Universitetet
i Stavanger

DET TEKNISK-NATURVITENSKAPELIGE FAKULTET

BACHELOROPPGAVE

Studieprogram/spesialisering: Bachelor i ingeniørfag / Datateknologi	Vårsemesteret 2021 Åpen eller Konfidensiell
Forfatter(e): Espen Eigestad, Remi Harbo, og Sarezh Pavel Nyland	
Fagansvarlig: Tormod Drengstig Veileder(e): Tormod Drengstig	
Tittel på bacheloroppgaven: Python og Lego EV3 Engelsk tittel: Python and Lego EV3	
Studiepoeng: 20	
Emneord: Python, EV3 Mindstorms, MatLab	Sidetall: 118 + vedlegg/annet: 109 Stavanger 15. mai 2021

Forord

Denne bacheloroppgaven er gjennomført for Universitetet i Stavanger som en avslutning på studiet som ingeniør innenfor datateknologi.

Forfattere av rapporten er Espen Eigestad, Remi Harbo og Sarezh Pavel Nyland. Vi kjente ikke hverandre før vi begynte på bachelor-oppgaven sammen. Alle 3 studerer datateknologi.

Bacheloren ble lyst ut av UiS på vegne av Tormod Drengstig. Etter vi fikk studert oppgaven nærmere, virket det ut som at dette var en meget interessant oppgave å skrive bacheloren sin om.

Vi vil takke veileder Tormod Drengstig fra Universitetet i Stavanger for å ha hjulpet oss der det har vært behov. Sammen med han har vi hatt jevne veiledningstimer siden bachelor-start. Veileder har også vært med på å bestemme vanskelige valg som vi har opplevd under arbeidet med bacheloren. Han har tatt seg tid til oss ved behov og har kommet med forslag der det har vært nyttig.

Vi ønsker også å takke Romuald Karol Bernacki for utlån av nødvendig elektronisk utstyr.

Espen Eigestad, Remi Harbo og Sarezh Pavel Nyland

Stavanger 15. mai 2021

Sammendrag

Oppgaven er skrevet på bakgrunn av faglige kompetanse som er blitt oppnådd utover studieperioden.

Hensikten med oppgaven var å lage et rammeverk i Python for prosjekt som utføres ved hjelp av en programmerbar EV3-robot. Dette rammeverket skal bli brukt til undervisning på UiS i programmering.

Det finnes allerede et eksisterende rammeverk som blir brukt på skolen, men dette rammeverket bruker MatLab. Dette rammeverket gir studenter innføring i programmering, og blir brukt til å utføre matematiske prosjekter. Dette rammeverket ble konvertert til en form som er kompatibel med Python.

Når vi endret form på dette rammeverket, er det flere problemstillinger som oppsto, og deretter ble løst.

Noen av problemene som oppsto og ble løst var:

- Hente verdier fra EV3-enheten (MicroPython).
- Overføre data til PC (Socket-tilkobling).
- Plotte data (MatPlotLib).
- Finne en prosjektstruktur som skal være pedagogisk å bruke for nye studenter (Bruk av lister eller dictionary).

Etter rammeverket kom på plass, utførte vi ett par prosjekt slik som en ny student må gjøre. Dette ble gjort for å sjekke kompatibiliteten på utførelsen, og for å oppdage problemer som studenter muligens kan møte på.

Innhold

Kapittel 1

Introduksjon

1.1 Oppgavebeskrivelse

Oppgaven består av å gjøre klar et rammeverk for ELE-130 som er et nytt fag for 2. semesterstudenter fra 2022 av for data- og elektrostudenter på UiS. Dette faget vil være en oppdatert versjon av ING-100 (Ingeniørfaglig innføringsemne - Data og elektro), som er et fag som alle 1. semesterstudenter i ingeniørfagene på UiS måtte ta. Emnebeskrivelsen til ING-100 for studieåret 2020-2021 er oppsummert i denne setningen på UiS sin webside:

> Emnet presenterer dataprogrammering i form av en grunnleggende innføring i MATLAB samt et prosjekt med fokus på programmering av roboter.[15]

I tillegg til å lære grunnleggende programmering ble det i ING-100 faget vist hvordan en kunne anvende matematikk og fysikk for å få programmere roboten. En dypere forklaring på gjennomføringen av ING-100 blir gitt i delkapittel 1.2.

Nye data/elektro - studentene som begynner på UiS fra 2022 av vil få en innføring i Python (DAT-120 Grunnleggende programmering) i 1. semester,

og vil deretter ta ELE-130 i 2. semester. Dette betyr at elevene i ELE-130 allerede har hatt ett semester med programmering, noe gamle ING-100 studenter ikke hadde. En del av emnebeskrivelsen for ELE-130 for studieåret 2021-2022 er gitt som følger på UiS sin webside:

> Anvende matematikk- og fysikkunnskaper til å løse ulike problemstillinger i robotprogrammering. Forstå og kunne forklare begrepene numerisk integrasjon, filtrering og numerisk derivasjon, samt kunne implementere og bruke disse numeriske metodene i MATLAB og Python.[5]

Denne oppgaven går ut på å gjøre klar prosjektdelen i dette nye faget. Ettersom undervisningen fortsetter i Python, og studentene allerede har hatt ett fag i Python før dette, er det naturlig at prosjektet også skal utføres ved bruk av Python. Prosjektet i ING-100 som var MATLAB-basert er da nødt til å oversettes til Python slik at det i ELE-130 skal være mulig å bruke enten MATLAB eller Python. Siden dette faget henter inn mer kunnskap om fysikk, blir det heretter mulig for fremtidige studenter å oppnå sivilingeniørstatusen, ettersom kravet på 7.5 studiepoeng i fysikk blir dekket. Dette var ikke mulig med ING-100 faget.

1.2 ING-100

I ING-100 blir elever delt inn i grupper på 3 eller 4 studenter hvor hver gruppe får utdelt en EV3-robot som kan bli programmert ved hjelp av MATLAB. Ved å utføre prosjekt på denne roboten ved hjelp av MATLAB vil studenter få en innføring i programmering samt en dypere forståelse for hvordan matematikk og fysikk kan brukes praktisk. Studenter får utdelt en omfattende prosjektbeskrivelse som inkluderer informasjon om EV3-roboten, MATLAB og selve utførelsen av prosjektene. Denne prosjektbeskrivelsen finnes i vedlegg K.

Som en motivasjon var en del av prosjektet lagt opp som en uformell konkurranse hvor man skal benytte en styrestikke til å kjøre en LEGO-robot langs en bane, og hvor målet er å gjøre dette på best mulig måte. Hva som er "best mulig" blir avgjort av endel forskjellige kvalitetsmål.

Hovedfokuset i prosjektet er å ta tak i forskjellige (enkle og kompliserte) praktiske problemstillinger, analysere disse og deretter bruke kunnskap fra matematikk og fysikk til å programmere en løsning på problemstillingen. En løsning kan typisk bestå av

1. Et MATLAB-program (Dette skal gjøres om til Python-program til det nye faget)
2. En LEGO-konstruksjon som utfører de forskjellige instruksjonene som er implementert i MATLAB-programmet.

Ofte er det mulig å utvide problemstillingene slik at man kan lage mer avanserte programmer av samme LEGO-konstruksjon.

1.2.1 Prosjektene

Prosjektet består av obligatoriske og frivillige deloppgaver. De obligatoriske prosjektene er grunnlaget for prosjektet i sin helhet. Om studentene ikke klarer å utføre disse prosjektene på en skikkelig måte, er det ikke nok grunnlag for at de kan stå i faget.

1.2 ING-100

Obligatoriske prosjekt:

1. Numerisk Integrasjon
2. Filtrering
3. Numerisk Derivasjon
4. Manuell kjøring

De obligatoriske prosjektene skal hjelpe studentene med å forstå konseptene bak signalbehandling og få enn innføring i hvordan fysikken kan bli brukt i praksis.

De frivillige prosjektene er eksempler på hva studentene kan finne på med roboten, etter de har gjort og forstått den obligatoriske delen av prosjektet. Online betyr at roboten skal være koblet til pc, mens prosjektene blir utført. Offline betyr at roboten kan ha kjørt tidligere, og samlet data i en fil som senere kan bli kjørt for å utføre tester. Her kommer ett par eksempler som studentene kan gjøre for å fordype sin kunnskap:

Eksempler på små prosjekt som kan bli utført:

1. Automatisk kjøring med P-regulator (ONLINE)
2. Rekkefølge derivering/integrering (OFFLINE)
3. Rekkefølge derivering/filtrering (OFFLINE)
4. Eulers forovermetode/trapesmetoden (OFFLINE)
5. Numerisk løsning av differensialligninger (OFFLINE)
6. Støvsuger med krasjdetektor (ONLINE)
7. Kreativ komponist (OFFLINE)
8. Filtrering av sammensatt signal (ONLINE)
9. Effekten av samplingsrate på IIR-filtrering (OFFLINE)
10. Søk og lokalisér (ONLINE)

11. Kjør roboten med trykknappen (ONLINE)

Eksempler på større prosjekt som også kan bli utført:

1. Turtallsregulering av LEGO-motor (ONLINE)
2. Automatisk kjøring med PID-regulator (ONLINE)
3. Tegn bane (OFFLINE)
4. Rotasjonshastighet ved bruk av enkoder (OFFLINE)
5. Estimer hastighet (OFFLINE)
6. Katapult (ONLINE)
7. Sykkelcomputer (OFFLINE)
8. Frekvensrespons (OFFLINE)
9. Nedfolding (OFFLINE)
10. Adaptiv cruisekontrol (ONLINE)
11. Pappeskeareal (OFFLINE)
12. Selvbalanserende Robot (ONLINE)

Alle prosjektene er eksempler på praktiske mulige oppgaver der studentene kan fremme sin kunnskap om matte, fysikk og programmering.

1.3 Mål

Målet med oppgaven er å gjør klar ett opplegg som skal kunne bli utført på EV3-roboten i Python. Prosjektet skal være likt som det prosjektet som har blitt utført i ING-100 tidligere. Eneste endringen vil være oergang til Python fra MATLAB. Det vil da være nødvendig å lage et grunnlag for hvordan prosjektet skal bli dokumentert og utført. Grunnbasisen for prosjektet leveres som en del av oppgaven. Dette sammen med en fasit på hvordan de obligatoriske prosjektene kan bli utført på denne nye måten. Det vil dermed bli levert 5 forskjellige mapper med kode for utførselen til oppgaven. Koden kommer til å ligge i en GitHub som er linket vedlegg K. Her kommer en liste over filene som blir levert sammen med bacheloroppgaven:

1. Prosjekt 0X (Kapittel: 3)
2. Prosjekt 01: Numerisk Integrasjon (Vedlegg: G)
3. Prosjekt 02: Filtrering (Kapittel: 5)
4. Prosjekt 03: Numerisk Derivasjon (Vedlegg: H)
5. Prosjekt 04: Manuell Kjøring (Vedlegg: I)

Forklaringer på de forskjellige mappene blir henvist til tilhørende kapitler.

1.4 Utviklingsverktøy

Programmeringsspråket som blir benyttet er Python. Python blir brukt til selve hovedoppgaven, og alle programmer som blir utført av pc-en, er Python-programmer. Det blir også brukt MicroPython, dette er på selve hjernen til EV3-roboten, fordi roboten har ikke tilgang til Python. Programmet som er brukt til å programmere i, er visual studio code. Programmet ble valgt ettersom det har blitt laget en utvidelse som blir benyttet i denne oppgaven. Utvidelsen heter "MicroPython for LEGO® MINDSTORMS® EV3", og er utvidelseen som trengs for at utførelsen av denne oppgaven overhode er mulig. Dette blir gått mer i dybden i kapittel: 2.

1.4.1 Benyttet utstyr

I denne oppgaven benyttes et grunnsett med en Lego EV3-robot og tilhørende sensorer. Det er også benyttet to forskjellige styrestikker, slik at det det kunne lages to ulike rammeverk. Grunnsettet blir beskrevet mer i vedlegg: F, her blir alle elementene om utstyret forklart.

Kapittel 2

Python på EV3en

I dette kapitlet gis det en innføring i oppsett og kjøring av Pythonkode på EV3 roboten. Blant annet vil det vises hvordan en kan klargjøre et nytt operativsystem på EV3en slik at det er mulig å kjøre Pythonkode. Programstrukturen, filene og mappene, som er grunnlaget for de videre prosjektene vil også gjennomgås her.

2.1 ev3dev

MicroPython er en implementering av Python som gjør det mulig å kjøre Pythonkode på mikrokontroller, inkludert EV3 [19]. For å kunne kjøre MicroPython på EV3en, brukes det ev3dev, som er et Debian-Linux type operativsystem som kan kjøre på produkter i LEGO MINDSTORMS serien [11]. Ved hjelp av Visual studio er det mulig å skrive Pythonkode på datamaskinen som deretter overføres til EV3en.

På hjemmesiden til MicroPython beskrives programmet som følger:

"*MicroPython is a lean and efficient implementation of the Python 3 programming language that includes a small subset of the Python standard library and is optimised to run on microcontrollers and in constrained environments.*"[18]

For å bruke MicroPython på EV3en, brukes et microSD-kort som settes inn i SD porten på EV3en. I figur 2.2 vises SD-porten til EV3en med et microSD-kort installert. microSD-kortet må være av typen microSDHC som er 'et' microSD kort med en kapasitet på mellom 2GB til 32GB. MicroSD kort med høyere kapasitet enn dette kalles for microSDXC, og dette er en litt annen teknologi og støttes ikke av EV3en.

2.2 Installasjon av MicroPython og oppstart av EV3en

For å kjøre MicroPython på EV3en må Visual Studio Code installeres på en datamaskin som brukes for å kommunisere med EV3en. For en kort innføring til Visual Studio Code, se vedlegg B. Visual Studio Code har en utvidelse (Eng: *Extension*) som er utgitt av LEGO som er veldig gunstig å bruke for MicroPython på EV3en. Denne utvidelsen gjør det mulig å opprette en direkte kobling til EV3en som da åpner opp for å jobbe direkte med filsystemet til ev3dev-operativsystemet på EV3en. Oppsettet av utvidelsen, som heter "LEGO MINDSTORMS EV3 MicroPython", er også ganske enkelt.[1] Etter installasjon av Visual Studio Code må utvidelsen "LEGO MINDSTORMS EV3 MicroPython" installeres. Utvidelsen er vist i figur 2.1, den røde sirkelen viser ikonet for utvidelser i Visual Studio Code.

Figur 2.1: Utvidelsen LEGO MINDSTORMS EV3 MicroPython i Visual Studio Code.

Klargjøringen av microSD-kortet er som følger:

1. Last ned **"EV3 MicroPython micro SD card image"** fra www.lego.com. Bla litt ned på siden så finnes fila under **"MIND-**

[1] Det er mulig å sette opp et miljø for utvikling av programvare til EV3en i PyCharm, se for eksempel https://www.ev3dev.org/docs/tutorials/setting-up-python-pycharm/

STORMS EV3 Python Documentation and Firmware ". Dette er en zip-fil som ikke skal pakkes ut etter nedlasting.

2. Deretter trengs det et program som kan flashe et operativsystem til SD-kort. Et godt alternativ her et Etcher som kan lastes ned fra www.balena.io (klikkbar lenke).

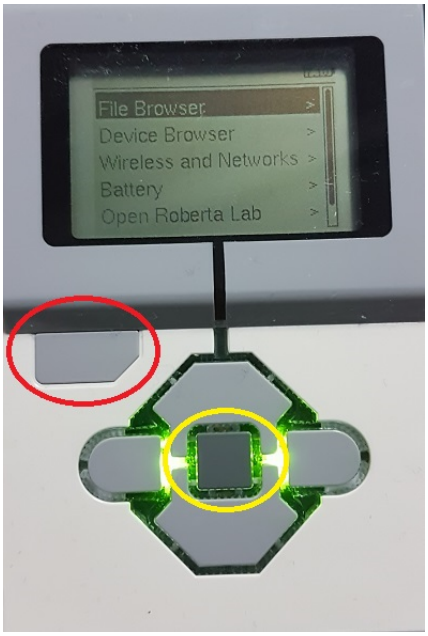
3. Sett microSDHC kortet inn i PC-en, enten via en SD port (hvor microSDHC kortet er satt inn i en SD-adapter) eller via en ekstern minnekortleser.

4. I Etcher velges image (zip-fila EV3 MicroPython micro SD card image som ble lastet ned). Deretter velges stasjonen (Eng: drive) hvor microSDHC kortet er montert (Eng: mounted), også trykker en på flash. Dette kan ta litt tid, og det er viktig at kortet ikke tas ut helt til flasheprosessen er ferdig.

Når kortet er klart er det bare å sette det inn i SD-porten på EV3en som vist i figur 2.2. For å starte opp EV3en holdes på-knappen, indikert med gul ring i figur 2.3, inne til det blinker rødt. Deretter vil det ta ca ett minutt til EV3en er klar for bruk. For å skru av enheten, trykker en på knappen indikert med rød ring i figur 2.3 og deretter velges "Shutdown" alternativet.

Figur 2.2: SD-porten på EV3en med et microSDHC kort installert.

EV3en kjører MicroPython kun via microSD-kortet. Dermed er det bare å ta ut kortet for å gå tilbake til standardoppsettet til EV3en. Å ta ut kortet er ikke lett, og det må muligens brukes en liten tang eller et lignende verktøy.



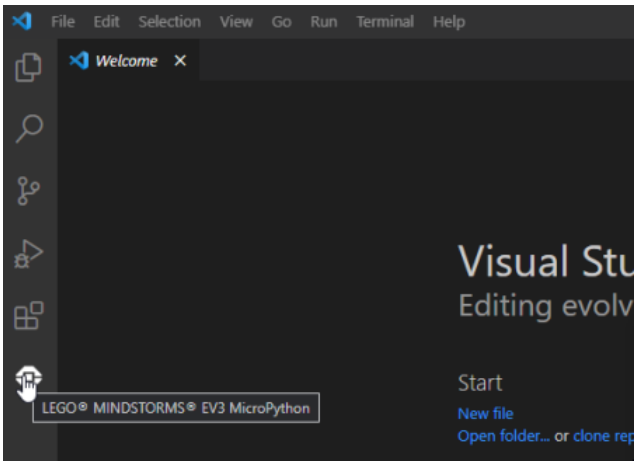
Figur 2.3: Grensesnittet til MicroPython på EV3en.

2.3 Kjøring av program på EV3en

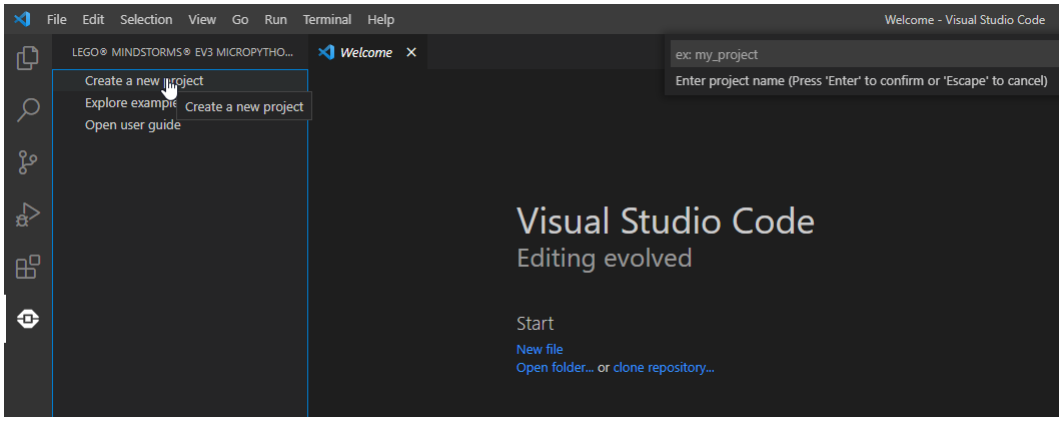
Denne seksjonen demonstrerer hvordan et Pythonprogram kjøres på EV3en. Det vises først hvordan et program blir forberedt på en datamaskin, også vises det hvordan programmet overføres til EV3en, for så å bli kjørt. Programmene som kjøres vil bli kjørt på to måter; enten direkte på EV3en (altså ved å bruke EV3ens knapper til å kjøre programmet), eller via en datamaskin (såfremt EV3en er tilkoblet datamaskinen).

2.3.1 Opprettelse av program på datamaskin

Selve programmeringen av program som kjøres på EV3en skjer på en datamaskin. Først og fremst må "LEGO® MINDSTORMS® EV3 MicroPython" utvidelsen åpnes. Det gjør en ved å klikke på ikonet vist i figur 2.4. Deretter klikker en på "Create a new project"og gir det et nytt navn, slik som vist i figur 2.5.

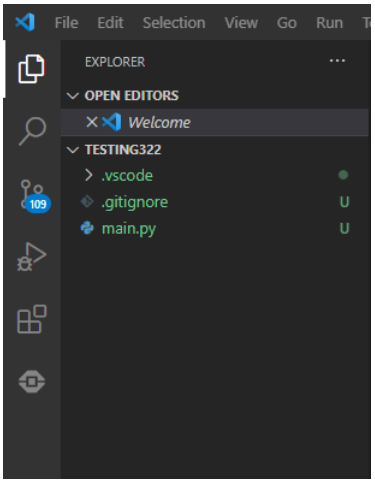


Figur 2.4: Ikonet til utvidelsen "LEGO® MINDSTORMS® EV3 MicroPython".



Figur 2.5: Opprettelse av et nytt prosjekt for EV3en i Visual Studio Code.

Resultatet er prosjekstrukturen i figur 2.6, hvor hovedfilen main.py er av interesse, siden dette er fila som blir kjørt av EV3en etter overføring.



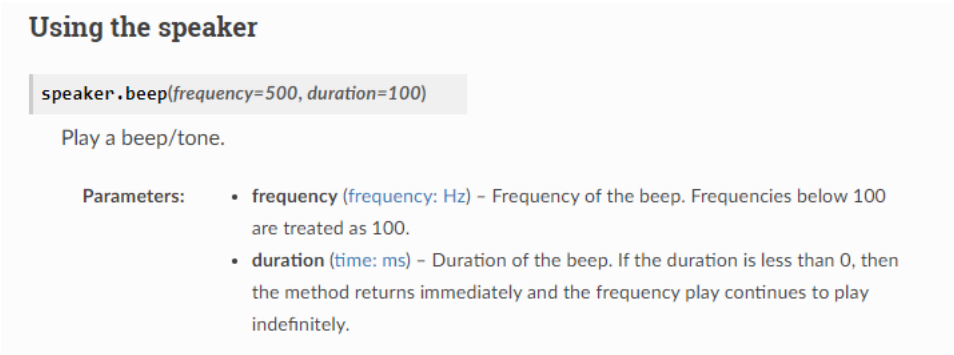
Figur 2.6: Resulterende prosjekstruktur i Vistual Studio Code etter opprettelse av nytt prosjekt.

Hver gang et nytt prosjekt startes, vil innholdet i main.py være som følger:

Kode 2.1: Initiell kode i main.py ved opprettelse av nytt prosjekt

```
1  #!/usr/bin/env pybricks-micropython
2  from pybricks.hubs import EV3Brick
3
4  from pybricks.ev3devices import (Motor, TouchSensor,
5  ColorSensor, InfraredSensor,
6  UltrasonicSensor, GyroSensor)
7
8  from pybricks.parameters import (Port, Stop, Direction,
9  Button, Color)
10
11 from pybricks.tools import wait, StopWatch, DataLog
12 from pybricks.robotics import DriveBase
13 from pybricks.media.ev3dev import SoundFile, ImageFile
14
15
16 # This program requires LEGO EV3 MicroPython v2.0 or higher.
17 # Click "Open user guide" on the EV3 extension tab for ...
       more information.
18
19
20 # Create your objects here.
21 ev3 = EV3Brick()
22
23
24 # Write your program here.
25 ev3.speaker.beep()
```

Som vi ser på linje 2 til 13 importerer fila mye fra `pybricks`-modulene, som sørger for at det er mulig å kommunisere mellom EV3en og programmene som skrives. Mer om disse modulene kan leses på https://pybricks.github.io . Etter importsetningene lages et objekt fra `EV3Brick`-klassen (en del av `pybricks` modulene) på linje 21, dette objektet blir kalt `ev3`. `ev3` er veldig viktig, siden all kommunikasjon mellom EV3en og programmet vil foregå via dette objektet. I linje 25 blir ev3 objektets `speaker.beep()` metode brukt; denne metoden produserer en liten pipelyd fra robotten som varer i et lite øyeblikk. Hva denne metoden gjør helt spesifikt kan finnes på `pybricks`-sida (https://pybricks.github.io), under `hubs - Programmable Hubs`; innholdet er gjengitt i figur 2.7.



Using the speaker

speaker.beep(*frequency=500, duration=100*)

Play a beep/tone.

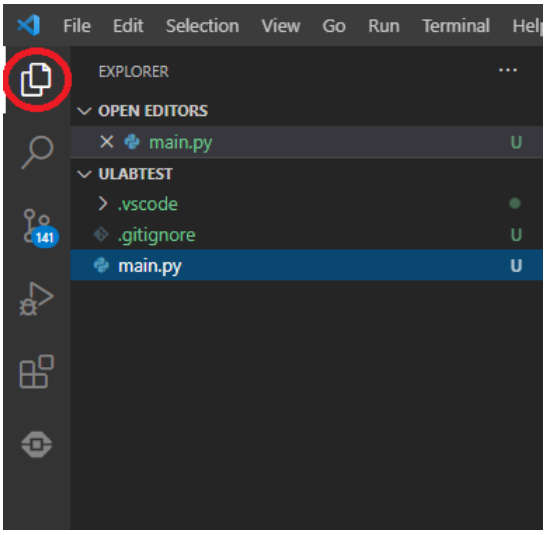
Parameters:

- **frequency** (frequency: Hz) – Frequency of the beep. Frequencies below 100 are treated as 100.
- **duration** (time: ms) – Duration of the beep. If the duration is less than 0, then the method returns immediately and the frequency play continues to play indefinitely.

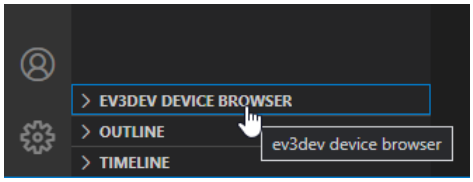
Figur 2.7: Beskrivelsen av speaker.beep()-metoden til EV3Brick-klassen slik den er på https://pybricks.github.io. Blant annet vil metoden ha standardparameter (Eng: *default parameters*) `frequency`=500 og `duration=100`.

2.3.2 Opprettelse av en tilkobling fra datamaskin til EV3en

For å kunne kjøre dette programmet på EV3en, må det først opprettes en kobling mellom PCen og EV3en i Visual Studio Code. Først og fremst må det være en kablet forbindelse mellom PCen og EV3en. Etter dette må explorer-taben være åpen i Vistual Studio Code. Ikonet for explorer-taben er vist i figur 2.8, og ved å klikke på ikonet vil programmet åpnes. På Windows kan en også bruke Ctrl+Shift+E som en snarvei for å åpne explorer-taben.

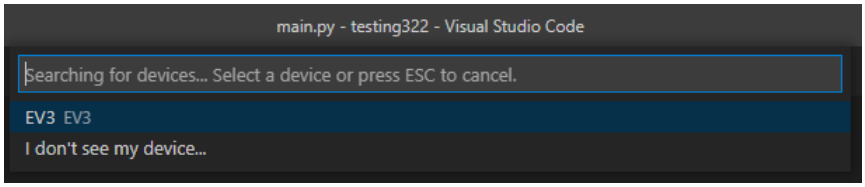


Figur 2.8: Ikonet til explorer-taben i Visual Studio Code.



Figur 2.9: EV3Dev Device Browser. Etter å ha åpnet explorer-taben som vist i figur 2.8 vil denne figuren vise i nedre venstre hjørne.

Etter å ha klikket på "EV3DEV DEVICE BROWSER", kommer det opp en knapp hvor det står "Click here to connect to a device". Ved å klikke på "Click here to connect to a device" skal innholdet i figur 2.10 komme frem øverst i Visual Studio Code. Her skal EV3-enheten viser. Navnet på EV3en kan avvike fra navnet vist i figur 2.10, men dette har ingen betydning.

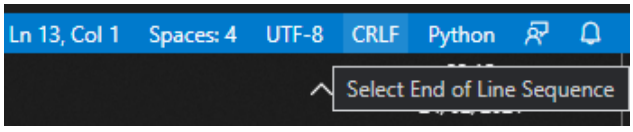


Figur 2.10: Navnet til EV3-enheten som vist i Visual Studio Code. Ved å klikke på navnet vil Visual Studio Code opprette en direkte tilkobling til enheten.

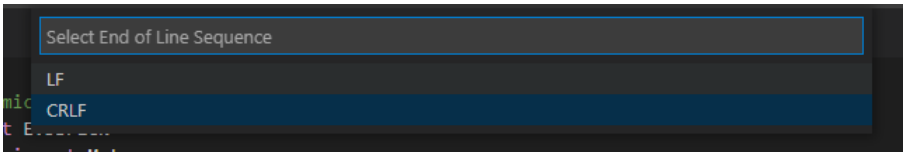
Ved å klikke på enheten og deretter vente i et par sekunder, vil koblingen opprettes, som kan bekreftes under "EV3DEV DEVICE BROWSER" i nedre venstre hjørne. Dette er vist i figur 2.13.

Før programmet sendes til EV3en, er det viktig å sette "End of Line Sequence" lik `LF` for å unngå kjørefeil. "End of Line Sequence" indikerer hvilke tegn som brukes for å markere avsluttingen på ei linje. `LF` ("Line Feed") indikerer at linjer avsluttes med `\n`. `CRLF` ("Carriage Return, Line Feed") bruker `\r\n` for å markere avsluttingen på ei linje. Windows bruker `CRLF`, mens Unix-varianter bruker `LF`. Siden EV3en kjører et Debian-basert operativsystem (Debian er en variant av Linux, som igjen er Unix-liknende), er det viktig at det brukes `LF`.

For å skifte til LF må det først klikkes på ikonet vist i figur 2.11. Dette ligger helt nederst til høyre i Visual Studio Code. Etter å ha klikket på ikonet velges `LF` øverst i Visual Studio Code som vist i figur 2.12.



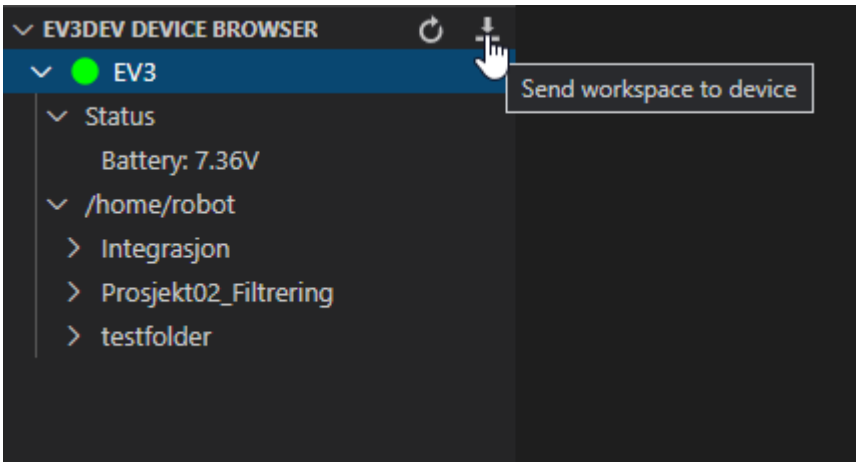
Figur 2.11: Ikonet som indikerer hvilken "End of Line Sequence" som brukes. Ved å klikke på ikonet kan en skifte "End of Line Sequence".



Figur 2.12: Her skal `LF` velges som "End of Line Sequence".

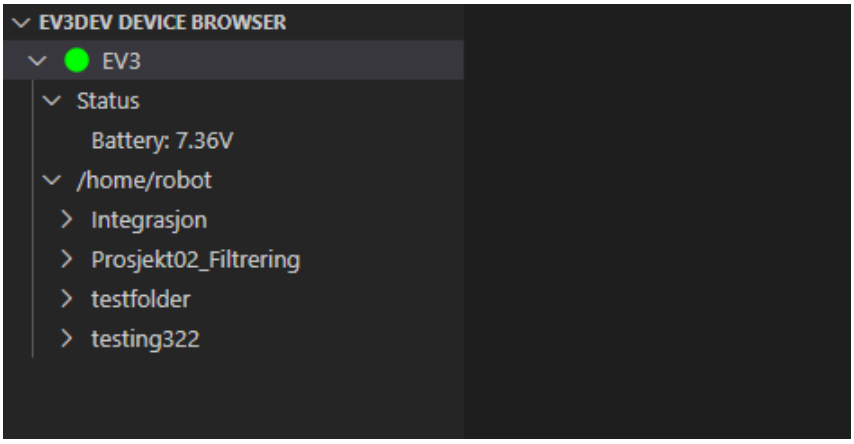
2.3 Kjøring av program på EV3en

For å laste ned program til EV3en er det bare å klikke "Send workspace to device", som er vist i figur 2.13. Her er det viktig å være klar over at hvis prosjektnavnet allerede finnes som en mappe på EV3en, vil denne mappa bli overskrevet etter overføring. Prosjektnavnet ble valgt da prosjektet ble opprettet, som vist i figur 2.5. Der ble prosjektnavnet `testing322` brukt. I figur 2.13 kan en se at det allerede ligger tre mapper (Integrasjon, Prosjekt02_Filtrering, og testfolder) på denne EV3en (dette er mapper som er lagt til manuelt, det er ikke snakk om noen standardmapper).



Figur 2.13: Filstrukturen til EV3en som vist i Visual Studio Code.

Etter at "workspacen", med `testing322` mappen er sendt til EV3en, vil filstrukturen til EV3en se ut som i figur 2.14

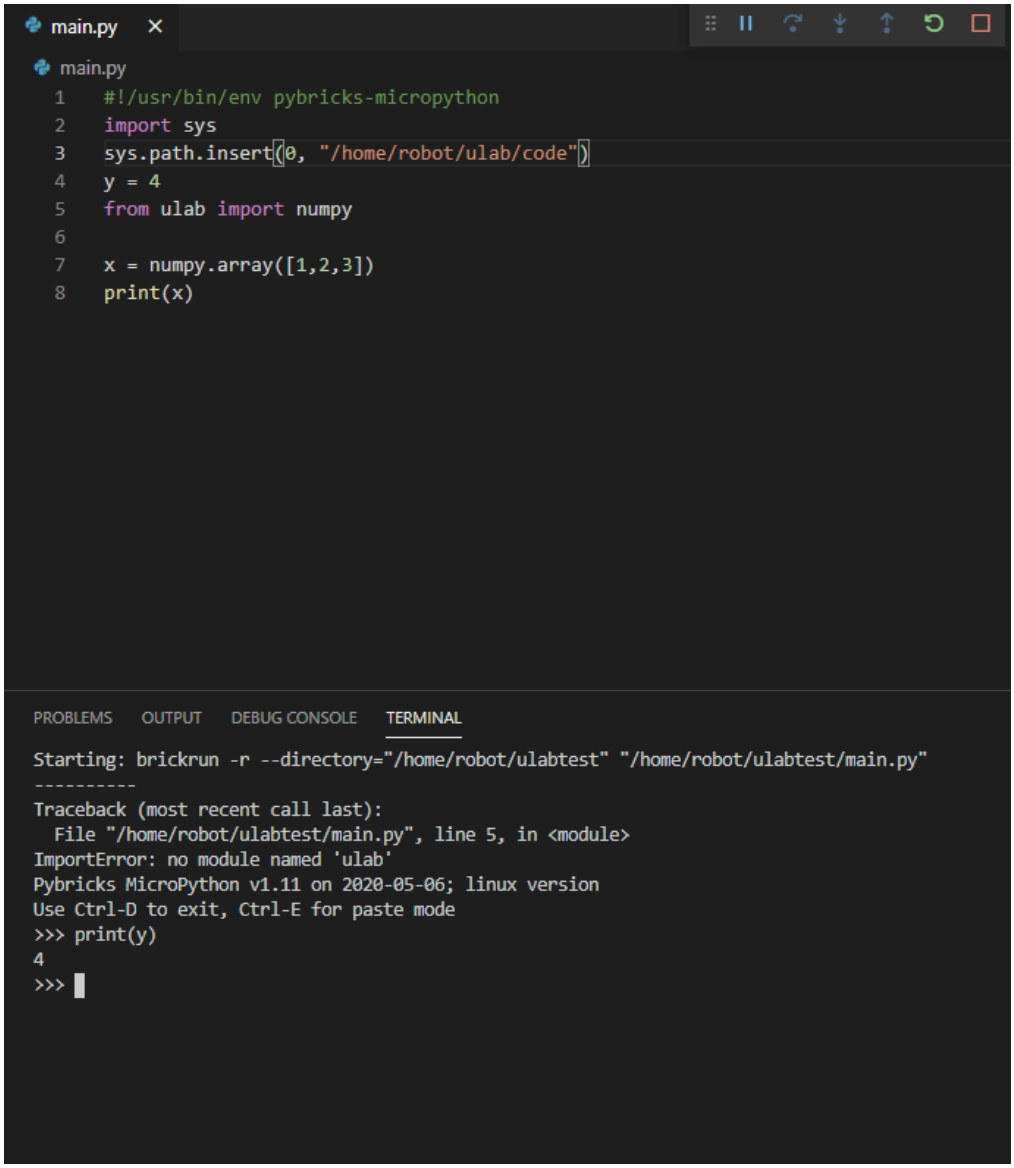


Figur 2.14: Filstrukturen til EV3en som vist i Visual Studio Code, etter nedlasting av `testing322` mappen.

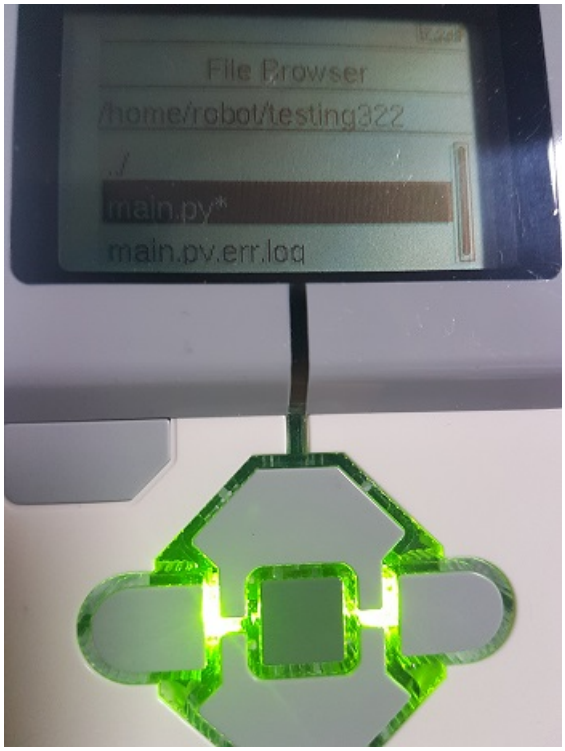
2.3.3 Kjøring av program på EV3en

Programmet `main.py` som ble overført til EV3en kan nå kjøres på to måter:

1. Det kan enten kjøres via Visual Studio Code ved å første klikke på den relevante mappen (`testing322` i dette tilfellet), også klikker en på hovedfila (`main.py` i dette tilfellet), og deretter velger en `Run` alternativet som kommer opp. Det er to `Run` alternativ å velge mellom:
   (a) `Run`: dette alternativet kjører programmet som et vanlig python-program.
   (b) `Run in interactive terminal`: dette alternativet kjører også programmet som vanlig, men det åpner også en *interactive terminal* som kan være til stor nytte for blant annet avlusing. I denne terminalen er det mulig å kjøre pythonkode; dette kan gjerne minne om IDLEen (*Integrated Development and Learning Environment*) som kommer med Python. Et eksempel som viser avlusing er vist i figur 2.15.
2. Fila kan også kjøres direkte på EV3en ved å gå igjennom filsystemet til EV3en, som vist i figur 2.3. Ved å klikke på `File Browser`, også finne den relevante mappen (`testing322` her), også klikke på main.py vil programmet også kjøres. Dette er vist i figur 2.16.



Figur 2.15: Eksempel på bruk av `Run in interactive terminal`. Det oppstår en feil på linje 5, men det er fortsatt mulig å se hva som har skjedd før det. I figuren blir variabelen y, deklarert på linje 4, printet ut.



Figur 2.16: Kjøring av program direkte fra EV3en.

2.4 Hvordan finne IP adressen til EV3en

I MATLAB prosjektene fra ING100 faget, ble det brukt en god del "live"-plotting av dataene (inkludert beregnede verdier) som ble hentet fra sensorene til EV3en. Dette gjorde det mulig for oss studenter å se på hvordan funksjonene våres reagerte på endringer i dataene som ble mottatt fra sensorene. Dette var et veldig godt pedagogisk virkemiddel for oss studenter, og vi fikk ganske god utnytte av dette. For å muliggjør dette i vårt prosjekt med python, er det nødvendig å opprette en form for kommunikasjon mellom EV3en og en datamaskin som plotter dataene til en skjerm. Dette vil vi gjøre ved bruk av socketforbindelser, og hvordan vi muliggjør dette vil komme i kapittel 4 hvor det blir presentert rammeverk for prosjektene som utføres. Mer generelt om socketer kan leses om i vedlegg C.

For å kunne bruke socketer som kommunikasjonsverktøy mellom datamaskinen og EV3en, er det viktig for datamaskinen å kunne vite hvilken IP adresse EV3en bruker. For å finne IP adressen til EV3en gjør en som følger:

1. Start opp EV3en, og koble den til en datamaskin ved hjelp av en USB-kabel.
2. Fra startskjerm som ble vist i figur 2.3, gå til `Wireless and Networks`.
3. Fra `Wireless and Networks` gå helt ned til `Offline Mode` og sørg for at dette alternativet ikke er huket av.
4. Deretter må en gå opp til `All Network Connections` alternativet også gå inn der.
5. Fra `All Network Connections` klikker en inn på `Wired` og klikker på `Connect`.

Etter å ha gjennomført stegene, vil IP adressen komme opp øverst på skjermen til EV3en.

2.5 Styrestikker som brukes i prosjektet

Med legosettet som er benyttet, var det to tilhørende styrestikker som brukes for å gi kommandoer til EV3en. Den første styrestikken, som også ble brukt i ING100, er fra Logitech og er vist i figur 2.18. Den andre er en nyere styrestikk fra Dacota Gaming, denne er vist i figur 2.17.

Figur 2.17: Styrestikken fra Dacota Gaming.

Figur 2.18: Styrestikken fra logitech.

Styrestikkene kobles direkte til EV3en via USB-porten, som vist i figur 2.19.

Figur 2.19: USB-porten på EV3en hvor det kobles til en styrestikk.

I kodeutdrag 2.2 er det vist et program som brukes for å identifisere hvilken styrestikk som er koblet til EV3en. Verdiene som styrestikkene gir til EV3en ved bevegelse på styrestikken er ulikt fra styrestikk til styrestikk; for eksempel gir Logitech styrestikken 1024 ved max "forover", mens Dacota gir kun 255. Dermed må en identifisere hvilken styrestikk som er koblet til, for så å skalere ned til felles verdi, slik at samme kode kan brukes for begge styrestikkene.

Identifikasjon av styrestikken gjøres ved å sammenligne en byte i filsystemet til EV3en (gitt at styrestikken er koblet til EV3en). Ved å koble til styrestikken vil en spesifikk fil bli opprettet i filsystemet. Navnet på stien til fila vil kunne variere litt mellom hver kjøring (for eksempel `/dev/bus/usb/001/329` ved en kjøring, også `/dev/bus/usb/001/004` på neste kjøring). Dermed brukes det ei `for`-løkke på linje 2 i kodeutdrag 2.2 som sørger for at alle mulige navn blir prøvd ut.

På linje 6 blir stien til styrestikken på filsystemet til EV3en lest inn i en variabel med navnet `joy`. `joy` vil da være av typen `bytes`. Ved hjelp av indeksering går det ann å lese av disse verdiene, som vil være faste (og ulike) for hver styrestikk. Dermed gjelder det å finne en byte hvor det er forskjell mellom de to styrestikkene. Den første forskjellen skjer i `joy[2]`, hvor `joy[2]=16` for Logitech-styrestikken, mens `joy[2]` er lik `0` for styrestikken fra Dacota Gaming.

Det er mulig at en ny styrestikk (forskjellig fra Logitech- og Dacota-styrestikken) som kobles til vil kunne gi `joy[2]` lik `0` eller `16`, og da vil programmet identifisere den nye styrestikken som "dacota" (`joy[2]=0`) eller "logitech"

(`joy[2]=16`). Dermed er det viktig å presisere at hvis det kobles til en ny styrestikk, så må det en ny test til. Det vil da kun være nødvendig med en byte hvor alle tre styrestikkene er forskjellig. Dette finner en enkelt ved å printe ut verdiene i `joy`.

Kode 2.2: `identifyJoystick.py`

```
1  def identifyJoystick():
2      for i in range(2, 1000):
3      # path er lokasjonen til styrestikk-informasjonen på EV3en
4      # denne vil variere litt, slik at det loopes gjennom ...
           mulige verdier (linje 2)
5      # for å unngå feilmeldinger, brukes en try/except blokk
6          path = ("/dev/bus/usb/001/{:03d}".format(i))
7          try:
8              # prøver å åpne styrestikk-info
9              with open(path, "rb") as f:
10                 # leser styrestikk-info inn til variabelen joy
11                 joy = f.read()
12                 if joy[2] == 16:  # identifikator for ...
                       logitech styrestikk
13                     return "logitech"
14                 elif joy[2] == 0: # identifikator for ...
                       dacota gaming styrestikk
15                     return "dacota"
16                 else:             # ukjent styrestikk
17                     return "Ukjent joystick."
18             break
19         except:
20             pass
```

Kapittel 3

Basisstrukturen for et prosjekt på EV3en

Som en del av oppgaven ble det undersøkt ulike rammeverk/strukturer for et python-prosjekt på EV3en for bruk i det nye faget. I dette kapitlet presenteres det to ulike rammeverk som ble produsert, og som kan være relevant ved overgang til nytt fag. Hovedforskjellene mellom de to rammeverkene, ligger i bruken av lister kontra dictionary (første strukturen som presenteres bruker lister oftere), og hvor mange filer som brukes (første struktur har 2 filer, mens den andre strukturen har hele 10 filer).

For å konkretisere koden og vise hvordan rammeverkene brukes i praksis vil integrasjonsprosjektet bli benyttet som eksempelkode i dette kapittelet. Integrasjonsprosjektet er ett av de obligatoriske oppgavene fra ING100 faget, og blir også gjennomgått i vedlegg G i dette skrivet. Ved å velge et spesifikt prosjekt som eksempel vil vi gi en konkret demonstrasjon på hvilke oppgaver de forskjellige delene av koden gjennomfører, spesielt med tanke på hvilke sensorer som brukes (lyssensoren), og hvilke verdier som beregnes basert på verdiene fra lyssensoren ("flow" og "volume").

Begge rammeverkene kan kjøres i enten online- eller offline-modus. Ved kjøring i online-modus blir det mulig med live plotting av data som hentes fra sensorene på EV3en. Datamaskinen mottar data fra EV3en og plotter dataene i tilnærmet sanntid (dersom for mye data sendes mellom EV3 og

datamaskin vil plottingen henge etter). Ved kjøring i offline-modus henter EV3en inn data fra sensorene som lagres lokalt på EV3en. Målingene kan i ettertid hentes ut fra EV3en og bearbeides på datamaskinen.

Når det kjøres i online-modus blir det anvendt socketer for å opprette en "portal" for kommunikasjon mellom EV3en og datamaskinen. En innføring til socketer finnes i vedlegg C.

3.1 Basisstrukturen for et EV3-prosjekt ved bruk av lister

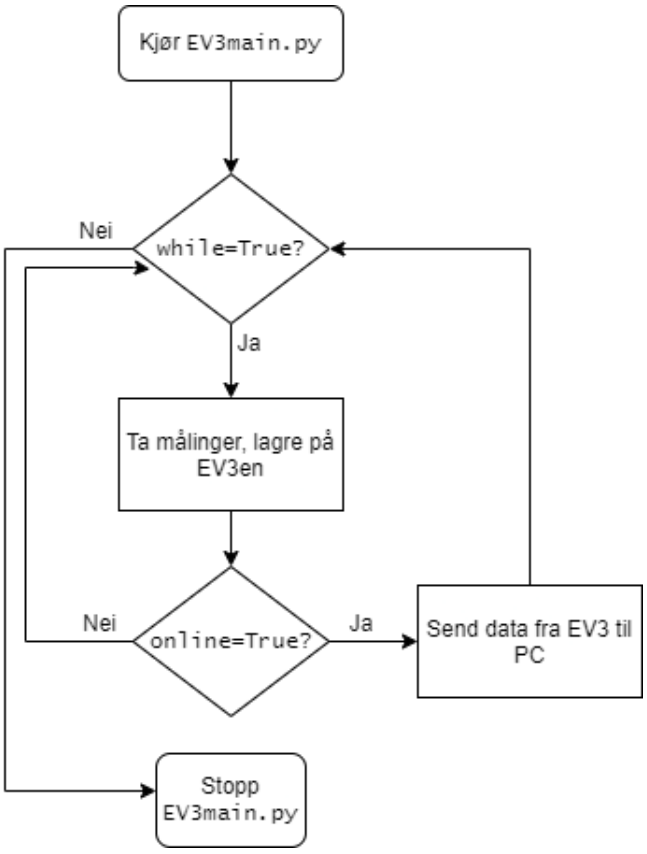
Først presenteres strukturen som er benyttet videre i prosjektet, blant annet for å gjøre de obligatoriske oppgavene fra ING100. I løpet av gjennomføring av obligatoriske oppgaver fra ING100 har forbedringer blitt gjort på denne strukturn, den er derfor me utviklet enn strukturen basert på flere filer og dictionaryer. Denne prosjektstrukturen består av 2 filer; `EV3main.py` som kjøres på EV3en, og `BeregnOgPlott.py`, som kjøres på datamaskinen. `EV3main.py` står for klargjøring og kjøring av EV3en, som inkluderer innhenting av data fra sensorer og styrestikk (hvis robot er koblet til). `BeregnOgPlott.py` som kjører på datamaskinen gjør beregninger på dataene som ble hentet fra EV3en (via `EV3main.py`), og plotter da dataen.

3.2 `EV3main.py`

`EV3main.py` er fila som kjører på EV3en, og må overføres på ny til EV3en for hver gang koden endres på datamaskinen. Kontrollflyten for programmet er vist i figur 3.1, og som indikert i figuren er det en `while`-løkke i `EV3main.py` som kjører helt til programmet avsluttes. Programmet kan avsluttes på to måter: Ved å trykke på tilbake-knappen på EV3en (den grå femkantete knappen nede til venstre for skjermen til EV3en, vist i figur 2.16). Gitt at det er en styrestikk koblet til EV3en kan programmet kan også avsluttes ved å trykke på en knapp på styrestikken.

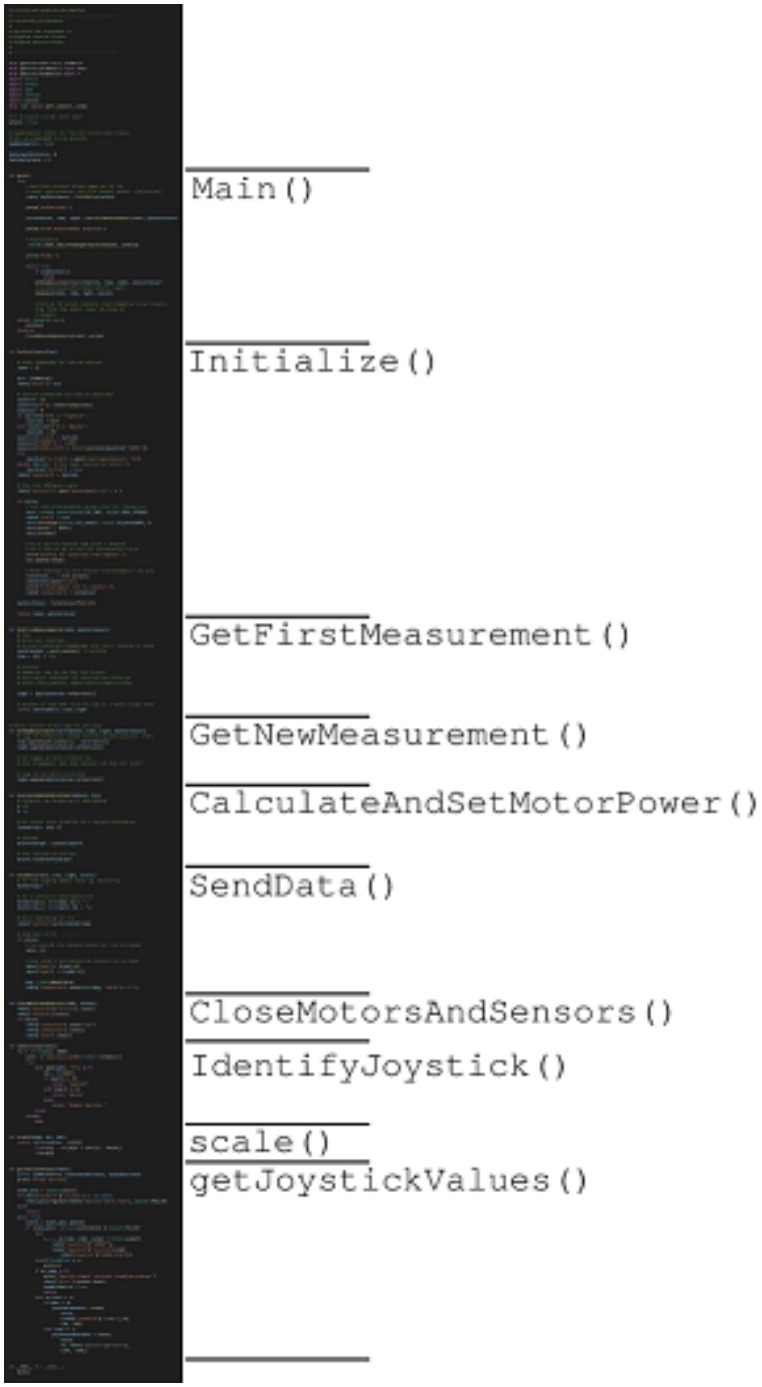
Som kontrollflyten i figur 3.1 viser, vil programmet kontinuerlig ta målinger

og lagre de på EV3en helt til programmet avsluttes. I `EV3main.py` er det en boolsk variabel, `online`, som settes til `True` dersom det kjøres i online-modus. Hvis det kjøres i online-modus blir data sendt til datamaskinen (via `BeregnOgPlott.py` programmet som skal kjøres på datamaskinen) og dette muliggjør live plotting av data.



Figur 3.1: Kontrollflyten til `EV3main.py`.

Figur 3.2 gir en oversikt over funksjonene i `EV3main.py`. Det er også en liste i starten av hver funksjonsbeskrivelse som viser hvor i filen den ligger.



Figur 3.2: Oversikt over koden i EV3main.py.

Begynnelsen av `EV3main.py` er vist i i kodeutdrag 3.1. Den første linja (`#!/usr/bin/env pybricks-micropython`-linja) er viktig å ha med for alle prosjektene som skrives for EV3en fordi linjen sørger for at programmet blir tolket ("kjørt av") MicroPython, og ikke vanlig Python.

Kode 3.1: Starten til `EV3main.py`.

```
1 #!/usr/bin/env pybricks-micropython
2 # -----------------------------------------------
3 # Prosjekt0X_ProsjektNavn
4 #
5 # Hensikten med programmet er
6 # Følgende sensorer brukes:
7 # Følgende motorer brukes:
8 #
9 # -----------------------------------------------
```

Linje 1 i kodeutdrag 3.1 er altså en spesiell linje, og generelt er en slik type linje brukt av Unix-lignende operativsystem som for eksempel macOS og alle Linux varianter. `#!/usr/bin/env` er stien til programmet som skal kjøre kildekoden. I eksempelet under er det pybricks-micropython som ligger i `#!/usr/bin/env` som skal kjøre koden. De to første karakterene, #!, blir kalt shebang, og mer om dette kan leses om på https://standford.edu.

I kodeutdrag 3.1 kan det også legges til kommentarer som kort beskriver hva prosjektet handler om, og hvilke sensorer og motorer som brukes.

I kodeutdrag 3.2 blir importsetningene i `EV3main.py` vist. På linje 1-3 importeres det moduler fra `pybricks` som muliggjør kommunikasjon til EV3en, portene på EV3en, og sensorer samt motorer som kobles til EV3en. De andre modulene (`struct, socket, json, _thread,` og `time`) vil bli presentert i de delene av kodene hvor de brukes.

Kode 3.2: Importsetninger i `EV3main.py`.

```
11 from pybricks.hubs import EV3Brick
12 from pybricks.parameters import Port
13 from pybricks.ev3devices import *
14 import struct
15 import socket
16 import json
17 import _thread
```

```
18 from time import perf_counter
```

På linje 22 i kodeutdrag 3.3 blir det lagt inn hvorvidt EV3en skal kjøre i online- eller offline-modus.

Kode 3.3: `online` variabelen som bestemmer om `EV3main.py` kjøres i online- eller offlinemodus.

```
22 online = True
```

På linje 26 i kodeutdrag 3.4 blir det laget en variabel, `joyMainSwitch`, som indikerer hvorvidt styrestikken er trykket inn (for å indikere programavslutning). Linje 28 og 29 inneholder variabler for å lagre x og y koordinatene til styrestikken, disse variablene blir overskrevet i `getJoystickValues`-funksjonen (kode 3.26) som blir beskrevet senere.

Kode 3.4: `joyMainSwitch` variabelen som indikerer programavslutning vha. styrestikken og variabler som indikerer koordinater til styrestikken.

```
26 joyMainSwitch = False
27
28 joyForwardInstance = 0
29 joySideInstance = 0
```

I kodeutdrag 3.6 vises definisjonen av hovedfunksjonen, `main`. Denne funksjonen kjører kun hvis `EV3main.py`-fila kjøres direkte, som vil si at hvis fila importeres (for å ta i bruk noen av funksjonene som defineres i fila), så vil ikke `main`-funksjonen kjøre. Dette blir sørget for på linje 473 og 475 og er vist i kodeutdrag 3.5. `main`-funksjonen som presenteres i kodeutdrag 3.6 gjør mange kall til ulike funksjoner som er definert i `EV3main.py`, disse vil bli forklart nærmere senere i kapittelet.

Kode 3.5: Linjer som sørger for at `main`-funksjonen ikke blir kjørt hvis den importeres.

```
464 if __name__ == '__main__':
465     main()
```

3.2.1 Main

```
Main()
Initialize()
GetFirstMeasurement()
GetNewMeasurement()
CalculateAndSetMotorPower()
SendData()
CloseMotorsAndSensors()
IdentifyJoystick()
scale()
getJoystickValues()
```

I kodeutdrag 3.6 blir det laget en `try-except-finally`-blokk. Dette sees på linje 33, 53, og 55. `try` setningen på linje 33 vil prøve å kjøre koden fra linje 34 til 52. "`except Exception as e`" på linje 53 vil kjøre dersom det skjer en feil i `try`-blokken. Den vil da fange feilen (unntaket, Eng: *Exception*) som `e`. Den printer feilen på linje 54. `finally`-blokken på linje 55 kjører uansett hva som skjer i `try`- eller `except`-blokken. På linje 56 gjør `finally`-blokken et kall til `CloseMotorsAndSensors`-funksjonen som sørger for at fila (standard er `measurements.txt`) hvor målingene som lagres blir lukket for å forhindre at filen blir korrupt. `motorA` stoppes på linje 63.

I `try`-blokken på linje 34 vil `main`-funksjonen prøve å initialisere en `robot`-dictionary, og alle sensorer og motorer. Denne `robot`-dictionaryen inneholder all informasjon som er relatert til roboten, inkludert styrestikk og målefila `out_file`. På linje 36 blir de første målingene tatt, samt at nulltida blir satt.

På linje 38 startes det en tråd som kjører `getJoystickValues`-funksjonen som sjekker om styrestikken er trykket inn for å indikere programavslutning og leser koordinatene til styrestikken. Hvis en knapp på styrestikken er trykket inn, vil funksjonen skifte `joyMainSwitch`-variabelen (som ble definert på linje 26, vist i kodeutdrag 3.4) til `True`. Funksjonen `getJoystickValues` blir forklart nærmere i kodeutdrag 3.26.

På linje 44 begynner hovedløkka som kjøres så lenge programmet ikke avsluttes manuelt. Det sjekkes først om styrestikken er trykket inn på linje

45, og hvis den er det, blir programmet avsluttet.

På linje 48 blir det tatt nye målinger. Målingene blir lagret i målefila på EV3en på linje 52 ved hjelp av `SendData`-funksjonen. Hvis det kjøres i online-mode, vil det også bli sendt data til datamaskinen.

Kode 3.6: `main`-funksjonen i `EV3main.py`.

```
32 def main():
33     try:
34         robot, myColorSensor, motorA = Initialize(online)
35         print("Initialized.")
36         zeroTimeInit, time, light = ...
               GetFirstMeasurement(robot, myColorSensor)
37         print("First measurements acquired.")
38 
39         # Read joystick
40         _thread.start_new_thread(getJoystickValues, [robot])
41 
42         print("Ready.")
43 
44         while True:
45             if joyMainSwitch:
46                 break
47 
48             GetNewMeasurement(zeroTimeInit, time, light, ...
                   myColorSensor,
49                               joyForward, joySide)
50             CalculateAndSetMotorPower(motorA, powerA, light,
51                                       joyForward, joySide)
52             SendData(robot, online, time, light)
53     except Exception as e:
54         print(e)
55     finally:
56         CloseMotorsAndSensors(robot, online)
57 
58         # Det er 3 forskjellige måter å stoppe motorene på:
59         # stop() stopper motorpådraget, men bremser ikke.
60         # brake() stopper motorpådraget, og bruker strømmen.
61         # generert av rotasjonen til å bremse.
62         # hold stopper motorpådraget og låser ...
               rotasjonsvinkelen.
63         motorA.stop()
64         # motorB.brake()
65         # motorC.hold()
```

3.2.2 Initialize

```
Main()
Initialize()
GetFirstMeasurement()
GetNewMeasurement()
CalculateAndSetMotorPower()
SendData()
CloseMotorsAndSensors()
IdentifyJoystick()
scale()
getJoystickValues()
```

På linje 73 til 155 i `EV3main.py` blir `Initialize`-funksjonen definert, og første delen av funksjonen er vist i kodeutdrag 3.7. Når starten av en funksjon starter med en flerlinjet streng som i kodeutdrag 3.7, defineres det som calles for en "doctstring". Dette brukes for å dokumentere funksjoner i Python, og vil si noe om hva funksjonen gjør, hva funksjonen tar inn som parametre og hva funksjonen returnerer.

Kode 3.7: docstring i `Initialize`-funksjonen i `EV3main.py`.

```
73 def Initialize(online):
74     """
75     Initialiserer robot-dictionaryen som innheolder ...
           robot-objektet og
76     en socket-forbindelse (hvis online = True) for ...
           kommunikasjon til roboten.
77     Initialiserer alle sensorer og motorer.
78     Initialiserer fila på EV3en som brukes for å lagre ...
           målinger.
79     Det eneste som skal forandres på fra prosjekt til ...
           prosjekt er sensorer,
80     motorer og hva som returneres av funksjonen.
81
82     Parametre:
83     online - bool; bestemmer om det kjøres i online modus ...
           eller ikke.
84     Ved kjøring i online modus blir det satt opp ett ...
           socket-objekt
85     for live kommunikasjon til PC.
86     """
```

I kodeutdrag 3.8 blir det initialisert en dictionary med navn `robot`, og denne dictionaryen inneholder all nødvendig info om EV3en. På linje 91 blir det initialisert et EV3-objekt med navn `ev3`, og dette objektet brukes for å kommunisere med EV3en. Objektet er et hentet fra pybricks-modulene. På linje 92 blir EV3-objektet `ev3` lagret i `robot`-dictionaryen under "brick"-nøkkelen.

Kode 3.8: Initialisering av `robot`-dictionaryen i `Initialize`-funksjonen.

```
88      # robot inneholder all info om roboten
89      robot = {}
90
91      ev3 = EV3Brick()
92      robot["brick"] = ev3
```

På linje 96 i kodeutdrag 3.9 blir det initialisert en dictionary med navn `joystick` som skal inneholde all nødvendig informasjon om styrestikken. På linje 97 brukes `IdentifyJoystick`-funksjonen fra kodeutdrag 3.24 for å sjekke om det Dacota- eller Logitech-styrestikken som er koblet til. Dette blir lagret i `joystick`-dictionaryen under "id"-nøkkelen.

Kode 3.9: Initialisering av joystick i `Initialize`-funksjonen.

```
95      # joystick inneholder all info om joysticken.
96      joystick = {}
97      joystick["id"] = IdentifyJoystick()
```

Oppsettet av styrestikken skjer i kodeutdrag 3.10, mye av informasjonen om dette oppsettet ble hentet fra https://antonmindstorms.com[32]. På denne sida sto det beskrevet hvordan man kan koble en PS4 Dualshock 4 kontroller til EV3en ved hjelp av Bluetooth. Det viste seg at det var mulig å bruke et liknende oppsett for styrestikken ved å koble styrestikken direkte til EV3en (i motsetning til å koble styrestikken til datamaskinen som sender bevegelsene til EV3en).

Siden de forskjellige styrestikkene har forskjellige maksverdier kjøres en if statement på linje 98 - 101 for å finne riktig skaleringsverd. Linje 102 fører denne verdien inn i `joystick`-dictionaryen for senere bruk i kodeutdrag 3.25.

Ved å koble styrestikken direkte til EV3en vil alle bevegelser på styrestikken bli lagret lokalt på EV3en i ei fil som er i byte format. Denne fila, som går under navnet `/dev/input/event2`, blir forsøkt åpnet på linje 105 i ei `try-except`-blokk. Dersom det er koblet til en styrestikk, blir `joystick["in_file"]` satt lik et filobjekt som representerer `/dev/input/event2`-fila. Dette filobjektet kan da brukes for å lese bevegelser fra styrestikken. Hvis det ikke er koblet til en styrestikk, blir `joystick["in_file"]` satt lik `None` for å indikere at ingen styrestikk er koblet til.

På linje 103 blir formatet til fila spesifisert: "llHHI", og dette indikerer følgende format: "long int, long int, unsigned short, unsigned short, unsigned int". For å lese innholdet i fila som styrestikken skriver til må innholdet pakkes ut ved hjelp av `struct` modulen, og dette skjer på linje 104. På linje 104 blir `calcsize`-funksjonen fra `struct`-modulen brukt for å regne ut lengden på "llHHI".

Til slutt, på linje 109, blir "joystick"-nøkkelen til `robot`-dictionaryen satt lik selve `joystick`-dictionaryen. Dette sørger for at det kun er nødvendig å sende `robot`-dictionaryen mellom ulike funksjoner som bruker både `robot`-, og `joystick`-dictionaryen.

Kode 3.10: Initialisering av joystick i `Initialize`-funksjonen.

```
97      joyScale = 0
98      if joystick["id"] == "logitech":
99          joyScale = 1024
100     elif joystick["id"] == "dacota":
101         joyScale = 255
102     joystick["scale"] = joyScale
103     joystick["FORMAT"] = 'llHHI'
104     joystick["EVENT_SIZE"] = ...
            struct.calcsize(joystick["FORMAT"])
105     try:
106         joystick["in_file"] = open("/dev/input/event2", "rb")
107     except OSError:  # hvis ingen joystick er koblet til
108         joystick["in_file"] = None
109     robot["joystick"] = joystick
```

I kodeutdrag 3.11 blir det laget ei fil for alle målingene som sensorene på EV3en gjør. Filnavnet kan endres, og det kan også lagringsplassen ved å spesifisere en sti med mapper. Hvis fila skal bearbeides i ettertid må fila

eksporteres til PCen. Her er "w" parameteren gitt til `open`-funksjonen, slik at for hver kjøring av programmet så vil den gamle fila slettes og erstattes med en ny. I Pythons dokumentasjon finnes andre mulige parametre som kan brukes.

Kode 3.11: Målefila hvor målinger lagres på EV3en.

```
111     # Fila hvor målingene lagres.
112     # OBS: Her er 'w' parameteren gitt, slik at for hver ...
            kjøring så vil
113     # alle gamle målinger slettes og erstattes med nye.
114     # For andre mulige parametre, se:
115     # https://docs.python.org/3/library/functions.html#open
116     robot["outfile"] = open("measurements.txt", "w")
```

Kodeutdrag 3.12 av fila kjøres kun hvis online-variabelen på linje 22 (vist i kodeutdrag 3.3) er satt til `True`. Dette er som nevnt variabelen som indikerer at det skal kjøres med live plotting på datamaskinen. Denne koden setter opp et socketobjekt som muliggjør live kommunikasjon mellom EV3en og PCen. Mer om socketer og socketprogrammering generelt står det om i Vedlegg C.

På linje 120 blir det laget et socket-objekt som blir satt lik `sock` variabelen. Parameteren `socket.AF_INET` spesifiserer hvilken familie med adresser (*address family* - AF) denne socketen kan kommunisere med. I dette tilfellet brukes det IPv4 adresser (INET), og dette betyr at formatet vil være på formen nettverksvert:port (*host:port* på engelsk). Parameteren `socket.SOCK_STREAM` spesifiserer at det er TCP protokollen som skal brukes her.

På linje 121 blir "sock"-nøkkelen i `robot`-dictionaryen satt lik socketobjektet `sock`.

Linje 122 sørger for at socketen umiddelbart godkjenner nye connections fra samme ip-addresse, ellers ville det vært lang ventetid mellom hver gang datamaskinen kan koble seg til EV3en.

På linje 123 blir socketen bundet til port 8070, og på linje 124 begynner socketen å søke etter tilkoblinger. Paremeteren 1 indikerer at det skal tillates maks 1 tilkobling, og etter første tilkobling vil socketen ikke tillate flere

tilkoblinger.

Kode 3.12: Oppsett av socketobjekt på EV3en.

```
117     if online:
118         # Sett opp socketobjektet, og
119         # hør etter for "connection"
120         sock = socket.socket(socket.AF_INET, ...
                socket.SOCK_STREAM)
121         robot["sock"] = sock
122         sock.setsockopt(socket.SOL_SOCKET, ...
                socket.SO_REUSEADDR, 1)
123         sock.bind(("", 8070))
124         sock.listen(1)
```

Når EV3en har satt opp socketobjektet, er roboten klar for å motta en kobling fra datamaskinen (som vil skje i `BeregnOgPlott.py`). På linje 127 og 128 i kodeutdrag 3.13 blir det printet "Waiting for connection from computer" i terminalen, også blir det gitt et pip fra roboten for å indikere at EV3en er klar for å motta en kobling fra datamaskinen.

Når datamaskinen prøver å oppnå en tilkobling til EV3en (som vil skje i `BeregnOgPlott.py`), vil linje 131 i kodeutdrag 3.13 motta tilkoblingen. `accept`-funksjonen som blir brukt på linje 131 returnerer to ting. Det første som returneres er et nytt socketobjekt som er selve koblingen til datamaskinen, denne blir kalt for `connection`. Dette socketobjektet kan brukes for å sende ting frem og tilbake mellom datamaskinen og EV3en. Det andre som returneres er adressen til datamaskinen, og denne brukes ikke, dermed blir adressen satt til `_` variabelen for å indikere at den ikke vil bli brukt.

På linje 132 i kodeutdrag 3.13 brukes `connection`-socketen til å sende en erkjennelse fra EV3en til datamaskinen om at koblingen er mottatt. Legg merke til at det sendes en streng, `"ack"`, men at det legges til en "`b`" foran strengen. Ved å legge en `b` foran en streng i Python, vil objektet bli omgjort til et `bytes`-objekt.

På linje 134 blir `connection`-objektet lagt til `robot`-dictionaryen.

Kode 3.13: Oppsett av socketobjekt på EV3en.

```
125         # Gi et pip fra roboten samt print i terminal
```

```
126         # for å vise at den er klar for socketkobling fra PC
127         print("Waiting for connection from computer.")
128         ev3.speaker.beep()
129
130         # Motta koblingen og send tilbake "acknowledgment" ...
                som byte
131         connection, _ = sock.accept()
132         connection.send(b"ack")
133         print("Acknowlegment sent to computer.")
134         robot["connection"] = connection
```

Etter at socket-koblingen mellom datamaskinen og EV3en er satt opp (gitt at `EV3main.py` ble kjørt i online-modus), blir alle sensorer og motorer initialisert, som vist i kodeutdrag 3.14. For både sensorene og motorene må det spesifiseres hvilken port de er koblet til på EV3en. For ordens skyld er det tatt med oppsettet for alle sensorene og motorene som har vært tilgjengelige. Alle sensorer og motorer som ikke skal brukes for et visst prosjekt kan enten fjernes eller bare kommenteres ut. Legg også merke til at alle motorer bruker `reset_angle`-funksjonen med `0` som parameter for å nullstille motorenes vinkelposisjon.

Kode 3.14: Oppsett av sensorer og motorer på EV3en.

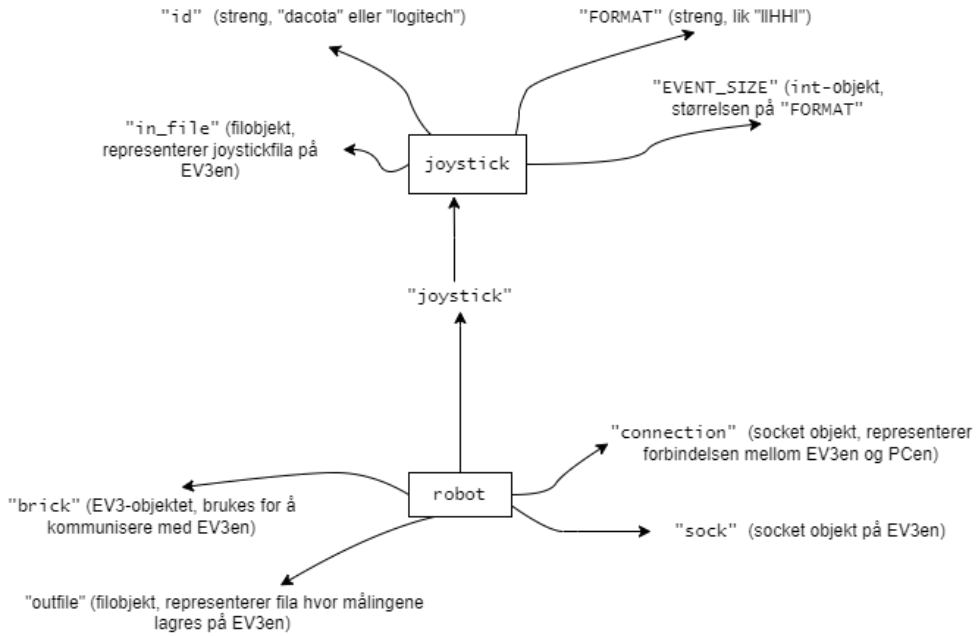
```
136     # Initialiser sensorer.
137     myColorSensor = ColorSensor(Port.S1)
138
139     myTouchSensor = TouchSensor(Port.S2)
140
141     myUltrasonicSensor = UltrasonicSensor(Port.S3)
142
143     myGyroSensor = GyroSensor(Port.S4)
144
145     # Initialiserer motorer.
146     motorA = Motor(Port.A)
147     motorA.reset_angle(0)
148
149     motorB = Motor(Port.B)
150     motorB.reset_angle(0)
151
152     motorC = Motor(Port.C)
153     motorC.reset_angle(0)
154
155     motorD = Motor(Port.D)
156     motorD.reset_angle(0)
```

I kodeutdrag 3.15 vises det hva som returneres av `Initialize`-funksjonen: `robot`-dictionaryen, samt alle sensorer og motorer som brukes.

Kode 3.15: Retur i `Initialize`-funksjonen

```
158     return robot, myColorSensor, motorA
```

I `Initialize`-funksjonen ble det opprettet to dictionaryer, hvor den ene dictionaryen har en referanse til den andre. Denne dictionary-strukturen er vist grafisk i figur 3.3. I denne figuren er det slik at et rektangel referer til en dictionary, mens hermetegn indikerer en nøkkel i en dictionary. `joystick`-dictionaryen har følgende nøkler: "id", "FORMAT", "in_file", og "EVENT_SIZE", mens `robot`-dictionaryen har følgende nøkler: "brick", "outfile", "sock", "connection", og "joystick". "joystick" nøkkelen i `robot`-dictionaryen referer altså til `joystick`-dictionaryen.



Figur 3.3: Dictionary strukturen som returneres av `Initialize`-funksjonen.

3.2.3 GetFirstMeasurement

```
Main()
Initialize()
GetFirstMeasurement()
GetNewMeasurement()
CalculateAndSetMotorPower()
SendData()
CloseMotorsAndSensors()
IdentifyJoystick()
scale()
getJoystickValues()
```

På linje 167 til 193 i `EV3main.py` blir `GetFirstMeasurement`-funksjonen definert, dette er vist (i tillegg til docstrings) i kodeutdrag 3.16. Denne funksjonen initialiserer alle lister for målinger (for eksempel fra lyssensoren), samt alle tidsmålinger.

Kode 3.16: Deklarasjon av `GetFirstMeasurement`-funksjonen og parametre som må inn.

```
167 def GetFirstMeasurements(myColorSensor):
168     """
169     Får inn første måling fra sensorer og motorer.
170 
171     Parametre:
172     robot - robot dictionaryen
173     myColorSensor - Fargesensoren
174 
175     Andre sensorer og motorer må legges til som parametre
176     hvis de skal brukes.
177     """
```

Alle målinger som blir gjort lagres i lister som returneres av funksjonen, og er vist i kodeutdrag 3.17. Funksjonen `perf_counter()` returnerer tid siden et ukjent tidspunkt med høy nøyaktighet, For å kunne lagre tid siden første måling setter vi nåværende returverdi av `perf_counter()` som referansepunkt i linje 180. På linje 181 blir tidslista `time` initialisert, og det blir lagt til 0 som første element. I kodeutdraget er det av sensorer og motorer kun tatt med lyssensoren (linje 184) og motor A (motoren som er koblet til port A på

EV3en, vist på linje 187). På linje 184 vises det også hvordan lyssensorens `reflection`-funksjonen brukes å ta en lysmåling. Utfyllende informasjon om funksjoner til lyssensoren, og de andre sensorene samt motorene, finnes på https://pybricks.github.io/ev3-micropython/ev3devices.html (klikkbar lenke). Styrestikkens nåværende posisjon blir også lagret i egne lister i linje 190 og 191 før nulltid og alle lister returneres av funksjonen.

Kode 3.17: Deklarasjon og retur av lister i `GetFirstMeasurement`.

```
179     # Tider
180     zeroTimeInit = perf_counter()  # nulltida
181     time = [0]  # tida
182
183     # Sensorer
184     light = [myColorSensor.reflection()]
185
186     # Motorer
187     powerA = [0]
188
189     # Joystick
190     joyForward = [joyForwardInstance]
191     joySide = [joySideInstance]
192
193     return zeroTimeInit, time, light, powerA, joyForward, ...
            joySide
```

3.2.4 GetNewMeasurement

```
Main()
Initialize()
GetFirstMeasurement()
GetNewMeasurement()
CalculateAndSetMotorPower()
SendData()
CloseMotorsAndSensors()
IdentifyJoystick()
scale()
getJoystickValues()
```

På linje 227 til 247 i `EV3main.py` blir `GetNewMeasurement`-funksjonen definert. Som parametre tar denne funksjonen inn listene som ble definert i `GetFirstMeasurement`. Dette er vist i kodeutdrag 3.18. Denne funksjonen legger til nye målinger i målelistene, for eksempel blir det tatt inn en ny verdi fra lyssensoren på linje 169 i kodeutdrag 3.18, som blir lagt til `light`-listen.

Kode 3.18: Nye målinger blir tatt og lagret i `GetNewMeasurement`-funksjonen.

```
227 def GetNewMeasurement(zeroTimeInit, joyForward, joySide, ...
        time, light, myColorSensor):
228     """
229     Får inn nye målinger fra sensorer og motorer. Denne ...
            blir kalt i while-løkka
230     i main() funksjonen.
231
232     Parametre:
233     zeroTimeInit - nulltida
234     joyForward, joySide - liste med verdier fra styrestikken
235     time - liste med tider
236     light - liste med målte lysverdier fra lyssensoren
237     myColorSensoren - lyssensoren, for å få inn nye lysverdier
238
239     Andre sensorer og motorer må legges til som parametre
240     hvis de skal brukes.
241     """
242
243     time.append(perf_counter() - zeroTimeInit)
244     light.append(myColorSensor.reflection())
245
246     joyForward.append(joyForwardInstance)
247     joySide.append(joySideInstance)
```

3.2.5 CalculateAndSetMotorPower

```
Main()
Initialize()
GetFirstMeasurement()
GetNewMeasurement()
CalculateAndSetMotorPower()
SendData()
CloseMotorsAndSensors()
IdentifyJoystick()
scale()
getJoystickValues()
```

På linje 300 til 325 i `EV3main.py` blir `CalculateAndSetMotorPower`-funksjonen definert, dette er vist i kodeutdrag 3.19. Denne funksjonen beregner motorpådrag til de ulike motorene. Som parametre blir de relevante motorene gitt, men kan også gis andre verdier som brukes for å beregne motorpådrag, for eksempel styrestikk posisjon eller `light`-lista med verdiene fra lyssensoren. Dersom motorpådraget skal bli brukt videre i prosjektet, til plotting og/eller andre utregninger, kan den også lagres i en liste før den sendes til motoren.

Kode 3.19: Beregning av motorpådrag i `CalculateAndSetMotorPower.py`.

```
300 def CalculateAndSetMotorPower(motorA, powerA, light, ...
        joyForward, joySide):
301     """
302     Beregner og setter pådrag til motorene som brukes.
303
304     Parametre:
305     motorA - Motor A
306     powerA - motorpådragliste for motor A
307     light - lysliste
308     joyForward, joySide - styrestikklister
309
310     Andre motorer og verdier som brukes for å beregne pådrag
311     må legges til som parametre.
312     """
313
314     # Parametre for beregning til motorpådrag
315     a = 1
316     b = 1
```

```
317
318     # Her brukes siste lysmåling for å beregne motorpådrag.
319     lysmaaling = light[-1]
320
321     # pådraget
322     powerA.append(lysmaaling + a * joyForward[-1] - b * ...
            joySide[-1])
323
324     # Sett hastigen på motorene.
325     motorA.run(powerA[-1])
```

3.2.6 SendData

```
Main()
Initialize()
GetFirstMeasurement()
GetNewMeasurement()
CalculateAndSetMotorPower()
SendData()
CloseMotorsAndSensors()
IdentifyJoystick()
scale()
getJoystickValues()
```

På linje 332 til 371 i `EV3main.py` blir `SendData`-funksjonen definert, og definisjonen av funksjonen, samt docstrings, er vist i kodeutdrag 3.20. Funksjonen brukes for å sende data til datamaskinen (hvis det kjøres i online-modus) og å lagre data lokalt på EV3en. I kodeutdrag 3.20 blir `time`, og `light` gitt som parametre.

Kode 3.20: Deklarasjon av `SendData`-funksjonen.

```
332 def SendData(robot, online, time, light):
333     """
334     Sender data fra EV3 til datamaskin, lagrer også målte ...
            og beregnede
335     verdier på EV3en i målefila som ble definert i ...
            Initialize().
336
337     Parametre:
```

```
338     robot - robot dictionaryen
339     online - bool; om det kjøres i online modus eller ikke.
340     Hvis det ikke kjøres i online modus vil ikke roboten
341     prøve å sende data til PCen.
342     time - tidliste
343     light - lysliste
344
345     Andre målte og beregnede verdier som skal
346     lagres på EV3en og skal sendes til
347     PCen må legges til som parametre.
348     """
```

`SendData`-funksjonen lagrer alle de siste målingene i `robot["out_file"]`. Definisjonen av `robot["out_file"]` ble vist i kodeutdrag 3.8. Nøkkelen "out_file" i `robot`-dictionaryen referer altså til `measurements.txt` fila som blir laget lokalt på EV3en. Hvis funksjonen som er definert her kjøres i online-modus, vil den også sende siste måling til datamaskinen via `robot["connection"]`-socketen. Dette muliggjør live plotting.

I kodeutdrag 3.21 vises det hvordan data klargjøres for sending til datamaskin og for lagring i `measurements.txt` fila. På linje 352 blir det definert en dictionary, `data`, som brukes for å sende data til datamaskinen. `dataString` blir definert på linje 353 og er en streng som brukes for å skrive data til `measurements.txt` fila i formatet `målingA,målingB,målingC,..`, hvor hver linje representerer ett sett med målinger. På linje 356 og 357 vises det hvordan `time`-lista og `light`-lista (med målinger fra lyssensoren) blir klargjort for skriving til `measurements.txt` fila (som skjer på linje 364).

Kode 3.21: Lagring av data i `SendData`-funksjonen

```
350     # for json må alle dataene pakkes inn i en dictionary
351     # for hver kjøring tømmes 'data og 'dataString'
352     data = {}
353     dataString = ""
354
355     # for å skrive til measurements.txt
356     dataString += str(time[-1]) + ","
357     dataString += str(light[-1]) + "\n"
358
359     # hver verdi i dictionaryen må referere til en tuple
360     data["time"] = (time[-1])
361     data["light"] = (light[-1])
362
```

```
363     # Skriv dataString til fil
364     robot["outfile"].write(dataString)
```

I kodeutdrag 3.22 vises det hvordan data blir sendt til datamaskinen, så fremt `EV3main.py` kjøres i online modus. på linje 369 blir `json`-modulen brukt for å serialisere dataene (klargjøre dem for overføring). Sendingen skjer på linje 370, her legges det også til et spørsmåltegn på slutten av meldingen slik at datamaskinen kan skille meldinger fra hverandre og vente på mer dersom den mottar bare deler av meldingen.

Kode 3.22: Sending av data til datamaskin i `F6_SendData.py`

```
367     # Send data til PC
368     if online:
369         msg = json.dumps(data)
370         robot["connection"].send(bytes(msg, "utf-b") + b"?")
```

I kodeutdrag 3.21 vises det hvordan `data`-dictonaryen blir laget (linje 352) og fyllt med data (linje 360 og 361), som er dataene som sendes til datamaskinen. Det blir derimot ikke sjekket om det faktisk kjøres i online modus før på linje 368 i kodeutdrag 3.22. Grunnen til at linje 352, 360, og 361 fra kodeutdrag 3.21 ikke ble flyttet til under linje 368 i kodeutdrag 3.22 er for å kunne sette et skille for nye studenter som skal bruke strukturen. Dette skillet sørger for at studenter ikke har noen grunn til å redigere koden fra linje 368 (fra kodeutdrag 3.21) og nedover (helt til slutten på `EV3main.py`).

3.2.7 CloseMororsAndSensors

```
Main()
Initialize()
GetFirstMeasurement()
GetNewMeasurement()
CalculateAndSetMotorPower()
SendData()
CloseMotorsAndSensors()
IdentifyJoystick()
scale()
getJoystickValues()
```

På linje 373 til til 399 i `EV3main.py` blir `CloseMotorsAndSensors`-funksjonen definert. Navnet på funksjonen indikerer at det er motorer og sensorer som lukkes, men det skjer altså ikke. Navnet er en arv fra prosjekstrukturen i MATLAB som ble brukt i ING100; i Python er det ikke nødvendig å lukke portene. I denne funksjonen er det filene (`robot["joystick"]["in_file"]` som inneholder bevegelser fra styrestikken, og `robot["outfile"]`) som blir lukket, samt alle socketforbindelsene.

På linje 385 i kodeutdrag 3.23 lukkes styrestikkfila, og på linje 386 blir "outfile" lukket, som er fila hvor målingene blir lagret. På linje 388 blir det sendt en siste melding til datamaskinen via `connection`-socketen som indikerer at forbindelsen skal brytes ("end"), og på linje 389 blir socketen lukket. Socketen som ble laget for roboten blir lukket på linje 390.

Kode 3.23: Lukking av filer og socketobjekter samt stopping av motorer i `CloseMotorsAndSensors.py`

```
373 def CloseMotorsAndSensors(robot, online):
374     """
375     Lukker filobjektene (målefila hvor målinger lagres på ...
            EV3en,
376     og joystick fila på EV3en), og socket-koblingene ...
            mellom EV3
377     og PC.
378     Denne funksjonen skal ikke forandres på.
379     Parametre:
380     robot - robot dictionaryen
381     online - bool; om det kjøres i online eller ikke. Hvis ...
            det ikke
382     kjøres i online så er det ingen socket-objekt å lukke
383     """
384
385     robot["joystick"]["in_file"].close()
386     robot["outfile"].close()
387     if online:
388         robot["connection"].send(b"end")
389         robot["connection"].close()
390         robot["sock"].close()
```

3.2.8 IdentifyJoystick

```
Main()
Initialize()
GetFirstMeasurement()
GetNewMeasurement()
CalculateAndSetMotorPower()
SendData()
CloseMotorsAndSensors()
IdentifyJoystick()
scale()
getJoystickValues()
```

På linje 403 til 423 i `EV3main.py` blir `IdentifyJoyStick`-funksjonen definert (kodeutdrag 3.24. Denne funksjonen er helt identisk til `identifyJoystick.py`-fila som ble vist i kodeutdrag 2.2, og er lagt opp slik at det ikke skal være nødvendig å redigere på funksjonen.

Kode 3.24: Deklarasjon av `IdentifyJoystick`-funksjonen.

```
403 def IdentifyJoystick():
404     """
405     Identifiserer hvilken styrestikk som er koblet til;
406     enten logitech eller dacota (eventuelt ukjent styrestikk)
407     Denne funksjonen skal ikke forandres på.
408     """
409
410     for i in range(2, 1000):
411         path = ("/dev/bus/usb/001/{:03d}".format(i))
412         try:
413             with open(path, "rb") as f:
414                 joy = f.read()
415                 if joy[2] == 16:
416                     return "logitech"
417                 elif joy[2] == 0:
418                     return "dacota"
419                 else:
420                     return "Ukjent joystick."
421             break
422         except:
423             pass
```

3.2.9 Scale

```
Initialize()
GetFirstMeasurement()
GetNewMeasurement()
CalculateAndSetMotorPower()
SendData()
CloseMotorsAndSensors()
IdentifyJoystick()
scale()
getJoystickValues()
```

På linje 426 til 429 i `EV3main.py` blir `scale`-funksjonen definert. `scale` tar inn en verdi samt mulig min/maks verdi og ny min/maks som verdien skal skaleres til. Eksempelvis gir maks bakover på Logitech-styrestikken verdien 1024, denne skaleres til -100 for å stemme overens med maks revers motorpådrag. Funksjonen `scale` er hentet fra https://antonmindstorms.com[32].

Kode 3.25: Deklarasjon av `scale`-funksjonen.

```
426 def scale(value, src, dst):
427     return ((float(value - src[0])
428              / (src[1] - src[0])) * (dst[1] - dst[0])
429              + dst[0])
```

3.2.10 GetJoystickValues

```
Main()
Initialize()
GetFirstMeasurement()
GetNewMeasurement()
CalculateAndSetMotorPower()
SendData()
CloseMotorsAndSensors()
IdentifyJoystick()
scale()
getJoystickValues()
```

På linje 432 til 466 i `EV3main.py` blir `GetJoystickValues`-funksjonen definert. Funksjonen sjekker om styrestikk-knappen er trykket inn (for å indikerte programavslutning), og leser styrestikkens posisjon. Denne funksjonen blir kjørt i en tråd, og starten på kjøringen av tråden ble vist på linje 40 i kodeutdrag 3.6.

Linje 439 bruker `struct.unpack` for å splitte opp den byte formaterte filen som styrestikken skriver til. `struct.unpack` tar inn format, fil, og hvor mange bits den skal lese, og returnerer verdier lest fra filen med tilsvarende format. Styrestikkens data er formatert som llHHI", vi får dermed 5 returverdier. De to første verdiene er tiden i secunder og mikrosekunder hendelsen fant sted, dette er ikke relevant for vårt bruk og disse kastes derfor ut med _ variabler. `ev_type` sier noe om hvilken type hendelse som har inntruffet, i vårt tilfelle er type 1 et knappetrykk, og type 3 en endring i posisjon av styrestikken. `code` identifiserer hvilken knap som er trykket på, og hvilken akse en endring har påvirket. X-aksen er identifisert som `code` 0 og y-aksen er `code` 1. `value` er tallverien av endringen, den går for logitech-styrestikken fra 0 - 1024 fra venstre til høyre og opp til ned.

For å kunne overskrive globale variabler, defineres i linje 433 variablene `joyMainSwitch`, `joyForwardInstance` og `joySideInstance` som globale. På linje 435 sjekkes det om det er en styrestikk koblet til, og tråden avsluttes dersom det ikke er det. Resten av funksjonen kjører i en evig løkke, som vist på linje 437. På linje 439 leses det fra styrestikken (hvis styrestikken er koblet til). Hvis styrestikken ble trykket inn, som blir sjekket for på linje 445, blir `GetJoystickValues` satt til `True` og funksjonen avslutter.

Siden `getJoystickValues()`-funksjonen kjøres i en egen tråd, legges verdiene for vestre/høyre og for/bak i globale variabler (linje 452 og 457) etter at de er blitt skalert av `scale`-funksjonen. De globale variablene kan overskrives flere ganger eller ikke endres mellom hver gang de leses. Kode 3.18 leser de globale variablene og verdiene blir lagt til i lister for lagring. Det er ikke nødvendig å redigere på `GetJoystickValues`-funksjonen for hvert EV3 prosjekt.

Kode 3.26: Deklarasjon av `GetJoystickValues`-funksjonen.

```
432 def GetJoystickValues(robot):
433     global joyMainSwitch, joyForwardInstance, joySideInstance
434     print("Thread started")
```

```
435     if robot["joystick"]["in_file"] is None:
436         return
437     while True:
438         try:
439             (_, _, ev_type, code, value) = struct.unpack(
440                 robot["joystick"]["FORMAT"],
441                 robot["joystick"]["in_file"].read(
442                     robot["joystick"]["EVENT_SIZE"]))
443         except Exception as e:
444             print("Thread:", e)
445         if ev_type == 1:
446             print("Joystick signal received, stopping ...
                    program.")
447             robot["brick"].speaker.beep()
448             joyMainSwitch = True
449             return
450         elif ev_type == 3:
451             if code == 0:
452                 joySideInstance = scale(
453                     value,
454                     (robot["joystick"]["scale"], 0),
455                     (50, -50))
456             elif code == 1:
457                 joyForwardInstance = scale(
458                     value,
459                     (0, robot["joystick"]["scale"]),
460                     (100, -100))
```

3.3 `BeregnOgPlott.py`

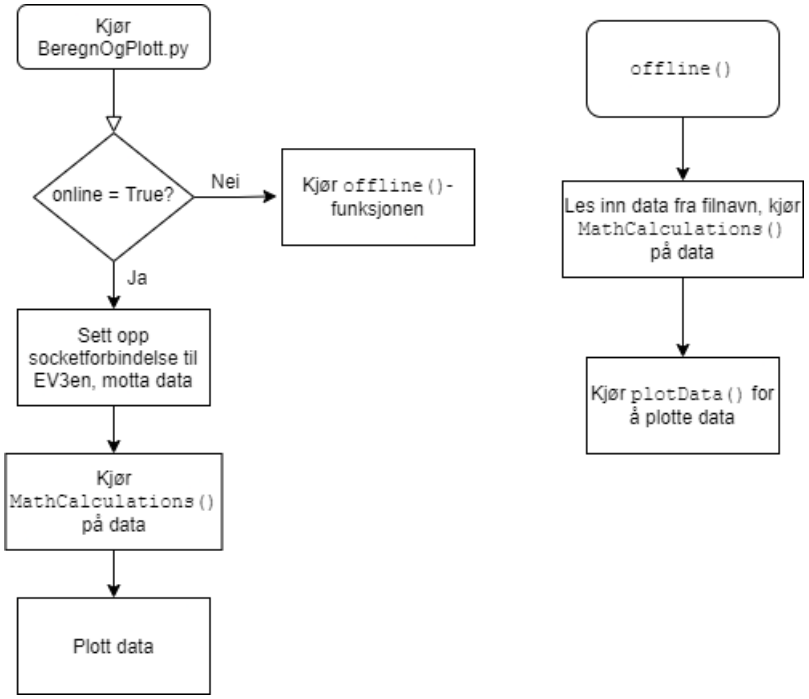
Figur 3.4 gir en oversikt over funksjonene i `BeregnOgPlott.py`. Det er også en liste i starten av hvert delkapittel som viser hvor i filen koden det omhandler ligger.

Figur 3.4: Oversikt over koden i `BeregnOgPlott.py`.

Dette er fila som kjøres på datamaskinen og det er ikke nødvendig å overføre

fila til EV3en. [1] Dersom prosjektet kjører i online-modus (for å muliggjør live plotting), vil denne fila bli kjørt etter at `EV3main.py` blir kjørt på EV3en. På linje 124 i kodeutdrag 3.12 i `EV3main.py` går EV3en i en "vente-status", og venter på en socket-tilkobling fra datamaskinen. Dette blir også indikert ved hjelp av printing til terminal (linje 127 i kodeutdrag 3.13 i `EV3main.py`) og et pip fra roboten (linje 112 i kodeutdrag 3.13 i `EV3main.py`). Etter dette er det klart for å kjøre `BeregnOgPlott.py` på datamaskinen.

Kontrollflyten for programmet er gitt i figur 3.5.



Figur 3.5: Kontrollflyten for `BeregnOgPlott.py`

Selv om figur 3.5 ikke viser det, er det antatt at dersom fila kjører i online modus, så er `EV3main.py` allerede kjørende på EV3en. Videre er denne kon-

[1] Det kan være gunstig å overføre fila (og andre filer) til EV3en, fordi EV3en kan brukes som en slags minnepenn for prosjektfiler. Da kan det være mulig for studenter å for eksempel ha med EV3en hjem, redigere på `EV3main.py` og `BeregnOgPlott.py` på PCen hjemme, for så overføre filene til EV3en igjen. Da kan andre studenter på gruppa, eventuelt studentassistenter, lett få tilgang til filene ved å laste opp filene fra EV3en igjen.

trollflyten forenklet, og hva de spesifikke funksjonene gjør er ikke tatt med. Kontrollflyten er med for å gi et overordnet blikk over fila og hva den gjør.

I kodeutdrag 3.27 vises starten av `BeregnOgPlott.py`, med importsetninger fra linje 1 til 5. `socket`-modulen brukes for å opprette en socket-forbindelse til EV3en, `sys`-modulen brukes for å avslutte programmet hvis en spesifikk feil skjer (mer om dette senere), `json`-modulen brukes for å pakke ut data som sendes fra EV3en. `matplotlib`-modulen som importeres på linje 1 og 2 brukes for å plotte data, og dette er en modul som er laget for å være mest mulig lik plottingen i MATLAB. `matplotlib`-modulen ble valgt fordi dette vil være mest gunstig for nye studenter som skal ta dette faget og skal velge mellom MATLAB og Python. Utfyllende om `matplotlib`-modulen står det om i vedlegg D.1.

Kode 3.27: `BeregnOgPlott.py`

```
1 import matplotlib.pyplot as plt
2 from matplotlib.animation import FuncAnimation
3 import socket
4 import sys
5 import json
```

`tmp` variabelen i linje 8 av kodeutdrag 3.28 brukes til å midlertidig lagre ufulstendig data mottat fra EV3en, dette blir nøyere gjennomgått i kodeutdrag 3.39.

Kode 3.28: Midlertidig lagring for mottak av ufulstendig data

```
7 # Tmp data
8 tmp = ""
```

I kodeutdrag 3.29 blir det definert tre viktige konstanter; `online` bestemmer om det kjøres i online modus eller ikke, `EV3_IP` er EV3ens ip adresse og er viktig for å muliggjør kommunikasjon til EV3en, mens `filename` er navnet på fila hvor dataene er lagret. `EV3_IP` brukes kun hvis det kjøres i online modus, mens `filename` brukes kun i offline modus. Hvordan en finner EV3ens ip adresse, ble vist og forklart i seksjon 2.4. Vær obs på at for hver gang det opprettes en tilkobling til datamaskinen kan ip adressen forandre seg, og da må `EV3_IP` oppdateres.

Kode 3.29: Noen viktige konstanter i BeregnOgPlott.py

```
10 # Set online flag, ip address and filename.
11 online = True
12 EV3_IP = "192.168.2.2"
13 filename = "measurements.txt"
```

I kodeutdrag 3.30 blir det laget lister for alle dataverdiene. Dataverdiene inkluderer data fra `measurements.txt` (hvis det kjøres i offline modus), eller fra EV3en (dersom det kjøres i online modus og data sendes fra EV3en), og andre beregnede verdier.

Kode 3.30: Lister for å holde på dataverdier i BeregnOgPlott.py

```
15 # Initialize lists.
16 light = []
17 time = []
18 ts = [0]
19 flow = []
20 volume = [0]
```

3.3.1 Plotting

Plotting
MathCalculations()
appendLists()
offline()
live()
Online oppsett

I kodeutdrag 3.31 lages det flere plott ved hjelp av `subplots`-funksjonen til `pyplot`. Her skal det være 3 rader (en for hver verdi som plottes: `light`, `lightDeviation`, og `lightIntegrated`) og 1 kolonne. Subplottene ligger i `ax` variabelen og kan nåes ved indeksering. `sharex=True` gjør at alle plottene har lik x-akse, dette gjør det lettere å se sammenhengen mellom dem.

Kode 3.31: Subplott og figuren i BeregnOgPlott.py

```
23 # Define figure and subplots.
24 fig, ax = plt.subplots(nrows=3, ncols=1, sharex=True)
```

Når plotting skjer samtidig som kjøring av program på EV3 er det hensiktsmessig å redusere mengden kode som kjøres i en uendelig løkke. Av denne grunn er funksjonen `initPlot` definert. Funksjonen ligger på linje 27 til 32 i `BeregnOgPlott.py` og setter opp titler og akse-merkelapper for de forskjellige subplottene.

Kode 3.32: `initPlot`-funksjon i `BeregnOgPlott.py`

```
27 def initPlot():
28     global ax
29     ax[0].set_title('Light')
30     ax[1].set_title('Flow')
31     ax[2].set_title('Volume')
32     ax[2].set_xlabel('Time [sec]')
```

På linje 61 i kodeutdrag 3.33 defineres `plotData`-funksjonen. Det er denne funksjonen som sørger for plotting. På linje 28 vises det at `ax[0]` blir plottet med `time` langs første akse, og `light` langs andre akse. Tilsvarende for de to andre subplottene. For å plotte fortløpende plottes det egentlig en ny linje hver gang plottet skal oppdateres, og i motsetning til plotting i MATLAB som matplotlib etterligner er det i matplotlib slik at gamle linjer ikke forsvinner når du tegner nye. Siden matplotlib automatisk tilegner nye linjer sin egen farge for å skille dem fra hverandre vil dette føre til at fargen på plottet tilsynelatende endrer seg så raskt som plottet oppdateres. Det er to måter å unngå dette: Enten kan man kalle `ax.cla()` på alle akser som plottes for å fjerne tidligere linjer, eller spesifisere fargen til plottet ved å sende `'b'` inn i plot funksjonen. Grunnen til at vi har valgt å bruke den andre metoden er at da vi først prøvde med den første metoden, viste det seg å ta så lang tid å kjøre `ax.cla()` flere ganger at plottingen ikke lenger var i sanntid.

Kode 3.33: `plotData`-funksjonen som står for plotting av data

```
34 def plotData():
35     ax[0].plot(time[0:-1], light[0:-1], 'b')
36
37     ax[1].plot(time[0:-1], flow[0:-1], 'b')
38
```

```
39      ax[2].plot(time[0:-1], volume[0:-1], 'b')
```

For å tillate utregninger som skal skje etter at plotting er fullført og/eller skrive verdier til terminalen - defineres `stopPlot`-funksjonen på linje 43 til 48. Eksempelvis skal i vedlegg I de siste kalkulerte verdiene legges inn i en tabell, da kan det være nyttig å skrive disse verdiene til terminalen. Linje 45 stopper kontinuerlig plotting dersom det er aktivt.

Kode 3.34: `stopPlot`-funksjonen som låser plot og regner ut eventuelle sluttverdier

```
43  def stopPlot():
44      try:
45          livePlot.event_source.stop()
46      except:
47          pass
48      # Alt som skal regnes ut når programmet stoppes legges her
```

3.3.2 MathCalculations

```
Plotting
MathCalculations()
appendLists()
offline()
live()
Online oppsett
```

På linje 20 i `BeregnOgPlott.py` blir `MathCalculations`-funksjonen definert. I kodeutdrag 3.35 blir eksempelet med "integrasjon" vist, hvor det beregnes `flow` og `volume` som beregnede verdier basert på `light`. I kodeutdraget er eksempelet for integrasjon gitt, og siden logikken bak beregningene ikke er viktig her, blir de ikke forklart nærmere. `MathCalculations`-funksjonen blir kalt en gang for hvært sett med data motatt.

Kode 3.35: `MatchCalculations`-funksjonen i `BeregnOgPlott.py`

```
52  def MathCalculations():
53      """
```

```
54      Legger til beregnede verdier basert på målte verdier.
55      Legger også til tidsskrittet 'ts'.
56
57      Parametre:
58      flow - flow
59      ts - tidsskritt
60      volume - volum
61      light - lys
62      time - tid
63
64      Andre målte og beregnede verdier må legges til som ...
            parametre
65      hvis de skal bruke
66      """
67
68      # Flow
69      flow.append(light[-1] - light[0])
70      if len(time) > 1:
71          # ts
72          ts.append(time[-1] - time[-2])
73          # Volume - "light integrated"
74          volume.append(volume[-1] + flow[-2] * ts[-1])
```

3.3.3 appendLists

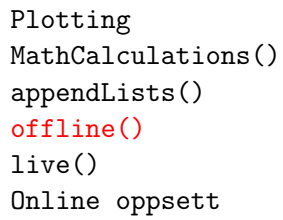
```
Plotting
MathCalculations()
appendLists()
offline()
live()
Online oppsett
```

For å gjøre det enklest mulig å endre hvilke data som blir tatt inn og lagret, er dette trukket ut til en egen funksjon uavhengig av om plotting er tilkoblet EV3 eller ikke. Når data kommer fra EV3 direkte er de pakket inn i en dictionary, `appendLists`-funksjonen oppfører seg derfor litt anderledes enn når data kommer i lister fra en fil. Dette er den siste funksjonen det er nødvendig å endre på for å fullføre en gitt oppgave.

Kode 3.36: `appendLists`-funksjonen i `BeregnOgPlott.py`

```
76 def appendLists(datum):
77     if isinstance(datum, dict):
78         light.append(datum["light"])
79         time.append(datum["time"])
80     else:
81         time.append(float(datum[0]))
82         light.append(int(datum[1]))
```

3.3.4 Offline



Dersom fila kjører i offline-modus, blir `offline`-funksjonen presentert i kodeutdrag 3.37 brukt. Forskjellen mellom `offline`-funksjonen og `live`-funksjonen er at `live`-funksjonen mottar data fra EV3en, mens `offline`-funksjonen bearbeider data som allerede ligger på datamaskinen. På linje 90 blir filen med navn gitt av `filename`-variabelen åpnet, dette er fila med alle målingene. Linje 91 deler filen opp i linjer før linje 92 videre deler linjene opp rundt ",", og legger de resulterende tekststrengene i listen `datum`. `appendLists`-funksjonen blir kalt på linje 93, og så kalles `MathCalculations` på linje 94. Linje 92 til 94 gjentas en gang for hver linje i filen som ble lastet inn.

Etter at all data er lest inn og prosessert kjøres `initPlot`, `plotData`, og `stopPlot` før linje 105 viser plottet. På linje 103 er det lagt til ei ekstra linje spesifikt for Mac datamaskiner for å unngå en Mac-spesifikk feil hvor plott-vinduet lukker seg automatisk. Fiksen for denne feilen ble funnet på stackexchange (klikkbar lenke).

Kode 3.37: `offline`-funksjonen i `BeregnOgPlott.py`

```
88 def offline(filename):
```

```
89      # Read from file row by row and append data to lists.
90      with open(filename) as f:
91          for row in f:
92              datum = row.split(",")
93              appendLists(datum)
94              MathCalculations()
95
96      # Plot data in lists.
97      initPlot()
98      plotData()
99
100     stopPlot()
101     # Set plot layout and show plot.
102
103     fig.set_tight_layout(True)# mac
104
105     plt.show()
```

3.3.5 Live

```
Plotting
MathCalculations()
appendLists()
offline()
live()
Online oppsett
```

`live`-funksjonen kjøres når `BeregenOgPlott.py` kjøres i online modus, og står for mottakelsen av dataene fra EV3en. Først henter den globale variabelen `tmp` definert i linje 8. Neste steg er å prøve å motta data fra EV3en ved hjelp av en `try-except`-blokk, som vist på linje 111 i kodeutdrag 3.38. Dersom det skjer en feil, vil linje 113 til 115 kjøre, og dette stopper plottinga, og returnerer fra funksjonen. Legg merke til at funksjonen antar at det allerede finnes en socket-forbindelse mellom datamaskinen og EV3en. Denne socket-forbindelsen blir opprettet senere i `BeregnOgPlott.py`, og funksjonen `live()` blir kalt etter dette.

Kode 3.38: `live`-funksjonen som mottar data fra EV3en

```
107 def live():
```

```
108 global tmp
109     # Recieve data from EV3.
110     try:
111         data = sock.recv(1024)
112     except:
113         print("Lost connection to EV3")
114         stopPlot()
115         return
```

Data som sendes over en socket vil sjeldent passe perfekt innenfor "bufferen" til socketen, dette kan føre til at når man mottar data får man mer enn en datapakke. for å kunne vite når en datapakke er ferdig, ble det i kodeutdrag 3.22 lagt inn et spørsmålstegn på slutten av hver data melding. Kodeutdrag 3.39 splitter datastrømmen (linje 118) rundt spørsmålstegn og gjør klart for å behandle vært sett med data. Linje 120 sjekker om vi har uferdig data fra tidligere og linje 121 kombinerer den midlertidige dataen med det første mottatte settet for å fullføre det før linje 122 setter `tmp` blank igjen.

Kode 3.39: Uttdrag fra `live`-funksjonen som splitter datastrøm i biter

```
117     try:
118         data = data.split(b"?")
119         # Reconstruct split data
120         if tmp != "":
121             data[0] = tmp + data[0]
122             tmp = ""
```

Videre kode kjøres en gang for hvert mottatte datasett, først sjekkes det om dataen er en tom streng, dersom den er det går koden videre til neste datasett. På linje 127 i kodeutdrag 3.40 blir det sjekket om det ble mottatt en **''end''** melding fra EV3en i byte-format. Hvis dette skjer, vil plottinga stoppe opp, og vi returnerer fra funksjonen.

Kode 3.40: Sjekk om end signalet er mottatt

```
123         for datum in data:
124             if datum == b'':
125                 continue
126             # If the data recieved is the end signal, ...
                    freeze plot.
127             elif datum == b"end":
128                 print("Recieved end signal")
```

```
129                 stopPlot()
130                 return
```

Dersom dataene som ble sendt fra EV3en ikke var en ''end'' melding i byte-format, prøves det å deserialisere dataene ved hjelp av `json`-modulen. Dette er vist på linje 135 i kodeutdrag 3.41. Dersom det skjer en feil, blir dataene lagt til i `tmp` variabelen og koden går videre til neste datasett (linje 138 til 139). Datasettet legges så til i listene definert i 3.30 og så kjøres utregninger.

Kode 3.41: Legg mottatt data til listene og kjør utregninger

```
133             # Append the recieved data to the lists.
134             try:
135                 datum = json.loads(datum)
136             except:
137                 # Save incomplete data
138                 tmp = datum
139                 continue
140             appendLists(datum)
141             MathCalculations()
```

Dersom en feil har oppstått under behandlingen av dataene skrives feilen til konsollen og plottet stoppes (linje 141 til 144). Helt på slutten av funksjonen blir det gjort et kall til `plotData`-funksjonen for å plotte de nye dataene.

Kode 3.42: Stopp eller fortsett plotting

```
140     except Exception as e:
141         print(e)
142         print("Data error")
143         stopPlot()
144         return
145 
146     # Plot the data in the lists.
147     plotData()
```

3.3.6 Online setup

```
Plotting
MathCalculations()
appendLists()
offline()
live()
Online oppsett
```

I resten av `BeregnOgPlott.py` fila blir det gjort kall til alle funksjonene som ble definert tidligere i fila. Først sjekkes det om det kjøres i online-, eller offline-modus, som vist på linje 150 i kodeutdrag 3.43. Dersom det kjøres i online-modus, blir det satt opp en socketforbindelse til EV3en. På linje 154 og 156 vises det at `EV3_IP` blir brukt, samt port nummer 8070, som er standard portnummer for applikasjonen (`EV3main.py`) på EV3en. Dersom det mottas en `''ack''`-melding i byte format er alt ting ok, og så lenge det ikke skjer en `socket.timeout` (linje 147), vil programmet fortsette med å motta data og plotte de. Dersom det ikke mottas en `''ack''`-melding i byte format er dette en indikasjon på at EV3en ikke kommuniserer med datamaskinen, og programmet avsluttes ved et kall til `exit`-funksjonen fra `sys`-modulen.

Kode 3.43: Kjøring av `BeregnOgPlott.py` i online modus

```
150 if online:
151     # If online, setup socket object and connect to EV3.
152     sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
153     try:
154         addr = (EV3_IP, 8070)
155         print("Attempting to connect to {}".format(addr))
156         sock.connect(addr)
157         data = sock.recv(1024)
158         if data == b"ack":
159             print("Connection established")
160         else:
161             print("no ack")
162             sys.exit()
163     except socket.timeout:
164         print("failed")
165         sys.exit()
```

Dersom socket-forbindelsen til EV3en ble opprettet uten feil, blir det star-

ted med plotting ved hjelp av `FuncAnimation`-funksjonen til matplotlib. Denne funksjonen tar figur (plot vindu) `live`, `init_func=initPlot()` og `interval=10` som en parameter. Dette er vist på linje 168 i kodeutdrag 3.44. Nå kjøres først `initPlot`-funksjonen, så kjøres `plot`-funksjonen hvert 10 millisekund som definert av `interval=10`. til slutt kjøres den samme mac-fiksen som i kodeutdrag 3.37 og plottet vises.

Kode 3.44: Kall til `FuncAnimation`-funksjonen fra `matplotlib`-modulen

```
167     # Start live plotting.
168     livePlot = FuncAnimation(fig, live, ...
            init_func=initPlot(), interval=10)
169
170     # Set plot layout and show plot.
171     fig.set_tight_layout(True)
172     plt.show()
```

I kodeutdrag 3.45 vises det hva som skjer dersom `BeregnOgPlott.py` kjøres i offline modus; det blir gjort ett kall til `offline`-funksjonen som tar seg av all utpakkingen av dataene samt plottinga.

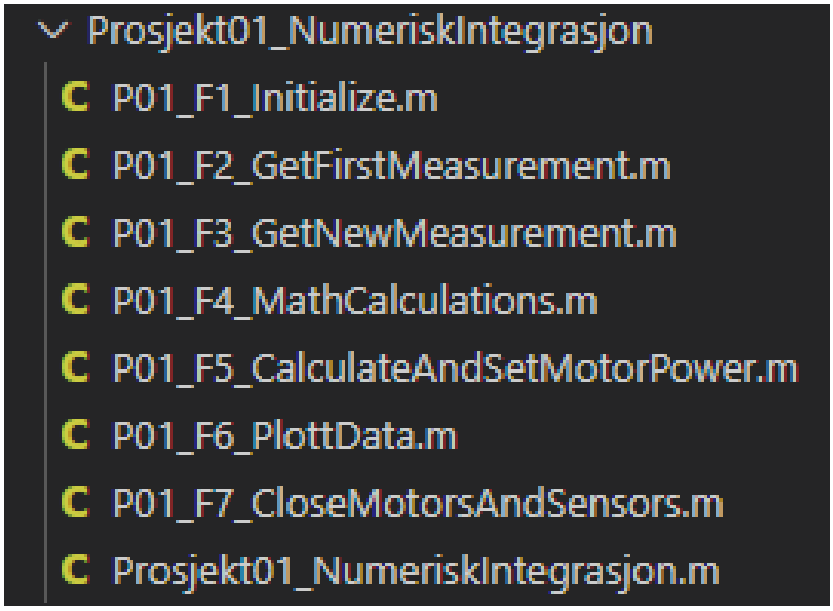
Kode 3.45: offline kjøring av `BeregnOgPlott.py`

```
173 else:
174     # If offline, plot from file defined by filename.
175     offline(filename)
```

Kapittel 4

Basisstrukturen for et EV3-prosjekt ved bruk av dictionary

En annen prosjektstruktur som ble sett på, var en struktur som brukte dictionary hyppigere enn forrige struktur, samt at funksjonene ble lagt i egne filer. Denne strukturen minnet mer om prosjektstrukturen til MATLAB fra ING-100 faget (som er presentert i 4.1), og ble designet for å etterligne denne MATLAB-strukturen. Denne alternative strukturen splitter altså opp funksjonene slik at de viktigste får sin egen fil, samt at alle listene for data blir lagt inn i dictionaries.



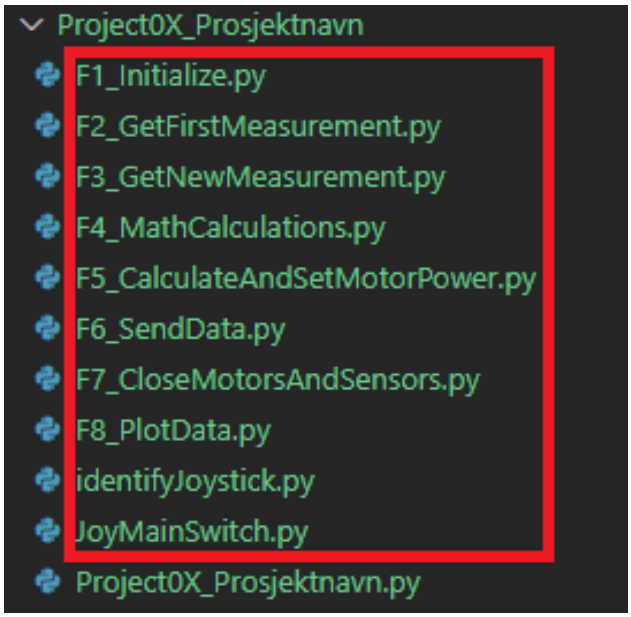
Figur 4.1: Filene som inngikk i basisstrukturen for et EV3-prosjekt på MATLAB fra ING100-faget. Hovedfila `Prosjekt01_NumeriskIntegrasjon.m` gjør kall til de andre filene som inneholder en spesifikk funksjon hver.

Dictionary strukturen har naturligvis mye til felles med forrige struktur, for eksempel er oppsettet av socketobjektene for kommunikasjon mellom EV3 og datamaskin identiske. For ordens skyld er all koden til den nye strukturen tatt med, og der det er identisk kode med den gamle strukturen vil det hele tiden refereres til kodeutdrag fra den gamle strukturen.

Selv om det er mye til felles mellom de to strukturene har dictionary strukturen noen mangler i forhold til liste strukturen. Liste strukturen ble videreutviklet i løpet av oppgaven, mens dictionary strukturen enda er i sin første iterasjon.

Et overordnet blikk av strukturen er vist i figur 4.2. Alle filene innenfor det røde rektangelet i figuren er moduler som blir kalt av hovedfila "`Prosjekt0X_Prosjektnavn.py`". Hovedfilas navn endres til det gjeldende prosjektets navn. For eksempel blir hovedfila i figur 4.2 kalt `Prosjekt01_NumeriskIntegrasjon.py` for integrasjonsprosjektet (slik som i MATLAB, se figur 4.1). Hovedfila er fila som kjøres når roboten skal kjøre

et program, og inne i denne fila er det importsetninger som henter inn alle de andre filene for å lage et fullverdig program. De andre filene, de som er innenfor det røde rektangelet i figur 4.2, skal ha det samme navnet for alle prosjektene.

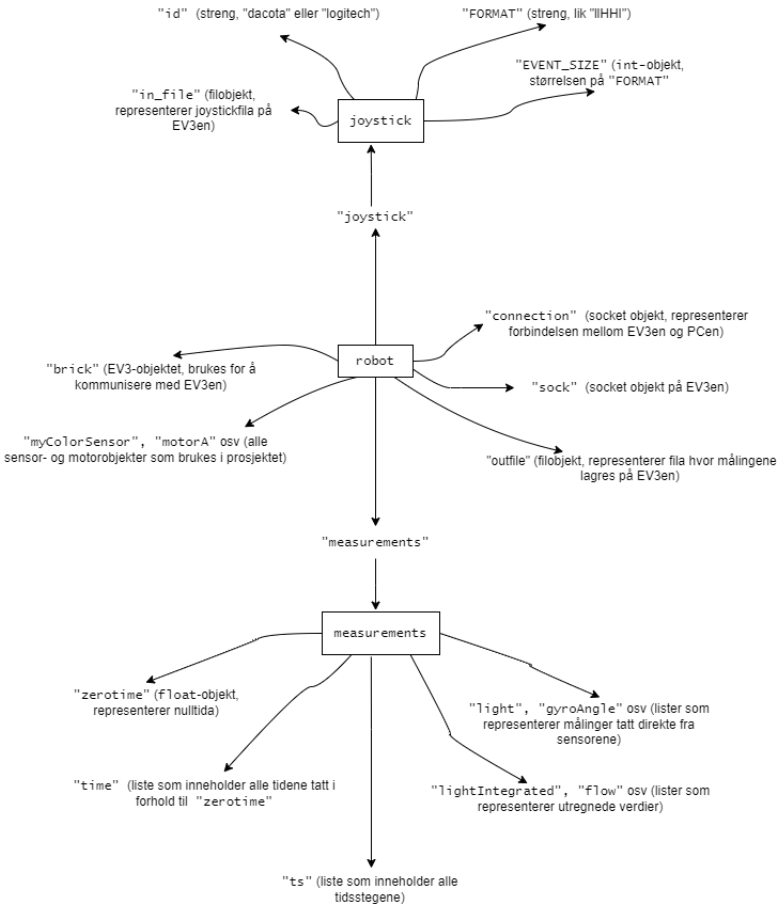


Figur 4.2: Filene som inngår i basisstrukturen for et EV3-prosjekt.

4.1 Dictionary-strukturen for dette rammeverket

Siden dette rammeverket bruker dictionaryer til et ganske så stor grad, presenterer vi her en figur over dictionary-strukturen. I figur 4.3 blir det de 3 dictionaryene som brukes i rammeverket vist. Et rektangel representerer en dictionary, mens pilene ut fra et rektangel representerer nøkler i dictionaryen, for eksempel er `robot` en dictionary, med flere nøkler (`''joystick''`, `''measurements''`, `''brick''`, `''myColorSensor''`, `''connection''`, `''sock''`, og `''outfile''`. Nøkkelen `''joystick''` i `robot`-dictionaryen referer til `joystick`-dictionaryen og `''measurements''`-nøkkelen i `robot`-dictionaryen referer til `measurements`-dictionaryen. `joystick`-dictionaryen og `measurements`-dictionaryen har igjen nøkler som referer til andre objekter.

Forskjellen fra forrige struktur er `measurements`-dictionaryen, som inneholder info om alle målingene som blir tatt fra sensorene til EV3en. I den forrige strukturen var disse gitt i egne lister, mens i denne strukturen er listene samlet inn og lagt til i en egen dictionary.

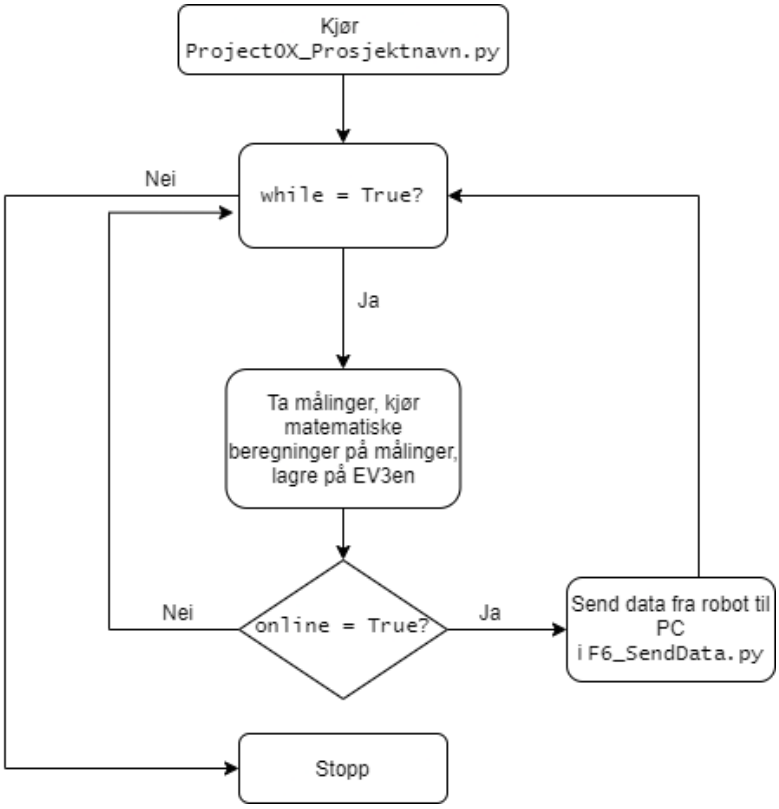


Figur 4.3: Dictionary-strukturen som blir brukt for et basisprosjekt. For hvert prosjekt vil noen av nøklene forandre seg (spesielt de i `measurements`-dictionaryen). Noen vil være konstante (alle i `joystick`-dictionaryen).

4.2 Project0X_Prosjektnavn.py

Dette underavsnittet viser de viktigste delene av `Project0X_Prosjektnavn.py`, og vil blant annet vise alle funksjonskallene uten å spesifikt si hva de funksjonene gjør (funksjonsbeskrivelsene vil bli gitt i hvert sitt underavsnitt). Et flytskjema for `Project0X_Prosjektnavn.py` er gitt i figur 4.4. Dette flytskjemaet går ikke dypt inn i detaljene (den tar ikke med alle de ulike funksjonene som brukes), men er med for å gi en overordnet oversikt.

I fila `Project0X_Prosjektnavn.py` er det ei evig while-løkke som kjøres helt til programmet avsluttes (manuelt eller på grunn av feil). Denne while-løkka vil ta målinger, og uavhengig av om fila kjøres i `online = True` (som betyr at det kjøres med live-plotting og data skal sendes til datamaskinen) eller `online = False`, så vil EV3en lagre målingene på EV3en. Dersom det kjøres med `online = True`. For å bryte ut av hovedløkka, `while True`, så kan man enten sende stopp-signal ved hjelp av styrestikken eller ved å trykke på tilbake-knappen på EV3en.



Figur 4.4: Flytskjema som illustrerer logisk flyt i `Project0X_Prosjektnavn.py`. Så lenge `while = True` vil programmet fortsette å kjøre.

Kontrollflyten for dictionary strukturen i figur 4.4 har tilnærmet identisk flyt med kontrollflyten for `EV3main.py` fra liste strukturen (vist i figur 3.1). Hovedforskjellen mellom de to prosjektstrukturene er at liste strukturen ikke kjører matematiske beregninger på EV3en; alt av beregninger skjer på datamaskinen (enten i online modus, mens EV3en sender data, eller i offline modus i ettertid).

Begynnelsen av fila er vist i kodeutdrag 4.1 og er helt identisk til starten til `EV3main.py` som ble presentert i kodeutdrag 3.1.

Kode 4.1: Starten til `Project0X_Prosjektnavn.py`

```
1 #!/usr/bin/env pybricks-micropython
```

```
2 # -----------------------------------------------
3 # Prosjekt0X_ProsjektNavn
4 #
5 # Hensikten med programmet er:
6 # Følgende sensorer brukes:
7 # Følgende motorer brukes:
8 #
9 # -----------------------------------------------
```

I kodeutdrag 4.2 kommer alle importsetningene. For eksempel, i linje 12 sier setningen `from F1_Initialize import F1_Initialize` følgende: fra filen `F1_Initialize.py` (`from F1_Initialize`) hent funksjonen `F1_Initialize` (`import F1_Initialize`). Vær obs på at i linje 16 så vises det tre prikker (...). Dette er lagt til automatisk av Latex for å indikerte at linja fortsetter (siden linja er så lang).

Kode 4.2: Importsetninger i `Project0X_Prosjektnavn.py`

```
11 import struct
12 from F1_Initialize import F1_Initialize
13 from F2_GetFirstMeasurement import F2_GetFirstMeasurement
14 from F3_GetNewMeasurement import F3_GetNewMeasurement
15 from F4_MathCalculations import F4_MathCalculations
16 from F5_CalculateAndSetMotorPower import ...
       F5_CalculateAndSetMotorPower
17 from F6_SendData import F6_SendData
18 from F7_CloseMotorsAndSensors import F7_CloseMotorsAndSensors
19 from JoyMainSwitch import JoyMainSwitch
20 from time import sleep
21 import _thread
```

I kodeutdrag 4.3 blir `online`-variabelen laget, som settes lik `True` dersom det kjøres i online-modus (for å muliggjøre live plotting). roboten blir "initialisert" på linje 25 ved hjelp av `F1_Initialize`-funksjonen fra `F1_Initialize.py`. Funksjonen returnerer en dictionary som inneholder masse informasjon relatert til EV3en (blant annet `measurements.txt` som er fila hvor målingene tatt av EV3en blir lagret, og EV3-objektet som muliggjør kommunikasjon til EV3en).

Første måling fra sensorene til EV3en blir tatt ved hjelp av `P0X_F2_GetFirstMeasurement`-funksjonen (som ble importert i linje 13 i kodeutdrag 4.2), og er vist på linje 27 i kodeutdrag 4.3. Deretter blir

joyMainSwitch-variabelen satt opp for å indikere om styrestikken er trykket inn for å signalisere programavslutning, og det startes det en tråd som kjører JoyMainSwitch-funksjonen. JoyMainSwitch-funksjonen er en tidligere versjon av getJoystickValues i EV3main.py fra liste strukturen, presentert i kodeutdrag 3.26. JoyMainSwitch-funksjonen fungerer på samme måte som getJoystickValues, men leser ikke av styrestikken posisjon.

Kode 4.3: Klargjøring før while-løkke i Project0X_Prosjektnavn.py

```
22 # Er du koblet til EV3en eller ikke?
23 online = True
24
25 robot = F1_Initialize(online)
26 print("Initialized")
27 measurements = F2_GetFirstMeasurement(robot)
28 print("First measurement acquired")
29
30 # variabel som settes lik True når styrestikken trykkes
31 # inn, og programmet vil da avslutte
32 joyMainSwitch = False
33
34 # Read joystick
35 _thread.start_new_thread(JoyMainSwitch, [robot])
36
37 print("Ready")
```

While-løkka er presentert i kodeutdrag 4.4 og starter på linje 39. Denne løkka vil kjøre så lenge programmet ikke stoppes av brukeren. Det første løkken gjør er å sjekke om joyMainSwitch er satt til True (av JoyMainSwitch-funksjonen), og vil da bryte ut av løkka og avslutte programmet.

På linje 43 til 47 blir de importerte funksjonene (fra kodeutdrag 4.1) brukt. Funksjonene står blant annet for innhenting av nye data fra sensorene, kalukasjon av beregnede verdier basert på dataene fra sensorene, kalkulasjon av motorpådrag, og sending av data til datamaskinen (gitt at det kjøres i online modus). Etter at while-løkka er ferdig med å kjøre, blir funksjonen F7_CloseMotorsAndSensors() kalt helt på slutten av fila.

Kode 4.4: while-løkka i Project0X_Prosjektnavn.py

```
39 while True:
```

```
40      if joyMainSwitch:
41          break
42
43      F3_GetNewMeasurement(robot, measurements)
44      F4_MathCalculations(measurements)
45      F5_CalculateAndSetMotorPower(robot, measurements)
46      # kan flyttes til etter while-løkka
47      F6_SendData(robot, measurements, online)
48
49  F7_CloseMotorsAndSensors(robot)
```

4.3 F1_Initialize.py

F1_Initialize.py
F2_GetFirstMeasurement.py
F3_GetNewMeasurement.py
F4_MathCalculations.py
F5_CalculateAndSetMotorPower.py
F6_SendData.py
F7_CloseMotorsAndSensors.py

I denne fila blir det initialisert objekter som gjør det mulig å jobbe med roboten (inkludert sensorer og motorer). Fila ligner på `Initialize`-funksjonen fra det liste strukturen, presentert i kodeutdrag 3.7.

Importsetningen på linje 1 i kodeutdrag 4.5 her er nødvendig for å kunne jobbe med et EV3-objekt (for kommunikasjon til EV3-roboten). Portklassen importeres på linje 2, og dette gjør det mulig å referere til de ulike portene til EV3en. På linje 3 blir alle de ulike sensor-klassene importert for å kunne kommunisere med de ulike sensorene som kobles til EV3en, og på linje 4 blir `identifyJoystick`-funksjonen fra `identifyJoystick.py` fila importert. `identifyJoystick`-funksjonen er identisk samme funksjon i liste strukturen.

Kode 4.5: Importsetninger i F1_Initialize.py

```
1  from pybricks.hubs import EV3Brick
2  from pybricks.parameters import Port
```

```
3  from pybricks.ev3devices import *
4  from identifyJoystick import identifyJoystick
5  import struct
6  import socket
```

Resten av filen er identisk med `Initialize`-funksjonen i `EV3main()`, med noen få unntak. Initialisering av styrestikken skjer i kodeutdrag 4.6 (tilnærmet lik kodeutdrag 3.9 og 3.10). Siden styrestikk bevegelse aldri ble utviklet under dictionary strukturen, mangler `scale` nøkkelen i `joystick`-dictionaryen og tilhørende kode.

Kode 4.6: Klargjøring av styrestikk i `F1_Initialize.py`

```
24      # joystick er en dictionary som inneholder all ...
            nødvendig info om joysticken.
25      joystick = {}
26      joystick["id"] = identifyJoystick()
27      joystick["FORMAT"] = 'llHHI'
28      joystick["EVENT_SIZE"] = ...
            struct.calcsize(joystick["FORMAT"])
29      try:
30          joystick["in_file"] = open("/dev/input/event2", "rb")
31      except OSError: # hvis ingen joystick er koblet til
32          joystick["in_file"] = None
33      robot["joystick"] = joystick
```

Oppsett av socket og målefil er lik `Initialize`-funksjonen. I kodeutdrag 4.7 og 4.8 initialiseres sensorer og motorer, forskjellen her er at både sensorer og motorer legges inn i `robot`-dictionaryen (linje 47 og 51).

Kode 4.7: Initialisering av sensorer i `F1_Initialize.py`

```
46      myColorSensor = ColorSensor(Port.S1)
47      robot["myColorSensor"] = myColorSensor
```

Kode 4.8: Initialisering av motorer i `F1_Initialize.py`

```
64
65      motorA = Motor(Port.A)
66      motorA.reset_angle(0)
67      robot["motorA"] = motorA
```

Helt på slutten av fila blir i kodeutdrag 4.9 `robot`-dictionaryen returnert av funksjonen. Med sensorer og motorer lagt inn i `robot`-dictionaryen er det bare denne dictionaryen som returneres.

Kode 4.9: Retur av `P0X_F1_Initialize.py`

```
92      # returnerer robot dictionaryen med all informasjon
93      return robot
```

4.4 F2_GetFirstMeasurement.py

F1_Initialize.py
F2_GetFirstMeasurement.py
F3_GetNewMeasurement.py
F4_MathCalculations.py
F5_CalculateAndSetMotorPower.py
F6_SendData.py
F7_CloseMotorsAndSensors.py

Som navnet indikerer er denne fila en erstatning for funksjonen `GetFirstMeasurement` fra det gamle oppsettet. Forskjellen ligger i at alle målingene som blir tatt fra sensorene til EV3en blir lagt til en dictionary for målinger, kalt measurements. I denne dictionaryen vil det også legges til alle andre verdier som brukes (som beregnede verdier og motorpådrag). Med liste strukturen blir alle målte verdier bearbeidet på datamaskinen ved hjelp av `BeregnOgPlott.py`, mens det i dette oppsettet blir gjort av EV3en.

Målingene vil fortsatt ligge i egne lister, men det vil altså være en overordnet struktur, en dictionary, som holder styr på alle målingene.

På linje 5 i kodeutdrag 4.10 blir selve `measurements`-dictionaryen laget. På linje 12, 14 og 16 blir `zerotime` (starttidspunkt) "time"-lista og "ts" (for tidsskritt) lagt til `measurements`.

Kode 4.10: Klargjøring av datalister i `F2_GetFirstMeasurement.py`

```
1  from time import perf_counter
2
3  def P0X_F2_GetFirstMeasurement(robot):
4      # dictionary for å holde styr på alle de ulike målingene
5      measurements = {}
6
7      # Hver måling (og tider) blir lagret i ei liste
8      # Disse listene blir så lagt til i measurements for ...
           fremtidig bruk
9
10     # Tid
11     zeroTime = perf_counter() # dette blir nulltida, og ...
           alle fremtidige tidsmålinger blir satt i forhold ...
           til denne
12     measurements["zerotime"] = zeroTime
13     time = [0] # tida vil bli gitt i forhold til zeroTime
14     measurements["time"] = time
15     ts = [] # tidsskritt
16     measurements["ts"] = ts
```

I kodeutdrag 4.11 blir det tatt målinger fra sensorer og motorer som så legges inn i `measurements`-dictionaryen.

Kode 4.11: Innhenting av målinger fra sensorene i`F2_GetFirstMeasurement.py`

```
18     # Sensorer
19     # Kommenter vekk de som ikke skal brukes!
20     # Alternative funksjoner til sensorene kan finnes på
21     # https://docs.pybricks.com/en/latest/ev3devices.html
22     light = [robot["myColorSensor"].reflection()]
23     measurements["light"] = light
24     #color = [robot["myColorSensor"].color()]
25     #measurements["color"] = color
26     #ambient = [robot["myColorSensor"].ambient()]
27     #measurements["ambient"] = ambient
28
29     touch = [robot["myTouchSensor"].pressed()]
30     measurements["touch"] = touch
31
32     distance = [robot["myUltrasonicSensor"].distance()]
33     measurements["distance"] = distance
34
35     robot["myGyroSensor"].reset_angle(0)
36     gyroAngle = [robot["myGyroSensor"].angle()]
37     measurements["gyroAngle"] = gyroAngle
38     gyroRate = [robot["myGyroSensor"].speed()]
```

```
39      measurements["gyroRate"] = gyroRate
40
41      # Motorer
42      # Kommenter bort de som ikke skal brukes!
43
44      motorAspeed = [robot["motorA"].speed()]
45      measurements["motorAspeed"] = motorAspeed
46      motorAangle= [robot["motorA"].angle()]
47      measurements["motorAangle"] = motorAangle
```

I kodeutdrag 4.12 blir lister for verdier som skal utregnes lagt til, disse er avhengige av hvilket prosjekt som jobbes med. Helt på slutten av fila, på linje 72, blir `measurements`-dictionaryen returnert.

Kode 4.12: Beregnede verdier i `F2_GetFirstMeasurement.py`

```
65      # Her legges til alle de verdiene som skal regnes ut i ...
            prosjektet.
66      # Nedenfor er verdiene som er brukt i prosjektet for ...
            integrasjon gitt.
67      flow = [0]
68      volume = [0]
69      measurements["flow"] = flow
70      measurements["volume"] = volume
71
72      return measurements
```

4.5 F3_GetNewMeasurement.py

F1_Initialize.py
F2_GetFirstMeasurement.py
F3_GetNewMeasurement.py
F4_MathCalculations.py
F5_CalculateAndSetMotorPower.py
F6_SendData.py
F7_CloseMotorsAndSensors.py

Denne fila erstatter `GetNewMeasurement`-funksjonen (som ble presentert i kodeutdrag 3.18) fra det liste oppsettet. Kodeutdrag 4.13 viser en av senso-

rene, metoden for resten er den samme. Alle målingene er i `measurements`-dictionaryen, slik at det må hele tiden refereres til denne dictionaryen for å hente ut eller endre på verdier i listene.

Kode 4.13: Innhenting av nye målinger fra sensorene i `F3_GetNewMeasurement.py`

```
1  from time import perf_counter
2
3  def P02_F3_GetNewMeasurement(robot, measurements):
4      # Legg til differansen mellom nulltida og ...
           perf_counter() ("nå")
5      measurements["time"].append(perf_counter() - ...
           measurements["zeroTime"])
6
7      # Her legges de målte verdiene til.
8      # Husk å kommenter vekk alle sensorer som ikke blir brukt!
9
10     # Legg til den målte lysverdien
11     measurements["light"].append(
12         robot["myColorSensor"].reflection())
13     # Legg til den målte fargeverdien
14     measurements["color"].append(
15         robot["myColorSensor"].color())
16     # Legg til den målte ambient-/lysintensitetsverdien
17     measurements["ambient"].append(
18         robot["myColorSensor"].ambient())
```

4.6 F4_MathCalculations.py

F1_Initialize.py
F2_GetFirstMeasurement.py
F3_GetNewMeasurement.py
F4_MathCalculations.py
F5_CalculateAndSetMotorPower.py
F6_SendData.py
F7_CloseMotorsAndSensors.py

Denne fila er en erstatter for `MathCalculations`-funksjonen fra `BeregnOgPlott.py`. Som nevnt skal alle beregninger gjøres av EV3en i dette

prosjektoppsettet, og denne fila står for de rene matematiske kalkylene. Det er fritt frem for å definere egne variabler til internt bruk i funksjonen dersom det vil være nødvendig for et gitt prosjekt. I kodeutdrag 4.14 vises det hvordan `flow` og `volume` fra integrasjonsprosjektet regnes ut. Her er et av problemene med bruk av `measurements`-dictionaryen, det kan bli vanskelig å formulere og forstå regnestykker i forhold til å bruke variabelnavn. Man kan trekke ut variabler før utregningene, men da er det lite grunn til å bruke dictionary til å begynne med.

Kode 4.14: Matematiske kalkulasjoner i `F4_MathCalculations.py`

```
1  def P02_F4_MathCalculations(measurements):
2      # light deviation
3      measurements["flow"].append(measurements["light"][-1] -
4                                  measurements["light"][0])
5      if len(measurements["flow"]) > 1:
6          # ts
7          measurements["ts"].append(measurements["time"][-1] -
8                                    measurements["time"][-2])
9          # light integrated
10         measurements["volume"].append(
11             measurements["volume"][-1] +
12             measurements["flow"][-2] *
13             measurements["ts"][-1])
```

4.7 F5_CalculateAndSetMotorPower.py

F1_Initialize.py
F2_GetFirstMeasurement.py
F3_GetNewMeasurement.py
F4_MathCalculations.py
F5_CalculateAndSetMotorPower.py
F6_SendData.py
F7_CloseMotorsAndSensors.py

Fila `F5_CalculateAndSetMotorPower.py` beregner motorpådragene og er en erstatning for `CalculateAndSetMotorPower`-funksjonen fra `EV3main.py` fra det gamle oppsettet. I denne funksjonen kan egne variabler defineres og

brukes, som vist på linje 8 og 9 i kodeutdrag 4.15. På linje 15 blir motorpådraget til motor A lagret i measurements dictionaryen før den blir satt på motoren (som skjer på linje 18).

Kode 4.15: Beregning av motorpådrag i `F5_CalculateAndSetMotorPower.py`

```
1  def P0X_F5_CalculateAndSetMotorPower(robot, measurements):
2      # pybricks siden har  alternative funksjoner som kan ...
           utnyttes
3      # for eksempel vil run_time() kjøre motoren med en ...
           konstant
4      # hastighet for en gitt tid (i millisekund)
5
6      # Parametre for beregning til motorpådrag
7      a = 1
8      b = 1
9
10     # Her brukes siste lysmåling for å beregne motorpådrag.
11     lys = measurements["light"][-1]
12
13     # Lagrer pådragene i measurements dictionaryen.
14     measurements["motorPaadragA"] = lys*a
15
16     # Sett hastigen på motorene.
17     robot["motorA"].run(measurements["motorPaadragA"])
```

4.8 F6_SendData.py

F1_Initialize.py
F2_GetFirstMeasurement.py
F3_GetNewMeasurement.py
F4_MathCalculations.py
F5_CalculateAndSetMotorPower.py
F6_SendData.py
F7_CloseMotorsAndSensors.py

Fila `F6_SendData.py` har samme funksjonaliteten som `SendData`-funksjonen fra `EV3main.py`. Fila lagrer alle de siste målingene i `robot["out_file"]`, som er den lokale fila på EV3en hvor målinger blir lagert. Hvis funksjonen

P02_F6_SendData (linje 3 i kodeutdrag 4.16) kjøres i online-modus, vil den også sende siste måling til datamaskinen via
robot["connection"]-socketen for å muliggjør live plotting.

Til forskjell fra fra liste strukturen er her alle målinger som skal sendes allerede i en dictionary, derfor er det mulig å skrive P0X_F6_SendData bare en gang og ikke endre den igjen. På linje 6 i kodeutdrag 4.16 itereres det over alle nøkler i measurement-dictionaryen for å kopiere over det siste resultatet i de tilhørende listene til data-dictionaryen (linje 10) som skal sendes til datamaskin og dataString (linje 11). Når data skal plottes, blir verken nulltida eller tidsstegene tatt med, så disse er ikke nødvendige å overføre til datamaskinen, linje 7 og 8 sørger for at de ikke blir lagt til i data-dictionaryen. Data skrives til fil på linje 14, på linje 11 ble det lagt til et komma før hver måling, derfor må denne ekslkuderes fra dataString når den skrives til fil. Strenger kan i Python indekseres som lister, så det som skrives til fil er dataString fra andre karakter til endes.

Kode 4.16: Klargjøring av data i F6_SendData.py

```
1  import json
2
3  def P02_F6_SendData(robot, measurements, online):
4      data = {}
5      datastring = ""
6      for key in measurements:
7          if key == "zeroTime" or key == "ts":
8              continue
9
10         data[key] = (measurements[key][-1])
11         dataString += "," + str(data[key])
12
13     # Write dataString to file, excepting the first comma.
14     robot["out_file"].write(dataString[1:])
```

Denne versjonen av sendData bruker en mindre effektiv måte å sørge for at data blir behandlet melding for melding. Istedenfor å sette inn en karakter for å indikere meldingsslutt, venter EV3 på beskjed om at meldingen er mottatt før den går videre (linje 17 kodeutdrag 4.17). Denne ventingen fører til at tidsoppløsningen for sensorer blir mye lavere, målinger blir gjort sjeldnere siden programmet her venter litt før det går videre.

Kode 4.17: Sending av data til datamaskin ved hjelp av F6_SendData.py

```
12      if online:
13          msg = json.dumps(data)
14          robot["connection"].send(msg)
15
16          # Wait until pc has acknowledged data before ...
                continuing.
17          if not robot["connection"].recv(3) == b"ack":
18              print("No data ack")
```

4.9 F7_CloseMotorsAndSensors.py

F1_Initialize.py
F2_GetFirstMeasurement.py
F3_GetNewMeasurement.py
F4_MathCalculations.py
F5_CalculateAndSetMotorPower.py
F6_SendData.py
F7_CloseMotorsAndSensors.py

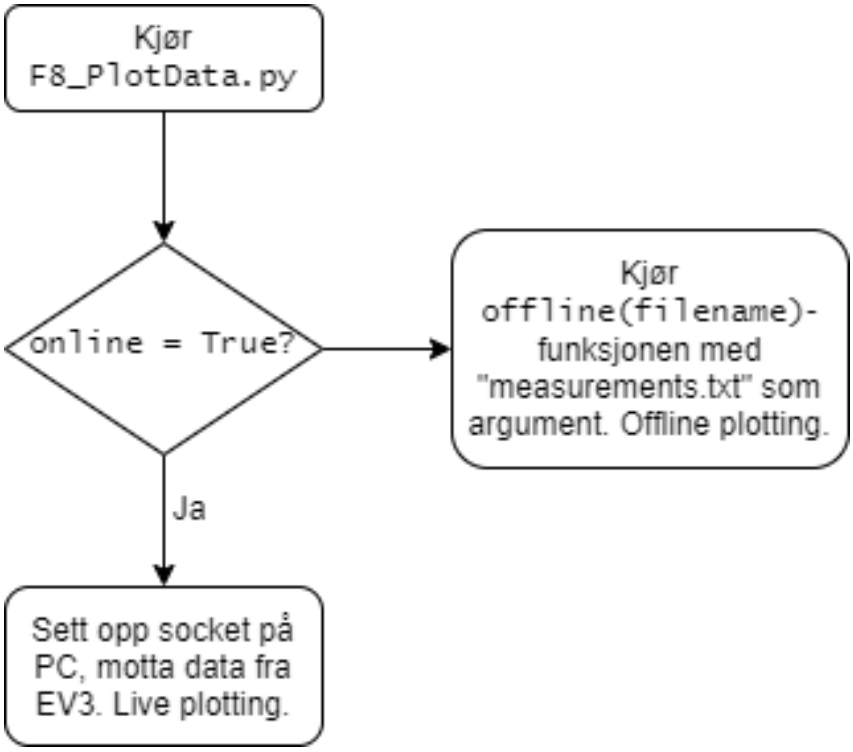
Denne fila er helt identisk med CloseMotorsAndSensors.py-funksjonen fra EV3main.py, presentert i kodeutdrag 3.23.

4.10 F8_PlotData.py

Denne fila kan kjøres enten i sanntid eller etter at et eksperiment er gjennomført. Dette vil da være en erstatning for BeregnOgPlott.py fra det forrige oppsettet, med forskjellen at F8_PlotData.py ikke utfører noen beregninger selv (dette blir gjort av F4_MathCalculations() fra F4_MathCalculations.py fila, se kodeutdrag 4.14). I likhet med BeregnOgPlott.py vil F8_PlotData.py-fila også kjøres på datamaskinen, og ikke på EV3en.

F8_PlotData.py har mange likheter med BeregnOgPlott.py fra det forrige oppsettet og det vil refereres tilbake til de relevante kodeutdragene fra BeregnOgPlott.py for nærmere kodeforklaringer.

Et forenklet flytskjerma for `F8_PlotData.py` er gitt i figur 4.5. Det første fila gjør er å sjekke om `online`-variabelen er satt til `True` eller `False`. Dersom `online = True` kjøres det med live plotting, og det blir satt opp socketobjekt for å motta data fra EV3en. Dersom `online = False` blir det plottet med verdier som allerede ligger lokalt på PCen. Verdiene vil være hentet fra filen med navn gitt av`filename`-variabelen.



Figur 4.5: Flytskjema for `F8_PlotData.py`

I kodeutdrag 4.18 vises importsetningene i `F8_PlotData.py`, som er identiske med importsetningene i kodeutdrag 3.27 for `BeregnOgPlott.py`

Kode 4.18: `F8_PlotData.py`

```
1 import matplotlib.pyplot as plt
2 from matplotlib.animation import FuncAnimation
3 import socket
4 import sys
5 import json
```

Innholdet i kodeutdrag 4.19 er identisk med innholdet i kodeutdrag 3.29 fra `BeregnOgPlott.py`. `online` bestemmer og det kjøres i online eller ikke, `EV3_IP` er EV3ens IP adresse, og `filename` er navnet på fila hvor målingene ligger (viktig dersom det kjøres i med `online = False`.

Kode 4.19: F8_PlotData.py

```
7  online = True
8  EV3_IP = "192.168.2.2"
9  filename = "measurements.txt"
```

Neste steg er å definere lister for verdier som skal plottes, dette gjøres på samme måte som i kodeutdrag 3.30

I kodeutdrag 4.20 blir det definert en figur og tre subplot, med innhold som er nesten likt kodeutdrag 3.31 fra `BeregnOgPlott.py`. Eneste forskjell er at her er de forskjellige aksene definert i egne variabler. Dette ble endret til liste strukturen da ved mange akser ble enklere å ha oversikt med indeksering, spesielt ved akser i to dimensjoner.

`plotData`-funksjonen i kodeutdrag 4.20 er veldig lik `plotData`-funksjonen fra `BeregnOgPlott.py`, som ble presentert i kodeutdrag 3.31. Her ble enda `ax.cla()` metoden brukt siden vi ikke hadde oppdaget ytelseskostnaden av dette, av samme grunn settes også titler og andre egenskaper i hoved plott funksjonen.

Kode 4.20: F8_PlotData.py

```
19 fig, (ax1, ax2, ax3) = plt.subplots(nrows=3, ncols=1, ...
       sharex=True)
20
21
22 def plotData():
23     ax1.cla()
24     ax2.cla()
25     ax3.cla()
26
27     ax1.plot(time[0:-1], light[0:-1])
28     ax1.set_title('Light')
```

```
29
30      ax2.plot(time[0:-1], lightDeviation[0:-1])
31      ax2.set_title('lightDeviation')
32
33      ax3.plot(time[0:-1], lightIntegrated[0:-1])
34      ax3.set_title('Light Integrated')
35      ax3.set_xlabel('Time [sec]')
```

I kodeutdrag 4.21 presenteres `live`-funksjonen, som prinsipielt fungerer likt `live`-funksjonen fra `BeregnOgPlott.py` (presentert i kodeutdrag 3.38 til 3.42). De vesentlige forskjellene er: Data legges til listene i `live`-funksjonen istedenfor `appendLists`-funksjonen, mangel av kall på `MathCalculations`-funksjonen, og sending av data mottatt beskjed.

Siden utregninger nå skjer på EV3 er det ikke nødvendig å gjøre dem her, data til lister ble flyttet til sin egen funksjon for å ha klarere skille mellom kode studenter skal/ikke skal endre på. Linje 66 sender signal til EV3 om at data er mottatt, fordelen med denne metoden overfor karakter terminerte meldinger er begrenset til at koden er enklere å skrive. Siden denne metoden resulterer i at programmet på EV3 kjører senere ble dette skrevet om til liste rammeverket, relevant kode presentert i seksjon 3.3.5.

Kode 4.21: F8_PlotData.py

```
38  def live():
39      # Recieve data from EV3.
40      try:
41          data = sock.recv(1024)
42      except:
43          print("Lost connection to EV3")
44          livePlot.event_source.stop()
45          return
46
47      if data == b"end":
48          print("Recieved end signal")
49          livePlot.event_source.stop()
50          return
51
52      # If data is not the end signal, decode it.
53      try:
54          data = json.loads(data)
55      except:
56          print("Lost connection to EV3")
57          livePlot.event_source.stop()
```

```
58          return
59
60      light.append(data["light"])
61      time.append(data["time"])
62      flow.append(data["flow"])
63      volume.append(data["volume"])
64
65      # Ensure no more data is sent before this set is handled
66      sock.send(b"ack")
67
68      plotData()
```

Funksjonen `offline()` som defineres på linje 70 i kodeutdrag 4.22 mangler også kall til en funksjon for matematisk beregninger, i likhet med `live`-funksjonen blir data lagt i lister her og ikke i egen funksjon.

Kode 4.22: F8_PlotData.py

```
70  def offline(filename):
71      # Read from file row by row and append data to lists.
72      with open(filename) as f:
73          for row in f:
74              row = row.split(",")
75              light.append(int(row[0]))
76              time.append(float(row[1]))
77              flow.append(int(row[2]))
78              volume.append(float(row[3]))
79
80      # Plot data in lists.
81      plotData()
82      # Set plot layout and show plot.
83
84      fig.set_tight_layout(True)# mac
85
86      plt.show()
```

Kodeutdrag 4.23 gjør kall til funksjonene i `F8_PlotData.py` (`live`, og `offline`), og er identisk med innholdet som ble presentert i kodeutdrag 3.43, 3.44, og 3.45 for `BeregnOgPlott.py`.

Kode 4.23: F8_PlotData.py

```
69  if online:
70      sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
```

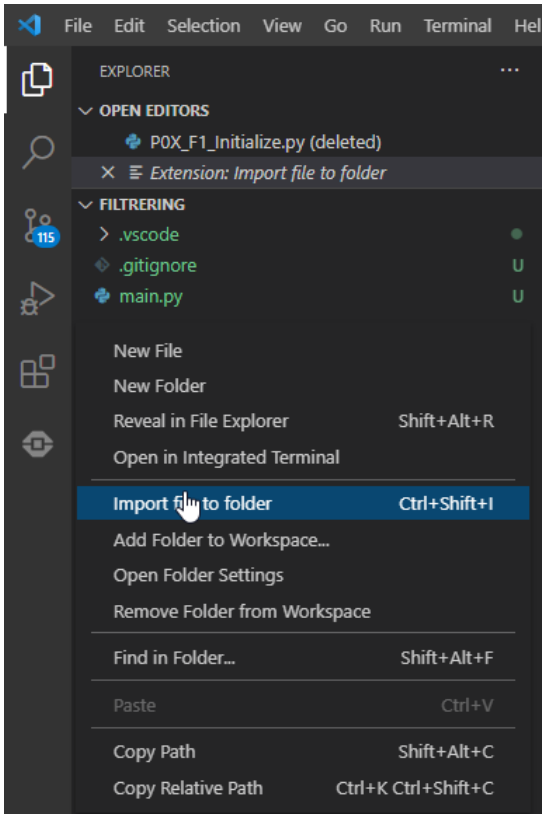
```
71      try:
72          addr = (EV3_IP, 8070)
73          print("Attempting to connect to {}".format(addr))
74          sock.connect(addr)
75          data = sock.recv(1024)
76          if data == b"ack":
77              print("Connection established")
78          else:
79              print("no ack")
80              sys.exit()
81      except socket.timeout:
82          print("failed")
83          sys.exit()
84
85      livePlot = FuncAnimation(fig, live, interval=10)
86      plt.tight_layout()
87      plt.show()
88  else:
89      offline("measurements.txt")
```

Kapittel 5

Filtrering

5.1 Hvordan et nytt prosjekt startes

Den enkleste måten å starte et nytt prosjekt på er ved å installere utvidelsen "Import file to folder". Mer om utvidelser og generell bruk av Visual Studio Code kan leses om i vedlegg B. Etter installasjon av utvidelsen er det bare å starte et nytt prosjekt som vist i seksjon 2.3. Ved å høyreklikke i mappevinduet under "Explorer" (åpnes ved å taste Ctrl+Shift+E), også klikke på "Import file to folder" (eller bare Ctrl+Shift+I) kommer alle filene opp. Alle filene i Prosjekt0X-mappa velges og importeres. Etter import av filene, må `main.py` filen slettes. Etter dette er det bare å begynne på det nye prosjektet.



Figur 5.1: "Import file to folder" i aksjon.

5.2 Introduksjon av filtreringsprosjektet

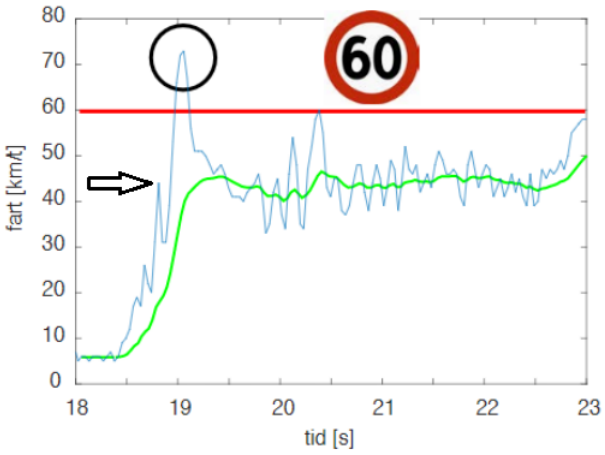
For å kunne verifisere prosjektstrukturen som er lagd, er man nødt til å gjennomføre prosjekter på samme måte som studentene er nødt til å lage i faget ELE-130. Derfor blir dette kapittelet et eksempel på hvordan elevene kan utføre dette på en praktisk måte. På grunn av at disse prosjektene ikke var hovedmålet med oppgaven, ble det bestemt å holde ett av disse prosjektene som ett kapittel, mens resten av prosjektene som ble lagd, er satt inn i vedlegg. De kan man lese mer om i vedlegg: G, H, I, og J.

I dette kapitlet blir det sett på hvordan signal fra lyssensoren til EV3en kan filtreres. Å filtrere et signal vil si å fjerne deler, ofte kalt støy, av signalet som regnes som uønsket. På grunn av støy som oppstår i den virkelige verden vil målte signal kunne gi et feilaktig bilde av en situasjon. Ved å filtrere signal kan en fjerne uønskede komponenter fra målingene og dermed gi et "finere" bilde av en situasjon.

Dette er en del av den obligatoriske delen av prosjektet fra prosjektbeskrivelsen i vedlegg K.

5.3 Praktisk bruk av filtrering

En situasjon hvor filtrering kan være nyttig er for politipatruljer som måler farten til bilister og andre forbikjørende ved hjelp av en lasermåler. En bilist kjører forbi og lasermåleren (uten filtrering) måler at bilisten kjørte langt over fartsgrensa på 60 km/t, som vist i den svarte sirkelen i figur 5.2. Den blå kurven viser målingene fra lasermåleren, og ved å se på kurven indikert med den svarte pilen danner det seg et rart bilde. Litt før det har gått 19 sekunder, går bilen fra å kjøre rundt 45 km/t, til 30 km/t, og deretter helt opp til nesten 75 km/t. I denne hypotetiske situasjonen klarer politipatruljen å se at dette absolutt ikke kan stemme, så heldigvis ble det ingen bot på bilisten. I praksis vil signalet fra lasermåleren bli filtrert for å gi et mer realistisk bilde av situasjonen, og et slikt filtrert signal er vist på den grønne kurven i figur 5.2. Ved å se på det filtrere signalet ser politipatruljen at bilisten ikke kjørte over fartsgrensa; den kjørte godt under grensa.



Figur 5.2: Teoretisk måling av fart ved hjelp av lasermåler. Den blå kurven representerer de faktiske målingene fra lasermåleren, den grønne kurven representerer en filtrering av den blå kurven.

En annen situasjon hvor filtrering er nyttig er i et digitalt steketermometer til bruk i et vanlig kjøkken. Et spesifikt eksempel er et steketermometer som settes ned i en kopp med kaffe, som vist i figur 5.3. Steketermometeret som brukes i figuren består av en motstand laget i platina og er innkapslet i en

tynn metallkappe. Navnet til motstanden er "PT100-element" som betyr at den er laget av platina ("PT") og motstanden er 100 Ω ved 0°C. I programvaren til steketermometeret er det implementer funksjonalitet for filtrering slik at displayet i steketermometeret gir mer brukervennlige verdier. Dette vil si at temperaturforandringene ikke skjer for fort når steketermometeret settes ned i kaffekoppen. Fra figuren ser vi at selv om den egentlige temperaturen til mostanden er lik temperaturen til kaffen i $t \approx 0$, vises ikke dette før $t = 5$.

Figur 5.3: Steketermometeret satt ned i en kopp med halvlunken kaffe for et eksperiment utført av Tormod Drengstig. På grunn av filter i programvaren til steketermometeret bruker displayet 5 sekunder før riktig temperatur blir vist, og siden dette steketermometeret har en oppdateringsfrekvenz på 1 Hz viser figuren det faktiske forløpet. Bildet er hentet fra figur 5.2 i "LEGO Mindstorms og MATLAB; anvendt mate-matikk/fysikk og programmering i skjønn forening".

5.4 Problemstilling

I kapitlet presenteres det to former for filtrering; FIR-filtrering (*Finite Impulse Response*), og IIR-filtrering (*Infinite Impulse Response*). Alle signalene som filtreres hentes fra lyssensoren til EV3en. Prosjektet kjøres i offline-modus; det vil si at det blir gjort målinger (uten noen form for live plotting av data) fra lyssensoren, som det blir kjørt beregninger på i ettertid.

Lyssignalene ble tolket som temperaturmålinger; da kan det lyssensoren sees på som et steketermometer. Ved å benytte arket i figur 5.4, kan det tenkes at den mørke delen av arket representerer lav temperatur, mens den lyse delen representerer høy temperatur. Ved å raskt føre lyssensoren fra den mørke delen av arket til den lyse delen, ble det simulert en situasjon hvor et steketermometer blir satt ned i en kaffekopp.

Figur 5.4: Ark som brukes for å simulere måling av temperatur ved hjelp av lyssensoren til EV3en. De mørke områdene har lav refleksjon og gir dermed lave verdier fra lyssensoren; disse områdene representerer lav temperatur. De lyse områdene har høy refleksjon og dette gir høye verdier som da representerer høy temperatur.

5.5 Teori

5.5.1 FIR-filtrering

(*Finite Impulse Response*) er en teknikk for å glatte ut signaler ved å ta en vektet sum av de m siste målingene. I denne oppgaven fikk alle m målinger lik vekt, slik at den vektede summen er lik middelverdien av de m siste målingene. En slik form for FIR-filtrering blir kalt for et glidende midlingsfilter (*moving average filter* eller MA-filter på engelsk). Den generelle formen for en slik type FIR-filtrering (basert på lysverdier fra lyssensoren som tolkes som temperaturverdier $Temp$) er gitt i likning 5.1.

$$Temp\_FIR(k) = \frac{1}{m}\sum_{n=0}^{m-1} Temp(k-n) \tag{5.1}$$

5.5.2 IIR-filtrering

IIR-filtrering (*Infinite Impulse Response*) er en form for rekursiv filtrering som bruker de siste målte verdiene samt gamle filtrerte verdier. Fordelen med denne formen for filtrering i forhold til FIR-filtrering er at IIR-filtrering ikke krever mange av de gamle målingene for å få effektiv filtrering.

Et eksempel på det som kalles en første-ordens filtrering er gitt i likning 5.2. Dette er en første-ordens filtrering siden den kun tar med siste filtrert verdi ($Temp\_IIR(k-1)$), mens en andre-ordens filtrering tar med de to siste ($Temp\_IIR(k-1)$ og ($Temp\_IIR(k-2)$)), og en tredje-ordens filtrering tar med de tre siste ($Temp\_IIR(k-1)$, $Temp\_IIR(k-2)$, $Temp\_IIR(k-3)$) og så videre.

$$Temp\_IIR(k) = \alpha \cdot Temp(k) + (1-\alpha) \cdot Temp\_IIR(k-1), \alpha \in (0,1) \tag{5.2}$$

IIR-filtrering er som sagt rekursiv, og det ser vi i likning 5.2 ved at $Temp\_IIR(k)$ er avhengig av $Temp\_IIR(k-1)$. I denne oppgaven ble $Temp\_IIR(0)$ satt lik $Temp(0)$ for å løse dette. Dette vil si at første filtrerte verdi, er satt lik

den første temperaturverdien. Da vil "første" filtrerte verdi $Temp\_IIR(1)$ være lik $\alpha \cdot Temp(1) + (1 - \alpha)Temp(0)$

Ved å velge $\alpha = 0$ vil likning 5.2 bli lik likning 5.3. Likning 5.3 er kun avhengig av siste filtrerte verdi, og vil ikke inneholde noe om fremtidige lysverdier.

$$Temp\_IIR(k) = \cdot Temp\_IIR(k-1) \tag{5.3}$$

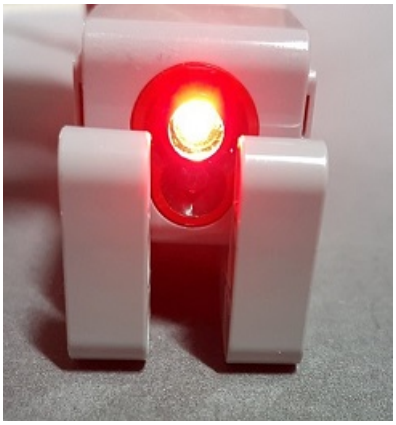
Ved å velge $\alpha = 1$ vil likning 5.2 bli om til likning 5.4. Likning 5.4 har ikke noe filtrering i det hele tatt siden den vil kun være avhengig av siste lysverdi.

$$Temp\_IIR(k) = \cdot Temp(k) \tag{5.4}$$

5.6 Forslag til løsning

5.6.1 Praktisk

For å simulere et steketermometer som blir satt ned i en kaffekopp ble arket i figur 5.4 anvendt. Lyssensoren ble brukt med avstandsklosser, dette er vist i figur 5.5 og figur 5.6. Lyssensoren ble holdt på mørkt område i ca. 4 sekunder, for så å bli skjøvet (veldig fort) til det lyse området. Deretter ble lyssensoren holdt på det lyse området i ca. 10 sekunder. Dette representerer en situasjon hvor et steketermometer ligger i ro i ca. 4 sekunder, for så å bli satt ned i en varm kaffekopp i ca. 10 sekunder.



Figur 5.5: Lyssensoren med avstandklosser fra fremsiden.

Figur 5.6: Lyssensoren med avstandklosser fra venstre side.

5.6.2 `EV3main.py`

`EV3main.py` ble som nevnt kjørt i offline-modus, det vil si at data ikke ble sendt til datamaskinen fra EV3en, men kun ble lagret lokalt på EV3en. Av sensoren var det kun lyssensoren som ble initialisert; den ble satt opp for å være koblet til port 1. Dette ble gjort i `Initialize`-funksjonen med følgende linje:

Kode 5.1: `EV3main.py`

```
126 myColorSensor = ColorSensor(Port.S1)
```

I `GetFirstMeasurement`-funksjonen ble første lysmåling tatt og lagt til `light`-listen (som også ble initialisert på samme linje). Nulltida ble også satt her, og tidslista blir initialisert med 0 som første element, for å representere første måling. Dette er vist i kodeutdrag 5.2.

Kode 5.2: `EV3main.py`

```
147 # Tider
148 zeroTimeInit = perf_counter()  # nulltida
149 time = [0]  # tida
150
151 # Sensorer
152 light = [myColorSensor.reflection()]
```

I `GetNewMeasurement`-funksjonen (som ble kalt for hver kjøring av while-løkka i `main`-funksjonen) ble det lagt til nye tider og nye lysmålinger. Dette er vist i kodeutdrag 5.3. Legg merke til at nye tidsmålinger ble tatt i forhold til nulltida, `zeroTimeInit`, som ble satt i kodeutdrag 5.2.

Kode 5.3: `EV3main.py`

```
174 time.append(perf_counter() - zeroTimeInit)
175 light.append(myColorSensor.reflection())
```

5.6.3 `BeregnOgPlott.py`

`BeregnOgPlott.py` ble også kjørt i offline-modus, siden `EV3main.py` ble kjørt i offline-modus. I `BeregnOgPlott.py`-fila ble det initialisert en del lister for å representere alle de ulike verdiene (både beregnede verdier, og målte verdier som ble hentet fra målefila som ble lagret på EV3en etter kjøring av `EV3main.py` i offline-modus). Dette er vist i kodeutdrag 5.4. `temp`-lista tilsvarer `light`-lista fra `EV3main.py`; `temp` ble fullt ut med målingene fra lyssensoren. `ts` tilsvarer tidssteget og ble beregnet basert på tidsmålingene som ble gjort. `avgts` ble satt lik det gjennomsnittlige tidssteget (dette kommer senere i koden).

Ellers er kalkulasjon av FIR- og IIR-filter basert på likningene 5.1 og 5.2.

Kode 5.4: `BeregnOgPlott.py`

```
12 # temperatur, tid, tidsskritt
13 temp = []
14 time = []
15 ts = []
16 avgts = 0
```

Det ble også initialisert lister for å holde på verdiene for FIR- og IIR-filtreringen. Dette er vist i kodeutdrag 5.5. Relevante parametre (`m` og `alpha`) er også tatt med her, og er gitt initialverdier (`m` = 4 og `alpha` = 0.5). Initialverdiene er ikke relevante, og ble variert på under kjøring av `BeregnOgPlott.py`.

Kode 5.5: `BeregnOgPlott.py`

```
19 # FIR-filtrering
20 temp_fir = []
21 # bestemmer "mengden" filtrering
22 m = 4
23
24 # IIR-filtrering
25 temp_iir = []
26 # "vekten"
27 alpha = 0.5
```

Beregningene blir utført i `MathCalculations`-funksjonen. Selv om prosjektet ble kjørt i offline-modus ble det også laget beregninger for online-kjøring. På linje 35 i kodeutdrag 5.6 blir det vist en sjekk for om det kjøres i offline-modus. Hvis det kjøres i offline-modus er all data tilgjengelig.

Kode 5.6: `BeregnOgPlott.py`

```
35 if not online: # hvis offline modus
```

I kodeutdrag 5.7 vises beregning av tidsstegget i offline-modus. `if`-setningen på linje 38 sørger for at det ikke prøves å hente ut en verdi i `time`-lista som ikke finnes; `time[len(time)]`.

Kode 5.7: `BeregnOgPlott.py`

```
36      #  tidssteg
37      for i in range(len(time)):
38          if i+1 < len(time):
39              ts.append(time[i+1] - time[i])
```

Kalkulasjon av FIR-filter i offline-modus er vist i 5.8. Den letteste måten å gjøre det på var å simulere online kjøring ved å lage ei ny liste, `newlist`, som da representerer ei liste som får inn "lysmålinger". "Lysmålingene" kommer direkte fra `temp`-lista. På linje 45 sjekkes det tilfellene hvor `m` er større enn antall målinger. I slike tilfeller kan ikke den vanlige utregningsmetoden brukes, siden det ikke er nok målinger. Da brukes en annen utregning hvor det fortsatt blir tatt snittet av de siste målingene. Ellers brukes `newlist[-m:]` for å ta ut de `m` siste verdiene i `newlist`-lista.

Kode 5.8: `BeregnOgPlott.py`

```
41      # FIR-filter
42      newlist = []
43      for i in range(len(temp)):
44          newlist.append(temp[i])
45          if i < m-1:
46              temp_fir.append( (1/(1+i)) * sum(newlist[-m:]) )
47          else:
48              temp_fir.append((1/m) * sum(newlist[-m:]))
```

I kodeutdrag 5.9 blir IIR-filter vist for offline-modus. På linje 51 legges første verdi i `temp` til `temp_iir`, siden IIR-filter er rekursiv og trenger et basetilfelle.

Kode 5.9: `BeregnOgPlott.py`

```
50      # IIR-filter
51      temp_iir.append(temp[0])
52      for i in range(len(temp)):
53          temp_iir.append(alpha*temp[i] + (1-alpha)*temp_iir[i])
```

Linje 55 i kodeutdrag 5.10 er `else`-setningen som tilhører `if`-setningen fra kodeutdrag 5.6. `else`-setningen kjøres hvis `BeregnOgPlott.py` kjøres i online modus. Dette ble ikke gjort i dette eksperimentet (bortsett fra for å sjekke at koden kjørte), men er tatt med for ordens skyld.

Kode 5.10: `BeregnOgPlott.py`

```
55  else:
```

Beregning av tidsskritt i online modus er vist i kodeutdrag 5.11. Siste tidsskritt tilsvarer de to siste verdiene i `time`-lista. På linje 57 blir det tatt en sjekk som sørger for at det er minst 2 verdier i `time`-lista.

Kode 5.11: `BeregnOgPlott.py`

```
56  # tidsskrittet
57  if len(time) > 1:
58      ts.append(time[-1] - time[-2])
```

Kalkulasjon av FIR-filter er enklere i online-modus, og er vist i kodeutdrag 5.12. Den samme sjekken som ble gjort for kalkulasjon av FIR-filter i offline-modus (som vist i kodeutdrag 5.8) blir også gjort her. Dette er altså fordi det må tas høyde for tilfeller hvor det ikke er nok målinger i `temp`-lista.

Kode 5.12: `BeregnOgPlott.py`

```
60  # FIR-filter
61  if i < m-1:
62      temp_fir.append( (1/(i+1)) * sum(temp[-m:]))
63  else:
64      temp_fir.append((1/m) * sum(temp[-m:]))
```

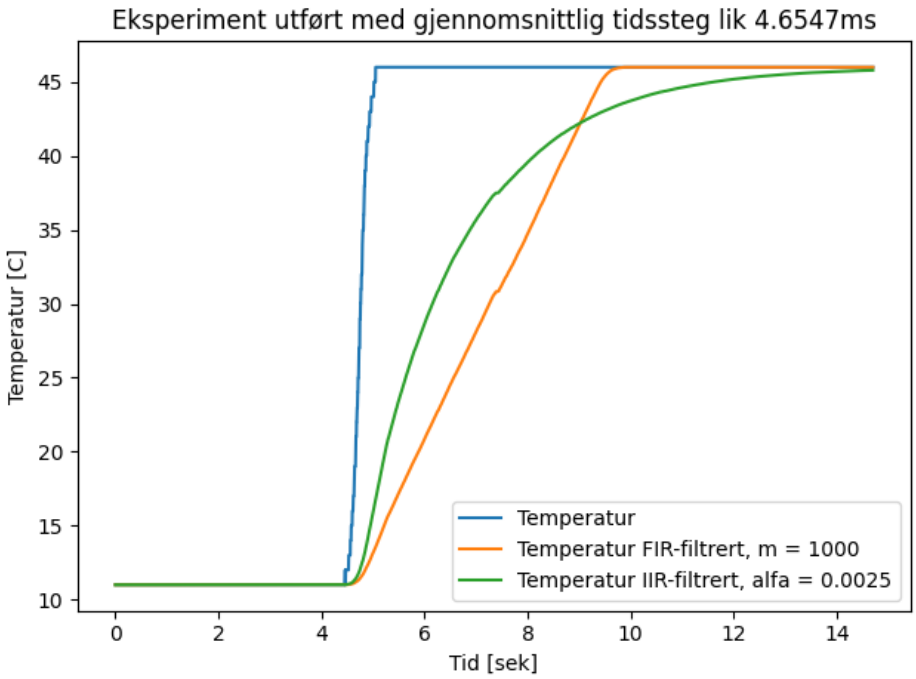
Kalkulasjon av IIR-filter i online-modus er vist i kodeutdrag 5.13. For online-modus er det nødvendig å legge til de første målte verdi i `temp` til `temp_iir`-lista, dette skjer på line 65.

Kode 5.13: `BeregnOgPlott.py`

```
63  # IIR-filter
64  if len(time) == 1:
65      temp_iir.append(temp[0])
66  temp_iir.append(alpha*temp[-1] + (1-alpha)*temp_iir[-1])
```

5.7 Resultat

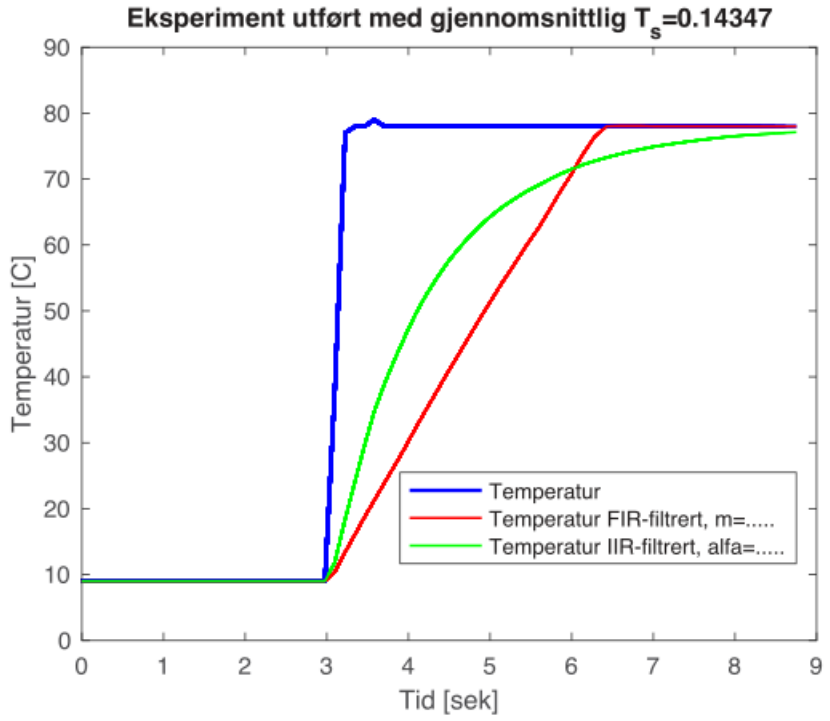
Etter å ha kjørt EV3en slik som beskrevet i 5.6.1, ble parametrene `m` og α justert til vi fikk responsen illustrert i figur 5.7. Dette var for å få en respons liknende den som ble vist i prosjektbeskrivelsen for ING100; den er gjengitt i figur 5.8 I figuren viser den blå kurven de faktiske målte verdiene fra lyssensoren (her kalt `Temperatur`), den oransje kurven viser den FIR-filtrere verdien av den blå kurven, mens den grønne kurven viser den IIR-filtrerte verdien av den blå kurven.



Figur 5.7: Filtrering av `Temperatur` (egentlig lysverdier fra lyssensoren til EV3en) med `m` = 1000 og $\alpha = 0.0025$. Det gjennomsnittlige tidssteget for eksperimentet var $4.6547ms$ som tilsvarer en samplingsfrekvens på ca 215 Hz.

I forhold til eksperimentene som blir utført ING100, er det mye kortere tidssteg her. Det fører til en høyere samplingsfrekvens, og mer data å jobbe med som er ganske positivt. Dette er en fordel med å bruke Python (eller MicroPython på EV3en) fremfor MATLAB.

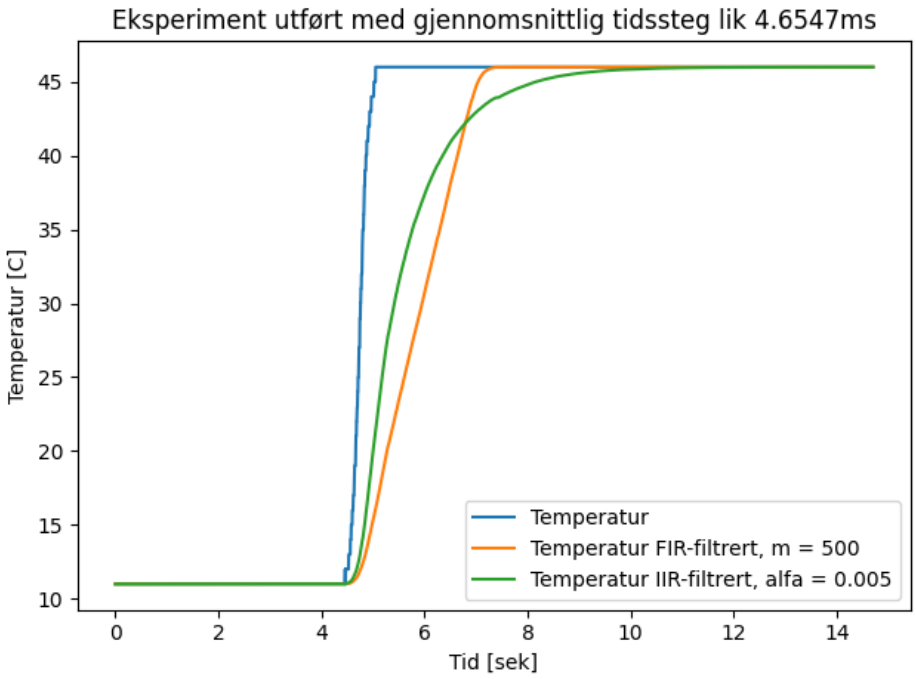
Ved å igjen tenke på situasjonen med et steketermometer som settes nedi kaffekoppen, kan det tenke seg at steketermometeret bruker en av de filtrere verdiene for å vise temperaturen. Istedenfor at temperaturen går rett fra lav til høy, kan en filtrer verdi brukes for å gi et mer brukervennlig grensesnitt.



Figur 5.8: Dette er figur 5.6 fra *"LEGO Mindstorms og MATLAB; anvendt matematikk/fysikk og programmering i skjønn forening"*. Oppgaven ber studentene om å variere på `m` og α til det kommer frem en respons liknende denne figuren. Dette fikk vi ved `m` $= 1000$ og $\alpha = 0.0025$, som vist i figur 5.2.

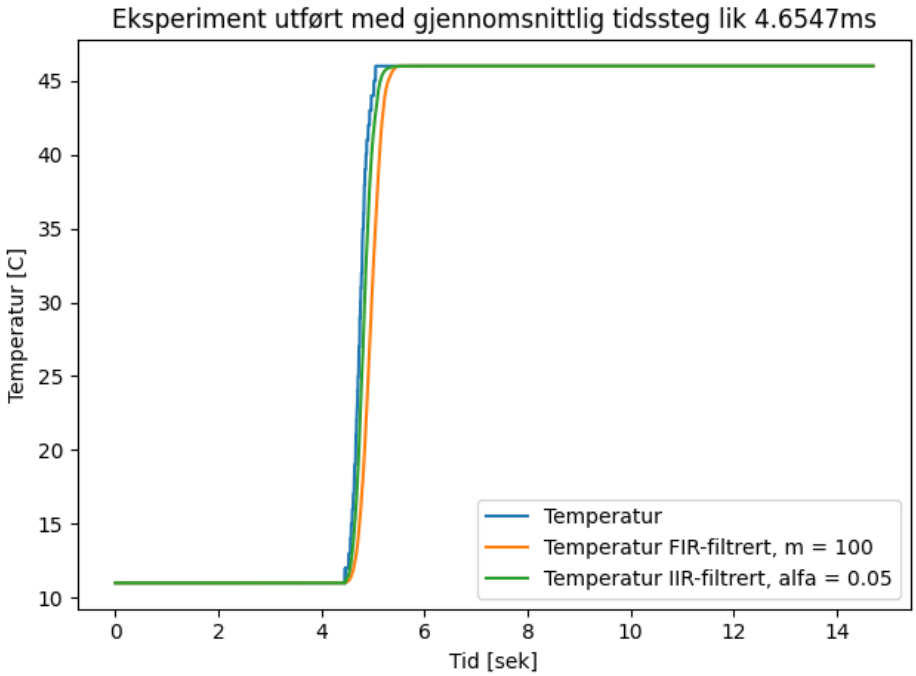
Ved å variere på `m` og α vil grafen se ganske forskjellig ut. Jo mindre `m` verdi som brukes, jo mindre FIR-filtrering vil det være, og jo større α verdi som brukes, jo mindre IIR-filtrering. Dette sees fra likningene i 5.1 og 5.2. For IIR-filtrering, vil en mindre α verdi føre til at `TEMP_IIR(k-1)` har en større effekt i forhold til `Temp(k)`. Det vil si at hvis α er veldig stor, så vil den forrige målte temperaturverdien vektlegges mye, og det vil da føre til mindre filtrering.

I figur 5.9 vises hva som skjedde når det ble brukt $\texttt{m} = 500$ og $\alpha = 0.005$. Dette tilsvarer en halvering av `m`, og en dobling av α fra eksperiment i figur 5.2. Det er merkbart mindre filtrering her.



Figur 5.9: Filtrering med $\texttt{m} = 500$ og $\alpha = 0.005$.

I figur 5.10 viser hva resultatet vil bli ved bruk av $\texttt{m} = 100$ og $\alpha = 0.05$. Dette tilsvarer en halvering av `m`, og en dobling av α. I dette tilfelle brukes det lite filtrering.



Figur 5.10: Filtrering med $\mathtt{m} = 100$ og $\alpha = 0.05$.

5.7.1 Filtrering av målestøy i en temperaturtransmitter

I denne delen av oppgaven simulerte vi en situasjon hvor temperaturmålingene kom fra den industrielle temperaturtransmitteren vist i figur 5.11. En transmitter som dette brukes for eksempel hos Nortura for å forsikre seg at melken (som fraktes i rør) har riktig temperatur. For å late som om målingene kom fra temperaturtransmitteren, la vi til støy i målingene, for å representere temperaturvariasjoner i melka. Oppgaven bruker kun IIR-filtrering, og kjøres i live-modus, samtidig som det blir manuelt lag til støy i målingene tatt fra lyssensoren. Dette er for å simulere tilfeldige temperaturvariasjoner.

Figur 5.11: En temperaturtransmitter som brukes i industrien [24]. Denne går under navnet `RTD temperature transmitter T1000` og blir laget i Taiwan.

Oppsettet fra ING100 sier at at studenter skal bruke `randn`-funksjonen i MATLAB for å generere støy. I dokumentasjonen til `randn`-funksjonen fra MATLAB[20], står det at funksjonen returnerer en tilfeldig skalar fra den standardiserte normalfordelingen (altså med $\mu = 0$ og $\sigma = 1$). I vanlig Python så kan `gauss`-funksjonen fra `random`-modulen i Python utføre den samme funksjonen som `randn`-funksjonen i MATLAB, dette står det om i Python-dokumentasjonen (klikkbar lenke). `gauss`-funksjonen tar inn to parametre, μ, som er forventningsverdien og σ som er standardavviket. Ved å bruke `random.gauss(0,1)` i Python oppnås det samme funksjonalitet som MATLABs `randn`-funksjon, dette er vist i kodeutdrag 5.14.

Kode 5.14: Generering av tilfeldige tall fra standardnormalfordelingen i Python

```
1 import random
2 random.gauss(0,1)
```

Koden fra kodeutdrag 5.14 ble lagt til i `live`-funksjonen i `BeregnOgPlott.py`, som vist i kodeutdrag 5.15.

Kode 5.15: Generering av støy i BeregnOgPlott.py

```
139 # Append the recieved data to the lists.
```

```
140  temp.append(data["light"] + random.gauss(0,1))
```

Som en liten ekstra oppgave sammenlignet vi datamaskinen og EV3ens evne til å kjøre matematiske funksjoner for å se om det var en forskjell i det gjennomsnittlige tidsskrittet (hvor lang tid det går mellom hver måling). Dette gjorde vi ved å kjøre generasjon av støy både på datamaskin (i `BeregnOgPlott.py`) og på EV3en (i `EV3main.py`).

For å utføre dette måtte det også brukes `random`-modul i MicroPython, og denne modulen ligner veldig på vanlig Pythons `random`-modul, bortsett fra at MicroPythons versjon ikke har en `gauss`-funksjon. MicroPython har ingen funksjon som direkte kan anvendes for å generere tall fra den standardiserte normalfordelingen (eller noen annen normalfordeling). For å kunne sammenligne EV3ens og datamaskinens evne til å kjøre matematiske operasjoner endte vi opp med å generere våre egne tall fra standardnormalfordelingen ved hjelp av *Box-Muller Transformation* metoden.[6]

Box-Muller Transformation metoden krever to uavhengige "prøver", kall dem x_1 og x_2, fra en uniform sannsynlighetsmodell på $[0, 1]$. Ved hjelp av x_1 og x_2 kan det genereres to tall z_1 og z_2 fra en normal fordeling med $\mu = 0$ og $\sigma = 1$, fra den standardiserte normalfordelingen. Formelene for z_1 og z_2 er gitt i ligning 5.5 og ligning 5.6.

$$z_1 = \sqrt{-2lnx_1}\cos(2\pi x_2) \tag{5.5}$$

$$z_2 = \sqrt{-2lnx_1}\sin(2\pi x_2) \tag{5.6}$$

For å implementere Box-muller metoden blir `uniform`-funksjonen fra `random`-modulen brukt, og denne funksjonen finnes både i MicroPython og vanlig Python. Den tar inn to parametre; en nedre og øvre grense, som blir satt lik 0 og 1 for å få den uniforme sannsynlighetsmodellen på $[0, 1]$. Siden man kun trenger ett tall om gangen, ble det valgt å kun implementere en ligning. Dette er ikke det mest effektive med tanke på kjøretid, siden man kan få to tall om gangen, og det andre tallet kan brukes på neste kjøring (lot være for å ikke skape mer "*overhead*" for EV3en). Det ble valgt ligning 5.5, og implementasjonen av denne ligningen er vist i kodeutdrag 5.16. Her brukes

`math`-modulen for å få tilgang til kvadratrot (`math.sqrt`), den naturlig logaritmen (`math.log`), og cosinus (`math.cos`). Funksjonen ligger i `EV3main.py` og kan brukes fritt av EV3en.

Kode 5.16: `ev3gauss()`-funksjonen

```
26 def ev3gauss():
27     """
28     Generates and returns a number from the
29     standard normal distribution.
30     """
31     x1, x2 = random.uniform(0,1)
32     return math.sqrt(-2*math.log(x1)) * math.cos(2*math.pi*x2)
```

I kodeutdrag 5.17 og kodeutdrag 5.18 vises bruken av `ev3gauss` i `GetFirstMeasurements`-funksjonen og i `GetNewMeasurements`-funksjonen i `EV3main.py`. Hver gang det tas en måling fra lyssensoren, blir det generert et tilfeldig tall fra den standardiserte normalfordelingen og lagt til målingen.

Kode 5.17: `ev3gauss` for første måling i EV3main.py

```
163 # Sensorer
164 light = [myColorSensor.reflection() + ev3gauss()]
```

Kode 5.18: `ev3gauss` for alle nye målinger i EV3main.py

```
187 light.append(myColorSensor.reflection() + ev3gauss())
```

I kodeutdrag 5.19 vises bruken av `ev3gauss`-funksjonen på datamaskinen, det vil si i `BeregnOgPlott.py`.

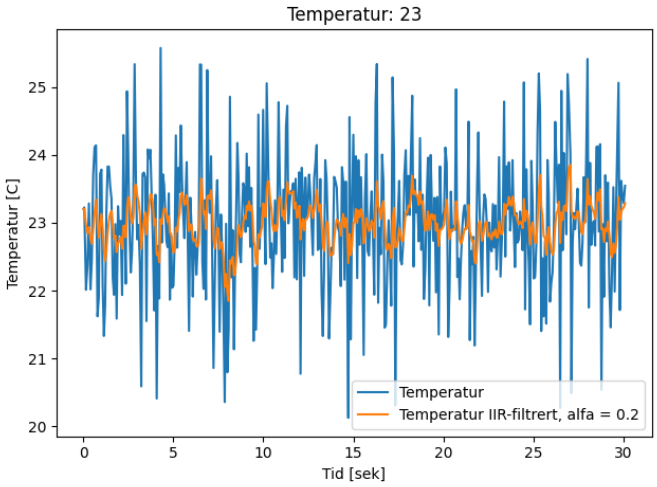
Kode 5.19: BeregnOgPlott.py

```
139 # Append the recieved data to the lists.
140 temp.append(data["light"] + ev3gauss())
```

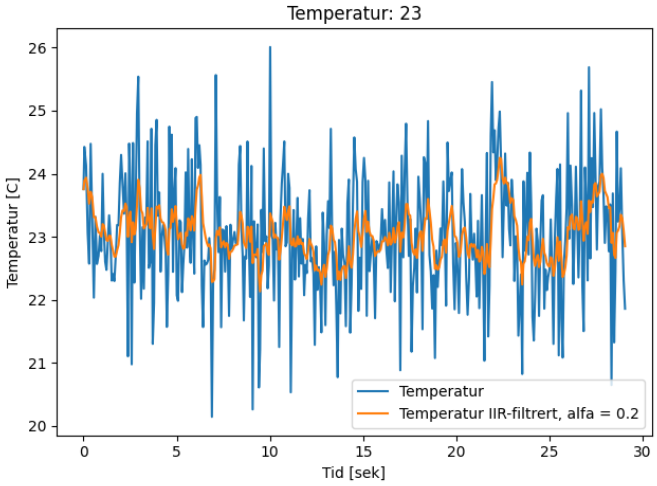
Det er naturlig å tenke seg at at alt som kjøres på en datamaskin går raskere enn på en EV3, og hypotesen vil da være at det gjennomsnittlige tidsskrittet vil være mindre dersom det kjøres med `ev3main` på datamaskinen (legges til i `BeregenOgPlott.py`).

For å simulere situasjonen med en temperaturtransmitter satt inn i ett rør, ble lyssensoren lagt midt på arket i figur 5.4. `EV3main.py` ble gangen kjørt i online-modus, og det samme gjelder for `BeregnOgPlott.py`. I `BeregnOgPlott.py` ble koden for IIR-filtrering i online-modus brukt, denne ble presentert i kodeutdrag 5.12. I `plotData`-funksjonen i `BeregnOgPlott.py` ble også `plt.title(Temperatur: "+ int(temp_iir[-1]))` lagt til for å gi den avrundede temperaturen fra IIR-filtreringen. For å få printe ut det gjennomsnittlige tidsskrittet ble linja `print(sum(ts)/len(ts))` lagt til i `MathCalculations`-funksjonen i `BeregnOgPlott.py`.

Resultatet av kjøring med $\alpha = 0.2$ er vist i figur 5.12 (støy i `EV3main.py` på EV3en) og figur 5.13 (støy i `BeregnOgPlott.py` på EV3en). Her var det veldig lite filtrering, og det var litt variasjon mellom de gjennomsnittlige tidsstegene.

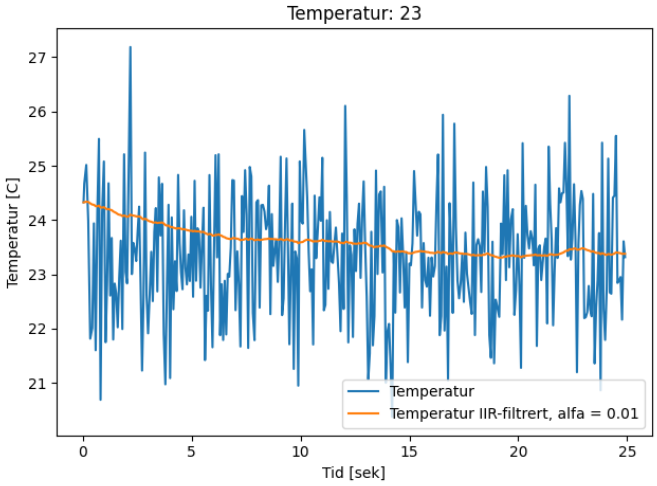


Figur 5.12: Kjøring med støy lagt til av EV3en. Gjennomsnittlig tidssteg var likt 69.84 ms.

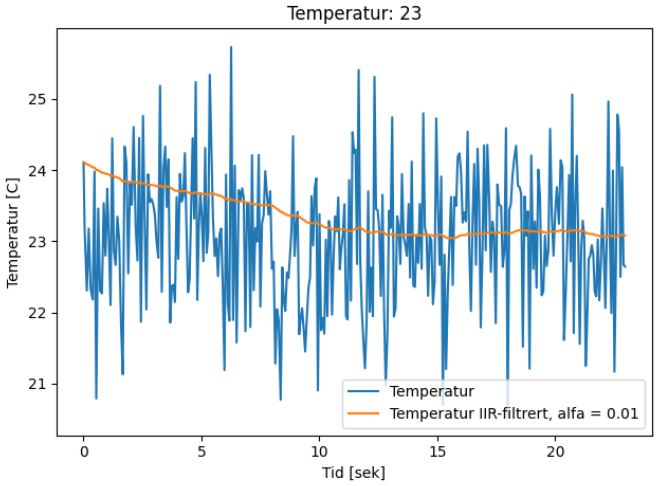


Figur 5.13: Kjøring med støy lagt til av datamaskinen. Gjennomsnittlig tidssteg var likt 72.12 ms.

Resultatet av kjøring med $\alpha = 0.01$ er vist i figur 5.14 (EV3en) og figur 5.15 (datamaskinen). Her var det betydelig mer filtrering, og det var også mindre forskjell i de gjennomsnittlige tidsstegene.

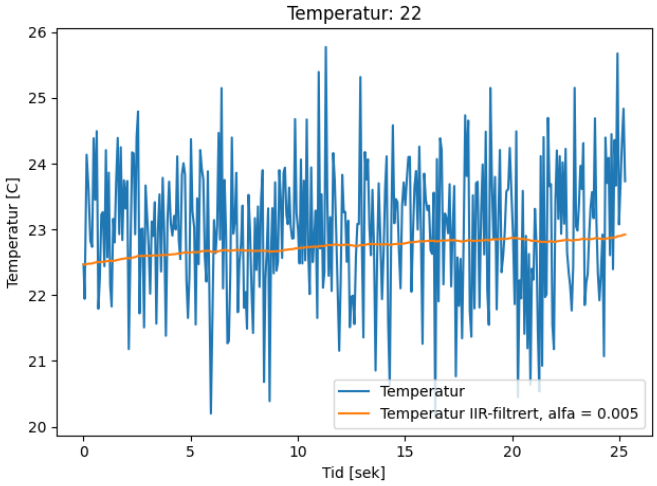


Figur 5.14: Kjøring med støy lagt til av EV3en. Gjennomsnittlig tidssteg var likt 70.12 ms.

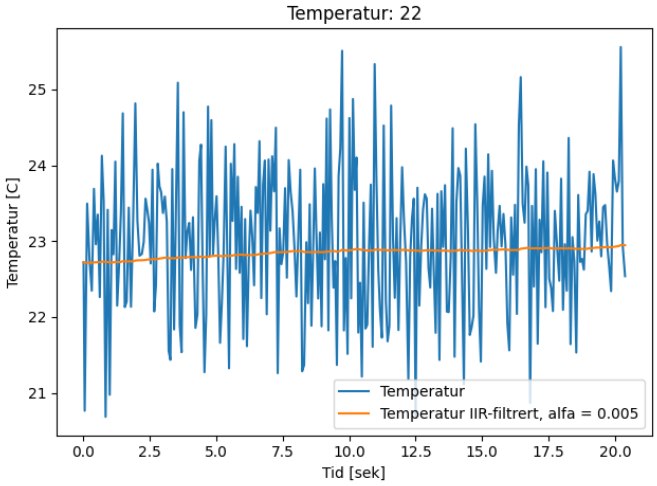


Figur 5.15: Kjøring med støy lagt til av datamaskinen. Gjennomsnittlig tidssteg var likt 69.33 ms.

Resultatet av kjøring med $\alpha = 0.005$ er vist i figur 5.16 (EV3en) og figur 5.17 (datamaskinen). Veldig mye filtrering her, og igjen er variasjonen i de gjennomsnittlige tidsstegene forskjellig fra første kjøring med $\alpha = 0.2$.



Figur 5.16: Kjøring med støy lagt til av EV3en. Gjennomsnittlig tidssteg var likt 70.97 ms.



Figur 5.17: Kjøring med støy lagt til av datamaskinen. Gjennomsnittlig tidssteg var likt 69.92 ms.

Siden det kun var 3 målinger per enhet, var det ikke vanskelig å se at det ikke var de store variasjonene i tidssteg. Den første kjøringa på datamaskinen, med $\alpha = 0.2$ hadde en litt merkbar høyere verdi (72.12 ms) enn de to andre kjøringene på datamaskinen (69.33 og 69.92 ms).

I figur 5.18 vises det resultatet av kjøring av noen statistiske-funksjoner de gjennomsnittlige tidsskrittene ved hjelp av `statistics`-modulen fra Python. Funksjonen som er brukt er `mean` (gjennomsnitt), `median` (median), og `stdev` (standardavvik). `ev3` er ei liste med de gjennomsnittlige tidsstegene for kjøringa på EV3en; tilsvarende for `pc`. Slik resultatene i figur 5.18 viser, er det ikke store forskjeller mellom de to metodene å kjøre `ev3gauss`-funksjonen på. Det eneste som var merkbart er den første kjøringa på datamaskinen som ga et gjennomsnittlig tidssteg på 72.12 ms, og dette første også til at standardavviket for de gjennomsnittlig tidsstegene på datamaskinen var ganske høyt.

```
>>> ev3 = [69.84, 70.12, 70.97]
>>> pc = [72.12, 69.33, 69.92]
>>> from statistics import *
>>> mean(ev3)
70.31
>>> mean(pc)
70.45666666666666
>>> median(ev3)
70.12
>>> median(pc)
69.92
>>> stdev(pc)
1.4703854369971643
>>> stdev(ev3)
0.5884725991921772
>>>
```

Figur 5.18: Statistikk for å undersøke forskjeller mellon kjøring av `ev3gauss`-funksjonen på datamaskin kontra EV3. Som resultatene viser var det ikke store forskjeller mellom de to ulike måtene å legge til `ev3gauss`-funksjonen på.

5.7.2 Filtrering som funksjon

Det ble lagt funksjoner for FIR-filtrering og IIR-filtrering. Siden filtrering kjøres i offline modus, er funksjonene optimalisert for offline-modus. Dette vil si at funksjonene antar at de mottar ei ferdig liste med målinger. Dette er altså i motsetning til online-modus hvor det kontinuerlig kommer inn nye målinger.

I kodeutdrag 5.20 vises funksjonen for FIR-filtrering. Denne funksjonen tar inn ei liste, `maalinger` (som består av målinger fra en sensor), og filtreringsparameteren `m`. Logikken i koden er helt lik koden fra kodeutdrag 5.8. Det lages ei ny liste, `newlist` (linje 5), som tar inn en og en måling fra `maalinger`-lista (linje 8), for å simulere live kjøring. Dette var enkleste måten for å beregne FIR-filtrering på. I tillegg er det en sjekk på linje 9 som

ser om det er færre målinger enn `m`. Siden FIR-filtrering er et gjennomsnitt av de `x` siste målingene, kan det ikke alltids deles på `m` (gitt at `x` $< m$). I slike tilfeller må det deles på faktisk antall målinger, som da er lik $i + 1$ (linje 10).

Kode 5.20: FIR-filtrering som funksjon

```
1  def FIRfilter(maalinger, m):
2      """
3      Funksjon som beregner FIR-filtrering
4      """
5      newlist = []
6      fir_maalinger = []
7      for i in range(len(maalinger)):
8          newlist.append(maalinger[i])
9          if i < m-1:
10             fir_maalinger.append( (1/(1+i)) * ...
                   sum(newlist[-m:]) )
11         else:
12             fir_maalinger.append((1/m) * sum(newlist[-m:]))
13     return fir_maalinger
```

I kodeutdrag 5.21 vises funksjonen for IIR-filtrering. Denne funksjonen tar også inn ei liste med målinger, `maalinger`, samt en parameter α. Også her er logikken i funksjonen helt likt tidligere arbeid som er gjort (se kodeutdrag 5.9). Siden IIR-filtrering er rekursiv, trengs det er basetilfelle for filtreringen. Da brukes den første målte verdien `maalinger[0]`, som vises på linje 7.

Kode 5.21: IIR-filtrering som funksjon

```
1  def IIRfilter(maalinger, alpha):
2      """
3      Funksjon som beregner IIR-filtrering
4      """
5      # iir_maalinger er rekursiv, legger til første
6      # temperaturmåling som basetilfelle
7      iir_maalinger = [maalinger[0]]
8      for i in range(len(maalinger)):
9          iir_maalinger.append(alpha*maalinger[i] + ...
               (1-alpha)*iir_maalinger[i])
10     return iir_maalinger
```

Kapittel 6

Konklusjon

Hovedmålene for oppgaven, var å lage en prosjektstruktur som kunne brukes for det nye faget, samt å gjennomføre de obligatoriske oppgavene fra ING100. Basisstrukturen for et EV3-prosjekt ved bruk av dictionary var den strukturen som først kom frem, da oppgaven begynte. Denne strukturen var en "direkte oversettelse" av strukturen som er fra ING100 hvor det brukes en fil for hver funksjon. Den hyppige bruken av dictionary i denne basisstrukturen er noe som skyldes av at vi er tredjeårsstudenter og har brukt Python i veldig mange fag opp igjennom data-studiet. Dette føles kanskje ikke naturlig for en førsteårsstudent som kun har hatt 1 semester med Python, og læremålene i det nye faget kan forsvinne i programmeringen. For å gjøre det enklere å forstå og jobbe med basekoden for en uerfaren koder kom vi frem til en ny prosjektstruktur med mindre bruk av dictionary (Basisstrukturen for et EV3-prosjekt ved bruk av lister).

I tillegg til at dictionary brukes mindre i den andre prosjektstrukturen, har denne strukturen også et helt annet oppsett. Det var jo ikke slik at MATLAB-prosjektstrukturen fra ING100 faget måtte følges slavisk, og dermed kom man frem til et oppsett med en fil for hver enhet (datamaskin og EV3). Å jobbe med denne prosjektstrukturen viste seg å være ganske praktisk med tanke på utføringen av de obligatoriske oppgavene og feilsøking underveis. Den gamle prosjektstrukturen hvor funksjonene får sin egen fil er nok mer oversiktlig for ferske studenter, men for oss som er mer erfarne syns vi den strukturen med 2 filer er bedre å jobbe med. Det er ikke

lett å si hvordan denne strukturen vil bli sett på av studentene i det nye faget, siden de ikke er helt ferske (de vil ha hatt 1 semester med Python-programmering), men de er ikke veldig erfarne heller. Det er også mulig å overføre denne nye prosjektstrukturen med 2 filer over til MATLAB igjen, men det er noe en kan vente med for å se hvordan studenter i det nye faget reagerer på de ulike strukturene. Siden man deretter har en fil per enhet, vil det si at man ikke er begrenset av Micropython når det kommer til PlottOgBeregn.py, det vil si at det ikke er noen begrensninger for hvilke moduler man ønsker å bruke på pcen. Da er muligheten åpen til å bruke Feks. Numpy til beregninger på pcen. Dette var ikke mulig når det ble brukt dictionary og flere filer som en struktur.

Sammenlignet med ING100 er gjennomføringen av de obligatoriske oppgavene ganske likt. Generelt vil alle prosjektene bestå av å først klargjøre sensorer og motorer, for så å lage kode for matematiske beregninger, også til slutt plotte data. Dette er uavhengig av om det er Python eller MATLAB som brukes for å gjennomføre et prosjekt. Forskjellen ligger for det meste i syntaksen, altså hvordan en for eksempel legger til data i ei liste (Python) eller array (MATLAB). En annen forskjell var måten å avlese data i et plot, med MatLab sin innebygde plotting kunne man sette inn punkter og avlese data direkte. Mens i matplotlib så er man nødt til å bruke ett tredjeparts-program til å redigere figuren og så legge til punktene. Dette ble ikke så veldig pent, men det ble ikke funnet noen annen måte å løse dette på.

For vår del var det både spennende og kult å ha vært med på å bidra til oppsettet av det nye faget. Vi fikk ikke gjort noe nytt med tanke på frivillige prosjekt slik vi gjorde i ING100, men det var heller ikke så uventet. Det gikk en god del arbeid inn i å komme seg inn i et helt nytt oppsett på EV3en, samt å sette opp en struktur som kan brukes for gjennomføring av de obligatoriske oppgavene. Her var det også mye frem og tilbake med tanke på hvilken struktur en skulle bruke, og hvor mye en skulle bruke dictionary fremfor rene lister. Etter å ha funnet en god struktur var gjennomføringen av de obligatoriske ganske simpelt - selv om det ble nødvendig å endre på et par funksjoner - og vi endte med å komme fram til en Fasitfor de obligatoriske prosjektene, som veileder kan se på mens han skal undervise faget.

Bibliografi

[1] https://se.mathworks.com/help/matlab/ref/double.html. Åpnet: 21/02/2021.

[2] 1. command line and environment — python 3.9.1 documentation. https://docs.python.org/3/using/cmdline.html#envvar-PYTHONPATH. Åpnet: 27/01/2021.

[3] 5. data structures. https://docs.python.org/3/tutorial/datastructures.html#list-comprehensions. Åpnet: 27/01/2021.

[4] 6. modules. https://docs.python.org/3/tutorial/modules.html. Åpnet: 14/01/2021.

[5] Anvendt matematikk og fysikk i robotprogrammering ele130. https://www.uis.no/nb/course/ELE130_1. Åpnet: 29/04/2021.

[6] Box-muller transformation. https://mathworld.wolfram.com/Box-MullerTransformation.html. Åpnet: 23/03/2021.

[7] Data types - matlab & simulink. https://se.mathworks.com/help/matlab/data-types.html. Åpnet: 08/02/2021.

[8] Data types — python 3.9.1 documentation. https://docs.python.org/3/library/datatypes.html. Åpnet: 08/02/2021.

[9] Element-wise power - matlab power .^. https://www.mathworks.com/help/matlab/ref/power.html. Åpnet: 12/01/2021.

[10] Ev3 motors and sensors explained. https://education.lego.com/v3/assets/blt293eea581807678a/blt957cc47836883f28/5f8806f9b74e541ab5584800/ev3_user_guide_no.pdf. Åpnet: 03/02/2021.

[11] ev3dev home. https://www.ev3dev.org/. Åpnet: 24/03/2021.

[12] Function basics. https://se.mathworks.com/help/matlab/function-basics.html. Åpnet: 18/01/2021.

[13] Guido van rossum tweet on the usage of multi-line strings as multi-line comments in python. https://twitter.com/gvanrossum/status/1126706055050772 48. Åpnet: 27/01/2021.

[14] How to execute python code from within visual studio code. https://stackoverflow.com/questions/29987840/how-to-execute-python-code-from-within-visual-studio-code/38995516. Åpnet: 09/02/2021.

[15] Ingeniørfaglig innføringsemne - data og elektro ing100. https://www.uis.no/nb/course/ING100_1. Åpnet: 29/04/2021.

[16] Matlab - mathworks - matlab & simulink. https://www.mathworks.com/products/matlab.html. Åpnet: 14/01/2021.

[17] Matlab vs. python: Top reasons to choose matlab. https://www.mathworks.com/products/matlab/matlab-vs-python.html. Åpnet: 14/01/2021.

[18] Micropython. https://micropython.org/. Åpnet: 09/02/2021.

[19] The micropython project. https://github.com/micropython/micropython. Åpnet: 24/03/2021.

[20] Normally distributed random numbers - matlab randn. https://www.mathworks.com/help/matlab/ref/randn.html. Åpnet: 23/03/2021.

[21] numpy home. https://numpy.org/. Åpnet: 7/04/2021.

[22] Pep 8 – style guide for python code. https://www.python.org/dev/peps/pep-0008/#block-comments. Åpnet: 27/01/2021.

[23] Port 8070 (tcp/udp). https://www.speedguide.net/port.php?port=8070. Åpnet: 12/03/2021.

[24] Rtd temperature transmitter t1000. https://www.directindustry.com/prod/golden-mountain-enterprise/product-33181-465897.html. Åpnet: 22/03/2021.

[25] socket — low-level networking interface. https://docs.python.org/3/library/socket.html. Åpnet: 12/03/2021.

[26] Syntax highlight guide. https://code.visualstudio.com/api/language-extensions/syntax-highlight-guide. Åpnet: 08/02/2021.

[27] Top ide index. https://pypl.github.io/IDE.html. Åpnet: 08/02/2021.

[28] Tutorials — matplotlib 3.3.3 documentation. https://matplotlib.org/tutorials/index.html. Åpnet: 11/02/2021.

[29] Usage guide — matplotlib 3.3.3 documentation. https://matplotlib.org/tutorials/introductory/usage.html#sphx-glr-tutorials-introductory-usage-py. Åpnet: 11/02/2021.

[30] What is a 'workspace' in visual studio code? https://stackoverflow.com/a/57134632. Åpnet: 11/02/2021.

[31] What is the matlab search path? https://se.mathworks.com/help/matlab/matlab_env/what-is-the-matlab-search-path.html. Åpnet: 18/01/2021.

[32] A. Vanhoucke. How to connect a ps4 dualshock 4 controller to your mindstorms ev3 brick with bluetooth. https://antonsmindstorms.com/2020/02/14/how-to-connect-a-ps4-dualshock-4-controller-to-your-mindstorms-ev3-brick-with-bluetooth/. Åpnet: 10/02/2021.

Figurer

Listings

Vedlegg A

Noen forskjeller mellom MATLAB og Python

Dette vedlegget er med i rapporten for å være til hjelp for studenter i ELE130. Fra høsten 2021 skal nye studenter innenfor elektrolinjene ha ett semester med pythonprogrammering før de lærer om MATLAB. Dette vil da være i motsetning til det gamle oppsettet hvor nye studenter begynte med MATLAB i første semester. Vedlegget kan brukes som et oppslagsverk hvor studenter raskt kan sammenligne kode fra Python med kode fra MATLAB. I vedlegget vil det også bli supplementert lenker som studenter kan anvende for videre lesing.

A.1 Generelle forskjeller mellom MATLAB og Python

Python er et "generelt" programmeringsspråk. Med det mener vi at det kan brukes til mange ulike formål. For eksempel kan Python brukes til å lage spill, håndtere store datamengder, nettverksprogrammering osv. MATLAB er også et programmeringsspråk, men det er mer spesialisert. På hjemmesiden til MathWorks[16], som er selskapet som eier MATLAB, skrives det blant annet at MATLAB er en kombinasjon av et skrivebordsmiljø (Eng: "desktop enviroment") og et programmeringsspråk spesielt egnet for vitenskapsfolk og ingeniører. Begge språkene anvender såkalte kommandotolkere. Kommandotolkere er program som kjører og utfører linjer med kode, en etter en. På siden til MathWorks finner en også argumenter for å bruke MATLAB over Python.[17]

Her vil vi gjengi noen konkrete forskjeller mellom disse to programmeringsspråkene som er relevante for oppgaven.

A.2 Kommentarer

Kommentarer blir ignorert av kommandotolkerene og legges til for leserens skyld. Kommentar på en linje:

Kode A.1: MATLAB

```
% Dette er en kommentar
a = 3; % kommentar
```

Kode A.2: Python

```
# Enda en kommentar
b = 73; # kommentar
```

Kommentar som består av flere linjer:

Kode A.3: MATLAB

```
%{
Denne kommentaren
bruker mer
enn 1 linje
%}
```

Kode A.4: Python

```
"""
En lengre kommentar
Kanskje forklarer
den hva en
funksjon brukes til
```

```
6 """
```

Strengt tatt er Python-versjonen ikke en offisiell flerlinjet kommentar. Det er en flerlinjet streng som kan brukes som kommentar. Python sin offisielle stilguide [22] sier at flerlinjete kommentarer heller skal være slik:

```
1 #Python sin offisielle stilguide
2 #har sagt at flerlinjete kommentarer
3 #skal se ut som
4 #dette
```

Guido van Rossum, grunnleggeren av Python, har tweetet at flerlinjete strenger kan brukes som kommentarer, så lenge det ikke er i "docstrings".[13] "Docstrings" er strenger som kommer rett etter en funksjonsdefinisjon, brukt for å forklare hva funksjonen gjør.

A.3 Datatyper

Det er betydelige forskjeller i datatypene mellom disse to språkene, og fullstendig dokumentasjon finnes på de respektive sidene til språkene.[7][8] Det vil derimot bli gjengitt noen forskjeller her i denne seksjonen.

A.3.1 Grunnleggende datatyper

Tall

Python har heltall (`int`), desimaltall (`float`), og komplekse tall (`complex`). Disse er illustrert i kodeutdrag A.5 hvor det blir definert en variabel av hver type. For å bekrefte at disse er av rett type blir `type`-funksjonen brukt. Denne funksjonen tar inn ett argument og returnerer typen til argumentet.

Kode A.5: Python

```
1 heltall = 5
2 desimaltall = 3.1415
3 kompleks = 5 + 3j
4
5 print(type(heltall)) # gir <class 'int'>
6 print(type(desimaltall)) # gir <class 'float'>
7 print(type(kompleks)) # gir <class 'complex'>
```

MATLAB har veldig mange ulike numeriske typer. Alt i MATLAB er såkalt "arrays", i ulike dimensjoner. Tall vil da være en "array" med 1x1 dimensjon. Standarden er at alle tall blir lagret som "double-precision floating point", ofte bare kalt "double". Verdiområdet for et slikt tall er avhengig av om det er et negativt eller positivt tall. For negative tall er verdiområdet likt $-1.797\,69 \times 10^{-308}$ til $-2.225\,07 \times 10^{308}$. For positive tall er verdiområdet likt $2.225\,07 \times 10^{-308}$ til $1.797\,69 \times 10^{308}$ [1].

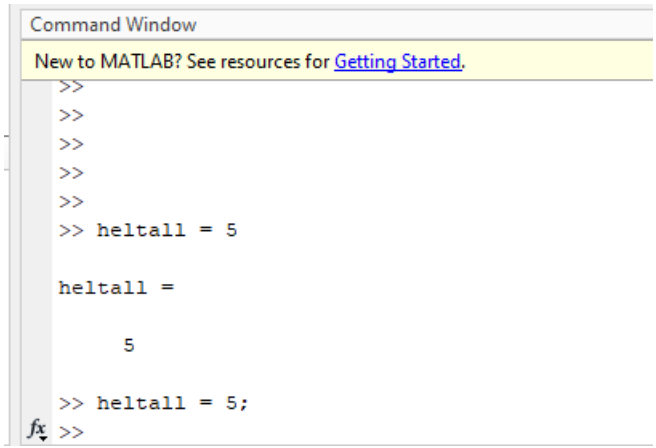
I kodeutdrag A.6 blir det vist en deklarasjon av to tall. Her blir `heltall` satt lik 5, mens `desimaltall` blir satt lik 3.1415. I Python vil en slik variabeldeklarasjon føre til at `heltall` og `desimaltall` får ulike typer. Dette ble vist i linje 5 og 6 i kodeutdrag A.5. I MATLAB vil begge disse verdiene være lik "double". For å vise dette brukes `class`-funksjonen til MATLAB, som ligner på Python sin `type`-funksjon.

Kode A.6: MATLAB

```
1 heltall = 5;
2 desimaltall = 3.1415;
3
4 class(heltall) % gir ans = 'double'
5 class(desimaltall) % gir ans = 'double'
```

Legg merke til bruken av ";" (semikolon) i linje 1 og 2 i kodeutdrag A.6. Ved å legge til et semikolon på slutten av ei linje, vil det ikke bli skrivet noe *output*. Dette er illustrert i figur A.1. Første kommando som ble skrevet inn er `heltall = 5`. Dette ga output i kommandotolkeren (vist under selve kommandoen). Neste kommando er `heltall = 5;`. Dette gir ingen output.

Semikolonen i MATLAB har også en alternativ bruk for vektorer og matriser. Dette blir tatt opp i seksjon A.3.2.



Figur A.1: Output fra kommandotolkeren til MATLAB.

Mer om numeriske typer i MATLAB kan leses på https://se.mathworks.com/ (klikkbar lenke). På den siden blir det også vist alle de ulike numeriske typene i MATLAB.

Strenger

Strenger er veldig like i begge språkene, som vist i kodeutdrag A.7 og A.8. I Python er det fritt frem for å bruke enten dobbelt anførselstegn (linje 1 i kodeutdrag A.8) eller enkelt anførselstegn (linje 2 i kodeutdrag A.8). Det viktigste er konsistens, altså at i kildekoden så brukes kun en type.

I MATLAB differensieres det mellom enkelt anførselstegn og dobbelt anførselstegn. Dobbelt anførselstegn brukes for å definere en faktisk streng. Dette er ekvivalent med strenger i Python. Enkelt anførselstegn brukes for å lage en 1xN "array" (se på det som en 1xN vektor) med karakterer. N er da lik lengden på strengen. Dette er vist i linje 2 i kodeutdrag A.7.

Kode A.7: MATLAB

```
1 streng = "Dette er en streng";
2 karakterarray = 'Dette er ikke det samme som en streng!';
```

Kode A.8: Python

```
1 streng1 = "Dette er en streng"
2 streng2 = 'Dette er en annen streng'
```

A.3.2 Vektorer og matriser

I Python finnes det ikke vektorer eller matriser. Det er mulig å importere moduler (mer om moduler blir tatt opp i seksjon A.5) som muliggjør slik funksjon. For de fleste formål holder det med `list` datatypen. Dette er en datatype som lagrer flere verdier sammen. Disse verdiene kan bestå av andre grunnleggende datatyper, for eksempel tall, eller strenger. Ei pythonliste kan altså brukes til som en vektor ved å kun legge til tall. Dette er vist i kodeutdrag A.9 hvor det lages ei liste, `vektor`, som inneholder tallene `1,2,4,8,16,32,64,128,256`. Deretter vises det hvordan de ulike elementene i lista kan printes. I linje 7 blir det brukt `len`-funksjonen som returnerer lengden til lista (9). Siden første element er gitt av `vektor[0]`, vil det være slik at siste element er gitt av `vektor[8]`, slik at det da må brukes `len(vektor)-1`.

Kode A.9: Python

```
1 vektor = [1,2,4,8,16,32,64,128,256]
2 print(vektor[0]) # printer forste element
3 print(vektor[1]) # printer andre element, osv
4
5 print(vektor[-1]) # printer siste element
6 print(vektor[-2]) # printer ant siste element, osv
7
8 print(vektor[8]) # printer siste element
9 print(vektor[len(vektor)-1] # printer siste element
```

I Python er det også mulig å lage en struktur som ligner på matriser. Dette gjøres ved hjelp av nestede lister, som demonstrert i kodeutdrag A.10. Her lages det en 3x5 "matrise". Denne består altså av ei liste som har 3 lister som innhold. Hver liste (inne i hovedlista `matrise`) kan tenkes å representere en rad i matrisen. I kodeutdraget vises det også hvordan det kan refereres til de ulike elementene i matrisen og listene som bygger matrisen.

Kode A.10: Python

```
1 matrise = [[1,2,3,4,5], [6,7,8,9,10], [11,12,13,14,15]]
2 # print forste liste/rad/radvektor
3 print(matrise[0])
4 # print andre liste/rad/radvektor
5 print(matrise[1])
6 # print tredje liste/rad/radvektor
7 print(matrise[2])
8
9 # print forste element i forste radvektor
10 print(matrise[0][0])
11 # print tredje element i siste radvektor
12 print(matrise[2][2])
```

I MATLAB derimot er det som nevnt slik at alle objekter er "arrays". En "array" kan sees på som en generell form for en vektor/matrise. For å for eksempel lage en radvektor kan en bare skrive som i linje 1 i kodeutdrag A.11. En kolonnevektor lages som vist i linje 2. Forskjellen her er bruken av ";". Den brukes for å starte en ny rad. Dette kan anvendes for å lage en matrise, som blir en kombinasjon radvektorer og kolonnevektorer. En matrise lages som vist i linje 3 i samme kodeutdrag.

Kode A.11: MATLAB

```
1 radvektor = [2 4 8 16 32 64 128 256 512 1024];
2 kolonnevektor = [10;20;30;40;50];
3 matrise = [3 6 9; 4 8 12; 5 10 15];
```

I MATLAB er det slik at det første elementet i en array starter med *subscript* lik 1. Dette er i motsetning til Python for første element alltid i ei liste har *subscript* lik 0. For å få ut første element i radvektoren som ble definert i linje 1 i kodeutdrag A.11, må det gjøres som i linje 1 i kodeutdrag A.12. MATLAB støtter også mer avanserte former for indeksering som er vist i samme kodeutdrag.

Kode A.12: MATLAB

```
1 radvektor(1) % gir ans = 2
2 % henter ut de 3 første elementene i vektoren -> 2, 4, og 8
3 radvektor([1 2 3])
4 % henter ut andre til fjerde element -> 4, 8, og 16
5 radvektor(2:4)
```

```
6
7  % indeksering av matrise
8  % henter ut element i rad 2, kolonne 2 ->  8
9  matrise(2,2)
10 % henter ut elementene i rad 2 og kolonne 1 til 2 -> 4, og 8
11 matrise(2,1:2)
12 % henter ut elementene i rad 1 til 2 og kolonne 1 til 2
13 % dette gir 3, 4, 6, og 8
14 matrise(1:2,1:2)
```

MATLAB har også innebygde funksjoner for å raskt fylle ut matriser, som vist i kodeutdrag A.13. Den første linja skaper et såkalt 3x3 magisk kvadrat, som er en matrise med den egenskapen at summen langs de horisontale, vertikale, og diagonale linjene er like. Den andre linja lager en 5x5 matrise med kun enere.

Kode A.13: MATLAB

```
1 magiskmatrise = magic(3);
2 enhetsmatrise = ones(5);
```

Mer om matriser i MATLAB og "arrays" generelt kan leses på https://se.mathworks.com (klikkbar lenke).

A.4 Regneoperasjoner

Denne seksjonen vil ta for seg noen av de mest brukte regneoperasjonene i MATLAB og Python. MATLAB har ganske mange regneoperasjoner, og har blant annet spesielle operasjoner for vektorer og matriser. Dette kan leses om på https://se.mathworks.com (klikkbar lenke).

A.4.1 Grunnleggende aritmetiske operasjon

De grunnleggende aritmetiske operasjonene (addisjon, subtraksjon, multiplikasjon, og divisjon) er ganske like i begge språk. Dette er illustrert i kodeutdrag A.14 og A.15.

Kode A.14: MATLAB

```
1 % definerer variabler
2 a = 3;
3 b = 4;
4 c = 5;
5
6 d = a + b; % gir 7
7 e = c - 3; % gir 2
8 f = a * a; % gir 9
9 g = (c * 16)/4; % gir 20
```

Kode A.15: Python

```
1 a = 3
2 b = 4
3 c = 5
4
5 d = a + b # gir 7
6 e = c - 3 # gir 2
7 f = a * a # gir 9
8 g = (c * 16)/4 # gir 20
```

Legg merke til at Python ikke printer noe til konsollen. For å få Python til å printe noe til konsollen må `print`-funksjonen brukes. MATLAB printer implisitt, dette stoppes ved å bruke semikolon, som vist i kodeutdrag A.14.

A.4.2 Potenser

Potenser i MATLAB og Python kan begge utføres på to måter hver, slik som det er vist i kodeutdrag A.16 og A.17.

Kode A.16: MATLAB

```
1 Z = X.^Y
2 Z = power(X, Y)
```

Kode A.17: Python

```
1 Z = X ** Y
2 Z = pow(X, Y)
```

Det er ingen forskjell mellom linje 1 og 2 i verken kodeutdrag A.16 eller kodeutdrag A.17. Resultatet blir altså det samme. Linje 2 anvender en innebygd funksjon, som finnes både i Python og MATLAB. MATLAB selv sier at det er den første varianten (linje 1 i kodeutdrag A.16) som blir brukt oftest.[9]

A.5 Funksjoner og søkestier

Funksjoner i MATLAB kan ikke defineres i kommandolinja. I MATLAB er funksjoner veldig ofte definert i egne .m-filer (MATLAB filer), og som filnavn brukes funksjonsnavnet. I slike situasjoner må funksjonsfila være i den

kallende .m-filens mappe, eller så må den være i MATLABs søkesti.[31] Fullstendig dokumentasjon om funksjoner i MATLAB kan leses på MathWorks side.[12]

I Python blir funksjoner ofte definert i samme fil som hovedprogrammet (så lenge selve programmet ikke er veldig stor). I andre tilfeller blir funksjoner i Python definert i egne filer, kalt moduler. En modul kan blant annet definere klasser og funksjoner som kan importeres i andre pythonprogram. Dette bidrar til mer lesbar kode siden det bidrar til bedre organisering av kode. Ved å bruke moduler slippes det å legge til samme funksjon i flere ulike programmer. Da kan en heller lage en modul som kan importeres, i en enkel setning, og deretter bruke funksjonen som om den var laget i programmet. Ved å legge til en import-setning for en modul, for eksempel

```
1 import eksempelmodul
```

vil Python se etter denne modulen på ulike plasser. Først og fremst antar Python at dette er en pythonfil med navnet *eksempelmodul.py*. Python vil se etter filen i "hovedmappen", det vil mappen hvor programmet som kjører ligger i. Det kan være slitsomt å hele tiden måtte flytte (eller kopiere) denne eksempelmodulfila (hvis den har funksjoner som brukes ofte) til hovedmappene til programmene du kjører. Det finnes det en løsning i Python, som ligner den for MATLAB, nemlig **PYTHONPATH**. Dette er en såkalt "environment variable". "Environment variables" er variabler (for eksempel PYTHONPATH) som kan være til hjelp for ulike prosesser (i dette tilfellet kommandotolkeren til Python). Ved å legge til mappestier i PYTHONPATH variabelen, vil kommandotolkeren lete i disse mappene når den ser en import-setning i programmet. Dermed er det mulig å ha en eller flere mapper hvor du legger til mange moduler med hjelpefunksjoner som du kan bruke om igjen i ulike programmer. Dokumentasjonen til PYTHONPATH finnes på Pythons hjemmeside.[2]

Her vil vi gi to eksempler på funksjoner i både MATLAB og Python. Den første, vist i kodeutdrag A.18 og A.19, er en funksjon som tar inn to argumenter og returnerer enn verdi. Deretter vises det en funksjon i kodeutdrag A.24 og A.25 som tar inn ett argument og returnerer to verdier.

I kodeutdrag A.18 og A.19 er det presentert en funksjon med navn `arealtrekant`

som regner ut arealet til en trekant. Kodeutdrag A.18 viser funksjonen i MATLAB, kodeutdrag A.19 viser den samme funksjonen i Python. Funksjonen tar inn to argumenter `side`, og `hoyde`. Den har en returverdi; `areal`.

Kode A.18: MATLAB

```
1 function areal = arealtrekant(side, hoyde)
2     areal = (side*hoyde)/2;
3 end
```

Kode A.19: Python

```
1 def arealtrekant(side, hoyde):
2     return (side * hoyde)/2
```

I MATLAB må det altså brukes nøkkelordet `function` samt navnet på returverdien (her like `areal`) til funksjonen. Legg merke til at funksjonsdefinisjonen slutter med `end`, i motsetning til Python. Dermed defineres hva returverdien faktisk er, også slutter hele funksjonen med nøkkelordet `end`.

For å kalle funksjonen:

Kode A.20: MATLAB

```
1 s = 3;
2 h = 4;
3 a = arealtrekant(s, h);
```

Kode A.21: Python

```
1 s = 3
2 h = 4
3 a = arealtrekant(s, h)
```

Kode A.22: MATLAB

```
1 a = arealtrekant(6, 8)
```

Kode A.23: Python

```
1 a = arealtrekant(6, 8)
```

Det er altså mulig å kalle funksjonene både via variabler eller direkte via tall.

Funksjonen i kodelisting A.24 og A.25 tar inn en mengde med tall, og returnerer gjennomsnittet til tallene. Legg merke til at denne funksjonen bruker innebygde funksjoner; **sum**, **numel** ("number of elements"), og **median** i MATLAB eksempelet. Dette er funksjoner som allerede er predefinert

av MATLAB og kan brukes fritt. En kunne også ha brukt den innebygde funksjonen **mean** funksjonen for å finne gjennomsnittet, istedenfor **sum** og **length**, med det samme resultatet. I Python bruker vi også de innebygde funksjonene **sum**, og **len**. Det finnes ikke en innebygd funksjon for medianen i Python, men den finnes i `statistics`-modulen, så den brukes her.

Kode A.24: MATLAB

```
function [snitt,median] = statistikk(vek)
    snitt = sum(x)/numel(x);
    median = median(x);
end
```

Kode A.25: Python

```
import statistics

def statistikk(vek):
    snitt = sum(vek)/len(vek)
    median = statistics.median(vek)
    return snitt, median
```

For å kalle funksjonen:

Kode A.26: MATLAB

```
% vek er en vektor med alle tallene fra 1 til 500
vek = 1:500;
[avg, med] = statistikk(vek)
```

Kode A.27: Python

```
# vek er ei liste med alle tallene fra 1 til 500
vek = [i for i in range(1,501)]
avg, med = statistikk(vek)
```

I MATLAB-eksempelet lages det en vektor **vek** med alle tallene fra 1 til 500. I Python finnes det ikke vektorer (så lenge de ikke importeres fra en modul), så her brukes det lister. Ved hjelp av "list comprehensions" er det

mulig å definere lister av tall med visse egenskaper på en enkel måte. Uten "list comprehensions" måtte en ha fylt ut listen eksplisitt, for eksempel slik:

```
1 vek = []
2 for i in range(1, 501):
3     vek.append(i)
```

Mer om "list comprehensions" kan leses i pythondokumentasjonen.[3]

Vedlegg B

En kort innføring til Visual Studio Code for Windows og macOS

Dette vedlegget gir en kort innføring til Visual Studio Code. Fokuset vil være på praktisk bruk av Visual Studio Code for å skrive kildekode i Python. Alt innholdet i vedlegget ble utført på en datamaskin som kjører Windows, men det som vises vil også fungere på en datamaskin som kjører macOS.

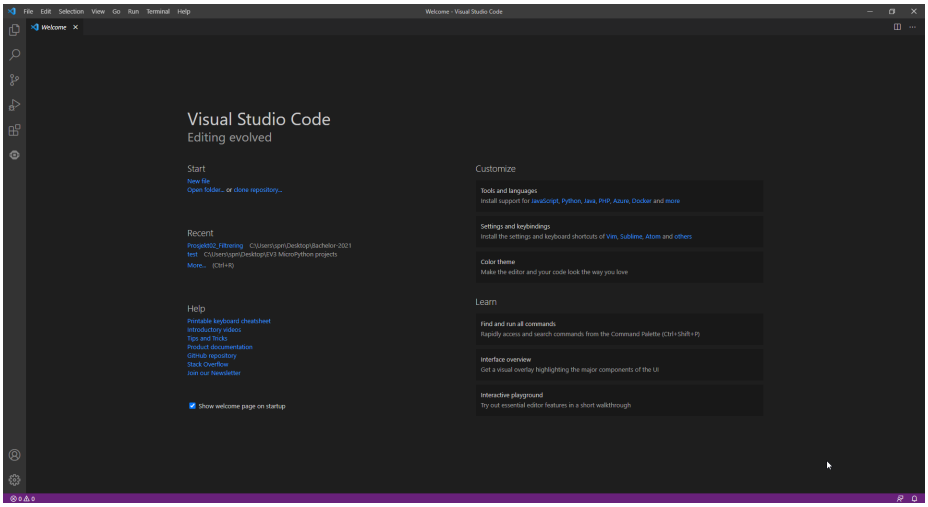
B.1 Visual Studio Code, kildekode-editorer og IDE-er

Visual Studio Code er en såkalt kildekode-editor, som er program laget for å redigere på kode. pyCharm er et eksempel på et såkalt integrert utviklingsmiljø (IDE på engelsk, som står for "Integrated Development Environment"). Kildekode-editorer og andre IDEer tilbyr verktøy og funksjoner som gjør det lettere å programmere. Slike funksjoner kan være alt fra sjekking av syntaks (før kommandotolkeren til f.eks. Python oppdager feilen) til en fin oversikt over prosjektet som en arbeider med. Det går fint ann å programmere uten bruk av disse programmene, men det vil bare vanskeliggjøre visse oppgaver (f.eks. å kompilere og kjøre et program i et kompilert programmeringsspråk). Visual Studio Code tilbyr mange utvidelser som kan lastes ned og kan være til stor hjelp. Dette kan være utvidelser som hjelper med alt fra assemblyprogrammering til utseende på selve Visual Studio Code.

Forskjellen mellom en kildekode-editor og IDE kan være litt uklar til tider. Kort sagt er det slik at IDEen som oftest har mange flere verktøy tilgjengelig. Mens en kildekode-editor kan sees på som en mye mer avansert form for Notepad, vil en IDE integrere flere andre programmer (kompilator, kommandotolker, avlusingsverktøy osv.). Det finnes mange eksempeler på integrerte utviklingsmiljø, som f.eks. PyCharm, IntelliJ, og Eclipse. Ved å analysere hvor ofte en IDEs nedlastingsside blir søkt opp på Google, er det kommet frem til at Visual Studio Code ligger på fjerdeplass.[27] Dette gjelder da for året 2021. En ser også at det altså går fort å blande mellom en kildekode-editor og en IDE. pyCharm som er kjent fra pythonfaget ligger på femteplass, mens Visual Studio er den desisert mest populære IDEen. Visual Studio er et annet program og må ikke forveksles med Visual Studio Code.

Visual Studio Code kan lastes ned på https://code.visualstudio.com/. Installasjonen er rett frem; for det meste er det bare å klikke på "Next", bortsett fra på slutten hvor en klikker på "Install" også "Finish" når installasjonen er ferdig.

B.1 Visual Studio Code, kildekode-editorer og IDEer

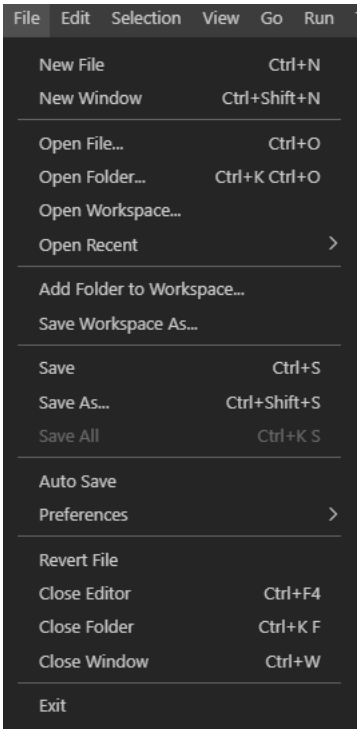


Figur B.1: Grensesnittet til Visual Studio Code.

I figur B.1 kan en se hvordan Visual Studio Codes velkomstskjerm ser ut.

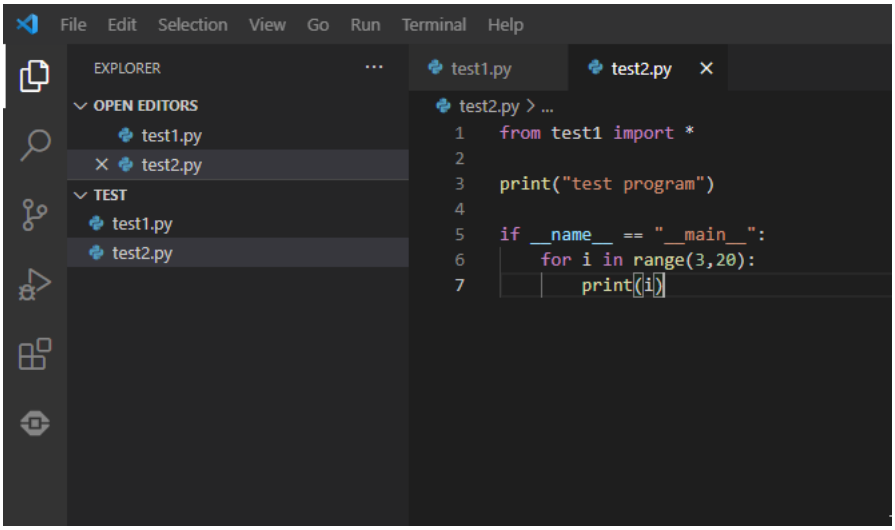
B.2 Praktisk bruk av Visual Studio Code

I øvre del venstre hjørne kan en klikke på "File", og her vil en få mange alternativ.



Figur B.2: Alternativene gitt under "File".

Ved å for eksempel klikke på "New File" (Ctrl+N) vil en da selvsagt få opp en ny fil som en kan redigere. Ved å klikke på "Save" (Ctrl+S) kan en lagre filen. Ved å klikke på "Open Folder" (Ctrl+K+O) kan en åpne ulike mapper, hvor det for eksempel kan finnes mange kodefiler som hører under et prosjekt. Et eksempel er vist i figur B.3.



Figur B.3: Et eksempelprosjekt i Visual Studio Code.

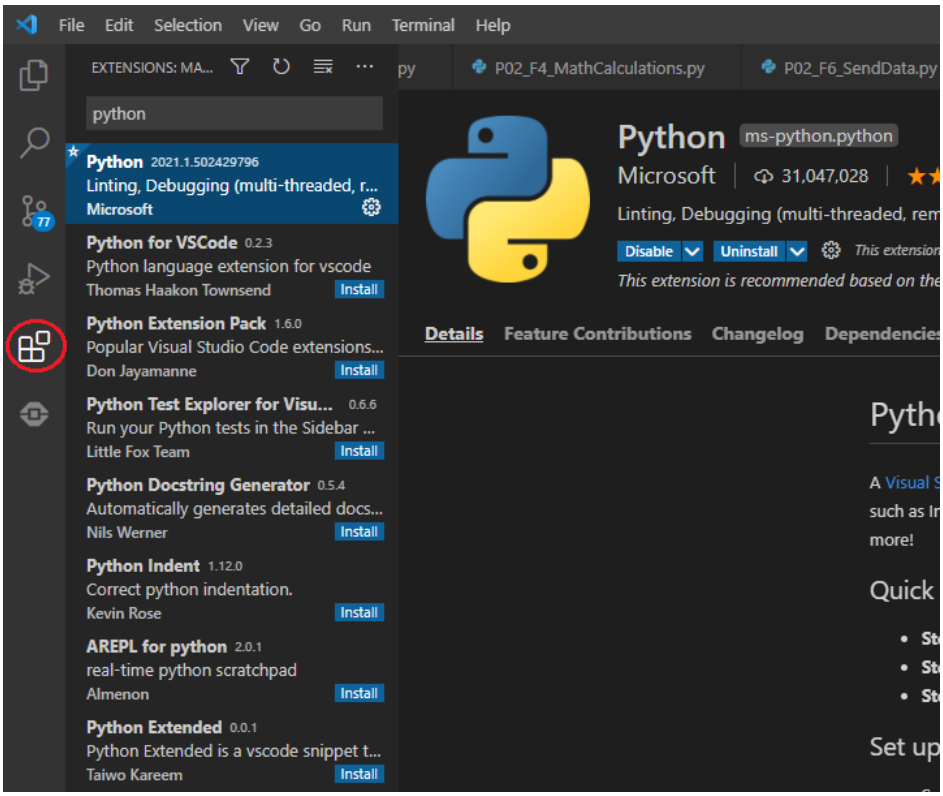
I figur B.3 ser en også noe av syntaksuthevingen til Visual Studio Code for Python. Mer om det kan leses på hjemmesiden til Visual Studio Code.[26]

I Visual Studio Code blir benevningen *workspace* ofte brukt. En norsk oversetning for dette vil være arbeidsområde, og dette betegner altså filene og mappene som inngår i et prosjekt. Veldig ofte vil det være slik at arbeidsområdet kun består av en mappe og alle filene som inngår i den. Derfor er det ganske vanlig å snakke om "folder" (mappe) når en snakker om "workspace" (arbeidsområde), og omvendt. Dette må ikke forveksles med *workspace* slik det brukes i MATLAB.

På stackoverflow har brukeren "jabacchetta" skrevet om hva et arbeidsområde i Visual Studio Code kan brukes til [30]. Blant annet kan arbeidsområdet ha spesifikke innstillinger (som kun gjelder for det aktuelle arbeidsområdet) og anbefalte utvidelser (mer om dette i neste seksjon). Utdypende informasjon om arbeidsområdet i Visual Studio Code kan leses på code.visualstudio.com (klikkbar lenke).

B.3 Utvidelser

Som nevnt tilbyr Visual Studio Code utvidelser slik at du som bruker kan (blant annet) legge til nye verktøy for å hjelpe deg til kodeskriving. Disse utvidelsene finnes på det som kalles for "Visual Studio Marketplace". Ved å taste inn Ctrl+Shift+X på tastaturet vil dette komme opp. Eventuelt kan en klikke på ikonet indikert med den røde sirkelen i figur B.4.



Figur B.4: Markedsplassen for utvidelser i Visual Studio Code. I bildet vises "Python"-utvidelsen. Den røde sirkelen viser ikonet for markedplassen.

Det finnes veldig mange utvidelser, for mange ulike formål. I figuren ovenfor er det blant annet vist en utvidelse for Python med navnet "Python". Dette er definitivt ikke den eneste utvidelsen for Python, som indikert i figur B.4. I beskrivelsen for utvidelsen "Python" står det blant annet *"Linting, Debugging (multi-threaded, remote), Intellisense, Jupyter Notebooks, code*

formatting, refactoring, unit tests, and more.". På denne PCen, hvor det er blitt laget og redigert mange pythonfiler, står det også *"This extension is recommended based on the files you recently opened."*. Den blå "Uninstall" knappen indikerer at utvidelsen er installert. Hvis den ikke var det, ville det stått en blå "Install" knapp istedenfor. Installasjonen er altså rett frem; klikk på den blå "Install" knappen så vil Visual Studio Code ta seg av resten.

B.4 Kjøring av Python program i Visual Studio Code

I utganspunktet er ikke Visual Studio Code konfigurert for å kjøre pythonfiler (eller filer fra andre programmeringsspråk). Visual Studio Code er som sagt en kildekode-editor og det er ikke innebygd støtte til kjøring av filer direkte i programmet. Det betyr ikke at det er umulig. Det finnes ulike måter å konfigurere Visual Studio Code til å kjøre Python på. En enkel og grei måte å gjøre det på er ved å installere utvidelsen "Code Runner". For å kjøre et program med Code Runner, er det bare å trykke på knappen indikert med rød sirkel i figur B.5. Denne knappen vil alltid være tilgjengelig ovenfor til høyre for en fremhevet fil. Eventuelt kan tastatursnarveien Ctrl+Alt+N brukes.

Figur B.5: Code Runner utvidelsen i Visual Studio Code. Code Runner støtter veldig mange ulike språk.

Det finnes andre måter som kan brukes om det er slik at Code Runner ikke fungerer, dette er beskrevet på blant annet stackexchange.[14]

Det finnes også, blant annet, en utvidelse med navnet "Matlab Code Run" for å kunne kjøre matlabkode i Visual Studio Code. I dette prosjektet er det hovedsakelig MicroPython og Code Runner utvidelsene som blir brukt. Oppsettet av MicroPython står det om i kapittel 2.2.

Vedlegg C

En kort innføring til socketer og socketprogrammering

Hensikten med dette vedlegget er å gi kort teoretisk bakgrunn til nettverkskoden som brukes på EV3en. Bruken av socketer viste seg å være nødvendig for å kunne kommunisere mellom datamaskinen og EV3en live. Dette ble brukt for å kunne kjøre live plotting av data mens EV3en kjørte. Skrivet går ikke dypt inn på socketprogrammering, men vil forklare noe av hensikten bak det og hvordan det kan brukes i Python.

C.1 Socketer

Socketer brukes for å muliggjør kommunikasjon mellom to "endepunkter". Med endepunkter menes en kombinasjon av en ip adresse + et portnummer. Ip adressen spesifiserer endepunktets nettverk, mens portnummeret spesifiserer programmet (prosessen) til endepunktet. Socketer er altså spesifikke til en prosess. For eksempel betyr `192.168.1.1:10` at vi har med portnummer `10` å gjøre på `192.168.1.1` nettverket. Det er en socket for hvert endepunkt. Dermed kreves det ett par med socketer for å muliggjør kommunikasjon mellom to endepunkter.

Typisk jobber vi med en såkalt klient-tjener modell, hvor det er en eller flere klienter som etterspør tjenester fra en tjener (Eng: *server*). For eksempel er nettlesere programvareklienter. De etterspør tjenester fra nettsteder, som kjøres på tjenere, eller servere. Servere venter altså på forespørsler fra klienter. Disse serverene blir kalt for webservere. Serveren setter opp en socket (på et spesifikt portnummer) som hører etter for tilkoblinger fra klienter. En klient setter også opp en socket (på et spesifikt portnummer), som da skal kommunisere med serverens socket. Når serveren mottar koblingen fra klienten, vil det dannes

For eksempel er det mulig å nå nettsiden til `duckduckgo` ved å skrive `52.142.124.215:80` inn i en nettleser. På port `80` på webserveren til `duckduckgo` er det altså en socket som venter på tilkoblinger fra klienter rundt omkring i verden.

Denne klient-tjener modellen blir brukt for kommunikasjon mellom EV3en og datamaskinen. EV3en fungerer som server, mens datamaskinen fungerer som klient. Når `EV3main.py` blir kjørt i online-modus på EV3en, blir det satt opp en socket som venter på tilkoblinger fra datamaskinen. Når `BeregnOgPlott.py` kjører på datamaskinen, blir det satt opp en socket som prøver å koble seg til EV3en. Hvis alt går bra, vil det da sendes data fra EV3en til datamaskinen via denne socketforbindelsen.

C.2 Socketprogrammering i Python

Socketprogrammering er altså socketer satt i praksis. I denne seksjonen skal det vises hvordan socketobjekter kan lages både for en server og en klient i Python. Dette gjøres ved hjelp av `socket`-modulen. Fullstendig dokumentasjon til denne modulen i Python-dokumentasjonen.[25]

Oppsettet for en server er vist i kodeutdrag C.1. Dette oppsettet brukes for EV3en i `EV3main.py`.

På linje 1 i kodeutdrag C.1 lages det er socketobjekt med navn `sock`. Det blir gitt to parametre til socket-konstruktøren; `socket.AF_INET` og `socket.SOCK_STREAM`. `socket.AF_INET` indikerer at familien med adresser som socketobjektet kan kommunisere med er av IPv4 typen. `socket.AF_INET6` er for IPv6, for eksempel. Til de fleste formål brukes `socket.AF_INET`. `socket.SOCK_STREAM` parameteren indikerer at det er TCP protokollen som skal brukes. Dette imotsetning til UDP protokollen (`SOCK_DGRAM`). Det er en god del forskjeller mellom disse to protokollene, men det viktigste er at TCP er en såkalt tilkoblingsorientert protokoll, mens UDP ikke er det. I praksis betyr det at TCP garanterer (så langt det lar seg gjøre) at ting som sendes fra avsender når mottakeren. UDP gjør ikke det. UDP sender data avgårde uten å garantere for at de faktisk når mottaker.

Når en socket avsluttes vil den gå i det som kalles for en `TIME_WAIT` tilstand. Da vil porten som socketen brukte være opptatt. Ved å starte en ny socket på samme port kan det oppstå feil (for eksempel ved å kjøre samme kodesnutt på nytt). `SO_REUSEADDR` gjør det mulig å opprette et socketobjekt på samme port som ble brukt tidligere, uten å måtte vente på at `TIME_WAIT` tilstanden slutter. Dette blir spesifisert på linje 2. For EV3en sin del, er dette nyttig for å kunne kjøre EV3en flere ganger i løpet av et kort intervall. For å kunne anvende `SO_REUSEADDR` må det i tillegg gis `socket.SOL_SOCKET` og `1` som parametre.

På linje 3 blir socketobjektet bundet til adressen `(, 8070)`. '' representerer det som kalles for `INADDR_ANY` som brukes for å indikerte at socketen kan motta koblinger fra alle slags nettverkskort. `8070` er portnummeret som socketen bindes til. Port 8070 er en port som egner seg for TCP kommunikasjon.[23]

På linje 4 blir `socket.listen()`-funksjonen brukt med `1` som parameter. Dette gjør at serveren aktivt hører etter for tilkoblinger fra andre socketer. Parameteren `1` indikerer at det er kun en tilkobling som skal aksepteres.

På linje 6 blir en tilkobling akseptert. `socket.accept()`-funksjonen er blokkerende, det vil si at programmet stopper opp ved dette punktet. Den vil da ikke gå videre til de neste linjene før det er mottatt en kobling. `socket.accept()`-funksjonen returnerer et nytt socketobject, som da skal brukes for å senda og motta data fra serveren. Funksjonen returnerer også adressen til socketobjektet til klienten som har opprettet en kobling til serveren. Her blir adressen satt like "`_`", for å indikere at den ikke blir brukt. Dette blir brukt i programmeringsspråket Golang, blant annet. På linje 7 sendes det en `ack`-melding (*acknowledgment*) tilbake til klienten etter at alt er satt opp og klart.

Kode C.1: Oppsett av socketobjekt på en server i Python

```
1  sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
2  sock.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
3  sock.bind(("", 8070))
4  sock.listen(1)
5
6  connection, _ = sock.accept()
7  connection.send(b"ack")
```

Oppsettet av en socketklient er ganske likt og er vist i kodeutdrag C.2. På linje 1 blir det satt opp et socketobjekt. På linje 2 blir adressen til serveren definert. Her brukes `EV3_IP` som da er lik IP adressen til EV3en. Porten som EV3en bruker for sitt socketobjekt er lik 8070. På linje 3 blir det prøvd å oppnå en forbindelse, og på linje 4 blir det mottatt data. For `socket.recv()`-funksjonen blir det spesifisert at det skal mottas 1024 byter med data om gangen. Det første som sendes fra serveren, som vist på linje 7 i kodeutdrag C.1 er en `ack`-melding i byte-format. Dette blir testet på linje 5 i kodeutdrag C.2. Hvis det ble mottatt noe annet vil programmet avslutte, som vist på linje 9.

Kode C.2: Oppsett av socketobjekt på en klient i Python

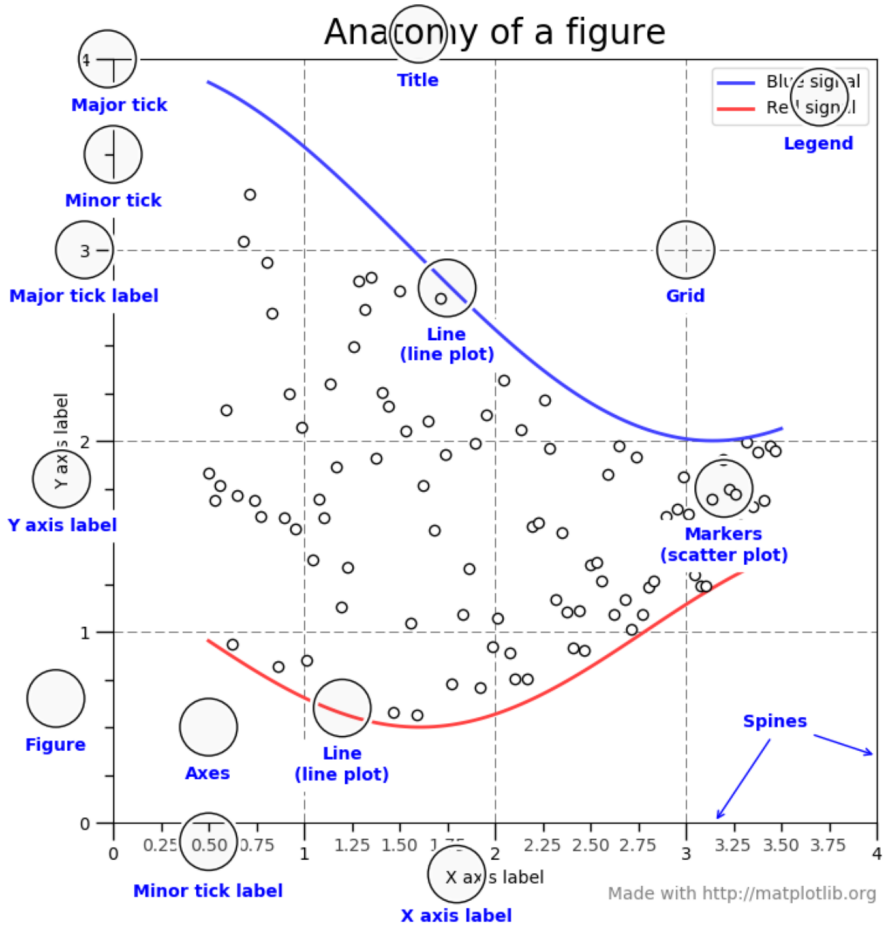
```
1  sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
2  addr = (EV3_IP, 8070)
3  sock.connect(addr)
```

```
4 data = sock.recv(1024)
5 if data == b"ack":
6     print("Connection established")
7 else:
8     print("no ack")
9     sys.exit()
```

Vedlegg D

En kort innføring til `matplotlib`

Matplotlib er et bibliotek (samling med moduler) for Python. Dette biblioteket lager plott som er veldig like plott fra MATLAB. Hvis Matplotlib skal brukes som en fullverdig erstatning for MATLAB sine plottefunksjoner, må Matplotlib kombineres med andre bibliotek fra Python (blant annet NumPy og SciPy). Matplotlib har en "bruksanvisning" på hjemmesiden, som inkluderer en innføring med enkle til mer kompliserte eksempler.[28][29]



Figur D.1: Matplotlib oversikt [29]

D.1 Funksjonalitet i Matplotlib

Matplotlib bringer fram en mengde nye begreper og funksjoner for å hjelpe oss å tegne opp data. I figur D.1 så viser figuren en liten oversikt over begrepene og funksjonene for en Matplotlib figur.

D.1.1 Grunnleggende Begrep

Figur

Figuren består av hele vinduet som kommer frem. Her blir all data og funksjonene sendt til, og lager ett vindu som skal grafisk vise frem all data som skal plottes. Figuren kan bli lagres som ett bilde om det er behov for det.

Axes

Axes er den delen som kan bli regnes som "en plot", dette er delen av figuren hvor som inneholder dataen. En gitt figur kan ha flere Axes, men en gitt Axes kan kun være i en figur. En axes kan inneholde 2 eller 3 axis objekter, alt etter om dataen skal bli plottet i 2d eller 3d. Hver axes har en Title (settes ved bruk av 'set_title()'), en x-label(settes ved set_xlabel()) og en y-label(settes ved set_ylabel()).

Axis

Dette er objektet som kan sette bregrensninger i graf-området. Her kan man justere om man vil ha 'ticks' over grafen, som vises i figur D.1. Man kan også justere hvor mange Axiser man vil ha, for å tegne dataen opp i 2d eller 3d.

Line plot

Når data blir vist på skjermen, kan man tenke seg at det som oftest blir vist som en linje. Hvis data er en liste av målinger over en tidsperiode, så får man en jevn linje. Og dess flere målinger man har over en gitt tidsperiode, jo jevnere blir linjen man plotter. Man kan også endre stilen til linjen ved å endre farge, og ved å definere om det skal være en stiplet linje.

Legend

Legend er det området som kommer fram med navnene på plottet. Her kan man se hvilke linjer som heter hva, slik at man kan enkelt separere forskjellige data.

Title

Title er selve tittelen på plotten vår. Den blir vist øverst i Axes, og er veldig genial å bruke dersom man har flere plots som foregår i en figur.

D.2 Bruksveiledning

Matplotlib er bygget på grunnlag av matlab sin plottefunksjonalitet, og dette vedlegget skal gi deg enn enkel innføring på hvordan dette biblioteket fungerer.

D.2.1 Hvordan implementere inn i programmet

For å starte å bruke dette biblioteket er man nødt til å importere det øverst i python koden sin. Koden man er nødt til å importere blir vist i kodeutdrag D.1.

Kode D.1: Import matplotlib

```
1 import matplotlib.pyplot as plt
2 import numpy as np
```

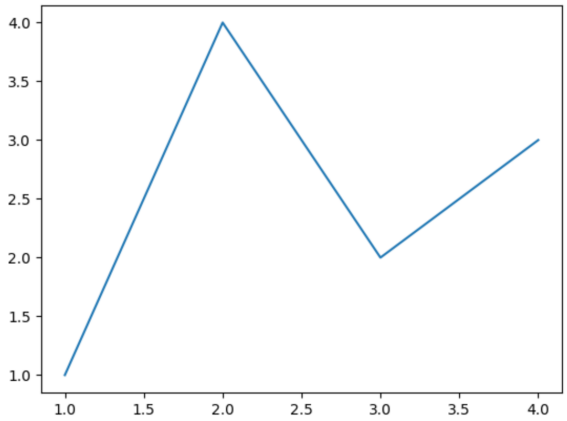
Når disse linjene er lagt til kan man referere til biblioteket ved å bruke plt, og derfor bruke alle de tilgjengelige funksjonene som biblioteket har å tilbyr.

D.2.2 Hvordan fungerer matplotlib

Matplotlib bruker **Figurer** til å tegne dine data ut av programmet på en grafisk måte. Inne i en **Figur** kan man ha en eller flere **Akser**. Ett eksempel på bruk av dette kan være å bruke plt.subplots, også kan man da bruke ax.plot() for å tegne dataen vår. Dette blir vist i kodeeksempel D.2 og figur D.2.

Kode D.2: Matplotlib kode 1

```
1 fig, ax = plt.subplots()  # Lager en figur som inneholder ...
      en enkelt akse
2 ax.plot([1, 2, 3, 4], [1, 4, 2, 3])  # Tegne opp dataen på ...
      aksen.
```



Figur D.2: Eksempel på plot fra pyplot

Samme resultat kan bli oppnådd ved å bare bruke denne koden D.3.

Kode D.3: Matplotlib kode 2

```
1 plt.plot([1, 2, 3, 4], [1, 4, 2, 3])  # Plot data
```

Dette ligner også på funksjonaliteten i kodespråket Matlab D.4.

Kode D.4: Matplotlib kode 3

```
plot([1, 2, 3, 4], [1, 4, 2, 3])  % Plot data
```

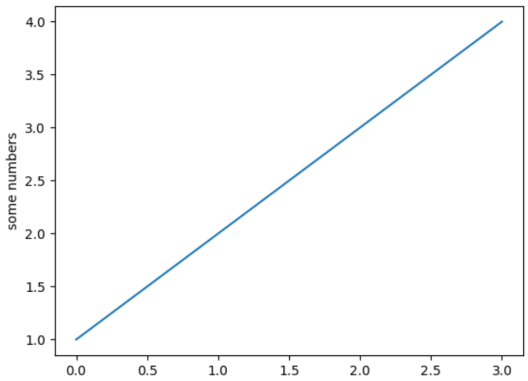
Alle disse tre metodene gir det samme resultatet som kom fram i figur D.2.

D.2.3 Pyplot

Matplotlib.pyplot er en samling av funksjoner som skal fungere like godt som MATLAB sine innebygde funksjoner gjør. Hver pyplot funksjon gjør en eller annen endring til en Figur. Enten det er å lage figuren, plotte noen linjer i ett gitt område, eller dekorere plottet ved labels og andre ting. Å lage grafisk visning av data kan bli gjort veldig enkelt slik (Kode:D.5 Fig: D.3).

Kode D.5: Matplotlib pyplot

```
import matplotlib.pyplot as plt
plt.plot([1, 2, 3, 4])
plt.ylabel('some numbers')
plt.show()
```



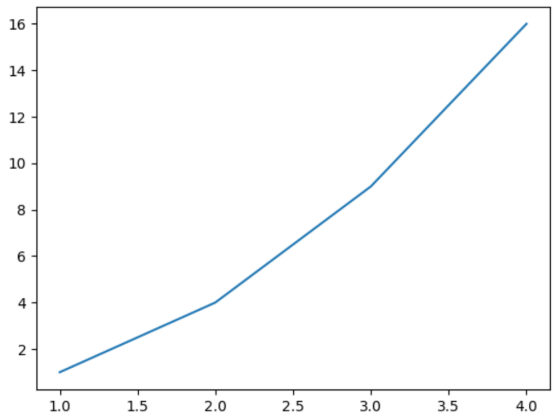
Figur D.3: Eksempel på plot

Man kan lure på hvorfor x-aksen varierer fra 0-3 og y-aksen varierer fra 1-4. Hvis man gir en enkelt liste eller array til plot()-funksjonen, så gjør matplotlib en antakelse at dataen er en sekvens av y verdier, og automatiserer x-verdiene for deg. Siden python starter listene automatisk fra 0 til

[index_slutt]. Men man kan også gi funksjonen vår 2 lister eller array, så vil den automatisk plotte dataen vår slik at x plottes opp mot y. (Fig: D.4 Kode: D.6)

Kode D.6: Matplotlib pyplot

```
1 plt.plot([1, 2, 3, 4], [1, 4, 9, 16])
```



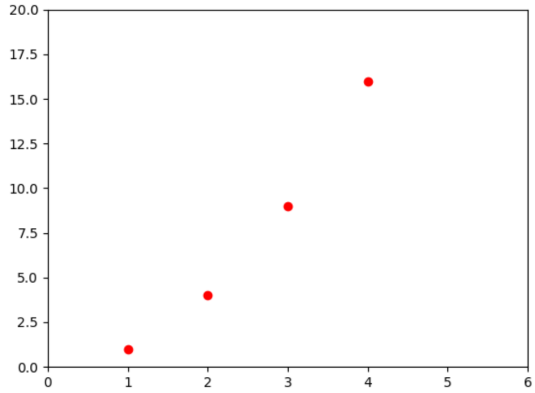
Figur D.4: Eksempel på automatisk skalering

D.2.4 Formattere pyplot

For hvert x,y - par av argumenter, så er det også et tredje argument som plot()-funksjonen kan ta inn. Dette er et valgfritt argument som er i formatet string. Dette argumentet kan endre fargen og type linje som du vil ha på plottet. Disse bokstavene og symbolene er direkte likt som de er på MATLAB. Original stylingen for en plot er 'b-'. Dette betyr at fargen skal være svart, og at linjen skal være sammenhengende. Ett eksempel på en plot som bare skal ha røde punkter, og ingen linjer mellom punktene er:(Fig:D.5 Kode:D.7)

Kode D.7: Matplotlib pyplot

```
1 plt.plot([1, 2, 3, 4], [1, 4, 9, 16], 'ro')
2 plt.axis([0, 6, 0, 20])
3 plt.show()
```



Figur D.5: Eksempel på automatisk skalering

Tabell D.1 viser alle tilgjengelige modifikasjoner man kan ha på ett plot.

Tabell D.1: Tabell over ulike modifikasjoner

Type	String	Beskrivelse
Linje style	-	Solid linje
	–	Dashed linje
	:	Stiplet linje
	-.	Dashed dotted linje
Markering	'o'	Sirkel
	'+'	Pluss
	'*'	Asterisk
	'.'	Punkt
	'x'	Kryss
	'_'	Horisontal linje
	'|'	Vertikal linje
	's'	Firkant
	'd'	Diamant
	'^'	Oppover trekant
	'v'	Nedover trekant
	'>'	Høgre trekant
	'<'	Venstre trekant
	'p'	Pentagram
	'h'	Hexagram
Farge	y	Gul
	m	Magenta
	c	Cyan
	r	Rød
	g	Grønn
	b	Blå
	w	Hvit
	k	Svart

D.2.5 De mest brukte funksjonene fra Matplotlib

Her nevner vi de viktigste funksjonene som man har bruk for når man bruker pyplot i prosjektet.

Funksjon	Beskrivelse
axis	Metode som henter eller setter iformasjonen på en akse
cla	Funksjon for å resette en subplott
close	Lukk ett figur-vindu
draw	Tegn opp figuren på nytt.
figure	Lag en ny figur, eller tegn opp en eksisterende figur.
legend	Sett ett navn på plottet
plot	Plott y opp mot x, kan også modifisere linjen.
savefig	Lagre figuren som ett bilde
subplot	Sett opp en ny subplot på figuren
subplots	Lage en figur, og sette opp flere subplots.
title	Sett en tittel for subplottet
xlabel	Sett navn på x-aksen
ylabel	Sett navn på y-aksen
funcAnimation(fig,func,interval)	FuncAnimation er den metoden vi bruker får å få til livePlotting. Her legger man inn hvilken figur som skal plottes, på hvilken måte og hvor stort intervall i millisekund som skal skje mellom hver måling.

D.2.6 Hvordan Pyplot brukes i prosjektet

Prosjektet bruker pyplot til å tegne opp målingene vi får fra roboten. Alle sensorene og motormålingene blir registrert over den tidsperioden som roboten er aktiv.

Flytdiagram: 4.5

Kode D.8: `F8_PlotData.py`

```
22 def plotData():
23     ax1.cla()
24     ax2.cla()
25     ax3.cla()
26 
27     ax1.plot(time[0:-1], light[0:-1])
28     ax1.set_title('Light')
29 
30     ax2.plot(time[0:-1], lightDeviation[0:-1])
31     ax2.set_title('lightDeviation')
32 
33     ax3.plot(time[0:-1], lightIntegrated[0:-1])
34     ax3.set_title('Light Integrated')
35     ax3.set_xlabel('Time [sec]')
```

Vedlegg E

Typiske feilmeldinger

Dette vedlegget er med for å gi en oversikt over noen feilmeldinger som elever sannsynligvis vil møte på. Ved å referere til dette vedlegget blir feilsøkingsprosessen forenklet. Generelle tips for å unngå feil kan oppsummeres slik:

1. Ikke rør kode som ikke skal røres (indikert vha. kommentarer).
2. Filer som kjører på EV3en må ha LF format, se figur 2.11 og figur 2.12
3. Pass på at rett ip adresse er lagt inn i `BeregnOgPlott.py`
4. ting4 etc

E.1 "`/usr/bin/env: 'pybricks-micropython\r': No such file or directory`" og andre feilmeldinger som involverer "`\r`"

Indikerer at `CRLF` er brukt som "End of Line Sequence". Skift til LF. Dette blir vist og forklart i figur 2.11 og 2.12. Etter å ha satt "End of Line Sequence" til `LF`, må workspace sendes til EV3en på nytt.

E.2 "JSONDecodeError: Expecting value: line 1 column 1 (char 0)"

Det kan være mange grunner til at denne feilmeldingen dukker opp. Under vårt arbeid kom denne feilmeldingen ofte opp hvis ingenting ble sendt til datamaskinen. Det kan det være mange grunner til det. For eksempel kan det være feil i koden som gjør at EV3en ikke tar inn målinger. Dette kan en se ved å sjekke om `measurements.txt` er tom eller ikke etter kjøring. Typisk løses denne feilen ved å kjøre `EV3main.py` i offline modus. Dette gjør en for å sørge for at det faktisk blir skrevet til `measurements.txt`, som da indikerer at det blir tatt målinger fra sensorer og motorer osv.

Det er altså lettest å først sørge for at alt fungerer i offline-modus før en sjekker om online-modus fungerer. Hvis det fortsatt oppstår en "`JSONDecodeError: Expecting value: line 1 column 1 (char 0)`" feil, etter å ha sørget for at offline-modus fungerer, må en sjekke `SendData` funksjonen i `EV3main.py`. Det er viktig at `json.dumps`-funksjonen mottar en dictionary. Eksempel på korrekt bruk er vist i kodeutdrag E.1.

Kode E.1: Klargjøring av data for `json`

```
1 data = {}
2 data["time"] = (time[-1])
3 data["light"] = (light[-1])
4 if online:
5     msg = json.dumps(data)
6     robot["connection"].send(msg)
```

E.3 "console-runner-service is busy ----- Exited with error code 1."

Program kjører på EV3 - avslutt programmet på EV3en først. Dette kan gjøres ved å trykke på tilbake-knappen på EV3en.

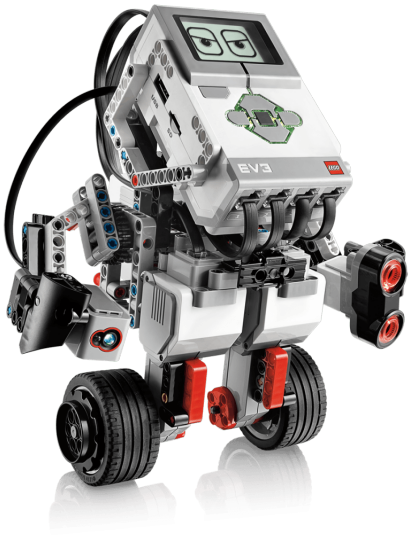
E.4 “OSError: [Errno 48] Address already in use” og lignende feilmeldinger

Dette kan for eksempel skje når du kjører EV3en (i online-modus) rett etter en tidligere kjøring av EV3en (også i online-modus). Dette skjer fordi det lages en socket på samme port som den forrige kjøringen av EV3en. Når serveren lukker en socketforbindelse, så vil serversocketen være i en såkalt `TIME_WAIT` tilstand. Dette kan vare i et par minutter. For å forsikre om at dette ikke skal skje, må linje 108 (kan variere fra prosjekt til prosjekt) i `EV3main.py` ha `sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)`. Det relevante her er `socket.SO_REUSEADDR`-parameteren. `ProjectOX` har dette som standard, men hvis det blir rørt med socket-objektet så kan det være lurt å sjekke at dette fortsatt står.

Vedlegg F

EV3-Teknologi og maskinvare

Dette kapittelet handler om grunnsettet som blir benyttet i oppgaven og teknologien bak det. Grunnsettet som er benyttet heter EV3-Mindstorms. Eksempel på robot blir illustrert i Figur F.1.



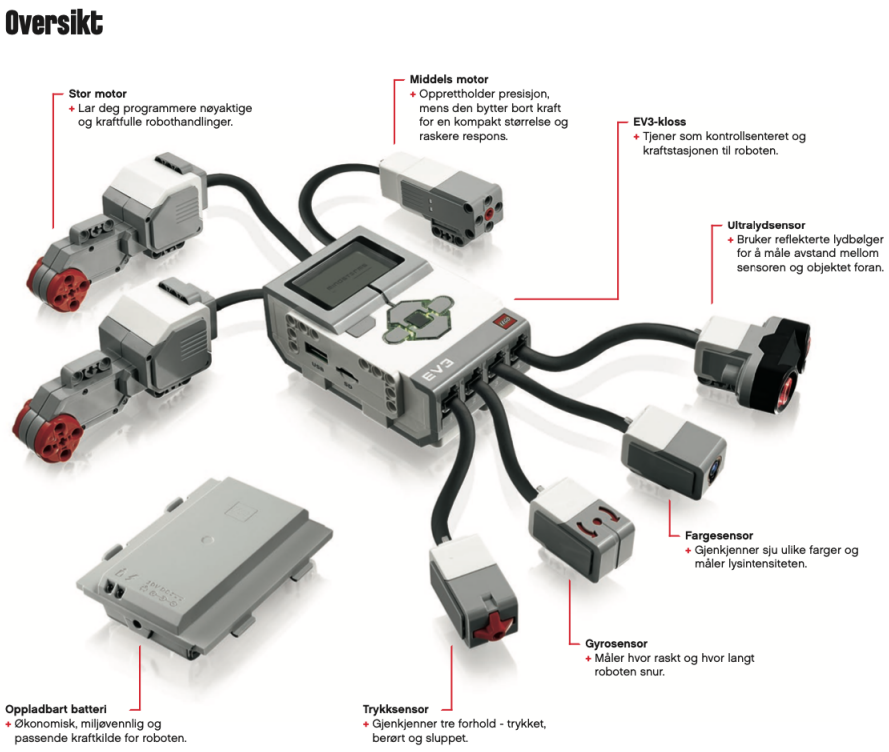
Figur F.1: EV3-Mindstorms [10]

Dette er den tredje utgaven av Lego Mindstorms robot kit. Dette kittet kom ut i 2013, og blir fortsatt brukt per dags dato. Roboten har uendelig med mulige prosjekt som kan bli utført, men hovedmålet til roboten er å gi en enkel og lærerik innførelse av programmering til barn og unge. Roboten skal programmeres i python ved hjelp av Lego Mindstorms utvidelsen som blir forklart i kapittel 2.

Roboten har flere ulike elementer. Disse elementene vil bli forklart i dette kapittelet ved hjelp av illustrasjoner og tilhørende beskrivelser av robotens bruksområder.

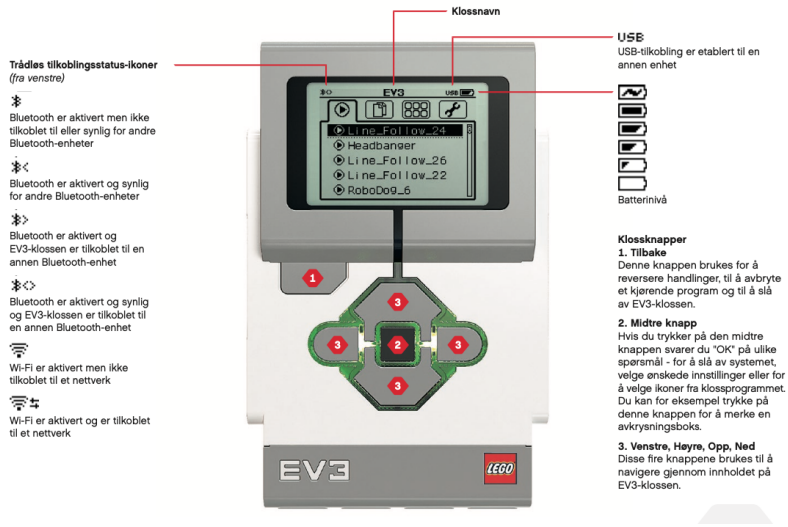
F.1 Ev3 Kloss

Figur F.2 illustrerer alle komponentene/elementene som grunnsettet inneholder. Alle komponentene er navngitt og tilknyttet en kort forklarelse. Den grafiske oversikten er hentet fra brukerveiledningen til Lego [10].



Figur F.2: Oversikt over innholdet i Mindstorms [10]

F.1.1 Oversikt over brukergrensesnittet



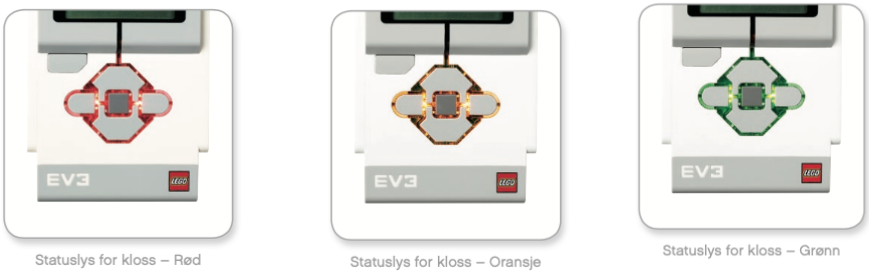
Figur F.3: Oversikt over EV3 Klossen [10]

Som illustrert på Figur F.3 er det en liten skjerm tilknyttet EV3-klossen. Denne skjermen viser hva som skjer på klossen, og gjør det mulig å benytte grensesnittet. Ved hjelp av skjermen er det også mulig å legge til tekst eller numeriske verdier under eksperimenteringen. Dette gjøres ved hjelp av knappene som ligger under skjermen. Det er også mulig å bruke dem til programmerbare aktivatorer. Brukergrensesnittet endres når MicroPython (Kapittel 2.2) blir installert på roboten. Dette gjøres for å endre kodespråket til klossen fra MATLAB til Python. Endringer i brukergrensesnittet blir forklart i Kapittel 2.2.

F.1.2 Status

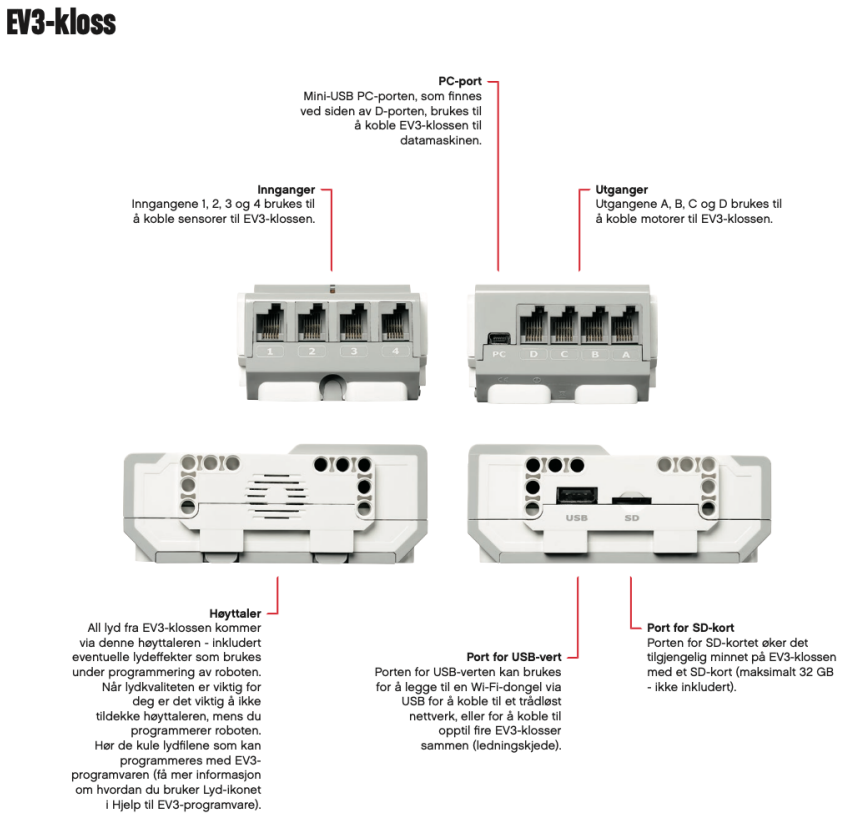
Bak knappene på klossen er det innebygde lys. Hensikten med disse lysene, er å informere brukeren om statusen til klossen. Dette gjøres ved hjelp av ulike farger som har hver sin mening. Disse blir vist i figur F.4 og forklaringene er som følger:

- Rød = Oppstart, oppdatering, system slås av
- Rød pulserende = Opptatt
- Oransje = Varsel eller klar
- Oransje pulserende = Varsel, kjører
- Grønn = Klar
- Grønn pulserende = Kjørende program



Figur F.4: Status [10]

F.1.3 Porter



Figur F.5: Oversikt over EV3 Klossen [10]

Figur F.5 illustrerer en oversikt over portene til klossen. Oversikten er hentet fra brukerveiledningen til Lego [10].

F.1.4 Tekniske Spesifikasjoner

De viktigste tekniske spesifikasjonene til EV3 klossen er [10]:

- Operativsystem: Linux
- 300 MHz ARM9-kontroller
- Flash minne på 16MB
- Ram på 64MB
- Klossens skjermoppløsning på 178x128
- Farge på skjerm: Svart/Hvit
- USB 2.0-kommunikasjon til verts-pc - Opptil 480 Mbit/sek
- USB 1.1-vertskommunikasjon - Opptil 12 Mbit/sek
- Micro SD-Kort - Støtter SDHC, Opptil 32GB
- Motor og Sensor porter
- Tilkobling RJ12
- Støtter Auto ID
- Strøm - 6xAA-batterier / Eller oppladbart batteri

F.2 Sensorer og deres funksjoner

Figur F.6: Sensorene i Mindstorms

Mindstorms settet inkluderer flere ulike sensorer (figur F.6). Sensorene er lagt slik at studenter og elever som bruker dem, får en morsom og lærerik innføring i matte og fysikk. Sensorene er ulike og skal gi ulik innsyn i matematikkens verden. Sensorene blir brukt sammen med EV3-klossen og programmering, som gjør det enklere for studentene å lære.

F.2.1 Ultralyd sensor

Det er flere ulike typer sensorer som følger med Mindstorms settet. En av disse sensorene er en ultralydsensor. Ultralydsensoren er en digital sensor som kan måle avstander til objekter foran. Dette skjer ved at sensoren sender ut høg-frekvens lydbølger og måler tiden det tar for at bølgene kommer tilbake. De høye frekvensene er for kraftige til at mennesker kan høre dem. Eksempel av sensor i figur F.7.

Figur F.7: Ultralydsensor Mindstorms

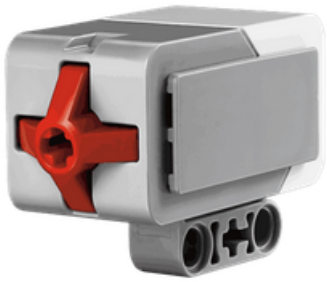
Avstanden til objektet kan bli målt i tommer eller i centimeter. Dette gir bruker muligheten til å programmere roboten til å stoppe en gitt mengde avstand foran ett objekt eller en vegg.

Den registrerbare avstanden til roboten ligger mellom dataverdiene 3 til 250 cm, med en unøyaktighet på +- 1 cm. Ved bruk av tommer, blir den registrerbare avstanden mellom 1 og 99 tommer, med en unøyaktighet på +- 0,394 tommer. Om avstanden er lenger enn 100 tommer eller 255 cm, så betyr det at roboten ikke kan registrere objektene foran seg. [10]

Ultralydsensoren blir brukt som øynene til roboten. Den kan bli programmert til å unngå kollisjon mellom møbler, spore bevegende mål, registrere inntrengere, eller pinge"med økende volum, dersom ett objekt nærmer seg sensoren.

F.2.2 Trykk sensor

Den andre sensoren som er en del av Mindstorms settet er trykksensoren. Trykksensoren er en digital sensor som registrerer når sensorens røde knapp er trykket inn og blir sluppet. Dette betyr at sensoren kan bli programmert etter tre ulike forhold: registrert trykket, registrert sluppet eller registrert trykket og sluppet. Eksempel av sensor i figur F.8:



Figur F.8: Trykksensor Mindstorms

Ved å bruke signalene fra sensoren, kan roboten for eksempel simuleres til å se verden som en blind person, ved å strekke ut hånden å reagere om knappen rører et objekt (registrert trykket).

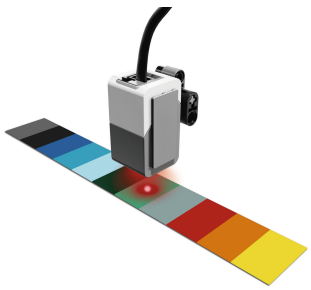
Ett eksempel på bruk av trykksensoren er å ha knappen trykket inn mot ett bord, samtidig som roboten kjører nedover langs bordet. Ettersom roboten nærmer seg kanten, så kommer knappen til å bli sluppet og roboten må stoppe.

F.2.3 Fargesensor

Den tredje sensoren som følger med i settet er fargesensoren. Fargesensoren er en digital sensor som kan registrere fargen eller intensiteten til lyset, som kommer inn i det lille vinduet foran på sensoren. Denne sensoren kan brukes i to forskjellige moduser: Fargemodus eller Reflektert lysintensitet. Eksempel av sensor i figur F.9:

Figur F.9: Fargesensor Mindstorms

I **fargemodus** gjenkjenner sensoren 7 forskjellige farger: Blå, Svart, Gul, Grønn, Brun og Hvit. Robotens evne til å se farger gir mulighet til å kunne skille mellom blokker eller baller med ulike farger. Eksempel i figur F.10:



Figur F.10: Fargemodus

Ved bruk av modusen **fargeintensitet**, måler sensoren intensiteten på fargen foran seg. Dette gjør sensoren med hjelp av en lysemitterende pære.

Lyset fra pæren reflekteres på overflaten med farge og styrken på refleksjonen forteller styrken på fargen. Årsaken til dette er at mørkere farger absorberer mer lys en lysere farger. Fargene sensoren leser blir kategorisert etter en skala fra 0 (meget mørkt) til 100 (meget lyst). Dette gir roboten evne til å følge en svart linje på et ark. Eksempel i figur F.11:

Figur F.11: Lysintensitet

For å få størst mulig nøyaktighet i målingene, bør sensoren holdes i rett vinkel og i nærheten av overflaten som sensoren skal måle. Det er også viktig at sensoren ikke rører overflate direkte, men bare ligger i nærheten.

F.2.4 Gyrosensor

Gyrosensoren er en digital sensor som registrerer rotasjonbevegelse på en enkel akse. Dette vil si at sensoren observerer hvor mye sensoren roterer seg i en enkel retning. Gyrosensoren kan bare gjøre dette i en bestemt retning, som er illustrert på toppen av sensoren (Figur F.12).

Figur F.12: Endepunktene på sensoren

Maksfrekvensen sensoren har mulighet til å måle er 440 grader per sekund. Sensoren benyttes til å fortelle når roboten snur seg, eller om roboten er i ferd med å falle over. Et eksempel på bruk av sensoren er et selv-balanserings prosjekt. Eksempel av sensor i figur F.13:

Figur F.13: Gyrosensor Mindstorms

Sensoren vil ha en oversikt over total rotasjon som er utført, i grader. Dette kan bli brukt til å registrere hvor mye roboten har rotert siden programstart. Ved bruk av gyrosensoren sensoren vil det være mulig å programmere roboten til å svinge. Dette vil sensoren kunne gjøre med en nøyaktighet på +-3 grader ved en 90 graders rotasjon

F.3 Motorer og deres funksjoner

Mindstorms settet inkluderer også 2 typer motorer, stor og medium. Motorene kan brukes til å bevege på selve roboten, eller å utføre prosjekt som har behov for å bevege gjenstander. Motorene har en innebygd rotasjonssensor med 1-graders oppløsning for nøyaktig kontroll. To store og en medium motor er inkludert i settet.

Den store motoren er den kraftigere av de to typene, men tar også opp mer volum og veier mer. Eksempel av motor i figur F.14:

Figur F.14: Stor motor

Medium motoren er mindre og lettere enn den store motoren. Motoren kan reagere fortere, og mer presist. Dette medfører at motoren ikke er like kraftig som den store og bør ikke benyttes til å flytte store objekter. Eksempel av motor i figur F.15

Figur F.15: Medium stor motor

F.4 Koble til sensorer og motorer

For at EV3 skal kunne benytte seg av de ulike motorene og sensorene vil det være nødvendig å koble de opp til EV3 klossen.

Sensorene kobles til klossen ved hjelp av de svarte tilkoblingskablene som er knyttet til hver sensor. Klossen har fire ulike porter som blir benyttet for å koble dem til. Hver av portene er designert en spesifikk sensor. Standard-innstillingene for portene er:

1. Trykksensor
2. Gyro-/Temperatursensor
3. Fargesensor
4. Ultralyd-/infrarødsensor

Hvis EV3-klossen er tilkoblet 'enheten' mens det programmeres, vil programvaren selv kunne identifisere de ulike sensorene. Det vil derfor være hensiktsmessig å benytte riktig port for tilhørende sensor. Eksempel i figur F.16

Figur F.16: Koble Motor og Sensor

F.4 Koble til sensorer og motorer

Ved tilkobling av motorene vil det være et lignende oppsett som med sensorene. Eneste ulikheten er at portene blir kategorisert med bokstaver A-D. Rekkefølgen vil også ha stor betydning her likt som med sensorene.

- Port A: Middels motor
- Port B og C: To store motorer
- Port C: Stor motor

Vedlegg G

Integrasjon

G.1 Problemstilling

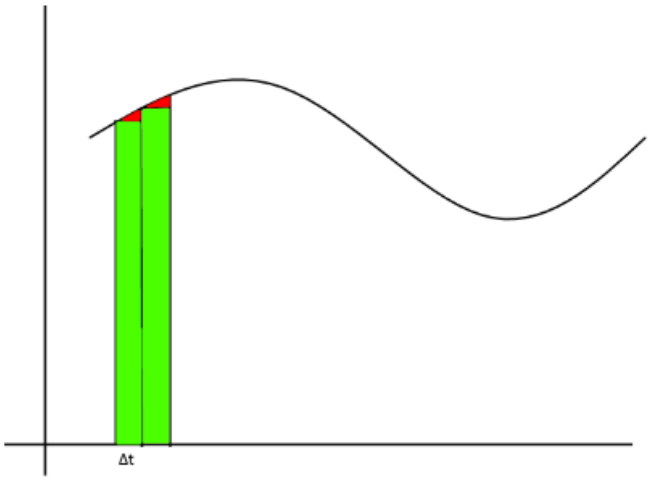
I ING100 faget handler det første obligatoriske prosjektet om integrasjonen. Dette prosjektet gir en innsikt i hvordan man kan integrere ett gitt signal, og forklarer behov for å beregne den integrerte. Kapittelet har tatt utgangspunkt i prosjektbeskrivelsen for ING100 (som beskrevet i vedlegg K), bare ved bruk av MicroPython og Prosjekt0X-rammeverket som er laget. Dette gir studenten mulighet til å regne ut totalt akkumulerte verdier for signaler.

G.2 Teori og integrasjon i praksis

Det er mange ulike bruksområder for integrasjon. En av de vanligste integrasjons metodene som benyttes er numerisk integrasjon. Numerisk integrasjon benyttes overalt i samfunnet. Eksempler på ulike bruksområder er illustrert i figur G.1.

Figur G.1: Praktiske anvendelser der numerisk integrasjon foregår i praksis

Flowmålere blir typisk brukt i prosessindustrien (meieri, olje- og gassindustri, vannverk,osv.), men de brukes også i bensin-/dieselpumper og sprøyte lakkeringsroboter til å beregne levert volum av melk/olje/vann/-drivstoff/lakk. **Automatiske strømmålere** og **hurtiglade-re til elbil** beregnerlevert effekt ut fra strøm- og spenningsmålinger. **Sykkelkomputeren** beregner tilbakelagt strekning ut fra farten (som igjen er basert på et magnetréle som registrerer at hjulet roterer). Dette er bare noen få eksempler på hvordan numerisk integrasjon blir benyttet av ulike signaler.



Figur G.2: Illustrasjon numerisk integrasjon

Ved bruk av numerisk integrasjon måles arealet under en oppgitt graf. Som illustrert på figur G.2 er denne metoden ikke 100 prosent nøyaktig, men feilmarginen er ikke stor nok til at det har en betydning. Det grønne området på figuren illustrerer det observerte arealet etter integrasjon. Det røde området er feilmarginen, området som ikke blir observert. Denne feilmarginen vil variere etter stigningen til grafen Δt.

G.3 Forslag til løsning

Ved integrasjon summeres arealet under en graf. Arealet under grafen blir delt inn i ulike søyler, likt som i figur G.2. Ved integrasjonregning, blir grafen hovedfaktoren for utregningen. For å kunne få nyttige verdier vil det være nødvendig å velge et start og et sluttsted for målingene. Dette vil da gi mulighet for et estimat av integralet. Dette kommer bedre frem senere i kapittelet.

Man tenker oss at det er en innverdi: a.

Man vet da at det bestemte integralet: $\int_0^{t_1} a\,dt = a * t_1$

Ved å bruke dette bestemte integraler vil det være mulig å lage en liste av integrasjoner. Hver av disse integrasjons prosessene vil være avhengig av resultatet fra integrasjonen før i rekka. Dette vil medføre en kontinuerlig oppføring av nye verdier til lista.

Denne metoden er ofte benyttet og det er dermed valgt å lage det om til en funksjon som kan bli brukt flere ganger. Denne funksjonen er gitt navnet EulerForward, som kommer fra metoden brukt for integreringen.

Kode G.1: `EulerForward()`-funksjonen i `BeregnOgPlott.py`

```
1 def EulerForward( IntValuOld,  FunctionValue ,  TimeStep):
2     IntValueNew =  IntValuOld + (FunctionValue * TimeStep)
3     return IntValueNew
```

Som en kan se i kode-snuttet (G.1) på linje 2, ser man at man hele tiden tar forrige verdi til den integrerte, og legger på målt verdi for lengden målt i T_s.

IntValueOld er så langt integrert verdi, og er den siste beregnede integreringsverdien. FunctionValue er den verdien man ønsker å integrere. TimeStep er tidsskrittet fra når forrige verdi ble registrert. Dette blir omberegnet til IntValueNew.

Ved å legge dette inn i en while-løkke som kontinuerlig måler mengde re-

flektert lys til en lyssensor, så kan vi integrere total mengde reflektert lys.

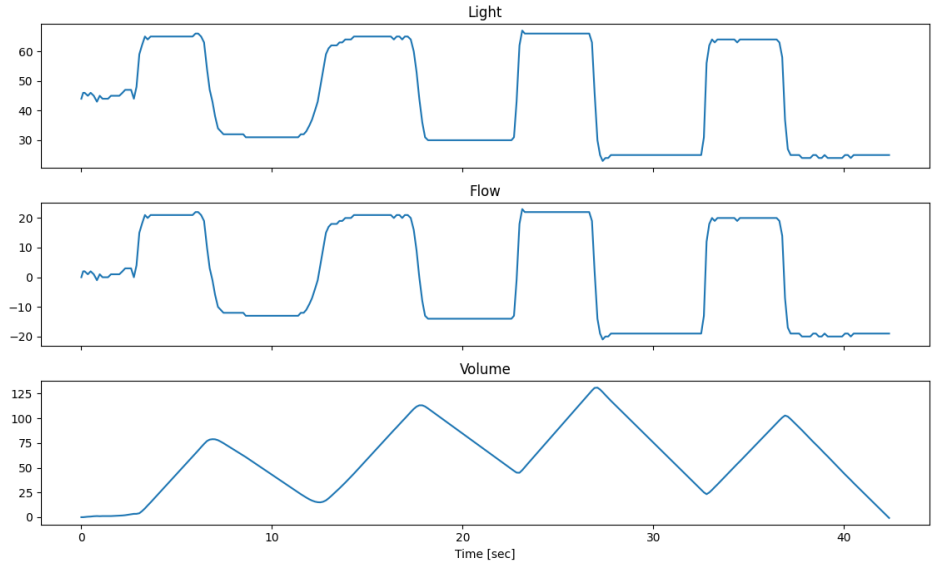
Vi definerer nullpunktet (G.2) til integreringa til å være lik første måling, dette for å kunne teste både negativ og positiv integrering.

Kode G.2: `Bruk av EulerForward()`-funksjonen i `BeregnOgPlott.py`

```
1 if online: # hvis online modus
2         # Flow
3         flow.append(light[-1] - light[0])
4         if len(time) > 1:
5             # ts
6             ts.append(time[-1] - time[-2])
7             # Volume - "light integrated" Euler metoden
8             volume.append(EulerForward(volume[-1], ...
                  flow[-2], ts[-1]))
```

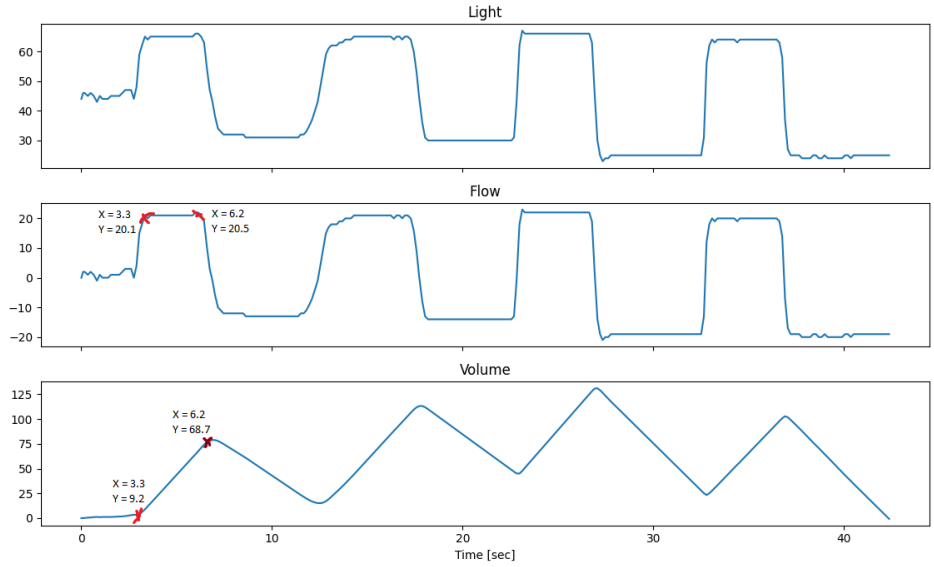
G.4 Resultat

For å sjekke at valgt løsningmetode fungerte var det viktig å teste hvordan det samsvarte med innsignalet til lyssensoren. Dette ble utført ved å endre styrken på lyset og observere utslaget. Dette ble gjort ved å dra sensoren over ett lysark (lysarket kommer dypere beskrevet her:5.4), deretter plottet man de integrerte verdiene og lysverdiene.



Figur G.3: Første verifisering av integrasjonsfunksjon

Her måtte man komme fram til ett område der man kunne utføre verifiseringen. Dette endte opp med å bli målingene mellom x = 3,3 og y = 6,2. Som illustrert i figur G.4.



Figur G.4: Første verifisering av integrasjonsfunksjon med tallverdier

Basert på målingene i figur G.4, vil det være mulig å sjekke om verdiene er korrekte.

Om funksjonen har funnet ønsket resultat, skal det tilsvare:

$$Integral = \Delta Y * (t_1 - t_0) \tag{G.1}$$

Hvis integralet stemmer vil det være mulig å sette inn de tidligere målingene fra figur G.4, subplot 2

$$(20,5 - 0) * (6,2 - 3,3) = 59,45 \tag{G.2}$$

Dette er samme verdi som er illustrert mellom de to første knekkpunktene i figur G.4 subplot 3. Her starter Y aksen på 9,2 og slutter på 68,7. Dette gir da en differanse på 59,5, tilnærmet likt som utregnet verdi.

Ved numerisk integrasjon er det stor sannsynlighet for å få avvik ifra inte-

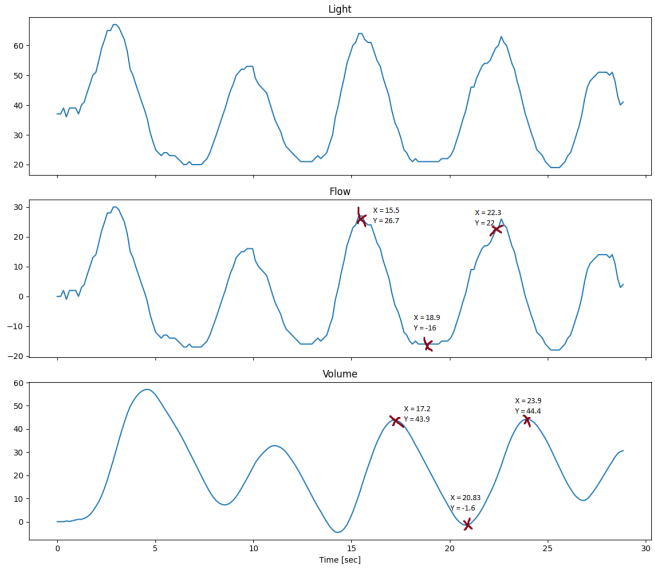
grert verdi, i dette tilfellet oppstod det ikke.

Dette potensielle avviket er sterkt påvirket av hvor ofte det vil være mulig å måle, noe som blir illustrert i Figur G.2. I dette tilfelle vil tiden mellom målingene være drastisk mye høyere en under annen moderne teknologi.

Neste verifikasjons metode benytter sinuskurver. Her er største utfordring å produsere en sinuskurve basert på målingene.

Metoden som ble benyttet for å oppnå dette var å bevege lyssensoren i en sirkel på et gråskalaark med ulike gråtoner. Gråtonene gikk fra hvit til svart, noe som er lett å lese for lys-sensoren.

For å bekrefte at metoden dannet ønsket sinuskurve var det nødvendig å finne et matematisk uttrykk for kurven. Dette ble gjort ved å plotte verdiene for så å kunne finne frekvens, amplitude og periodetid. Området som var mest stabilt på figur G.5, subplot 2 ble valgt som utgangspunkt. Det ble dermed satt ned tre punkter som ble benyttet for å finne ønsket verdier.



Figur G.5: Avmerkede punkter på sinuskurve

G.4 Resultat

Et sinussignal kan skrives på formen$f(t) = A * sin(w * t)$ Dermed kan vi nå finne det mattematiske utrykket til denne figuren.

Og som indikert i figur G.5, data:

$$Amplitude : A = \frac{28.7 - (-16)}{2} = 22,35$$

$$Periodetid : T_p = 22.3 - 15.5 = 6.8$$

$$Frekvensen : f = \frac{1}{T_p} = 0.147$$

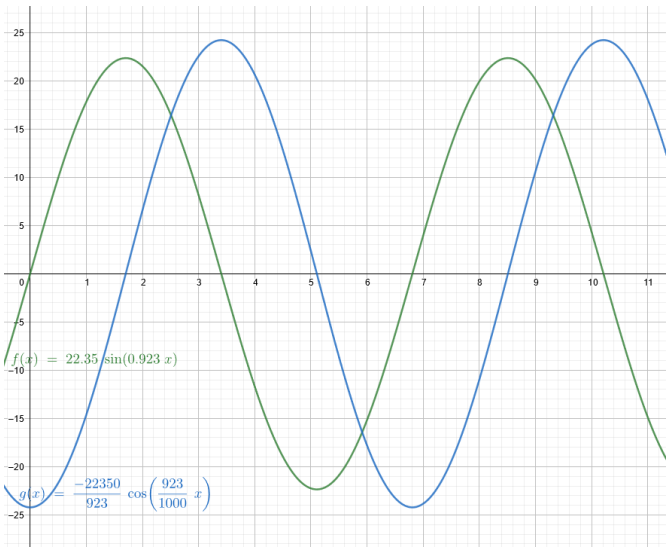
$$Vinkelfrekvens : w = 2 * pi * f = 2 * pi * 0.147 = 0.923$$

$$Signalet : 22,35 * sin(0.923t)$$

Ved å regne ut integralet av dette utrykket, skal resultatet stemme overens med det nederste subplott på figur G.5

Hvis utrykket settes inn i en kalkulator, skal resultatet bli:

$$\int 22,35 * sin(0.923t) = -24.214437 * cos(0.923t) \qquad \text{(G.3)}$$



Figur G.6: De matematiske uttrykkene plottet over hverandre

Figur G.6 illustrere de to ulike utrykkene i samme plot. Her observeres det at kurvene er svært like, men er forskjøvet fra hverandre med 2 sekunder. Ved å sjekke Figur G.6 mot tidligere Figur G.5, ser vi at teorien stemmer. Dette kan bevises ved å måle amplitudene til grafene. Da ser vi at amplituden på figur G.6 er 24,2 noe som tilnærmer amplituden i G.5.

Eventuelle avvik kan komme fra; tidsforsinkelser mellom kommunikasjon, det faktum at prosjektet integrerer numerisk, det at man alltid kun kan integrere det som har skjedd og ikke det som kommer til å skje, etc.

Vedlegg H

Numerisk Derivasjon

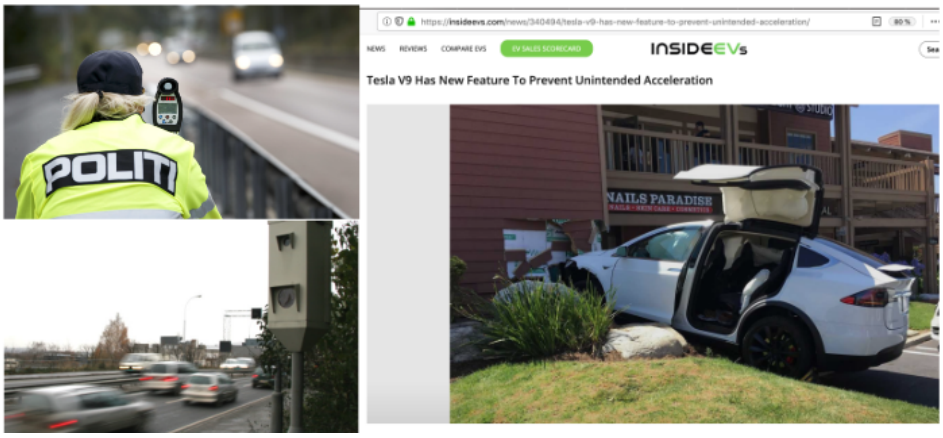
H.1 Problemstilling

I mange tilfeller kan det være nyttig å kunne derivere ett signal. Den deriverte kan fortelle oss om endringer i prosesser mens de foregår. Dette gir oss muligheten til å justere prosessen før det går for langt.

Dette er en del av den obligatoriske delen av prosjektet fra prosjektbeskrivelsen i vedlegg K.

H.2 Teori og Derivasjon i praksis

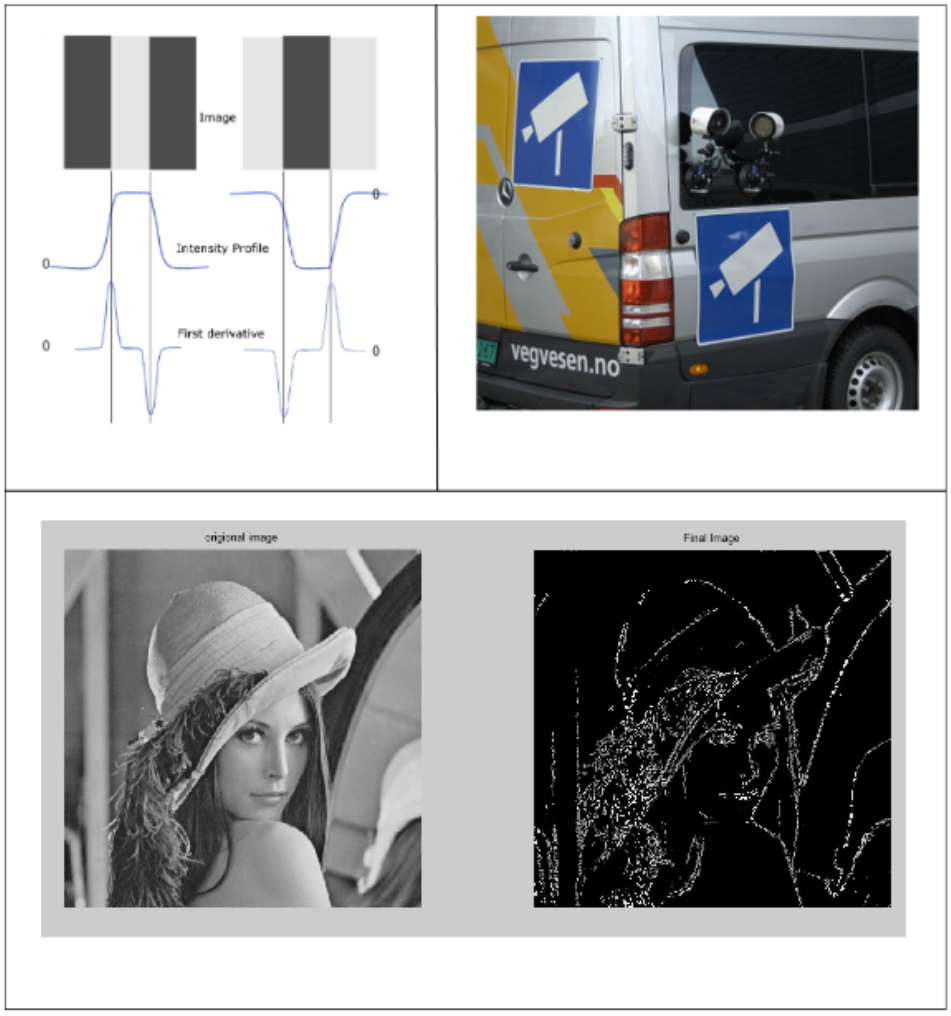
I dette kapittelet skal du få en introduksjon til numerisk derivasjon. I figur H.1 vises noen praktiske eksempler hvor numerisk derivasjon inngår som endel av beregningsgrunnlaget.



Figur H.1: Praktiske eksempler hvor numerisk derivasjon er en vital del av funksjonaliteten.

Når politiet gjennomfører fartskontroll, er farten de beregner basert på en laser som måler avstanden til bilen. Ved først å filtrere avstandsmålingen og deretter derivere denne, får de et estimat på farten. Samme prinsipp gjelder for fotobokser. Når Tesla har innført begrensinger i akselerasjon, kan de enten gjøre det ved å bruke et akselerometer eller numerisk derivere farten.

Numerisk derivasjon blir også brukt i applikasjoner som automatisk skiltgjenkjenning som Statens Vegvesen bruker for å finne ut om du som bilist har betalt veiavgiften. I tillegg brukes det i kantdeteksjon i andre bildebehandlingsoppgaver, se figur H.2.



Figur H.2: Andre praktiske eksempler hvor numerisk derivasjon brukes til kantdeteksjon i bildebehandling.

Som vist øverst til venstre i figur H.2, brukes den deriverte av bildeintensiteten til å avgjøre hvor det er en kant.

H.3 Forslag til løsning

Ettersom signalene man får inn fra roboten er en serie med verdier, så kan vi ikke derivere på normalt vis, men vi kan tilnærme oss en derivert ved å bruke tidligere målte verdier. Så det man i realiteten finner er sekanten mellom to tidligere målepunkter, og er strengt tatt ikke den deriverte.

Selve formelen for en sekant:

$$sekant\, mellom\, t\, og\, t + \Delta t = \frac{f(t + \Delta t) - f(t)}{\Delta t}$$

Som en ser, så ligner dette på definisjonen av den deriverte, med unntak av at vi ikke klarer å få Δt til å gå mot 0, vi må berre godta at kommunikasjonen med EV3 enheten vil skape forsinkelser i systemet. Deretter kan Δt bli liten, og detetter gir en nokså konkret sekant.

Dette gir følgende kode i Python i Kode: H.1

Kode H.1: `Derivasjon()`-funksjonen i `BeregnOgPlott.py`

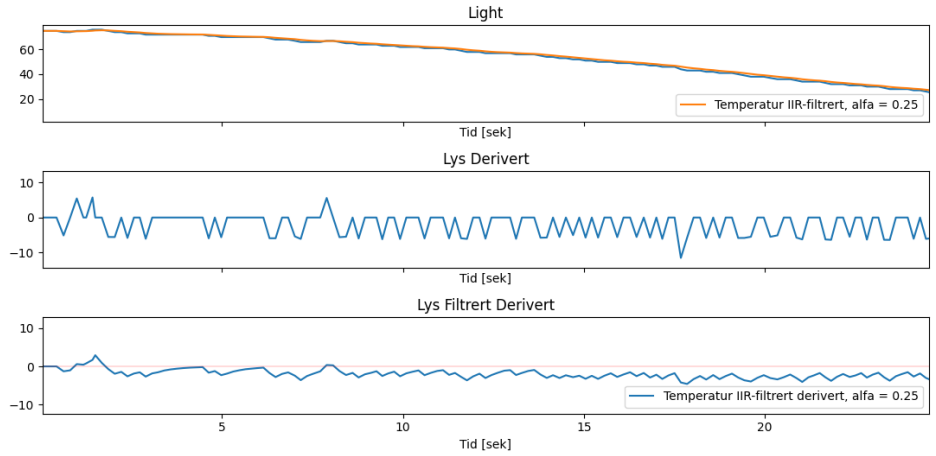
```
1 def Derivasjon(light, timestep):
2     #Derivation of input values
3     #Returns the derivative of the secant between two points
4     if timestep[-1] == 0 or len(light) < 2:
5         sekant = 0
6     else:
7         sekant = ((light[-1]-light[-2])/(timestep[-2]))
8     return sekant
```

Koden tar inn siste verdi fra timestep-listen forklart fra MathCalculations. Dersom det ikke er nok elementer i lys-listen, så kommer koden bare til å returnere 0. Hvis siste verdi i timestep-listen blir 0, så returnerer man 0 som sekant-verdi. Denne funksjonen ble lagt for Online-modus.

H.4 Resultat

Som en del av den obligatoriske delen av dette prosjektet ble det verifisert at derivasjonsfunksjonen H.1 fungerer.

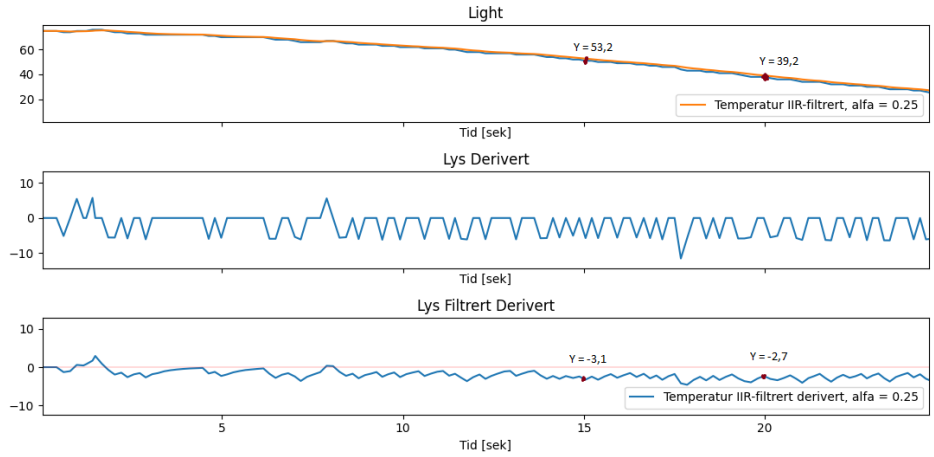
Den første utfordringen var da å produsere ett signal som kan illustrere dette på en effektiv og god måte. Dette oppnås med å bevege lyssensoren sakte over ett gråskala ark (Forklart tidligere). Her i Figur H.3 kan en se at den deriverte er lik null når når sensoren ikke registrerer noen bevegelser. Også ser man at den deriverte blir negativ mens verdiene i lyssensoren minsker i verdi, dette skjer fordi man beveger sensoren sakte langs skalaen.



Figur H.3: Sakte bevegelse av lyssensor nedover en gråskala

Som en også kan se fra grafene i Figur H.3, så vil det være vanskelig å derivere signaler med støy. Derfor bør man alltid sjekke sensorens signaler om de kan ha vært utsatt for støy eller ikke. Dersom signalet ble utsatt for mye støy, bør det bli behandles, enten med å sortere ut feilmålingene, eller bruke filtreringen fra Prosjekt 02.

For å identifisere om målingene er korrekt, ble det valgt å finne et område i datasettet som var tilnærmet lineært. Dette området ble imellom x = 15 og x = 20. Dette kan en se i figur H.4.



Figur H.4: Datapunkter på x = 15 og x = 20

Her ble det funnet y-verdiene for både Lys og for den deriverte. Her ble de filtrerte lys signalene valgt for de var de mest nøyaktige.

Basert på punktene er det nå mulig å sjekke om den deriverte er korrekt.

$$\frac{dy}{dx} \approx \frac{\Delta y}{\Delta x} \tag{H.1}$$

Basert på tallene fra figur H.4 så får vi følgende reknestykke:

$$\frac{39,2 - 53,2}{20 - 15} = -2.8$$

Om man leser av den deriverte fra figur H.4 så registreres det at den deriverte ligger mellom -3.1 og -2.7.

Videre i den obligatoriske delen av prosjektbeskrivelsen, har vil man på samme måte som prosjekt01 derivere et sinussignal.

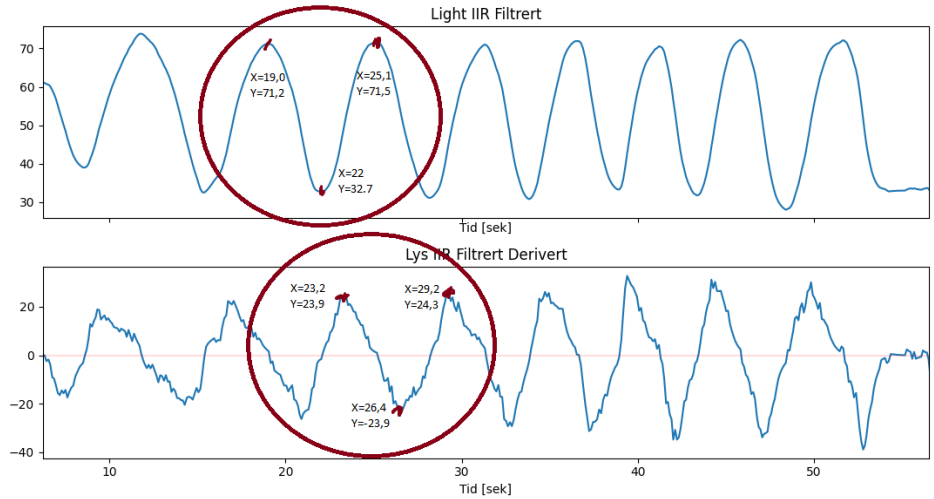
Man skal deretter verifisere koden ved å derivere en sinusfunksjon med konstantledd (bias) b, amplitude a og frekvens w [rad/s] som vist i ligning (H.2)

$$d/dt * (b + a * sin(w * t)) =? \tag{H.2}$$

H.4 Resultat

Da ble neste utfordring å lage ett sinussignal ved hjelp av lyssensoren. Dette utførtes ved å bevege sensoren sakte men nøyaktig frem og tilbake på gråskala-arket. Dette måtte bli utført så forsiktig som mulig får å få en fin sinus kurve. Man kan se i figur (H.5) hvordan utfallet ble.

Deretter ble man nødt til å finne ett område som kunne bli brukt til å beregne på sinuskurven. Området som var best for ett uttryk er sirklet rundt i figur H.5. Her ble det funnet verdiene for toppunktene og amplituden.



Figur H.5: Sinusgraf med punkter

Neste steg er å lage ett uttrykk og verifisering av uttrykket ved hjelp av GeoGebra.

$$Likevektslinje = (fmaks + fmin)/2 = (71,2 + 32,7)/2 = 51,95$$

$$Amplitude : A = 71,2 - 51,95 = 19,25$$

$$Periode : Tp = 25,1 - 19 = 6,1$$

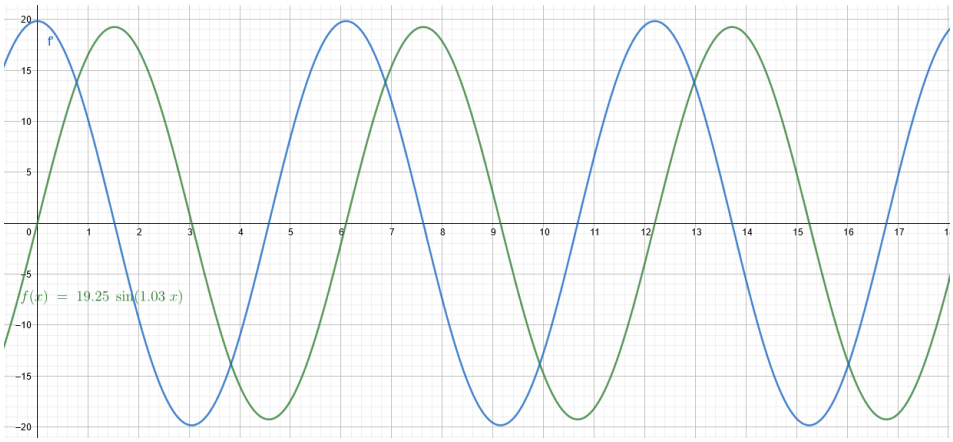
$$Frekvens : f = 1/Tp = 1/6,1$$

$$w = 2 * pi * f = 1,03$$

Signalet blir da:

$$19,25 * sin(1,03 * t). \tag{H.3}$$

Da kan man regne den deriverte ved å sette dette inn i GeoGebra, og deretter bruke den innebyge funksjonen til å derivere uttrykket. Deretter kunne man plotte funksjonene sammen og kom fram til figuren i figur H.6.



Figur H.6: Sinusgraf Geogebra

Fra figur H.6, kan man se at kurvene er forskøvet med litt under 2 sekund på x-aksen. Dette kan man sjekke opp med tidligere figuren i figur H.5, da ser man at det ikke er så stor forskjell fra teori til virkelighet.

I tillegg kan man sjekke opp amplituden, og ser at den deriverte amplituden skal ligge rett under 20. Deretter kan man sjekke det opp med figuren i H.5, og ser at amplituden skal ligge rundt 20, så dette stemmer meget overens.

Eventuelle avvik kan komme fra; tidsforsinkelser mellom kommunikasjon, det faktum at vi brukte direkte plotting / tegning, ujevne datamålinger,

det at man er nødt til å verifisere datamålingene i paint, etc.

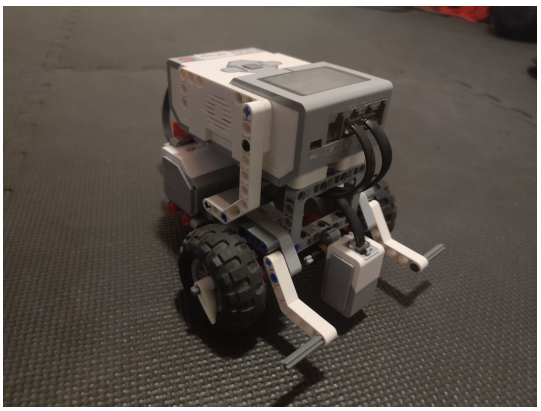
Vedlegg I

Manuell Kjøring

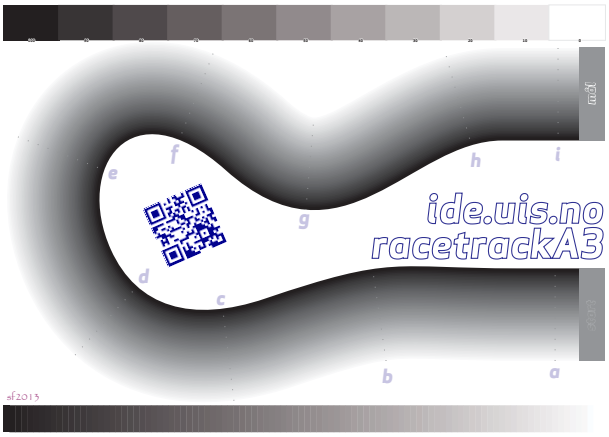
I.1 Problemstilling

Det siste av de obligatoriske prosjektene fra ING100 omhandler kjøring av roboten som vist i figur I.1 med to drivhjul, fritt roterende kule bak og lyssensor foran som peker ned i bakken. Man skal kjøre roboten rundt banen vist i midten av figur I.2, med mål om å holde lyssensoren på samme gråskalaverdi under hele kjøringen. Hensikten bak oppgaven er å gjøre målinger og beregne kvalitetsmål for å vurdere hvor bra målet ble oppnådd.

Det kan ofte være nyttig å ha konkrete tall på hvor bra en oppgave er utført, slike kvalitetsmål kan benyttes til å gjøre endringer for å forbedre f.eks effektivitet og presisjon. Spesifikt hva som måles og beregnes vil variere basert på hva som er viktig i de enkelte prosjekter som utføres. I dette eksempelet er det relevant å vurdere hvor man har kjørt i forhold til ønsket kjørelinje, og hvor mye innsats i form av motorpådrag man må bruke for å oppnå et gitt resultat. Motorpådraget kan brukes for å vurdere kostnaden av oppnåde kvalitetsmål, det kan f.eks være at det er å foretrekke å bruke litt mindre motorpådrag for å spare motorene og heller akseptere at avvik vil bli større.



Figur I.1: Robot satt opp til kjøring



Figur I.2: Bane for kjøring av LEGO-robot

I.2 Forslag til løsning

For å bedømme kvaliteten av kjøringen, brukes følgende kvalitetsmål:

- Avvik fra startverdi:

$$e = Referanse - Lys(k) \tag{I.1}$$

  Dette viser hvor langt fra man har vært i forhold til startverdien til enhver tid.

- Integral av absolutt avvik:

$$IAE = \int_0^t |e(t)| \tag{I.2}$$

  Her ser man det akkumulerte avviket over tid basert på absoluttverdien slik at positivt og negativt avvik vil gi samme resultat.

- Middelverdi av absolutt avvik:

$$MAE = \frac{1}{k} \sum_{n=1}^{k} |e(n)| \tag{I.3}$$

  Dette gir gjennomsnittet av absoluttverdien for avvik, i motsetning til IAE(I.2) kan verdien gitt av MAE(I.3) synke dersom kjøringen etter hvert forbedres.

- Total varians:

$$TV = \sum_{n=1}^{k} |u(n) - u(n-1)| \tag{I.4}$$

  Hvor u(n) er motorpådragssignalet. TV gir altså totalt endring i motorpådrag for hver motor.

Etter endt kjøring beregnes også middelverdi I.5 og standardavvik I.6 av lysmålingene gjort under kjøringen. For å gi videre innblikk i hvor jevnt man har kjørt blir alle kvalitetsmålene plottet mot tid (vist i figI.3).

$$\mu = \frac{1}{k}\sum_{n=1}^{k} x(n) \tag{I.5}$$

$$\sigma = \sqrt{\frac{1}{k}\sum_{n=1}^{k}(x(n) - \mu(n))^2} \tag{I.6}$$

Videre blir de forskjellige sjåførenes resultater lagt inn i en tabell (tabI.1 for sammenlikning. I tillegg plottes målte lysverdier i et histogram (figI.4) for hver sjåfør for å gi en mer visuell oversikt.

I.2.1 EV3main.py

For å få roboten til å kjøre må det hentes inn data fra styrestikken, dette skjer i kode 3.26. Verdien blir så skalert om fra styrestikkens maksimum og minimumsverdi til EV3-ens i kode 3.25. Linje 194 til 196 i kodeutdrag I.1 sjekker om lyssensoren har kommet utenfor banen, dette vurderes med at refleksonsverdien som måles er høy nok til å indikere hvit papir. Styrestikk posisjonen hentes på linje 198 og 199.

Kode I.1: Utdrag fra `GetNewMeasurement()` funksjonen i `EV3main.py`

```
192 light.append(myColorSensor.reflection())
193
194 if light[-1] > 70:
195     global joyMainSwitch
196     joyMainSwitch = True
197
198 joyForward.append(joyForwardInstance)
199 joySide.append(joySideInstance)
```

I `CalculateAndSetMotorPower()` funksjonen (kode I.2) beregnes motorpådraget for hver av de to motorene (linje 204-205) gitt ved en skalering av styrestikkeposisjon og variablene definert i linje 200-201, det beregnede pådraget legges i lister. Linje 208-209 setter pådraget for de to motorene til de siste beregnede verdiene.

Kode I.2: `CalculateAndSetMotorPower()` funksjonen i `EV3main.py`

```
196 def CalculateAndSetMotorPower(motorB, powerB, motorC, powerC,
197                                joyForward, joySide):
198
199     # Parametre for beregning til motorpådrag
200     a = 2
201     b = 1
202
203     # pådraget
204     powerB.append(joyForward[-1] * b + joySide[-1] * a)
205     powerC.append(joyForward[-1] * b - joySide[-1] * a)
206
207     # Sett hastigheten på motorene.
208     motorB.dc(powerB[-1])
209     motorC.dc(powerC[-1])
```

I.2.2 BeregnOgPlott.py

Før man kan begynne å regne ut kvalitetsmålene, må noen lister initialiseres. I kode I.3 initialiseres de nødvendige listene, og initialverdier blir satt der utregning av verdier er avhengig av tidligere resultater. Videre beregner funksjonen `MathCalculations()` (kode I.4) kvalitetsmålene, her er rekkefølgen av utregningene relevant da noen av de senere ligningene bruker utregnede verdier fra forekommende ligninger. Linje 58 finner avviket - e - som beskrevet i ligning I.1. Linjene 59 og 62-63 bruker `EulerForward()` funksjonen (kode G.1) til å regne ut IAE (ligning I.2) og TV (I.4) for begge motorene. MAE (ligning I.3) regnes ut i linje 60. Alle kvalitetsmålene lagres i lister for å kunne plotte dem mot tidsforløpet.

Kode I.3: Utdrag fra `BeregnOgPlott.py`

```
22 light = []
23 initLight = []
24 e = []
25 time = []
26 ts = [0]
27 powerB = [0]
28 powerC = [0]
29
30 IAE = [0]
31 MAE = [0]
32 TV_B = [0]
```

```
33 TV_C = [0]
```

Kode I.4: Utdrag fra `MathCalculations()` i `BeregnOgPlott.py`

```
58 e.append(light[0] - light[-1])
59 IAE.append(EulerForward(IAE[-1], abs(e[-1]), ts[-1]))
60 MAE.append(IAE[-1] / len(light))
61
62 TV_B.append(EulerForward(TV_B[-1], abs(powerB[-1] - ...
       powerB[-2]), ts[-1]))
63 TV_C.append(EulerForward(TV_C[-1], abs(powerC[-1] - ...
       powerC[-2]), ts[-1]))
```

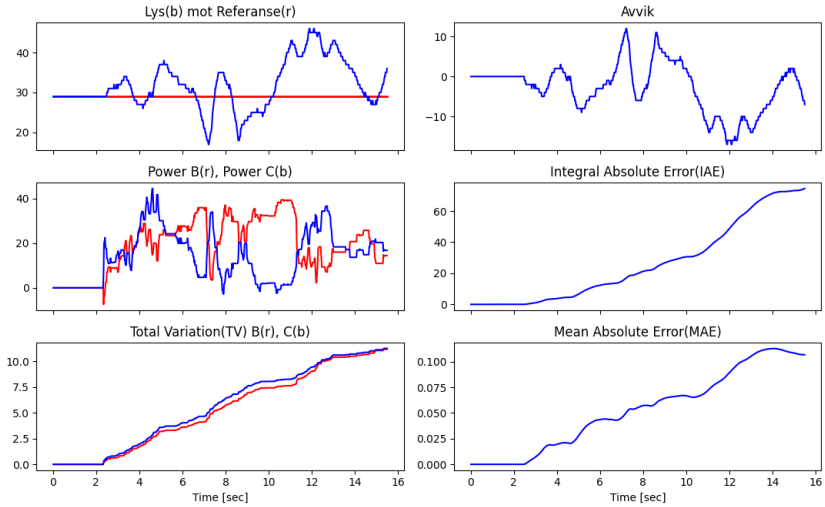
For å gjøre det enklere å lese de endelige kvalitetsmålene, skriver (kode I.5) verdiene som skal plasseres i tabell ut til konsollen. Middelverdi og standardavvik beregnes med funksjoner importert fra numpy[21] biblioteket.

Kode I.5: Utdrag fra `stopPlot()` i `BeregnOgPlott.py`

```
182 print("Referanse: ", light[0])
183 print("Middelverdi: {:.1f}".format(np.mean(light)))
184 print("|Referanse - middelverdi|: ...
        {:.1f}".format(abs(light[0] - np.mean(light))))
185 print("Standardavik: {:.1f}".format(np.std(light)))
186 print("Kjøretid [sek]: {:.1f}".format(time[-1]))
187 print("IAE: {:.1f}".format(IAE[-1]))
188 print("MAE: {:.1f}".format(MAE[-1]))
189 print("TV_B: {:.1f}".format(TV_B[-1]))
190 print("TV_C: {:.1f}".format(TV_C[-1]))
191 print("Middelverdi av Ts [sek]: {:.3f}".format(np.mean(ts)))
192 print("Antall målinger: ", len(light))
```

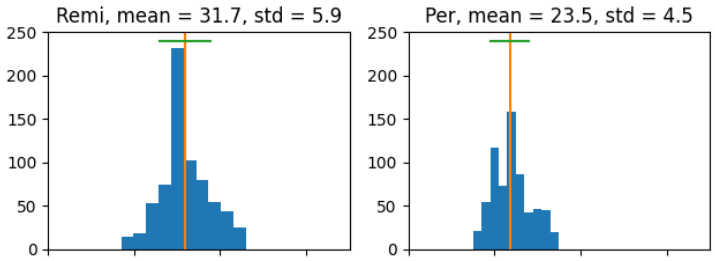
I.3 Resultat

Etter gjennomført kjøring er neste steg å vurdere hvor godt man har klart å kjøre ifølge de forskjellige kvalitetsmålene. Figur I.3 viser visuelt hvordan kvalitetsmålene utvikler seg iløpet av kjøringen, her kan man og prøve å lese av grafen hvor på banen man var til forskjellige tidspunkt.



Figur I.3: Plott av diverse kvalitetsmål

Figur I.4 viser histogram av lysverdien de forskjellige sjåførene samler inn mens de kjører, middelverdien og standardavviket er også plottet. Histogrammet vil vise hvor "jevnt"man har kjørt, altså hvor godt man har klart å holde seg til en spesifikk lysverdi. Middelverdien kan man sammenlikne med referansemålingen fra tabell I.1 for å se om man har klart å holde oss til den ønskede lysverdien. Tabell I.1 og figur I.4 brukes også til å sammenlikne hvor bra de individuelle sjåførene har kjørt.



Figur I.4: Histogram av målte lysverdier, med middelverdi og standardavvik

Tabell I.1: Tabell med kvalitetsmål for sammenlikning

	Remi	Per
Online/Offline	Offline	Offline
Referanse	29	23
middelverdi μ	31.7	23.5
$\lvert$*Referanse* - $\mu\rvert$	2.7	0.5
standardavvik σ	5.9	4.5
kjøretid [sek]	15.5	14.6
IAE	74.6	51.5
MAE	0.1	0.1
TV_B	11.2	14.9
TV_C	11.3	14.2
middelverdi av T_s [sek]	0.022	0.022
antall målinger (k)	700	665

For å komme til en endelig konklusjon på hvem som har kjørt best må en bestemme hvordan de forskjellige kvalitetsmålene skal vektlegges, om man legger mer vekt på presisjon, hastighet, minst mulig motorpådrag osv. Forskjellige prosjekter vil ha forskjellige mål og begrensninger, man kan for eksempel velge at man er villige til å ha mye variasjon i motorpådrag (TV I.4) og dermed måtte erstatte motorer oftere for å oppnå et bedre resultat. Et slikt tilfelle vil indikere at Per er en bedre sjåfør selv om han pådrar seg en del mer TV (I.4) siden han på alle kvalitetsmål slår Remi.

Vedlegg J

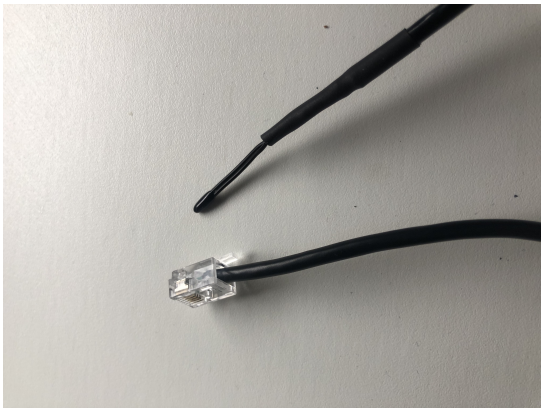
Termometer - sensor

J.1 Introduksjon av sensoren

Det ble etterspurt av veileder om man kunne undersøke en termometer-sensor som ble laget av professor Ståle Freyer fra instituttet for Data og Elektro på UiS. Formålet med sensoren er at den skal kunne bli brukt i dette nye faget når det blir aktivt fra januar 2022 av. Det finnes en offisiell Mindstorms - sensor, men denne gikk det desverre ikke an å bestille lenger. Derfor ble det behov for å lage denne lokalt på UiS.

Sensoren skal kunne bli brukt sammen med Mindstorms-roboten for å utføre matematiske forsøk, f.eks. dynamikkberegning med kald/varm kaffekopp, sjekke forskjellige temperaturer, verifisering av temperatur i kjøleskap, osv.

Denne sensoren er loddet sammen med en RJ-11 tilkobling, en kabel og så selve sensoren i enden. Figurene (J.1 og J.2) viser hvordan kabelen er satt sammen. Kabelen blir koblet til roboten, og klarer å samle inn data fra sensoren, og videreføre data til roboten.



Figur J.1: Endepunktene på sensoren

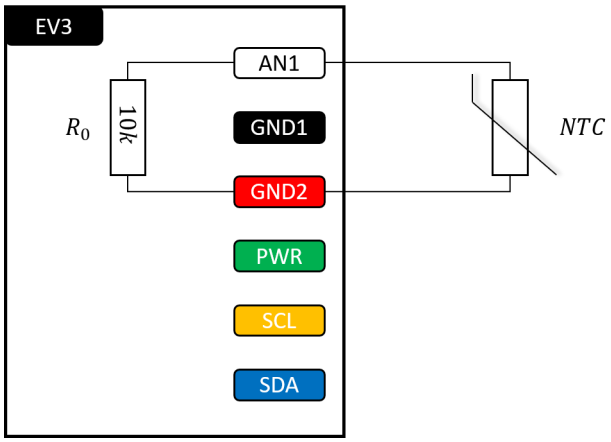
Figur J.2: Hele kabelen

J.1.1 Teori om sensor

Data som sensoren sender videre er i volt, etter det er det en kode som kjører på roboten som beregner verdien over til celsius. Denne koden blir forklart dypere inn i neste delkapittel.

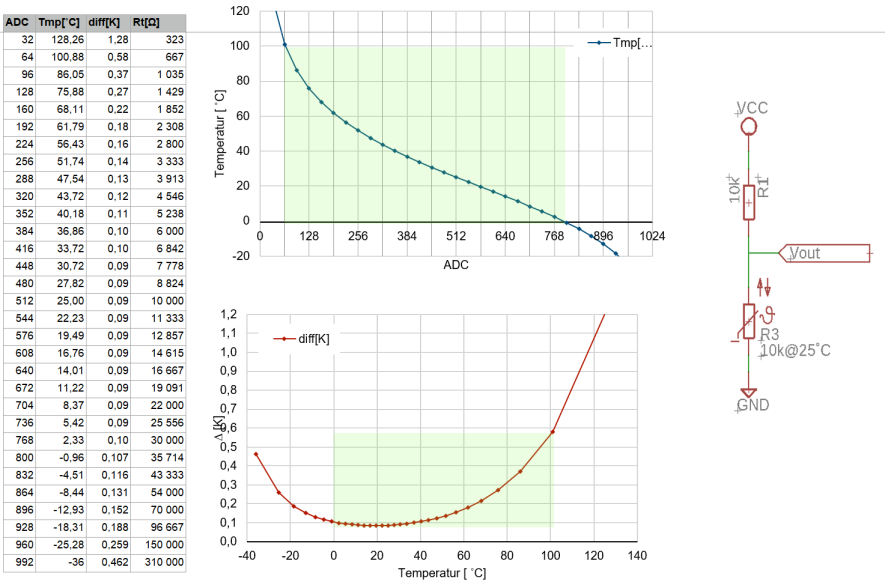
I figur J.3 vises det hvordan RJ-11 oppkoblingen er oppbygd. Her ligger det en motstand som øker volten om det blir en lavere temperatur. Dette fører til at ADC-verdien er omvendt fra temperatur, ved at temperaturen synker

når ADC-verdien øker.



Figur J.3: Tilkobling til robot (Illustrert av professor Ståle Freyer)

I figur J.4 kan man se de forskjellige verdiene, og hvilken temperatur som hører til en gitt ADC-verdi. Denne oversikten ble laget av instituttet for Data og Elektro. Oversikten forklarer at temperaturstigningen ikke er lineær i forhold til en gitt ADC-verdi. Stigningstallet er ujevnt og unøyaktig, helt frem til rundt romtemperatur hvor ΔK ligger på rundt 1, som er tilnærmet lineært.



ADC	Tmp[˚C]	diff[K]	Rt[Ω]
32	128,26	1,28	323
64	100,88	0,58	667
96	86,05	0,37	1 035
128	75,88	0,27	1 429
160	68,11	0,22	1 852
192	61,79	0,18	2 308
224	56,43	0,16	2 800
256	51,74	0,14	3 333
288	47,54	0,13	3 913
320	43,72	0,12	4 546
352	40,18	0,11	5 238
384	36,86	0,10	6 000
416	33,72	0,10	6 842
448	30,72	0,09	7 778
480	27,82	0,09	8 824
512	25,00	0,09	10 000
544	22,23	0,09	11 333
576	19,49	0,09	12 857
608	16,76	0,09	14 615
640	14,01	0,09	16 667
672	11,22	0,09	19 091
704	8,37	0,09	22 000
736	5,42	0,09	25 556
768	2,33	0,10	30 000
800	-0,96	0,107	35 714
832	-4,51	0,116	43 333
864	-8,44	0,131	54 000
896	-12,93	0,152	70 000
928	-18,31	0,188	96 667
960	-25,28	0,259	150 000
992	-36	0,462	310 000

Figur J.4: Excel ark som viser ulike verdier. (Laget av professor Ståle Freyer)

J.1.2 Kode til sensor

I lag med kabelen følgte det også med Python-kode som ble brukt for testing. Koden ble laget av professor Ståle Freyer. Denne koden lager en enkel oppkobling med roboten slik at roboten klarer å avlese verdier og beregne volt-verdiene fra sensoren til temperaturverdier i celsius. Denne temperaturen blir printet ut på displayet på EV3en og i konsollvinduet på datamaskinen som er koblet til EV3en.

Denne koden fungerte å kjøre når en mindstormsrobot var koblet inn i pcen. Denne roboten trengte å ha micro-python innstallert. Når disse kravene er utført kan man bruke visual studio code til å lage en sammenkobling mellom robot og pc. Deretter kan man kjøre koden ved å trykke på F5", og deretter trykker på ev3-roboten, da blir koden kjørt og innformasjon printet ut.

I de første linjene i kodeutdraget J.1 blir de forskjellige modulene som er i bruk importert. Her kan man se med en gang at koden importerer en annen grunnbasis for ev3, som er ev3dev2. I Prosjekt0X importerer man pybricks sin egen modul.

Kode J.1: Koden for å teste temperatur-sensor(linje 1-10)

```
1 #!/usr/bin/env python3
2
3 from math import log
4 import time as time
5 from sys import stderr
6
7 from ev3dev2.port import LegoPort
8 from ev3dev2.sensor import INPUT_4
9 from ev3dev2.sensor.lego import Sensor
```

De neste linjene i kodeutdraget J.2 definerer en ny klasse kalt Ntc(). Dette er klassen som lagrer og beregner data fra sensoren. Her blir det også definert visse variabler som blir brukt i beregninger.

Kode J.2: Koden for å teste temperatur-sensor (linje 11-21)

```
11 class ntc():
12
13     def __init__(self):
14         self.beta = 3984.0
15         self.To = 25.0
16         self.Ro = 10.0e3
17         self.Rt = None
18         self._maxADC = None
19         self._bitsADC = None
20         self.bitsADC(10)
```

I kodeutdraget J.3 blir det definert to metoder som hjelper med ADC-beregninger. BitsADC() hjelper med å beregne ADC-verdien ifra bits, samtidig som maxADC() skal sette en max-verdi som begrenser det mulige utfallet ADC-verdien kan være.

Kode J.3: Koden for å teste temperatur-sensor (linje 22-31)

```
22     def bitsADC(self, newval=None):
23         if newval != None:
24             self._bitsADC = newval
25             self._maxADC = 2 ** self._bitsADC
26         return self._bitsADC
27
28     def maxADC(self, newval=None):
29         if newval != None:
```

```
30              self._maxADC = newval
31          return self._maxADC
```

I kodeutdraget J.4 har man to nye metoder. Den første er en formel som skal beregne temperaturen fra ADC-verdien om til Temperatur i Celsius. tempC() - metodden setter de variablene som skal brukes, også bruker dem inn i steinhardt-formelen. Til slutt blir det returnert en variabel som forteller hvilken temperatur man har fått ut basert på innsatte verdier.

Kode J.4: Koden for å teste temperatur-sensor (linje 33-41)

```
33      def _steinhart_temperature_C_(self, r):
34          steinhart = log(r / self.Ro) / self.beta      # ...
                log(R/Ro) / beta
35          steinhart += 1.0 / (self.To + 273.15)         # ...
                log(R/Ro) / beta + 1/To
36          steinhart = (1.0 / steinhart) - 273.15   # Invert, ...
                convert to C
37          return steinhart
38
39      def temp_C(self, ADC: [int, float]):
40          self.Rt = self.Ro / ((self._maxADC / ADC - 1))
41          return self._steinhart_temperature_C_(self.Rt)
```

I kodeutdraget J.5 så definerer man hvordan sensoren skal bli brukt. Man definerer ett objekt av klasse ntc() i linje 44, og definerer de variablene som brukes i linje 45 og 46. Variablene er krevd for at beregningene skal være korrekte. Etter dette blir selve sensoren definert. Linje 49-54 definerer port som skal brukes (port4), og hvilken type sensor (nxt-analog) som er koblet inn i denne porten.

Kode J.5: Koden for å teste temperatur-sensor (linje 43-54)

```
43 #Definerer NTC parametere:
44 a = ntc()
45 a.beta = 3984
46 a.maxADC(5000)
47
48 #Definerer port 4 til å være analog inngang:
49 p4 = LegoPort(INPUT_4)
50 p4.mode = 'nxt-analog'
51 p4.set_device = 'nxt-analog'
52
```

```
53 #Definerer at temperatursensoren er koblet til port 4:
54 ntc_sensor = Sensor(INPUT_4)
```

Til slutt blir selve koden J.6 utført. Mens koden blir kjørt (while True), så blir sensorverdien hentet, verdien blir deretter beregnet til temperatur, og så printet ut på konsoll og robot. Til slutt er det en time.sleep() lagt inn på 1 sekund, denne kan bli fjernet om man ikke skal printe verdiene.Linjen er satt inn slik at konsoll ikke skal bli overfylt av linjer.

Kode J.6: Koden for å teste temperatur-sensor (linje 55-68)

```
55 while True:
56
57     #Leser av sensorverdi i volt:
58     msr = ntc_sensor.value()
59
60     #Konverterer til temperatur:
61     t = a.temp_C(msr)
62
63     # Printer temperaturen til EV3 LCD skjerm
64     print('The temperature is: %f deg' %t)
65     # Printer temperaturen til VS Code output panel
66     print('The temperature is: %f deg' %t, file=stderr)
67
68     time.sleep(1)
```

J.2 Problemstilling

Målet med denne sensoren er å kunne sjekke om det var en mulighet å få til en oppkobling mellom sensoren og vår kode i Project0X. Importmodulen som ble brukt i test-filen ovenfor er en helt annen modul enn det ble brukt i koden hos Project0X. Ev3dev2 har de mulighetene til å sette en sensor som en analog sensor, denne muligheten ble ikke desverre ikke funnet i undersøkelse inni Pybricks sin modul. Det var mulig å sette hvilken sensor som skal bli satt på de ulike portene. Pybricks hadde ikke mulighet til å sette en sensor som ikke er direkte fra Lego-settet. Det gjorde det umulig å finne tilpasning på å bruke denne sensoren i lag med resten av koden.

Deretter prøvde man å kjøre begge modulene parallelt med hverandre i samme kode (ev3main.py), men dette ble også mislykket. Siden vår kode kjører på pybricks-micropython, og koden som testfilen kjører på har python3 som første linje J.7. Dette førte til at man ikke kunne kombinere de modulene man hadde pga de er grunnlagd på to forskjellige Python-enviroments.

Kode J.7: Første linje i hver fil som definerer python enviroment.

```
1 #ev3main.py
2 #!/usr/bin/env pybricks-micropython
3 #test fil for sensor
4 #!/usr/bin/env python3
```

Deretter ble det undersøkt om det var en mulighet for å plotte data direkte fra testfila. Siden det gikk fint å printe ut dataene i konsollen, var dette en indikasjon på at plotting også fungerer. Dette lyktes desverre heller ikke fordi Matplotlib sin modul ikke ble funnet, og det er fordi det ikke er mulig å importere Matplotlib på selve roboten.

J.3 Resultat

Selv om det ikke ble funnet ønsket resultat, kan fortsatt test-koden bli brukt som en grunnbasis for ett prosjekt som skal bli utført med denne sensoren. Det vil fortsatt være mulig å gjøre målinger, lagre data i en `measureements.txt`-fil, og gjøre verifiseringer av data som blir printet ut mens man holder på med ett forsøk. Dette kan fort bli tungt i lengden, men det fører til en økt kunnskap i matte og fysikk, som studentene kan ta med seg videre.

Vedlegg K

GitHub-repository

Her er link for å komme inn på GitHuben der alle filene og eksterne vedlegg ligger. En liste over alle filene som ligger inne i repositoriet:

- Prosjektbeskrivelse.pdf - Denne beskriver alle prosjektene i MatLab, og brukes til å gi oss en forklaring på prosjektene.
- Prosjekt0X - med bruk av Lister (Besluttet til å bli brukt i aalle de andre prosjektene)
- Prosjekt0X - med bruk av Dictionary
- Prosjekt01 - Numerisk Integrasjon
- Prosjekt02 - Filtrering
- Prosjekt03 - Numerisk derivasjon
- Prosjekt04 - Manuell kjøring
- Prosjekt05 - Temperatur-sensor

Klikk her for å komme til Github-repositoriet