



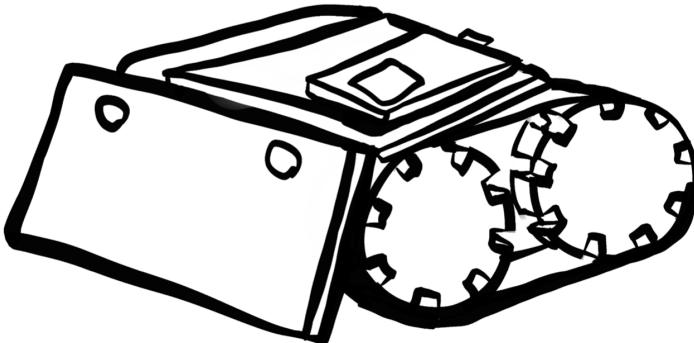
NTNU

Kunnskap for en bedre verden

BIELSYS, BIAIS OG BIELDIG

PROSJEKT I OPPSTARTSUKA

Robotbil



Introduksjon

Velkommen til faglig del i Oppstartsuka 2024 for studieprogrammene BIELSYS, BIAIS og BIELDIG!

Temaet er mikrokontrollerprogrammering med Pololu Zumo 32U4 OLED robot! Dette opplegget er en del av ditt bachelorprogram og er designet for å gi deg en praktisk og spennende start på studiet. I løpet av dette kurset vil du jobbe i grupper og gjennomføre en rekke utfordrende oppgaver som vil hjelpe deg å forstå grunnleggende konsepter innen styring av en drone ved å programmere ”hjernen”, også kjent som en mikrokontroller. Denne kan tenkes på som en mini datamaskin som utfører grunnleggende instruksjoner. For å styre den skal dere programmere i språket Arduino-C (ligner C++).

Ved å styre mini datamaskinen, kan dere lese av sensorer og gi pådrag til aktuatorer (eksterne komponenter, eksempelvis en elektromotor). Dere har nå fått ”skjelettet” til koden, og deres jobb er å bygge ut resten ved å fullføre en rekke **funksjoner**. På fredag blir det en konkurranser blant gruppene så da er det bare å kjøre på!

Vedlagt [her](#) nede i dokumentene er det forklaringer av hvordan språket Arduino-C (C++) skrives og tydes, hvordan du setter opp koden og litt om hva de forskjellige begrepene betyr. Dette anbefales å lese gjennom dersom du ikke har programert så mye fra før av, eller ikke har kjennskap til C++.

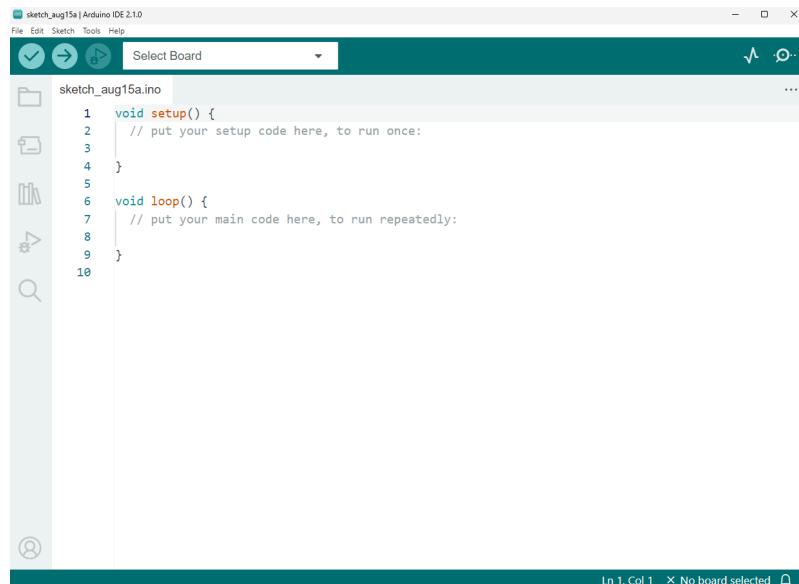
Til de av dere som kanskje har programmert i C++ litt før er det noen utfordringer lagt til i form av ekstraoppgaver. Dette er ment mer som noe å bryne seg litt over hvis man kommer fort gjennom opplegget og er derfor ikke viktig å bli ferdig med.

Da gjenstår det bare å si lykke til!

1 Installer programvaren og utforsk

Last ned Arduino IDE

Det første som må gjøres, for at alt annet skal fungere, er at programvaren Arduino IDE versjon 2 må lastes ned. Har du versjon 1 fra før av, anbefales det likevel å laste ned og bruke versjon 2. Det kan gjøres her: <https://www.arduino.cc/en/software>. Når programvaren er lastet ned riktig og du åpner den for første gang skal det se omtrent slik ut:



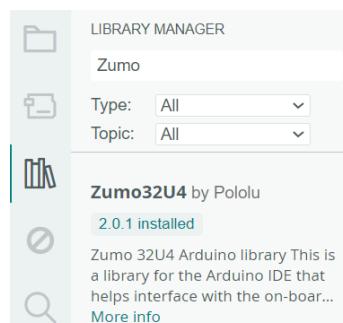
Last ned brettutvidelsen for Pololu Zumo 32U4

Får å kunne laste opp kode til robotbilen må man sørge for at Arduino IDE vet hvordan den ”snakker” med robotbilen.

Det første som må gjøres er å legge denne lenken: https://files.pololu.com/arduino/package_pololu_index.json i files -> preferences inn i ”Additional Board Manager URLs”. Dette vil gjøre det mulig å laste ned det riktige brettet til Arduino IDE. Gå så inn på Tools -> Board -> Board Manager og søk etter **Pololu**. Last så ned pakken kalt **Pololu A-star Boards**.

Last ned kodebibliotek for Pololu Zumo 32U4

Etter at vi har sørget for å kunne ”snakke” med robotbilen, må vi også vite hva vi skal si. For å forenkle ”språket” til robotbilen ønsker vi å laste ned et bibliotek som gjør det lettere å ”snakke”. Åpne opp **Library Manager** i venstre sidemeny. Søk opp **Zumo**. Last ned biblioteket i bilde.



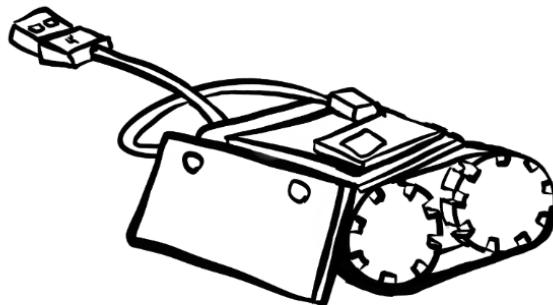
Last ned eksempelkoden fra Github

På Github-siden: <https://github.com/aheyerdahl/oppstartsuka-intro-2024> trykk på den store, grønne knappen og last ned innholdet som en .zip-fil. Åpne mappen kalt Eksempelkode og deretter mappen kalt Robotbil_motor. I denne mappen ligger det en .ino fil med samme navn, åpne denne.

Koble til bilen og velg board

Koble robotbilen til PC-en deres med den utdelte kabelen.

Deretter må dere ha Arduino IDE åpnet og velge **Pololu A-star 32U4** som **board** også riktig **port**. Hva riktig port er kan variere, på Windows kan det f.eks. stå COM5. Hvis du sliter med å finne riktig port (det kommer opp flere på listen) så spør om hjelp.



Last opp kode

Nå er vi klare til å laste opp koden til robotbilen! Trykk på knappen markert med en oransje sirkel, slikt vist i bilde under. Vent til det står opplasting ferdig eller noe lignende, nederst i Arduino IDE. Du kan nå koble bilen fra datamaskinen og skru bilen på med den lille bryteren på baksiden. Koden vil nå kjøre automatisk.



Utforsk og erfar

Nå er det din tur til å prøve selv! Bli kjent med, test ut og eksperimenter med eksempelkodene i denne rekkefølgen:

- Robotbil_motor.ino
- Robotbil_alt.ino

Hvis du gjør dette først, blir de neste oppgavene mye enklere. Om du står fast, les kommentarene i koden, sjekk på internettet eller spør om hjelp.

Ressursnettsidene for robotbilen er som følgende: <https://www.pololu.com/docs/0J63> og <https://pololu.github.io/zumo-32u4-arduino-library/>

2 Mønsterkjøring

Kort bakgrunn

Robotbilen er utstyrt med to motorer som kan bevege seg i to retninger. De to motorene styrer bevegelsen til hver sin side av bilen, ved å bevege ”hjulet” enten forover eller bakover. Vi skal nå programmere bilen til å kjøre fremover, bakover, snu i begge retninger og svinge i begge retninger.

Sammenlagt skal dette brukes til å forsere en hinderløype med flere poster. Hver post har spesifikke oppgaver som skal løses, og det er poenggivende både å kjøre fra post til post korrekt og å utføre oppgavene på hver post så nøyaktig som mulig.

Hovedmålene er å:

1. Navigere hinderløypen:
 - Robotbilen skal programmeres til å kjøre fra en post til den neste ved å følge et bestemt kjøremønster som kan være en rett linje, kurvet linje eller S-kurve, uten å bruke noen sensorer.
 - Poeng gis for hvor nøyaktig roboten følger den angitte linjen og hvor nært den stopper ved neste post.
2. Løse oppgaver ved hver post:
 - Ved hver post er det spesifikke oppgaver som robotbilen skal utføre, for eksempel å kjøre i en firkant eller velte kjegler.
 - Poeng gis for hvor nøyaktig og effektivt oppgavene løses.

Begrensninger:

For å øke utfordringen og fremme forståelsen av grunnleggende programmeringskonsepter, skal dere ikke bruke noen av sensorene som finnes i robotbilen i denne delen av oppgaven. Dette betyr at all navigasjon og oppgaveløsning må baseres på forhåndsprogrammerte sekvenser og tidsstyrte bevegelser.

Muligheter:

Det er flere mulige måter å løse denne oppgaven på. Hvordan dere gjør det er opp til dere selv.

- En mulig måte er å lage en funksjon/sett av instrukser for hver navigasjon og post. Deretter laster dere opp en kode som sier hvilken navigasjon eller post som skal kjøres akkurat nå. Så kan dere gjøre alle delene i etapper.
- En annen måte er å ta i bruk knappene på bilen. Disse kan settes til å aktivere en navigasjon eller post. Slik slipper en å laste opp ny kode for hver del.
- En kan også gjøre det mer komplisert (eller enklere, avhengig av hvordan en ser på det) og ta i bruk OLED-skjermen til robotbilen. Her kan en lage et meny-system, hvor knappene styrer menyen, også velger en hvilken navigasjon eller post som skal kjøres. Da kan en programmere inn alle også aldri laste opp mer kode.
- Et siste alternativ er å programmere robotbilen til å gjøre alle delene på rekke og rad. Dette er enklere sagt enn gjort.

Det er altså ingen krav til å gjøre alt på rekke og rad.

Da er det på tide å prøve!

a) Navigasjon 1: Kjør en rett linje

- Plasser robotbilen i startrektaangelet for hele løypa, og la den navigere langs en rett linje til rektangelet til Post 1. Det plasseres en kjegle midtveis mellom startrektaangelet og post 1-rektangelet. Gruppen får poeng for å velte kjeglen, og etter hvor nært post 1-rektangelet robotbilen stopper.

b) Post 1: Kjør en firkant

- Sett robotbilen i Post 1-rektangelet, og la den kjøre i en firkant. Oppgaven er å starte fra rektangelet, kjøre rektangulært og ende opp i samme rektangel. Det settes opp 3 kjegler for å indikere hjørnene på firkanten.

c) Navigasjon 2: Kjør en kurvet linje

- Kjør robotbilen i en sirkelbue fra rektangelet i Post 1 til Post 2. Det plasseres en kjegle midtveis.

d) Post 2: Kjør firkant med pируett

- Beskrivelse: Kjør en firkant, og utfør en pirusett (360 grader) i hvert av de tre hjørnene du passerer.

e) Navigasjon 3: Kjør en kurvet linje

- Kjør robotbilen i en s-kurve fra rektangelet i Post 2 til Post 3. Det plasseres to kjegler langs kurven.

f) Post 3: Kjeglekjøring

- I et 30x30 cm felt plasseres et antall kjegler. La robotbilen kjøre fra starthjørnet til det diagonale hjørnet, og velte så mange kjegler som mulig underveis.

g) Navigasjon 4: Kjør en kurvet linje

- Kjør robotbilen i en sirkelbue fra rektangelet i Post 3 til Post 4. Det plasseres en kjegle midtveis.

h) Post 4: Kombinasjonsoppgave

- Kombiner ferdighetene fra de tidligere oppgavene: Start i et rektangel, kjør en kurvet linje, gjør en pirusett i et gitt punkt, kjør gjennom et felt med kjegler og avslutt med å kjøre i en firkant.



Hvordan utføre tidsstyring? Hint: bruk den innebygde Arduino-funksjonen `delay()` eller `millis()`

3 Tegn et drage-kretskort

Læringsassistent Sara har over denne sommeren laget et drage-kretskort. Denne kan sees i bilde under. Hun ønsker nå å få masseprodusert tegninger av dette drage-kretskortet, men har dessverre ikke tid til å tegne alle selv ;)

Oppgaven deres er å hjelpe Sara med masseproduksjonen. Dere skal nå sette fast/teipe fast en blyant eller penn til robotbilen deres. Deretter skal den programmeres til å, så nøyaktig som mulig, tegne drage-kretskortet slik det vises i bilde under. Dere trenger ikke tegne de indre delene, som LED-lyset eller motstandene, men dere må tegne selve formen av drage-kretskortet.

Hittil har dere kunnet gjort oppgavene i etapper, altså en og en navigasjon og post. Nå økes nivået! Av hensyn til produksjonshastigheten, må robotbilen kunne tegne hele figuren på en gang, og så raskt som mulig. Dere må altså tidsplanlegge hele kjørerunden på en og samme gang. Det er viktigst at tegningen blir så nøyaktig som mulig, så prioriter resultat over tiden det tar robotbilen å tegne.

Noen viktige punkter:

- Det er åpenbart ikke lov å håndtegne i denne oppgaven ;) sorrynotsorry
- Størrelse på tegninga deres er ikke viktig, om det er enklere å gjøre det stort så har vi A3!
- I tillegg er det ikke urimelig at noen av kantene heller kanskje blir halvsirkler. Dere får se om dere klarer å løse det problemet og



Figure 1: drage-kretskortet

4 Ekstra oppgave: Linjefølging

Denne oppgaven er for de som ønsker en ekstra utfordring.

Litt om robotbilens sensorer

Nå som vi har lært hvordan vi kan få bilen til å kjøre, er det på tide å få den til å samhandle med omgivelsene. Dette oppnår vi ved hjelp av sensorer, som fungerer som systemets øyne og ører - altså sansene til systemet. Sensorer er ansvarlige for å "se" og oppfatte hva som foregår rundt bilen. De fanger opp fysiske tilstander og konverterer dem til elektriske signaler som vi kan tolke og bruke. I denne oppgaven vil dere lære å bruke IR-sensorene til robotbilen, den kan fortelle noe om fargen på bakken den kjører på. Denne sensoren spiller en avgjørende rolle i å gi bilen informasjon om posisjon og bevegelse, slik at den kan tilpasse seg og reagere på riktig måte.

I figuren under kan dere se de fleste sensorene som sitter på robotbilen. Sensoren brukt i denne oppgaven heter i bilde **line sensor array**.

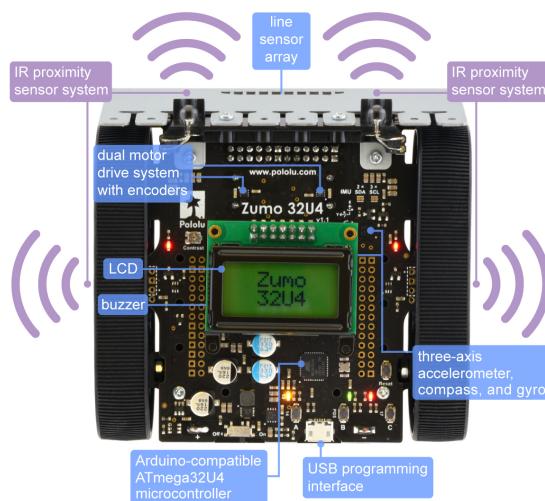


Figure 2: <https://www.pololu.com/docs/0J63/all>

IR-sensorene (line sensor array)

Robotbilen er utstyrt med 5 sensorer som oppfatter infrarødt lys (IR). IR-sensorer kan brukes som nærhetsdetektorer, eller så kan de brukes til å "se" linjen på bakken ved å sjekke hvor mye infrarødt lys som reflekteres fra en IR LED-lampe. Denne skal vi bruke til "se" mørk teip, eller "linjer", som da skal brukes til linjefølging. Før vi kan bruke linjesensorene må vi kalibrere de.

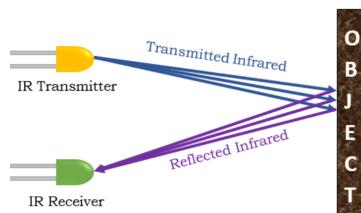


Figure 3: <https://www.mathaelectronics.com/ir-sensor-circuit-diagram-types-working-with-applications/>

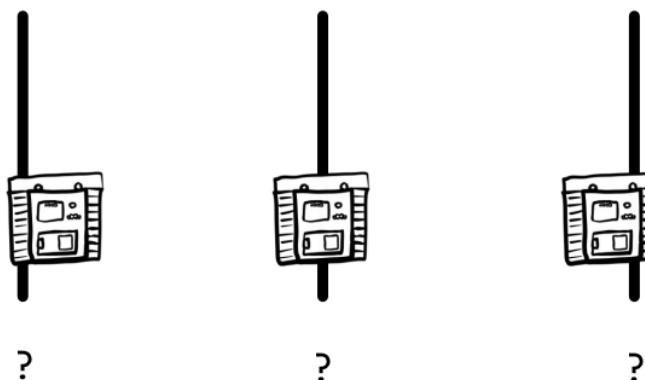
IR-sensorene på undersiden av bilen gir ut ett sett med verdier. Ved bruk av betingelser og sensorverdiene kan vi få robotbilen til å kjøre enten til venstre eller høyre, eller rett fremover. Vi kan hente ut verdiene fra IR-sensorene med kommandoen:

```
float position = lineSensors.readLine(lineSensorValues)
```

Her blir da dataen fra sensoren lagret i variabelen kalt `position`. Se eksempelkode fra Github.

Oppgave 1 - Les av linjedata

Les av linjedata, og vis den på OLED-skjermen i sanntid. Finn ut av hvilke verdier som representerer de 3 forskjellige scenarioene i bilde under og noter disse. Husk å kalibrere sensorene først!



Oppgave 2 - Reagere på linjedata

Nå som vi kan lese av dataen fra linjefølgeren, så kan vi få bilen til å utføre handlinger basert på hva sensor-dataen forteller oss!

Ved hjelp av en if-setning i void loop(): Skriv kode som gjør at robotbilen svinger til høyre når den står til venstre for teipen, og motsatt. Bruk kodesnuttene under som inspirasjon.

```
if (position > ??) {  
    motors.setLeftSpeed(?);  
    motors.setRightSpeed(?);  
}  
else {  
    motors.setLeftSpeed(?);  
    motors.setRightSpeed(?);  
}
```

Oppgave 3 - En mer avansert styring

Den forrige metoden for å følge linjen fungerer, men den har potensiale for forbedring! I denne oppgaven skal vi forsøke å gjøre den mer dynamisk.

Prøv å lage en styremetode som bruker variablen **`position`** for å bestemme hva bilen skal gjøre.

Her må du kanskje skalere position variabelen litt? Eventuelt lage en ny variabel og skalere denne?

```
motors.setLeftSpeed(50 + position);  
motors.setRightSpeed(?);
```

Vedlegg 1: Forklaring av noen programmeringsbegreper



All koden som blir referert til kan finnes ved å trykke [her](#)

Lagring - Variabler

Variabler er på mange måter likt som det dere har brukt i algebra, men nå er dere ikke lengre begrenset til tall! Vi kan se på en variabel som en boks, der du først må gi et navn til boksen. I tillegg til å navngi boksen må vi vite hva vi skal legge i den, for eksempel masse tall eller masse bokstaver. Ved å si hva som skal være oppi boksen passer vi på at den holder på det vi legger oppi. Hvis vi f.eks vil legge vann i boksen, må den helst være vanntett!

For å lage en variabel må vi derfor fortelle programmet hva vi vil putte i den. For tall sier vi at det skal være en ”integer” (heltall) ved å skrive ”int” før variabel-navnet som sett på linje 1, og så bruker vi ”=” for å si hva variabelen skal være (hva innholdet er). Hvis vi bare vil ha en variabel som forteller oss om noe stemmer (true) eller ikke (false) kan vi bruke det som kalles for en boolsk variabel, en boolean. Her skriver du ”bool” etterfulgt av variabelnavnet, og så sier at det skal inneholde enten ”true” eller ”false” som vist på linje 3. Dette er noe som kan være meget nyttig senere for å få en fin og ryddig kode. Til slutt har vi ord eller tekst som vi kan lage ved å sette ”String” forran variabelnavnet vist på linje 5. Her er det viktig å bruke ” ” rundt innholdet. Det er også mulig å lage en variabel uten å legge noe i den med en gang, da trenger du bare skrive typen og navn som vist på linje 7.

Men vi trenger ikke lage nye variabler om vi trenger andre verdier! Vi kan nemlig bruke ”=” tegnet for å gi variablen en ny verdi, dette finner dere et eksempel på i linje 31. Det finnes også eksempler på slike operasjoner [her](#).

Betingelser - If og Else

I hverdagen gjør vi ofte valg utifra om noe stemmer, f.eks om det ikke er egg i kjøleskapet, kjøper man flere egg. På samme måte vil vi ofte sjekke om alt ligger til rette for å gå videre med noe i en kode. Her kan vi bruke If setninger. En If setning er en ”sjekk” som tar inn en påstand og hvis det stemmer går den videre med å kjøre sitt tilhørende kodeavsnitt. Det er viktig å skille mellom ”=” og ”==”, der ”=” brukes for å gi noe en verdi mens ”==” brukes for å sjekke om to ting er like.

Når vi bruker en If-setning er det vanlig å sammenligne en variabel med en annen verdi, eller annen variabel. På linje 22 kan vi se at vi sammenlikner variabelen ”tall” med tallet 2. Vi ser at vi i parantesen har påstanden ”(tall == 2)”. Hvis det er riktig kjøres koden i krølleparantesen, men hvis det ikke stemte så hadde koden bare hoppet til neste avsnitt i koden.

Av og til vil vi gjøre noe hvis en påstand stemmer, men alltid gjøre noe annet hvis og bare hvis det ikke stemmer. Her kommer Else-setningen inn i bildet. Den kan settes etter en If-setning. Hvis påstanden til If-setningen stemmer, så hopper vi over Else koden, men ellers kjører vi else koden. Vi ser igjen på linje 25 at hvis if setningen fra linje 22 ikke stemmer, så kjøres Else-setningen på linje 25.

Funksjoner

Funskjoner når vi programmerer fungerer veldig likt som en matematisk funskjon, bare at vi igjen ikke er begrenset til kun tall. Her kan vi ta inn så mange variabler vi vil og så kjøre en kode, litt som en If-setning bare at vi ikke sjekker om noe er sant eller ikke. Her kan vi gjøre det vi vil med variablene og enten returnere et resultat eller ikke. Syntaksen for å lage en funskjon er veldig lik som en variabel, bare at du nå har paranteser rundt variablene som skal gå inn, og krølleparanteser rundt det funksjonen skal gjøre. Det er viktig å merke seg at funskjoner må defineres utenfor void `setup()` og `void loop()` ettersom dette i seg selv er funskjoner.

På samme måte som vi måtte forklare hva vi ville at variablene skulle inneholde må vi forklare hva vi vil ha ut av funksjonen. Hvis vi f.eks vil lage en matematisk funskjon kan vi se på linje 16 der vi tar inn en integer (tall) med navnet `x`. På samme måte som vi må definere at en variabel med tall er en "int" må vi definere en funskjon med "int" for at den skal kunne returnere en tallverdi. Her står int forran funskjonsnavet "matte-funksjon".

Av og til vil vi sjekke om flere ting stemmer samtidig, noe som kan skape en enorm if-setning. For å unngå dette kan vi heller lage en funskjon som sjekker noe, og sier da om dette stemmer eller ikke. En slik funskjon defineres med "bool". Vi kan si at hvis noe stemmer, så skal den returnere true, men hvis det ikke stemmer kan den returnere false. Skjønnheten ved denne metoden i if-setninger er at du kan sette denne funksjonen inn i en if-setning igjen, som gjør koden mye penere. Et eksempel på dette er skrevet på linje 21.

Til slutt har vi kanskje den enkleste men også rareste måten å definere en funskjon, nemlig å ikke få noenting tilbake! Dette gjøres ved å definere funksjonen som en "void". Et eksempel på en slik funskjon er på linje 30, som enkelt bare setter tall-variabel til 2 hvis f.eks den har blitt endret. Nå kan vi også forstå hvordan `void setup()` og `void loop()` fungerer. Der `setup` er en egen funksjon som kjøres i starten av et Arduino-program, men returnerer ingenting. `Void loop()` er liknende bare at mikrokontrolleren kjører den om og om igjen helt til universets varmedød eller den blir myrdet.

Kode eksempel:

```
1 int tall_variabel = 2;
2
3 bool state = true;
4
5 string ord = "hei!";
6
7 int flere_tall;
8
9 if(tall_variabel == 2){
10     Serial.println("tallet er lik 2");
11 }
12 else{
13     Serial.println("det er ikke lik 2");
14 }
15
16 int matte_funskjon(int x){
17     int resultat = 2*x;
18     return resultat;
19 }
20
21 bool sjekk_om_sant(int tall){
22     if (tall == 2){
23         return true;
24     }
25     else{
26         return false;
27     }
28 }
29
30 void oppdater_variabel(){
31     tall_variabel = 2;
32 }
33
34
35 void setup() {
36     // put your setup code here, to run once:
37 }
38
39
40 void loop() {
41     // put your main code here, to run repeatedly:
42 }
43
```

Vedlegg 2: Arduino-tips

variabel1 == variabel2	Sjekker om variabel1 er lik variabel2
variabel1 != variabel2	Sjekker om variabel1 ikke er lik variabel2
variabel1 > variabel2	Sjekker om variabel1 er større enn variabel2
variabel1 < variabel2	Sjekker om variabel1 er mindre enn variabel2
int variabelnavn = 1; (heltall) float variabelnavn = 4.5; (desimaltall) bool variabelnavn = true; (sant/usant)	Lager informasjonen etter = som en variabel med navnet foran =. Husk å bruke riktig type variabel til det du skal lagre.
delay(heltall)	Får mikrokontrolleren til å vente i et bestemt antall millisekunder (tusen millisekunder blir ett sekund).
If(variabel1 == variabel2) { valgfri kode 1; } else { valgfri kode 2; }	Sjekker om påstanden inne i parantesen stemmer, for eksempel om variabel1 er lik variabel 2. Hvis det stemmer kjøres koden inne i måkeparentesen til <i>if</i> , og koden etter <i>else</i> hoppes over. Hvis det ikke stemmer kjøres koden i måkeparentesen etter <i>else</i> , og koden etter <i>if</i> hoppes over.
for(int teller = 0, teller < 10, teller += 1) { valgfri kode; }	Kjører valgfri kode ti ganger.