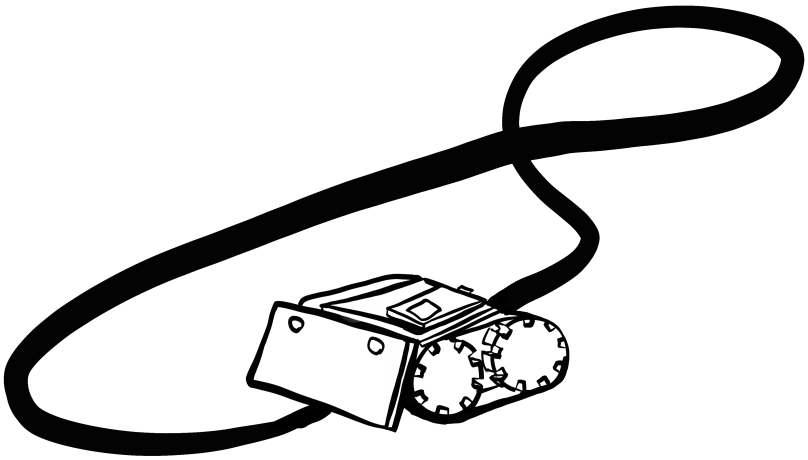
NTNU

Kunnskap for en bedre verden

BIELDIG og BIAIS

Prosjekt i oppstartsuka

Robotbil og regulering



August 13, 2024

Introduksjon

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

Dere skal i løpet av de neste dagene sammen lage et automatisk system for en selvkjørende bil. Den skal kunne følge etter en linje på bakken, og justere vinkelen sin slik at den holder en konstant retning. For å oppnå dette kommer dere til å jobbe i grupper ifra studiene, og etterhvert sammarbeide med studenter fra BIELSYS som skal utforske andre sensorer på bilen. På fredag blir det en konkurranse blant gruppene, så det er bare å kjøre på!

For å bygge dette systemet skal dere benytte en mikrokontroller kalt for "ESP32". Denne kan tenkes på som en liten datamaskin som utfører enkle instruksjoner, og for å styre den skal dere programmere i språket "Arduino-C" (ligner C++). Dere har nå fått "skjelettet" til koden, og deres jobb er å bygge ut resten ved å fullføre en rekke funskjoner

Vedlagt her nede i dokumentet er det forklaringer av hvordan språket 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++.

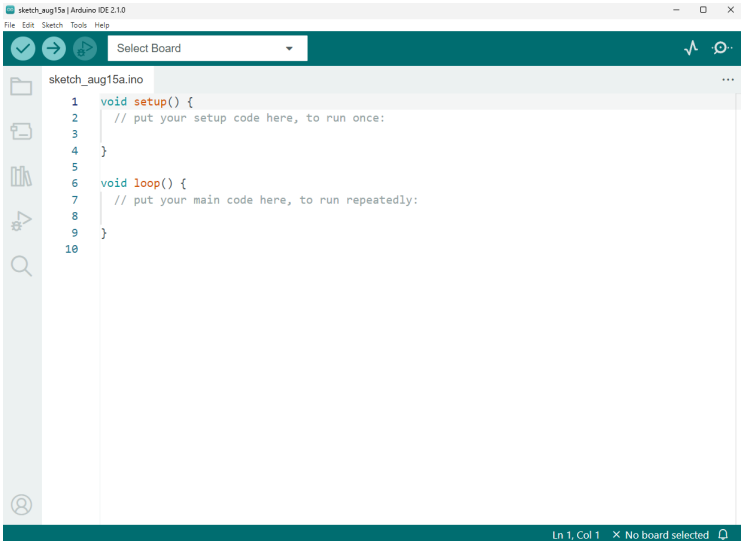
Til de av dere som kanskje har brukt C++ litt før er det noen utfordringer lagt til i form av ekstraoppgaver. Dette er ment mer som noe å bryne seg litt over hvis man kommer fort gjennom opplegget og er derfor ikke viktig å bli ferdig med.

Da gjenstår det bare å si lykke til!

1 Installer programvaren og utforsk

Last ned Arduino IDE

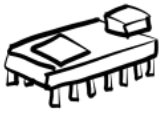
Det første som må gjøres, for at alt annet skal fungere, er at programvaren Arduino IDE versjon 2 må lastes ned. Har du versjon 1 fra før av, anbefales det likevel å laste ned og bruke versjon 2. Det kan gjøres her: https://www.arduino.cc/en/software. Når programvaren er lastet ned riktig og du åpner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



Last ned brettutvidelsen for ESP32

Får å kunne laste opp kode til ESP32-en må man gjøre Arduino IDE kompatibel med brettet (board). Det første som må gjøres er å legge denne lenken: https://espressif.github.io/arduino-esp32/package_esp32_index.json i files -> preferences og legge linken inn i "Additional Board Manager URLS". Dette vil gjøre det mulig å laste ned den riktige brettutvidelsen. Gå inn på Tools -> Board -> Board Manager og søk etter **ESP32**. Last så ned pakken kalt **esp32 by esspresif systems**, **IKKE!** Arduino ESP32 Boards by Arduino.

OBS! Det er også svært viktig at du laster ned enten 2.0.11 eller 2.0.14 versjonen av denne brettutvidelsen, ikke versjon 3 eller høyere!



Last ned COM-port driver for ESP32

Nå som vi har lært Arduino-IDE å "snakke" med ESP32-en, må vi lære PC-en vår å gjøre det samme. For å få til dette må flere av dere laste ned en såkalt COM-port driver. Om du ser at ESP32-en dukker opp i Arduino IDE når du kobler den til, i formen av en ny COM-port, kan du hoppe over dette steget. Om ikke må du laste ned driveren, som finnes her: https://www.silabs.com/developers/usb-to-uart-bridge-vcp-drivers?tab=downloads. Det er dessverre slikt at det er forskjellige drivere for forskjellige PC-er. Det kan altså hende at du må laste ned 2 eller 3 av driverne i lenken før en av dem funker ;)

Last ned nettside-biblioteker for ESP32

Finn "Library manager", søk opp **AsyncTCP** og last ned den nyeste versjonen.

På Github-siden: https://github.com/me-no-dev/ESPAsyncWebServer trykk på den store, grønne knappen og last ned innholdet som en .zip-fil. Åpne så Arduino IDE og gå inn på Sketch -> Include Library -> Add .ZIP library. I filutforskeren finner du så zip-filen du nettopp lastet ned og velger denne. Vi trenger dette biblioteket for at ESP32-en skal kunne sette opp en server dere kan koble til med nettleseren deres.

Last ned robot-car-biblioteket med eksempel for ESP32/Zumo

På Github-siden: https://github.com/aheyerdahl/oppstartsuka-bieldig-biais-2024 trykk på den store, grønne knappen og last ned innholdet som en .zip-fil. I mappen finner du en mappe kalt ***carLibrary*** og en mappe kalt ***carCode***. I ***carLibrary*** ligger det en del utvidelser til Arduino, biblioteker, som vi trenger for at vårt system skal funke. Alt i denne mappen må plasseres i mappen med lokasjon "Dokumenter/Arduino/libraries" på PC-en din. I ***carCode*** ligger en fil med samme navn, som vi skal åpne. Dette er koden som skal styre bilen. I løpet av opplegget er det du som skal skrive den ferdig!

En kort forklaring av koden under går som følgende:
De to linjene før *void setup* gjør klar ekstrafunksjoner som bilen trenger for å fungere. All kode i void setup kjører én gang når bilen skrus på. Her ligger en funksjon som setter i gang bilen, car.initCar().

Koden i *void loop* kjøres om og om igjen så lenge mikrokontrolleren er skrudd på. Her skal vi senere skrive kode for å automatisk styre bilen og overvåke sensorene.

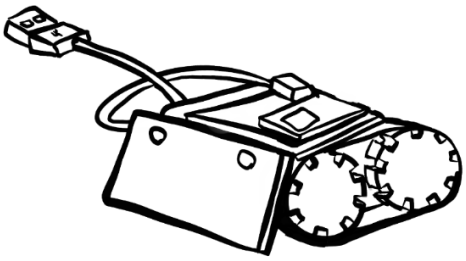
Kode som skrives i *void triangle, circle* og *square* kjøres når man trykker inn eller slipper knappene på tastaturet mens man er inne på nettsiden.

```
#include "Car.h"

Car car("NTNU-IOT", "");

void setup() {
   car.initCar();
}

void loop() {
   // kode som kjører mange ganger
}

// dette er koden for gjøre noe når trekant trykkes på nettsiden,
// eller tasten 1 trykkes ned på tasturet/mobilen
void triangle(bool button) {
   if (button == DOWN) {
      // knappen trykkes ned, skru på linjefølging?
   }
   if (button == UP) {
      // knappen slippes opp, skru av linjefølging?
   }
}
```

Inne på Github-siden finner dere en liten guide på forsiden. Denne kan være til god hjelp underveis i opplegget.

Koble til ESP32-en og velg board

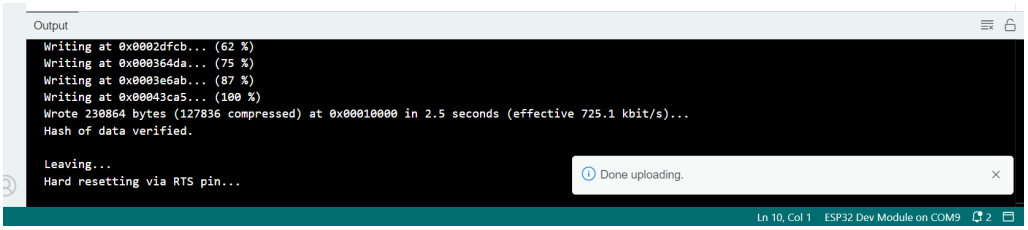
Koble ESP32-en på toppen av bilen til datamaskinen med den utdelte kabelen. Deretter må dere ha Arduino IDE åpnet og velge **ESP32 Dev Module** som **board** også riktig **port**. Hva riktig port er kan variere, på Windows kan det f.eks. stå COM5. Hvis du sliter med å finne riktig port (det kommer opp flere på listen) så spør om hjelp.



Last opp kode

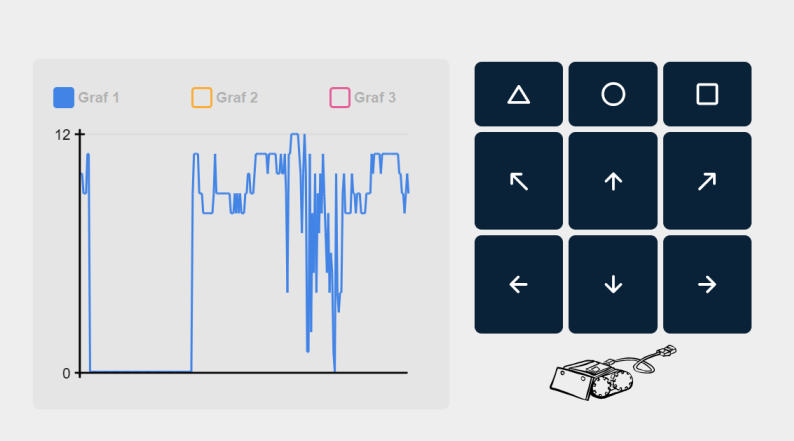
Nå er vi klare til å laste opp koden til ESP32-en! Trykk på knappen markert med en oransje sirkel, slikt vist i bildet under.

Vent til det står "Hard resetting via RTS pin...", nederst i Arduino IDE, slikt vist i bildet under. Da er koden lastet opp riktig på ESP32-en. Om det står noe veldig annerledes enn det som vises i bilde her kan noe ha gått galt, selv om det ikke kommer opp en konkret feilmelding som er åpenbar.



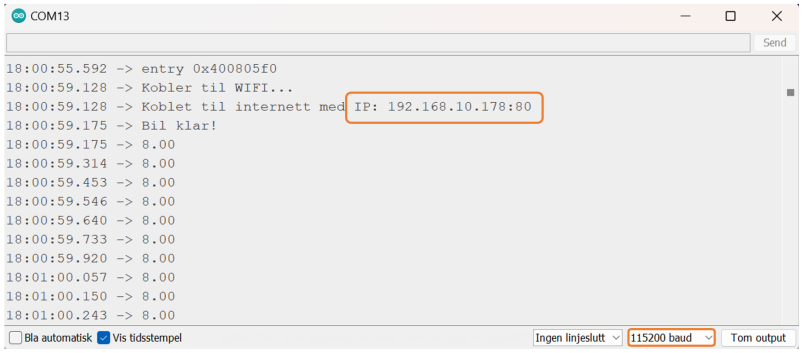
Du kan nå koble ESP32-en fra datamaskinen og skru robotbilen på med den lille bryteren på baksiden. Koden vil nå kjøre automatisk og robotbilen vil forskyne ESP32-en med strøm. Du kan nå kommunisere trådløst med ESP32-en (og dermed robotbilen også). Det vises hvordan på neste side om kontrollpanelet.

Kontrollpanelet (nettsiden)



Dette er kontrollpanelet vi bruker for å styre bilen. For å vise må du laste opp koden på ESP32. Deretter må du åpne serial-monitor (Ctrl+Shift+M), og lese av IP-adressen. Denne skrives så inn i nettleseren, der du vanligvis legger inn en nettsideadresse. Slikt kobler du til kontrollpanelet.

OBS! Sørg for at baudrate (nederst til høyre) er satt til 115200!



På venstre side av kontrollpanelet har vi en graftegner. For å sende data til denne bruker vi funksjonen *car.sendData()*. I parentesen velger vi graf 1, 2 eller 3, og hvilken variabel vi vil sende. Under sendes en variabel som heter «hastighet» til graf 1:

$$car.sendData(1, hastighet)$$

Knappene med figurer på høyre side brukes til å sende kommandoer til bilen, og er koblet til ulike taster på PC-en.

2 Linjefølging

Introduksjon

Nå som vi har lært hvordan vi kan laste opp kode til bilen, 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 løpet av dette prosjektet vil dere lære om ulike typer sensorer slik som IMUer og fotosensorer. Disse sensorer 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 noen av sensorene som sitter på Zumoen.

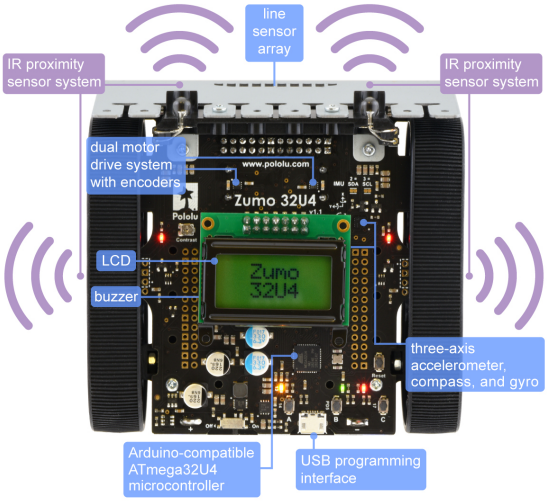


Figure 1: https://www.pololu.com/docs/0J63/all

IR-sensorene (line sensor array)

Zumoen 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". Dette skal brukes videre senere til linjefølging.

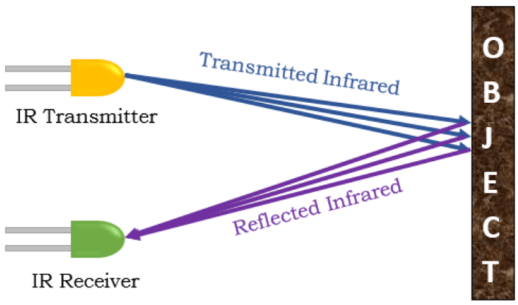
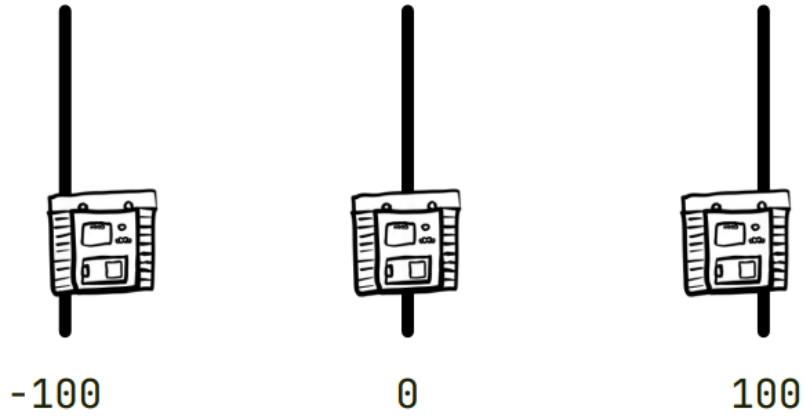


Figure 2: https://www.mathaelectronics.com/ir-sensor-circuit-diagram-types-working-with-applications/

Linjesensor

IR-sensorene på undersiden av bilen kan oppfatte fargen på gulvet. Disse kan vi bruke til å få bilen til å følge en linje. Vi kan hente ut dataen med `car.data[LINE].value`. Dette gir oss en verdi mellom -100 og 100, avhengig av hvor linjen er i forhold til bilen.



Før vi kan bruke linjesensorene må vi kalibrere de. Vi må da bruke funksjonen `car.calibrateLine(BLACK)`. Pass på at bilen står på en linje når denne funksjonen kjøres. Tallet vi får fra sensorene kan vi lagre i en variabel. Denne kan for eksempel kalles linjetall og under vises det hvordan dette kan gjøres.

```
int linjetall;

linjetall = car.data[LINE].value;
```

Linjedataen må leses av i void loop()

Oppgave 1 - Les av linjedata

Les av linjedata, og visualiser den i nettleseren. For å sende for eksempel en variabel, linjetall, til grafen på kontrollpanelet kan vi skrive `car.sendData(1, linjetall)`.

Oppgave 2 - Skru av og på linjefølgeren

Lag en kode til en av de tre knappene (trekant, firkant eller runding) som kan brukes til å skru linjefølgeren av og på.

Du kan kjøre mye kode innenfor en if-setning!
if(linjefølger == true) { ..... }

Oppgave 3 - 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 Zumo svinger til høyre når den står til venstre for teipen, og motsatt. Bruk kodesnutten under som inspirasjon.

```
if (linjetall > ??) {
    car.drive(??, ??);
}
```

Oppgave 4 - 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 ***linjedata*** for å bestemme hva bilen skal gjøre.

```
car.drive(50 + linjetall, ...
```

Oppgave 5 - Skalere linje-dataen

I den forrige oppgaven kan det hende at bilen ikke kjørte så veldig godt. Endre på koden slik at bilen følger linjen bedre, ved å skalere linjedata variablen.

```
car.drive(50 + (??*linjedata), ...
```

3 Bruk av IMU

En IMU (Inertial Measurement Unit) er en enhet som brukes til å måle og registrere bevegelse, orientering og akselerasjon av et objekt. Den består vanligvis av flere sensorer, inkludert akselerometre, gyroskoper og ofte også magnetometre (kompass).

Akselerometre måler akselerasjon og lar IMU-enheten registrere endringer i fart eller bevegelse i ulike retninger. Gyroskoper måler vinkelhastighet og gir informasjon om rotasjon rundt aksene. Magnetometre oppdager magnetfelt og brukes til å lese av orientasjonen i forhold til jordens magnetfelt, altså den fungerer som et kompass.

Ved å kombinere dataene fra disse sensorene kan IMU-enheten beregne og gi informasjon om objektets bevegelse i sanntid. IMU-er brukes til mye forskjellig, blant annet navigasjonssystemer, droner, og robotikk. Det sitter f.eks også en IMU i mobilen din! IMU-er gir verdifull informasjon om bevegelse og orientering, og spiller derfor en viktig rolle når Zumoen skal oppfatte egen tilstand.

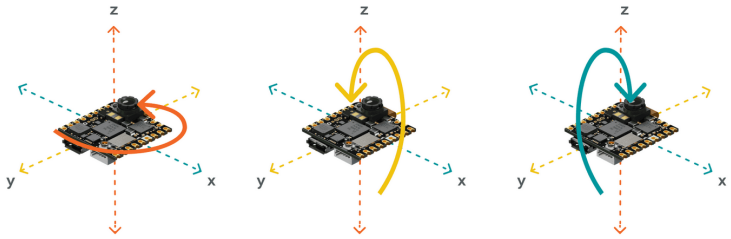


Figure 3: https://docs.arduino.cc/tutorials/nicla-vision/nicla-vision-imu

Oppgave 1 - Visualisere gyroskop-data

Bruk kodesnutten under som inspirasjon for å lese av og visualisere dataen fra gyroskopet. Dataen vi har tilgang til gir oss verdier mellom -180 og 180 som forteller oss vinkelen til bilen om z-aksen.

```
car.calibrateGyro(2048);
car.sendData(2, car.data[GYRO].value);
```

Oppgave 2 - Stabilisator

Legg til kodesnutten under.

```
int angle = car.data[GYRO].value;
int turnspeed = (angle)*100 / 180;
car.drive(turnspeed * 10, -turnspeed * 10);
```

Denne må plasseres inne i loop, og vil gjøre at bilen motsetter seg endringer i vinkel om z-aksen.

Legg gjerne til en variabel og en if-setning som lar deg skru av og på denne delen, tilsvarende som for linjeføLgeren.

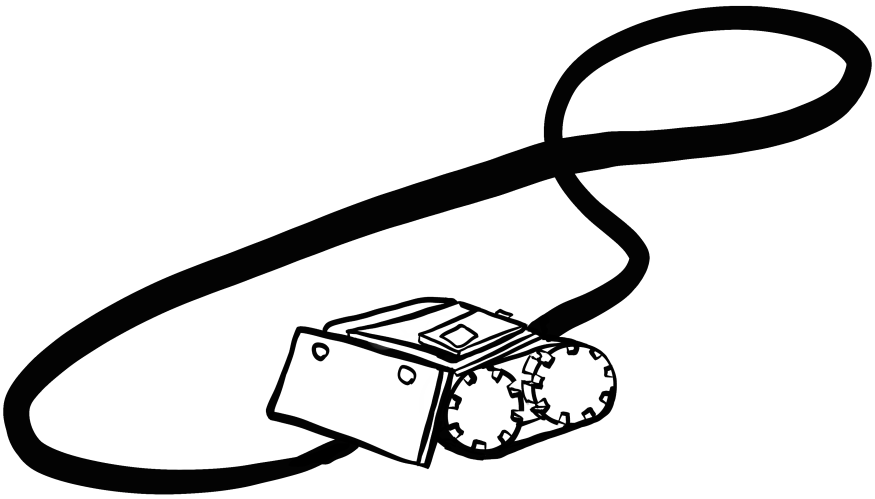
4 PID-regulator

En PID-regulator kan brukes til mye nyttige og er noe flere av dere vil være godt kjent med innen dere er ferdig med studiet. Likevel kan begrepet/konseptet være ukjent for mange, iallfall første uke av studiet. Deres oppgave nå blir å gjøre litt egen ettforskning om hva en PID-regulator er og prøve deres beste å forstå hva de tre parameterene P, I og D gjør og betyr.

Deretter kan dere hente fram eksempelkoden fra Github-siden kalt ***carPID***. Dette eksempelet viser dere hvordan en PID-regulator kan implementeres på robotbilen. Deretter skal dere bruke kunnskapen deres fra tidligere oppgaver til å integrere denne eksempelkoden, i den koden dere hittil har skrevet. Slikt at dere kan skru den av og på etter ønske, sammen med resten av systemet.

Når dette er gjort skal dere teste ut PID-regulatoren. Funker den med en gang? Kan den gjøres bedre? Forandre på de forskjellige parameterene i koden og observer hvordan oppførselen til bilen forandrer seg, da mens den kjøre langs teiplinja. Klarer den å kjøre knappe, brå svinger f.eks.?

Denne oppgaven spesielt har ingen riktige svar, deres oppgave er å lære.



5 Fartsmåling

Nå som dere suser av gårde i alle retninger må dere kanskje passe på å unngå en stor bot. Da er det lurt å kunne måle farten. Dette kan vi gjøre ved å se på hvor mange ganger hjulene har rotert.

Vi kan måle farten ved å se på hvor mange ganger hjulene som styrer beltene har rotert per tidsenhet. For å måle dette er det montert det som kalles for en encoder på motorene. Encoderen bruker en magnetisk sensor til å måle rotasjonen av motoren. Den måler 12 steg per motorrotasjon. Vi kan bruke `car.data[ENCODERS].value` og `car.data[READ_TIME].value`. Disse gir oss henholdsvis antall motorrotasjoner (ganger 12) og tiden det tok å telle antal rotasjoner.

> Merk at enkoderene teller 12 ganger per motorrotasjon. Hjulene roterer én gang per 75te motorrotasjon. Hjulenes diamater, inkludert beltene, er 39mm. Ved full hastighet (100) skal bilen kjøre noe mellom 10 - 40 cm/s.

Ta utgangspunkt i denne kodesnutten for å lese av encoderen og se om du klarer å regne ut hvor fort bilen kjører.

```
if (car.data[ENCODERS].flag) {
   int encoderData = car.data[ENCODERS].value;
   car.data[ENCODERS].flag = false;
}
```

Vedlegg 1: Forklaring av noen programmeringsbegreper

Alle 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 variablelen 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 ifra linje 22 ikke stemmer, så kjøres Else-setningen på linje 25.

Funksjoner

Funskjoner når vi programerer fungerer veldig likt som en mattematisk 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 funskjonen 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-funskjon".

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 funskjonen 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 funskjonen 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 mikrokontrollerern kjører den om og om igjen helt til universets varmedød eller den blir myrdet.

Kode eksempel:

```
int tall_variabel = 2;

bool state = true;

string ord = "hei!";

int flere_tall;

if(tall_variabel == 2){
  Serial.println("tallet er lik 2)");
}
else{
  Serial.println("det er ikke lik 2");
}

int matte_funskjon(int x){
  int resultat = 2*x;
  return resultat;
}

bool sjekk_om_sant(int tall){
  if (tall == 2){
    return true;
  }
  else{
    return false;
  }
}

void oppdater_variabel(){
  tall_variabel = 2;
}


void setup() {
  // put your setup code here, to run once:

}

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

}
```

Vedlegg 2: Arduino-tips

variabel1 == variabel2 variabel1 != variabel2 variabel1 > variabel2 variabel1 < variabel2	Sjekker om variabel1 er lik variabel2 Sjekker om variabel1 ikke er lik variabel2 Sjekker om variabel1 er større enn variabel2 Sjekker om variabel1 er mindre enn variabel2
int variabelnavn = 1; (heltall) float variabelnavn = 4.5; (desimaltall) bool variabelnavn = true; (sant/usant)	Lagrer 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 måkeparantesen til *if*, og koden etter *else* hoppes over.
	
	Hvis det ikke stemmer kjøres koden i måkeparentesen etter *else*, og koden etter *if* hoppes over.

for(int teller = 0, teller < 10, teller += 1) {
  valgfri kode;
}
	Kjører valgfri kode ti ganger.

Vedlegg 3: Oversikt over robot-car biblioteket

car.initCar()	Må skrives i setup. Kobler til WiFi, skrur på bilen og kontrollpanelet.
car.drive(leftSpeed, rightSpeed)	Denne funksjonen får bilen til å kjøre. leftSpeed og rightSpeed må være heltall mellom -100 og 100.
triangle(bool button) circle(bool button) square(bool button)	Alle disse funksjonene brukes til å motta knappetrykk fra knappene på. Når button er DOWN, vil det si at knappen har blitt trykket ned. Når button er UP, har knappen blitt sluppet opp. Disse knappene kan brukes til å starte linjefølgemodus f.eks.
car.data[LINE].value	Gir et tall mellom -100 og 100 som forteller hvor linjen er i forhold til bilen. Positive tall sier at linjen er til høyre for bilen.
car.data[PROXIMITY].value	Gir et tall mellom 0 og 12 som forteller hvor nærme et objekt er fronten til bilen. Jo høyere tall jo nærmere.
car.data[ENCODERS].value	Teller motorrotasjoner. Verdien blir oppdatert omtrent 20 ganger i sekundet. Merk at enkoder-sensorene teller 12 ganger per motorrotasjon, og at motoren roterer mange ganger for hver gang hjulet roterer.
car.data[GYRO].value	Gir et tall mellom -180 og 180 som forteller bilens vinjel om z-aksen.
car.data[READ_TIME].value	Gir ut tiden i milliseukunder det tok å lese av sensorene én gang. Denne kan være nyttig for å vite hvor lang enkoderene bruker på en måling.
car.sendData(graf, data)	Denne funksjonen sender data til kontrollpanelet, slik at du kan se det på en graf! Graf byttes ut med enten 1, 2 eller 3, for å vise datapunktet data på graf 1, 2 eller 3 i kontrollpanelet. Om funksjonen kjøres kontinuerlig inne i `void loop` vil dataen i grafen bli oppdatert ca 10 ganger i sekundet.
car.calibrateLine(BLACK)	Kalibrerer linjesensorene. Denne funksjonen må kjøres før man kan bruke linjesensorene.
car.calibrateGyro(2048)	Kalibrerer gyroskopet. Dette må kjøres før man kan lese av gyroskop-dataen.