skrive og laste ned programmene til mikrokontrollerkortet. Denne funksjonen er foreløpig bare tilgjengelig for PIC-prosessoren, for Arduino må man skrive og teste ut koden, men ikke laste ned fysisk til kretskortet (Rev. 0.9.9).

4.4 Oppbygging på koblingsbrett (Breadboard)[13]

Vi skal her beskrive oppbygging av en liten krets på koblingsbrettet trinn for trinn. Som eksempel velger vi å bygge opp roboten Abot som styres ved hjelp av en Arduino UNO. Sammenkoblingen av servoer og avstandssensoren bygges opp på et lite (tiny) koblingsbrett.



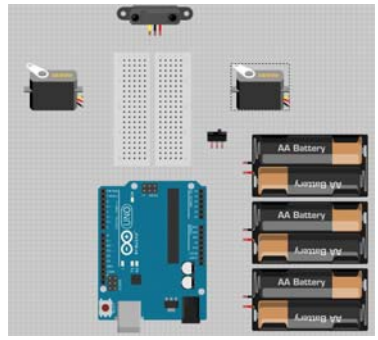


1. **Velg størrelse på koblingsbrettet**
   Klikk på koblingsbrettet som ligger i arbeidsfeltet slik at koblingsbrett menyen kommer opp i komponentfeltet til høyre. Velg *tiny* for størrelse (*size*). Grip brettet med cursoren og legge det mitt på arbeidsfeltet.

2. **Drei koblingsbrettet**
   Komponenter kan dreies 90° ved å høyreklikke på komponenten for så å velg *Rotate* og ønsket dreining og retning. Koblingsbrettet kan bare roteres et helt antall 90°. Man kan også bruke "Dreie" ikonet nede til venstre.

3. **Hent inn komponentene**
   Til vårt formål trengs to 360° servoer. En slik finnes i komponentmenyen, øverst til høyre, under kategorien *output*. Servoer har gjerne tre ledninger (en *gul*, en *rød* og en *sort*). Rød er +5V, sort er GND (0V), og gul er styreinngangen som bestemmer dreiehastigheten. Videre hentes Arduinoen inn og legges i arbeidsfeltet. En avstandssensor (IR proximity sensor) og en bryter (toggle switch) hentes også inn i arbeidsfeltet slik at resultatet blir omtrent som på figuren under til høyre.

4. **Batteri**
   Vi har her valgt å benytte et 9 V batteri. Siden Arduinoen og komponentene normalt benytter 5V, så har vi valgt å benytte 5V regulatoren på Arduino-kortet til å senke spenningen fra 9V til 5V.



5. **Fest komponentene**
   Dersom en plages med ufrivillig flytting av komponenter. Høyreklikker man på komponenten og velger *Lock Part* i nedtrekksmenyen.

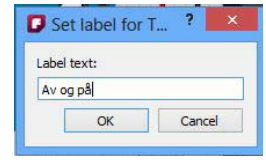
13. Deler av denne fremstillingen er hentet fra en eldre utgave av Fritzing.

6. **Trekk forbindelseslinjer**
   Dernest trekkes forbindelser mellom terminalene til komponentene på følgende måte: Finn underkategorien *Breadboard view* i komponentmenyen og velg ikonet  . Forbind så to terminaler med hverandre ved å klikke på den ene og dra musa til den andre, mens venstreknappen holdes nede. Endepunktene skal feste seg til terminalene.

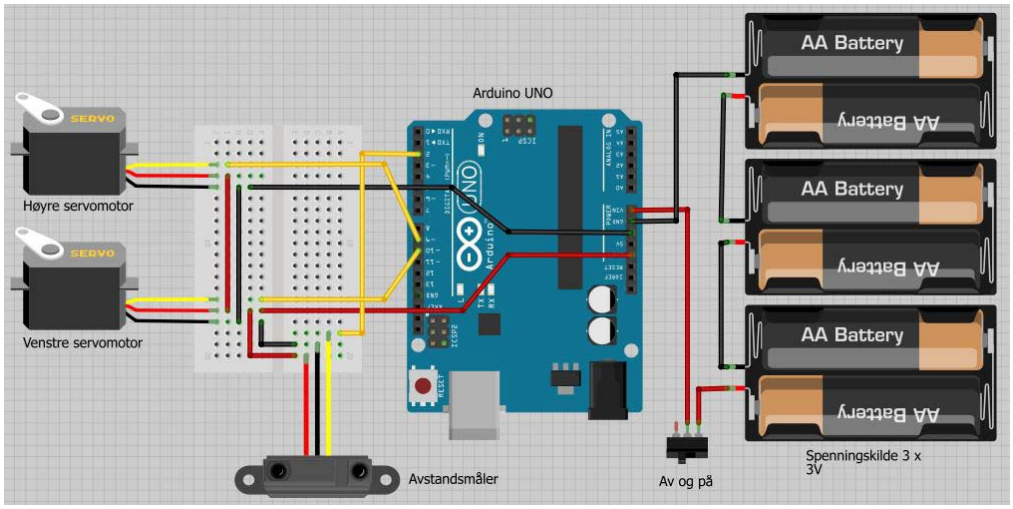
7. **Lag og fjern knekkpunkter på forbindelse**
   Dersom en ønsker en knekk på ledningen, klikker man på ledningen omtrent der en ønsker en knekk. Dra så i knekkpunktet til ønsket posisjon. Et knekkpunkt kan fjernes ved å høyreklikke på punktet, for så å velge *Remove bendpoint* i nedtrekksmenyen.

8. **Endre farge på forbindelseslinjene**
   Det kan være lurt å merke forbindelse med ulike farger. F.eks. rødt for + og sort for –. Dernest kan en velge fritt slik en finner det hensiktsmessig. Farge velges ved å høyreklikke på forbindelsen og så velge *Wire color* i nedtrekksmenyen.

9. **Legge til merkelapper til komponenter**
   Dersom man ønsker å sette en merkelapp på en eller flere av komponentene, høyreklikker man på komponentene og velger *Show part label*, dermed kommer det opp en default merkelapp. Ved å dobbelklikke på denne, kommer det opp en innboks (se figur til høyre) hvor man kan erstatte innholdet med det man måtte ønske.

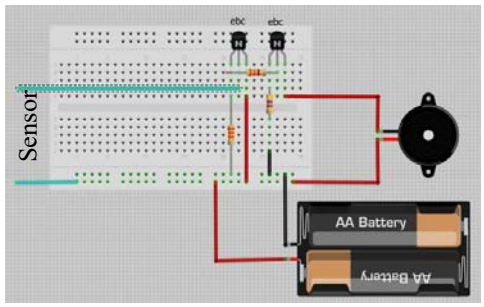


10. **Ferdig oppkobling**
    Figuren under viser den ferdige oppkoblingen som enten kan brukes for dokumentasjon, eller hjelp for elevene til å koble opp etter oppskrift om det er ønskelig.

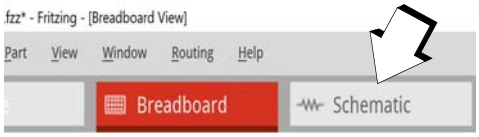


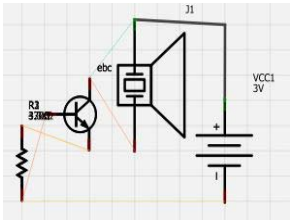
4.4.1 Overføring til koblingsskjema

Etter å koblet opp kretsen på koblingsbrettet og testet at den fungerer som ønsket, kan kretsen overføres til et standard koblingsskjema. Som et eksempel har vi valgt kretsen vist på figuren til høyre.



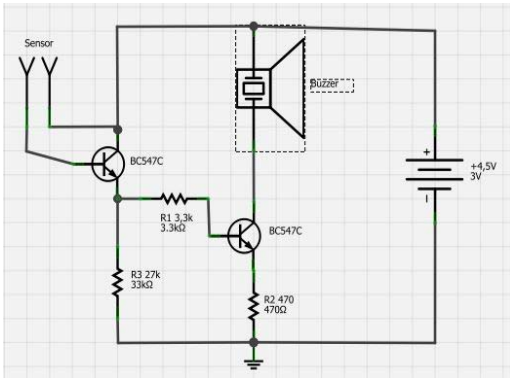
1. **Konvertering til koblingsskjema**
   Ved hjelp av menyen øverst i venstre hjørne kan oppkoblingen på koblingsbrettet konverteres til et koblingsskjema med symboler ved å velge *Schematic*.





2. **Justering av skjema layout**
   Den automatiske konverteringen gir ingen god layout som vist nederst på figuren til venstre. *De to transistorene og de tre motstandene er plassert oppå hverandre.* Derfor er antallet komponenter for lav. Skjemaet må derfor justeres for hånd. Dette gjøres ved å flytte komponentene slik de normalt vil være plassert i et koblingsskjema, med signalflyten fra venstre mot høyre. Flyttingen skjer ved å "gripe" komponenten ved å venstreklikke på komponenten, for så å dra den dit den skal.

3. **Trekke opp forbindelsene på nytt**
   Etter at komponentene er plassert der de skal være, trekkes forbindelsene opp på nytt. Komponentene er forbundet med tynne streker som indikerer sammenkoblingene. Om noen forbindelser mangler, skyldes det at det er brudd i forbindelsen på koblingsbrettet. Om dette er tilfelle går en tilbake og retter opp forbindelsen. I det en trekker opp forbindelsene på nytt, blir linjene markert med en tykkere strek. Figuren til høyre viser resultatet.

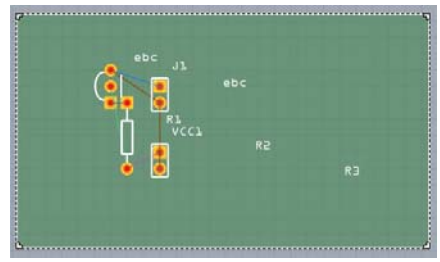


4. **Tekst**
   Teksten kan endres ved å dobbelklikke på tekstrubrikken. Skriv ny tekst inn i innboksen og trykk OK. Tekstfeltet kan så flyttes til ønsket posisjon.

4.5 Trykte kretskort

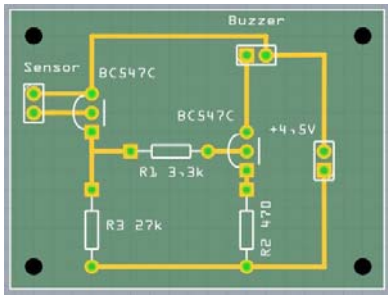
4.5.1 Fra skjema til PCB (Printed Circuit Board)

Når skjemaet er klart, kan dette rutes til et kretskort. Ved å trykke på PCB på menyen (øverst på figuren til høyre) konverteres skjemaet til en layout. Denne er svært uferdig og forutsetter at en manuelt plasserer komponentene slik at PCB-layout-en blir tilfredsstillende (nederst på figuren til høyre).



Vi legger merke til at begge transistorene legges på samme sted, tilsvarende med motstandene.

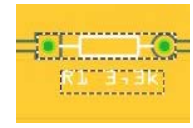
Layout



I utgangspunktet er forbindelseslinjene tynne rette linjer som knytter sammen beina til komponentene. Når komponentene er plassert kan en begynne å trekke de endelige ledningsforbindelsene. Idet man venstreklikker på en av linjeforbindelsene vil denne konverteres til en kobberbane med tilhørende loddeland. Knekkpunkter langs kobberbanen legges inn ved å klikke på banen. Ved å høyreklikke på et knekkpunkt kan en fra nedtrekksmenyen velge å fjerne punktet.

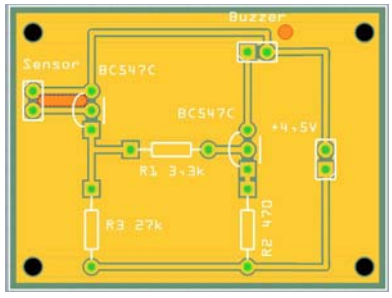
På et tosidig kort ligger de gule linjer på loddesiden av kretskortet, og de oransje linjene ligger på komponentsiden. Dersom en ønsker at alle linjene skal ligge på loddesiden høyreklikker man på kobberbanen og velger *Move to top layer*.

Tekst:



Ved å venstreklikke på komponenten markeres både komponent og tilhørende tekst. En må nå skrive inn ny tekst i tekstrammen samtidig som man kan gripe ramma og flytte den dit man ønsker.

Jordplan:



Det er ikke uvanlig at alle mellomrom mellom kobberbanene fylles med en kobberflate. Ved å forbinde denne til jord, vil man skape et robust jordplan som reduserer spredning av elektrisk støy. Et slikt jordplan legges inn i mellomrommene ved å dra ikonet  fra komponentkategorien *PCB view*.

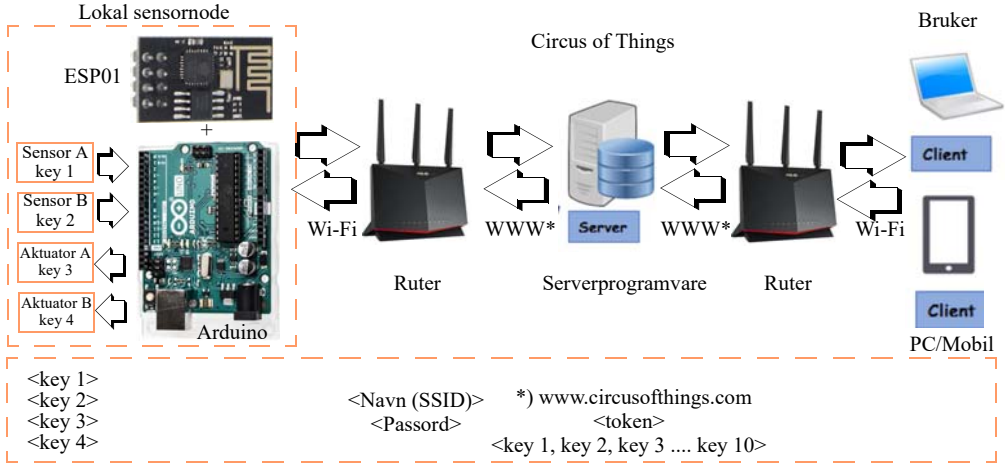
5 Oppkobling via ESP01s Wi-Fi-enhet

I dette kapittelet skal vi se hvordan vi kan bruke Wi-Fi-enheten til ESP01 til å koble Arduino til Internett og videre til Circus of Things (CoT).

5.1 Circus of Things – Arduino + ESP01 brukt som sensornode[14]

I dette avsnittet skal vi se på det som gjør ESP01 spesiell, nemlig Wi-Fi enheten. Kommunikasjonen med nettet er i vårt tilfelle så tett koblet til Circus of Things at dette konseptet vil bli en vesentlig del av denne beskrivelsen.

Circus of Things er en gratis tjeneste som er etablert og drives av ***Jaume Miralles***, som holder til på Mallorca, Spania. Tjenesten gjør det mulig å bygge opp software *konsoller* som viser innsamlede data fra sensorer og konsoller for fjernstyring av aktuatorer. Flere konsoller kan samles i *paneler*. Tjenesten tilbys både for PC/MAC og smarttelefon. Dessuten kan signalene offentliggjøres slik at andre kan følge målingene. Eneste begrensning er inntil videre at maksimalt antall signaler til hver bruker er satt til 10. Dette kan ev. økes ved å henvende seg til Jaume. Tjenesten er heller ikke særlig rask. Den egner seg derfor ikke til "real time" styring.



Figuren over viser hvordan sensorer og aktuatorer hos den lokale sensornoden, overfører data via en sentral server og videre til brukeren (klienten). Hos serveren gis signalene knyttet til sensorene og aktuatorene hver sine *nøkkelkoder* ("key1 – N") og hvert sitt *konsoll* (bryter, glidebryter, display). Konsollene organiseres i paneler som kan inneholde flere konsoller som ønskes relatert til hverandre. Brukeren får når vedkommende blir registrert, en *brukeridentitet* ("token") som også knyttes til signalene som overføres.

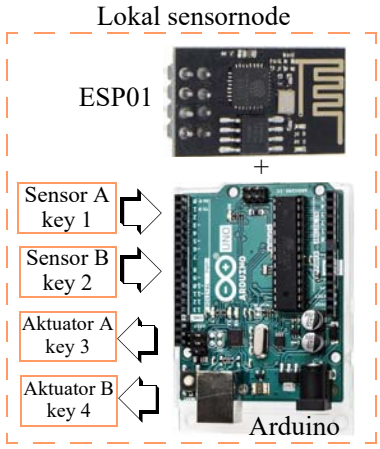
Prosedyre for å etablere konsoller og paneler:

Vi skal nå gi en oversikt over hvordan vi kan lage paneler og konsoller på nettstedet Circus og Things (CoT): www.circusofthings.com. Senere skal vi se i detalj hvordan vi gjør dette.

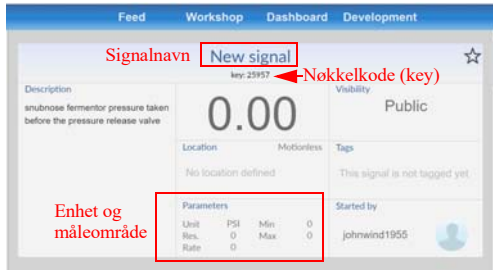
14. Dette kapittelet bygger på en presentasjon utarbeidet av Kåre-Benjamin H. Rørvik

- **Opprett bruker** med passord hos www.circusofthings.com.
  Man får da en identifikasjonskode ("token"), en lang bokstav- og tallkode, som identifiserer brukeren. For detaljer om oppretting av konto og signatur se avsnitt 7.2.6 side 126.

- **Oppkobling av sensornode**: Koble opp ESP01 + Arduino med sensorer og aktuatorer.

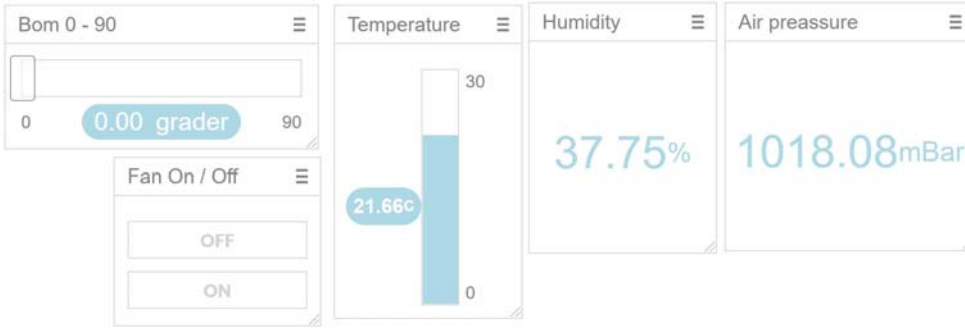


- **Gi navn og nøkkelkode til signaler:** I *"Workshop"* hos CoT spesifiseres signalnavn, enheter og måleområde som harmonerer med sensorene og aktuatorene som er koblet opp til den lokale ESP01.

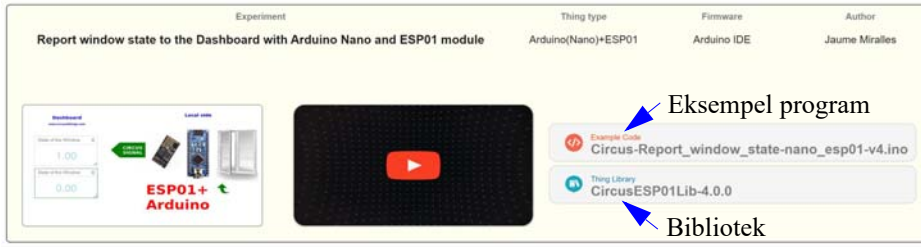


Samtidig får man tildelt et nøkkelkode ("key") knyttet til signalet. Se figur over.
Dette er omtalt i detalj i avsnitt 7.2.7 side 128.

- **Bygg opp panelet:** Panelet er sammenstillingen av brytere og displayer for å styre aktuatorene og vise sensordata fra sensornoden. Dette gjøres i fanen "Dashboard" i Circus of Things. Bildet under viser monitorering av tre sensorverdier for temperatur, luftfuktighet og lufttrykk. Til venstre er to konsoller for styring av et rele (ON/OFF) og en servo (Glidebryter 0 – 90°).
  Panelet:



- **Programmer til Arduino + ESP01:** Så må man skrive programmet som leser av sensorene og sender data over til serveren slik at de kan vises på panelet. Likeså må programmet kunne motta styredata fra serveren slik at den kan styre servoer, lys og releer, o.l. fra konsollene.
  - ***Hent biblioteket:*** Først henter man biblioteket som trengs for å kommunisere med Circus of Things (CoT). Dette finner vi under oversikt over biblioteker på nettsiden til CoT[15].



    Disse er gjerne knyttet til ulik type hårdvare som brukes. Vi bruker Arduino + ESP01 og henter inn følgende:

```
#include <CircusESP01Lib.h>
```

    Biblioteket legges inn i starten av programfila. For detaljer se avsnitt 7.2.8 side 132.

  - ***Så defineres variablene*** som holder informasjonen som trengs for å opprette kommunikasjonen med nettstedet via den lokale routeren:

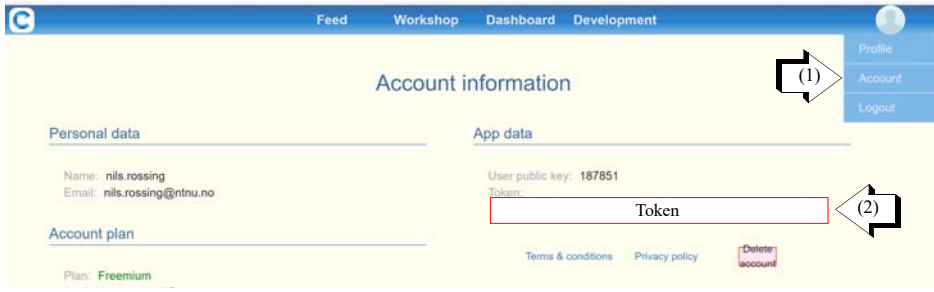
```
char ssid[] = "<Ruterens navn (SSID) legges inn her>";
char password[] =  "<Ruterens passord legges inn her>";
char server[] = "www.circusofthings.com"; // Nettsiden hvor CoT-serveren er
char token[] = "<token settes inn her>";
```

15. https://circusofthings.com/dev.jsp under Libraries under Libraris: Arduino + ESP01 – CircusESP01Lib-1.1.0

Her trenger vi blant annet *navn* og *passord* til den lokale routeren vi ønsker å kommunisere med. Denne informasjonen legges inn i de respektive variablene.

Tilsvarende må vi oppgi nettadressen til Circus of Things: www.circusofthings.com

Tilslutt må vi finne var identifikator, *"token"*, knyttet til vår personlige kontor ved Circus of Things, som er en ganske lang kode som vi finner under *Account* (1) og *Token* (2):



- **Deklarasjon av nøkkelkoder:** Nøkkelkodene er knyttet til hvert enkelt konsoll som framkommer hver gang vi oppretter et signal i Workshop hos CoT (se punktet over). Disse legger inn i variabler med et meningsfylt navn:

```
char order_key_relay[] = "27094"; // Nøkkel for å kommunisere med releet
char order_key_gate[] = "19499";  // Nøkkel for å kommunisere med bommen
char temperature_key[] = "21251"; // Nøkkel for å kom. med temperatursensoren
char humidity_key[] = "20852"; // Nøkkel for å kom. med luftfukt.sensoren
char pressure_key[] = "13531"; // Nøkkel for å kom. med lufttrykksensoren
```

  I dette eksempelet har vi definert 5 signaler som har fått hver sin nøkkelkode. Nøkkelkodene gjengitt her vil være individuelle og må skiftes ut med det som framkommer når hver enkelte bruker etablerer signalene.

- ***Opprett "software Serial":*** Normalt overfører vi data til Wi-Fi-enheten (ESP01) via den innebygde UART Rx/Tx som er koblet til D0/D1. Denne brukes også for å kommunisere fra Arduino og tilbake til monitoren i PC'en. For å beholde muligheten til å kommunisere med monitoren så velger vi å opprette en software UART via portene D6 (Tx) og D7 (Rx):

```
int TXPinToESP01 = 6; // IO port som brukes som TX for seriekommunikasjon med ESP01
int RXPinFromESP01 = 7; // IO port som brukes som RX for seriekommunikasjon med ESP01
SoftwareSerial ss(RXPinFromESP01,TXPinToESP01);
```

- ***Deklarering av flere variabler:*** I tillegg ønsker vi å definere noen konstanter som vi vil bruke når vi deklarerer objektet som vi ønsker å bruke når vi setter opp kommunikasjonen via ESP01:

```
int esp01BaudRate = 9600; // Dataraten for kommunikasjon med ESP01.
int debugLevel = DEBUG_YES; // Velg debug-nivå. DEBUG_NO, DEBUG_YES or DEBUG_DEEP
int enableSSL = 0; // Bruk SSL-kryptering eller ikke under kommunikasjonen med CoT.
                   0 -> Disable. 1 -> Enable.
```

- ***Opprettelsen av et objekt:*** Dernest opprettes et objekt som vi velger å kalle `circusESP01()` av typen `CircusESP01Lib`

```
CircusESP01Lib circusESP01(&ss,esp01BaudRate,server,token,ssid,password,
debugLevel,enableSSL);
```

Legg merke til at her inngår server-adressen, ruternavnet (SSID) og passordet til ruteren med mer.

- ***Initialisering:*** I void setup()-funksjonen initialiseres funksjonene som står for kommunikasjonen med serveren ved CoT:

```
circusESP01.begin();        // Initialiser Circus of Things
```

- ***Les data mottatt fra panelet hos CoT:*** Så leser vi data som sendes over fra panelet og de ulike konsollene:

```
double dashboard_order_gate = circusESP01.read(order_key_gate,token);
```

Det er styredata for å dreie servoen et visst antall grader som er vist i eksempelet over. Verdien som overføres fra bryterkonsollet med nøkkelkoder: `order_key_gate` legges i variabelen: `dashboard_order_gate`. Vi legger også merke til at sammen med nøkkelkoden til konsollet, så sendes også brukeridentiteten ("token").

Tilsvarende gjentas for alle signaler som mottas av sensornoden.

- ***Skriv data til panelet hos CoT:*** Dernest sendes data over til panelet til de ulike konsollene (displayene) på følgende måte:

```
circusESP01.write(temperature_key,temperature,token);
circusESP01.write(humidity_key,humidity,token);
circusESP01.write(pressure_key,pressure,token);
```

I eksempelet over oversendes måledata fra temperatur-, luftfuktighet- og lufttrykksensorene hos BME280 til panelet ved CoT. Sensordataene er lest inn i variablene `temperature`, `humidity` og pressure. Disse knyttes til nøkkelnumrene til de tre displayene `temperature_key, humidity_key` og `pressure_key`. Vi legger også merke til at brukeridentiteten ("token") knyttes til hver av signalene.

I tillegg må signalene behandles på vanlig måte, både ved avlesning av sensorene og pådrag til aktuatorene.

6 Oppgaver – Sensordel

Før vi beskriver de ulike delene av værstasjonen så kan det være greit å lage en oversikt over hvilke porter vi skal bruke til hver av sensorene.

Tabell 1: Oversikt over bruk av porter hos Arduino UNO

Deloppdrag	Type	Sensor/Aktuator	Avsnitt	Term. 1 SDA	Term. 2 SCL	Adr.
1	BMP280	Lufttrykk og temperatur	6.1 side 69	A4	A5	0x76
2	SSD1306	OLED display	6.2 side 73	A4	A5	0x3C
3	TC74	Temperatur	6.3.1 side 76	A4	A5	0x48
4	DHT11	Luftfuktighet og temperatur	6.4.1 side 79	D8	GND	-
5	LDR	Styring av rele eller LED	6.5.2 side 84	A1 og D9	GND	-
6	BH1750	Lysintensitet	6.5.3 side 86	A4	A5	?
7	SEN-15901	Anemometer (vindhastighet)	6.6 side 91	D2	GND	-
8	SEN-15901	Vindretning	6.7 side 95	A0	GND	-
9	SEN-15901	Nedbør (regn)	6.8 side 101	D3	GND	-
10	HC-SR04	Avstand (snødybde)	6.9 side 104	D4 (ECHO)	D5 (TRIG)	-
11	TC74/SEN-15901	Vindavkjølingsindeks	6.10 side 109	A4 og D2	A5 og GND	0x48
12	DHT11	Varmeindeks	6.11 side 116	D8	GND	-
Oppkobling til Wi-Fi og Circus of Things (CoT)						
13	DHT11	Luftfuktighet og temperatur	7.1 side 119	D8	GND	-
14	ESP01	Wi-Fi-oppkobling	7.2.4 side 125	D6 (Rxd)	D7 (Txd)	-
15	-	Opprett bruker, konsoll og panel	7.2.6 side 126	-	-	-
16	Arduino + ESP01	Skriv kom. program m/CoT	7.2.8 side 132	D6 (Rxd)	D7 (Txd)	-
17	Rele (JQC-3FF-S-Z)	Fjernstyring av rele	7.3 side 135	D9	GND	
18	Rele (JQC-3FF-S-Z)	Automatisk fjernstyring av rele	7.4 side 140	D9	GND	
19	Motor Input 1	Styring av retning 1	7.5 side 143	D11	GND	
	Motor Input 2	Styring av retning 2		D12	GND	
20	Motor En1	Enable og styring av hastighet	7.6 side 146	D10	GND	
21	Motor En1	Styring av farten fra CoT	7.7 side 148	D10	GND	
22	Fukt og motor	Sammenstill luftfuktighet og vifte	7.8 side 151	D8/10	GND	
23	Legg til en sensor	Værstasjon - Legg til en sensor	7.9 side 156	-	-	
24	-	Presentasjon av målinger over tid	7.10 side 157	-	-	
25	-	Inkluder målinger fra andre gr.	7.11 side 160	-	-	

6.1 Måling av lufttrykk med BMP280

Læringsmål

Oppdraget trener følgende læringsmål:

- Bruk av I^{2}C-buss
- Henting og installasjon av biblioteker
- Bruk av feilmelding ved henting av måledata
- if-else-setning og do-while-løkke

Kort omtale

BMP280 er en nyere versjon av BMP180 som er sensorer for måling av atmosfærisk trykk. Selve sensoren anvender en piezo-resistiv teknologi, dvs. at med økende lufttrykk vil motstandsverdien i det piezo-resistive materialet endres. Sensoren inneholder dessuten en temperatursensor som leses av via I^{2}C-bussen sammen med lufttrykket. Trykkdata leveres med en oppløsning på 16 – 19 bit, mens temperatur leveres med 16 bit.

Gjennomsnittlig måletid er 5,5 msek, så en kan foreta raske avlesninger med en punktprøvingsrate på 157 sps (samples pr. sec.). Temperaturmålingen kan nøye seg med langt lavere punktprøvingsrate (1 Hz). Økt samlingstakt vil dessuten øke strømforbruket.

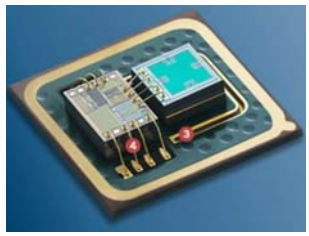
BMP280 leverer absoluttverdier av trykkmålinger med en nøyaktighet på ±1hPa og en relativt nøyaktighet på ±0,12 hPa (±1m) i området fra 300 hPa – 1100 hPa (fra –500 til 9000 moh.). Oppløsningen er imidlertid 0,16 Pa. Den leverer også temperaturmålinger med en absolutt nøyaktighet på ±0,5°C @ 25°C, ±1,0°C fra 0 – 65°C, men med en oppløsning 0,01°C. *Her kan man lett bli lurt ved at man tror at oppløsningen tilsvarer nøyaktigheten.*

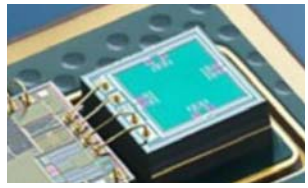
Parameter	BMP180	BMP280
Footprint	3.6 × 3.8 mm	2.0 × 2.5 mm
Minimum V_{DD}	1.80 V	1.71 V
Minimum V_{DDIO}	1.62 V	1.20 V
Current consumption @3 Pa RMS noise	12 µA	2.7 µA
RMS Noise	3 Pa	1.3 Pa
Pressure resolution	1 Pa	0.16 Pa
Temperature resolution	0.1°C	0.01°C
Interfaces	I²C	I²C & SPI (3 and 4 wire, mode '00' and '11')
Measurement modes	Only P or T, forced	P&T, forced or periodic
Measurement rate	up to 120 Hz	up to 157 Hz
Filter options	None	Five bandwidths

Sensorene kan operere på batterispenninger fra 1,7 V til 3,6 V. Vanlig spenning er 3,3 V. Kretsene kan kobles direkte til mikrokontrolleren via en I^{2}C buss.

Tabellen over sammenligner data for BMP180 og BMP280.

Teknologi





Teknologien bygger som nevnt på piezo-resistiv effekt som gjør at resistiviteten i en tynn skive (figur nederst til høyre) av piezo-resistivt materiale endrer resistivitet når den utsettes for trykk og dermed nedbøyning. Denne endringen i resistivitet vil så omdannes til en spenning i en målebro slik at de små endringene i resistivitet medfører endringer i en tilsvarende elektrisk spenning.

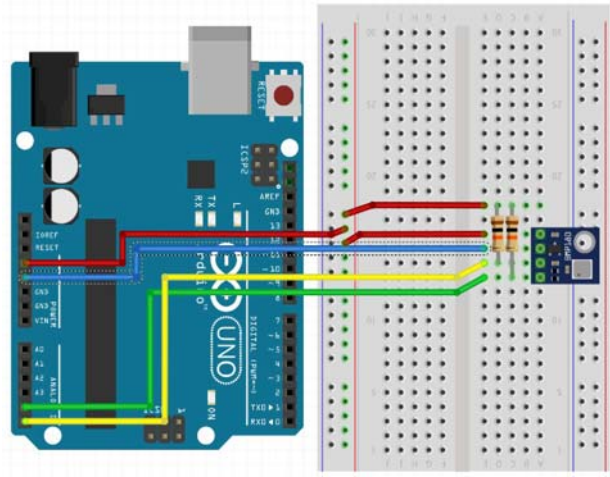
Ved måling av absolutt lufttrykk legges den tynne plata over et vakuumkammer slik at alle målinger blir gjort relativt til vakuum.

Deloppdrag 1

Koble opp trykksensoren BMP280 og les av temperatur og lufttrykk. Skriv resultatene til monitoren med tekst foran verdien og benevning etter. Lag to funksjoner som leser av sensorene, skriver ut verdiene og returnerer henholdsvis lufttrykk og temperatur når de kalles fra void loop()-funksjonen.

Oppkobling

I denne sammenhengen skal vi bruke BMP280. Sensoren har et I^2C grensesnitt som lett lar seg koble til Arduino som vist på figuren til under. Her har vi også valgt å legge inn motstander til +3,3 V, som strengt tatt ikke er nødvendig når vi kun har ett element på I^2C-bussen.



Programmering

1. **Hent biblioteket for BMP280**
   Biblioteket for BMP280 finner man lett ved å søke på nettet. Her er et sted hvor biblioteket til BMP280 kan hentes fra: https://github.com/adafruit/Adafruit_BMP280_Library

2. **Legg inn bibliotekene**
   Bibliotekene legges inn helt først i programmet med følgende kommandoer:

```
#include <Wire.h>     // Inkluder bibliotek for kommunikasjon via I2C
#include <BMx280I2C.h>// Inkluder bib. for måling av lufttrykk
```

3. **Adressen til BMP280**
   Databladet for BMP280 oppgir at adressen er til BMP280 er HEX 0x76 som vi legger inn i en konstant:

```
#define I2C_ADDRESS 0x76
```

   Denne legges inn før void setup()-funksjonen.

4. **Deklarasjoner**
   Så deklarerer vi instansen *bmx280* av typen (klassen) *BMx280I2C*. Vi legger merke til at den inneholder adressen til sensoren.

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

   Deklarasjonen legges inn i starten før void setup()-funksjonen.
   I tillegg deklarer vi to variabler for P (lufttrykk) og T (temperatur).

```
float P;
float T;
```

5. **Initialisering**
   Så må vi initialisere I^2C-bussen og trykkmåleren:

```
Wire.begin();        // Initialisering av I2C-bussen
bmx280.begin();      // Initialisering av BMT280
```

   Disse legges inn i void setup()-funksjonen.

6. **Intern samling i BMP280**
   BMP280 har en intern punktprøvingsrate som kan settes til forskjellige verdier. Dette gjøres en gang før vi begynner å bruke sensoren. Det gjelder både for trykk- og temperatursensoren. Vi velger å sette oversamlingen til 16 ganger med følgende kommandoer:

```
bmx280.writeOversamplingPressure(BMx280MI::OSRS_P_x16);
bmx280.writeOversamplingTemperature(BMx280MI::OSRS_T_x16);
```

   Også disse kommandoene legges i void setup()-funksjonen.

7. **Legg inn feilmelding og vent**
   Først må man undersøke om sensorene er klar til å levere data, det gjør vi med følgende kommando: `if (bmx280.measure() == 0)`. Om så ikke er tilfelle så skrives det ut en melding: `Vent til forrige ....` Deretter legges programmet i en venteløkke (`do {...} while (<betingelse>))`, som venter på at måleverdier er klare til å leveres. Så lenge betingelsen er oppfylt (`bmx280.hasValue() == 0`) så er ikke dataene klare til å hentes og programmet vil bli stående i do-while-løkken. Programmet kan skrives slik:

```
if (bmx280.measure() == 0)
{
  Serial.println("Vent til forrige maaling er avsluttet");
```

```
}
else
{
  do // Vent til måling er klare, les så verdiene og skriv ut
  {
    delay(100);
  } while (bmx280.hasValue() == 0);

  P = bmx280.getPressure() / 100; // Returner avlest lufttrykk i mBar
  T = bmx280.getTemperature();    // Returner avlest temperatur

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

Når så betingelsen ikke lengre er oppfylt, dvs. at `bmx280.hasValue() == 1` går programmet videre å henter inn de målte verdiene for trykk (`P = bmx280.getPressure();`) og temperatur (`T = bmx280.getTemperature();`), som så skrives ut.

Kommentar:

Vi har to måter å bruke while på:

```
do
{
  // Gjør dette så lenge betingelsen til while() er sann
}
while (<betingelse>); // Legg merke til ";"
```

Eller vi kan skrive:

```
while (<betingelse>)
{
// Gjør dette så lenge betingelsen til while() er sann
}
```

Forskjellen er at hos `do ... while()` vil betingelsen undersøkes *etter* at innholdet i sløyfen er kjørt en gang, mens hos `while()` så testes betingelsen *før* innholdet i sløyfen kjøres.

8. **Lag funksjoner**
   Lag to funksjoner: `getPressure();` og `getTemperature();` som skriver ut og returnerer trykk og temperatur. Husk at testen for om måledata er klare bør inngå i begge funksjonene. Er du usikker på hvordan du lager egne funksjoner se avsnitt 2.5.13 på side 48.

6.2 Bruk av display

6.2.1 Grafisk OLED-display med SSD1306

Læringsmål

Oppdraget trener følgende læringsmål:

- Bruk av I^{2}C-buss
- Installasjon av biblioteker
- Skrive til OLED-display

Kort omtale

Det valgte displayet er relativt lite og billig og kan typisk vise tre til 5 linjer med kort tekst. Det egner seg godt for å skrive ut måleverdier og krever fra 3 – 5 V. Displayet programmeres via I^{2}C-buss og et bibliotek som hentes fra nettet. Det leveres vanligvis med en stiftlist med fire kontakter Vcc (5V), GND (jord), SDA (seriell data), SCL (seriell klokke). Og egner seg godt for å kobles til I^{2}C-bussen til f.eks. Arduino m.fl. Default adresse er 0x3C (hex), men kan omprogrammeres til 0x3D ved å flytte strapen mot høyre (innringet på bildet til høyre). En legger merke til at merkingen av kretsen synes å antyde at de to adressene er 0x78 eller 0x7A. Dersom man er i tvil hva som er riktig kan man bruke en I^{2}C scanner programvare som lastes inn i Arduino. Med displayet tilkoblet I^{2}C-bussen og med denne programvaren kan man lese av adressen. Se vedlegg E.1 side 181.

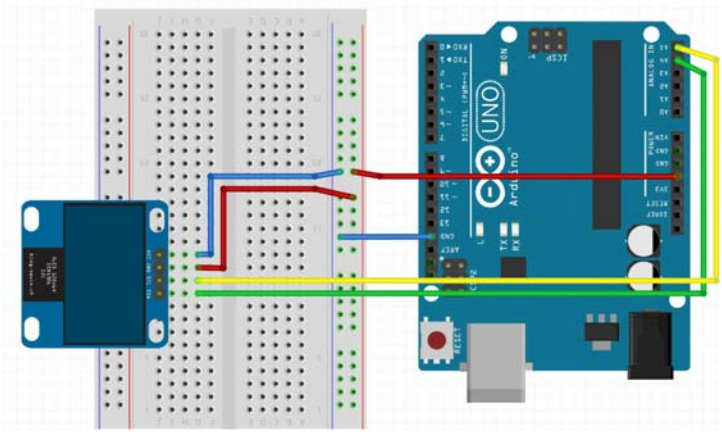
Displayet kan leveres som en-farget eller to-farget. Sistnevnte betyr at øvre og nedre del av displayet vises i forskjellige farger (se figuren til høyre over), dvs. at fargene er gitt av hardware.

Deloppdrag 2

> *Monter OLED-displayet SSD1306 og skriv ut temperatur og lufttrykk med navn på parametere og benevning. Temperatur og lufttrykk skal skrives ut under hverandre. Lag en funksjon som skriver temperatur og lufttrykk til OLED-displayet.*

Oppkobling

Figuren under viser hvordan vi kan koble displayet opp til en Arduino UNO-krets med kun to linjer (SDA og SCL) i tillegg til spenningsforsyning (5V) og jord (GND).



Programmering

Displayet krever at man installerer bibliotekene `Adafruit_GFX.h` og `Adafruit_SSD1306.h`. Disse kan hentes fra https://github.com/adafruit/Adafruit-GFX-Library og https://github.com/adafruit/Adafruit_SSD1306 (se avsnitt 2.5.15 side 51 for å se hvordan man kan installere biblioteker) eller man kan velge Sketch/Include Library/Manage Libraries/ og skrive Adafruit GFX og Adafruit SSD1306 i søkefeltet.

I tillegg kreves to biblioteker som omhandler serie-kommunikasjon `SPI.h` og `Wire.h`, disse er imidlertid inkludert i standardpakken som leveres med Arduino editoren (IDE) og trenger derfor ikke installeres.

1. **Inkluder bibliotekene i programmet**
   Øverst i programmet inkluderer man de nevnte bibliotekene. 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 `void setup()`.

2. **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
```

   Det er noe uklart hva 4-tallet i argumentet innebærer.

3. Initialiser og sett adressen til displayet

Displayet har som nevnt en adresse som programmet må vite om, denne er 3C (60) i heksadesimal kode (angitt som 0x3C) og er satt av fabrikken.

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

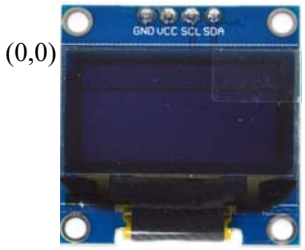
Initialiseringen av displayet gjøres i void setup()-funksjonen.

4. Posisjoner markør, tøm displayet og skriv ut tekst og variabler

Før vi skriver til displayet må vi tømme det, 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 void 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 eksempelvis ut tekst og variabelen i
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 plasseres 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 i øverste venstre hjørnet i karakterruta (origo), se figuren over.

5. Skriv ut trykk og temperatur

Bruk det som er omtalt over og skriv ut ønsket tekst og variabler, f.eks. lufttrykk og temperatur på de to øverste linjene, ev. andre sensordata.

6.3 Måling av temperatur

I dette avsnittet skal vi se på ulike måter å måle temperatur. Vi har alt sett at BMP280 også måler temperatur.

6.3.1 Måling av temperatur med TC74

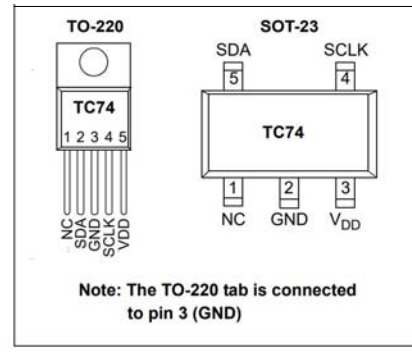
Læringsmål

Oppdraget trener følgende læringsmål:

- Måling av temperatur med TC74
- Bruk av I^2C-buss
- Installasjon av biblioteker

Kort omtale

TC74 er en halvlederbasert digital temperatursensor bygget inn i noe som minner om en krafttransistor (TO-220). Den kan derfor egne seg til å skrus fast i en komponent som man ønsker å måte temperaturen til. Sensoren leveres også for overflatemontasje (SOT-23).



TC74 har følgende egenskaper:

- ±2°C (maks.) nøyaktighet fra +25°C to +85°C
- ±3°C (maks.) nøyaktighet fra 0°C to +125°C
- Målt temperatur leveres som et 8-bits digitalt ord
- Verdien leses via en I^2C-buss
- Supplyspenning fra 2.7 V til 5.5 V
- Lav effekt: 200 µA (typ.) driftsstrøm og 5 µA (typ.) standby-strøm

Sensoren kan altså legges i dvale og trekker da svært lite strøm.

```
A0  =  1001 000
A1  =  1001 001
A2  =  1001 010
A3  =  1001 011
A4  =  1001 100
A5  =  1001 101 *
A6  =  1001 110
A7  =  1001 111
* Default Address
```

Kretsen finnes i flere utgaver og er blant annet optimalisert for henholdsvis 3,3 V og 5,0 V. Dette framgår av komponentbetegnelsen: TC74A0-3.3VAT hvor A0 angir adressen (B1001 000 = 0x48), 3.3 angir den optimale spenningen (3,3 V), V angir temperaturområdet (-40 til +125°C) og AT angir type kapsel (TO-220-5). Selv om den er optimalisert for 3,3 V så kan den også bruke for 5 V.

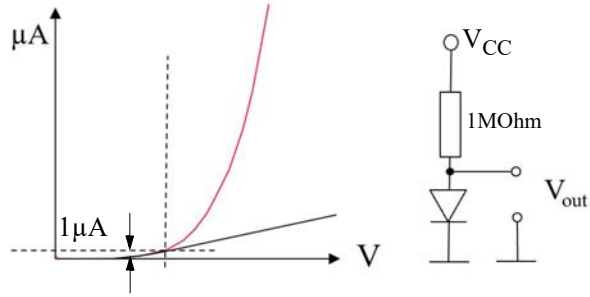
Adressen finnes i tabellen over på linja merket A0 (B1001000 (0x48)). Er man i tvil kan man bruke en I^2C scanner.

TABLE 4-4: TEMPERATURE-TO-DIGITAL VALUE CONVERSION (TEMP)

Actual Temperature	Registered Temperature	Binary Hex
+130.00°C	+127°C	0111 1111
+127.00°C	+127°C	0111 1111
+126.50°C	+126°C	0111 1110
+25.25°C	+25°C	0001 1001
+0.50°C	0°C	0000 0000
+0.25°C	0°C	0000 0000
0.00°C	0°C	0000 0000
-0.25°C	-1°C	1111 1111
-0.50°C	-1°C	1111 1111
-0.75°C	-1°C	1111 1111
-1.00°C	-1°C	1111 1111
-25.00°C	-25°C	1110 0111
-25.25°C	-26°C	1110 0110
-54.75°C	-55°C	1100 1001
-55.00°C	-55°C	1100 1001
-65.00°C	-65°C	1011 1111

Temperaturen angis binært som 2'ers komplement som vist i tabellen til venstre (Table 4-4).

Temperatur kan måles på mange forskjellige måter, bruk av pn-overgangen i en halvlederdiode er sannsynligvis den metoden som brukes her.



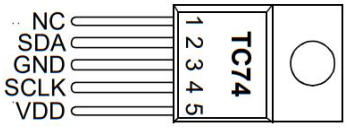
Det viser seg at spenningen over en slik pn-overgang (diode) vil være avhengig av temperaturen. Dioden forspennes i lederetning med en stor motstand slik at det går en ørliten, nærmest konstant strøm (1µA) gjennom dioden (se figuren over til høyre). Dette gir en tilnærmet lineær endring av spenningen over dioden som funksjon av temperaturen. Det er denne spenningsvariasjonen som utnyttes til å bestemme temperaturen. Spenningen forsterkes, kalibreres og digitaliseres slik at den kan avleses via I^2C-bussen.

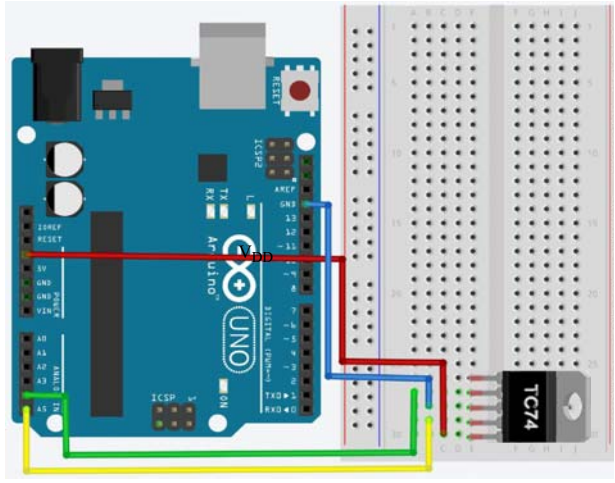
Deloppdrag 3

Monter sensoren TC74 og skriv programmet som leser av temperaturen. Skriv ut den avleste temperaturen til monitoren eller til et OLED-display. Legg avlesningen av temperaturen i en egen funksjon og la funksjonen returnere temperaturen.

Oppkobling

Som det framgår av figuren til høyre så tilføres kretsen 3,3 V (V_{DD}) og GND. Dataene leses av via SCLK (seriell klokke I^2C) og SDA (serielle data I^2C). Den siste pinnen brukes ikke (NC).





Programmering

For denne kretsen kan vi enten velge å bruke et bibliotek fra Adafruit eller programmere registrene inne i komponenten direkte. Vi velger her å skrive og lese direkte til registrene via I^2C-bussen. Uansett må vi inkludere biblioteket som lar oss kommuniserer via I^2C-bussen:

1. **Inkluder biblioteket**
   Biblioteket for kommunikasjon på I^2C-bussen inkluderes i programmet ved å skrive:
   ```
   #include "Wire.h"
   ```
   Denne kommandoen legges øverst i programmet.

2. **Deklarer adressen og globale variable**
   Deklarerer en konstant som holder adressen til TC74:
   ```
   #define address B1001 000
   ```
   Denne kommandoen legges inn før void setup().

3. **Initialiser I^2C-bussen**
   Følgende kommando initialiserer I^2C-bussen, samtidig deklarerer vi en variabel som skal holde temperaturen:
   ```
   Wire.begin();
   int temperature;
   ```
   Disse kommandoene plasseres i void setup()-funksjonen.

4. **Start overføring**
   Start overføring av data til komponenten via I^2C-bussen
   ```
   Wire.beginTransmission(address); //start overføring
   ```
   Denne inkluderes i void loop()-funksjonen eller ev. en egendefinert funksjon før overføringen av data starter.

5. **Overfører data**
   Data hentes fra TC74 med adressen `address` og register `0x00`^{16} og skriver ut resultatet:
   ```
   Wire.write(0x00); // Registeret der temperaturen lagres
   Wire.requestFrom(address, 1);
   if (Wire.available())
     {
       temperature = Wire.read();
       Serial.print("Temp.: ");
       Serial.print(temperature);
     }
    else
     {
       Serial.println("---""); // Ingen data, skriv ut streker
     }
   ```
   Denne plasseres i void loop()-funksjonen etter `Wire.beginTransmission(address);`
   Legg merke til at det skrives ut streker dersom ingen data er rapportert tilgjengelig. Dernest prøver man å lese av verdien på nytt i neste runde gjennom void loop()-funksjonen.

16.https://ww1.microchip.com/downloads/en/DeviceDoc/21462D.pdf (side 8)

6. **Avslutt overføring**
   Vi avslutter overføringen med en kommando tilsvarende den vi begynte med:
   ```
   Wire.endTransmission(); //Avslutt overføring
   ```
   Plasseres etter at overføringen er ferdig.

Forslag til tilleggsoppgaver:

- **Termostat**
  La temperatursensoren registrere temperaturen. Legg inn en terskel og start en vifte dersom temperaturen overskrider terskelen. Ev. slå på en ovn dersom temperaturen synker under terskelen.
- **Værstasjon**
  La temperatursensoren inngå som temperaturmåler i en værstasjon

6.4 Måling av luftfuktighet

Sensoren måler relativ fuktighet. Store norske leksikon definerer relativ fuktighet slik:

> *Relativ fuktighet er forholdet mellom den absolutte fuktigheten i luften og den luftfuktigheten som må til for å oppnå metning ved en gitt temperatur. Den oppgis i prosent, slik at luft som er helt mettet med vanndamp, har en relativ fuktighet på 100 prosent.*

Som vi ser av definisjonen så er den relative fuktigheten avhengig av temperaturen. Derfor vil måling av fuktighet og temperatur henge nøye sammen og kombineres gjerne i samme sensor.

6.4.1 Måling av temperatur og luftfuktighet med DHT11

Læringsmål

Oppdraget trener følgende læringsmål:

- Hva relativ luftfuktighet er
- Bruk av I^2C-buss
- Installasjon av biblioteker

Kort omtale

Relativ fuktighet

Luft har evnen til å oppta vanndamp. Men mengden som kan tas opp er ikke ubegrenset. Når lufta ikke klarer å ta opp mer, sier vi at den er ***mettet***. Luft som er mettet med vanndamp defineres å ha en relativ fuktighet på 100%. Dersom lufta er helt fri for vanndamp er den tørr og vil ha en relativ fuktighet på 0%.

100% fuktighet tilsvarer et visst antall gram vann pr. m^3. Hvor mange gram dette tilsvarer er avhengig av både lufttrykket og ikke minst av temperaturen. Jo høyere temperatur jo mer vanndamp kan lufta oppta før den er mettet. Når skyer driver inn mot land, blir de gjerne presset opp

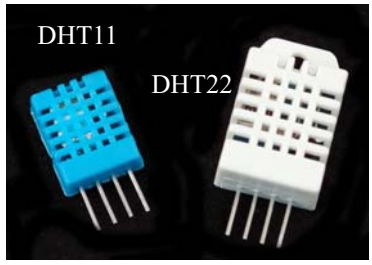
i kaldere luftlag. Når mettet luft blir kaldere vil den gi fra seg noe av fuktigheten som går over til tåke eller regn. Det er verdt å merke seg at vanndamp er usynlig. Det vi ser av skyene der derfor ikke først og fremst vanndamp, men små vanndråper.

Det er vanlig at man måler trykk og temperatur sammen med fuktighet. Et slikt kombinert måleinstrument kalles ofte en PTU-sonde (**P**ressure - **T**emperature - h**U**midity).

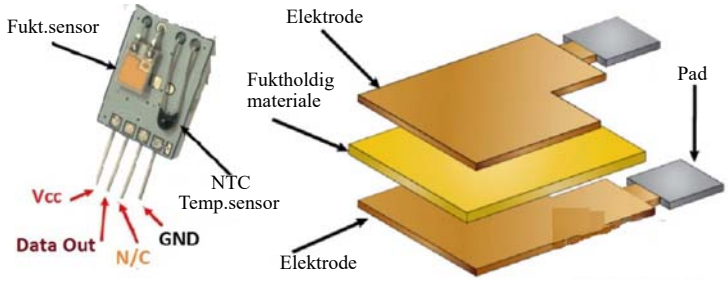
Fuktighetssensoren

DHT11 og DHT22 er to sensorer for måling av relativ luftfuktighet og temperatur. DHT22 er en noe nyere versjon av DHT11. Begge er basert på en kapasitiv fuktighetssensor. Temperaturen og luftfuktigheten overføres via en *entråds-buss*. De to er sammenlignet i tabellen under:

Parameter	DHT11	DHT22
Pris	Typ. 5$	Typ. 10$
Spenningsforsyning	3 – 5 V	3 – 5 V
Strømtrekk	2,5 mA	2,5 mA
Måleområde fuktighet	0 – 80% Tol. 5%	0 – 100% Tol. 2-5%
Måleområde temperatur	0 – 50°C Tol. ±2%	-40 – 80°C Tol. ±0,5%
Punktprøvningsrate	1Hz	0,5Hz
Størrelse	15.5 x 12 x 5.5mm	15.1 x 25 x 7.7mm
Pinne avstand	4 pin. avst. 2,52 mm	4 pin. avst. 2,52 mm



Sensorens virkemåte[17]*:*



Sensoren DHT11 og DHT22 består av en kapasitiv fuktighetssensor og en NTC-motstand. Den kapasitive fuktighetssensoren består av en tynn skive som lett absorberer og avgir fuktighet i takt med fuktigheten i lufta. Metallelektroder ligger på begge sider av materialet slik at det dannes en kondensator. Materialets fuktighet påvirker dielektrisitetsegenskapene til materialet slik at kapasitansen endrer seg med fuktigheten. Sensorene inneholder elektronikk som behandler målingene av kapasitansen til fuktighetsmåleren og resistiviteten i NTC-motstanden slik at målingene kan leveres digitalt i relativ fuktighet og grader Celsius langs en entråds seriebuss.

En-tråds databuss

Dette er et kommunikasjonsformat som ble utviklet av Dallas Semiconductor Corp. for enkel lavhastighetskommunikasjon mellom en *master* f.eks. Arduino, og en sensor eller en minnebrikke, *slaven*. Datahastigheten er normalt 16.3 kbps (kilo bit pr. sekund) som kalles "Standard speed". Det finnes også en variant som opererer på høyere hastighet, denne går under betegnelsen "Over-

17.https://www.theengineeringprojects.com/2019/02/introduction-to-dht22.html

drive". De fleste komponentene som benytter denne type dataoverføring tillater begge hastighetene. Se avsnitt 2.2.3 på side 32 eller referanse [10] for mer informasjon.

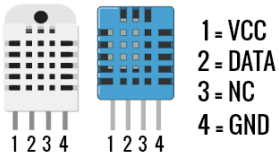
Deloppdrag 4

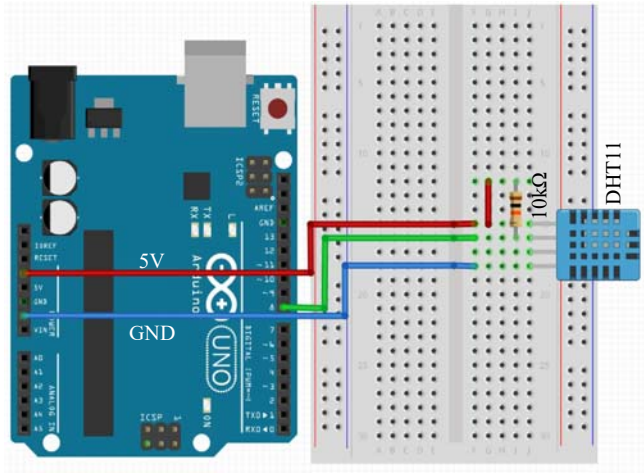
Koble opp og les av temperatur og luftfuktighet. Skriv resultatet til monitoren eller et OLED-display med beskrivende tekst (f.eks. Luftfukt.) og benevning (f.eks. mBar). Lag en funksjon som leser av temperatur og luftfuktighet, skriver ut resultatet og returnerer luftfuktighet.

Oppkobling:

Oppkoblingen er særdeles enkel. En må imidlertid sørge for å koble en ekstern motstand på 5 – 10kΩ mellom datautgangen og pluss-spenningen som vist på figuren til høyre.Pinningen på DHT11 og DHT22 er lik.

I denne sammenhengen velger vi å bruke DHT11.





Programmering[18]:

- **Last ned bibliotekene.**
  Det trengs to biblioteker for å kunne bruke DHT11. Det er det generelle sensorbiblioteket til Adafruit: https://github.com/adafruit/Adafruit_Sensor og det spesielle biblioteket for DHT11: https://github.com/adafruit/DHT-sensor-library Bibliotekene kan også installeres ved hjelp av Arduino Library Manager og søke etter *DHT sensor library Adafruit* og *Adafruit unified sensor library.*

1. **Inkluder bibliotekene i koden.**
   Dernest inkluderes de to bibliotekene i koden ved å skrive:
   ```
   #include <Adafruit_Sensor.h>
   #include "DHT.h"
   ```
   i starten av programmet.

2. **Deklarer instansen DHT.**
   Så deklareres instansen *dht* av typen *DHT*. Denne skal ha to parametere. Den ene er hvilken type sensor vi skal bruke, DHT11 eller DHT22, og den andre parameteren, hvilken port på Arduino'en som skal brukes for å overføre data. Vi velger port D8. Disse to parameterne defi-

18.https://learn.adafruit.com/dht/using-a-dhtxx-sensor

neres på forhånd som to konstanter:

```
#define DHTPIN 8         // Digital port tilknyttet DHT sensoren
#define DHTTYPE DHT11    // Alternativt DHT22 (AM2302), AM2321
DHT dht(DHTPIN, DHTTYPE);// Deklarasjon av instansen dht( );
```

Denne deklarasjonen legges inn før void setup()-funksjonen.

3. **Initialiser sensoren DHT11.**
   I void setup()-funksjonen initialiserer vi DHT-sensoren med følgende kommando:

   ```
   dht.begin();
   ```

4. **Lesing av data fra sensoren.**
   Vi leser verdiene for luftfuktighet og temperatur fra sensoren med følgende funksjoner:

   ```
   float h = dht.readHumidity();    // Les relativ luftfuktighet i %
   float t = dht.readTemperature(); // Les temperatur i grader Celcius
   ```

   Disse legges inn i void loop()-funksjonen. Legg gjerne inn en forsinkelse i loopen slik at avlesning skjer hvert andre sekund e.l. Selve lesingen tar ca. 250 msek.

5. **Sjekk feil ved lesning**
   Det kan være lurt å legge inn en sjekk om lesing av sensoren har vært vellykket. Det gjør vi ved å bruke funksjonen `isnan(h)` for luftfuktighet og `isnan(t)` for temperaturen. Dersom en av disse returnerer "1" så har lesningen vært mislykket. Returnerer begge "0" kan verdiene skrives ut.

   ```
   if (isnan(h) || isnan(t))
   {
     Serial.println(F("Mislykket lesing av DHT11!"));
     return;
   }
   else
   {
       // Skriv ut luftfuktighet og temperatur her
   }
   ```

   Skriv gjerne inn kommandoer for skriving til monitor eller OLED-display i koden over.

6. **Skriv og test programmet**
   Skriv programmet og test at det virker etter forventningene.

7. **Lag en funksjon**
   Lag en funksjon som leser av luftfuktighet og temperatur fra DHT11 og skriver resultatet til monitoren eller et display.

Forslag til tilleggsoppgaver:

Her er noen forslag til tilleggsoppgaver:

- **Værstasjon**
  Lufttemperatur og luftfuktighet er to viktige målinger i forbindelse til en værstasjon

- **Alarm**
  Lufttemperatur kan knyttes til en alarm i en bil for å advare mot glatte veier. Synker luft-temperaturen under 3°C så kan det være fare for ising på veien.
- **Funksjon med argument**
  Bruk argumentet til funksjonen til å bestemme om det skal være temperatur eller luftfuktighet som returneres.

6.5 Måling av lysstyrke

I dette avsnittet skal vi se på et par sensorer som kan måle eller registrere endringer i lysstyrke.

6.5.1 Candela, lux og lumen

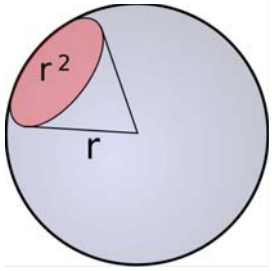
Måleenhetene for lysintensitet er mange og kan lett oppfattes som forvirrende. Vi henter følgende informasjon fra Teknisk Ukeblad[19]

Lysintensitet – candela (cd)

Candela (cd) er den fundamentale enheten for lysintensitet definert i SI-systemet. Ordet betyr stearinlys på latin, og lysstyrken 1 cd er omtrent den intensiteten som strømmer mot deg når du ser på et brennende stearinlys. Én candela we lysstyrken i en gitt retning til en lyskilde.

Den vitenskapelige definisjonen av 1 cd er ganske kompleks og tilsvarer 1/683 watt pr. steradian[20] ved 556 nanometer bølgelengde (grønt lys – 540 000 GHz), som er den bølgelengden øyet er mest følsomt for. Det er derfor ikke så rart at ambulansene har skiftet farge til gul/grønt.

Lysstyrke – lumen (lm)



Lumen angir hvor mye lys som totalt strømmer ut fra kilden, og er den betegnelsen som nå brukes i stedet for effekten slik tilfellet var for glødelamper. Lumen angir derfor den totale lysmengden som strømmer ut av en lyskilde, vi kaller denne for *lysstyrken*. Summerer vi lysintensiteten (candela) i alle retninger, finner vi lysstyrken (lumen).

Siden lysintensiteten kan variere avheng av retningen i forhold til lyskilden, er det ingen enkel sammenheng mellom lysintensitet og lysstyrke, men vi kan da finne en sammenheng under visse betingelser. Dersom vi antar at lysintensiteten er like stor i alle retninger så kan vi skrive at:

$$1 \text{ candela} = 4\,\pi\,r^2 \text{ lumens} = 12.57 \text{ lumen (når r = 1 m)}$$

Hvor $4\,\pi\,r^2$ er hele kuleflata rundt målt i steradianer.

19. https://www.tu.no/artikler/dette-er-forskjellen-pa-lumen-lux-og-candela/275787
20. En steradian er den romvinkelen som med toppunkt i sentrum av en kule med radius *r*, avgrenser et areal av kuleoverflaten som er lik r^2. Ettersom arealet av overflaten av en kule er $4\pi r^2$, følger det av definisjonen at en kule måler 4π steradianer, eller ≈12,56637 steradianer. På samme måte er 1steradian lik 1/12,56637, eller ca. 8% av en kuleflate (https://no.wikipedia.org/wiki/Steradian).

Belysningsstyrke (illuminans) – lux (lx)

Lux er en betegnelse for *belysningsstyrke* (*illuminans*) og forteller hvor mye lys som faller på en flate. Lux er dermed lumen pr. kvadratmeter og forteller hvor opplyst en flate er, f.eks. et arbeidsområde.

Øyet ser ikke belysningsstyrken, det ser hvor mye lys som reflekteres fra f.eks. arbeidsområdet. Da snakker vi om *luminans*, eller *lystetthet*, som er en funksjon av belysningsstyrken og hvor mye lys som reflekteres fra overflaten. Det er enorm forskjell på hvor mye sollys som reflekteres fra hvit snø og fra svart asfalt. Enheten for luminans er den samme som for belysningsstyrke, som er lumen pr. kvadratmeter eller lux.

Lysintensitet måles i lux. 1 lux er 1 lumen pr. m^2 som tilsvarer:

- *Fullt sollys 11 000 lux (eller ca. 1000 W/m^2)*
- *Sollyset en tidlig morgen 6 000 lux*
- *Belysningen i et TV-studio 1 000 lux*
- *Et godt opplyst kontor 400 lux*
- *Lyset fra en fullmåne 1 lux*

Rekkevidde – meter (m)

Rekkevidden til en lyskilde er den maksimale avstanden fra lyskilden til der lys kan belyse et område på minst 0,25 lux, hvilket tilsvarer omtrent en fullmånes belysning. Rekkevidden kan beregnes direkte fra intensiteten.

6.5.2 Deteksjon av lys med LDR – Light Dependent Resistor

Læringsmål

Oppdraget trener følgende læringsmål:

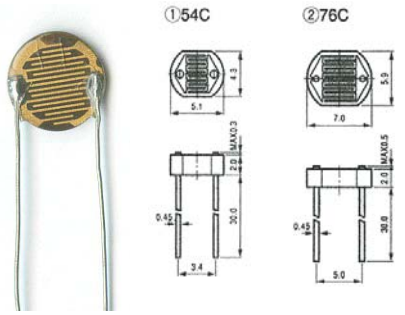
- Omforming fra variabel motstand til variabel spenning
- Lysdeteksjon versus lysmåling
- AD-konvertering
- Omregning fra digitalt tall til spenning
- Bruk av terskelverdi

Kort omtale

En fotomotstand (LDR) egner seg bedre for å skille lys fra mørke enn til å måle lysstyrke eller belysningsstyrke siden sammenhengen mellom belysningsstyrke og resistansen er sterkt ulineær, vi sier at vi *detekterer* lys eller mørke. Med deteksjon mener vi derfor evnen til å skille mellom lys og mørke ved en gitt belysningsstyrke, en *terskelverdi*. Ved terskelverdien vil *detektoren* svitsjer fra høy til lav eller omvendt.

I dette oppdraget skal vi se hvordan vi kan bruke en fotomotstand som en lysdetektor.

Fotomotstand (LDR – Light Dependent Resistor)



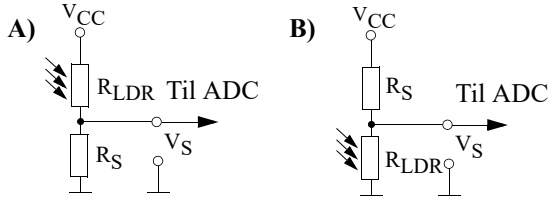
Fotomotstander har gjennom tidene vært en gjenganger i mange elektronikkprosjekter og brukes i ulike sammenhenger som f.eks. for automatisk tenning av gatebelysning, som enkle lysmålere, som automatisk tenning av frontlyktene på bil, i forbindelse med innbruddsalarm ved at en lysstråle brytes o.l.

Fotomotstander har tradisjonelt vært laget av Cadmium-Sulfid (CdS) belagt med fingerelektroder som vist på figuren til høyre. I mørket vil stoffet CdS være omtrent isolerende og kan gi en motstand på over 1 MΩ. Belyses stoffet, kan resistansen i fotomotstanden falle til under 1 kΩ. Lysfølsomme motstander er imidlertid relativt langsomme og avhengig av temperaturen. For mer informasjon, se avsnitt 8.2.

Fra variabel motstand til variabel spenning

For å konvertere endring i resistans til endring i spenning, kan vi bruke en enkel *spenningsdeler* (se figuren til høyre).

Siden mikrokontrolleren krever en spenning, kobler vi LDR-motstanden i serie med en fast motstand (seriemotstand) som vist i figuren til høyre. Velg verdien på seriemotstanden lik den typiske resistansen til fotomotstanden (LDR) i det aktuelle belysningsstyrken der fotomotstanden skal brukes.



$$V_S = \frac{R_S}{R_{LDR} + R_S} V_{CC} \qquad V_S = \frac{R_{LDR}}{R_{LDR} + R_S} V_{CC}$$

Ønsker man maksimal spenningsvariasjon fra dypt mørke til sterkt lys kan optimal seriemotstand beregnes ut fra følgende ligning [8]:

$$R_S = \sqrt{R_{LDR(light)} \times R_{LDR(dark)}} \tag{6.1}$$

hvor $R_{LDR(light)}$ er resistansen i sterkt lys og $R_{LDR(dark)}$ er resistansen i mørke.

Spenningsnivået V_S beregnes fra formlene som angitt på figuren over. Legg merke til at oppkoblingen på tegning A gir økende spenning (V_S) med økende lysstyrke, mens oppkoblingen i tegning B gir fallende spenning (V_S) med økende lysstyrke. For en utdypende diskusjon av spenningsdeleren se avsnitt 8.1 side 164.

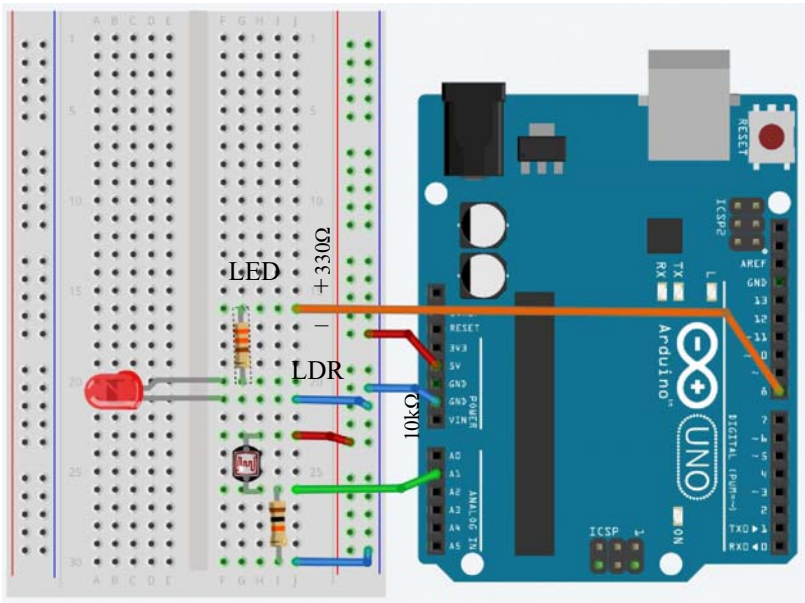
Deloppdrag 5

Koble opp en fotomotstand (LDR) og en lysdiode (LED) og lag et program som er slik at lysdioden tenner når det blir mørkt og slukker når det blir lyst. Legg inn en terskelverdi for spenningen fra LDR-kretsen slik at omslaget fungerer når fotomotstanden dekkes til med et

A4-ark. Skriv ut terskelverdien og målt spenning fra LDR-kretsen i monitoren. Legg målingen av belysningstyrken og utskrift i en funksjon og la funksjonen returnere en sann verdi dersom den detekterer mørke (TRUE) og en usann verdi dersom den detekterer lys (FALSE).

Oppkobling

Oppkoblingen er ganske enkel. Påse at langt bein hos lysdioden (anoden) kobles nærmest + og kort bein (katoden) nærmest –.



Programmering

1. **Deklareringer**
   Deklarer variabler knyttet til analogt signal fra LDR (A1), signal til LED (D8), avlest digital verdi LDR, beregnet spenningsverdi LDR og terskelspenning LDR.
2. **Void setup()**
   Forbered skriving til monitor og LED-signal (D8) som utgang.
3. **Les LDR verdi**
   Les den digitale LDR-verdien og regn om til spenning, når vi vet at AD-konverteren har 10 bit og referansespenningen er 5.0 V.
4. **Tenn/slukk LED**
   Tenn LED dersom lyset dempes på fotomotstanden og slukk LED dersom fotomotstanden belyses. Sett terskelspenningen slik at omslaget blir funksjonelt. Skriv i monitoren det som skjer med lysdioden (slukkes/tennes).

6.5.3 Måling av lys med BH1750 (GY-302)

Læringsmål

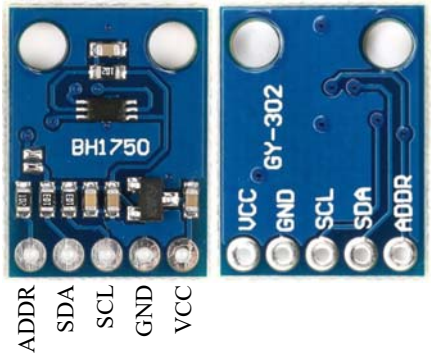
Oppdraget trener følgende læringsmål:

- Betydningen av candela, lumen og lux

- Måling av belysningsstyrke
- Installasjon av biblioteker
- Bruk av I^2C-buss
- Bestemme middelverdien

Kort omtale

LDR egner seg som tidligere nevnt, ikke umiddelbart for måling av belysningsstyrke da den er ganske ulineær og sterk avhengig av seriemotstanden. BH1750 er linearisert og nettopp egnet til slike målinger. Av praktiske årsaker vil vi her bruke et "break out" kort (GY-302) som gjør kretsen mer lettere å koble opp til vårt mikrokontrollerkort.



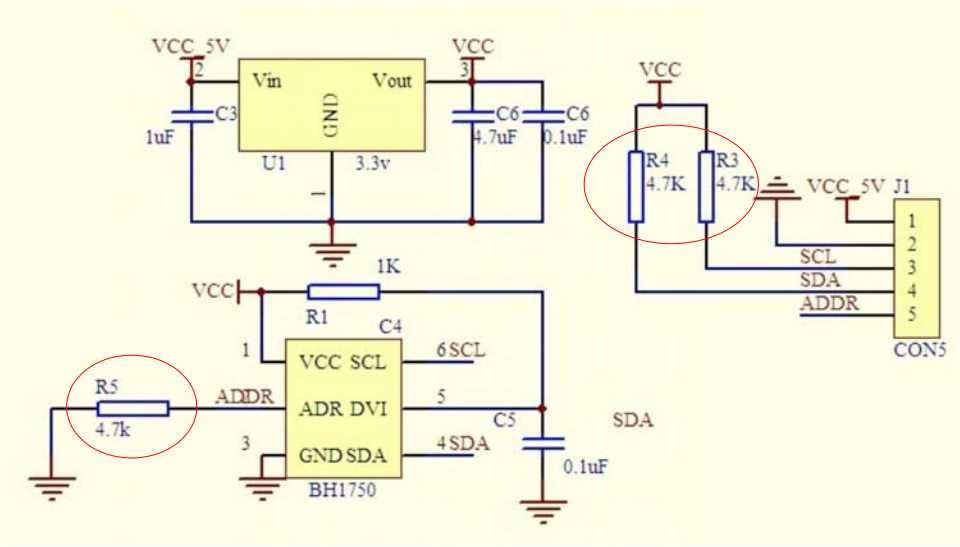
VCC – Forsyningsspenning 3 – 5V
GND – Jord (–)
SCL – Serieklokke for I^2C
SDA – Seriedata for I^2C
ADDR – Adresse
ADDR – LAV – 0x23 (Default)
ADDR – HØY – 0x5C

Nøkkelparametere for GY-302:

- Sensor: BH1750FVI montert på "break out" GY-302
- Spenningsforsyning: 3 – 5 V, reguleres ned til 3,0 V på kortet
- Strømforbruk: 0,12 mA
- Måleområde: 0 – 65535 Lux, innebygget 16 bit AD-konverter
- Måleoppløsning 1 Lux
- Målevariasjon ±15%
- I^2C-buss for kommunikasjon
- Unødvendig med kalibrering
- IR-stråling har liten innvirkning på måleresultat
- Kretsen undertrykker flimring som skyldes støy fra nettspenningen, 50/60Hz
- Spektralfølsomheten ligner på øyets
- Lite følsom for spekteret til lyskilden
- Adressen er HEX 0x23 dersom man ikke gjør noen ting med denne terminalen. Se kretsskjema under.

Kretsskjema

Figuren under viser kretsskjemaet for "break out" kortet GY-302. Vi legger mekre til at ADDR-linjen er lagt til jord (0) via en 4,7 kΩ motstand. Vi legger også merke til at I^2C-bussen er lagt høy med to 4,7 kΩ motstander (innringet).



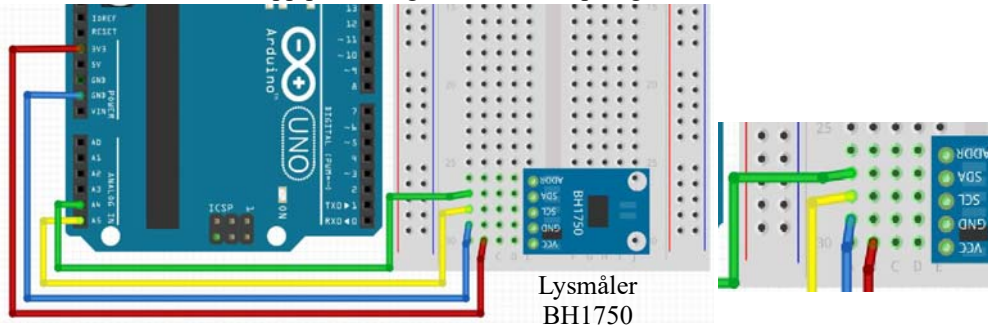
DVI er lagt til jord via en 0,1µF kondensator som lades opp gjennom R1 på 1 kΩ. Dette sikrer at DVI holdes lav et kort øyeblikk etter at strømmen er slått på, og sørger for at de interne registrene resettes ved påslag.

Deloppdrag 6

> *Monter og lag et program som leser av belysningsstyrken i lux og skriver resultatet til monitoren eller et OLED-display. Gjør 100 målinger og finn middelverdien. Sammenlign måleresultatet med en kalibrert lysmåler. Legg avlesning, midling og skriving til monitor eller display, i en egen funksjon. Returner lysintensiteten i Lux.*

Oppkobling

Kretskortet kobles enkelt opp på koblingsbrett som vist på figuren under.



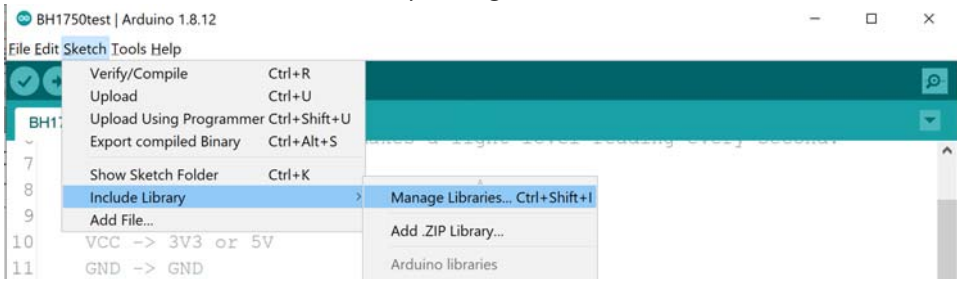
Lysmåler
BH1750

Programmering

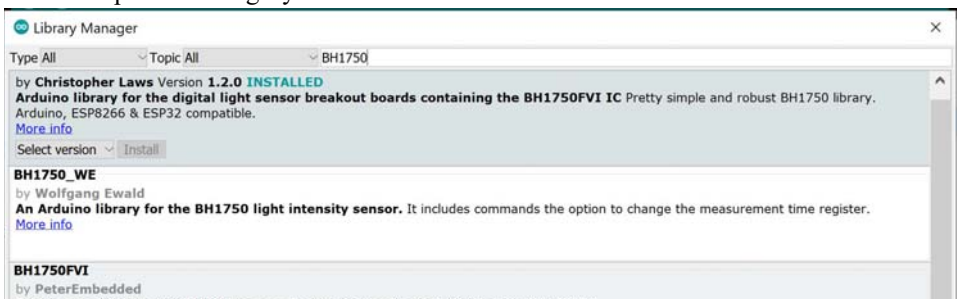
Oppbygging av programmet

1. **Installasjon av bibliotek**

   Det første som må gjøres er å installere et bibliotek som kommuniserer med BH1750 via I^{2}C-bussen. Det enkleste er å bruke *Library manager:*



   Når vi skriver BH1750 i søkefeltet får vi opp følgende bilde og velger den øverste varianten av Christopher Laws og trykker INSTALL.



2. **Inkluder nødvendige biblioteker**

   For å bruke lyssensoren BH1750 trenges to biblioteker: `BH1750.h` og `Wire.h` for å bruke I^{2}C-bussen. Disse inkluderes ved å bruke kommandoene:

```
#include <Wire.h>    // Bibliotek for bruk av I2C-buss
#include <BH1750.h>  // Bibliotek for henting av data fra BH1750
```

   Disse plasseres gjerne helt i begynnelsen av programmet og før void setup().

3. **Deklarer et objekt for BH1750**

   Det neste som må gjøres er å deklarere et objekt *lightMeter* av typen *BH1750*.

```
BH1750 lightMeter; // Deklarasjon av objektet lysstyrke type BH1750
```

   Også denne kommandoen skal plasseres før void setup().

4. **Initialisering av bibliotekene**

   Dernest initialiseres de to bibliotekene

```
Wire.begin();        // Initiering av I2C-bussen
lightMeter.begin(); // Initialisering av sensoren
```

   Initialiseringen plasseres i void setup()-funksjonen.

5. **Lesing av måledata**
   Så er vi klare til å lese ut måledata

```
float lux = lightMeter.readLightLevel(); // Leser måleverdien
```

   Her deklarerer vi variabelen lux og leser samtidig måleverdien fra sensoren inn i variabelen. Normalt plasseres avlesningen i void loop()-funksjonen eller i en egendefinert funksjon. Variabelen blir da lokal for void loop().

6. **Skriver ut den målte verdien**
   Tilslutt skriver ut verdien til monitoren:

```
Serial.print("Lysstyrke: ");   // Skriv ut tekst før verdien
Serial.print(lux);             // Skriv ut måleverdien
Serial.println(" lx");         // Skriv ut benevning
```

   Utskriften plasseres vanligvis i void loop()-funksjonen.

7. **Midling**
   Dersom vi studerer måleverdiene så ser vi at de variere en del fra måling til måling selv om belysningstyrken oppfattes som konstant. Årsaken kan være tilfeldig støy, se figuren under. Tilfeldig støy kan fjernes ved en enkel midling.

```
float lux = 0;
for (int i=0; i<100; i++)
  {
    lux = lux + lightMeter.readLightLevel();
  }
lux = lux/100;
```

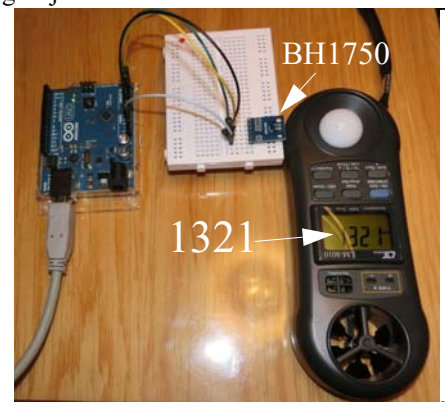
   For-loopen plasseres i void loop()-funksjonen eller i egendefinerte funksjoner.

```
Light: 627.50 lx   Lysstyrke: 623.12 lx
Light: 633.33 lx   Lysstyrke: 621.93 lx
Light: 612.50 lx   Lysstyrke: 623.54 lx
Light: 628.33 lx   Lysstyrke: 623.32 lx
Light: 620.83 lx   Lysstyrke: 623.05 lx
Light: 636.67 lx   Lysstyrke: 623.42 lx
Light: 609.17 lx   Lysstyrke: 622.16 lx
Light: 603.33 lx   Lysstyrke: 620.90 lx
Light: 620.00 lx   Lysstyrke: 620.07 lx
```

Uten midling | Med midling over 100 målinger

8. **Kalibrering**
   Fabrikanten reklamerer med at sensoren ikke trenger å kalibreres. Men det vil alltid være nyt-

tig å sjekke måleverdiene med et kalibrert måleinstrument.




```
Lysstyrke: 1346.32 lx
Lysstyrke: 1348.67 lx
Lysstyrke: 1348.40 lx
Lysstyrke: 1342.43 lx
Lysstyrke: 1345.35 lx
Lysstyrke: 1342.52 lx
Lysstyrke: 1344.52 lx
Lysstyrke: 1340.29 lx
Lysstyrke: 1338.83 lx
```

Av bildet overs er vi at overensstemmelsen er rimelig god. Vi har brukt LM-8010 som sammenligning med BH1750.

6.6 Måling av vindhastighet med SEN-15901

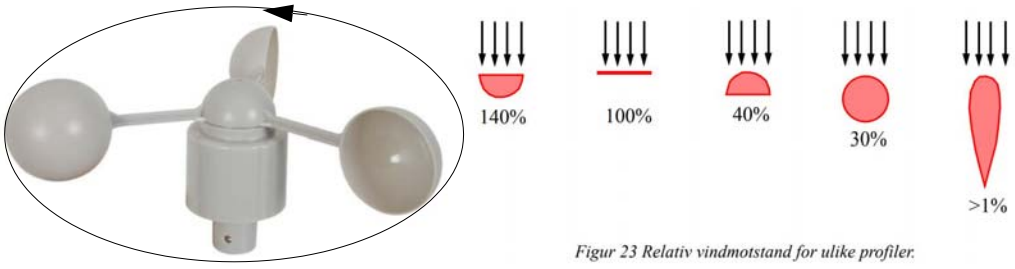
Læringsmål

Oppdraget trener følgende læringsmål:

- Virkemåte til et skål-anemometer
- Registrering av rotasjon
- Bruk av interrupt
- Bruk av millis()-funksjonen
- Omregning fra omdreininger pr. sekund til vindhastighet

Kort omtale

Settet SEN-15901 inneholder bl.a. en roterende vindstyrkemåler med skåler. Målinger viser at luftmotstanden mot framsiden av en skål er langt større enn mot baksiden som er en halvkule[21]. Dette er vist til på figuren under. Referansen i figuren er en flat sirkulær plate (nr. 2 fra venstre).



Figur 23 Relativ vindmotstand for ulike profiler.

21.https://www.ntnu.no/documents/2004699/11799071/SL+06+-+Luft+og+Str%C3%B8mninger.pdf/786a57a6-6831-4886-949a-8efa95b3d8d9?t=1573645769821

Vi ser at en halvkule med den flate siden mot vinden øver 1,4 ganger så stor luftmotstand som en sirkulær flate med samme tverrsnitt. Mens en halvkule som vender den runde siden mot vinden bare yter en luftmotstand på 0,4 ganger som den sirkulære flata. Dermed vil skål-anemometeret påvirkes av en resultantkraft som får den til å rotere kraftig mot venstre, som vist til venstre på figuren over, uansett fra hvilken retning vinden kommer. Dersom anemometeret har en god utforming skal det være en lineær sammenheng mellom vindstyrke og rotasjonshastighet.

For hver *halve* omdreining slutter et reed-rele kontakten og anemometeret gir ut en puls. Ved å telle pulser pr. tidsenhet kan vindhastigheten beregnes. Dersom vi også kjenner vindhastigheten som funksjon av antall klikk pr. sekund så kan vi beregne vindhastigheten. Leverandøren av vindmåleren oppgir den til å være 2,4 km/time pr klikk pr.sekund[22], hvilket tilsvarer 0,667 m/s.

Målemetode

Metoden for å måle vindhastigheten er ganske enkel. Vindmåleren har to ledninger. Disse slutter kretsen to ganger pr. rotasjon, vi kaller det et "klikk". Ved å telle antallet klikk i løpet av en tidsperiode kan vi beregne gjennomsnittlig antall omdreininger pr. sekund. når vi at leverandøren av vindmåleren oppgir den til å være 2,4 km/time pr. klikk pr. sekund[23], hvilket tilsvarer 0,667 m/s.

Midlere vindhastigheten kan da beregnes ut fra følgende formel:

Midlere vindhastighet = Vindhastighet pr. klikk pr. sek. x antall klikk / midlingstiden [s] (6.2)

Hvor *vindhastighet pr. klikk pr. sekund* = 0,667 m/s

Bruk av interrupt

Klikkene er kortvarige og skal man sjekke tilstanden med jevne mellomrom så går man ganske sikkert glipp av et klikk i ny og ne. Vi bruker derfor hardware interrupt som avbryter programmet uansett hvor det er i koden og hopper til *interrupt-rutinen* og registrerer klikket. Arduino UNO har to innganger som kan benyttes i forbindelse med interrupt, det er D2 (INT0) og D3 (INT1).Vi velger INT0 for vindmåleren.

Ellers anbefales å ta en titt på avsnitt 2.5.14 på side 50 som omhandler bruk av interrupt.

Kalibrering

Kalibrering kan gjøres med et kalibrert anemometer som vist på figuren til høyre.

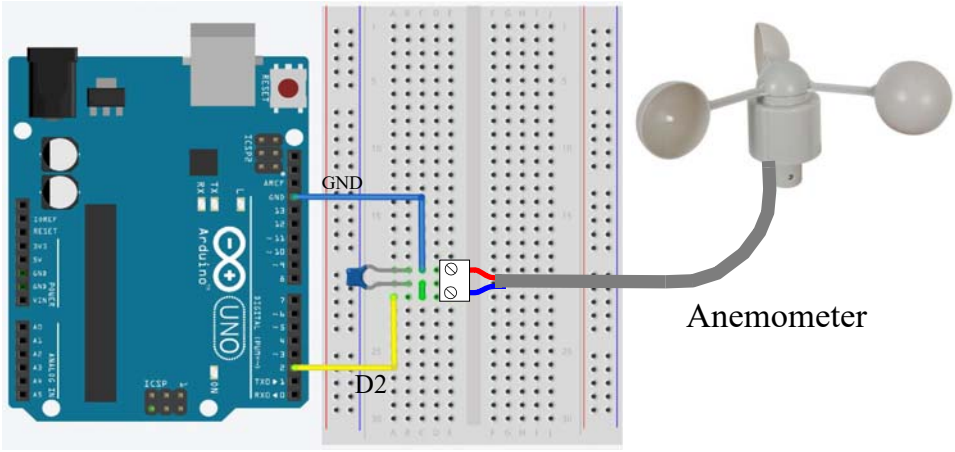
Deloppdrag 7

Monter og skriv et program som måler vindhastigheten i meter pr. sekund. Skriv ut resultatet i monitoren eller OLED-display med tekst og benevning. Legg målingen i en funksjon og returner vindhastigheten.

22.https://cdn.sparkfun.com/assets/8/4/c/d/6/Weather_Sensor_Assembly_Updated.pdf
23.https://cdn.sparkfun.com/assets/8/4/c/d/6/Weather_Sensor_Assembly_Updated.pdf

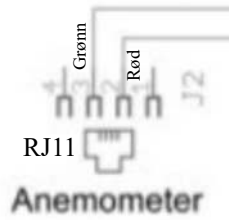
Oppkobling

Oppkoblingen er ganske enkel. Vi legger D2 til 5V med en intern pull up motstand og trekker denne lav (0V) med reed-releet inne i anemometeret, to ganger for hver omdreining. Vi kan ev. legge en liten kondensator mellom de to terminalene for å redusere "prell"[24].



Anemometrene som tilhører SEN-15901 leveres med en modular RJ11 kontakt. Det er ikke uten videre enkelt å montere denne til koblingsbrettet. Dette kan gjøres på flere måter:

1. **Man klipper av ledningen** og kobler de to aktuelle ledningene til en 2-polt rekkeklemme for kretskortmontasje som stikkes ned i brettet. Dette er en ganske brutal metode og bør unngås.
2. **Man benytter en modular kontakt** av hunkjønn med tilhørende ledninger som kobles til rekkeklemmen. Dermed beholder man kontakten på vindstyrke-måleren som senere kan brukes som tiltenkt.



Programmering

Vi skal nå bygge opp programmet trinn for trinn:

1. **Velg interrupt-inngang**
   Vi velger D2 som interrupt-inngang og deklarerer portnummeret som en konstant:

   ```
   const byte pinWindSpeed = 2;
   ```

   Denne kunne like godt vært definert som et heltall. Definisjonen er global og plasseres før void setup()-funksjonen.

24. Prell – Flere tett etterfølgende omslag idet releet slår inn eller ut.

2. **Deklarer variabler** for interrupt rutinen

```
volatile long lastWindIRQ = 0;// Holder siste tidspunkt for interrupt
volatile int windClicks  = 0;// Holder antall pulser
```

Bruk av `volatile` er en måte å sørge for at variabelen får en fast plass i RAM-lageret. Det er vanlig å bruke denne muligheten i forbindelse med interrupt[25]. Deklarasjonene plasseres før void setup()-funksjonen.

3. **Pullup på interrupt-inngang**
Vi ønsker at interrupt-inngangen skal ligge fast høy når den ikke trekkes lav av releet. Dette gjør vi ved å legge inngangen høy med en intern pullup motstand:

```
pinMode(pinWindSpeed, INPUT_PULLUP);
```

Denne kommandoen legges inn i void setup()-funksjonen.

4. **Tilknytt interrupt**
Vi skal nå knytte sammen interruptet med en *Interrupt Service Rutine* (`wspeedIRQ`) som programmet skal oppsøke når interrupt inntreffer på porten D2. Dessuten vil vi definere at interruptet skal iverksettes bare når spenningen på porten *går fra høy til lav spenning* (`FALLING`):

```
attachInterrupt(digitalPinToInterrupt(pinWindSpeed), wspeedIRQ, FALLING);
```

Denne kommandoen legges også inn i void setup()-funksjonen.

5. **Tillat interrupt**
Normalt vil dette skje automatisk når vi starter programmet så denne kommandoen burde ikke være nødvendig. Vi gjør det likevel for at det skal være helt klart at interrupt er slått på:

```
interrupts();
```

Denne kommandoen legges inn i void setup()-funksjonen og slår på alle interrupter. Tilsvarende kan alle interuptene slås av med kommandoen: `nointerrupts();`

6. **Etabler ISR**
Vi skal nå skrive Interrupt Service Rutinen som programmet skal hoppe til når et interrupt inntreffer. Primært vil vi at denne rutinen skal telle opp antall pulser etter som vindmåleren roterer. Vi ønsker imidlertid å hindre at ustabilitet i overgangen fra høy til lav skal resultere i mange tett påfølgende pulser (prell). Vi legger derfor inn en betingelse om at avstanden mellom to pulser skal være større enn 5 ms. Kommer pulsene tettere antar vi at det skyldes prell som vi ikke ønsker å telle med:

```
void wspeedIRQ()
{
  if (millis() - lastWindIRQ > 5) // For å hindre at pulser mindre
                                     enn 5ms blir talt med
  {
    lastWindIRQ = millis();       // Ta vare på nåtidspunktet
    windClicks++;                 // En puls pr. sekund tilsvarer
```

25.https://www.arduino.cc/reference/en/language/variables/variable-scope-qualifiers/volatile/

```
                                        0,667 m/s (2/3m/s).
    }
  }
```

For å sjekke om det er gått tilstrekkelig lang tid mellom pulsene, bruker vi millis() til å ta tiden mellom pulsene. Dette gjør vi ved å huske tidspunktet (`lastWindIRQ`) sist det ble registrert et interrupt. For den som ønsker å repetere bruken av millis() se avsnitt 2.5.6 på side 41.

7. **Lag en funksjon**
   Lag en funksjon som beregner, skriver ut vindhastigheten til monitoren og returnerer vindhastigheten. Kall funksjonen:

   ```
   float avgWindSpeed();
   ```

   Legg funksjonen nederst i programmet eller under en egen fane og kall den opp i void loop().

8. **Test programmet**
   Test programmet og sjekk om det oppfører seg som forventet. Kontroller at den målte vindhastigheten er i overensstemmelse med en kalibrert vindmåler. En enkel måte å lage vind på er å bruke en hårføner. Pass imidlertid på at vinden ikke blir for varm.

Forslag til tilleggsoppgaver

- **Bauforts vindskala**[26]
  Bruk resultatene fra vindmåling i meter pr. sekund og angi vindstyrke med Bauforts betegnelser (stille, flau vind ....). Tips: Lag en tabell og bruk switch case kommandoen. (Se beskrivelsen i oppgave Måling av vindretning avsnitt 6.7 side 95)

6.7 Måling av vindretning med SEN-15901

Læringsmål

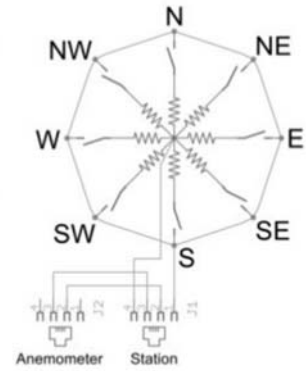
Oppdraget trener følgende læringsmål:

- Virkemåte til en vindretningsmåler
- Spenningsdeleren
- Bruk av array i programmering
- Buk av switch ... case
- Kalibrering

26.https://no.wikipedia.org/wiki/Beauforts_skala

Kort omtale

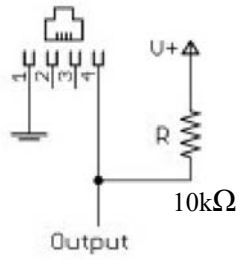
Settet SEN-15901 inneholder bl.a. et instrument som måler vindretningen. Denne krever bl.a. at retningene kalibreres i forhold til nord.



Til høyre på figuren over ser vi et koblingsskjema for vindretningsmåleren. Måleren har åtte reed-releer som kobles inn etter som vindpila dreier. Når pila står i en av de åtte retningene kobler tilhørende rele inn. Når pila står omtrent midt mellom to av de åtte hovedretningene, så kobles releene på hver side inn samtidig. Istedet for å trekke 9 ledninger (en for hver av releene pluss jord) ned til mikrokontrolleren, har man valgt å la hvert av releene koble ulike motstander til jord, slik at en kan nøye seg med to ledninger. Dermed vil motstanden over de to ledningene variere med pilretningen. Dessuten vil to motstander parallellkobles når to releer kobler til samtidig, slik at vi får en ny unik motstandsverdi, lik parallellkoblingen av de to innkoblede motstandene.

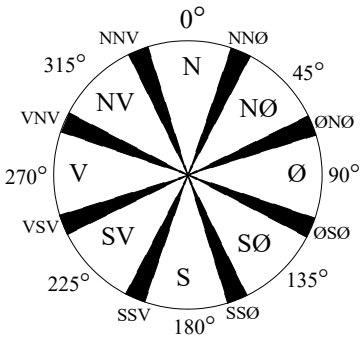
Direction (Degrees)	Resistance (Ohms)	Voltage (V=5v, R=10k)
0	33k	3.84v
22.5	6.57k	1.98v
45	8.2k	2.25v
67.5	891	0.41v
90	1k	0.45v
112.5	688	0.32v
135	2.2k	0.90v
157.5	1.41k	0.62v
180	3.9k	1.40v
202.5	3.14k	1.19v
225	16k	3.08v
247.5	14.12k	2.93v
270	120k	4.62v
292.5	42.12k	4.04v
315	64.9k	4.33v
337.5	21.88k	3.43v

Siden det er vanskelig for en mikrokontroller å måle motstandsverdier, må den gjøres om til en spenning, hvilket vi gjør med en spenningsdeler som vist på figuren til høyre (se også avsnitt 8.1 på side 164).



Av tabellen over ser vi at spenningsområdet (nedre (nV) og øvre spenning (øV)) for hver vinkel, varierer fra 0,04 – 0,68V som vist i tabellen under.:

–	V	+	nV	øV		Vinkel
0,32	0,32	0,09	0,00	0,36	→	112,5
0,09	0,41	0,04	0,37	0,42	→	67,5
0,04	0,45	0,17	0,43	0,50	→	90,0
0,17	0,62	0,28	0,51	0,76	→	157,5
0,28	0,90	0,29	0,77	1,05	→	135,0
0,29	1,19	0,21	1,06	1,29	→	202,5
0,21	1,40	0,48	1,30	1,64	→	180,0
0,48	1,98	0,27	1,65	2,11	→	22,5
0,27	2,25	0,68	2,12	2,59	→	45,0
0,68	2,93	0,15	2,60	3,00	→	247,5
0,15	3,08	0,35	3,01	3,25	→	225,0
0,35	3,43	0,41	3,26	3,63	→	337,5
0,41	3,84	0,20	3,64	3,94	→	0,0
0,20	4,04	0,29	3,95	4,18	→	292,5
0,29	4,33	0,29	4,19	4,48	→	315,0
0,29	4,62	0,38	4,49	5,00	→	270,0



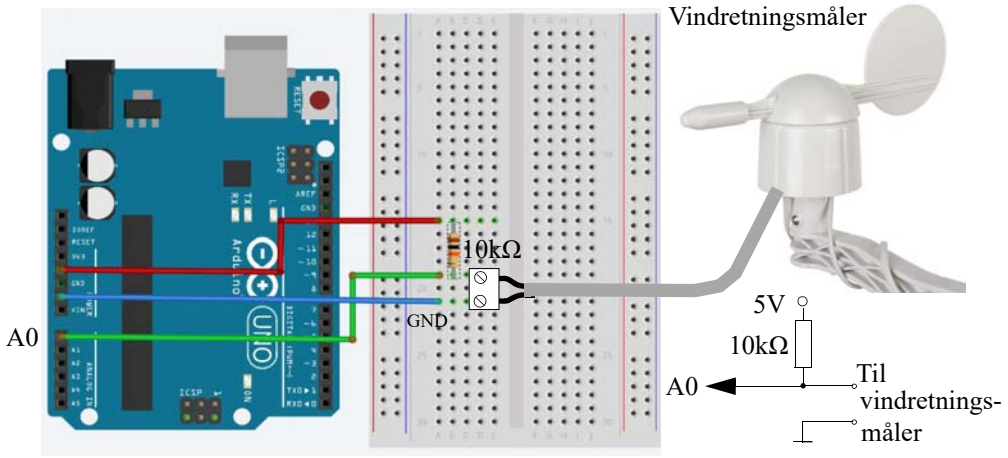
Målinger viser at det er stor forskjell mellom størrelsen til vinkelområdet der det forutsettes at to releer er tilkoblet samtidig og der kun ett rele er tilkoblet. Dette er antydet på figuren over til høyre.

Deloppdrag 8

Koble vindretningsmåleren til mikrokontrolleren og skriv et program som registrerer riktig vindretning. La programmet angi vindretningen både som vinkel 0 – 360° og som navn på en av 16 vindretninger (nord, nord nordøst, nordøst, øst sydøst ... osv.). Skriv resultatet til monitoren eller til OLED-displayet. Legg avlesningen og utskriften i en funksjon og la funksjonen returnere vinkelen i grader.

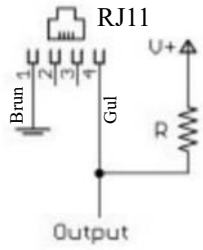
Oppkobling

Oppkoblingen er ganske enkel. Vi lager en spenningsdeler med en 10 kΩ motstand til +5 V, som vist nederst til venstre på figuren under. Midtpunktet til spenningsdeleren går til analog inngang A0.



Vindretningmåleren som tilhører SEN-15901 leveres med en modular RJ11 kontakt. Det er ikke uten videre enkelt å montere denne til koblingsbrettet. Dette kan gjøres på flere måter:

1. **Man klipper av ledningen** og kobler de to aktuelle ledningene til en 2-polt rekkeklemme for kretskortmontasje som stikkes ned i brettet. Dette er en ganske brutal metode og bør unngås.
2. **Man benytter en modular kontakt** av hunkjønn med tilhørende ledninger som kobles til rekkeklemmen. Dermed beholder man kontakten på vind-retningsmåleren som senere kan brukes som tiltenkt.



Programmering

Vi skal nå bygge opp programmet trinn for trinn:

1. **Deklarerer variabler**
   Vi må etablere en sammenheng mellom spenning og vinkel. Selv om vi har oppgitt en forventet spenning ut av spenningsdeleren for hver av de 16 retningene (se tabellen over), så er det en viss usikkerhet, ikke minst fordi supply-spenningen kan avvike noe fra 5V og motstandsverdiene kan avvike noe fra de oppgitte verdiene. Vi velger derfor å sette opp et spenningsområde som definerer hver av retningene. Disse vil være forskjellig fra område til område og forskjellene kan være store. Spenningene varierer heller ikke systematisk med vinkelen (f.eks. stigende spenning med stigende vinkel e.l.). Vi legger verdiene inn i et to-

dimensjonalt array, med en rad pr. retning (16 rader), hver rad med fire kolonner (kol. 0 – er en teller, kol. 1 – er nedre spenningsgrense, kol. 2 – er øvre spenningsgrense, kol. 3 – vinkelen i grader). Arrayet kan deklareres slik:

```
float vindretning[16][4] =
{
  {0, 0.00, 0.36, 112.5},  // Rad 0
  {1, 0.36, 0.42,  67.5},  // Rad 1
  {2, 0.42, 0.50,  90.0},  // Rad 2
  {3, 0.50, 0.76, 157.5},  // Rad 3
  {4, 0.76, 1.05, 135.0},  // Rad 4
  {5, 1.05, 1.29, 202.5},  // Rad 5
  {6, 1.29, 1.64, 180.0},  // Rad 6
  {7, 1.64, 2.11,  22.5},  // Rad 7
  {8, 2.11, 2.59,  45.0},  // Rad 8
  {9, 2.59, 3.00, 247.5},  // Rad 9
  {10, 3.00, 3.25, 225.0}, // Rad 10
  {11, 3.25, 3.63, 337.5}, // Rad 11
  {12, 3.63, 3.94,   0.0}, // Rad 12
  {13, 3.94, 4.18, 292.5}, // Rad 13
  {14, 4.18, 4.48, 315.0}, // Rad 14
  {15, 4.48, 5.00, 270.0}  // Rad 15
};
```

Deklarasjonen av arrayet legges helt først i programmet før void setup()-funksjonen sammen med andre variabler. Dersom du er usikker på bruken av array, se avsnitt 2.5.4 på side 40.

2. **Void setup()-funksjonen**
   Husk å initialiser seriekommunikasjon for skriving til monitoren

   ```
   Serial.begin(9600);
   ```

3. **Spenning fra vindretningsmåleren**
   Vi leser av verdien fra spenningsdeleren. Denne gir et tall mellom 0 – 1023. Siden vi har valgt å operere med spenning, så må vi gjøre om tallet til en spenning. Dette gjør vi ved å multiplisere tallet med 5,00 V og dele på antall kvantiseringer i AD-konverteren, nemlig 1024 (10 bit):

   ```
   vindretningSpenning = 5.00 * analogRead(pinVindretning) / 1024;
   ```

   Resultatet av denne beregningen vil være et desimaltall. Utregningen legges i void loop()-funksjonen eller i en annen egendefinert funksjon.

4. **Oppslag i tabellen (arrayet)**
   Vi skal nå sjekke hvor i tabellen (arrayet) den målte spenningen passer inn, slik at vi kan bestemme vinkelutslaget. Vi bruker da en for-løkke for å bla oss gjennom tabellen. Akkurat når vi er i det riktige spenningsområdet skriver vi ut vinkelen:

   ```
   for (int i = 0; i < 16; i++)
   {
     if (vindretningSpenning >= vindretning[i][1] && vindretningSpenning
     < vindretning[i][2])
     {
   ```

```
    // Skriv ut vindretning i grader
    Serial.print("Vindretning: ");
    Serial.print(vindretning[i][3], 1);
    Serial.print(" grader");
  }
}
```

Legg merke til at vi bruker tellevariabelen *i* fra for-sløyfa til å bla oss gjennom arrayet, rad for rad, og sjekker om den målte spenningen ligger mellom nedre og øvre grenseverdi. Kun der det er tilfelle, får den match med betingelsen til if-setningen og skriver ut vinkelen.

5. **Skriv ut navn på himmelretningene**
   Vi skal nå bruke *`switch(var) ... case n`* kommandoen for å skrive ut navnene på de 16 himmelretningene. *`Switch(var) ... case n`* fungerer slik at verdien til variabelen *`var`* bestemmer til hvilket av tilfellene (case) den skal gå. Er du usikker på *`switch(var) ... case n`* kommandoen så sjekk med avsnitt 2.5.12 på side 47.

```
switch (i)
{
  case 0:  Serial.println(", "); // 112.5
  case 1:  Serial.println(", EAST-SOUTH-EAST"); break;  //  67.5
  case 2:  Serial.println(", EAST"); break;             //  90.0
  case 3:  Serial.println(", SOUTH-SOUTH-EAST"); break; // 157.5
  case 4:  Serial.println(", SOUTH-EAST"); break;       // 135.0
  case 5:  Serial.println(", SOUTH-SOUTH-WEST"); break; // 202.5
  case 6:  Serial.println(", SOUTH"); break;            // 180.0
  case 7:  Serial.println(", NORTH-NORTH-EAST"); break; //  22.5
  case 8:  Serial.println(", NORTH-EAST"); break;       //  45.0
  case 9:  Serial.println(", WEST-SOUTH-WEST"); break;  // 247.5
  case 10: Serial.println(", SOUTH-WEST"); break;       // 225.0
  case 11: Serial.println(", NORTH-NORTH-WEST"); break; // 337.5
  case 12: Serial.println(", NORTH"); break;            //   0.0
  case 13: Serial.println(", WEST-NORTH-WEST"); break;  // 292.5
  case 14: Serial.println(", NORTH-WEST"); break;       // 315.0
  case 15: Serial.println(", West"); break;             // 270.0
}
```

   Dersom vi legger denne `switch(var) ... case n` inn i for-sløyfa og if-setningen vil *i* i variabelen for-sløyfa angi hvilken case som skal skrives ut.

6. **Lag en funksjon**
   Legg alt som har med måling av vindretningen å gjøre inn i egen funksjon, og kall funksjonen fra void loop()-funksjonen. Funksjonen skal returnere vinkelen til vindretningen i grader og skrive vinkel og navn på himmelretning til monitoren. Du står fritt til å velge navn på funksjonen, men her er et forslag:

```
get_wind_direction()
```

7. **Skriv og test**
   Skriv og test ut programmet og se om det fungerer som tiltenkt.

Kalibrering

Vindretningsmåleren kalibreres med et kompass idet den settes opp. Det kan være enklest å sette en strek på vindretningsmåleren der NORD-NORD-ØST er. Denne retningen ligger 22,5° øst for nord og er et relativt smalt område.

Undersøk om det finnes en metode å kalibrere måleren i software.

6.8 Måling av nedbør med SEN-15901

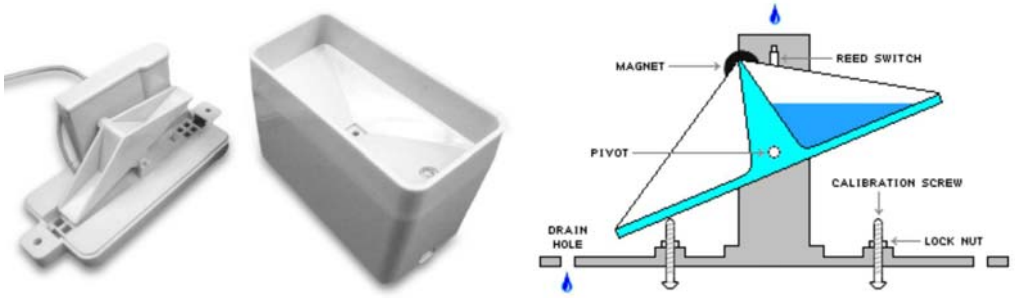
Læringsmål

Oppdraget trener følgende læringsmål:

- Virkemåte til en regnmåler
- Bruk av interrupt
- Omregning fra antall tømminger til nedbør i millimeter
- Kalibrering

Kort omtale

Settet SEN-15901 inneholder bl.a. en selvtømmende nedbørsmåler som gir en puls pr. tømming. Leverandøren oppgir at nedbørsmåleren vil tømmes og gi en puls pr. 0,2794 mm nedbør[27]. Regnmåleren vil derfor ha en oppløsning på 0,2794 mm (ca. 1/4 mm).



Til høyre på figuren over ser vi prinsippet bedre. Vippepunktet er så lavt at når beholderen blir tung nok til at den bikker og tømmes, så vil den forbli i denne posisjonen, mens beholderen på den andre siden fylles opp. En magnet er festet i bakkant av vippen. Idet den vipper vil magneten passere et reedrele plassert i holderen i bakkant og bryteren slutter kretsen.

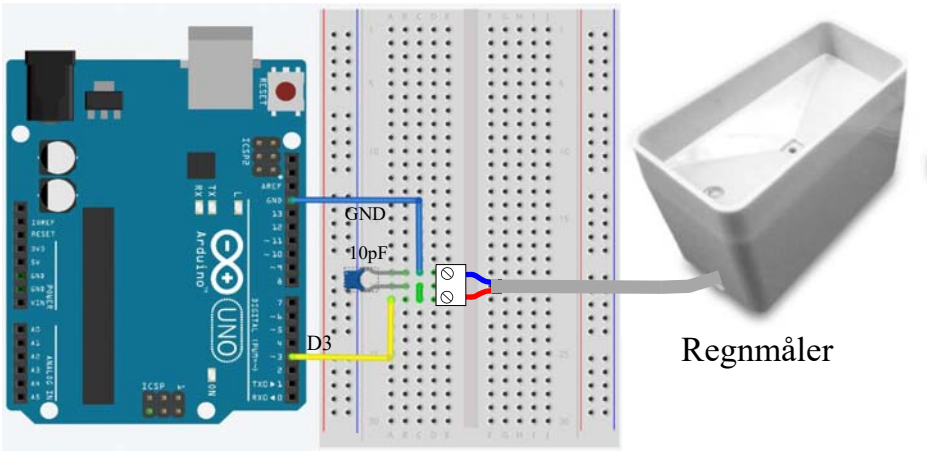
Deloppdrag 9

Koble nedbørsmåleren til mikrokontrolleren og skriv et program som registrerer nedbør i mm. Skriv resultatet til monitoren eller til OLED-displayet. Legg avlesningen og utskriften i en funksjon og la funksjonen returnere nedbør i millimeter. Kalibrer nedbørsmåleren med oppmålt mengde vann.

27.https://cdn.sparkfun.com/assets/8/4/c/d/6/Weather_Sensor_Assembly_Updated.pdf

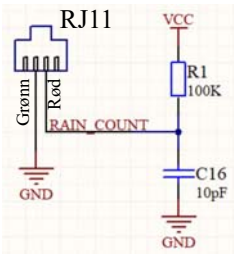
Oppkobling

Oppkoblingen er ganske enkel. Vi legger D3 til 5V med en intern pull up motstand og trekker denne lav (0V) med reed-releet inne i regnmåleren, en gang for hver gang en av beholderne tømmer seg. Vi kan ev. legge en liten kondensator mellom de to terminalene for å redusere "prell".



Regnmåleren som tilhører SEN-15901 leveres med en modular RJ11 kontakt. Det er ikke uten videre enkelt å montere den til koblingsbrettet. Dette kan gjøres på flere måter:

1. **Man klipper av ledningen** og kobler de to aktuelle ledningene til en 2-polt rekkeklemme for kretskortmontasje som stikkes ned i brettet. Dette er en ganske brutal metode og bør unngås.
2. **Man benytter en modular kontakt** av hunkjønn med tilhørende ledninger som kobles til rekkeklemmen. Dermed beholder man kontakten på regnmåleren som senere kan brukes som tiltenkt.



Bruk av interrupt

Klikkene er kortvarige og skal man sjekke tilstanden med jevne mellomrom så går man ganske sikkert glipp av et klikk i ny og ne. Vi bruker derfor hardware interrupt som avbryter programmet uansett hvor vi befinner oss i koden, går til interrupt-rutinen og registrerer klikket. Arduino UNO har to innganger som kan tilknyttes en av interrupt-inngangene D2 (INT0) og D3 (INT1).Vi velger INT1 for regnmåleren.

Ellers anbefales å ta en titt på avsnitt 2.5.14 på side 50 som omhandler bruk av interrupt.

Programmering

Vi skal nå bygge opp programmet trinn for trinn:

1. **Velg interrupt-inngang**
   Vi velger D3 (INT1) som interupt-inngang og deklarerer portnummeret som en konstant:

   ```
   const byte pinRainGauge = 3;
   ```

   Denne kunne like godt vært definert som et heltall. Definisjonen er global og plasseres før void setup().

2. **Deklarer variabler** for interrupt-rutinen

   ```
   volatile long lastRainIRQ = 0;// Holder siste tidspunkt for interrupt
   volatile int rainClicks  = 0;// Holder antall pulser
   ```

   Bruk av `volatile` er en måte å sørge for at variabelen får en fast plass i RAM-lageret. Det er vanlig å bruke denne muligheten i forbindelse med interrupt[28]. Plasseres før void setup()-funksjonen.

3. **Pull up på interruptinngang**
   Vi ønsker at interruptinngangen skal ligge fast høy når den ikke trekkes lav av releet. Dette gjør vi ved å legge inngangen høy med en pullup motstand:

   ```
   pinMode(pinRainGauge, INPUT_PULLUP);
   ```

   Denne kommandoen legges inn i void setup()-funksjonen.

4. **Tilknytt interrupt**
   Vi skal nå knytte sammen interruptet med en Interrupt Service Rutine som vi har valgt å kalle `rainIRQ`. Programmet skal oppsøke denne et interrupt inntreffer på porten D3. Dessuten vil vi definere at interruptet skal iverksettes når spenningen på porten går fra høy til lav spenning (`FALLING`):

   ```
   attachInterrupt(digitalPinToInterrupt(pinRainGauge), rainIRQ, FALLING);
   ```

   Denne kommandoen legges inn i void setup()-funksjonen.

5. **Tillat interrupt**
   Normalt vil dette skje automatisk når vi starter programmet, så denne kommandoen burde ikke være nødvendig. Vi gjør det likevel for å gjøre det helt tydelig at interruptet er slått på:

   ```
   interrupts();
   ```

   Denne kommandoen legges inn i void setup()-funksjonen. På tilsvarene kan vi slå av alle interrupt med denne kommandoen: `nointerrupts();`

6. **Etabler ISR**
   Vi skal nå skrive Interrupt Service Rutinen som programmet skal hoppe til når et interrupt inntreffer. Primært vil vi at denne rutinen skal telle opp antall pulser når regnmåleren tipper. Vi ønsker imidlertid å hindre at ustabilitet i overgangen fra høy til lav skal resultere i mange pulser (prell). Vi legger derfor inn en betingelse om at avstanden mellom to påfølgende pulser (interruft) skal være større enn 5 ms. Kommer pulsene tettere antar vi at det skyldes prell som vi ikke ønsker å telle:

28.https://www.arduino.cc/reference/en/language/variables/variable-scope-qualifiers/volatile/

```
void rainIRQ()
{
  if (millis() - lastRainIRQ > 5) // For å hindre at pulser mindre
                                     enn 5ms blir talt med
  {
    lastRainIRQ = millis();       // Ta vare på nåtidspunktet
    rainClicks++;                 // En puls pr. sekund tilsvarer
                                     0,2794 mm
  }
}
```

For å sjekke om det er gått tilstrekkelig lang tid mellom pulsene bruker vi millis() til å ta tiden. Dette gjør vi ved å huske tidspunktet (`lastRainIRQ`) sist det ble registrert et interrupt. For den som ønsker å repetere bruken av millis() se avsnitt 2.5.6 på side 41.

7. **Lag en funksjon**
   Lag en funksjon som beregner, skriver ut nedbørsmengden til monitoren og returnerer akkumulert nedbørsmengde. Kall funksjonen:

   ```
   float akkRainLevel();
   ```

   Legg funksjonen nederst i programmet eller under en egen fane og kall den opp fra void loop()-funksjonen.

8. **Test programmet**
   Test programmet og sjekk om det oppfører seg som forventet.

9. **Kalibrer målingene**
   Finn en metode for å kontrollere at nedbørsmåleren måler riktig.

6.9 Måling av avstand med HC-SR04 (Snødybde)

Læringsmål

Oppdraget trener følgende læringsmål:

- Virkemåte til en avstandsmåler
- Bruk av kommandoen `delayMicroseconds(); og pulseIn(echoPin, HIGH);`
- Kvaliteten for måling mot ulike flater, f.eks. ulike typer av snø
- Kalibrering

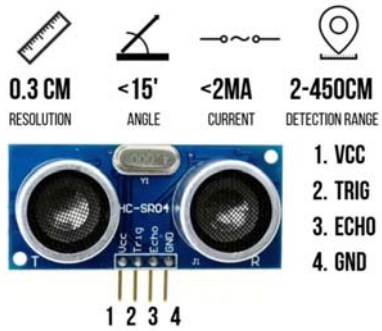
Kort beskrivelse[29]

HC-SR04 er en ultralydsensor som er konstruert for å måle korte avstander. Sensoren sender ut ultralydpulser på 40 kHz, som beveger seg gjennom lufta. Dersom en puls treffer en gjenstand, reflekteres en del av signalet tilbake til sensorens mottaker. Ved å måle tiden fra utsendelse til mottak kan man beregne avstanden til målet dersom man kjenner lydhastigheten.

29.https://create.arduino.cc/projecthub/abdularbi17/ultrasonic-sensor-hc-sr04-with-arduino-tutorial-327ff6

Her er noen sentrale parametere til sensoren:

- Forsyningsspenning: +5 VDC
- Stand by strømtrekk: < 2 mA
- Strømtrekk under måling: 15 mA
- Vinkelen til målet: <15° (dvs. avvik fra 90° på strålegangen)
- Rekkevidde: 2 cm – 400 cm
- Oppløsning: 0.3 cm
- Trigger pulsbredde: 10 µs
- Ultralyd frekvens: 40 kHz
- Dimensjoner: 45 mm x 20 mm x 15 mm

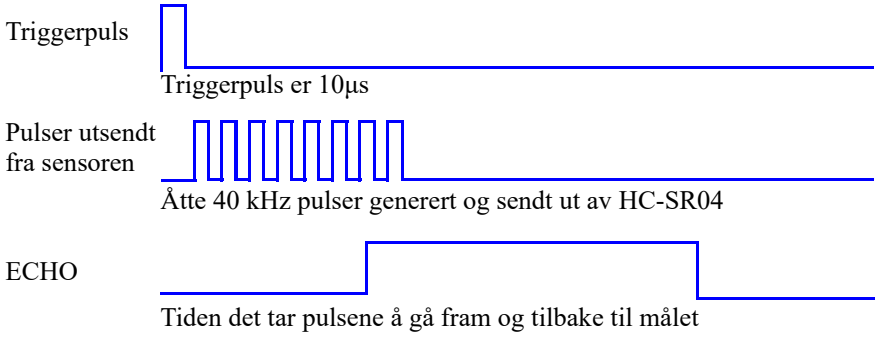


Kretsen har fire terminaler (pinner) med følgende funksjoner:

1. VCC – Forsyningsspenning, 5 V
2. TRIG – Igangsetter utsendelse av nytt pulstog
3. ECHO – Pulslengde som gjengir tidsforsinkelsen
4. GND – Jord

TRIG og ECHO kan kobles til hvilke som helst digitale porter hos Arduino.

For å generere et ultrasonisk pulstog må TRIG-inngangen få en høy puls på minimum 10µs. Dette resulterer i at det sendes ut et pulstog på åtte pulser. Pulstoget beveger seg med lydens hastighet. Ekkoet mottas og ECHO-porten leverer en pulstengde tilsvarende tiden pulsene bruker på å gå fram og tilbake til målet som vist på figuren under.



For en gjenstand som f.eks. befinner seg i en avstand på 20 cm fra sensoren vil ultralydpulsene trenge ca. 588 µsek fra sensoren og fram til gjenstanden, med en lydhastighet på 340 m/s. Siden lyden må gå både fram og tilbake, vil lengden på pulsen vi får fra ECHO utgangen være dobbelt så lang, siden lyden trenger dobbelt så lang tid for å gå fram og tilbake. Vi må derfor dele puls-lengden fra echo-pinnen på 2 for å finne tiden det tar lyd å bevege seg fra sensoren og fram til gjenstanden.

Lydhastighet som funksjon av temperatur

I eksempelet over har vi antatt en lydhastighet på 340 m/s. Imidlertid er lydhastigheten avhengig av temperaturen (og luftfuktigheten). Ligning (6.3)[30] gir lydhastigheten i helt tørr luft (RH = 0%):

$$v_{luft} = (331.3 + 0.606 * T)\ m/s \qquad (6.3)$$

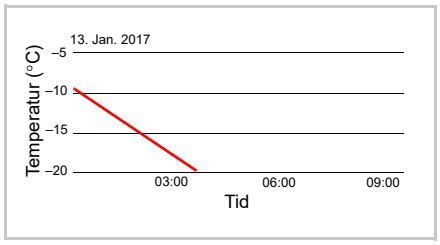
hvor T er lufttemperaturen i °C. Dette gir en hastighet på 331,3 m/s ved 0°C og 346,5 m/s ved 25°C.

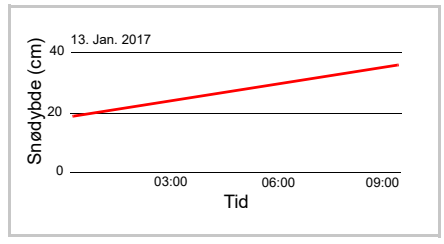
Refleksjon fra snø

Gode avstandsmålinger forutsetter at den reflekterende flaten:

- ... ikke er for skrå i forhold til lydens retning (< 15°)
- ... ikke er for liten i utstrekning
- ... ikke er for langt unna (< 330 cm)
- ... ikke er for myk eller demper ekkoet for mye

Sistnevnte, at objektet er for mykt, kan være et problem når det gjelder snø. Imidlertid kan mye tyde på at det er verdt et forsøk. Følgende målinger er gjort en natt under et snøfall i Bucharest (Romania) på taket av en 10 etasjers bygning[31].



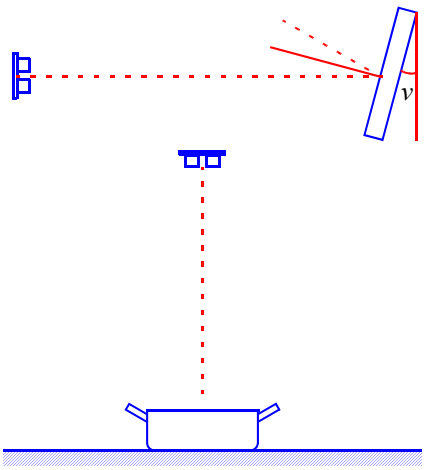


Vi ser at temperaturen faller jevnt og går under skalaen, dessuten snør det også jevnt gjennom natta og ut på morgenen, skal man tro målingene.

30.https://en.wikipedia.org/wiki/Speed_of_sound
31.https://www.geekstips.com/arduino-snow-depth-remote-sensing-with-ultrasonic-sensor/

Deloppdrag 10

Plasser avstandsmåleren slik at den måler horisontalt langs bordet. Sett opp en vertikal hindring mellom 20 og 50 cm fra sensoren. Skriv et program som måler avstanden mellom sensoren og hindringen og skriv resultatet ut i monitoren med riktig benevning. Bruk et målebånd og sjekk nøyaktigheten til målingene. Bruk hindringer av forskjellig materiale og undersøk hvordan dette påvirker målingene. Undersøk også hvordan målingene påvirkes av at hindringen avviker fra en rett vinkel (v) som vist på figuren til høyre.

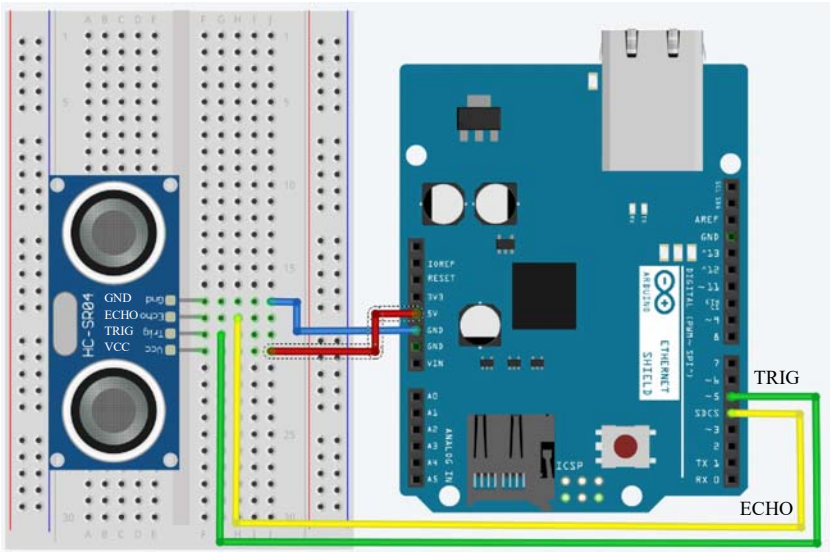


Plasser avstandsmåleren ca. 40 – 50 cm over bordet og undersøk hvordan måleresultatet blir dersom hindringen er vann, våt eller tørr snø.

Beskriv en praktisk løsning for en sensor som skal plasseres utendørs gjennom vinteren.

Oppkobling

Oppkoblingen er ganske enkel. Vi kan fritt velge de digitale portene som er ledige, og vi velger TRIG → D5 og ECHO → D4. HC-SR04 er dessuten designet for en supply-spenning på 5 V.



Programmering

Programmet er ganske enkelt.

1. **Deklarer variabler**
   Her står man fritt, men det er naturlig å deklarere en variabel som holder pulslengden som indikerer tiden fra trigger til mottatt ekko, og beregnet avstand:

```
long duration; // Varighet til utsendt til mottatt pulstog
int  distance; // Målt distanse
```

   Disse plasseres foran void setup()-funksjonen.

2. **Sender TRIG-puls**
   Følgende kommandoer klargjør og sender pulser med lengde 10 µs.

```
digitalWrite(trigPin, LOW); // Klargjør Trig-pin ved å sette utg. lav
delayMicroseconds(2);
digitalWrite(trigPin, HIGH);// Sett Trig-pin høy i 10 microsek.
delayMicroseconds(10);
digitalWrite(trigPin, LOW); // Sett Trig-pin lav etter 10 microsek.
```

   Disse kommandoene plasseres i void loop().

3. **Måling av pulslengde**
   Pulslengden måles med funksjonen `pulseIn(echoPin, HIGH);` Hvor `echoPin` er nummeret til porten som mottar pulsen, og HIGH/LOW betyr at den måler en puls som enten går høy eller en som går lav.

```
duration = pulseIn(echoPin, HIGH); // Leser pulslengde fra Echo-pin,
  returnerer "reise"-tiden til pulstoget i microsekunder
```

   Plasseres enten i void loop()-funksjonen eller i en egendefinert funksjon.

4. **Beregning av avstanden**
   Når vi kjenner lydhastigheten så er det lett å beregne avstanden til gjenstanden ekkoet reflekteres fra:

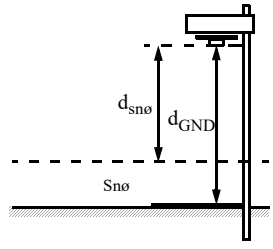
```
distance = duration * 0.034/2;   // Beregner avstanden
```

   Plasseres enten i void loop()-funksjonen eller i en egendefinert funksjon.

5. **Lag en funksjon**
   Legg måling og beregning av avstand i en egen funksjon. F.eks. kan du kalle funksjonen `findDistance()`. Funksjonen skal returnere den beregnede avstanden i cm.

Kalibrering

Avstandsmåleren kan kalibreres med å montere den i en gitt avstand fra en plan vegg eller plate. Siden den på sikt skal kunne måle snødybde så kan en tenke seg en oppstilling som vist på figuren til høyre. Man monterer avstandsmåleren på en fot som presses ned i bakken. Avstanden mellom sensoren og bakken måles med et målebånd (d_{GND}). Dersom det er avvik mellom avstanden målt med målebåndet og sensoren, kan man legge en plate på bakken slik at refleksen blir tydeligere. Er det fortsatt et systematisk avvik, kan man korrigere resultatet i programmet.



Snødybden framkommer som:

$$\text{Snødybde [cm]} = d_{GND} - d_{snø} \tag{6.4}$$

Kontrollmålinger med målebånd etter at snøen er falt kan være lurt.

6.10 Beregning av vindavkjølingsindeksen eller følt temperatur

Læringsmål

Oppdraget trener følgende læringsmål:

- Forståelse av hva vindavkjølingsindeksen er
- Matematiske beregning som del av programmer
- Omregning fra m/s til km/time
- Funksjoner med argumenter og retur av beregnede verdier
- Sammenstilling av måling av vindhastighet og temperatur

Kort beskrivelse[32]

Det er etter hvert blitt ganske vanlig å oppgi *vindavkjølingsindeksen* eller *følt temperatur* sammen med *målt temperatur.* Dette skyldes at nedkjølingen av bar hud skjer mye raskere dersom det i tillegg til lav temperatur er vind. Vinden vil gjøre at luften nær huden stadig blir byttet ut med ny kald luft og på den måten rekker ikke huden å varme opp luftlaget nær huden og den avkjøles raskere. Dermed føles det kaldere enn man måler samtidig som faren for forfrysning øker.

På siden til yr.no finner vi følgende definisjon for vindavkjølingsindeksen:

Vindavkjølingsindeksen

> *Den **følte temperaturen** eller **vindavkjølingsindeksen** er den temperaturen vi måtte hatt i vindstille forhold for å oppleve samme kulde som ved de eksisterende vind- og temperaturforhold.*

32.https://hjelp.yr.no/hc/no/articles/360001695513-Effektiv-temperatur-og-f%C3%B8les-som-

Som nevnt over er den viktigste grunnen til at man oppgir følt temperatur at faren for forfrysning øker i betydelig grad med økende vindstyrke. Dette kommer tydelig fram i tabellen under. Av tabellen ser vi bl.a. at en temperatur på –25°C blir til en følt temperatur på –35°C ved lett bris (4,5 m/s). Derimot må man helt opp i liten storm (22,5 m/s) for at –25°C skal føles som –45°C. Med andre ord, det er de første vindpustene som gjør mest av seg.

		Lufttemperatur											
		5°	0°	-5°	-10°	-15°	-20°	-25°	-30°	-35°	-40°	-45°	-50°
Vindstyrke (m/s)		Indeks											
Svak vind	1,5	4	-2	-7	-13	-19	-24	-30	-36	-41	-47	-53	-58
	3	3	-3	-9	-15	-21	-27	-33	-39	-45	-51	-57	-63
Lett bris	4,5	2	-4	-11	-17	-23	-29	-35	-41	-48	-54	-60	-66
	6	1	-5	-12	-18	-24	-31	-37	-43	-49	-56	-62	-68
Laber bris	7,5	1	-6	-12	-19	-25	-32	-38	-45	-51	-57	-64	-70
	9	0	-7	-13	-20	-26	-33	-39	-46	-52	-59	-65	-72
Frisk bris	10,5	0	-7	-14	-20	-27	-33	-40	-47	-53	-60	-66	-73
Liten kuling	12	-1	-7	-14	-21	-27	-34	-41	-48	-54	-61	-68	-74
	13,5	-1	-8	-15	-21	-28	-35	-42	-48	-55	-62	-69	-75
Stiv kuling	15	-1	-8	-15	-22	-29	-35	-42	-49	-56	-63	-70	-76
	16,5	-2	-9	-15	-22	-29	-36	-43	-50	-57	-63	-70	-77
Sterk kuling	18	-2	-9	-16	-23	-30	-37	-43	-50	-57	-64	-71	-78
	19,5	-2	-9	-16	-23	-30	-37	-44	-51	-58	-65	-72	-79
	21	-2	-9	-16	-23	-30	-37	-44	-51	-59	-66	-73	-80
Liten storm	22,5	-3	-10	-17	-24	-31	-38	-45	-52	-59	-66	-73	-80
	24	-3	-10	-17	-24	-31	-38	-45	-52	-60	-67	-74	-81

Som forventet vil hudens evne til å motstå forfrysning svekkes betydelig med økende vindstyrke, det viser tabellen under. Vi legger merke til at tiden det tar for å forfryse huden ved –30°C, reduseres fra 15 til 5 minutter når vindstyrken økes fra 2,5 m/s (svak vind) til 15 m/s (stiv kuling). Vi legger også merke til at sannsynligheten for forfrysninger regnes som lite sannsynlig selv ved –20°C til tross for en vind på 5,0 m/s.

Så lang tid tar det å forfryse i minutter

	Temperatur							
Vind m/s	-15°	-20°	-25°	-30°	-35°	-40°	-45°	-50°
2,5 m/s	*	*	22	15	11	8	7	6
5,0 m/s	*	*	14	10	7	6	5	4
7,5 m/s	*	18	11	8	6	4	4	3
10,0 m/s	42	14	9	6	5	4	3	2
12,5 m/s	27	12	8	5	4	3	2	2
15,0 m/s	22	10	7	5	3	3	2	2
17,5 m/s	18	9	6	4	3	2	2	2
20,0 m/s	16	8	5	4	3	2	2	1

* = usannsynlig med forfrysninger.

Beregning av vindavkjølingsindeksen

I 2001 vedtok National Weather Service å bruke en ny metode for beregning av vindavkjølingsindeksen, som bruker en modell som viser hudens temperatur under varierende vindhastigheter og temperaturer. Resultatene av denne modellen beregnes ved hjelp av denne denne formelen[33]. Nøyaktigheten er oppgitt til å være en grad:

$$W = 13{,}12 + 0{,}6215 \cdot T - 11{,}37 \cdot V^{0{,}16} + 0{,}3965 \cdot T \cdot V^{0{,}16} \quad (6.5)$$

Hvor:

W = Følt temperatur i °C
T = Temperatur i °C målt i 2 meters høyde over bakken
V = Vindstyrke i km/timen målt i 10 meters høyde over bakken

Det er imidlertid viktig å merke seg at formelen har et gyldighetsområde fra –105 til +5°C og en vindhastighet fra 2 til 41 m/s (≈7 til ≈148km/h). Utenfor dette området vil ikke formelen gi tilstrekkelig nøyaktige verdier. Dette er det viktig å legge inn i omregningen i programmet.

Omregning fra m/s til km/h gjørs slik: $v_{km/h} = 3{,}6\ v_{m/s}$

Deloppdrag 11

Monter temperaturmåler og vindmåler i samme oppsett og skriv et program som måler både vindhastighet og temperatur. Skriv ut resultatet av målingene i monitoren eller på displayet med riktig benevning.

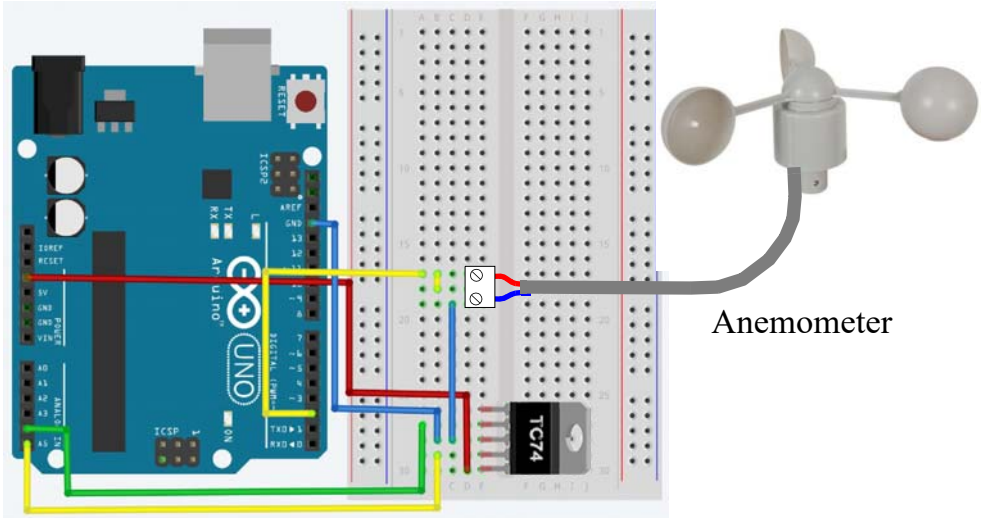
Regn vindhastigheten om fra m/s til km/timen og sett både temperaturmålingen og vindhastigheten inn i formelen og beregn følt temperatur. Skriv ut resultatet i monitoren eller på displayet. Kontroller at følt temperatur endres med vindhastigheten.

Legg beregningen av følt temperatur i en egen funksjon med temperatur og vindhastighet som argumenter ved kallet og med vindavkjølingsindeksen eller følt temperatur som retur.

33.https://no.wikipedia.org/wiki/Vindavkj%C3%B8lingseffekt

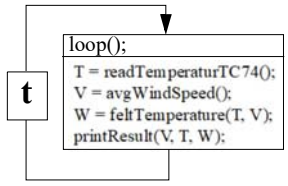
Oppkobling

I denne oppkoblingen kombinerer vi temperaturmåling med TC74 (avsnitt 6.3.1 side 76) og vindstyrkemåleren SEN-15901 (avsnitt 6.6 side 91). I testoppkoblingen er det unødvendig med kondensator.



Programmering

For å lage dette programmet vil vi oppmuntre til å bruke funksjoner hvor vi overfører måleverdiene beregnet i én funksjon gjennom argumentet til en annen. For bedre å få oversikt over programstrukturen har vi laget et flytdiagram over programmet. Vi lager oss følgende funksjoner:



void loop() kaller følgende funksjoner:

```
T =  readTemperaturTC74();  (Leser av temperatur)
V =  avgWindSpeed();        (Leser av vindhastighet)
W = feltTemperature(T, V);  (Beregner følt temperatur)
printResult(V, T, W);       (Skriver ut resultatet)
```

De to første returnerer målte verdier for temperatur (T) og vindhastighet (V). Den tredje overfører disse (T, V) for å beregne og returnere følt temperatur, W. Til sist overføres målt temperatur (T), vind (V) og følt temperatur (W) for utskrift ved hjelp av utskriftsfunksjonen `printResult()`.

Siden vind ikke kan leses av som en øyeblikksverdi, men må måles over tid (f.eks. t = 10 sek.), så legger vi inn en tidsforsinkelse i loopen.

wspeedIRQ();
Teller to vindklikk for hver runde.

T = readTemperaturTC74();
Leser av og returnerer temperatur

V = avgWindSpeed();
Leser av og returnerer vindhastighet

W = feltTemperatur(V,T);
Leser av og returnerer vindhastighet

printResult(V, T, W);
Mottar og skriver ut vind, temp. og følt temp.

For hver runde vindmåleren går så vil en kontakt sluttes to ganger og det registreres to "vind-klikk". Siden registrering av vindklikk ikke kan utsettes, bruker vi interrupt og programmet hopper til interrupt request funksjonen (wspeedIRQ) som øker en teller for hvert klikk.

Vi skal nå bygge opp programmet trinn for trinn, men henvise til avsnitt 6.3.1 på side 76 for måling av temperatur med TC74 og avsnitt 6.6 på side 91 for måling av vindhastighet med SEN-15901.

1. **Inkluder bibliotek og adresse for temp. sensor**
   Vi henter ut det biblioteket vi trenger for å lese av temperaturen fra TC74 via I^{2}C-bussen og deklarerer en konstant som holder adressen til TC74:

```
#include "Wire.h"        // Inkluder I2C-biblioteket
#define address B1001000 // Adressen til temperaturesensoren
```

   Disse legges først i programmet.

2. **Inkluder globale variabler for vindsensoren**
   Vi henter inn og deklarerer variablene vi trenger for å måle vindhastigheten

```
const byte pinWindSpeed = 2;   // Port for registrering av vindhastighet
volatile long lastWindIRQ = 0; // Holder siste tidspunkt for interrupt
volatile int windClicks = 0;   // Holder antall pulser
float windspd_avg_time = 10; // Midlingstid i sek. for beregning av vindhast.
float current_time = 0;        // Løpende tid for beregning av vindhastighet
float windspd_pr_click = 0.667;// Vindhastighet pr. klikk og sekund
```

   Vi kobler vindmåleren opp til port D2 som kan betjene interrupt. For å unngå å få flere klikk for hvert tilslag av reedrelet til vindmåleren, så måtte vi sørge for at vi neglisjerte multiple omslag som kom tett etter hverandre, derfor måtte vi huske sist vi fikk et tilslag (`lastWindIRQ`) og sjekke om det var tilstrekkelig lenge siden, mer enn 5 sek. for at vi skal regne klikket som et nytt unikt klikk. Disse deklarasjonene settes før void setup().

3. **Deklarasjon av nye variabler**
   Vi deklarere noen nye globale variabler som vi trenger for beregningene våre.

```
float  T = 0;          // Avlest temperatur
float  V = 0;          // Beregnet midlere vindhastighet i m/s
double W = 0;          // Beregnet følt temperatur
float minV =    2.0;  // m/s -   7 km/h - Laveste lovlige verdi
float maxV =   41.0;  // m/s - 147 km/h - Høyeste lovlige verdi
float minT = -105.0;  // C - Laveste lovlige verdi
float maxT =    5.0;  // C - Høyeste lovlige verdi
```

   T, V og W er de verdiene vi er interessert i og vil skrive ut. Som vi har sett har formelen for beregnet "Følt temperatur" et gyldighetsområde, som vi her har definert med sine maksimal- og minimalverdier. Når disse overskrides så kan vi ikke beregne "Følt temperatur". Disse deklarasjonene plasseres foran void setup().

4. **void setup()-funksjonen**
   Her initialiserer vi lesing via I^{2}C-bussen og definerer D2 (`pinWindSpeed`) som en input med pullup-motstand, dvs. at inngangen ligger høy så lenge releet i vindmåleren ikke kobler inn.

```
pinMode(pinWindSpeed, INPUT_PULLUP);
Wire.begin();
```

Dessuten må vi knytte D2 til et interrupt (`digitalPinToInterrupt(pinWindSpeed)`) og en funksjon som skal utføres når interruptet inntreffer (`wspeedIRQ`) og fallende eller stigende signalflanke (`FALLING`). Til slutt må vi aktivere interruptet:

```
attachInterrupt(digitalPinToInterrupt(pinWindSpeed), wspeedIRQ, FALLING);
interrupts(); // Slå på interrupts
```

5. **void loop()-funksjonen**
   Her organiserer vi programmet vårt og kaller de ulike funksjonene i riktig rekkefølge. Legg spesielt merke til hvordan funksjonen `readTemperaturTC74()` returnerer T (temperatur) og `avgWindSpeed()` returnerer V (vindstyrke), mens verdiene "sendes" til funksjonen `feltTemperature(T,V)` som underlag for beregning av følt temperatur, som igjen returnerer den beregnede verdien, W, for følt temperatur. Til slutt sendes verdiene V, T og W til funksjonen `printResult(V,T,W)` for utskrift:

```
if (millis() - current_time > 1000 * windspd_avg_time)
{
  T = readTemperaturTC74(); // Mål og skriv ut temperatur
  V = avgWindSpeed();       // Beregn og skriv ut midlere vindhastighet
  W = feltTemperature(T, V);// Beregn følt temperatur, W
  printResult(V, T, W);     // Skriv ut midlere vindhastighet
  current_time = millis();  // Nullstill løpende tid
}
```

   Den omsluttende if-setningen sjekker om tiden vindmålingen skal midles over er gått, i så fall kan målingene utføres og beregnes og skrives ut med funksjonen `printResult(V,T,W)`.

   For oppbygging av egne funksjoner se avsnitt 2.5.13 på side 48, for bruk av millis() se avsnitt 2.5.6 på side 41 og for bruk av interrupt se avsnitt 2.5.14 på side 50.

   I de neste punktene skal vi bygge opp de ulike funksjonene. Her vil vi henvise til tidligere oppdrag.

6. **void wspeedIRQ()**
   Dette er funksjonen som teller antall vindklikk fra reed-releet hver gang det slår inn. For nærmere beskrivelse se avsnitt 6.6 på side 91.

7. **float readTemperaturTC74()**
   Dette er funksjonen som leser av temperaturen fra TC74 og returnerer den fra funksjonen. For nærmere beskrivelse se avsnitt 6.3.1 på side 76. Vi har valgt å definere en lokal variabel for den avleste temperaturen.

8. **float avgWindSpeed()**
   Dette er funksjonen som beregner vindstyrken ved å bruke antall vindklikk som er samlet inn i løpet av tiden målingen pågår (windspd_avg_time). Legg merke til at funksjonen brukt her kan forenkles i forhold til den som ble brukt i avsnitt 6.6 på side 91 siden tidsrommet for midlingen er lagt inn i void loop()-funksjonen.

9. **`double feltTemperature(double temp, double vind_mps)`**
   Denne funksjonen skal beregne "følt temperatur" og trenger da å få overført vindhastighet i m/sek (vind_mps) og temperatur (temp). Siden formelen trenger vindhastighet i km/h (vind_kmph) så må vi regne om fra m/s til km/h:

```
double feltTemperature(double temp, double vind_mps)
{
double vind_kmph = 3.6 * vind_mps; // Omregning fra m/s til km/h
double feltTemp = 13.12 + 0.6215 * temp - 11.37 * pow(vind_kmph, 0.16) +
                  0.3965 * temp * pow(vind_kmph, 0.16);
return feltTemp;
}
```

   Vi legger merke til $vind\_kmph^{0.16}$ skrives: `pow(vind_kmph, 0.16)`[34]. Se avsnitt 2.5.7 på side 42 for en liten oversikt over matematiske operatorer. Ev. se https://www.arduino.cc/reference/en/

10. **void printResult(float vind, float temp, double feltTemp)**
    Denne funksjonen skriver ut vindhastighet i m/s, temperatur i C og følt temperatur i C.
    I tillegg sjekkes om vindhastighet og temperatur er innen de tillatte verdiene slik at følt temperatur beregnes når formelen er gyldig. Denne siste delen kan utformes som en if-setning:

```
Serial.print(", Felt temp.: ");
if (vind >= minV && vind <= maxV && temp >= minT && temp <= maxT)
{
  Serial.print(feltTemp, 1); Serial.println("°C");
}
else
{
  Serial.println("OL - Vind < 2m/s or Temp > 5°C");
}
```

    OL i siste linje står for *Out of Limit*. Vi ser at dersom vind og temperatur er innen grenseverdiene så skrives den beregnede verdien ut, ellers påpekes at målingene er utfor gyldighetsområdet, OL.

11. **Skriv og test programmet**
    Skriv programmet og test om det fungerer etter hensikten.
    Dersom man befinner seg innendørs kan det være vanskelig å oppnå temperaturer under 2°C. Da kan man snu fortegnet på den avleste temperauren slik at man får temperaturer på under –20°C. Ved å bruke en hårføner eller blåse på anemometeret burde man komme innenfor gyldighetsområdet til formelen.

34.https://www.arduino.cc/reference/en/language/functions/math/pow/

6.11 Beregning av varmeindeks

Læringsmål

Oppdraget trener følgende læringsmål:

- Forstå hva varmeindeks er
- Avlesning og/eller beregning av varmeindeks ved hjelp av DHT11

Kort omtale[35]

For at vi skal ha det rimelig behagelig når det er varmt, så må kroppen være istand til å kvitte seg med varmen. Det gjør den vanligvis ved å skille ut svette slik at huden blir dekket av fuktighet som fordamper. For at svetten skal gå over til damp kreves varme som tas fra kroppen slik at den avkjøles.

Fordampning skjer lettere når lufta er tørr enn når den er fuktig. Varm fuktig luft vil derfor oppleves mindre behagelig siden kroppen har større problemer å kvitte seg med overskuddsvarmen. Etter som lufttemperaturen nærmer seg kroppstemperaturen, vil vi føle at det er varmere enn hva termometeret viser. Dersom lufta blir varmere enn kroppstemperaturen, vil man i verste fall oppleve at den fuktige lufta kondenserer på huden og dermed avgir varme og dermed gjør vondt verre. Varmeindeksen angir derfor *følt temperatur på varme dager* og er derfor en temperatur som gjerne ligger over den målte temperaturen.

> ***Varmeindeksen* angir derfor *følt temperatur* på varme og fuktige dager, på samme måte som *vindavkjølingsindeksen* angir *følt temperatur* på svært kalde og vindfulle dager.**

Tabellen til høyre viser hvordan den følte temperaturen øker med både målt temperatur og luftfuktighet.

Vi legger merke til at ved en temperatur på 30°C og en relativ fuktighet på 40%, så er følt temperatur lik målt temperatur. Dersom luftfuktigheten stiger, vil den følte temperaturen også øke. Blir derimot lufta tørrere en 40% relativ fuktighet så vil temperaturen oppleves kaldere enn den målte temperaturen.

Temp. C	20	22	24	26	28	30	32	34	36	38	40	42	44
Fugtighed %													
0	16	18	20	22	24	26	27	29	31	33	35	37	38
5	18	19	21	23	24	26	28	30	32	34	36	38	40
10	19	20	22	23	25	27	29	30	33	35	37	39	42
15	20	21	23	24	25	27	29	31	33	36	38	41	44
20	21	22	23	24	26	28	30	32	34	37	40	43	46
25	22	23	24	25	27	28	30	33	36	39	42	45	49
30	23	23	24	25	27	29	31	34	37	40	44	48	52
35	23	24	24	26	27	29	32	35	38	42	46	51	56
40	24	24	25	26	28	30	33	36	40	44	49	54	60
45	24	24	25	26	28	31	34	37	42	46	52	58	64
50	24	24	25	27	29	31	35	39	44	49	55	61	
55	24	25	25	27	29	32	36	41	46	52	58	66	
60	25	25	25	27	30	33	37	42	48	55	62		
65	25	25	26	27	30	34	39	44	50	58	66		
70	24	25	26	28	31	35	40	46	53	61			
75	24	25	26	28	32	36	42	48	56	65			
80	24	25	26	29	32	37	42	51	59	68			
85	24	25	26	29	33	39	45	53	62				
90	24	25	26	30	34	40	47	56	66				
95	24	25	27	30	35	42	49	59	69				
100	24	25	27	31	36	43	52	62					

Varmeindeks	Påvirkning af kroppen ved vedvarende påvirkning
32°C til 40°C	Minimal risiko, men man skal passe på ved længere ophold
40°C til 54°C	Tiltagende risiko for hedeslag
Over 54°C	Meget stor fare for hedeslag

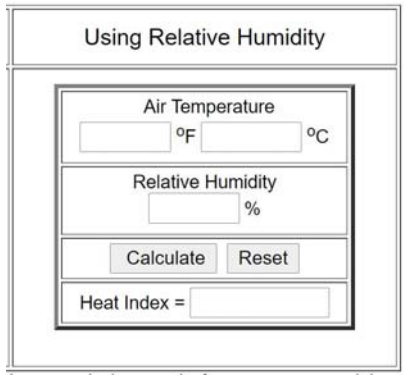
35.https://snl.no/varmeindeks

Beregning av varmeindeksen

Beregning av varmeindeksen kan gjøres ved hjelp av en temmelig komplisert formel[36] som inneholder flere ledd med både relativ fuktighet og målt temperatur og en mengde konstanter.

$$HIC = c_1 + c_2T + c_3R + c_4TR + c_5T^2 + c_6R^2 + c_7T^2R + c_8TR^2 + c_9T^2R^2 \qquad (6.6)$$

Hvor:
HIC = varmeindeks i °C
T = temperatur i °C
R = relativ fuktighet i %
c_1= –8.78469475556
c_2= 1.61139411
c_3= 2.33854883889
c_4= –0.14611605
c_5= –0.012308094
c_6= –0.0164248277778
c_7= 0.002211732
c_8= 0.00072546
c_9= –0.000003582



Alternativt kan man benytte en nettbasert kalkulator hvor man kun slår inn temperatur og relativ fuktighet som vist over til høyre[37].

Finn varmeindeksen med DHT11

Som vi oppdaget da vi brukte DHT11 så har denne komponenten en innebygget "kalkulator" som beregner varmeindeksen på bakgrunn av målt temperatur og relativ fuktighet. Så om man ikke ønsker å gjøre beregningen selv etter den foreslåtte formelen, så kan man ta den rett ut fra DHT11, enten i °Fahrnheit eller i °Celcius.

Deloppdrag 12

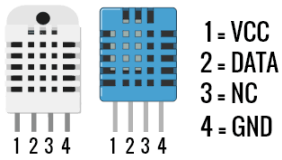
> *Koble opp DHT11 og lag et program som skriver ut temperatur, relativ fuktighet og varmeideksen i °C. Varm opp sensoren med en hårføner eller lignende og undersøk om den beregnede indeksen stemmer med forventet verdi fra tabeller som viser varmeindeksen.*

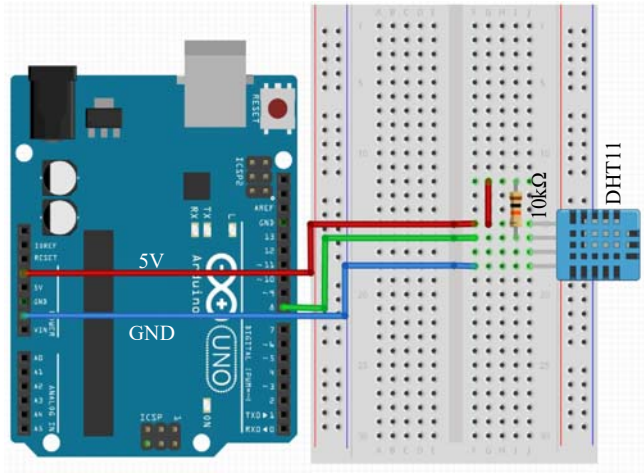
36.https://en.wikipedia.org/wiki/Heat_index
37.https://www.wpc.ncep.noaa.gov/html/heatindex.shtml

Oppkobling:

Oppkoblingen er særdeles enkel. En må imidlertid sørge for å koble en ekstern motstand på 5 – 10 kΩ mellom datautgangen og pluss-spenningen som vist på figuren til høyre.

I denne sammenhengen velger vi å bruke DHT11.





Programmering

Ta utgangspunkt i programmet skissert i avsnitt 6.4.1 på side 79 og føy til utskrift av varmeindeksen.

1. **Lesning av varmeindeks i °C**
   Bruk følgende kommando for å lese ut varmeindeksen:

   ```
   float hic = dht.computeHeatIndex(t, h, false);
   ```

   Som vi ser så tilføres funksjonen `dht.computeHeatIndex` målt temperatur (t) og relativ fuktighet (h) via argumentet, false angir at beregningen skal returneres i °C. For flere detaljer se avsnitt 6.4.1 på side 79.

2. **Skriv og test programmet**
   Test om varmeindeksen stemmer med tabellen over. Varm opp sensoren med en hårføner og se om varmeindeksen fortsatt stemmer.

7 Oppgaver – Oppkobling til nett og styring av aktuatorer

I denne delen skal vi se hvordan vi kan overføre sensordata via nett og vise resultatene på nettstedet Circus of Things (CoT). Videre skal vi vise hvordan vi fra det samme nettstedet kan styre aktuatorer.

De følgende oppgavene vil ha et visst fokus på ventilasjon i et lokale der det er mer naturlig å anvende og fjernstyre aktuatorer av typen rele og viftemotorer. Etter hvert vil vi komme tilbake til værstasjonen.

7.1 Oppkobling av sensor som måler luftfuktighet og temperatur (DHT11)

Læringsmål

Oppdraget trener følgende læringsmål:

- Oppkobling og bruk av sensoren DHT11 (temperatur og luftfuktighet), repetisjon

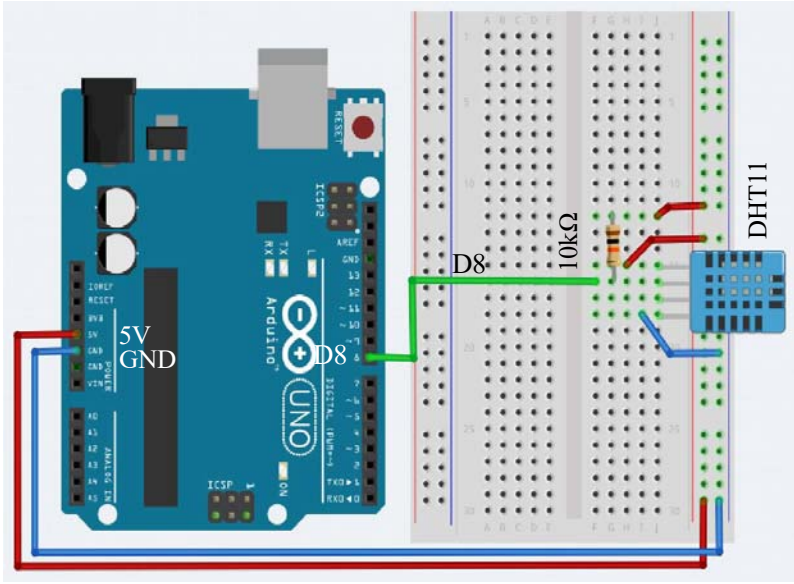
Som introduksjon til oppkobling til Internett og Circus of Things, velger vi å ta utgangspunkt i DHT11 som måler luftfuktighet og temperatur som vi gjorde i oppdrag 6.4.1 side 79.

Deloppdrag 13

Bygg opp og lag et program som leser temperatur og luftfuktighet fra sensoren DHT11. Skriv ut verdiene i monitoren.

Oppkobling

Vi kobler først opp kretsen slik vi gjorde tidligere, se figuren under. Signalet fra DHT11 kobles til port D8.



Programmer

Skriv inn programmet som vi gjorde i oppdrag 6.4.1 og test at det fungerer som forventet.

Vi skal nå ta utgangspunkt i denne kretsen og supplere oppkoblingen med ESP01 slik at vi kan koble oss opp til en lokal router og etter hvert til nettstedet Circus of Things for å kunne legge dataene ut på nettet.

7.2 ESP01 – opprettelse av kontakt mellom sensornoden og CoT

Læringsmål

Oppdraget trener følgende læringsmål:

- Oppkobling og bruk av ESP01 for tilknytning til lokal router via Wi-Fi
- Opprette bruker ved Circus of Things (CoT)
- Etablere signaler, lage og bruke paneler og konsoller i CoT
- Opprette bruker og koble opp til serveren ved CoT
- Skrive program for å overføre sensordata til CoT

7.2.1 Valg av teknologi og nettsted for styring og overvåking

Internet of Things (IoT) handler om å bygge opp og programmere små sensornoder tilknyttet gjenstander av ulike slag, og som kommuniserer trådløst over Internett via lokale Wi-Fi-nettverk. Det er derfor et viktig poeng å kunne koble våre kretser opp mot et lokalt nettverk. Dette kan gjøres på et utall forskjellige måter. En vanlig måte er å benytte et mikrokontroller-kort med innebygget Wi-Fi som kommuniserer via en lokal router på 2,4 GHz. Dernest må man bygge opp et panel som viser måleverdier og som gir mulighet til å fjernstyre aktuatorer. Dette kan man bygge opp fra bunnen eller bruke nettleverandører som tilbyr verktøy for oppbygging av slike brukergrensesnitt. Her finnes det mange å velge mellom. Blynk, ThingSpeak og Circus of Things er tre eksempler av et større utvalg. I disse oppdragene skal vi benytte Circus of Things, eller bare CoT. De viktigste årsakene til dette valget er:

- Tjenesten er gratis
- Den er enkel å ta i bruk
- Vi har god kontakt med designeren
- Tjenesten tilbyr bruk av ulike teknologiske plattformer (bl.a. ESP32, ESP01, Raspberry Pi m.fl.)
- Vi har erfaring med å bruke denne leverandøren

Figuren under viser tre alternative mikrokontrollerkort med mulighet for oppkobling via Wi-Fi.

Arduino UNO Wi-Fi | Arduino UNO + ESP01 m/Wi-Fi | ESP32 m/Wi-Fi

Arduino UNO m/Wi-Fi er kanskje et naturlig valg siden den er en naturlig etterfølger av Arduino UNO. Imidlertid er denne ganske dyr (ca. kr. 500,-) og CoT tilbyr ikke biblioteker for oppkobling til nettstedet på en enkel måte.

ESP32 m/Wi-Fi er en god og relativt billig (ca. kr. 200,-) løsning som blir mye brukt. CoT støtter også denne kretsen slik at den er lett å koble til CoT. Imidlertid er det noen som kvier seg for å gå over fra Arduino UNO til ESP32, selv om programmeringsverktøyet er det samme som for Arduino.

Arduino UNO pluss ESP01 kan da være en løsning. I dette tilfellet kan man forsette å bruke Arduino UNO som fange skoler har, for så og kjøpet et billig tillegg (ESP01) for å komme på nett. ESP01 er også støttet av CoT. Ulempen kan være at ESP01 kan være litt vanskelig å skaffe fra norske leverandører (ca. kr. 100,-). Det er også en fordel å skaffe seg en adapter slik at den lett kan kobles til et koblingsbrett. Dessuten så har Arduino UNO lite dynamisk minne (2 kbyte) som er lite dersom man ønsker å legge inn biblioteker for flere sensorer. Hovedårsaken er at biblioteket til CoT tar mye plass.

Vi har likevel valgt å gå for den siste løsningen.

7.2.2 Hva er ESP01?

Da ESP01 (utviklet av AI-Thinker) kom på markedet i august 2014 skapte den en del oppmerksomhet, da den ga mulighet for å koble kretser til lokale Wi-Fi-nettverk. I seg selv er den også en ganske kraftig mikro-kontroller som i tillegg inkluderer sender-mottaker, antenne, og noen få IO-porter. Den har også en UART (Tx/Rx) slik at den kan kobles opp mot andre mikrokontrollere som f.eks. Arduino UNO som normalt ikke har Wi-Fi.

Egentlig er den en nedstrippet ESP8266. Som vi ser av figuren over finnes det en rekke slike nedstrippede utgaver av ESP8266[38]. Til vårt formål har vi altså valgt ESP-01.

ESP01 finnes i to versjoner ESP01 og ESP01S som har noe større minne (1Mbyte) og har i tillegg til Wi-Fi også Bluetooth. ESP01 har et minne på 512 kByte. Begge har en 32-bits prosessor (Tensilica L106) med en typisk klokkefrekvens på 80 MHz, men kan programmeres til opp til 160MHz. Wi-Fi enheten sender og mottar på 2,4 GHz, men kan gå opp til 2,6 GHz og har en maksimal sendereffekt på +20 dBm (ca. 100 mW). Under normale forhold vil strømtrekket være opp til 170 mA, men kan i spesielle tilfeller være opp til maksimalt 320 mA. Hver port kan levere 12 mA. Kretsen kan legges i "Deep sleep" og skal da trekke mindre enn 20 µA. Normalt vil ESP01 inneholde programvare (firmware) levert av fabrikken som inneholder SDK (Software Development Kit) og ATI som gjør at man kan bruke AT-kommandoer for å kommunisere med sette opp firmware'en.

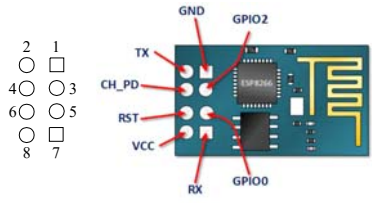
Antennen er integrert på kretskortet som en meanderformet trykt kretskortlinje (MIFA – Meandered Inverted-F Antenna).

ESP01 kan operere i tre modus:

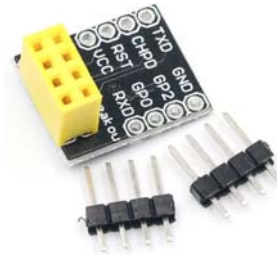
1. AP (Access Point Mode)
2. STA (Station Mode)
3. Both AP and ST Modes

Oppkobling

La oss se nærmere på tilkoblingsterminalene:



1. GND – Jord
2. GPIO 1 (Tx) – Generell IO-port og Seriell Tx
3. GPIO 2 – Generell IO-port
4. CH_PD – Chip enable. Høy aktiv krets, lav stand by
5. GPIO0 (Flash) – Generell IO-port. Hvis lav mens Reset og power er høy så går kretsen inn i programmeringsmodus
6. RESET – Kretsen resettes ved å legge denne porten lav.
7. GPIO 3 (Rx) – Generell IO-port og Seriell Rx
8. V_{CC} – Forsyningsspenning 2,5 – 3,6 V



ESP01 har to LED – En rød som lyser når kretsen har power og en blå som lyser når det foregår dataoverføring.

En av ulempene med ESP01 er at den ikke er særlig koblingsbrettvennlig, hvilket gjør at det kreves en adapter (se figuren til høyre) eller at man bruker hunn-hann jumpere.

38.https://create.arduino.cc/projecthub/ahmedibrrahim/iot-using-esp8266-01-and-arduino-afa35e

7.2.3 Bruk av AT-kommandoer – Programmering av datarate

Normal- og programmeringsmodus

ESP01 kan operere i enten *normalmodus* eller *programmeringsmodus,* som krever litt forskjellig oppkobling:

Normalmodus: Dette er den tilstanden som brukes under vanlig bruk. I dette tilfellet er både GPIO 0 og GPIO 2 lagt høy via en motstand på 10 kΩ.

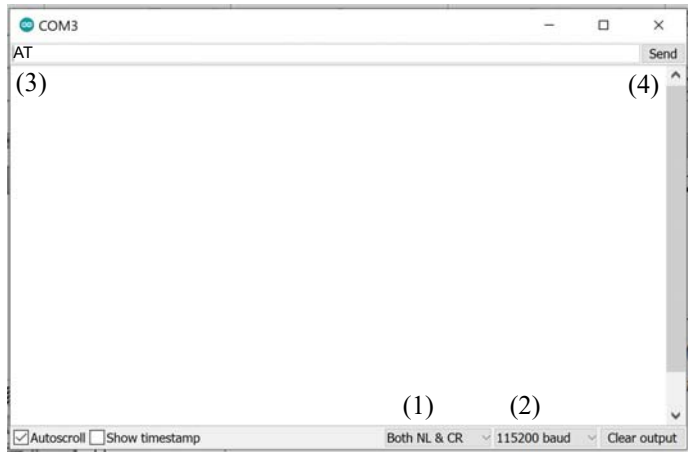
Programmingsmodus: Denne tilstanden brukes når vi ønsker å legge inn nye programmer i ESP01, ev. endre firmware. Slik programmering vil overskrive det som alt ligger i minnet. Til dette formålet kan man bruke en Arduino eller seriekonverter (USB til TTL). For å programmere ESP01 kreves at GPIO 0 er lav og GPIO 2 er høy.

CH340 Adapter

CH340 er en overgang fra USB-porten på PC'en til et signal som kan kommunisere med Tx/Rx portene hos ESP01. CH340 tilpasser også nivåene og forsynings-spenningen (3,3V/5,0V) slik at den passer til ESP01. Dessuten har den en sokkel som gjør det lett å koble til ESP01.

Vi kan bruke AT-kommandoer for å kommunisere med programvaren (firmware) som er installert i ESP01 fra fabrikken. Ved hjelp av AT-kommandoene kan en undersøke hvordan programvaren i ESP01 er konfigurert ev. endre på konfigureringen, som f.eks. hvilken kommunikasjonshastig-het den er satt opp med.

La oss se hvordan vi kan bruke monitoren hos Arduino-editoren til å endre kommunikasjonshastigheten.



Forberede bruk av ESP01 – Endring av parametere i firmware

På PC'siden må vi ha en pro-gramvare som kan sende AT-kommandoene til ESP01 via CH340. Det enkleste valget er å bruke Arduino IDE monito-ren som vist på figuren til høyre. Den kan ikke bare motta informasjon via USB-porten, men også sende informasjon til USB-porten og videre til den tilkoblede ESP01.

Vi skal nå sjekke forbindelsen med ESP01 og endre dataraten fra 115200 til 9600 bit/sek (baud).

Det er ikke nødvendig å installere tillegget for ESP01 i IDE'en for å gjøre det vi nå skal gjøre. Vi går da fram på følgende måte:

1. Plugg USB-adapteren med ESP01 inn i USB-porten på PC'en
2. Åpne Arduino editoren (IDE) med en tom skisse.
3. Velg rett port
   - *Husk å bytte til riktig COM-port ved å gå inn i menyen og velge Verktøy/Port (Tools/Port) og velg den porten som er tilkoblet adapteren.*
4. Sett opp monitoren:
   - *Sett linjeformatering til "Both NL & CR" (1)*
   - *Sett baudrate til 115 200 Baud, da ESP01 normalt er satt opp med denne hastigheten fra fabrikken.*
5. Sjekk kommunikasjonen
   - *Skriv AT (3) i kommunikasjonslinjen og trykk Send (4)*

   - *Dersom alt er OK skal det skrives ut AT og deretter OK (5)*
6. Endre kommunikasjonshastigheten til 9600 baud[39]
   - *Skriv AT+UART_DEF=9600,8,1,0,0 Om dette ikke fungerer, prøv: AT+CIOBAUD=9600 Kommandoene skal skrives inn uten mellomrom.*

   - *Du skal da få en respons i monitoren som vist på figuren over (6).*
   - *Fra nå av må man kommunisere med ESP01 med 9600 baud.*

Da skal kretsen være klar for bruk.

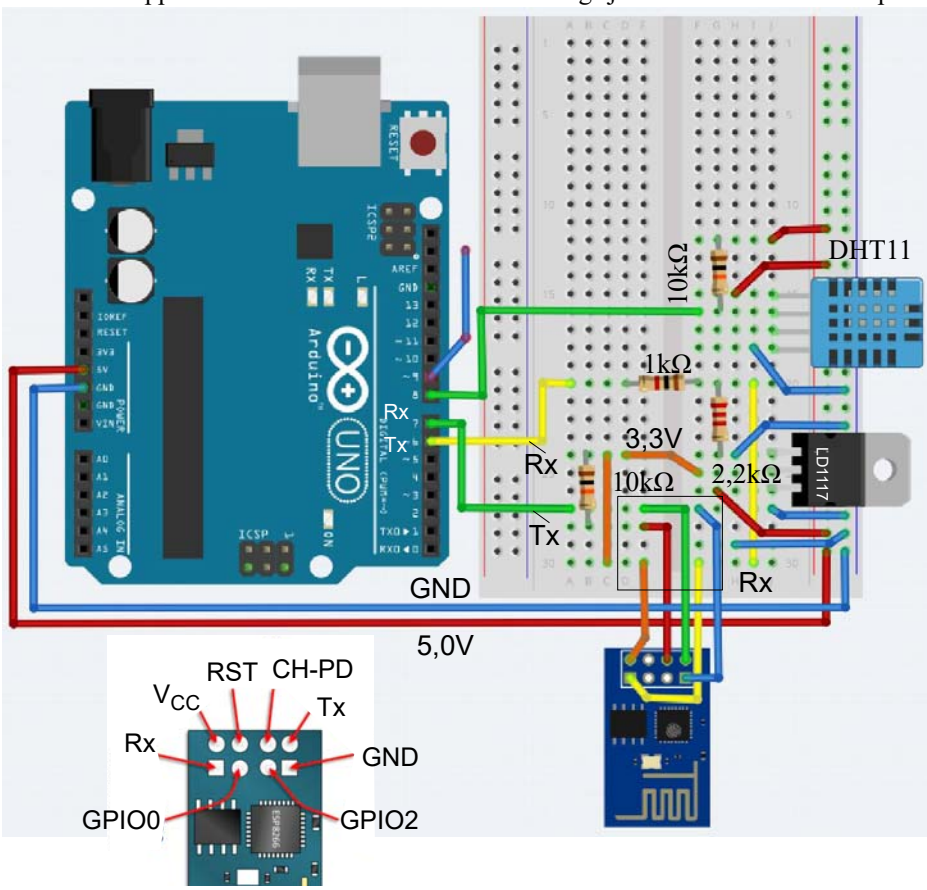
Deloppdrag 14

Koble opp ESP01 på koblingsbrettet sammen med DHT11, dersom DHT11 er plassert på den foreslåtte plassen, skal det også være plass til ESP01. Gå over og sjekk at oppkoblingen er riktig, dette gjelder spesielt spenningene siden ESP01 skal ha 3,3 V forsyningsspenning.

39.https://create.arduino.cc/projecthub/ahmedibrrahim/iot-using-esp8266-01-and-arduino-afa35e

7.2.4 Oppkobling av ESP01

Vi skal nå koble opp ESP01 sammen med Arduino UNO og sjekke om vi kommer oss på nett.



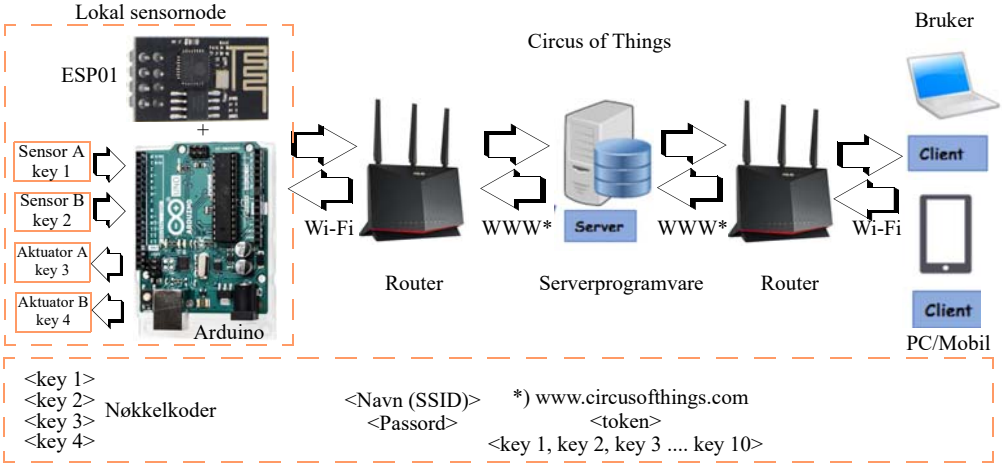
Vi bruker en adapter for å kunne plugge ESP01 direkte ned i koblingsbrettet. Dernest bruker vi regulatoren LD1117 for å regulere ned spenningen til 3,3 V. En spenningsdeler reduserer spenningsnivået hos Tx signalet fra Arduino'en slik at det skal passe til ESP01 som arbeider på 3,3 V.

Vi er nå klare til å opprette en brukerkonto hos Circus of Things som er et nettsted hvor vi kan vise målingene og fjernstyre aktuatorer våre. Via CoT kan vi nå værstasjonen vår fra hvor vi måtte befinne oss.

7.2.5 Oversikt over signalgangen fra sensornoden til CoT

Figuren under viser hvordan sensorer og aktuatorer hos den lokale sensornoden (værstasjonen) overfører data via en sentral server og videre til brukeren (klienten). Ved serveren hos Circus of Things (CoT), gis signalene som kommer fra sensorene eller skal sendes til aktuatorene, et navn og en *nøkkelkode* ("key 1 – N"). Nøkkelkoden, som er et fire eller femsifret tall, brukes til å identifiserer signalene både ved sensornoden og CoT.

Deretter gis hvert signal et *konsoll.* For sensordataene fra mikrokontrolleren er dette et display som viser sensorverdien ved CoT. For signaler som skal styre en aktuator ved sensornoden, så kan det være snakk om trykk- eller glidebrytere. Konsollene organiseres i *paneler* som kan inneholde flere konsoller som er relatert til hverandre. Brukeren får når vedkommende blir registrert, en *brukeridentitet* ("token") som også knyttes til signalene som overføres.



7.2.6 Oppretting av konto ved CoT (www.circuitofthings.com)

Circus of Things (CoT) er en open-source IoT felles plattform for datalogging, overføring og utveksling av data. CoT kan fritt brukes for utviklings-, utdannings- og næringslivsformål. Plattformen gir også rike muligheter for å utveksle data med andre i fellesskapet.

Plattformen gjør det mulig å lage enkle brukergrensesnitt, etablere trådløs kontakt mellom en sensornode med f.eks. Arduino + ESP01 og andre teknologier, se figuren til høyre. Kontakten kan gå begge veier fra noden til en PC, Smart phone eller nettbrett og vice versa. Innsamlede data kan vises i "real time" som tall eller grafer, og aktuatorer kan fjernstyres ved hjelp av software-brytere og glidere for å justere fart eller lysstyrke.

Plattformen gjør det også mulig å dele design og data. Hvem som helst kan gå ut og observere åpne data som er lagt ut av medlemmer av fellesskapet, eller man kan la dataene være private eller dele dem med et utvalg partnere. Det er også mulig å inkludere andres data i egne paneler dersom dataene er gjort tilgjengelig (publisert).

CoT ble etablert og drives av ***Jaume Miralles*** som holder til på Mallorca, Spania og kan kontaktes via info@circusofthings.com[40].

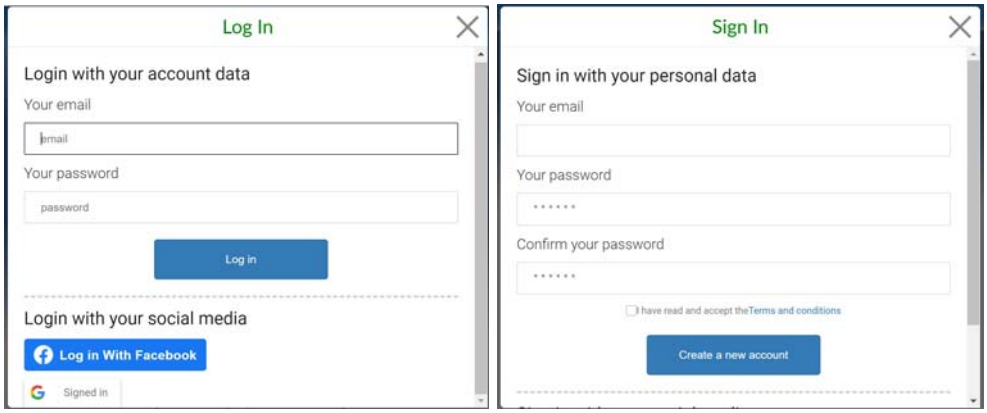
40.https://circusofthings.com/welcome#about

Deloppdrag 15

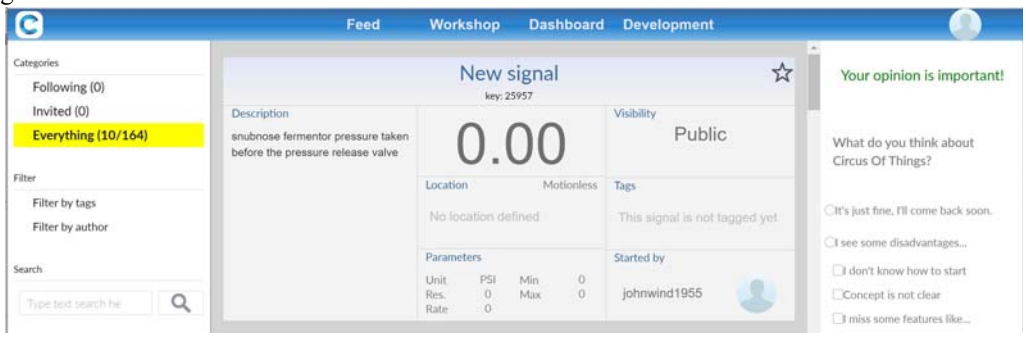
Opprett en bruker ved Circus of Things og lag to konsoller, ett for temperatur og ett for luftfuktighet. Sammenstill de to konsollene i et panel. Registrer identifikasjonskoden for din bruker ("token") og nøkkelkoden for konsollene. Disse skal brukes når programmet hos sensornoden skal skrives.

Pålogging / Signing in – Opprett en personlig konto

Gå til https://circusofthings.com/welcome og velg "*Sign in*" dersom du ønsker å opprette en egen konto. Velg "*Log in*" dersom du har en konto eller ønsker å logge inn via din Google- eller Facebook-konto.



Velg f.eks. *Sign in* og skriv inn e-post og et unikt passord. Du får da en e-post med en nettadresse for å bekrefte at du ønsker å opprette en bruker. Deretter blir du logget på og blir møtt med følgende vindu:



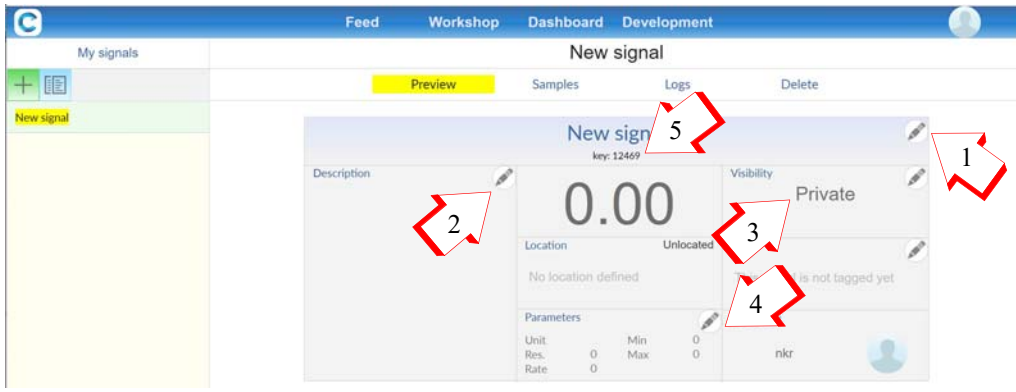
Øverst (blå linje) finner vi fire valg: *Feed*, *Workshop*, *Dashboard* og *Development*. Du befinner deg nå i Workshop hvor du kan definere hvilke signaler du ønsker å utveksle mellom CoT og sensornoden.

7.2.7 Signaler, konsoller og kontrollpaneler

Utgangspunktet er at det går *sensorsignaler* fra sensornoden til CoT og *styresignaler* fra CoT til sensornoden. Et *konsoll* er et display som viser avleste sensorverdier ved sensornoden, eller en bryter som genererer styresignaler til aktuatorer ved sensornoden. Vi ønsker innledningsvis å definere to *sensorsignaler* for temperatur og luftfuktighet med tilhørende *konsoller* samlet i et *panel*:

1. Dersom du ikke alt er der, så velg *Workshop*. Dette er stedet hvor vi kan gi signalene våre navn og der signalene får nøkkelkoder med mere. Vi får opp følgende vindu:

2. Vi skal nå legge til det signalet vi ønsker å vise. Vi velger + lengst oppe til venstre og får opp følgende vindu hvor vi kan fylle inn informasjon om signalet vårt.



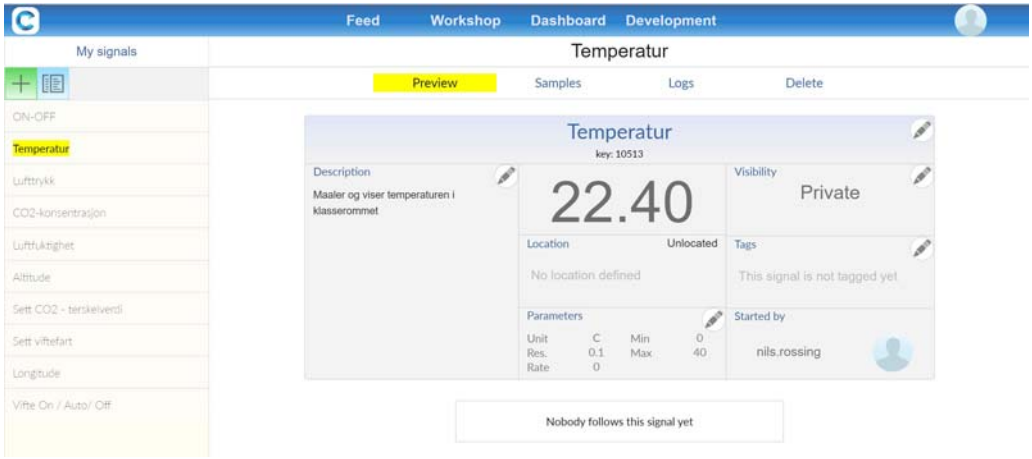
Her fyller vi ut følgende:

3. **Vi gir signalet et navn (1)** ved å klikke på blyanten til høyre for *New Signal*. Vi får da opp følgende vindu:

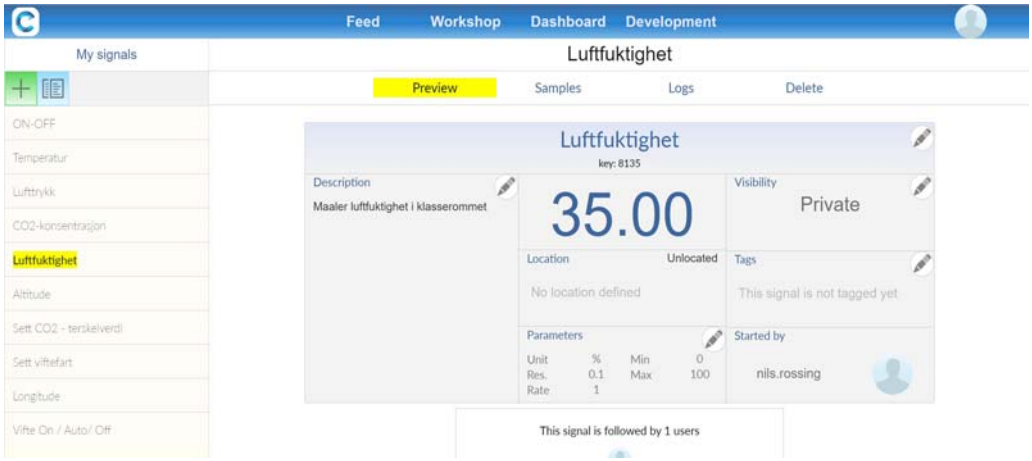
Vi velger å kalle signalet vårt *Temperatur* og trykker så ***done*** nederst i vinduet.

4. **Beskriv signalet (2).** Her kan vi beskrive signalet vårt. I vårt tilfelle kan vi f.eks. skrive at signalet viser utetemperaturen på vår adresse, eller innetemperaturen i et lokale.

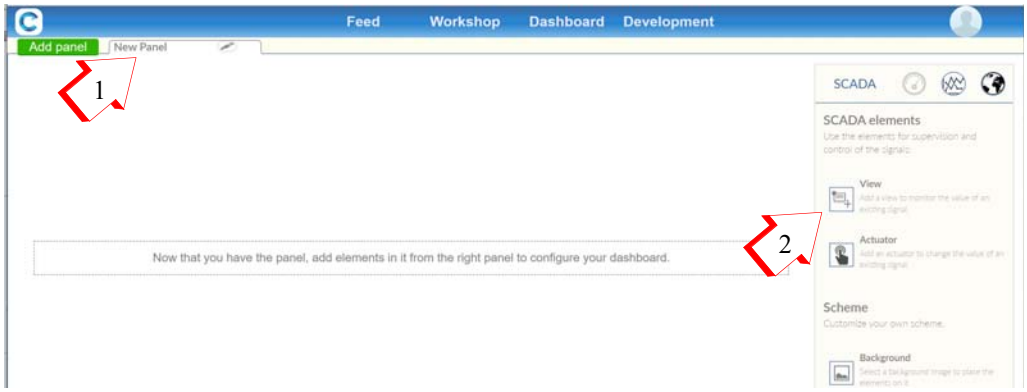
5. **Privat eller offentlig (3).** Her kan vi velge å gjøre signalet tilgjengelig for offentligheten. Vi velger foreløpig å la det være privat. Dersom elever ønsker å bruke hverandres signaler, må de først gjøre dem offentlig tilgjengelig ved å velge "Public".
6. **Parametere (4).** Her setter vi enhet på måleverdien (°C) og område f.eks. 0 – 40°C.
7. **Kopier nøkkelkode (5):** Når vi oppretter et nytt signal vil dette få en *nøkkelkode* (key), dvs. et nummer som vi kan bruke til å opprette forbindelse mellom signalet i programmet hos vår sensornode (Arduino) og konsollet ved serveren hos CoT. Vi bruker det også når vi skal velge signaler til konsollene på *Dashbordet* vårt. Vi kopierer derfor nøkkelkoden (10513) som vil være unik for hvert signal og hver bruker. I det vi forlater Workshop'en ser bildet slik ut:



8. **Luftfuktighet:** Gjør tilsvarende for signalet som formidler luftfuktighet. Hos oss får denne nøkkelkoden 8135:

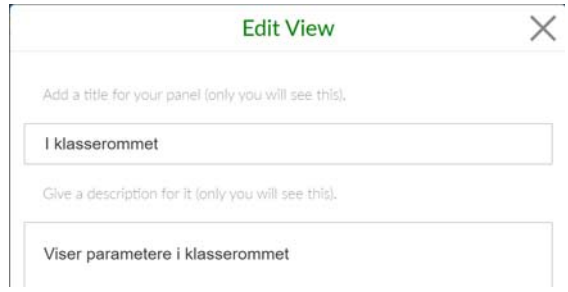


9. **Gå til Dashbordet** og få opp følgende vindu:

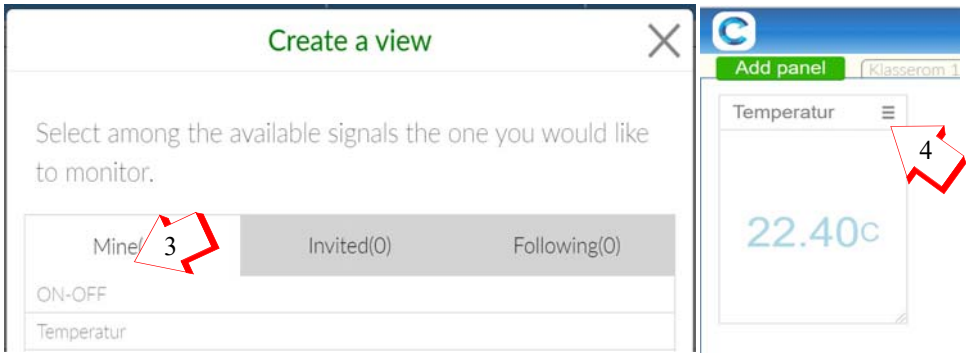


Vi skal nå lage et panel og bestykke det med våre to konsoller, displayer for å vise sensordata og etter hvert brytere for styring av aktuatorer.

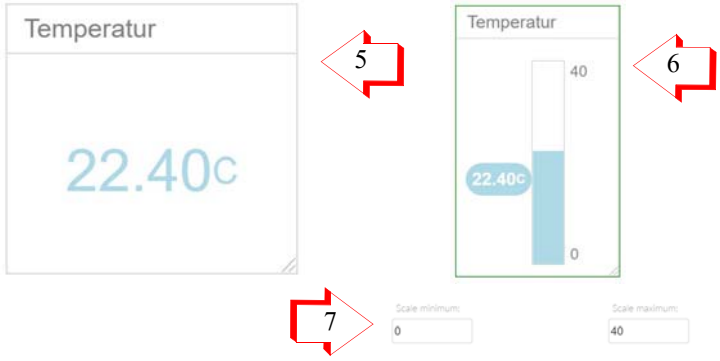
10. **Legg til et panel (1):** Her kan vi legge til ett nytt panel ved å velge *Add panel*. Dersom det alt ligger et ubrukt panel med navnet *New Panel,* så velger vi å bruke det og gi det et passende navn. Det gjør vi ved å velge blyanten til høyre for *New Panel*. Vi velger å kalle det *Weather station*.



11. **Legg til en visning (view) (2):** Vi vil lage et konsoll som vi plasserer i panelet, for å vise temperaturen utendørs her vi befinner oss. Vi går da til ikonet *View* til høyre for panelet (2). Når vi klikker på *View* kan vi velge vårt nyregistrerte signal: *Temperatur* under *Mine* (3).



12. **Definer konsollets visning:** Når vi velger *Temperatur* får vi et forslag til et konsoll. Oppe til høyre på konsollet er det et menyikon (4). Her kan vi velge mellom to visninger: Digital visning (5), eller søylevisning (6). Under søylevisningen er det to innbokser hvor vi kan sette nedre og øvre grense for skalaen (7).



Når vi har bestemt oss for en visning, trykker vi *save* nederst i høyre hjørne.

13. **Konsoll for luftfuktighet:** Så gjør vi tilsvarende for luftfuktigheten. Her velger vi digital visning. Vi har da laget vårt panel "Weather station" som viser to konsoller, ett for temperatur og ett for luftfuktighet. Vi kan dra i nederste høyre hjørne for å tilpasse størrelsen på konsollene.

Vi skal nå skrive programmet som sender dataene fra sensorene til Circus of Things.

7.2.8 Program for overføring av data til Circus of Things (CoT)

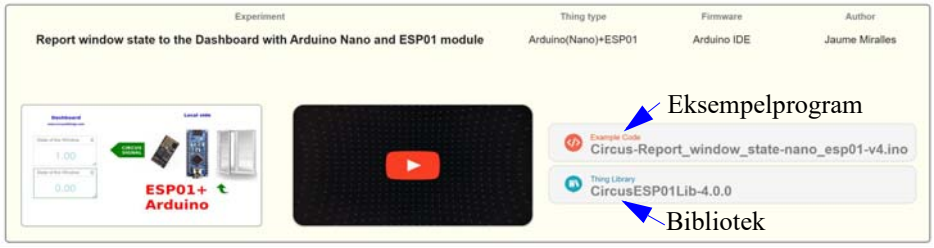
Deloppdrag 16

Skriv et program som overfører temperatur og luftfuktighet til konsollene hos Circus of Things. Sjekk at data overføres tilfredsstillende og vises i konsollene hos CoT.

Vi skal nå lage et lite program som sender informasjon fra Arduino'en, via ESP01 og til CoT. Vi skal bygge opp programmet trinn for trinn og forklare hvert trinn. Vi tar utgangspunkt i overføring av temperatur og luftfuktighet fra sensoren DHT11.

1. **Nedlasting og installasjon av biblioteker**
   Vi trenger to biblioteker, ett for å hente data fra DHT11 og ett for å koble oss opp mot internett og Circus of Things. Biblioteket for DHT11 kan vi hente fra:
   https://github.com/adafruit/DHT-sensor-library
   Biblioteket kan også installeres ved hjelp av Arduino Library Manager og søke etter *DHT sensor library Adafruit.*

   Deretter henter vi CoT biblioteket fra
   https://github.com/jaumemirallesisern/CircusESP01Lib/releases/tag/v1.1.0
   Et eksempelprogram kan hentes fra CoT
   https://www.circusofthings.com/dev.jsp#experiments
   og gå ned til følgende eksempel:



   Biblioteket som ligger sammen med eksempelet kan være en gammel versjon CircusESP01Lib-4.0.0. Skal du laste ned biblioteket herfra så sørg for at det har betegnelsen: CircusESP01Lib-1.1.0 (dette var ikke oppdatert pr. 26.03.22). Ikke pakk ut .zip filene, men installer dem slik de er.

   Installer bibliotekene på vanlig måte ved og gåd til Sketch/Include library/Add .ZIP library.

2. **Inkluder bibliotekene**
   Bibliotekene inkluderes i programmet med følgende kommandoer:

```
#include "DHT.h"
#include <CircusESP01Lib.h>
```

   Disse legges inn før void setup()-funksjonen.

3. **Deklarer en instans for DHT11**
   Vi deklarerer en instans (objekt) som bl.a. tar vare på data fra DHT11 og velger ut riktig sensor og hvilken port hos Arduino'en som skal motta data. Dette gjør vi med følgende kommandoer:

```
#define DHTPIN 8          // Digital port koblet til DHT11
#define DHTTYPE DHT11     // DHT 11
DHT dht(DHTPIN, DHTTYPE); //Lager en instans dht av typen DHT
```

Også disse plasseres før void setup()-funksjonen.

4. **Deklarasjoner knyttet til Circus of Things**
   Vi skal nå legge inn informasjon som trengs for å logge seg på din lokale router, nettsiden til Circus of Things og for å identifisere deg og din brukerkonto ("token"), dessuten må signalene sendes til sine respektive konsoller. Vi har alt definert konsollene for temperatur og luftfuktighet og hver av disse har fått sin nøkkelkode.

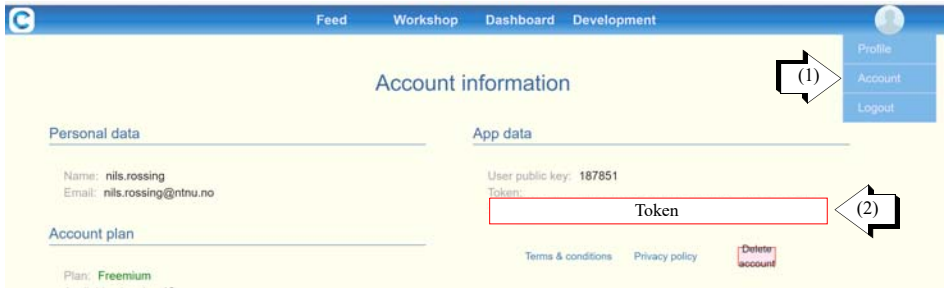
   - ***Deklarerer variablene*** som holder informasjonen som trengs for å opprette kommunikasjonen fra sensornoden og fram til CoT:

     ```
     char ssid[] = "<Ruterens navn (SSID) legges inn her>"; // v/NTNU: NTNU-IOT
     char password[] =  "<Ruterens passord legges inn her>"; // v/NTNU: ""
     char server[] = "www.circusofthings.com"; // CoT-serverens nettside
     char token[] = "<token settes inn her>";
     ```

     Her må du skrive inn *navnet* og *passordet* til din lokale router. Denne informasjonen legges inn i de respektive variablene (med hermetegn, men uten < >).

     Tilsvarende må vi oppgi nettadressen til Circus of Things: www.circusofthings.com

     Tilslutt må vi finne var identifikator, *"token"*, knyttet til vår personlige konto ved Circus of Things, som er en ganske lang kode som vi finner under *Account* (1) og *Token* (2):

   - **Deklarasjon av variabler som holder nøkkelkoder:** Nøkkelkodene er knyttet til hvert enkelt signal som vi definerer i Workshop hos CoT (se avsnitt 7.2.6 på side 126). Disse legger vi inn i variabler med et meningsfylt navn:

     ```
     char humiditySignalKey[] = "8135";        // Nøkkelkoden til luftfuktighet
     char temperatureSignalKey[] = "10513";    // Nøkkelkoden til temperatur
     ```

     I dette tilfellet har vi definert 2 signaler som har fått hvert sin nøkkelkode. Nøkkelkodene gjengitt her vil være individuelle og må skiftes ut med det som framkommer når hver enkelte bruker etablerer signalene sine.

   - ***Opprett "software Serial":*** Normalt overfører vi data til Wi-Fi-enheten (ESP01) via den innebygde UART Rx/Tx som er koblet til D0/D1. Denne brukes også for å kommunisere fra Arduino og tilbake til monitoren i PC'en. For å beholde muligheten til å kommunisere med monitoren, velger vi å opprette en software UART via portene D6 (Tx) og D7 (Rx):

     ```
     int TXPinToESP01 = 6; // IO port som brukes som TX for seriekommunikasjon med ESP01
     ```

```
int RXPinFromESP01 = 7; // IO port som brukes som RX for seriekommunikasjon med ESP01
SoftwareSerial ss(RXPinFromESP01,TXPinToESP01);
```

- ***Deklarering av flere variabler:*** I tillegg ønsker vi å deklarere noen variabler som vi vil bruke når vi oppretter objektet som vi ønsker å bruke når vi setter opp kommunikasjonen via ESP01:

```
int esp01BaudRate = 9600;   // Dataraten for kommunikasjon med ESP01.
int debugLevel = DEBUG_YES; // Velg debug-nivå. DEBUG_NO, DEBUG_YES or DEBUG_DEEP
int enableSSL = 0; // Bruk SSL-kryptering eller ikke under kommunikasjonen med CoT.
                   0 -> Disable. 1 -> Enable.
```

  esp01BaudRate er datahastigheten vi bruker for å kommunisere med ESP01. Denne bør ikke være for høy. 9600 baud er passe.

  debugLevel angir hvor mye informasjon som gjøres tilgjengelig for feilfinning. DEBUG_YES kan være greit når man tester ut forbindelsen mens man utvikler programvaren. DEBUG_DEEP bør brukes dersom man ikke får opprettet forbindelse med Circus of Things og vil vite hvor feilen oppstår. DEBUG_NO bør brukes under normal drift.

  enableSSL angir om man skal bruke SSL-kryptering eller ikke. Normalt er det lurt å bruke kryptering, men under uttestingen av ESP01 var det ikke mulig å få opprettet forbindelse med CoT med SSL påslått. Dette kan skyldes gammel firmware hos ESP01. Vi velger derfor å slå av krypteringen.
  Avslått kryptering: enableSSL = 0; Påslått kryptering: enableSSL = 1;

  Disse kommandoene plasseres før void setup()-funksjonen.

- ***Opprettelsen av et objekt:*** Dernest opprettes et objekt som vi velger å kalle `circusESP01()` av typen `CircusESP01Lib`. Objektet fylles opp med verdiene som er lagt inn i variable foran:

```
CircusESP01Lib circusESP01(&ss,esp01BaudRate,server,token,ssid,password,
debugLevel,enableSSL);
```

  Legg merke til at her inngår server-adressen, ruternavnet (SSID) og passordet til ruteren med mere. Denne plasseres før void setup()-funksjonen.

- ***Initialisering:*** I void setup()-funksjonen initialiseres funksjonene som står for kommunikasjonen med DHT11 og serveren ved CoT med følgende kommandoer:

```
circusESP01.begin();       // Initialiser Circus of Things
dht.begin();         // Initialiser DHT11
```

- ***Les data fra sensoren:*** I selve loop()-funksjonen leser vi først av temperatur og luftfuktighet fra DHT11:

```
float  H = dht.readHumidity();    // Returner avlest lufttrykk
float  T = dht.readTemperature(); // Returner avlest temperatur
```

  De avleste verdiene legges inn i desimaltalls-variablene *H* og *T*.

- ***Send dataene til panelet hos CoT:*** Dernest skal de avleste sensorverdiene sendes over til CoT for å vises i de ulike konsollene (displayene). Dette gjøres med følgende kommandoer:

```
circusESP01.write(humiditySignalKey,H);
circusESP01.write(temperatureSignalKey,T);
```

I eksempelet over sendes måledata for temperatur og luftfuktighet til panelet hos CoT. Sensordataene er lest inn i variablene *temperature* og *humidity*. Disse knyttes til nøkkelkodene til de to displayene `temperatureSignalKey` og `humiditySignalKey`. Vi legger merke til at brukeridentiteten ("token") knyttes til hvert av signalene.

I tillegg kan det være lurt å skrive ut måledataene til monitoren. Ellers skal nå sensorverdiene vises på konsollene hos CoT.

7.3 Fjernstyring av rele

Læringsmål

Oppdraget trener følgende læringsmål:

- Bruk av aktuator-konsoller i CoT
- Koble opp et rele
- Skrive programvare for å motta kommandoer for fjernstyring av releet fra CoT
- Sette opp konsoller ved CoT for fjernstyring av aktuatorer

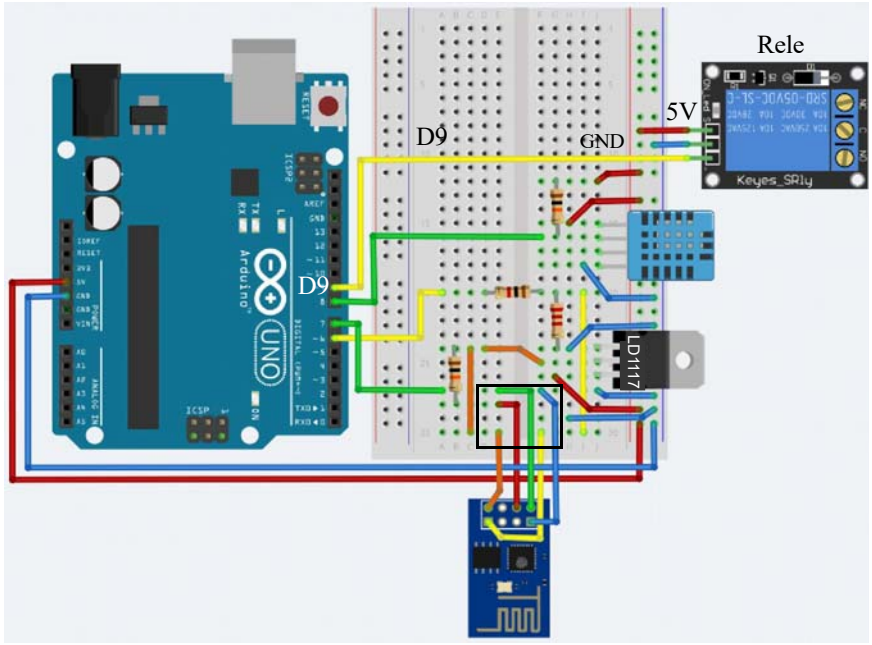
I denne oppgaven skal vi lage et lite tillegg til programmet foran som gjør det mulig å slå av og på et rele. Releet kan f.eks. senke en persienne eller slå opp en markise dersom temperaturen blir for høy eller solinnstrålingen blir for intens.

Deloppdrag 17

Koble opp et rele, lag et konsoll for å slå av og på releet og skriv programmet som mottar kommandosignaler fra CoT og styrer releet ved sensornoden.

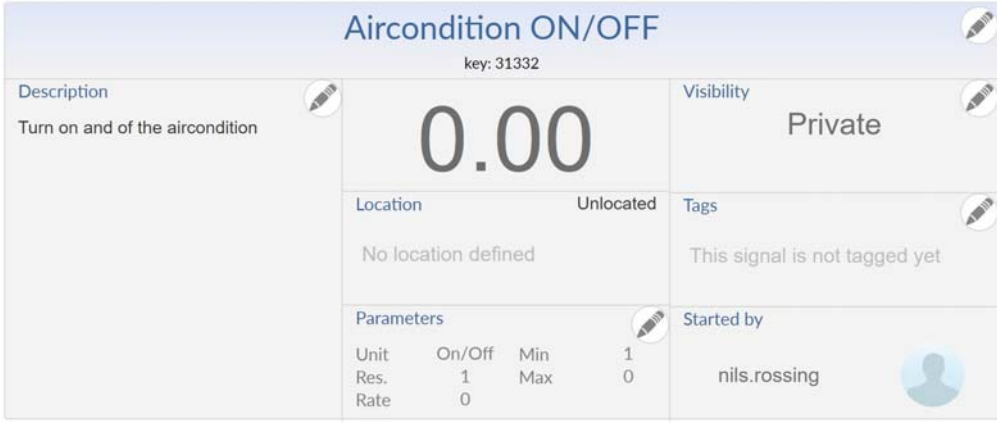
Oppkobling

Vi kobler opp et rele i tillegg til DHT11 og ESP01 som vist på figuren til høylyre. Releet kobles til +5V og jord (GND). Styresignalet til releet kobles til port D9 på mikrokontrolleren.



Signal, konsoll og panel

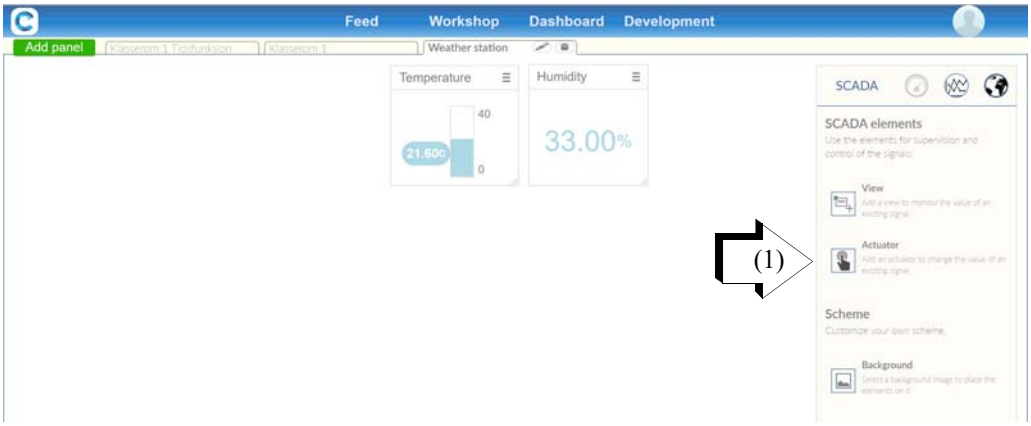
Siden vi trenger signalets nøkkelkode før vi kan skrive programmet lager vi konsollet før programmet. Vi ønsker å lage et konsoll hvor vi kan slå av og på ventilasjonsanlegget. Vi må først gå til "Workshop'en" og definere et signal som vi f.eks. kan kalle "Aircondition On/Off".



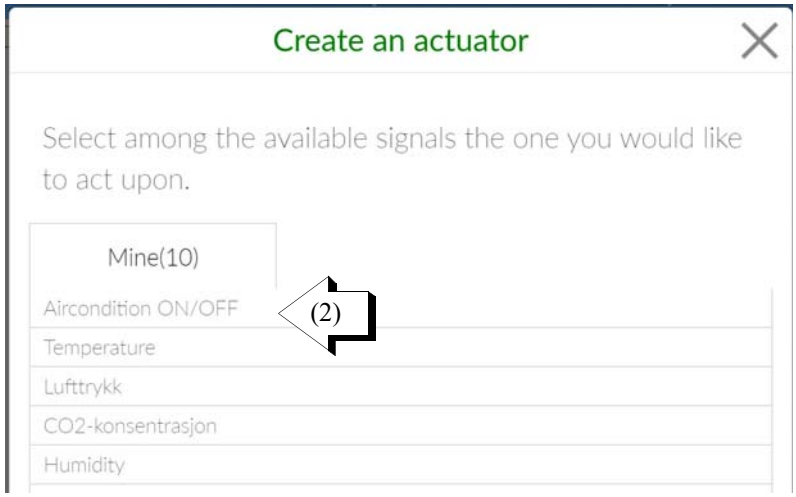
Vi merker oss nøkkelkoden: 31332

Dersom du er usikker på hvordan du skal definere konsollet se avsnitt 7.2.7 side 128.

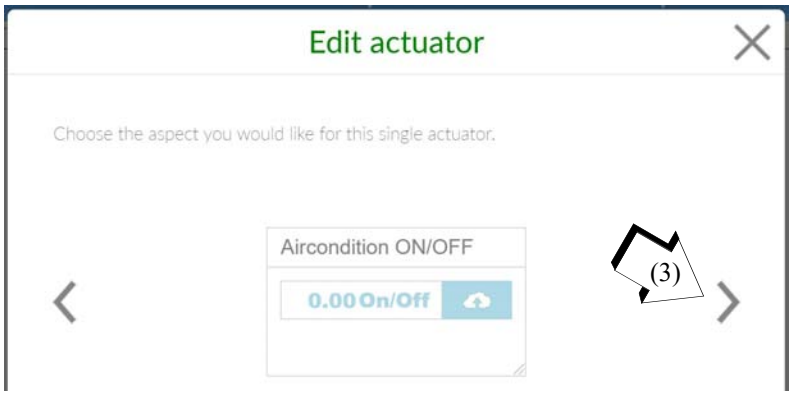
Dernest går vi til panelet vårt og legger inn det nye konsollet. Siden det denne gangen handler om å styre en aktuator og ikke bare vise målinger av en sensor, så velger vi "Actuator" på høyre side av panelet (1).



Vi får da opp et vindu som gir oss mulighet til å velge hvilket signal vi ønsker å knytte til konsollet. Vi velger "Aircondition On/Off" (2). Vinduet lukkes når vi har valgt signal.



Konsollet dukker opp i panelet som et tomt konsoll med det angitte signalnavnet "Aircondition On/Off". Ved å klikke på menystripene øverst til høyre på konsollet så får vi opp følgende vindu:



Pilene til høyre (3) og venstre gir oss mulighet til å velge mellom et konsoll med tre ulike utforminger, bl.a. ett med trykknapper og ett med en glidebryter, vi velger trykknapper:

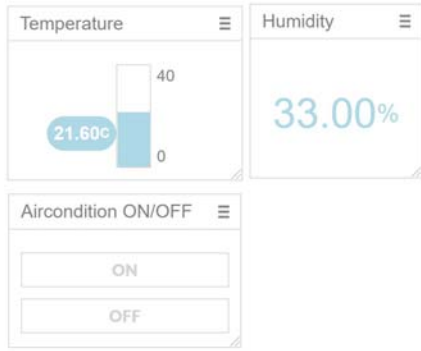
Glidebryter Trykknappbryter

Vi må deretter legge til knapper og definere hvilke tallverdier knappene skal sende over til sensornoden når vi trykker på dem. Trykk "ADD BUTTON" og få opp rubrikk for "Label" (ON) og "Value" ("1"). Legg inn to knapper en ved navnet "ON" og verdien "1", og en med navnet "OFF" og verdien "0".

Avslutt med å velge "Save" nederst i vinduet. Her er det også mulig å velge "Remove actuator" dersom man ikke ønsker konsollet likevel.

Ved å velge "Save", dukker det ferdig utformede konsollet opp i panelet.

Plasser konsollet på ønsket plass, og trekke i nederste høyre hjørne for å gi det den størrelsen du ønsker slik at knappene kommer tydelig fram.



Dermed har vi laget oss et panel som viser temperatur og luftfuktighet, og der vi også kan slå av og på et rele som ev. kan styre en markise, et vifteanlegg eller noe annet.

Da er vi klare til å endre programmet i mikrokontrolleren.

Programmering

Vi tar utgangspunkt i programmet som vi laget i avsnitt 7.2.8 side 132. Her vil vi kun beskrive tilleggskommandoene vi trenger for å kunne styre releet med konsollet "Aircondition ON/OFF".

1. **Port for styring av rele**
   Som tidligere kan det være greit å sette navn på porten som vi skal styre releet med. Vi husker at denne var port D9.

   ```
   int pinRelay = 9;
   ```

   Denne legges før `void setup()`-funksjonen. Husk å angi denne som en "`OUTPUT`" med `pinMode()`-kommandoen i `void setup()`-funksjonen.

2. **Legg inn nøkkelkode for releet**
   Da vi definerte signalet "Aircondition ON/OFF" fikk vi tildelt nøkkelkoden 31332. Vi deklarer en `char` variabel for å holde nøkkelkoden til dette konsollet. Vi gir variabelen navnet `relaySignalKey[]`:

   ```
   char relaySignalKey[] = "31332";
   ```

   Denne legger vi inn sammen med de andre nøkkelkodene før void setup()-funksjonen.

3. **Les verdien sendt fra konsollet**
   Vi bruker nå nøkkelkoden til å lese hva konsollet for styring av releet sender over fra CoT. Det gjør vi med denne kommandoen:

   ```
   int R = circusESP01.read(relaySignalKey);
   ```

   Kommandoen velger vi å legge i void loop()-funksjonen eller i en egendefinert funksjon. Legg merke til at vi legger resultatet fra lesningen i variabelen R. som vi nå skal bruke for å sjekke hvilken tilstand releet skal settes i "ON" (1) eller "OFF" (0).

4. **Test variabel og styr releet**
   Vi vil bruke verdien til R for å slå releet på eller av, dette gjør vi ved å teste på R med en if()-setning

   ```
   if(R == 1) digitalWrite(pinRelay, HIGH);
   else       digitalWrite(pinRelay, LOW);
   ```

Legg merke til at når det kun er en kommando som skal utføres for hver if-kommando så kan vi sløyfe klammeparentesene. Dersom R == "1" så slår vi releet på, er den "0" eller noe annet, slår vi det av.

5. **Skriv programmet og test**
   Skriv programmet, last opp og sjekk om mikrokontrolleen lystrer trykknappbryteren i panelet hos CoT.

7.4 Automatisk fjernstyring av releet

Læringsmål

Oppdraget trener følgende læringsmål:

- Bruk av glidebrytere hos CoT
- Skrive programvare som tar imot og bruker terskelverdier for styring av releet

I forrige deloppdrag opprettet vi et konsoll for manuell styring av releet (ventilasjonsanlegget eller markisen). I dette oppdraget skal vi endre programmet slik at det er den målte temperaturen som styrer ventilasjonsanlegget.

Deloppdrag 18

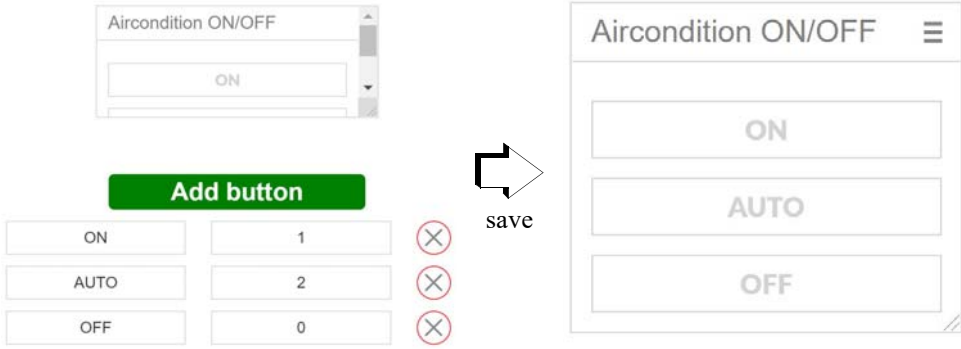
Legg inn en tredje knapp, "AUTO" i trykknappkonsollet som er slik at når auto er trykket så er det temperaturen som styrer om ventilasjonsanlegget slås på eller av. Det skal også defineres et signal og lages et konsoll som gjør det mulig å sette terskelverdien for temperaturen for omslag.

Oppkobling

Det er ikke behov for ekstra komponenter til dette deloppdraget.

Konsollet

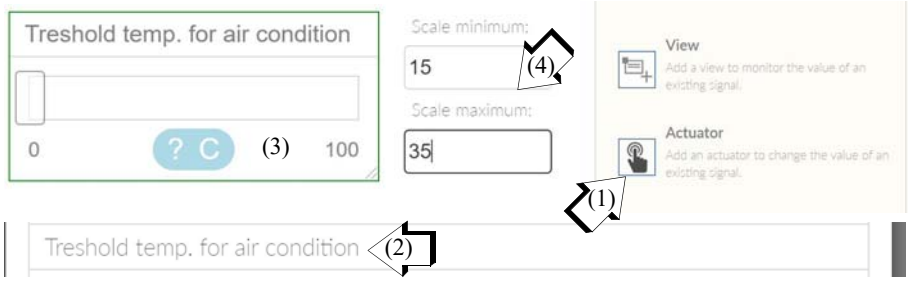
Det første vi skal gjøre er å legge inn en ekstra auto-knapp i "Aircondition ON/OFF"-konsollet. Vi ordner knappene slik at "AUTO" blir liggende mellom "ON" og "OFF". Vi knytter verdien "2" til "AUTO".



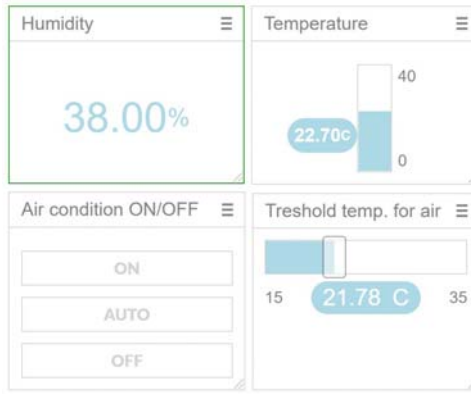
I tillegg må vi opprette et nytt signal i workshop'en som gir oss mulighet for å stille terskelverdien ved hjelp av en glidebryter. En slik bryter krever at vi må sette yttergrensene for terskelkontroll-signalet. Vi velger å la terskelverdien fra 15 – 35°C.

Vi legger merke til at nøkkelkoden er: 17726.

Når vi har definert signalet kan vi lage konsollet i panelet. Vi går til panelet og velger "Actuator"-ikonet på høyre side (1), velger det aktuelle signalet (2) og velger å bruke en glider. Vi setter ytter-verdiene til 15 og 35°C.



Vi kan da ende opp med et panel som ligner på dette:



Vi er dermed klare til å skrive programmet.

Programmering

Vi bygger på programmet utviklet for forrige oppdrag og begynner med å legge inn nøkkelkoden for glidebryteren for terskeltemperaturen:

1. **Legg inn nøkkelkode for terskelnivå**
   Da vi definerte signalet "Treshold temp. for air condition" fikk vi tildelt nøkkelkoden 17726. Vi deklarer en `char` variabel for å holde nøkkelkoden til dette konsollet:

   ```
   char tresholdSignalKey[] = "17726";
   ```

   Denne legger vi inn sammen med de andre nøkkelkodene før void setup()-funksjonen.

2. **Les verdien sendt fra konsollet**
   Vi bruker nøkkelkoden til å lese hva konsollet for innstilling av terskelnivå har oversendt fra CoT. Det gjør vi med denne kommandoen:

   ```
   float Tr = circusESP01.read(tresholdSignalKey);
   ```

   Denne kommandoen velger vi å legge i void loop()-funksjonen eller i en egendefinert funksjon. Legg merke til at vi legger resultatet fra lesningen i variabelen *Tr.* som vi nå skal bruke for å ta vare på den mottatte terskelverdien.

3. **Test variabel og styr releet**
   I tillegg til "ON" (1) og "OFF" (0) for styring av releet, har vi lagt inn en tredje mulighet med "AUTO" som er gitt verdien 2. I tillegg til å sjekke for R==0 og R==1, må vi også sjekke for R==2. Vi får da følgende betingelser

   ```
   if (R == 1) digitalWrite(pinRelay, HIGH);
   else if (R == 2)
   {
     // Her legges aksjonen som skal utføres ved "AUTO"
   }
   else       digitalWrite(pinRelay, LOW);
   ```

4. **Aksjon for "AUTO"**
   Dersom "AUTO" er valgt må vi i tillegg finne ut om den målte temperaturen er over eller under terskelverdien. Det gjør vi med en ekstra if-setning

```
else if(R == 2)
  {
    if       (T >= Tr) digitalWrite(pinRelay, HIGH);
    else if  (T <  Tr) digitalWrite(pinRelay, LOW);
  }
```

   Dersom den måte temperaturen T er over terskel verdien slås ventilasjonen på, om dette ikke er tilfelle slås den av.

5. **Skriv og test programmet**
   Skriv programmet og test om det oppfører seg som forventet.

7.5 Styring av viftemotor med H-bro (L293B)

Læringsmål

Oppdraget trener følgende læringsmål:

- Oppkobling og bruk av H-bro for styring av rotasjonsretning hos en liten DC-motor

I de to foregående deloppdragene opprettet vi et konsoll for manuell og automatisk styring av rele (ventilasjonsanlegget). I dette oppdraget skal vi styre retningen og hastigheten til viftemotoren.

Deloppdrag 19

Monter og koble opp styrekretser for styring av en vifte (liten DC-motor). Skriv et program som slår av og på viften og endrer mellom inn- og utlufting. Dvs. vifta blåser lufta inn i 5 sekunder for så å blåse den ut i 5 sekunder.

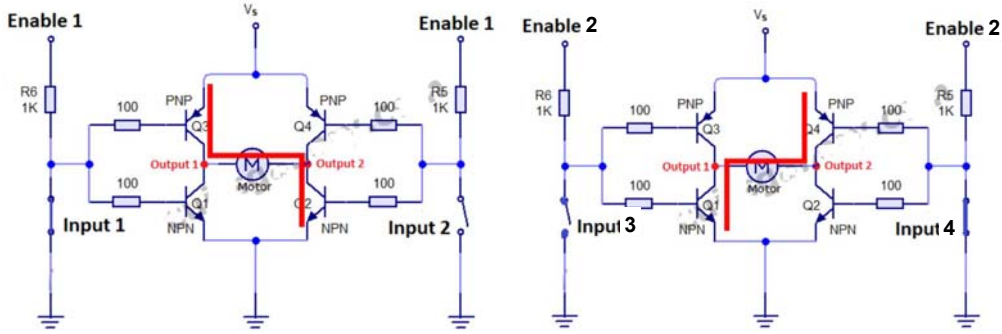
Lag en funksjon for styring av viftemotoren der argumentet angir rotasjonsretningen.

Kort omtale
... av H-broen L293B

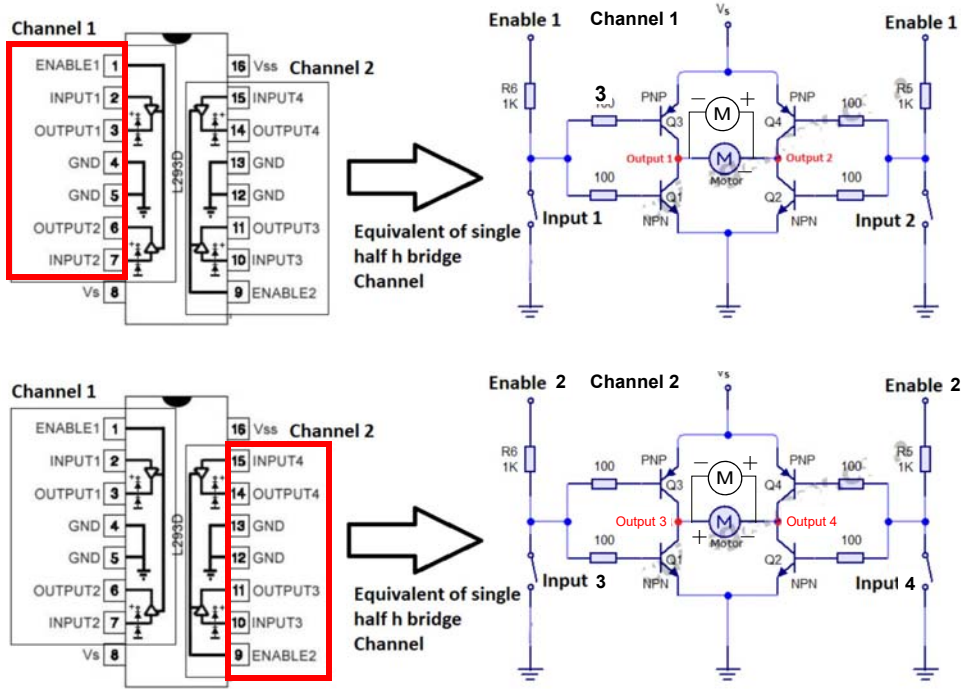
Noen ganger vil det være aktuelt å snu rotasjonsretningen til en motor. Dette er spesielt aktuelt i forbindelse med bygging av roboter, men også i tilfeller dersom man ønsker å snu luftstrømmen i en vifte. Dette kan gjøres ved hjelp av to transistorbrytere styrt av mikrokontrolleren. Figuren[41] under viser hvordan vi skifter polaritet ved å lukke henholdsvis bryter Input 1 og bryter Input 2. Bryterne lukkes ved å legge inngangene til høy (1). Enable 1skal normalt ligge høy (5V), men kan

41. Figurene er hentet fra: https://www.engineersgarage.com/l293d-pin-description-and-working/

brukes til å regulere farten til motoren ved å pulsbreddemodulere Enable 1-inngangen.



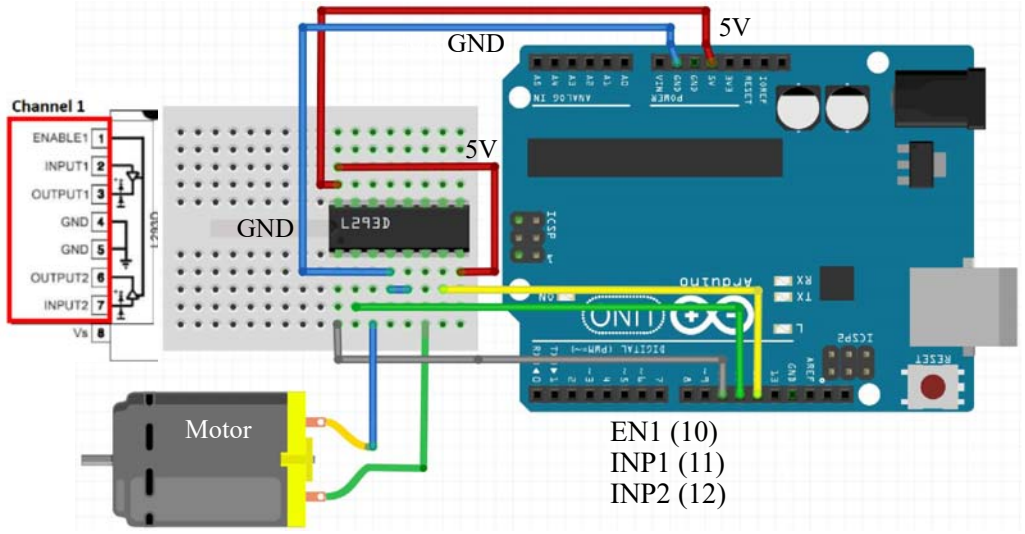
Kretsen har to like kretser som er vist i figuren over. Den andre er helt lik og styres av Input 3 og 4, og farten kan reguleres med Enable 2 som vist på til venstre på figuren under.



Enable-inngangene kan pulsbreddemoduleres slik at vi etter hvert også kan styre farten omtrent trinnløst (se avsnitt 7.6 på side 146).

Oppkobling

Figuren under viser oppkobling av deloppdraget, med H-bro og motor. Vi har valgt å bruke et lite koblingsbrett til dette eksperimentet for så senere å inkludere det i resten av oppkoblingen.



Programmering

Programmet for dette oppdraget kan gjøres svært enkelt. Følgende kan være viktig å huske på:

1. **Sett Enable inngangen**
   Enable er en inngang som enten slår motoren på (En1 = 1) eller slår den av (En1 = 0). Vi skal senere bruke denne inngangen til å pulsbreddemodulere motoren slik at vi kan styre farten.
2. **Sett INP1 og INP2**
   Disse to inngangene styrer bryterne inne i H-broa og ***må alltid være i motfase***. Dvs. at når INP1 = 1 (på) så må INP2 = 0 (Av) og omvendt.
3. **Valg av motor**
   Det er ingen ulempe å velge en motor som trekker relativt lite strøm. Vi har derfor valgt å bruke solcellemotorer i dette eksempelet. L293B tåler 600 mA og er beskyttet med dioder, men USB-inngangen kan bli overbelastet siden den også skal drive andre ting enn motoren.
4. **Skriv og test programmet**
   Skriv programmet og test at det fungerer som tiltenkt.
5. **Lag en funksjon**
   Lag en funksjon hvor du ved hjelp av argumentet til funksjonen kan bestemme rotasjonsretningen.

7.6 Styring av farten til viftemotor med PWM

Læringsmål

Oppdraget trener følgende læringsmål:

- Bruk av pulsbreddemodulasjon for kontinuerlig styring av farten til motoren

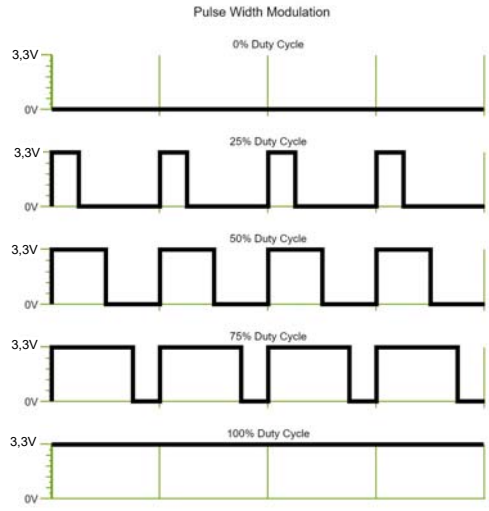
Vi skal i dette oppdraget styre farten til viftemotoren ved hjelp av pulsbreddemodulasjon (PWM).

Deloppdrag 20

Lag et program som i tillegg til å snu retningen på motoren øker farten fra 0 til maks fart i løpet av de 5 første sekundene, for deretter å senke farten fra maks fart til 0 fart de neste 5 sekundene. Deretter gjøres det samme mens motoren kjøres i motsatt retning.

Kort omtale

Pulsbreddemodulasjon for styring av motorhastighet[42].



Normalt ville vi ha styrt hastigheten til slike motorer ved å endre på spenningen. Motoren får i dette tilfellet tilført 5V. Reduseres denne spenningen vil hastigheten til motorene også reduseres tilsvarende. Til dette trenger vi en kraftig digital til analog omformer som kunne styrt spenningen til motorene. Dette er en relativt kostbar komponent som sjelden inkluderes i mikrokontrollere.

Istedet har vi mulighet til å hakke opp Enable-signalet til H-broa slik at motoren kun får full spenning i korte tidsrom. Når Enable legges lav så vil strømmen til motoren brytes og motoren stopper. Dersom man slår av strømmen kun i korte tidsintervaller, så vil hastigheten reduseres noe, men ikke rekke å stoppe. Forholdet mellom på- og av-tiden vil dermed bestemme hvor stor del av tidsintervallet som motoren får strøm, og dermed vil farten reduseres. På denne måten vil motoren oppføre seg som om den ble regulert av en variabel spenning lik en middelverdi av de pultbreddemodulerte pulsene.

Figuren over til høyre viser fra øverst til nederst hvordan andelen på-tid øker fra 0 – 100%. Denne prosentvise andelen kalles på engelsk "duty cycle".

Arduino UNO har tilrettelagt for å lage slike signaler på 6 av sine digitale porter (D3, D5, D6, D9, D10 og D11). La oss se nærmere på hvordan vi kan lage et slikt signal.

42.https://www.electronicshub.org/esp32-pwm-tutorial/

Kommando for pulsbreddemodulasjon

Vi merker oss at kun 6 av de 14 digitale portene kan pulsbreddemoduleres (D3, D5, D6, D9, D10 og D11). Disse er merket med en *tilde* (~) som vist på figuren til høyre.

Vi har derfor valgt port D10 som utgang for styring av EN1 og dermed også farten til motoren. Ved å pulsbreddemodulere EN1 (enable) så styrer vi strømmen til motoren med kortere eller lengre pulser og slik varierer vi hastigheten.

Kommandoene vi kan bruke er disse:

```
pinPWMmotor = 10; // Port som PWM-enable inngangen
int fart;         // Kan ha verdier fra 0 (0 fart) til 255 (full fart)
analogWrite(pinPWMmotor, fart);
```

Hvor:

`pinPWMmotor`	er porten vi ønsker å styre motoren med, i vårt tilfelle D10
`fart`	er en verdi fra 0 – 255 som angir pulsens "påtid". En verdi på 255 gir en puls som er på hele tiden, dvs. maksimal fart.

Oppkobling

Det er ingen ekstra oppkobling for å realisere dette oppdraget. Vi beholder oppkoblingen fra oppdrag 19.

Programmering

Vi tar utgangpunkt i programmet fra oppdrag 19 (avsnitt 7.5 side 143), der vi hadde laget oss en funksjon `void motor(int retning)` som kjørte motoren i en retning angitt av argumentet `retning`. Vi skal nå utvide anvendelsen til denne funksjonen til også å angi farten.

1. **Regulering av farten med PWM**
   Vi husker at vi kan styre farten med denne kommandoen:

   ```
   analogWrite(pinPWMmotor, fart); // pinPWMmotor (= port nummer) og fart
   ```

   hvor variabelen `fart` kan ha verdiene 0 – 255, og vi har valgt D10 som port for styring av Enable.

2. **Endring av motor()-funksjon**
   Legg inn et argument nr. 2, `int fart`, i funksjonskallet

   ```
   void motor(int retning, int fart)
   ```

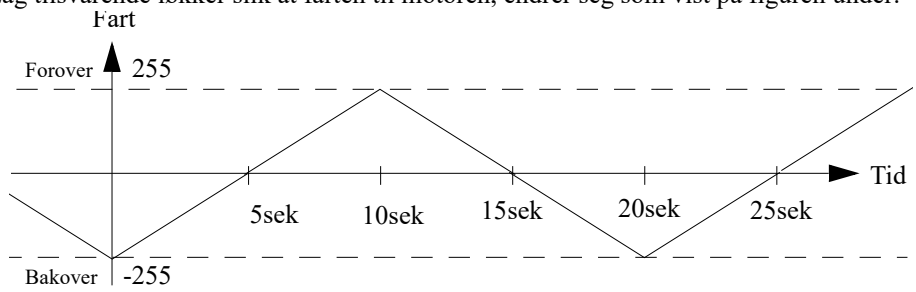
   fart kan ha verdiene 0 – 255.

3. **Lag en for-løkke som teller opp eller ned**
   Motoren skal nå enten øke farten i 5 sek. fra 0 – 255 i trinn på 16, eller redusere farten i 5 sek. i trinn på 16. Til dette bruker vi en for-loop og teller enten opp eller ned. Under ser vi hvordan vi kan telle opp:

```
int R = OUT;                  // R angir retningen OUT/IN
for(int i=0; i<=255; i=i+16) // Teller gradvis opp i fra 0 - 255 i trinn på 16
  {
    motor(R,i);               // Kaller funksjonen motor med retning (R) og fart (i)
    delay(syklusTid/16);      // Siden den skal øke farten i syklustiden (5 sek)
                                 i 16 trinn (255/16)
  }
```

Først setter vi R = OUT og lar for-sløyfen telle opp i trinn på 16 fra 0 til 255. Dette blir 16 trinn dersom vi hopper 16 verdier for hvert trinn. Siden alle de 16 trinnene må være gjort unna i løpet av 5 sekunder (syklusTid), så må løkke være på hvert trinn i syklusTid/16. Merks at vi har definert konstantene IN = 1 og OUT = 0.

4. **Lag de resterende for-løkkene**
   Lag tilsvarende løkker slik at farten til motoren, endrer seg som vist på figuren under:



5. **Forenkle**
   Lag den samme sekvensen ved hjelp av én for-løkke.

7.7 Styring av farten til vifta fra CoT

Læringsmål

Oppdraget trener følgende læringsmål:

- Bruk av glidebryter hos CoT for kontinuerlig fjernstyring av farten til en motor
- Lage program for å motta informasjon for styring av farten til en motor

Vi skal i dette oppdraget styre farten til viftemotoren via nettstedet Circus of Things.

Deloppdrag 21

Lag et program som gjør det mulig å fjernstyre farten til motoren ved hjelp av en glidebryter hos Circus of Things. La glidebryteren gå fra -255 til +255. La fortegnet styre retningen til motoren, + fortegn blåser luft inn i rommet, – fortegn blåser luft ut av rommet.

Kort omtale

Vi antar at –255 gir full fart med en rotasjon mot klokka og +255 gir full fart med en rotasjon med klokka.

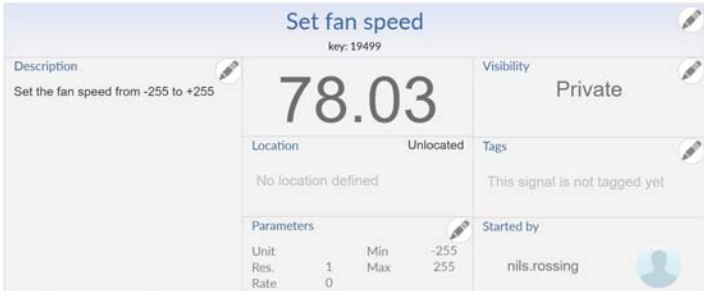
Oppkobling

Det er ingen ekstra oppkobling for å realisere dette oppdraget. Vi beholder oppkoblingen fra oppdrag 20.

Konsoll

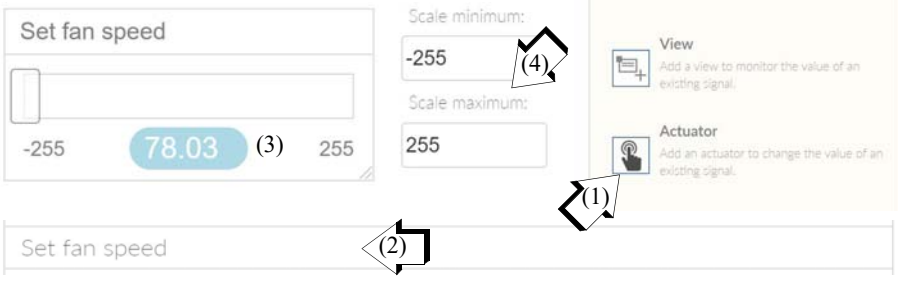
Vi skal nå lage et konsoll som vi kan styre viftefarten med.

1. **Definer signalet**
   Gå til "Workshop" hos Circus of Things og definer et signal som du kan kalle "Set fan speed".



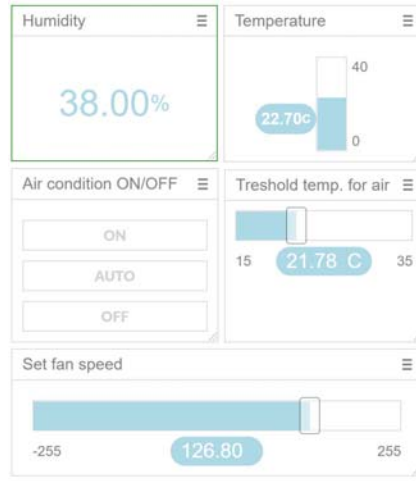
   Vi merker oss nøkkelkoden til signalet som hos oss er: 19499.

2. **Lag et aktuatorkonsoll i panelet**
   Gå til Panelet ("Dashboard") og velg aktuator til høyre (1) og du vil få opp lista over dine signaler. Velg signalet som du nettopp har definert (2): "Set Fan speed" og du vil få anledning til å velge utformingen av styrekonsollet (3). Vi har bestemt oss for en glidebryter som går fra -255 til +255 (4).



3. **Organiser panelet**
   Glidebryteren plasseres inn sammen med de andre konsollene i panelet "Weather station" da vi etter hvert skal kombinere de ulike styre- og sensor-signalene i ett og samme program.

   Figuren til høyre viser en måte å gjøre dette på.



Vi er nå klare til å skrive programmet

Programmering

Vi skal nå legge inn all informasjonen vi trenger for å fjernstyre farten og rotasjonsretningen til viftemotoren og koble opp mot den lokale routeren og mot CoT. Dette har vi nå gjort flere ganger så det burde gå greit. I veiledningen under gir vi derfor bare stikkord for hva som bør gjøres.

1. Koble opp til router og CoT
   - *Inkluder CoT- biblioteket*
   - *Legg inn informasjon for å koble til lokal router og til CoT*
   - *Deklarer variabel for nøkkelkoden til glidebryteren for styring av viftefarten*
   - *Deklarer og gi verdier til alle parametere som trengs for å etablere objektet knyttet til CoT (se også neste punkt)*

   Disse plasseres før void setup().

2. **Deklarer objektet**
   Deklarer objektet vi skal bruke for å overføre informasjon mellom CoT og vår lokale Arduino-enhet. Vi har valgt å kalle argumentet `circusESP01` som er av typen `CircusESP01Lib`:

```
CircusESP01Lib circusESP01
(&ss,esp01BaudRate,server,token,ssid,password,debugLevel,enableSSL);
```

   Som vi ser så inneholder argumentet (...) de parametrene vi tidligere har definert under punkt 1 Deklarasjonen plasseres før void setup().

3. **Applikasjonsspesifikke variabler og konstanter**
   Vi kjenner igjen følgende konstanter og variabler fra oppdrag 20:

```
int pinEn1   = 10; // Styrer farten på motoren med PWM
int pinInp1  = 11; // Styrer retningen på motoren
int pinInp2  = 12; // Styrer retningen på motoren
const int OUT= 0;  // Definerer luft ut
const int IN = 1;  // Definerer luft inn
```

   Disse plasseres også før void setup()

4. **void setup()**
   Denne inneholder det vanlige i tillegg til at vi initialiserer biblioteksfunksjonene fra CoT-biblioteket:

```
circusESP01.begin();
```

Ikke glem og definere de nye portene som utganger. Disse kommandoene legges i void setup()-funksjonen.

5. **void loop()**
   For enkelhets skyld legger vi koden som skal lese og tolke informasjonen fra CoT her. Under har vi listet opp en strategi for hva som skal gjøres. Vi har beholdt funksjonen som styrer motoren fra oppdrag 20 som overfører fart og retning `void motor(int retning, int fart)`, denne skal vi bruke.

   Gjør følgende:

   - *Deklarer lokale variabler som skal holde viftemotorens fart og retning*
   - *Les overført fart fra CoT som er et tall med fortegn. Fortegnet bestemmer retningen.*

     ```
     int fanSpeed  = circusESP01.read(fanSpeedSignalKey);
     ```

   - *Skill ut fortegnet og la det bestemme rotasjonsretningen*
   - *La absoluttverdien av den overførte farten bestemme farten til viftemotoren. Bruk:*

     ```
     abs(); // Returnerer absoluttverdien av argumentet
     ```

   - *Kall funksjonen* `motor(retning,fart)` *med de variablene du har valgt.*

6. **Skriv programmet og test**
   Skriv og test ut programmet oppkoblet til CoT.

7. **Skriv om motor()-funksjonen**
   Skriv om funksjonen slik at omformingen av fortegnet til farten gjøres om til rotasjonsretning til motoren. Positiv fart (0 – 255) skal blåse lufta inn (retning = IN), negativ fart (–255 – 0) skal blåse lufta ut (retning = OUT).

8. **Rediger programmet og test**
   Test at programmet med den endrede funksjonen fungerer som forventet.

7.8 Sammenstill måling av luftfuktighet med styring av ventilasjonsvifta

Læringsmål

Oppdraget trener følgende læringsmål:

- Sammenstilling av målinger fra sensorer og kontrollsignaler fra CoT
- Bruk av terskelnivåer som del av styrefunksjonen

Vi skal i dette oppdraget kombinere sensormålinger, terskelverdier og styring av vifta.

Deloppdrag 22

Lag et program som leser av temperatur og luftfuktighet i rommet og viser resultatet av målingene i panelet på CoT. Sett terskelverdi for påslag av ventilasjonsvifta. Når luftfuktigheten i rommet overskrider terskelverdien for luftfuktighet, skal vifta blåse luft ut av rommet med en

manuelt satt viftefart. Når derimot luftfuktigheten er under terskelverdien skal vifta blåse luft inn i rommet.

Kort omtale

Vi antar i dette tilfellet at temperatur og luftfuktighet måles inne i et rom og at ventilasjonsvifta enten kan blåse luft inn eller ut av rommet styrt av sensormålinger.

Oppkobling

Vi beholder oppkoblingen fra oppdrag 20, men vi kan være oppmerksom på at når vi kombinerer en følsom sensor og en motor på samme spenningsforsyning, så kan vi oppleve at motoren lager problemer for sensoren. Det kan derfor være nødvendig med en avkobling over sensor spenningen. En avkobling gjøres med en kondensator. Det er ikke uvanlig at den avkobles med en stor elektrolytt kondensator for å ta de langsomme variasjonene og en mindre keramisk kondensator for å "kvele" de raske støypulsene. Det samme gjelder for H-broa.

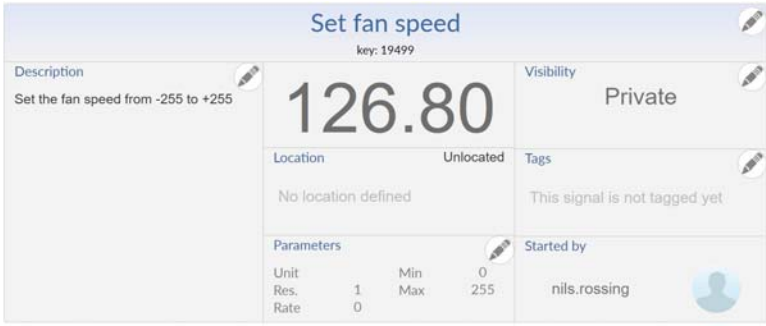
Konsoll

Vi skal endre litt på konsollene vi brukte i forrige deloppdrag.

1. **Signal for å sette terskelverdi**
   Vi endrer dette konsollet slik at det framgår at det handler om å sette terskelverdien for luftfuktighet (og ikke temperatur). Dessuten endrer vi nedre og øvre grense for terskelverdien til 20 – 60%.

   Det nye konsollet kan da f.eks. bli seende slik ut:

2. **Signal for å sett viftefart**
   Vi ønsker dessuten å kunne sette viftefarten manuelt fra 0 til 255. Siden det er luftfuktigheten som bestemmer vifteretningen så har vi ikke behov for å sett viftefarten fra -255 til +255.



3. **Panelet**
   Panelet får da følgende utforming:

   Husk å gå inn i konsollene for terskelverdi og viftefart å endre øvre og nedre grenseverdi.

Da skal konsollene være oppdatert. Så la oss se på programmet.

Programmering

Vi kan med fordel kombinere programmet fra deloppdrag 18 (“Automatisk fjernstyring av releet” på side 140) og deloppdrag 21 (“Styring av farten til vifta fra CoT” på side 148). Det kan også være nyttig og se til noen av de andre oppdragene.

1. **Inkluder biblioteker**
   Vi må inkludere både biblioteket for ESP01 og DHT11

```
#include "DHT.h"
#include <CircusESP01Lib.h>
```

2. **Konstanter for DHT11**
   Vi må også definere noen konstanter som vi skal bruke når vi definerer objektet for DHT11

```
#define DHTPIN 8        // Digital port koblet til DHT11
#define DHTTYPE DHT11   // DHT 11
```

3. **Deklarasjon av variabler for CoT**
   Som vanlig skal vi definere en rekke variabler som er nødvendig for å deklarere objektet som skal opprette og holde kontakten med Circus of Things. Denne brukes uforandret fra tidligere oppdrag. Her vil vi barne nevne de fem nøkkelkodene som *vi* bruker:

```
char relaySignalKey[] = "31332"; // Nøkkelkoden til styring av rele
char humiditySignalKey[] = "8135"; // Nøkkelkoden til luftfuktighet
char temperatureSignalKey[] = "10513"; // Nøkkelkoden til temperatur
char tresholdSignalKey[] = "17726"; // Nøkkelkoden til terskelverdi
char fanSpeedSignalKey[] = "19499"; // Nøkkelkoden til viftefart
```

   Her må man passe på å bruke de aktuelle nøkkelkodene som defineres når man oppretter signalene i "Workshop" hos CoT

4. **Deklarasjon av objekt for CoT**
   Som nevnt i forrige deloppdrag så må vi deklarere et objekt for å kunne kommunisere med CoT. Vi har valgt å kalle objektet `circusESP01` som er av typen `CircusESP01Lib`:

```
CircusESP01Lib circusESP01
(&ss,esp01BaudRate,server,token,ssid,password,debugLevel,enableSSL);
```

   Som vi ser så inneholder argumentet (...) de parametrene vi tidligere har definert under punkt 3. Deklarasjonen plasseres *før* void setup().

5. **Deklarasjon av objekt for DHT11**
   Vi må heller ikke glemme å deklarere et objekt for sensoren vår, DHT11

```
DHT dht(DHTPIN, DHTTYPE);
```

   Som vi har kalt dht. Her forteller vi hvilken pinne signalet vil komme på og hvilken type sensor vi bruker (DHT11, siden biblioteket brukes også for DHT22).

6. **Applikasjonsspesifikke variabler og konstanter**
   Her kan vi deklarere andre globale variabler som vi ønsker å bruke.

```
float H;           // Variabel som holder luftfuktighet
float T;           // Variabel som holder temperatur

int pinEn1  = 10;  // Styrer farten på motoren med PWM
int pinInp1 = 11;  // Styrer retningen på motoren
int pinInp2 = 12;  // Styrer retningen på motoren

const int OUT = 0; // Definerer luft ut
const int IN = 1;  // Definerer luft ut
```

   Her står en selvfølgelig fritt til å velge navn på variablene, og også hvilke variabler en velger å gjøre globale og hvilke man velger som lokale.

7. **void setup()-funksjonen**
   I setup-funksjonen initialiserer vi sensoren og kommunikasjonen med CoT med følgende to kommandoer:

```
circusESP01.begin();// Initiering av CoT biblioteket
dht.begin();        // Initialiser DHT11
```

   Vi må heller ikke glemme å definere porter som inn- og utganger.

8. **float readTempHumidSendCoT()**
   Vi velger å lage en egendefinert funksjon som vi har valgt å kalle `float readTempHumid-SendCoT()`. Som navnet sier så leser den temperatur og luftfuktighet fra sensoren DHT11 og sender data til CoT. Legg også merke til at funksjonen er av typen `float` hvilket betyr at den returnerer en `float` variabel. I dette tilfellet luftfuktigheten. Denne funksjonen vil vi senere kalle fra loop()-funksjonen:

```
float  H = dht.readHumidity();          // Returner avlest lufttrykk i mBar
float  T = dht.readTemperature();       // Returner avlest temperatur
circusESP01.write(humiditySignalKey, H); // Sender luftfuktighet H til CoT
circusESP01.write(temperatureSignalKey, T); // Sender temperatur T til CoT
return H; // Returnerer luftfuktighet fra funksjonen
```

   Funksjonen kan f.eks. legges til slutt i fila. Egendefinerte funksjoner kan i prinsippet legges hvor som helst bare at de legges på utsiden av setup() og loop()

9. **void motor(int retning, int fart)**
   Vi beholder også funksjonen som styrer motoren, vi legger merke til at vi har valgt å holde fast på *to* argumenter, ett for å styre retningen og ett for farten, da dette syntes mest hensiktsmessig her:

```
analogWrite(pinEn1, fart);
digitalWrite(pinInp1, retning); // Kjør vifte
digitalWrite(pinInp2,!retning); // Kjør vifte
```

   Også denne funksjonen kan f.eks. legges til slutt i fila.

10. **void loop()-funksjonen**
    Vi ønsker å bruke en overført parameter til å styre motorfunksjonen. Denne parameteren vil vi gjerne bruke til å slå av motoren (`OFF`), slå den på (`ON`) eller la den bli styrt av om luftfuktighetene er over eller under terskelnivået (`AUTO`).

    Det første vi må gjøre er å hente inn all nødvendig informasjon fra sensorene og fra CoT:

```
float humid = readTempHumidSendCoT();// Les av og sende luftfuktighet og temp.
int   fanSpeed = circusESP01.read(fanSpeedSignalKey); // Hent viftehastighet
float tresh = circusESP01.read(tresholdSignalKey); // Hent terskelverdi
int   onOffAuto = circusESP01.read(relaySignalKey); // Hent On/Off/Auto
```

    I tillegg så kan det være at vi trenger å definere noen lokale variabler

```
int R; // Motor retning
int Speed; // Motorfart som sendes til motorfunksjonen
```

Vi skal nå bruke informasjonen i variabelen `onOffAuto` (1/0/2) til å bestemme om vi skal slå på ventilasjonsvifta, slå den av eller la luftfuktigheten bestemme om den skal sende lufta ut (luftfuktighet over terskelverdi) eller trekke luft inn (luftfuktighet over terskelverdi). Dessuten skal `fanSpeed` brukes til å sette hastigheten til vifta når den er på.

Her overlater vi til dere å skrive koden.

11. **Skriv og test programmet**
    Skriv programmet og test om det fungerer slik som forutsatt.

7.9 Værstasjon

Nå har vi fått oversikt over mange aktuelle sensorer som kan anvendes i forbindelse med en værstasjon. Dessuten har vi lært hvordan vi kan overføre og vise sensordata i konsoller hos nettstedet CoT. Vi har også sett på styring av et par aktuatorer (rele og motor) og hvordan disse kan styres fra nettstedet CoT. Dette er nyttig kunnskap selv om aktuatorer ikke vanligvis brukes ved en værstasjon. Vi har nå det vi trenger for å lage en avansert værstasjon som kan fjernavleses.

7.9.1 Velg en sensor til værstasjonen

Vi skal i dette oppdraget utfordre dere til gradvis å bygge opp en værstasjon med ulike sensordata. Dere har gjennomgått en rekke sensorer og vi har vist både hvordan resultatene kan vises med ulike konsoller hos Circus of Things.

Deloppdrag 23

Velg ut en aktuell sensor som du ønsker at værstasjonen din skal inneholde. Gå til det aktuelle deloppdraget og koble opp og lag programmet. Koble sensoren opp mot CoT og lag et passende konsoll og et panel som passer for å presentere den målte størrelsen.

Ev. suppler med flere sensorer etter ønske.

Kort omtale

Ved bruk av Arduino UNO så støter man raskt på et problem ved at kortet har lite dynamisk minne (2kByte). Variabler og installasjon av flere biblioteker tar plass i det dynamiske minne. Arduino UNO og Arduino MEGA2560 har to forskjellige mikrokontrollere som tilbyr litt forskjellige lagerkapasitet, se under.

I tradisjonelle Arduino mikrokontrollerkort så er det i hovedsak tre typer lager:

1. **"Flash memory":** Her lagres programmet som blir værende selv om strømmen brytes. UNO har 32 kByte og MEGA har 256 kByte flash minne
2. **SRAM** (Static Random Access Memory): Her opprettes og behandles variabler mens programmet kjører. UNO har 2 kByte og MEGA har 8 kByte
3. **EEPROM** (Electronic Erasable Programable Read Only Memory): Dette er et lite lager der programmet kan lagre en mindre mengde data som ikke slettes når strømmen brytes. UNO har 1 kByte og MEGA har 4 kByte

Som vi ser så skal det ikke så mye til for å fylle opp SRAM hos UNO'en. En bør derfor unngå følgende når det ikke er strengt tatt nødvendig:

- Karakterstrenger som ikke skal forandres mens programmet går bør heller tvinges til å legges i Flash minne ved hjelp av kommandoen PROGMEM. Dette gjelder f.eks.:
  char ssid[], char password[], token[], server[] og SignalKeys[] osv.
  Vær oppmerksom på at bruk av PROGMEM også krever ekstra tiltak når man leser fra Flash minne, se https://www.arduino.cc/reference/en/language/variables/utilities/progmem/
- Det samme gjelder dersom man har behov for å lagre større tabeller med tall eller karakterstrenger.
- En bør også passe på å ikke bruke unødig resurskrevende variabler dersom det ikke er nødvendig:
  char = 1 byte (karakter)
  byte = 1 byte (heltall)
  int = 2 byte (heltall)
  long = 4 byte (heltall)
  float = 4 byte (desimaltall)

Man kan også velge Arduino UNO Wi-Fi, denne har et dynamisk minne på 6 kbyte.

Oppkobling

Gå til den aktuelle sensoren og studer hvordan den skal kobles opp. Dersom flere sensorer skal kobles opp så må man selvfølgelig plassere komponentene på koblingsbrettet slik at de ikke kommer i konflikt med hverandre.

Det er også nyttig å lage seg en oversikt over hvilke porter som brukes til de enkelte sensorene slik at man kan utnytte ressursene til mikrokontrollerkortet best mulig. Se Tabell 1: side 68.

Signaler, konsoller og panel

Bestem hvilke signaler som trengs for å betjene det utvalget av sensorer og aktuatorer som er valgt. Vær klar over at hver konto hos CoT har 10 signaler til rådighet. Dersom man trenger flere kan man ta kontakt med Jaume ved CoT på e-post: info@circusofthings.com

Det er altid lurt å bygge opp større prosjekter gradvis, ved at man tar for seg modul for modul, bygger, programmerer og tester ut før man inkluderer neste modul.

Programmering

Dersom man føler seg usikker på hvordan man skal programmere de ulike sensorene så er det bare å gå tilbake i heftet og se etter.

7.10 Presentasjon av målinger over tid

CoT gir muligheter for å vise hvordan sensordataene har endret seg over tid. I dette avsnittet skal vi vise et eksempel på hvordan vi kan plotte måledataene våre som tidsserier.

Læringsmål

Oppdraget trener følgende læringsmål:

- Hvordan man kan bruke CoT til å vise hvordan signaler utvikler seg over tid

Deloppdrag 24

> *Lag et nytt panel som viser hvordan temperaturen endrer seg over tid. Legg til luftfuktighet eller lufttrykk i samme diagram.*

For å få til dette må vi ha noen data å vise fram. Det oppnår vi ved å sende data til CoT over noe tid. Her lar vi temperatur, luftfuktighet og lufttrykk være eksempler, men prinsippet kan vi bruke hvilke som helst data som samles inn over tid. Du kan samle inn data over noen timer, gjerne et sted temperaturen endrer seg.

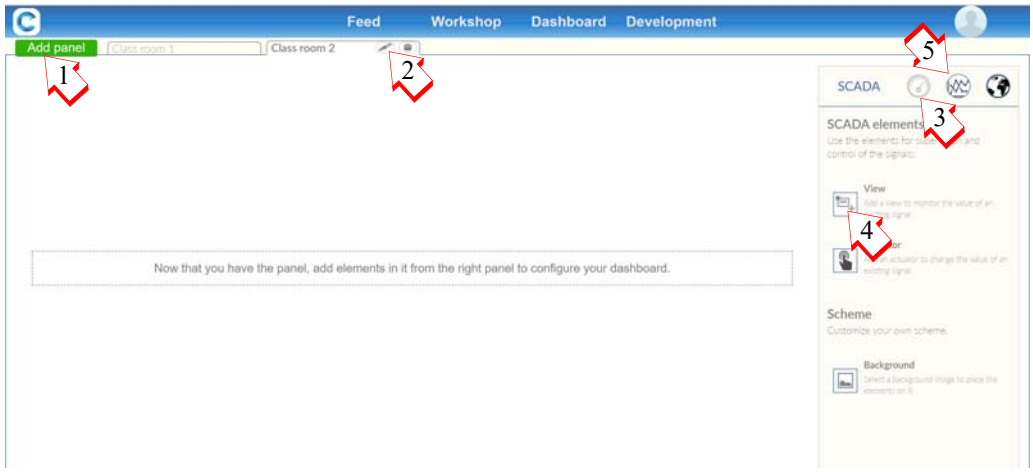
Oppkobling

Til dette oppdraget bruker man den oppkoblingen som trengs for å framskaffe ønskede data.

Kontrollpanelet

Alt vi skal gjøre i dette oppdraget gjør vi i Circus of Things.

1. **Logg på:** Logg på Circus of Things på vanlig måte.
2. **Opprett nytt panel:** Gå til *Dashboard* og velg *Add panel (1)* øverst til venstre. Velg blyanten til høyre for *New Panel (2)* og gi det et passende navn. Pass på at du befinner deg i *SCADA mode* (3). Velg så *View* (4) og hent opp konsollet ditt som viser temperatur.



3. **Vis tidslinje:** Nå er det på tide og velge å vise *Time line* (5, over) og så bestemme seg for tidsrommet du ønsker å vise i diagrammet.

Tidsperioden velges til høyre for panelet (6) som viser tidslinjen for, i dette tilfellet, temperaturen. Vi velger også å krysse av for *Y axis – Fit to values* (7), alternativt kan man sette et spesifikt temperaturområde for y-aksen.

Figuren over viser to måleserier tatt i en hytte og spenner tilsammen over to døgn. Til venstre fyres det opp i ovnen, mens til høyre stiger temperaturen jevnt ut over formiddagen. I perioden mellom er sensornoden avslått. "Spikeren" til høyre skyldes oppvarming med fingeren på sensoren.

4. **Foreta egne målinger:** Gjør egne målinger over noe tid og forsøk å forstå hva endringene skyldes.

5. **Bruk flere sensorer:** Gå tilbake til *SCADA mode* og se hva som skjer dersom du også henter fram konsollet for luftfuktighet (ev. en annen tilgjengelig måling).

6. **Les av verdier langs kurven:** Ved å flytte musa langs kurvene, kan du lese av verdiene ved hvert tidspunkt.

Programmering

Det trengs ikke gjøres noe med programmet i mikrokontrolleren (sensornoden) i dette oppdraget.

7.11 Inkluder målinger fra andre publiseringskilder

Vi har tidligere nevnt at det kan være aktuelt å hente måledata fra andre elevgrupper for å bygge opp et mer fullstendig panel (avsnitt 1.3 side 13). I så fall må vi finne ut hvordan vi kan få tilgang til data fra andre brukere av CoT.

Skal vi være istand til å bruke andres data forutsetter det at disse har gjort dataene sine tilgjengelige for allmennheten. La oss derfor først se hvordan vi kan gjøre målingene vår tilgjengelig for andre.

Læringsmål

Oppdraget trener følgende læringsmål:

- Gjøre egne signaler allment tilgjengelig og inkludere andres signaler i eget panel

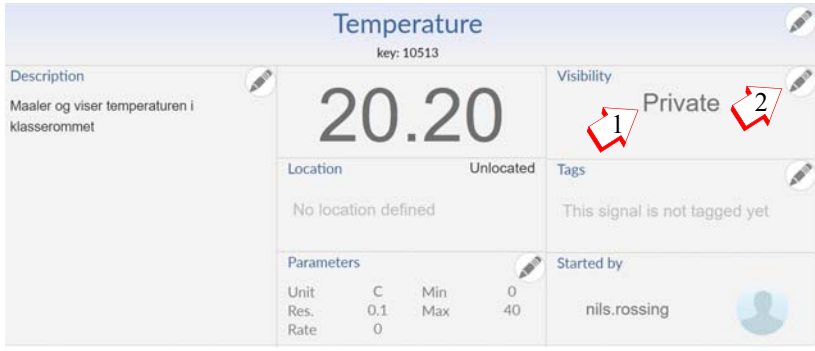
Deloppdrag 25

> *Publiser signalene dine slik at andre får tilgang til dem. Lag et panel som viser ditt eller dine signaler og inkluder signaler fra andre.*

Konsollet

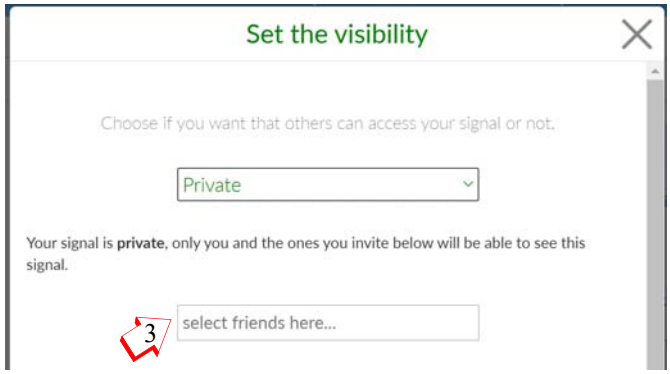
Vi tenker oss i en klasse er det flere elevgrupper som samler inn data. Vi ønsker at den enkelte elevgruppe skal gjøre sine data tilgjengelig for de andre gruppene. Dette kan gjøres slik:

1. **Gjøre ditt signal tilgjengelig for andre**
   Gå til workshop og hent opp det aktuelle signalet som ønskes delt med de andre som vist på figuren under.



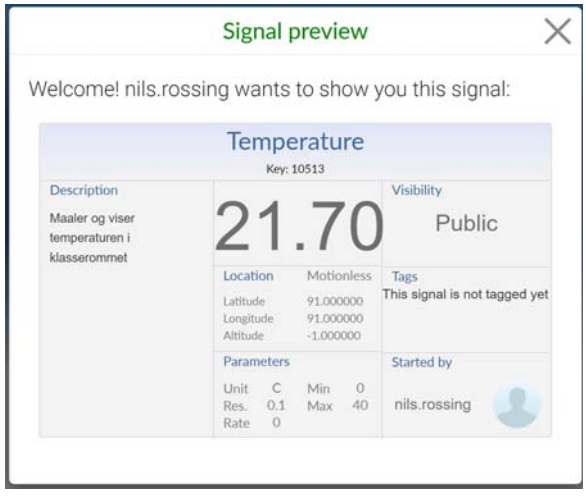
   Signalet er "Private" (1) dersom vi ikke aktivt går inn å endrer på dette. La oss velge blyanten øverst i høyre hjørnet av feltet (2). Et nytt vindu åpnes som gir oss muligheter til å dele signalet med andre. Velger man fortsatt "Private" så kan man skrive inn e-postadressen til de man

ønsker å dele signalet med (3).



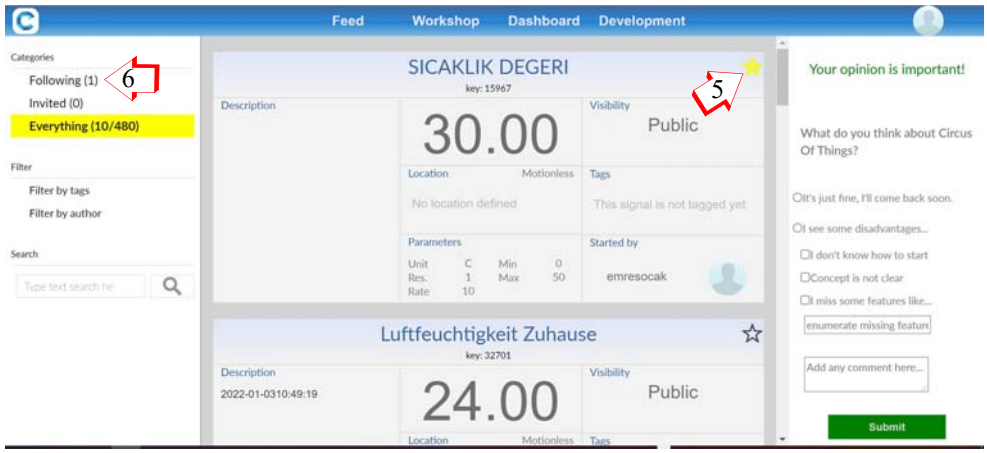
Den man inviterer til å følge signalet vil da få en melding om at signalet kan følges på en gitt nettadresse. Alternativt kan man velge "Public" som vist i figuren under:

Da legges signalet ut i "Feed'en" og er synlig for alle andre som bruker CoT. Man kan også tillate andre som ikke har en konto på CoT å se signalet ved å sende dem nettadressen: signal.circusofthings.com/10513 som blant annet inneholder nøkkelkoden til signalet (4). Ved å gå inn på adressen er de istand til å følge signalet via et vindu som kommer opp som vist på figuren til høyre.



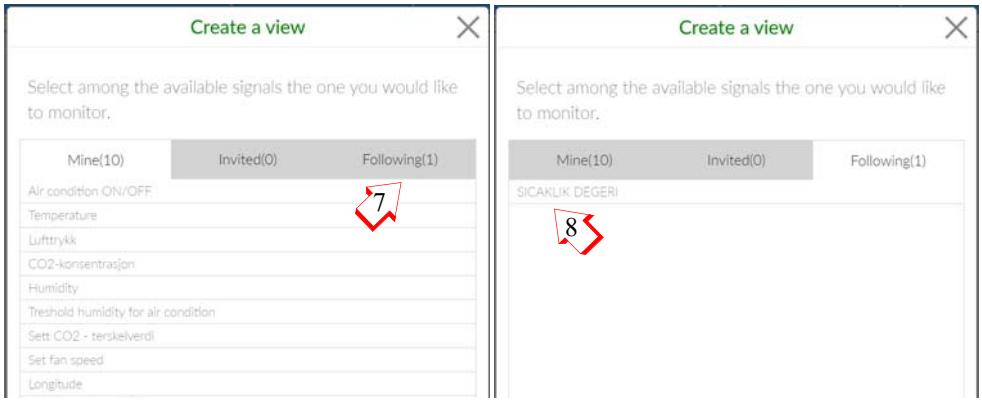
2. **Gjør andres signal tilgjengelig for deg**
   Når andre elevgrupper har gjort signalene sine alment tilgjengelig, "Public", så kan andre inkludere disse i sine paneler. La oss se hvordan vi gjør det.

   Når signalet er publisert så blir det liggende i "Feed'en" til Circus of Things. Ved å velg "Feed" får man opp en liste over alle publiserte signaler og kan søke på de som er lagt ut eller finne dem ved å bla seg gjennom lista.



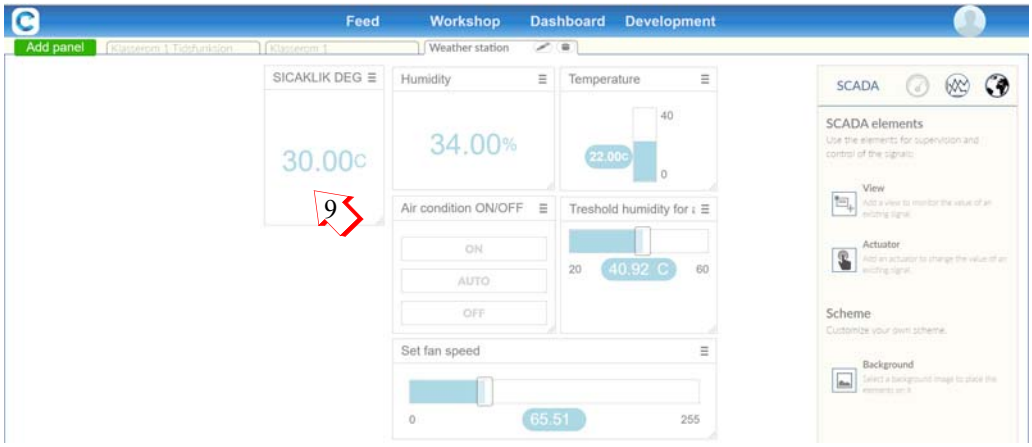
   Når man har funnet signalet man ønsker å vise, velger man å følge dette signalet ved å markere stjernen øverst i høyre hjørne (5), som blir gul når man klikker på den. Signalet blir dermed et signal man ønsker følge og det legger seg inn i lista over "Following(1)" (6).

3. **Inkluder signaler fra andre inn i ditt panel**
   Når man har funnet alle signalene man ønsker å følge, går man inn i panelet sitt og velger "View" og får da opp vinduet "Create a view", der vi automatisk kommer inn i "Mine(10)", mine signaler. Vi har imidlertid to andre faner: "Invited(0)" og "Following(1)" (7).



   Velger vi fanen "Following(1)" får vi opp vinduet til høyre på figuren over, der vi finner igjen signalet vi valgt å følge. Velger vi det kommer det opp i panelet vårt og blir et signal vi får vist i panelet vårt sammen med våre egne signaler.

Figuren under viser våre egne konsoller sammen med konsollet til signalet vi har valgt å følge (9).



På denne måten kan elevgrupper dele signaler og sammen bygge opp innholdsrike paneler etter eget ønske. Hver enkelt elevgruppe kan dermed bygge opp sitt panel, med det innholdet som er tilgjengelig.

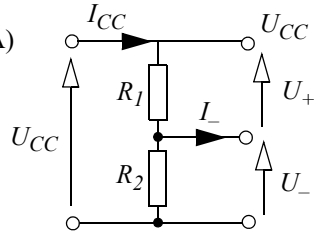
8 Utdypende teori om noen utvalgte komponenter

I dette kapittelet skal vi omtale noen av komponentene vi bruker litt mer uttømmende enn i oppgavesamlingen.

8.1 Spenningsdeleren

En spenningsdeler er nyttig for å etablere spenningsnivåer som normalt ikke er tilgjengelig fra strømforsyningen, men også for å konvertere en *variasjon i en motstandsverdi*, f.eks. i en LDR (lysfølsom motstand), til en *spenningsvariasjon* som mikrokontrollere kan registrere på en av sine analoge innganger.

A)



B)

$$U_- = \frac{R_2}{R_1 + R_2} U_{CC}$$

Ukjent R_2

$$R_2 = \frac{U_- \cdot R_1}{U_{CC} - U_-}$$

Ukjent R_1

$$R_1 = \frac{\langle U_{CC} - U_- \rangle \cdot R_2}{U_-}$$

Dersom vi har en spenning U_{CC} (f.eks. batterispenningen), så kan vi ved hjelp av to motstander ta ut en mindre del av denne spenningen. Tegning A i figuren over viser en enkel spenningsdeler. Spenningen U_- vil være en neddeling av spenningen U_{CC}, bestemt av motstandsverdiene til R_1 og R_2.

Vi ønsker å finne sammenhengen mellom U_{CC} og U_- som funksjon av R_1 og R_2.

Vi antar at $I_- = 0$, dvs. spenningsdeleren er ubelastet, hvilket er en god tilnærmelse dersom den kobles til en av inngangene til en mikrokontroller. Vi kan da sette opp en sammenheng mellom strømmer og spenninger i kretsen:

$$U_{CC} = (R_1 + R_2)\, I_{CC} \tag{8.1}$$

$$I_{CC} = \frac{U_{CC}}{(R_1 + R_2)} \tag{8.2}$$

Siden $I_- = 0$, vet vi at hele I_{CC} går gjennom R_2. Vi kan da bruke Ohms-lov på R_2 og finner da et uttrykk for U_- som er spenningen over R_2:

$$U_- = R_2 \cdot I_{CC} = \frac{R_2}{R_1 + R_2} U_{CC} \tag{8.3}$$

Tilsvarende kan vi finne U_+ som er spenningen over R_1:

$$U_+ = R_1 \cdot I_{CC} = \frac{R_1}{R_1 + R_2} U_{CC} \qquad (8.4)$$

Vi vet også at:

$$U_{CC} = U_- + U_+ \qquad (8.5)$$

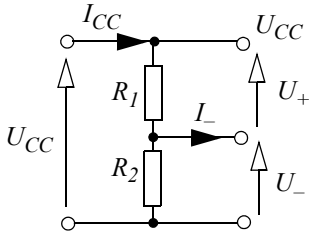
En vanligere situasjon er at vi kjenner spenningen U_-, men ønsker å bestemme verdien til en av motstandene. Dersom vi kjenner R_1 og U_-, men ønsker å bestemme R_2, kan vi bruke følgende uttrykk:

$$R_2 = \frac{U_- \cdot R_1}{U_{CC} - U_-} \qquad (8.6)$$

Dersom vi kjenner R_2 og U_-, men ønsker å bestemme R_1, kan vi bruke følgende uttrykk:

$$R_1 = \frac{\langle U_{CC} - U_- \rangle \cdot R_2}{U_-} \qquad (8.7)$$

På tilsvarende måte kan vi bestemme resistansene når vi kjenner U_+.



Spenningsdeleren

La oss se hvordan vi kan få en intuitiv forståelse av hvordan en spenningsdeler fungerer.

Fra Ohms lov vet vi at:

$$U_+ = R_1 \cdot I_{CC} \qquad (8.8)$$

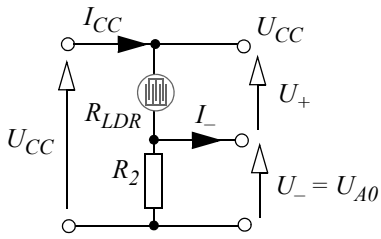
og

$$U_- = R_2 \cdot I_{CC} \qquad (8.9)$$

Siden strømmen er den samme i begge motstandene, så kan vi dividere de to ligningene med hverandre og skrive:

$$\frac{U_+}{U_-} = \frac{R_1}{R_2} \qquad (8.10)$$

Vi ser altså at forholdet mellom spenningene er forholdet mellom resistansene til de to motstandene.



Spenningsdeleren med LDR

I figuren til venstre har vi byttet ut R_1 med en resistiv sensor, i dette tilfellet en lysfølsom motstand

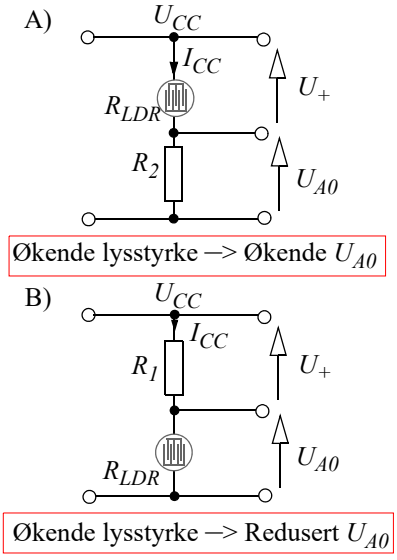
Vi tenker oss at U_- kobles til en analog inngang på mikrokontrolleren vi kaller den U_{A0} siden vi kobler den til den analoge inngangen *A0*.

Fra ligning 8.3 vet vi at forholdet mellom spenningen U_{A0} og motstandsverdien til LDR'en, R_{LDR} kan uttrykkes slik:

$$U_- = \frac{R_2}{R_{LDR} + R_2} U_{CC} \qquad (8.11)$$

som vi ser så er det ikke nødvendigvis en lineær sammenheng. Men fra ligning (8.7) så vet vi at vi kan regne oss tilbake å finne R_{LDR} med følgende omregning:

$$R_{LDR} = \frac{\langle U_{CC} - U_{A0} \rangle \cdot R_2}{U_{A0}} \qquad (8.12)$$



Spenningsdeleren med LDR

Ofte holder det at vi forstår rent kvalitativt hvordan spenningsdeleren fungerer da vi gjerne bruker LDR i en detektor.

Vi antar at *lysstyrken øker*. Det betyr at motstandsverdien til R_{LDR} *blir mindre.* Når R_{LDR} blir mindre, vil strømmen I_{CC} øke. Siden den samme strømmen går gjennom R_2 forteller Ohms lov oss at spenningen U_{A0} *øker* ($U_{A0} = I_{CC} R_2$).

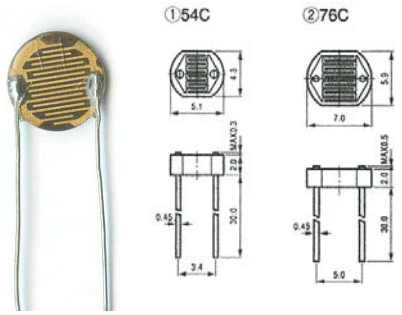
Vi kan imidlertid tenke oss en alternativ plassering av LDR'en ved at den erstatter R_2 istedet for R_1.

Dersom vi nå antar at *lysstyrken øker* og motstandsverdien til R_{LDR} *blir mindre.* Så vil strømmen I_{CC} øke som sist. Siden I_{CC} også går gjennom R_1, forteller Ohms lov oss at spenningen U_+ også vil øke. Siden $U_{A0} = U_{CC} - U_+$ så vil U_{A0} *avta*.

8.2 Lysfølsom sensor – LDR

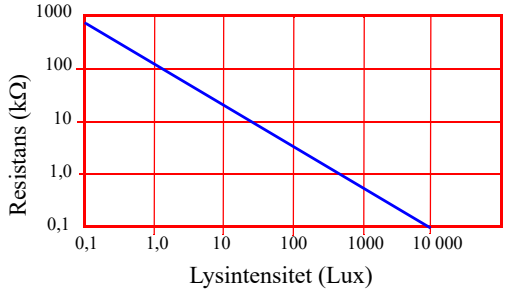
Deteksjon av lys kan gjøres på mange ulike måter. I dette avsnittet skal vi se hvordan vi kan bruke *LDR* (Light Dependent Resistor) som lysfølsomme komponenter.

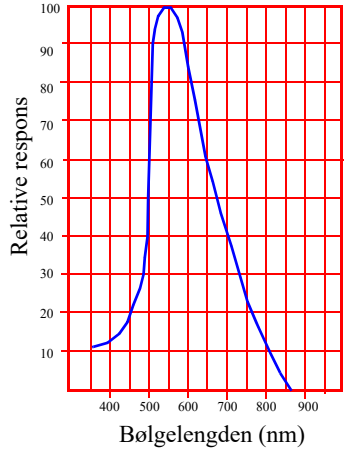
8.2.1 Fotomotstand (LDR - Light Dependent Resistor)



Grunnen til at vi velger å omtale fotomotstander er at den gjennom tidene har vært en gjenganger i mange elektronikkprosjekter og brukes i mange sammenhenger som f.eks. for automatisk tenning av gatebelysning, som enkle lysmålere, som automatisk tenning av frontlyktene på bil, innbruddsalarm ved at en lysstråle brytes o.l.

Fotomotstander har tradisjonelt vært laget av Cadmium-Sulfid (CdS) belagt med fingerelektroder som vist på figuren til høyre. I mørket vil stoffet CdS være omtrent isolerende og kan gi en motstand på over 1 MΩ. Belyses stoffet, kan resistansen i fotomotstanden falle til under 1 kΩ. Årsaken er at fotoner (lys) med tilstrekkelig energi, eksiterer elektroner fra valensbåndet til ledningsbåndet, hvor de kan bevege seg fritt og bidra til ladningstransporten. Effekten er imidlertid ikke like framtredene for alle frekvenser. Til høyre på figuren under ser vi at materialet er spesielt følsomt for lys i det synlige området av spekteret nær 540 nm (nanometer, 10^{-9}m). Vi ser også (til venstre på figuren) at det er en omtrent lineær sammenheng mellom lysstyrken målt i lux og resistansen (begge skalaer er logaritmiske). Økende lysstyrke gir fallende resistans, dvs. økt ledningsevne.





Resistans som funksjon av lysintensitet (venstre), følsomhet som funksjon av bølgelengde.

Lysfølsomme motstander er imidlertid relativt langsomme. En endring i lysstyrken på noen µsek, kan gi en responstid på opp til 100 msek. hos fotomotstanden, men som i mange tilfeller er mer enn godt nok. Bruker man fotomotstander bør man også være klar over følgende [8]:

- Resistansen for fotomotstander kan variere mye fra eksemplar til eksemplar. Dette gjelder spesielt resistansen ved mørke.
- Resistansen hos en LDR vil også være temperaturavhengig
- En LDR kan også brenne opp ved for store strømmer. Dette er noe en bør være oppmerksom på dersom den skal brukes i sterkt sollys. Under slike forhold vil motstanden fall og strømme øke. I slike tilfeller må man sørge for en tilstrekkelig høy seriemotstand slik at strømstyrken og dermed også effekttapet holdes under verdier som kan være skadelige for LDR'en.

- Dersom sensoren går i metning ved sterkt sollys, forsøk å reduser serieresistansen (men pass på effekttapet). Ønsker man større følsomhet i det mørke området øker man serieresistansen

For å konvertere endring i resistans til spenning, kan vi bruke en enkel spenningsdeler (se figuren under). Her trengs normalt ingen målebro eller forsterker for å registrere endring i resistans siden endringen er så stor.

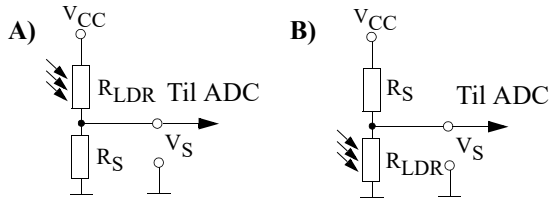
Lysintensitet måles i lux. 1 lux er 1 lumen pr. m^2.

Dette tilsvarer:

- Fullt sollys 11 000 lux (eller ca. 1000 W/m^2)
- Sollyset en tidlig morgen 6 000 lux
- Belysningen i et TV-studio 1 000 lux
- Et godt opplyst kontor 400 lux
- Lyset fra en fullmåne 1 lux

Oppkobling mot ADC

Siden grensnittet til kontrolleren krever en spenning, kobles LDR-motstanden i serie med en motstand som vist i figuren til høyre. Velg verdien på seriemotstanden lik den typiske resistansen til fotomotstanden (LDR) i det aktuelle lysintensitetsområdet der fotomotstanden skal brukes.



$$V_S = \frac{R_S}{R_{LDR} + R_S} V_{CC} \qquad V_S = \frac{R_{LDR}}{R_{LDR} + R_S} V_{CC}$$

Ønsker man maksimal spenningsvariasjon fra dypt mørke til sterkt lys kan seriemotstanden beregnes ut fra følgende ligning [8]:

$$R_S = \sqrt{R_{LDR(light)} \times R_{LDR(dark)}} \tag{8.13}$$

hvor $R_{LDR(light)}$ er resistansen i sterkt lys og $R_{LDR(dark)}$ er resistansen i mørke.

Spenningsnivået V_S beregnes fra formlene som antydet på figuren. Legg merke til at oppkoblingen på tegning A gir økende spenning V_S med økende lysstyrke, mens oppkoblingen i tegning B gir fallende spenning med økende lysstyrke. For en utdypende diskusjon av spenningsdeleren se avsnitt 8.1.

Kalibrering:

Utfordringen blir å finne en omregningsformel fra lysstyrke til spenning:

1. Mål spenning som funksjon av lysstyrke (krever lysmåler)
2. Bruk regresjon for å finne et best tilpasset funksjonsuttrykk

3. Legg omregningsformelen inn i prosessoren

9 Referanser

[1] Essiane S.N., Essama B.G.O, Development of micro weather station using Arduino and Internet of Things, Physics Education, 2022

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

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

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

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

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

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

[8] Elliot Williams, *Make: AVR Programming – Learing to Write Software for Hardware*, Make Community, LLC; 1 edition (February 25, 2014)

[9] **Entråd kommunikasjonsbuss:**
https://www.maximintegrated.com/en/design/technical-documents/app-notes/7/74.html

[10] **Entråd kommunikasjonsbuss:**
https://www.maximintegrated.com/en/design/technical-documents/app-notes/7/74.html

Vedlegg A Komponentoversikt

Følgende komponenter trengs for å gjennomføre undervisningsopplegget i dette heftet for 16 deltakere som arbeider i grupper på to og to.

Måling	Komponent	Læringsmål	Tilgjengelighet	Pris	Status	Kalibrering
Temperatur	TC74	Halvleder temp. sensor I²C-bus, Kalibrering Installer bibliotek	ELFA 300-46-786	15,00	Kjøpes På lager	Termometer (Har)
Temperatur	NTC	NTC-motstand Topunktsligning linearisering Kalibrering	ELFA 301-71-534	2,5	Kjøpes På lager	
Luftfuktighet	DHT11	Luftfukt. sensoren En tråd-bus Installer bibliotek	ELFA 301-39-181	59,-	Kjøpes På lager	Har ikke
Lufttrykk	BMP280	Lufttrykksensoren I²C-bus Installer bibliotek	Mouser	109,-	Har	Har ikke
Lysstyrke	BH1750	Lysstyrkesensoren I²C bus, Kalibreres Installer bibliotek	Elkim	69,-	Kjøpes På lager	Har
	LDR	Spenningsdeler Vanskelig å kalibrere, men mer læring	ELFA[a] 301-39-054	11,58	Kjøpes På lager	
Nedbør	SEN-15901	Telling, Bruk av interrupt, Kalibrering	Mouser	853,-	Kjøpes På lager	Kan måle
Vindstyrke						Har
Vindretning		Spenningsdeler Bruk av array Switch … case Kalibrering				Har
Snødybde	HC-SRO4	Avlesning av sensor	ELFA 301-39-186	48,20	Har	Har
Display	SSD1306	I²C bus Installer bibliotek Koordinatsystem	Banggood	60,-?	Har	
NETT-Kobling	ESP01	Rx/Tx UART Oppkobling til nett	Mouser 474-WRL-17146	75,-	Kjøpes Har?	
	ESP01 sokkel	For montering	Banggood	10,-?	Har	
Spennings-regulator	LD1117 AV33	Spenningsregulering	ELFA 301-70-560	7,00	Har	
H-bro	L293B	Motorstyring	RS-Online 714-0613	58,70	Kjøpes	
Totalt				1377,98 (1150,55)	Inkl MVA	

a. https://www.elfadistrelec.no/en/cds-photoresistor-5v-adafruit-161/p/30139054

Komponentorganisering

På kurset bør komponentene deles ut på følgende måte:

Samling 1 – 1. dag

Hvert deltakerpar (maks 8 stk) får utlevert følgende

1 stk. Sparkfun Inventors kit
1 stk. Arbeidsbok – Lag en høydemåler
1 stk. BMP280 – Lufttrykk sensor
1 stk. SSD1306 – OLED display
1 stk. LED
1 stk. TMP36 – Temperatursensor

Samling 1 – 2. dag

Hvert deltakerpar (maks 8 stk) får utlevert følgende

1 stk. Sparkfun Inventors kit
1 stk. Arbeidsbok – Værstasjon
1 stk. BMP280 – Lufttrykk sensor
1 stk. SSD1306 – OLED display
1 stk. BH1750 – Lysmåler
1 stk. LDR
1 stk. TC74 – Temperatursensor
1 stk. HC-SR04 – Avstandssensor
1 stk. SEN-15901 – Værstasjon
1 stk. DHT11 – Luftfuktighet og temperatur
2 stk. RJ11 – kabel
2 stk. 2 polt koblingsplint

Samling 2 – 1. og 2. dag

Hvert deltakerpar (maks 8 stk) får utlevert følgende

1 stk. Sparkfun Inventors kit
1 stk. Arbeidsbok – Værstasjon
1 stk. DHT11 – Luftfuktighet og temperatur
1 stk. ESP01
1 stk. ESP01-sokkel
1 stk. USB-TTL 3,3 V
1 stk. Rele JQC-3FF-S-Z
1 stk. L293B – H-Bro
1 stk. LD1117AV33 – Regulator 5,0 - 3,3 V
1 stk. Tiny koblingsbrett
1 stk. Solcellemotor

Vedlegg B Endre konfigurasjon av ESP01

Når ESP01 leveres så ligger det alt et program i kretsen, vi kan kalle dette for kretsens firmware. Vi kan kommunisere med dette programmet ved hjelp av AT-kommandoer som gjerne har formen AT+<spesialkommando>. Vi skal bruke noen av disse kommandoene for å lese av MAC-adressen og sette hastigheten til kommunikasjonen mellom Arduino'en og ASP01. AT-kommandoene for ESP8266 (ESP01) finnes her:

http://room-15.github.io/blog/2015/03/26/esp8266-at-command-reference/

B.1 Oppkobling

CH340 Adapter

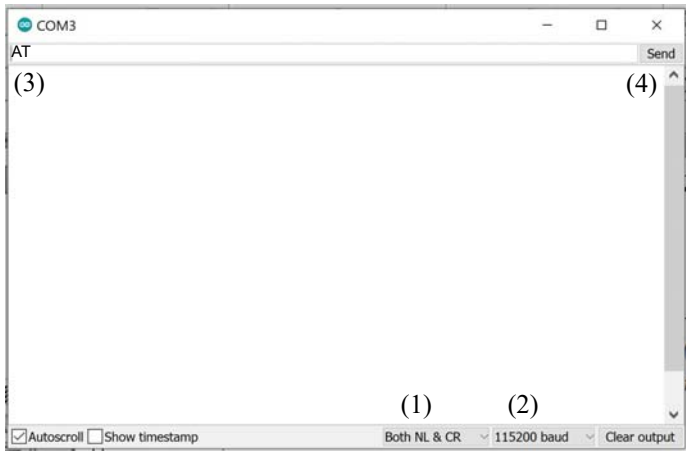
CH340 er en overgang fra USB-porten på PC'en til et signal som kan kommunisere med Tx/Rx portene hos ESP01. CH340 tilpasser også nivåene og forsynings-spenningen (3,3V/5,0V) slik at den passer til ESP01. Dessuten har den en sokkel som gjør det lett å koble til ESP01.

Pass på å sette ESP01 riktig vei ned i sokkelen, se figuren over til høyre.

B.2 Bruk av Arduino monitoren for å kommunisere med ESP01

Ved hjelp av AT-kommandoene kan vi undersøke hvordan programvaren i ESP01 er konfigurert ev. endre på konfigureringen, som f.eks. endring av kommunikasjonshastigheten.

PC'en må på sin side ha en programvare som kan sende AT-kommandoer til ESP01 via CH340. Det enkleste valget er å bruke Arduino IDE monitoren som vist på figuren til høyre. Den kan ikke bare motta informasjon *fra* USB-porten, men også sende informasjon *til* USB-porten og videre til den tilkoblede ESP01.



Vi skal nå sjekke forbindelsen med ESP01. Vi går da fram på følgende måte:

1. Plugg USB-adapteren med ESP01 inn i USB-porten på PC'en som vist på figuren til høyre.

2. Åpne Arduino editoren (IDE) med en tom skisse.
3. Velg rett port
   - *Husk å bytte til riktig COM-port ved å gå inn i menyen og velge Verktøy/Port (Tools/Port) og velg den porten som er tilkoblet adapteren.*
4. Sett opp monitoren:
   - *Sett linjeformatering til "Both NL & CR" (1)*
   - *Sett baudrate til 115 200 Baud, da ESP01 normalt er satt opp med denne hastigheten fra fabrikken.*
5. Sjekk kommunikasjonen
   - *Skriv AT (3) i kommunikasjonslinjen og trykk Send (4)*

   - *Dersom alt er OK skal det skrives ut AT og deretter OK (5)*

B.3 Endring av kommunikasjonshastigheten

Normalt vil ASP01 være satt opp med en kommunikasjonshastighet på 115 200 baud. Vi skal nå endre denne til 9 600 baud.

La oss se hvordan vi kan bruke monitoren hos Arduino-editoren til å endre kommunikasjonshastigheten mellom ESP01 og Arduino'en. Merk! Når vi gjør denne endringen så må vi neste gang vi kommuniserer med ESP01 bruke hastigheten 9 600 baud.

1. Endre kommunikasjonshastigheten til 9 600 baud[43]
   - *Skriv: AT+UART_DEF=9600,8,1,0,0*
     Om dette ikke fungerer, prøv:
     AT+CIOBAUD=9600
     Kommandoene skal skrives inn uten mellomrom.

43.https://create.arduino.cc/projecthub/ahmedibrrahim/iot-using-esp8266-01-and-arduino-afa35e

- *Du skal da få en respons i monitoren som vist på figuren over (6).*
- *Fra nå av må man kommunisere med ESP01 med 9 600 baud.*

B.4 Avlesning av MAC-adressen

Alle kretser som kan kobles opp mot nettet har en MAC-adresse (Media Access Control address), noen ganger også kalt Ethernet adresse. For å kunne koble upersonlige enheter opp mot laboratorienettet ved NTNU, så må enhetenes MAC-adresser registreres på forhånd. Vi må derfor lese av denne adressen og sende den over til nettadministrasjonen ved NTNU.

1. Koble opp kretsen som omtalt foran
2. Åpne monitorvinduet hus Arduino-editoren
3. Sett kommunikasjonshastigheten til riktig verdi (115 200 eller 9 600 baud avhengig hva den er innstilt på)
4. Skriv følgende kommando inn på kommandolinjen:

   AT+CIPSTAMAC?

   ESP01 vil da respondere med følgende:

   AT+CIPSTAMAC="18:aa:35:97:d4:7b"

   Hvor adressen selvfølgelig være en individuell adressen for din enhet, men strukturen er den samme som antydet over.

Vedlegg C Kalibrering

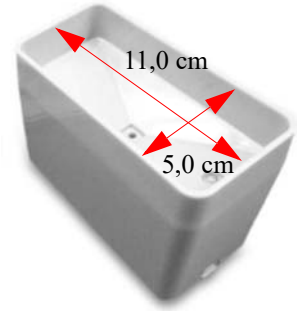
C.1 Kalibrering av regnmåleren

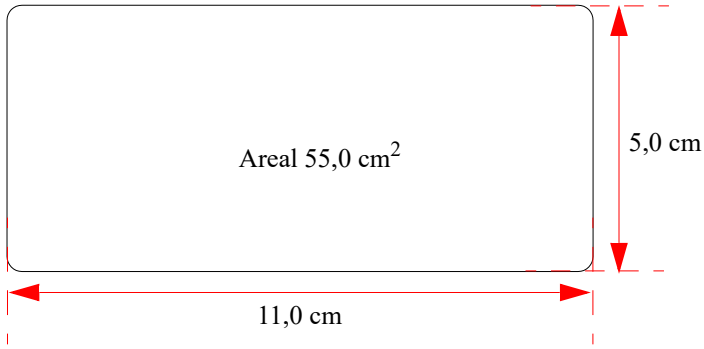
Det er sikkert flere måter å kalibrere regnmåleren på. Men følgende kan være en.

Vi tar utgangspunkt i den valgte regnmåleren som følger med settet SEN-15901.

Alternativ 1

Vi bruker en 50 mL sprøyte og fyller den med 55 mL vann.





Vi beregner hvor mange mm nedbør 55 mL vann tilsvarer for et areal på 55,0 cm^2. Vi vet at 55mL vann tilsvarer 55 cm^3 vann.

Nedbør [mm] = Vannmengde [cm^3] / Innsamlingsareal [cm^2] = 55 cm^3 / 55,0 cm^2 = 10 mm

Vi vet at produsenten oppgir at hver beholder er full og vil tippe etter at det er falt 0,2794 mm nedbør. Dermed kan vi beregne antall forventede tømminger og dermed antall registrerte pulser lik:

Antall tømminger = Nedbør [mm] / Nedbør pr. tømming [mm]
= 10 mm/ 0,2794 mm = 35,79 tømminger ≈ 35 hele tømminger

Når vi gjennomfører forsøket er det viktig at beholderne fylles tilstrekkelig langsomt slik at de ikke rekker å renner over før de tømmes. Dette skal være tatt hånd om av størrelsen på hullet i bunnen av trakta. Hullet har en diameter på 4 mm.

Alternativ 2

Alternativt kan man bruke en mer nøyaktige sprøyter, f.eks. på 10 mL og telle antall hele tømminger, for så å registrere antall mL vann som er gått med. På denne måten vil man finne antall mm nedbør i hver beholder som tømmes.

Nedbør pr. tømming [mm] = Nedbør [mm] / Antall tømminger

Nedbør pr. tømming [mm] = (Vannmengde [cm^3] / Innsamlingsareal [cm^2]) / Antall tømminger

Nedbør pr. tømming [mm] = (Vannmengde [cm^3] / 55,0 cm^2) / Antall tømminger

Nedbør pr. tømming [mm] = Vannmengde [cm^3] / (Antall tømminger x 55,0 cm^2)

Vi bestemmer oss for et antall registrerte tømminger (pulser) og måler den medgått vannmengden når siste tømming skjer for så å sette inn i formelen over. Dermed vil vi finne nedbør pr. tømming i mm.

Vedlegg D Datablad SEN-15901

Shenzhen Fine Offset Electronics Co., Ltd
深圳市欧赛特电子有限公司
Address: 4/F,Block C, Jiujiu Industrial City, Shajing Town, Bao'an District, Shenzhen City, China

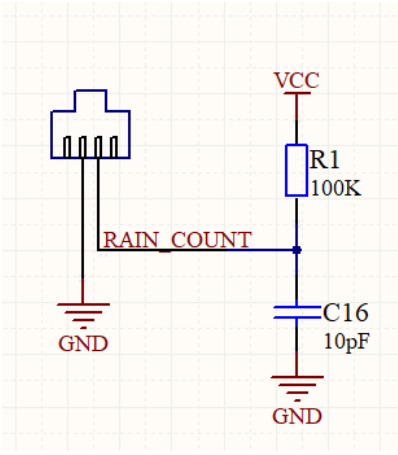
This kit includes a wind vane ,cup anemometer, and tipping bucket rain gauge, with associated mounting hardware .These sensors contain no active electronics, instead using sealed magnetic reed switches and magnets to take measurements. A voltage must be supplied to each instrument to produce an output.

Assembly

The wind sensor arm mounts on top of the two-piece metal and supports the wind vane and anemometer. A short cable connects the two win sensors. Plastic clips on the underside of the arm hold this cable in place. Screws are provided to secure the sensors to the arm. The rain gauge may be mounted lower on the mast using its own mounting arm and screw, or it may be mounted independently.

Rain Gauge

雨量计为翻斗式的，每次发生一次翻斗为 0.3mm 的雨量。当雨斗翻转时会造成干簧管的一次闭合，可通过微处理器中断输入来记录翻斗次数,在 RJ11 水晶头的一端需连接一个电阻电容，如图所示：



The rain gauge is a self-emptying tipping bucket type. Each 0.2794mm of rain cause one momentary contact closure that can be recorded with a digital counter or microcontroller interrupt input. The gauge's switch is connected to the two center conductors of the attached RJ 11-terminated cable.

Anemometer

杯式风速计测量风速的方法是当磁铁通过开关时关闭触点。风速 0.33m/s 导致开关关闭一次. 风速计将连接到风向计上，与风向计公用 RJ11 电缆，风速计占用 RJ11 水晶头的 2、3 脚。

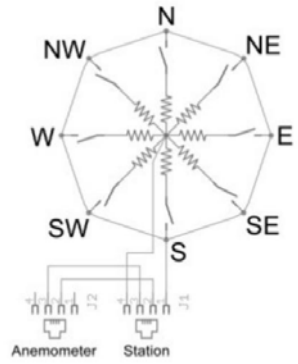
The cup-type anemometer measures wind speed by closing a contact as a magnet moves past a switch. A wind speed of 2.4km/h causes the switch to close once per second.

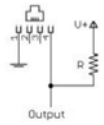
The anemometer switch is connected to the inner two conductors of the RJ 11 cable shared by

the anemometer and wind vane(pins 2 and 3)

Wind Vane

风向计它有八个开关，每个开关连接到一个不同的电阻器。叶片的磁铁可以同时关闭两个开关，使多达 16 个不同的位置被指示。外部电阻可以用来形成一个分压器，产生一个电压输出，可以用 a/d 转换器来测量，如下所示。





Example wind vane interface circuit. Voltage readings for a 5 volt supply and a resistor value of 10k ohms are given in the table.

Direction (degrees)	Resistance （ohms）
0	33K
22.5	6.57K
45	8.2K
67.5	891
90	1K
112.5	688
135	2.2K
157.5	1.41K
180	3.9K
202.5	3.14K
225	16K
247.5	14.12K
270	120K
292.5	42.12K
315	64.9K
337.5	21.88K

表中给出了所有 16 个可能位置的电阻值。

The wind vane is the most complicated of the three sensors. It has eight switches, each connected to a different resistor. The vane's magnet may close two switches at once, allowing up

to 16 different positions to be indicated . An external resistor can be used to form a voltage divider , producing a voltage output that can be measured with an analog to digital converter, as shown below.

The switch and resistor arrangement is shown in the diagram to the right. Resistance values for all 16 possible positions are given in the table.

Resistance values for positions between those shown in the diagram are the result of two adjacent resistors connected in parallel when the vane's magnet activates two switches simultaneously.

Vedlegg E Ressurser

E.1 I²C scanner for Arduino UNO

```
// ---------------------------------------------------------- /
// Arduino I2C Scanner
// Re-writed by Arbi Abdul Jabbaar
// Using Arduino IDE 1.8.7
// Using GY-87 module for the target
// Tested on 10 September 2019
// This sketch tests the standard 7-bit addresses
// Devices with higher bit address might not be seen properly.
// ---------------------------------------------------------- /

#include <Wire.h> //include Wire.h library

void setup()
{
  Wire.begin(); // Wire communication begin
  Serial.begin(9600); // The baudrate of Serial monitor is set in 9600
  while (!Serial); // Waiting for Serial Monitor
  Serial.println("\nI2C Scanner");
}

void loop()
{
  byte error, address; //variable for error and I2C address
  int nDevices;

  Serial.println("Scanning...");

  nDevices = 0;
  for (address = 1; address < 127; address++ )
  {
    // The i2c_scanner uses the return value of
    // the Write.endTransmisstion to see if
    // a device did acknowledge to the address.
```

```
    Wire.beginTransmission(address);
    error = Wire.endTransmission();

    if (error == 0)
    {
      Serial.print("I2C device found at address 0x");
      if (address < 16)
        Serial.print("0");
      Serial.print(address, HEX);
      Serial.println("  !");
      nDevices++;
    }
    else if (error == 4)
    {
      Serial.print("Unknown error at address 0x");
      if (address < 16)
        Serial.print("0");
      Serial.println(address, HEX);
    }
  }
  if (nDevices == 0)
    Serial.println("No I2C devices found\n");
  else
    Serial.println("done\n");

  delay(5000); // wait 5 seconds for the next I2C scan
}
```

Heftet er ment for bruk i videregående opplæring i Arduino-programmering for lærere på elektro yrkesfag. Undervisningsopplegget forutsetter at man har grunnleggende kunnskaper om Arduino-programmering.

Værstasjon er valgt som tema da måling av måledata knyttet til været synes å være noe mange elever er opptatt av. En værstasjon kan dessuten bygges opp gradvis fra kun å registrere temperatur til å registrere alle mulige egenskaper ved været. Det være seg temperatur, luftrykk, luftfuktighet, lysintensitet, UV-stråling, vindhastighet, vindretning, nedbør og snødybde med mer. Været er dessuten noe alle både har et forhold til og har tilgang til overalt i landet. Dessuten er været et kontinuerlig fenomen hvor man kan gjøre målinger over tid og sammenligne data fra en periode med en annen.

Målingene krever stor variasjon i måleteknikker fra temperaturmåling til måling av nedbør og snødybde. Noen av sensorene kan også lages fra bunnen av som f.eks. nedbørsmåling og måling av vindretning, f.eks. ved bruk av 3D-printing. Prosjekter knyttet til temaet kan derfor gjøres både små slik at de kan gjennomføres i løpet av et par timer, eller særdeles omfattende og strekke seg over uker og måneder. En ulempe er at sensorteknologien gjerne får en særdeles dominerende rolle, mens behovet for aktuatorer kan bli relativt beskjedent. I del to har vi derfor inkludert styring av rele og motorstyring.

Nils Kr. Rossing
Dosent ved Skolelaboratoriet
E-post: nils.rossing@ntnu.no
Prosjektleder ved Vitensenteret
E-post:nkr@vitensenteret.com

Rev 2.5 – 26.03.22 ISBN

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