ELE130 Anvendt matematikk og fysikk i robotprogrammering

Universitetet i Stavanger

PROSJEKT - VÅREN 2024

Prosjekt-oppgaven	LEGO Mindstorms og MATLAB/Python i skjønn forening

Gruppenummer	2423		
Gruppens medlemmer	Navn	Studentnummer	Bilde
	Marie Elisabeth Herland	267842	
	Ole Martin Voster	272039	
	Kristoffer Holen	271449	
	Mikal Fredriksen	280082	

Innhold

Innhold **ii**

Sammendrag **vii**

1 Numerisk integrasjon (utført av hele gruppen) **1**

1.1 Introduksjon 1

1.2 Teori 2

1.3 Kode for numerisk integrasjon 4

1.4 Relasjon til det bestemte integral som en tallverdi ved hjelp av legoutstyr 5

1.5 Relasjon til det bestemte integral som et analytisk uttrykk 9

1.5.1 Forsøk 1 14

1.5.2 Forsøk 2 14

2 Filtrering (utført av hele gruppen) **17**

2.1 Introduksjon 17

2.2 Teori 18

2.2.1 FIR-filter 19

2.2.2 IIR-filter 20

2.2.3 Kode for FIR-filter og IIR-filter 21

2.3 Filtrere målt kaffetemperatur 22

2.3.1 Forsøk med ulike verdier av M i et FIR-filter 24

2.3.2 Betydningen av α i et IRR-filter 25

2.4 Filtrering av målestøy i en temperaturtransmitter 26

2.5 Filtrering av lydsignal 30

3 Numerisk derivasjon (utført av Ole Martin og Marie) 31

3.1 Introduksjon 31

3.2 Teori 32

3.3 Kode for bakoverderivasjon 33

3.4 Fartsmåling med laserpistol 34

3.5 Relasjon til den deriverte som funksjonsuttrykk 35

3.6 Betydningen av endret frekvens ω 40

4 PID regulatoren (utført av hele gruppen) 42

4.1 Introduksjon 42

4.2 Teori 43

4.3 Kode for PID-regulator 45

4.4 Regulering av PID-regulator 47

4.4.1 Kalibrering av P-ledd (proporsjonal) 47

4.4.2 Kalibrering av I-ledd (integral) 50

4.4.3 Kalibrering av D-ledd (derivativ) 53

4.4.4 Samlet regulering 54

5 Manuell kjøring av Lego-robot (utført av hele gruppen) 56

5.1 Introduksjon 56

5.2 Teori 58

5.3 Kode for manuell kjøring 60

5.4 Resultater fra forsøket 61

6 Kjøring med turtallsregulator (utført av Ole Martin og Marie) 66

6.1 Introduksjon 66

6.2 Teori 67

6.3 Kode 68

6.4 Resultat 69

7 Kjøring med PID (utført av Ole Martin, Marie og Mikal) 71

7.1 Introduksjon 71

7.2 Teori 71

7.3 Kode for regulator og automatisk kjøring 72

7.4 Tuning av regulatorparametre og grunnfart 73

7.5 Resultat 73

8 Hengende last (utført av Ole Martin) 76

8.1 Introduksjon 76

8.2 Teori 78

8.3 Utdrag av kode 78

8.4 Gjennomføring og resultat 79

8.4.1 Oppsummert 82

9 Estimert hastighet (utført av Ole Martin og Mikal) 83

9.1 Introduksjon 83

9.2 Teori 84

9.3 Matlab-kode 84

9.4 Gjennomføring og resultat 85

9.4.1 Forsøk nr. 1 85

9.4.2 Forsøk nr. 2 87

9.4.3 Forsøk nr. 3 88

10 Konklusjon 91

Bibliografi **93**

Vedlegg **93**

A Timelister **94**

Sammendrag

Formålet med denne rapporten er å utforske og beskrive sentrale teorier innen robotprogrammering, samt å gjøre en praktisk anvendelse av disse teoriene i ulike forsøk. Rapporten dekker teori for temaene som numerisk integrasjon, numerisk derivasjon, filtrering, og PID-regulering. Disse teoriene anvendes i praktiske forsøk gjennom programmering i MATLAB av LEGO-utstyr.

Gjennom rapporten presenteres en rekke praktiske forsøk, deriblant simulering av attraksjonen "Stupet" i Kongeparken, automatisert kjøring og styring av hengende last. De første fire kapitlene fokuserer på en detaljert teoretisk forklaring av teori, hvor forsøkene relaterer til disse teoriene. De påfølgende kapitlene, fra 5 til 9, gjør vi flere forsøk der vi anvender teori fra de 5 første.

I forsøkene har vi både resultater som bekrefter teoriene, f.eks forsøkene på integrasjon, og resultater der vi ser at modellen fra teorien i videre forsøk kunne vært justert, f. eks innen derivasjon. Denne rapporten gir dermed både innsikt i teoriens styrker og dens grenser når den anvendes i praktiske scenarier.

Kapittel 1

Numerisk integrasjon (utført av hele gruppen)

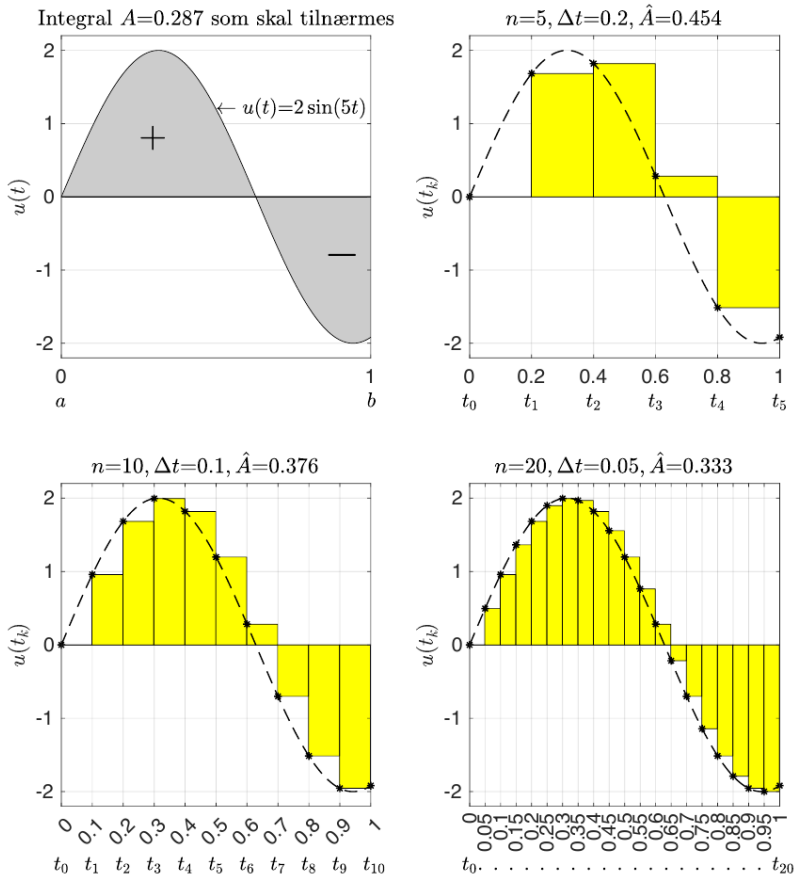
1.1 Introduksjon

Integrasjon av en funksjon er en matematisk beregning som anvendes overalt i verden, fra industrien til våre daglige liv. Gitt en funksjon, for eksempel volumstrøm som en funksjon av tid, vil integralet av funksjonen være lik arealet under grafen. I eksempelet med volumstrøm, vil arealet tilsvare det totale volumet for hele tidsperioden. Det er dette prinsippet en *flowmåler* i en bensinpumpe bygger på når den måler liter drivstoff ved fylling [2]. Et annet eksempel er hurtigladere til elbiler som oppgir en levert effekt ved lading. Numerisk integrasjon brukes også ved beregning av renter eller til å simulere væskedynamikk i rørledninger.

Senere i kapittelet skal vi se nærmere på prinsippene bak numerisk integrasjon og gjennomgå et par forsøk vi har utført for å illustrere numerisk integrasjon i praksis. Kapittelet tar først for seg den teoretiske bakgrunnen før vi beskriver forsøkene som ble gjort; ett hvor det ble simulert påfylling og drikking fra et vannglass, og ett hvor det ble simulert et stempel som presser vann inn i en ballong. På denne måten får vi demonstert og verifisert koden vår for numerisk integrasjon, som vil bli brukt videre i prosjektet, og sammenlignet resultatene med de analytiske uttrykkene. I tillegg undersøker vi også relasjoner mellom et sinussignal og dens numerisk integrerte.

1.2 Teori

I matematikken forutsetter integrasjon at vi kjenner en matematisk funksjon hvor vi kan utføre integrasjonen. I praksis er det derimot sjeldent at man kjenner funksjonsuttrykket som skal integreres. I stedet registrerer vi en rekke relevante målinger gitt som et tidsseriesignal, $u(t_k)$, og utfører numerisk integrasjon på dette signalet. Numerisk integrasjon av et tidsseriesignal gir et estimat av arealet under grafen til tidsseriesignalet, og dermed et estimat av den informasjonen vi er ute etter. Forenklet kan vi si at den integrerte er summen av arealet til n antall rektangler med bredde Δt og høyde $u(t_k)$ (funksjonsverdien); nøyaktigheten er dermed avhengig av antall rektangler. Dette ser vi i figur 1.1, hvor arealestimatet $\hat{A}$ kommer nærmere og nærmere det sanne arealet A med økende antall n.



Figur 1.1: Illustrasjon av prinsippet bak Eulers foroverintegrasjon. Figuren er hentet fra [2].

1.2 Teori

For å beregne det estimerte arealet i sanntid, har det blitt utviklet ulike metoder som benytter forrige eller fremtidige arealestimat og målinger til å beregne det nye arealestimatet. Beregningene utføres altså *rekursivt*. Vi har valgt å fokusere på Eulers bakovermetode, videre referert til som bakoverintegrasjon. To andre kjente metoder er Eulers foroverintegrasjon og trapesintegrasjon.

Gitt tidsskrittet T_s, som tilsvarer Δt, og indeks k i stedet for tidspunktet t_k, er integralet y_k av tidsseriesignalet u_k definert som følgende ved bakoverintegrasjon [2]:

$$y_k = y_{k-1} + T_s \cdot u_k \quad \forall k = 2, \ldots, n, \quad \text{gitt } y_1 \tag{1.1}$$

Bakoverintegrasjon gir altså det nåværende arealestimatet som det forrige arealestimatet summert med produktet av nåværende måling og tidsskrittet.

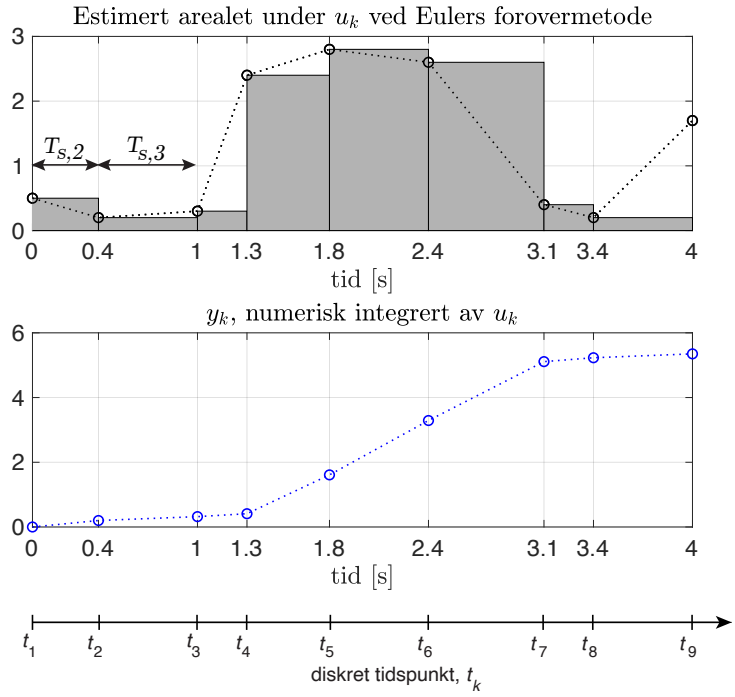
På grunn av forsinkelser i overføring av data mellom enheter, pågående beregninger eller andre forstyrrelser, vil tidsskrittet T_s variere for hver måling, som vist i figur 1.2. Bakoverintegrasjon antar et konstant tidsskritt og må dermed modifiseres for å kunne gi riktig resultat ved numerisk integrasjon. Det modifiserte tidsskrittet kan uttrykkes som:

$$T_{s,k} = t_k - t_{k-1} \tag{1.2}$$

Ligningene 1.2 og 1.1 kan da kombineres til:

$$\boxed{y_k = y_{k-1} + T_{s,k} \cdot u_k \quad \forall k = 2, \ldots, n, \quad \text{gitt } y_1} \tag{1.3}$$

Dette er ligningene vi tar utgangspunkt i videre. Det første tidsskrittet, $T_{s,1}$, settes til en nominell verdi slik at tidsskritt-vektoren blir like lang som de andre vektorene, ettersom ligning 1.2 ikke kan beregnes for $k = 1$.



Figur 1.2: Prinsippet for numerisk integrasjon av tidseriedata, her vist med Eulers forovermetode (foroverintegrasjon). Eulers bakovermetode (bakoverintegrasjon) følger tilsvarende prinsipp. For hver ny måling u_k legges det inn et rektangel med bredde $T_{s,k}$, og totalarealet av rektanglene tilsvarer integralverdien. Figuren er hentet fra [1].

1.3 Kode for numerisk integrasjon

Vi har valgt å bruke Matlab for implementering av kode og behandling av dataene fra forsøkene. Funksjonen for bakoverintegrasjon er vist i kodeutdrag 1.1.

Kode 1.1: Funksjon for bakoverintegrasjon brukt i prosjektet.

```
function IntValueNew = EulerBackward(IntValueOld, Timestep, ...
    FunctionValue)

    IntValueNew = IntValueOld + Timestep * FunctionValue;

end
```

Funksjonen beregner den nye integrerte verdien fra uttrykket i ligning 1.3; den gamle verdien summeres med produktet mellom tidsskrittet og nåværende funksjonsverdi. Funksjonen kalles som vist i kodeutdrag 1.2. Merk if-løkken som sørger for at funksjonen følger forbeholdet i ligning 1.3, dvs. $\forall k = 2, \ldots, n$.

Kode 1.2: Funksjonskall for bakoverintegrasjon.

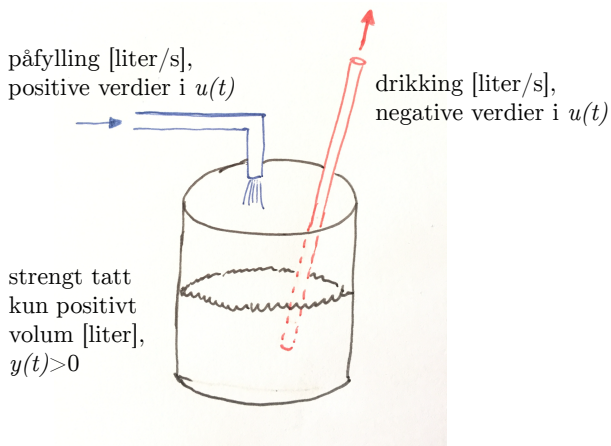
```
if k > 1
    y(k) = EulerBackward(y(k-1), Ts(k), u(k));
end
```

1.4 Relasjon til det bestemte integral som en tallverdi ved hjelp av legoutstyr

For å samle inn data til forsøkene ble LEGO MINDSTORMS EV3 benyttet. Dette er et programmerbart robotikksett med mulighet for tilkobling til motor, styrestikke og ulike sensorer, som fargesensor/lyssensor og ultralydsensor. Ved å koble EV3-enheten til en datamaskin, kan man programmere enheten i Python eller Matlab og registrere og behandle sensordata. Som nevnt tidligere har vi valgt å benytte Matlab.

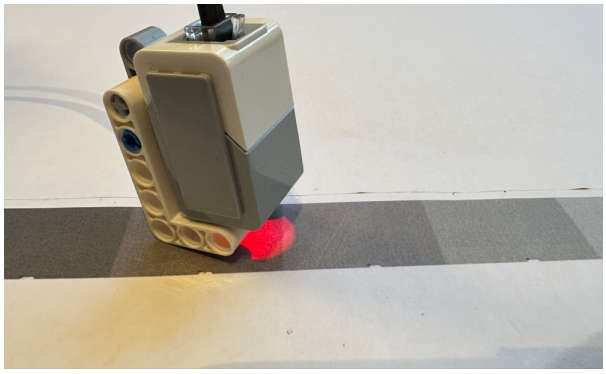
Som et praktisk eksempel på numerisk integrasjon, simulerte vi påfylling og tapping (drikking) av et vannglass (se figur 1.3). På denne måten fikk vi laget et tidsseriesignal som hadde tilnærmede konstante verdier for fylling og tapping, og brukte funksjonen for bakoverintegrasjon (kodeutdrag 1.1) til å estimere volumet i vannglasset.

1.4 Relasjon til det bestemte integral som en tallverdi ved hjelp av legoutstyr



Figur 1.3: Konseptuell skisse av et vannglass hvor det fylles på med vann og tappes for vann (ved å drikke gjennom et sugerør). Påfylling innebærer positive verdier på $u(t)$, mens drikking innebærer negative verdier. Vannvolumet i glasset tilsvarer den integrerte av $u(t)$, $y(t)$. Figuren er hentet fra [1].

For å simulere selve påfyllingen og tappingen, benyttet vi et ark med en gråskala og en lyssensor, som vist i figur 1.4. Ved å kjapt flytte sensoren til et mørkere eller lysere punkt på gråskalaen og holde sensoren der en stund, simulerte vi henholdsvis tapping og påfylling med en tilnærmet konstant verdi for hver.



Figur 1.4: Lyssensoren brukt i forsøket for å utføre numerisk integrasjon på et simulert scenario med fylling og tapping av vann fra et glass. Sensoren er koblet opp til en LEGO MINDSTORMS EV3-enhet og registrerer lysintensiteten fra en gråskala.

Et nøytral punkt, `nullflow`, ble satt til den første målingen registrert midt på skalaen, som vist i kodeutdrag 1.3. Dette punktet representerer en tilstand uten

endring i strømning, et nullpunkt. Ved å trekke denne verdien fra måleverdiene fra lyssensoren (`Lys(k)`), fikk vi positive verdier for påfylling og negative verdier for tapping og dermed det ønskede tidsseriesignalet vårt, strømning med tid.

Kode 1.3: Kode for beregning av strømning, $u(k)$, fra måleverdier fra lyssensoren, `Lys(k)`, minus det nøytrale punktet `nullflow`.

```
nullflow = Lys(1);
u(k) = Lys(k) - nullflow;
```

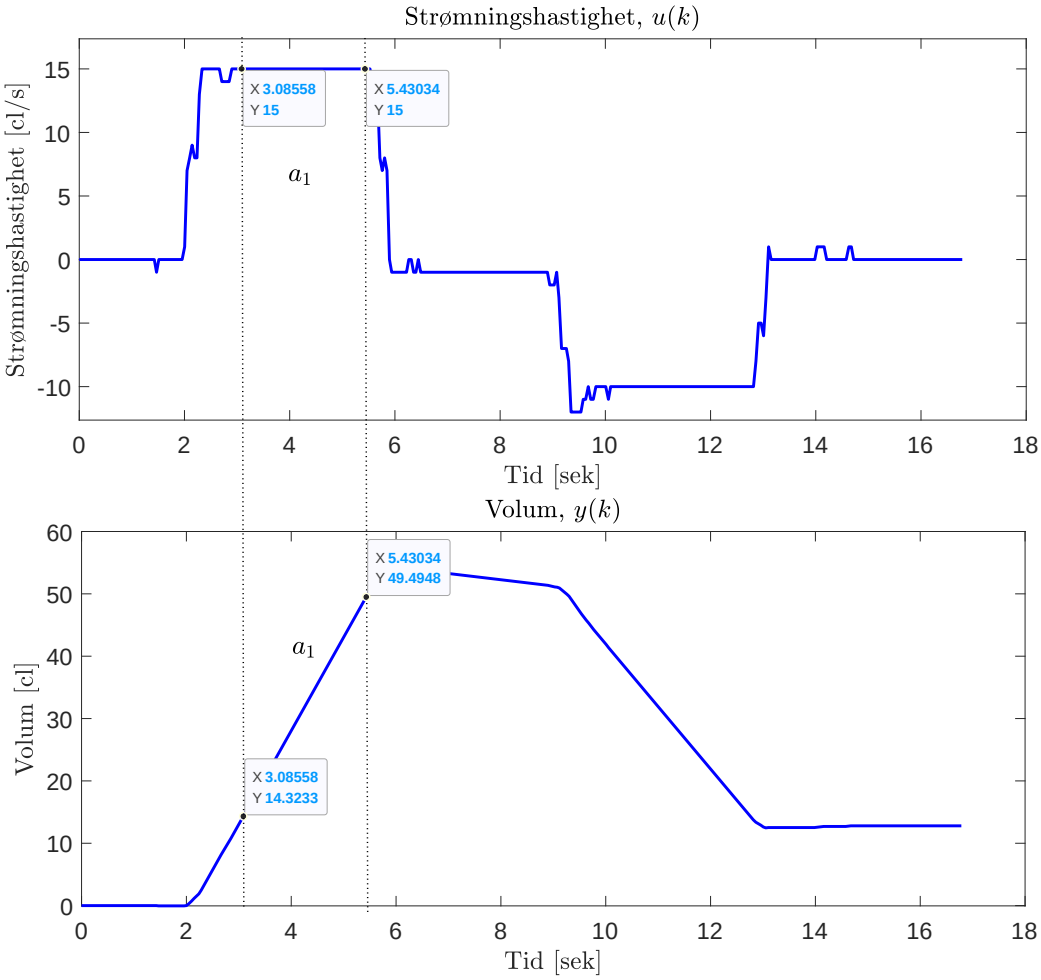
Ettersom tidsskrittet varierer, konverterte vi den registerte tidsvektoren T_k til $T_{s,k}$ basert på ligning 1.2 som vist i kodeutdrag 1.4. Det variable tidsskrittet, `Ts(k)` i koden, ble så benyttet for å beregne det estimerte volumet med funksjonen i kodeutdrag 1.1 og funksjonskallet i kodeutdrag 1.2.

Kode 1.4: Kode for beregning av det variable tidsskrittet, $T_{s,k}$ fra den registrerte tidsvektoren `Tid(k)`.

```
if k > 1
    Ts(k) = Tid(k) - Tid(k-1);
end
```

Resultatet av forsøket er vist i figur 1.5. Den øverste grafen viser strømningshastighet $u(k)$. Den nederste grafen viser volumet $y(k)$ estimert med bakoverintegrasjon.

1.4 Relasjon til det bestemte integral som en tallverdi ved hjelp av legoutstyr



Figur 1.5: Resultatet fra forsøket. Den øverste grafen viser måleverdiene for simulering av påfylling (positive verdier) og tapping/drikking (negative verdier) fra vannglasset. Den nederste grafen viser volumet beregnet med bakoverintegrasjon. Verdiene i området a_1 brukes for å verifisere koden for integrasjon.

For å verifisere koden vår sammenligner vi den numerisk integrerte verdien fra koden vår med analytisk utregning. Vi antar at fylling og tapping skjer med en konstant verdi, en rimelig antakelse når vi ser på kurven til $u(k)$ i figur 1.5, og fokuserer på fylling, indikert med a_1 i figuren. Vi ser i figuren at for strømningshastighet blir $a_1 = 15.0$ cl/s for tidsintervallet $t_0 = 3.09$ s til $t_1 = 5.43$ s. Vi kan da regne den analytiske løsningen for arealet under grafen:

$$
\begin{aligned}
A_1 = \int_{t_0}^{t_1} u(t)dt &= \int_{t_0}^{t_1} a_1 dt \\
&= \int_{3.09}^{5.43} 15.0\, dt \\
&= 15.0\,\mathrm{cl/s} \cdot (5.43 - 3.09)\,\mathrm{s} \\
&= 15.0\,\mathrm{cl/s} \cdot 2.34\,\mathrm{s} \\
&= 35.10\,\mathrm{cl}
\end{aligned} \tag{1.4}
$$

Nå har vi en fasit for å sjekke at koden vår utfører numerisk integrasjon riktig. For å verifisere dette bruker vi fundamentalteoremet. Ved bruk av fundamentalteoremet skal vi komme frem til det samme svaret vi fikk i analytisk utregning, bare at vi nå bruker den alleredet integrerte verdien i grafen for volum ved bruk av samme tidspunkter som vi brukte i den analytiske utregningen $t_0 = 3.09$ s til $t_1 = 5.43$ i figur 1.5:

$$
\begin{aligned}
\int_{t_0}^{t_1} u(t)dt &= y(t_1) - y(t_0) \\
&= y(5.43) - y(3.09) \\
&= 49.49\,\mathrm{cl} - 14.32\,\mathrm{cl} \\
&= 35.17\,\mathrm{cl}
\end{aligned} \tag{1.5}
$$

Vi får tilnærmet like svar i ligning 1.4 og 1.5. Vi har dermed verifisert koden vår og vil benytte den videre der det kreves numerisk integrasjon.

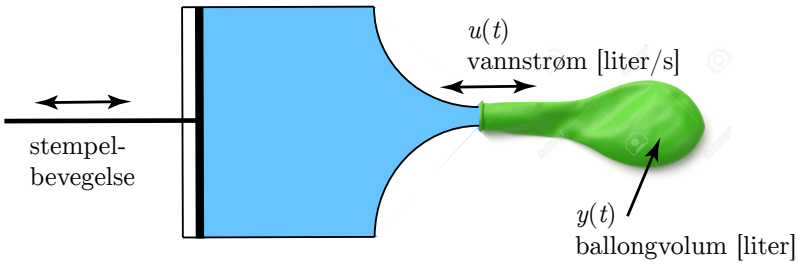
1.5 Relasjon til det bestemte integral som et analytisk uttrykk

I virkeligheten er fenomener sjeldent så rett fram som vår simulering av fylling og tapping av vannglasset med konstant $u(k)$. I stedet er flere fenomener sykliske. For å teste implementeringen vår av numerisk integrasjon på et mer reelt fenomen og utforske ulike relasjoner mellom et sinussignal og den integrerte av signalet, simulerte vi en pumpe med et stempel som pumper vann inn og ut av en ballong

med en regelmessig rytme, som vist i figur 1.6. Sinussignalet for vannstrømmen $u(k)$ blir da

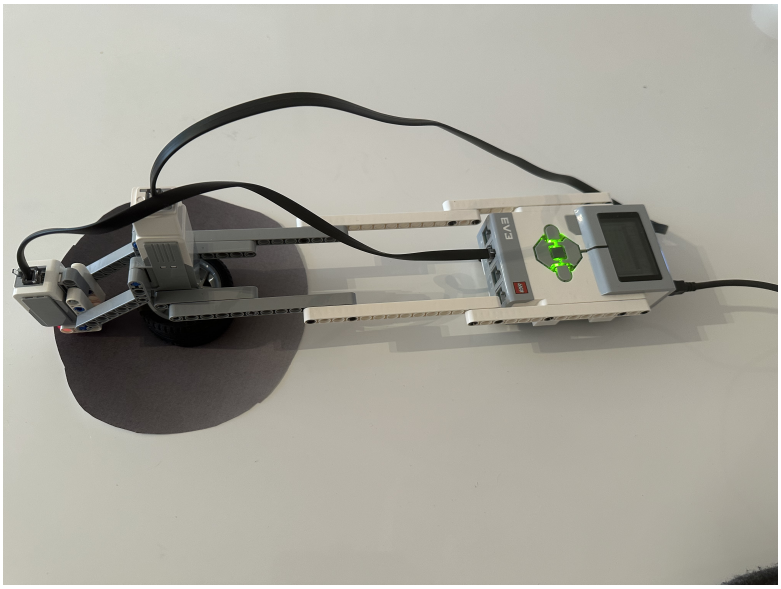
$$u(t) = U \cdot \sin(\omega \cdot t) \tag{1.6}$$

hvor U er amplituden til signalet og ω er vinkelhastigheten. Vi benyttet igjen en gråskala (denne gangen skalert opp til et A4-ark) og lyssensoren for å lage signalet.



Figur 1.6: Konseptuell illustrasjon av en pumpe med stempel som pumper vann inn og ut av en ballong. Figuren er hentet fra [1].

For å simulere vannstrømmen med en så konstant U og ω som mulig, monterte vi lyssensoren til et gummihjul som igjen var tilkoblet en motor. Dermed kunne vi bruke gummihjulet til å rotere arket under sensoren og dermed oppnå mer konstant verdi for U og ω enn hvis vi hadde rotert lyssensoren for hånd. Vi startet med lyssensoren midt i mellom gråskala-verdiene for å sørge for at grafen skuller starte med bunnpunktet som betyr at stempelet står åpent og ballongen er tom for vann. Lego-konstruksjonen er vist i figur 1.7.



Figur 1.7: Lego-konstruksjon for å lage et sinussignal. Et gummihjul tilkoblet en motor sørger for å rotere gråskala-arket under konstruksjonen. Lyssensoren registrerer da en syklisk variasjon mellom lyse og mørke toner.

For å finne volumet i ballongen $y(t)$ kan vi regne integralet av vannstrømmen. Uttrykket for vannstrømmen kan uttrykkes som $u(t) = U\sin(\omega t)$, hvor U er amplituden og ω er vinkelhastigheten, da kan vi uttrykke det integrerte signalet $y(t)$ som:

$$y(t) = \int_0^t u(\tau)\,d\tau + y(0) = \int_0^t U\sin(\omega\tau)\,d\tau + y(0)$$

Dette integreres videre til:

$$y(t) = -\frac{U}{\omega}\cos(\omega t) + \frac{U}{\omega} + y(0)$$

hvor $\frac{U}{\omega}$ representerer den nye amplituden, og vi har lagt til $y(0)$ for å inkludere initialbetingelsen. Vi kan forenkle og skrive dette som:

$$y(t) = -Y\cos(\omega t) + y_A$$

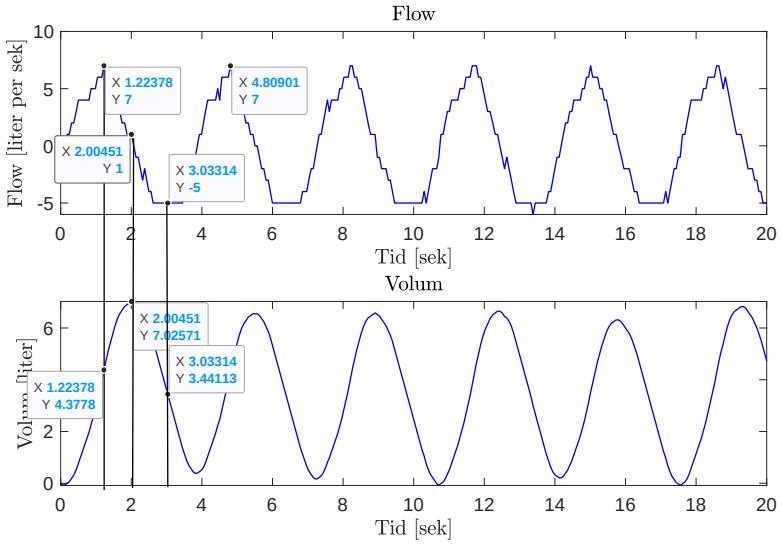
hvor $Y = \frac{U}{\omega}$ og $y_A = \frac{U}{\omega} + y(0)$.

Siden vi nå får en cosinusfunksjon når vi integrerer sinus må vi inkludere en faseforskyvning $\varphi = -\frac{\pi}{2}$ radianer, dette gjør at vi igjen får reflektert et sinussignal som er det vi er ute etter:

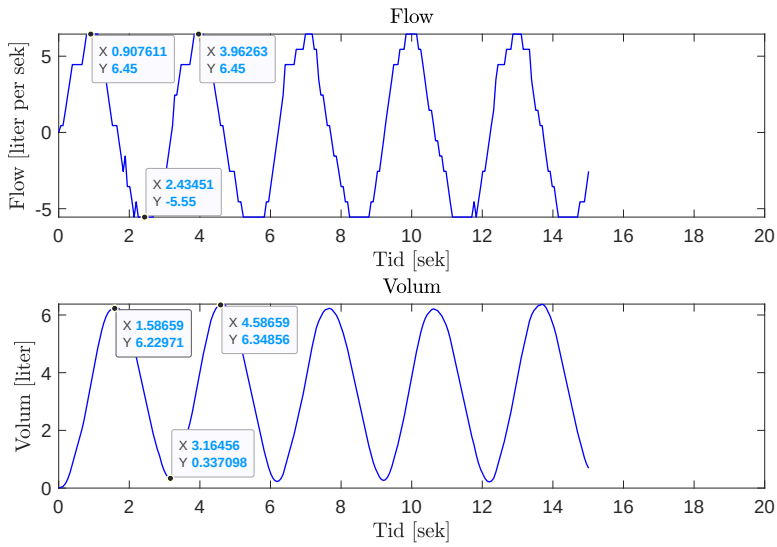
$$y(t) = Y\sin(\omega t + \varphi) + y_A \tag{1.7}$$

Faseforskyvningen er ofte referert til som faseforverrende eller faseforringende.

Vi vil sammenligne to pumpeforsøk med ulik hastighet på pumpene som fyller og tømmer ballongen:



Figur 1.8: Dataene fra forsøket er justert slik at volum og flow starter på 0. Det blir lettere å se at det er en pumpe som fyller ballongen opp og tømmer den.



Figur 1.9: Samme forsøk som i figur 1.8, men med høyere frekvens

Vi ser fra avlesninger fra figurene at:

- Første forsøk, figur 1.8:
  Flow amplituden $U = 7$ liter/s
  Tidsperioden $Tp = 4.81 - 1.22 \approx 3.6$

- Andre forsøk, figur 1.9:
  Flow amplituden $U = 6.45$ liter/s
  Tidsperioden $Tp = 3.96 - 0.91 \approx 3.1$

Vinkelhastigheten ω for hvert forsøk beregnes som:

$$\omega_1 = \frac{2\pi}{Tp} = \frac{2\pi}{3.6}$$

$$\omega_2 = \frac{2\pi}{Tp} = \frac{2\pi}{3.1}$$

Amplituden Y:

$$Y_1 = \frac{U}{\omega_1} = \frac{7}{\omega_1} = 4.01$$

$$Y_2 = \frac{U}{\omega_2} = \frac{6.45}{\omega_2} = 3.18$$

Dette viser at en økning i frekvens (reduksjon av perioden T) fører til økt ω, og en redusert amplitude Y, hvilket er konsistent med de observerte lavere volumene i det raskere andre forsøket.

1.5.1 Forsøk 1

- Beregnet amplitude $Y_1 = \frac{U}{\omega} = \frac{7}{\frac{2\pi}{3.6}} = 4.01\,\text{liter}$
- Beregnet vinkelhastighet $\omega_1 = \frac{2\pi}{Tp} = \frac{2\pi}{3.6}$
- Likevektsverdi $y_A = \frac{U}{\omega} + y(0) = \frac{7}{\frac{2\pi}{3.6}} = 4.01\,\text{liter}$
- Faseforskyvning $\varphi = -\frac{\pi}{2}$ radianer (-90 grader)

$$y_1(t) = 4.01 \sin\left(\frac{2\pi}{3.6}t - \frac{\pi}{2}\right) + 4.01$$

Ved $t = 2$ sekunder i det første forsøket, blir responsen:

$$y_1(2) = 4.01 \sin\left(\frac{2\pi}{3.6} \times 2 - \frac{\pi}{2}\right) + 4.01 = 7.78\,\text{liter}$$

1.5.2 Forsøk 2

- Beregnet amplitude $Y_2 = \frac{U}{\omega} = \frac{7}{\frac{2\pi}{3.6}} = 3.18\,\text{liter}$
- Beregnet vinkelhastighet $\omega_1 = \frac{2\pi}{Tp} = \frac{2\pi}{3.1}$
- Likevektsverdi $y_A = \frac{U}{\omega} + y(0) = \frac{7}{\frac{2\pi}{3.1}} = 3.18\,\text{liter}$

- Faseforskyvning $\varphi = -\frac{\pi}{2}$ radianer (-90 grader)

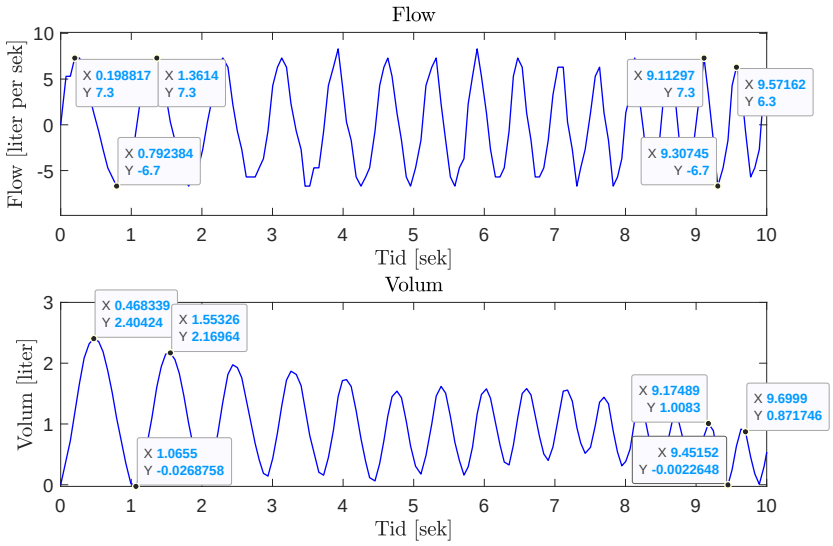
$$y_2(t) = 3.18 \sin\left(\frac{2\pi}{3.1}t - \frac{\pi}{2}\right) + 3.18$$

Ved $t = 1.6$ sekunder i det andre forsøket, blir responsen:

$$y_2(1.6) = 3.18 \sin\left(\frac{2\pi}{3.1} \times 1.6 - \frac{\pi}{2}\right) + 3.18 = 6.34\,\text{liter}$$

Diagram og integral viser ikke nøyaktig samme volum, spesielt ikke i forsøk 1 (7.0 l mot 7.8 l), men vi anser presisjonsnivået i forsøkene til å være akseptabelt for illustrere at en økt frekvens på pumpen vil gi et redusert volum i ballongen. Vi får også demonstrert effekten av faseforskyvningen.

Vi gjør et nytt forsøk med chirpsignal (økende frekvens):



Figur 1.10: Målingene i dette forsøket er gjort med økende fart

Utfra de tidligere forsøkene og teorien overfor forventer vi at at maksvolumet skal være synkende med økende frekvens. Vi ser forsøket med chirp-signalet, se figur

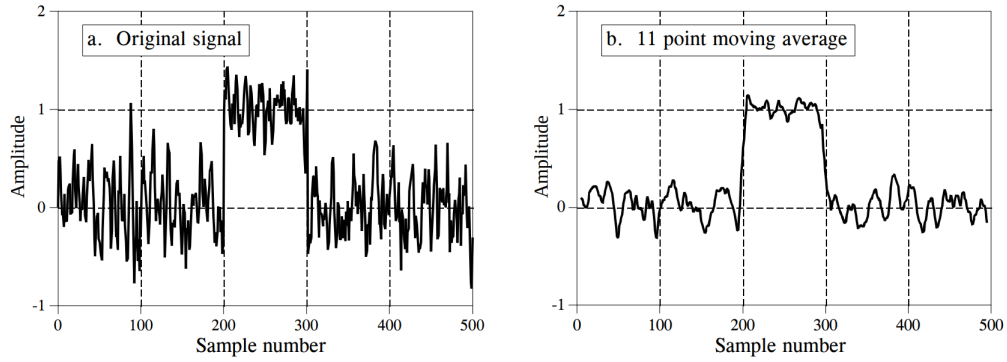
1.10, her ser vi at etterhvert som frekvensen øker i signalet for flow, og vi integrer det så får vi et signal som representerer volum hvor amplituden minker etterhvert som frekvensen øker, samt at signalet får en liten forskyvning i likevektslinjen til volum signalet. Dette kan vi se i likning 1.5, hvor Y er grunnen til at amplituden blir mindre desto høyere frekvensen, ettersom at $Y = \frac{U}{\omega}$. Forskyvningen vi får på likevektslinjen ser vi i y_A, y_A kan skrives som $\frac{U}{\omega} + y(0)$, dette gjør at likevektslinjen endrer seg også i forbindelse med endring i frekvensen. I tillegg så får vi en faseforskyvning, $\varphi = -\frac{\pi}{2}$, dette fordi flow signalet er et sinus signal, og når et sinus signal blir integrert får vi et negativt cosinus signal, dette skriver vi om til $sin(\omega t + \varphi)$

Kapittel 2

Filtrering (utført av hele gruppen)

2.1 Introduksjon

Digitale filtre, i likhet med numerisk integrasjon, benyttes overalt for å manipulere eller forbedre et signal. Et enkelt eksempel kan være å ta et lydsignal og fjerne målestøy eller bakgrunnsstøy, som vist i figur 2.1. Digitale filtre brukes også til å fjerne uønskede frekvenser (over eller under et visst nivå), fremheve viktig informasjon som trender og nivåer, og separere individuelle komponenter i et komplekst signal [2]. Foruten lydsignaler finner vi digitale filtre i hverdagslige ting som termometre og digitale vekter, verktøy som lasermålere og multimetere, samt i bildebehandling.

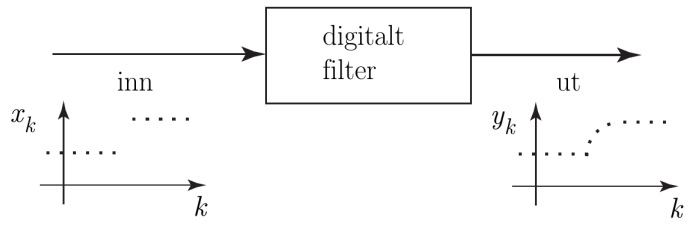


Figur 2.1: Eksempel på et signal som behandles med et digitalt filter, i dette tilfellet med et MA-filter (*Moving Average filter*). Filteret jevner tydelig ut støy i det originale signalet. Figuren er hentet fra [3].

I dette kapittelet tar vi for oss to vanlig brukte filtre, FIR-filter og IIR-filter, og tre simulerte scenario for å se nærmere på hvordan filtrering virker i praksis. Vi undersøker også ulike forhold mellom parametrene i de to filtrene. Vi går først gjennom det matematiske grunnlaget bak filtrene og hvordan dette kan oversettes til kode. Deretter beskriver og diskuterer vi forsøkene vi gjorde for å simulere de tre scenarioene; ett hvor vi simulerte et termometer som måler temperaturen i en kaffekopp, ett som simulerte målinger fra en industriell temperaturtransmitter, og ett hvor vi filtrerte et lydsignal. Dermed får vi sett på både prinsippene bak digital filtrering og nytten med filtrering ved f.eks målestøy.

2.2 Teori

Forenklet tar et digitalt filter et inngangssignal, x_k, og matematisk konverterer det til et endret utgangssignal, y_k, som illustrert i figur 2.2. På den måten kan vi dele opp et signal, fjerne uønskede verdier og/eller fremheve deler av signalet.



Figur 2.2: Illustrasjon av prinsippet bak digitale filtre. Et digitalt filter tar et inngangssignal, x_k, og konverterer det til et endret utgangssignal, y_k. Figuren er hentet fra [2].

Det finnes flere ulike filtre ut fra hva den ønskede effekten er. Som nevnt innledningvis er FIR-filter og IIR-filter to filtre som blir mye brukt. Et FIR-filter er et ikke-rekursivt filter hvor impulsresponsen er endelig. Varianten av FIR-filte som vi benytter her kalles glidende midlingsfilter (*Moving Average*-filter, *MA*-filter), og er spesielt nyttig der man ønsker å fjerne støy, men samtidig beholde en tydelig sprangrespons [3]. Denne effekten ser vi i figur 2.1. Et IIR-filter, imens, er rekursivt, har en uendelig impulsrespons (responsen blir aldri lik null) og kalles gjerne eksponentielt vektet glidende midlingsfilter (*Exponentially Weighted Moving Average*-filter, *EWMA*-filter) [2].

2.2.1 FIR-filter

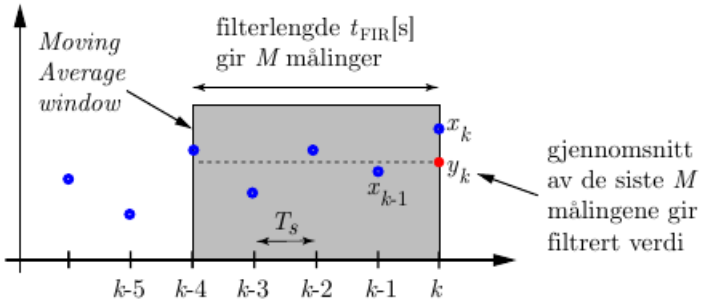
FIR-filter står for *Finite Impulse Response filter* og har, som nevnt, en endelig impulsrespons; det vil si, ved en kort impuls (en endring i signalet) vil til slutt responsen bli null. I FIR-filter vektes hver måleverdi med en filterkoeffisient. Dersom vi lar alle måleverdiene, x_{k-n}, vektes likt og filterkoeffisientene summeres til 1, får vi MA-filteret [2]:

$$y_k = \frac{1}{M} \sum_{n=0}^{M-1} x_{k-n} \tag{2.1}$$

hvor filterets lengde eller antall målinger, M, er gitt ved tidsperioden t_{FIR} og samplingstiden T_s:

$$M = \frac{t_{\mathrm{FIR}}}{T_s} + 1 \tag{2.2}$$

Et MA-filter vil altså glidende beregne gjennomsnittet av M antall målinger fra inngangssignalet x_k. To eksempler på et MA-filter er vist i figur 2.1 og 2.3. I figur 2.3 ser vi også filtervinduet, den grå boksen (i figuren) som indikerer lengden på filteret og hvilke målinger som inkluderes.



Figur 2.3: Eksempel på et MA-filter med lengde $M = 5$. Her er antall målinger lik lengden M gitt samplingstiden T_s og tidsperioden t_{FIR}. Den grå boksen som indikerer lengden på filteret og hvilke målinger som inkluderes, kalles filtervinduet. Figuren er hentet fra [1].

2.2.2 IIR-filter

IIR-filter står for *Infinite Impulse Response filter* og har, i motsetning til FIR-filteret, en uendelig impulsrespons; det vil si, ved en kort impuls vil responsen aldri bli null. Et generelt IIR-filter består av en rekursiv del, som benytter tidligere utgangsverdier i beregningen, og en ikke-rekursiv del, som benytter tidligere inngangsverdier i beregningen. Utifra antall filterkoeffisienter i den rekursive delen, kan IIR-filtre settes til ulike ordner (førsteordens, andreordens, etc). Vi benytter et førsteordens IIR-filter i dette prosjektet. Matematisk er filteret gitt ved [2]:

$$y_k = (1 - \alpha) \cdot y_{k-1} + \alpha \cdot x_k \tag{2.3}$$

I dette filteret er filterkoeffisientene til utgangssignalet og inngangssignalet samlet til ett parameter, α. Dermed justerer man kun det ene parameteret, som videre kan beregnes fra samplingstid T_s og en ønsket hastighet på filteret gitt av tidskonstanten τ:

$$\alpha = \frac{T_s}{\tau + T_s} \tag{2.4}$$

For at filteret skal være stabilt, må $0 \leq \alpha \leq 1$; for eksempel vil $\alpha > 1$ føre til et stigende utgangssignal som til slutt spinner ut av kontroll [3].

2.2.3 Kode for FIR-filter og IIR-filter

Kodeutdrag 2.1 under viser implementeringen av FIR-filteret i Matlab. Fordi M kan være større enn antall målinger k for de første målingene, settes $M = k$ så lenge $M > k$. Deretter beregnes den filtrerte verdien for utgangssignalet y_k slik beskrevet i ligning 2.1.

Kode 2.1: Matlab-funksjon for FIR-filter brukt i forsøkene.

```
function [FilteredValue] = FIR_filter(Measurement, M)
    k = length(Measurement);

    if k < M
        M = k;
    end

    FilteredValue = sum(Measurement(end-M+1:end)) / M;
end
```

Implementeringen av IIR-filteret i Matlab er vist i kodeutdrag 2.2 under. Funksjonen beregner den filtrerte verdien til utgangssignalet y_k slik beskrevet i ligning 2.3.

Kode 2.2: Matlab-funksjon for førsteordens IIR-filter brukt i forsøkene.

```
function [FilteredValue] = IIR_filter(OldFilteredValue, ...
    Measurement, Parameter)

    FilteredValue = (1 - Parameter) * OldFilteredValue + ...
        Parameter * Measurement;

end
```

Funksjonene kan så kalles som vist i kodeutdrag 2.3.

Kode 2.3: Funksjonskall for FIR-filter og IIR-filter.

```
for k = 1:length(Lys)
    y_FIR(k) = FIR_filter(Lys(1:k), M);

    if k == 1
        y_IIR(k) = Lys(k);
    else
        y_IIR(k) = IIR_filter(y_IIR(k-1), Lys(k), alfa);
    end
end
```

Som en del av forsøkene, lagde vi signaler med tilfeldig støy for å undersøke effekten av filtrering. Støyen ble tillagt målingene som vist i kodeutdrag 2.4. Tilsvarende ble gjort for et forsøk hvor vi filtrerte lyd, vist i kodeutdrag 2.5.

Kode 2.4: Kode for tilfeldig støy.

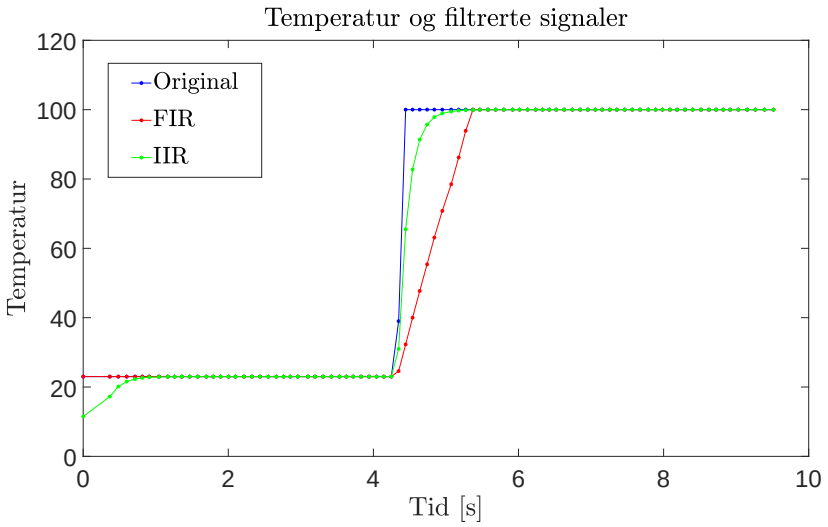
```
%Kode for å lage tilfeldig støy
Lys(k) =
double(readLightIntensity(myColorSensor,'reflected')) + randn;
```

Kode 2.5: Utdrag fra kode for filtrert lyd.

```
%Lyssignal som tillegges støy og en konstant på 100 for å få ...
    lyden i riktig frekvensområde
    rawLys = readLightIntensity(myColorSensor, 'reflected');
    x(k) = rawLys + randn + 100;

%Lyden filtreres når trykksensoren trykkes inn
if read(myTouchSensor) == 1
    y_IIR(k) = IIR_filter(y_IIR(k-1), x(k), alfa);
    playTone(mylego, y_IIR(k)*10, 10, 50);
```

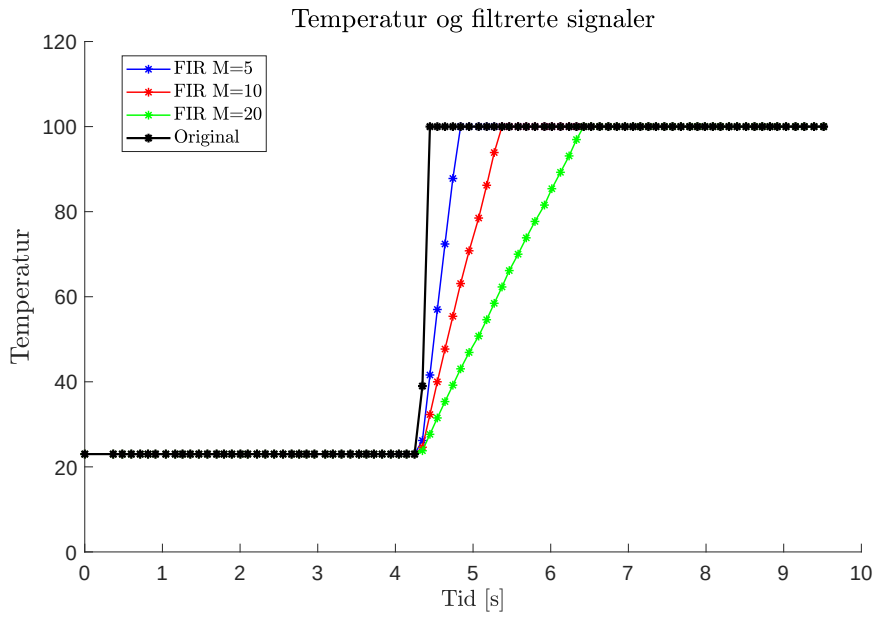
2.3 Filtrere målt kaffetemperatur

I figur 2.4 ser vi forskjellene mellom de to filtrene, der FIR-filteret gir en skarpere og mer umiddelbar respons på signalendringer. Da vil signalet kunne beholde mer av det opprinnelige signalets form. IIR-filteret en jevnere overgang. Dette gjør IIR-filteret ofte vil være mer effektivt til å utjevne støy over tid.



Figur 2.4: Når temperaturen endres brått ser vi effekten for de to ulike filtertypene

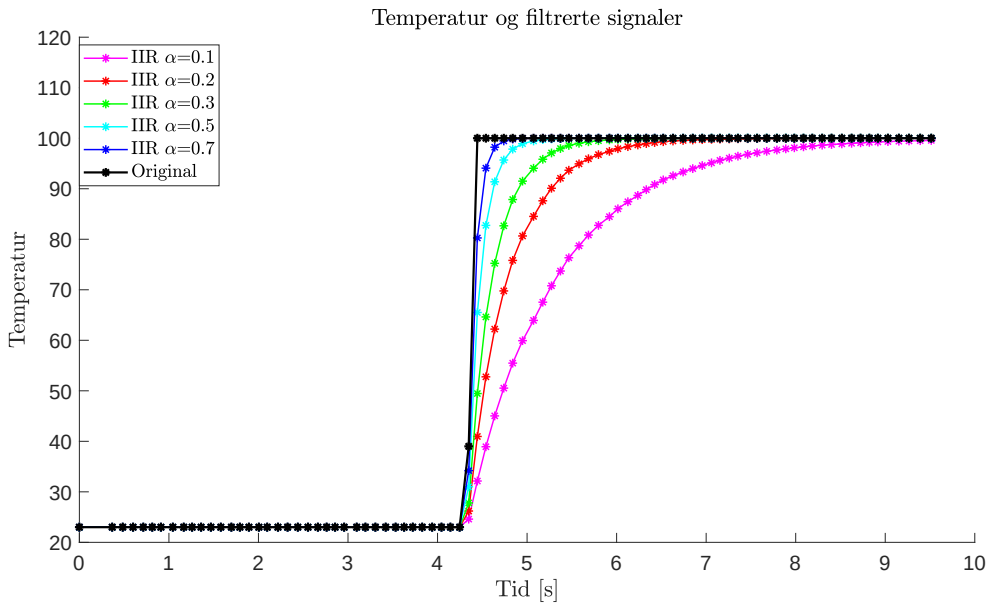
2.3.1 Forsøk med ulike verdier av M i et FIR-filter



Figur 2.5: Filtrerte signal med ulike M-verdier

I figur 2.5 har vi gjort et forsøk med å filtrere 'temperatur'-endringen med ulike verdier av M. Her kan vi se at ved en høyere M verdi, altså desto flere målinger som blir tatt med i filtreringen desto 'glattere' blir det filtrerte signalet. Dette går derimot på bekostning av responsen til signalet, desto mer filtrert signalet er desto tregere blir responsen. Dette kan vi se i figur 2.5 på forskjellen mellom den grønne grafen hvor $M = 20$ og den blå grafen hvor $M = 5$. Man kan se at den blå grafen har en brattere stigning enn den grønne, dette er på grunn av forskjellen i M-verdien.

2.3.2 Betydningen av α i et IRR-filter



Figur 2.6: Filtrerte signal med ulike α-verdier

Vi gjorde et nytt forsøk med IIR-filteret hvor vi brukte forskjellige verdier for α for å vise effekten α har på filteret. Ved å justere α, ser vi hvordan filteret reagerer ulikt på endringer i inngangssignalet, som vist i figur 2.6. En lavere α gir et "glattere" filtrert signal med mindre respons på raske endringer, mens en høyere α gir en raskerefilter som følger inngangssignalets endringer bedre. Dette kan forklares ut i fra utrykket gitt i ligning 2.3. I første ledd i likningen har vi:

$$(1 - \alpha) \cdot y_{k-1}$$

Her kan vi se at desto høyere α er desto mindre "aktivt" vil denne delen av likningen spille inn, det motsatte gjelder for lavere α verdier. Desto lavere α er desto mer vil dette leddet spille inn. La oss se på den andre delen av likningen.
I andre del av likningen har vi:

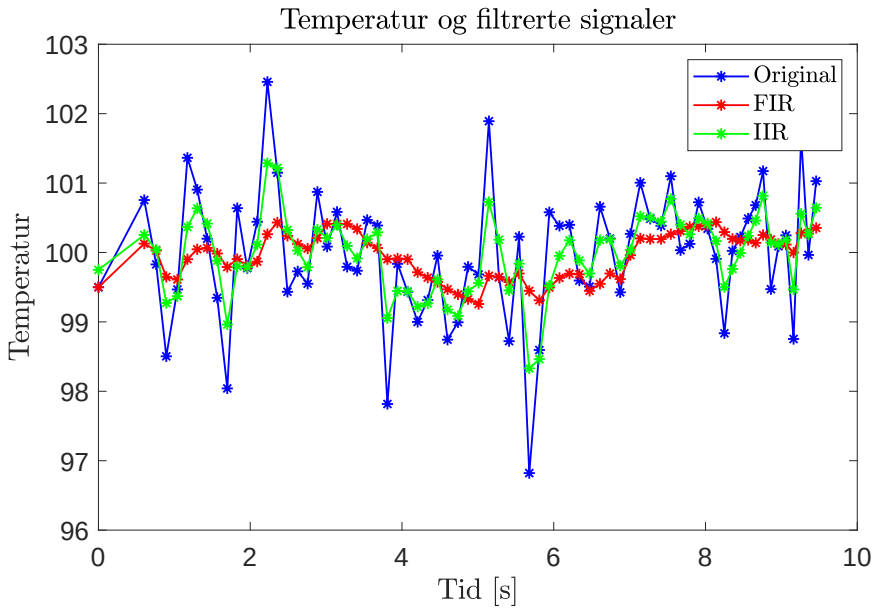
$$\alpha \cdot x_k$$

Her er det litt motsatt av første del; desto høyere α er her desto mer vil inngangsignalet komme gjennom, desto lavere α er desto mindre del av inngangsignalet komme gjennom, altså signalet blir filtrert mer.

2.4 Filtrering av målestøy i en temperaturtransmitter

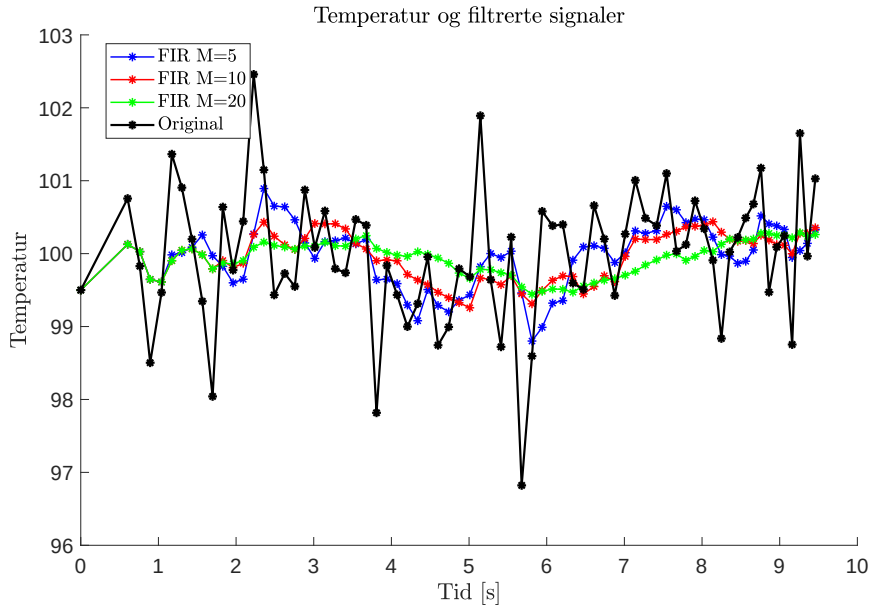
I dette forsøket så vi på hvordan vi kan simulere signalene som mates inn i en industriell temperaturtransmitter, ved å benytte lysintensitet. Endringen i temperatur simuleres ved å raskt flytte lyssensoren fra mørk overfalte til lys overflate. Endringen i fra en stabil temperatur til en ny stabil temperatur gir en sprangrespons, og vil vil i de neste forsøkene se på hvordan sprangresponsen blir når vi bruker FIR-filter og IRR-filter.

I de neste forsøkene har vi i utgangspunktet et jevnt signal som vi har tillagt støy ved hjelp av koding i matlab, se kodeutdrag 2.4. Nedenfor i figur 2.7 viser signalet tillagt støy som er filtrert med FIR-filter og IRR-filter.



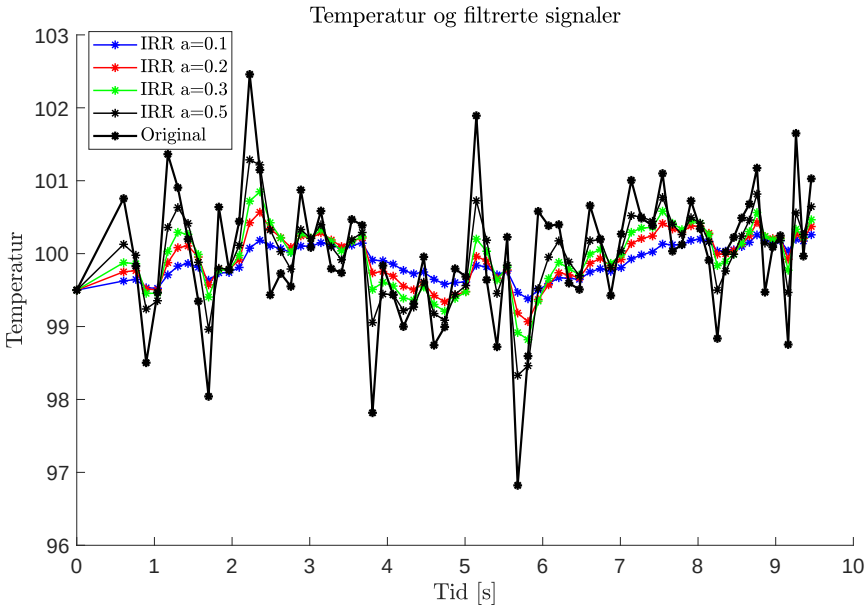
Figur 2.7: Signal med tillagt støy. Signalet er deretter filtrert med både IRR-filter og FIR-filter

I det neste forsøket filtrerer vi signalet med ulike verdier av M. Som forventet ser vi i figur 2.8 at en høy M gir et glattere signal med tregere respons sammelignet med signalene som er filtrert med en lav M.



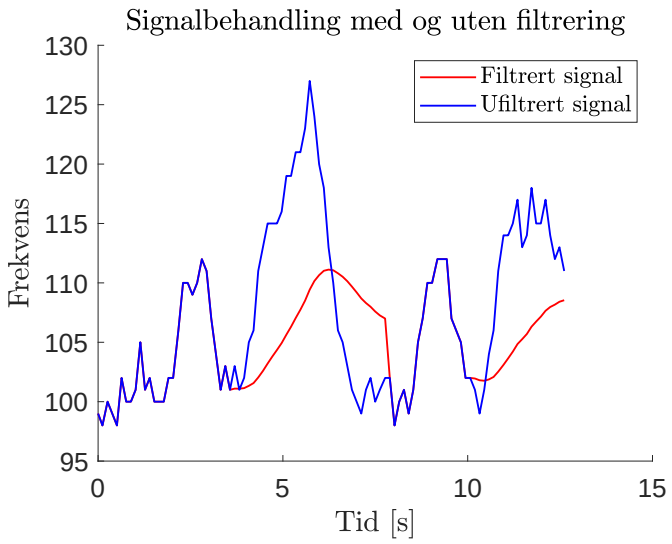
Figur 2.8: Signal med støy som er filtrert med FIR-filtre. Høy M verdi gir mer filtrering, lav M gir kjappere respons og mindre filtrering.

Vi forsetter med å gjøre en nærmere analyse i figur 2.9 av effekten ved å endre på α. Som drøftet tidligere ser vi at en lav verdi av α gir et glatt signal som filtrerer vekk mer ujevnheter sammenlignet med høy α. Vi legger også merke til at en høy α gir en raskere respons som følger signalet tettere enn en lav α.



Figur 2.9: Signal med støy som er filtrert med IRR-filtre. Vi ser at signalet blir filtrert mer dess lavere a-verdi

2.5 Filtrering av lydsignal



Figur 2.10: Vi får en jevnere lyd når lydsignalet blir filtrert

Vi gjør et til forsøk for å vise en praktisk bruk av filter. Legoutstyret er satt opp med en lyssensor som føres over et gråskalaark. Se kodeutdrag 2.5. Lyssignalet blir tillagt målestøy og en kosntant for å få signalene i ønsket frekvensområde. Signalet blir så konvertert til lydsignal. Det er satt inn et IRR-filter i koden som settes på når en bryter trykkes inn. Når legoutstyret kjører uten at bryteren trykkes inn så spilles en ujevnlyd, og når bryeteren trykkes inn får vi en jevnere lydsignal, se figur 2.10

Kapittel 3

Numerisk derivasjon (utført av Ole Martin og Marie)

3.1 Introduksjon

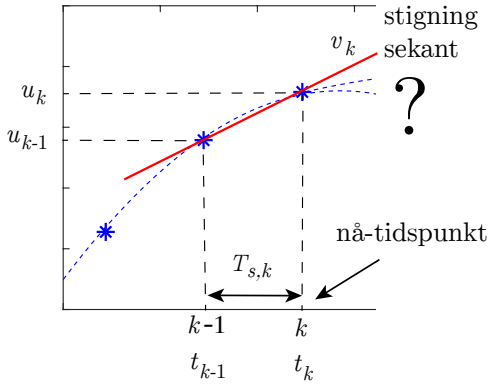
Derivasjon handler om hvordan en størrelse endrer seg; mer spesifikt, hvordan den momentane endringen er. Analytisk finner vi den momentane endringen ved å derivere funksjonsuttrykket. Ettersom vi sjeldent kjenner eller kan utlede funksjonsuttrykket til et innsamlet datasett, benytter vi i stedet numerisk derivasjon for å estimere endringen. I likhet med numerisk integrasjon og digital filtrering, har numerisk derivasjon en rekke bruksområder, både innen industrien og i våre hverdagslige liv. Blant annet kan numerisk derivasjon brukes til å analysere og beregne trender – både innen økonomi, men også smittetrender som under korona-epidemien – samt til å beregne fart fra avstandsmålinger, som ved fartskontroller [2].

For å demonstrere praktisk bruk av numerisk derivasjon, har vi utført tre forsøk hvor vi tok utgangspunkt i beregning av fart fra avstandsmålinger. På denne måten får vi illustrert prinsippet bak for eksempel politiets fartskontroller, men også hvor viktig det er å filtrere måledataene før videre bruk. I tillegg får vi sett nærmere på matematiske relasjoner mellom et funksjonsuttrykk og dens numerisk deriverte.

3.2 Teori

Fra matematikken vet vi at derivasjon av en funksjon gir stigningstallet til tangenten til et punkt, for eksempel ved tiden t. Dette gjør vi ved å la Δt gå mot 0. Ved numerisk integrasjon må vi i stedet la Δt være endelig og beregner da en sekant som vi bruker til å gi oss en tilnærming til tangenten. Denne sekanten kan beregnes med tre ulike metoder: bakoverderivasjon, foroverderivasjon og senterderivasjon. Til forsøkene vi utførte, benyttet vi bakoverderivasjon.

Bakoverderivasjon følger tilsvarende prinsipp som bakoverintegrasjon; den benytter nåværende og forrige måling til å beregne den deriverte. Vi lar Δt tilsvare det variable tidsskrittet $T_{s,k}$, definert i ligning 1.2 i kapittel 1, til tidsseriesignalet u_k, og kaller da sekanten – utgangssignalet vårt – for v_k, se figur 3.1.



Figur 3.1: Illustrasjon som viser prinsippet til bakoverderivasjon. Sekanten, som tilnærmer tangenten til punktet, beregnes ved å benytte nåværende og forrige måling, u_k og u_{k-1}, med $\Delta t = T_{s,k}$, som er det variable tidsskrittet. Figuren er hentet fra [1].

Ut fra figur 3.1 definerer vi da sekanten beregnet med bakoverderivasjon som:

$$\boxed{v_k = \frac{u_k - u_{k-1}}{T_{s,k}}, \quad \forall k = 2, \ldots, \quad \text{gitt } v_1 = 0} \tag{3.1}$$

3.3 Kode for bakoverderivasjon

Basert på ligning 3.1 har vi definert en Matlab-funksjon for bakoverderivasjon, vist i kodeutdrag 3.1.

Kode 3.1: Matlab-funksjon for bakoverderivasjon brukt i forsøkene.

```
function Sekant = BakoverDerivasjon(FunctionValues, Timestep)

    Sekant = (FunctionValues(2) - FunctionValues(1)) / Timestep;

end
```

Funksjonen benyttes som vist i kodeutdrag 3.2 under.

Kode 3.2: Funksjonskall for bakoverderivasjon.

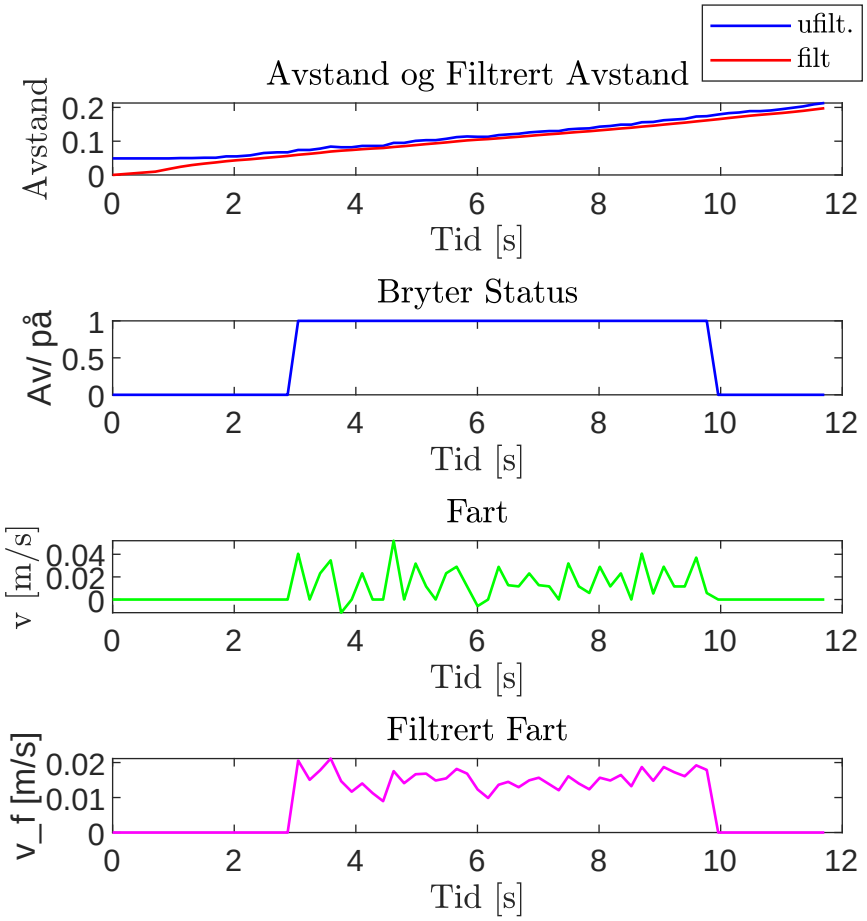
```
if k > 1:
    v(k) = BakoverDerivasjon([u(k-1)], u(k)], Ts(k));
end
```

Merk at det variable tidsskrittet `Ts(k)` må være definert slik som i kodeutdrag 1.4.

3.4 Fartsmåling med laserpistol

Figur 3.2: Ultralyssensor og bryter satt opp som en laserpistol. Dette utstyret er brukt til å simulere lasermåling.

I det første forsøket måler i fart ved bruk av en modell av en laserpistol. Vi beveger laseristolen bort fra en vegg og måler avstand. Når vi trykker inn knappen så regner vi ut farten farten med å bruke metoden for bakoverderivasjon til å regne ut farten både med filter og uten filter. Figur 3.3 viser at uten filteret så blir det vanskelig å tyde hva farten blir, når vi bruker verdier fra den filtrerte avstanden så ser man at farten kommer tydeligere frem og at farten ligger rundt 1.5 cm per sekund.



Figur 3.3: Simulering av lasermåling der en α på 0.2 er brukt ved IIR-filtrering

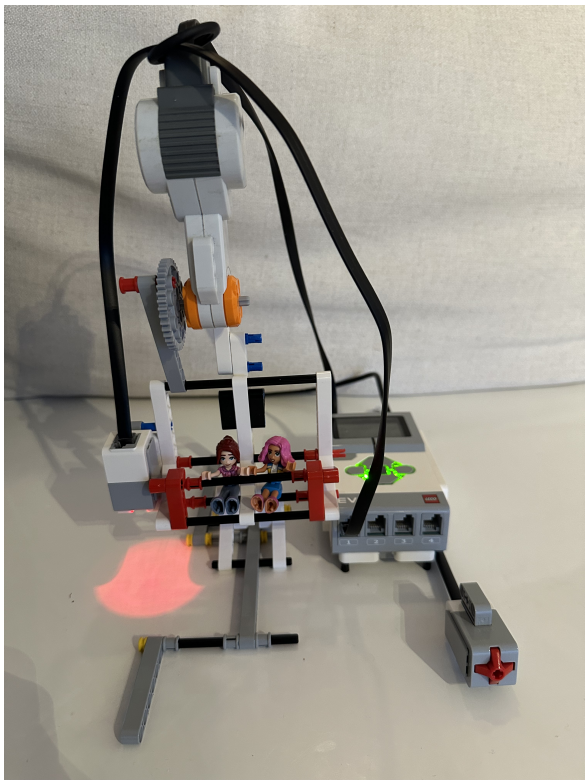
3.5 Relasjon til den deriverte som funksjonsuttrykk

I neste forsøk har vi laget en modell vist i figur 3.4 i lego som skal simulere stupet i Kongeparken, og vi vil gjøre ulike beregninger for å finne farten. Vi ser i målingene i figur 3.5 at stupet bruker lengre tid opp enn ned, noe som også vil være tilfelle

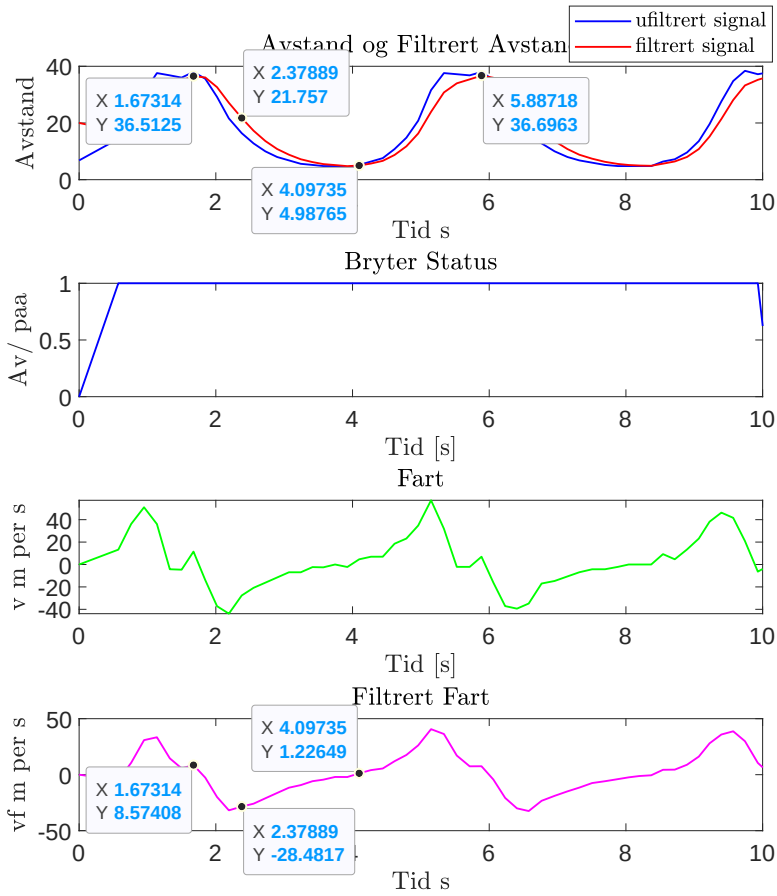
i praksis. Det første vi gjør er å finne amplituden U og likevektsverdien C ut fra to målinger, men på topp og en på bunn.

$$U = \frac{36.5\,\mathrm{m} - 5.0\,\mathrm{m}}{2} = 15.8\,\mathrm{m}$$

$$C = \frac{36.5\,\mathrm{m}5.0\,\mathrm{m}}{2} = 20.8\,\mathrm{m}$$



Figur 3.4: Legoutstyr med lyssensor. Dette utstyret fungerte best til å simulere stupet



Figur 3.5: Simulert avstandsmåling stupet

Videre beregner vi ω ved å først finne perioden T mellom to påfølgende topppunkter i en syklus. Første topppunkt er ved $t_1 = 1.5$ sekunder og det neste ved $t_2 = 5.7$ sekunder. Perioden T beregnes da som:

$$T = t_2 - t_1 = 5.8\,\mathrm{s} - 1.7\,\mathrm{s} = 4.1\,\mathrm{s}$$

Dette resultatet representerer periodetiden T_p som beskriver endringen i avstand mellom de to målingene.

Deretter kan vinkelfrekvensen ω beregnes ved å bruke formelen for vinkelfrekvens:

$$\omega = \frac{2\pi}{T_p} = \frac{2\pi}{4.1}\,\text{rad/s} = 1.496\text{rad/s}$$

Avstanden kan nå uttrykkes som:

$$u_f(t) = U\sin(\omega t) + C$$

Setter vi inn verdiene får vi avstanden som:

$$u_f(t) = 15.8\sin(1.496t) + 20.8m$$

Fartskurven

Fartskurven $v_f(t)$ gitt som den tidsderiverte av $u_f(t)$ kan skrives som:

$$v_f(t) = \frac{d}{dt}\left(U\sin(\omega t) + C\right) = U\omega\cos(\omega t)$$

Dette tilsvarer:

$$v_f(t) = 23.57\sin(1.496t + \frac{\pi}{2})$$

Den deriverte av funksjonen $u_f(t)$ resulterer i en faseforskyvning på $+\frac{\pi}{2}$ (90°), som utgjør forskjellen i topp/bunnfart og maks/min avstand

Analyse beregninger

Vi validerer videre resultatene, men oppnår ikke de forventede resultatene farten ved å beregne den deriverte av avstand. Dette kan skyldes at forsøket ikke ga perfekte sykluser. Vi ser f.eks at bunnpunktet ikke kommer midt mellom topppunktene. Ved å redusere faseforskyvningen til 45° blir resultatene nærmere forventning i de målte punktene når farten reduseres.

$$v_f(t) = 23.57\sin\left(1.496t + \frac{\pi}{4}\right)$$

3.5 Relasjon til den deriverte som funksjonsuttrykk

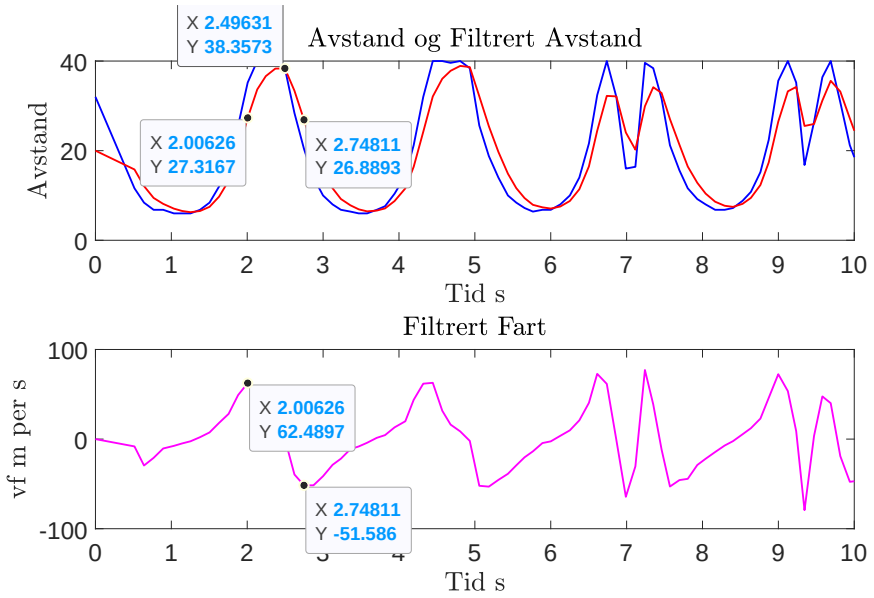
Vi legger også merke til at farten er nærmere forventning på 0 på toppunktene for avstand sammenlignet med den numerisk deriverte som gir den kalkulerte farten i forsøkene:

tid (s)	1,7	2,4	4,1	5,9
avstand i forsøk (m)	36,5	21,0	5,0	36,7
avstand kalkulert (m)	29,7	14,6	18,4	29,7
differanse	6,8	6,4	-13,4	7,0
fart i forsøk m/s	8,6	-28,0	1,2	7,5
fart kalkulert m/s, endret faseforskyvning	-4,4	-21,9	14,0	-4,4
differanse m/s	13,0	-6,1	-12,8	11,9

Figur 3.6: Verifisering av resultater

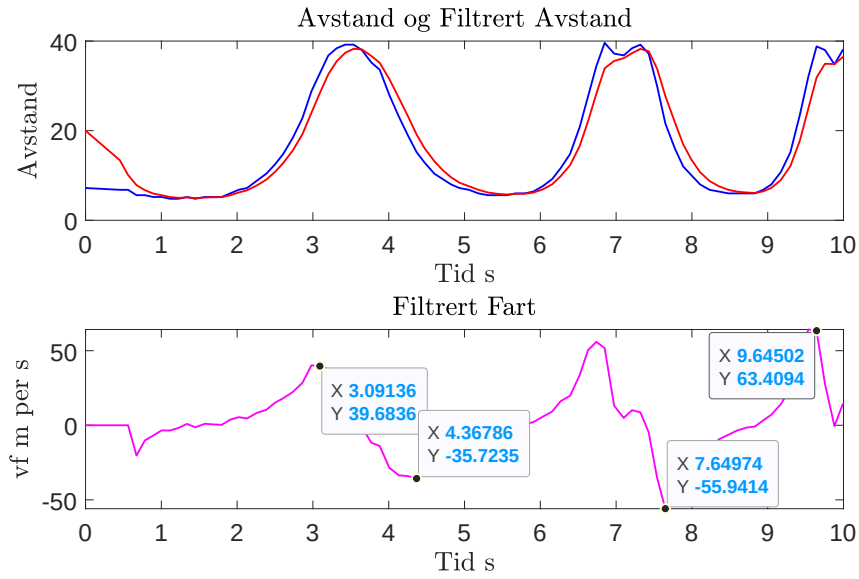
Videre forsøk og beregninger kunne sett på om faseforskyvningen i flere punkter, spesielt når farten øker, og vurdert om faseforskyvningen må varieres etter hvor i syklusen man er.

3.6 Betydningen av endret frekvens ω



Figur 3.7: Ny simulering av stupet med gradvis økende fart/frekvens

Endringen i frekvens, ω i numerisk derivasjon, sier noe om hvordan frekvensendringer påvirker hvor raskt stupet går opp og ned. Dess høyere frekvens, dess raskere endring. Høyden forventes å være på samme nivå, mens den deriverte av standen (farten) forventes å gi høyere verdier. Vi gjør et nytt forsøk der vi øker pådraget til motoren i modellen. Vi ser ut fra 3.7 at våre antagelser stemmer om resultatene stemmer. Målt høyde er på 38 meter som er på samme nivå som i forrige forsøk. Farten målt er høyere, i intervallet -51 m/s til 62 m/s, mot tidligere ca -30 m/s til 30 m/s.



Figur 3.8: Ny simulering av stupet med gradvis økende fart/frekvens

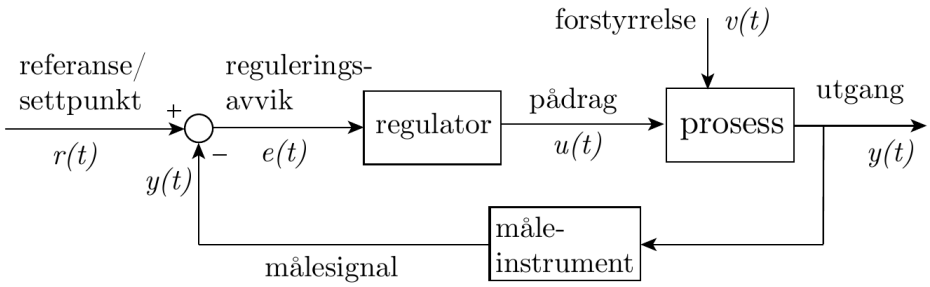
Vi gjør et siste forsøk der vi gir motoren et økende pådrag. Utfra teorien skal amplituden til avstanden være den samme, men frekvensen til avstanden vil være redusert. Den deriverte at avstanden skal være økende. Forsøket bekrefter teorien i figur 3.8

Kapittel 4

PID regulatoren (utført av hele gruppen)

4.1 Introduksjon

En PID-regulator er en mye brukt regulator for å kontrollere ulike prosesser som turtall, temperatur, og strømning i aparater som kjøkkenkomfyrer og kaffemaskiner. Regulatoren kombinerer tre operasjonener – proporsjonalitet, integrering og derivering – for å regulere et utgangssignal til å følge en ønsket (referanse)verdi, ved å sammenligne målingene fra prosessen med referanseverdien og justere prosessen deretter [1]. Dette gjøres gjennom negativ tilbakekobling ved å kontinuerlig trekke målesignalet $y(t)$ fra referansen $r(t)$, hvilket gir reguleringsavviket $e(t)$ som regulatoren bruker til å styre pådraget $u(t)$ til prosessen en ønsker å regulere, for eksempel turtall. Prinsippet bak denne negative tilbakekoblingen er illustrert i figur 4.1.



Figur 4.1: Forenklet blokkskjema som viser prinsippet bak en regulator, blant annet PID-regulatoren som dette kapittelet tar for seg. Målesignalet $y(t)$ blir kontinuerlig sammenlignet med referansen $r(t)$ gjennom negativ tilbakekobling for å gi reguleringsavviket $e(t)$, som regulatoren benytter til å lage et pådrag $u(t)$ til prosessen. Prosessen kan bli utsatt for en forstyrrelse $v(t)$ som påvirker utgangen og målesignalet, som igjen føres tilbake og sammenlignes med referansen. Figuren er hentet fra [1].

Ved å bygge videre på arbeidet i kapittel 1, 2 og 3, har vi programmert en PID-regulator til å regulere motorene på en LEGO-robot. PID-regulatoren ble senere brukt til manuell kjøring av roboten med turtallsregulator, beskrevet i kapittel 6, samt automatisk kjøring som beskrevet i kapittel 7. Dette kapittelet tar for seg selve kodingen og utprøvingen av regulatoren. Først gjennomgår vi kort den teoretiske bakgrunnen for en PID-regulator. Deretter går vi steg for steg gjennom de tre leddene som utgjør en PID-regulator; proporsjonalvirkning (P), integralvirkning (I), og derivatvirkning (D). Gjennom forsøkene vi har gjort med fokus på én del om gangen, illustrerer vi virkemåten og utfordringene til de ulike delene. Til slutt går vi gjennom et siste forsøk hvor vi satte sammen alle delene i regulatoren og benyttet prøve- og feilemetoden for å få de optimale verdiene, slik at vi senere fikk roboten til kjøre automatisk langs en bane.

4.2 Teori

Som nevnt innledningsvis består en PID-regulator av tre ledd, samt videre også tre regulatorparametre, K_p, K_i og K_d, som justeres utifra prosessen som skal reguleres [1]:

- **Proporsjonalvirkning (P):** Proporsjonalleddet produserer et pådrag som er proporsjonal med den nåværende feilen mellom settpunktet og målever-

dien, dvs. reguleringsavviket $e(t)$ forsterkes med regulatorparameteren K_p:

$$P = K_p \cdot e(t) \tag{4.1}$$

Proporsjonalleddet gir en rask respons på feil, men gir et pådrag kun når $e(t) \neq 0$, hvilket vi vil se senere i dette kapittelet. Dermed vil heller aldri $y(t) = r(t)$, slik at det vil alltid være et konstant avvik med en regulator som kun benytter P-leddet.

- **Integralvirkning (I):** Integraldelen akkumulerer feilen over tid og forsøker å eliminere det konstante avviket som kan oppstå med bare et P-ledd i regulatoren. Matematisk kan den uttrykkes som

$$I = \int_0^t K_i \cdot e(\tau) d\tau \tag{4.2}$$

hvor K_i er regulatorparameteren. Integralleddet reagerer langsommere på feil enn proporsjonalleddet, men kan gi et pådrag selv når $e(t) = 0$. Integralleddet er utsatt for metning, dvs. det vil fortsette å vokse og vokse dersom ikke $e(t) = 0$ ved et tidspunkt, eller leddet begrenses.

- **Derivatvirkning (D):** Derivatleddet reagerer på endringshastigheten av feilen og er gitt ved

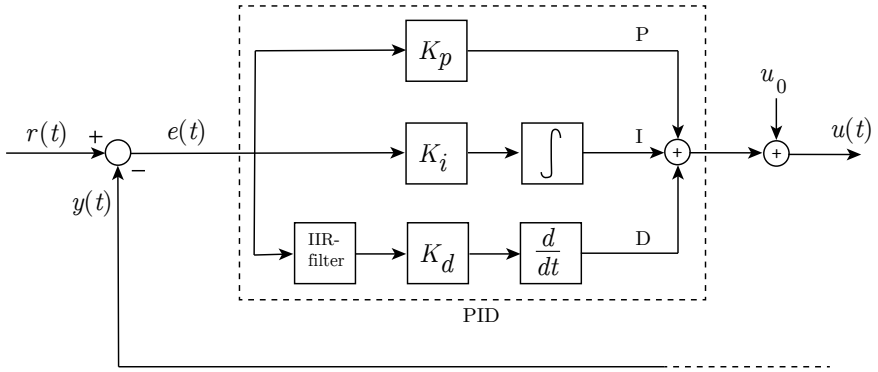
$$D = \frac{d}{dt}(K_d \cdot e_f(t)) \tag{4.3}$$

hvor $e_f(t)$ er den IIR-filtrerte verdien av $e(t)$. Derivatleddet er dermed proporsjonal med endringshastigheten og vil hjelpe med å dempe systemet og redusere oversvingning. På denne måten kan man stabilisere regulatoren. Derivatleddet er derimot sensitivt for støy og kan føre til mer ustabilitet dersom α-verdien i IIR-filteret og K_d ikke er optimalisert for prosessen.

Kombinasjonen av disse tre leddene gjør PID-regulatoren svært effektiv – gitt riktig justering av regulatorparametrene (kalt *controller tuning*), hvilket vi vil se nærmere på senere i dette kapittelet. Legger vi et basispådrag u_0 til P-, I- og D-leddene, får vi det matematiske uttrykket for PID-regulatoren

$$u(t) = u_0 + K_p \cdot e(t) + \int_0^t K_i \cdot e(\tau) d\tau + \frac{d}{dt}(K_d \cdot e_f(t)) \tag{4.4}$$

hvilket kan også uttrykkes visuelt ved hjelp av et blokkskjema, som vist i figur 4.2. Basispådraget u_0 sørget for at $u(t) \neq 0$, hvilket er nødvendig for prosesser som turtall for kjøring, men kan settes lik null der et basispådrag ikke er nødvendig.



Figur 4.2: Blokkskjema over en PID-regulator med regulatorparametrene K_p, K_i og K_d, henholdvis for proporsjonalvirkning (P), integralvirkning (I) og derivatvirkning (D). Basispådraget u_0 kan sløyfes dersom det ikke er nødvendig at $u(t) \neq 0$. Figuren er hentet fra [1].

4.3 Kode for PID-regulator

Ettersom vi senere skulle benytte en LEGO-robot til å utføre automatisk kjøring og manuell kjøring med turtallregulator, benyttet vi en av motorene i settet til å programmere og teste ut en regulator. Koden for implementeringen av PID-regulatoren er basert på ligningene 4.1-4.4 og grunnkode for regulatoren gitt i [1]. Vi har brukt følgende notasjon i koden:

- `u_A(k)` for pådraget (fra motor A)
- `x(k)` for motortilstandene, hvor
  - `x1(k)` er vinkelposisjon [°]
  - `x2(k)` er vinkelhastigheten [°/s], som tilsvarer turtallet
  - `x2(k)` er den deriverte av `x1(k)`
- `y(k)` for målinger, som vi setter lik vinkelhastigheten: `y(k) = x2(k);`
- `r(k)` for referansen, som hentes fra potensiometeret på styrestikken til LEGO-settet med følgende kode: `r(k) = 10 * JoyPot(k);`

- Variasjonsområdet for vinkelhastigheten er da $-1000°/\text{s} < \texttt{r(k)} < 1000°/\text{s}$.

Vi beregnet motortilstandene og reguleringsavviket slik gitt i kodeutdrag 4.1 under. `Ts(k)` ble beregnet som vist i kodeutdrag 1.4 i kapittel 1, mens koden for funksjonen `BakoverDerivasjon()` er gitt i kodeutdrag 3.1 i kapittel 3.

Kode 4.1: Matlab-kode for å beregne vinkelhastigheten `x2(k)` fra vinkelposisjonen `x1(k)`, referansen $r(k)$ og reguleringsavviket $e(k)$.

```
% Motorens tilstander
x1(k) = VinkelPosMotorA(k);
x2(k) = BakoverDerivasjon(x1(k-1:k), Ts(k));

% Beregning av reguleringsavvik
y(k) = x2(k);
r(k) = 10*JoyPot(k);
e(k) = r(k)-y(k);
```

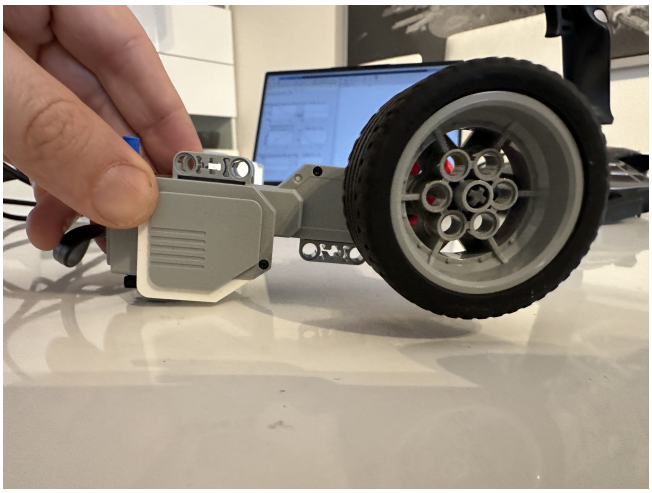
Videre ble P-, I- og D-leddene, samt selve pådraget $u(k)$, implementert som vist i kodeutdrag 4.2, hvor koden for funksjonen `IIR-filter()` er gitt i kodeutdrag 2.2 i kapittel 2.

Kode 4.2: Matlab-kode for å beregne P-, I- og D-ledd, samt pådraget u_A

```
P(k) = Kp * e(k);
I(k) = I(k-1) + Ki * e(k) * Ts(k);
e_f(k) = IIR_filter(OldFilteredValue, e(k), alfa);
D(k) = Kd * BakoverDerivasjon([e_f(k-1), e(k)], Ts(k));

% Beregning av pådrag til motor
u_A(k) = u0 + P(k) + I(k) + D(k);
```

I forsøkene satte vi `u0 = 0;` ettersom vi ikke ønsket pådrag på motoren da referanseverdien for turtallet var 0.

4.4 Regulering av PID-regulator



Figur 4.3: Legoutstyr, motor med hjul. Dersom vi ønsekr motstand settes hjulet ned i underlaget

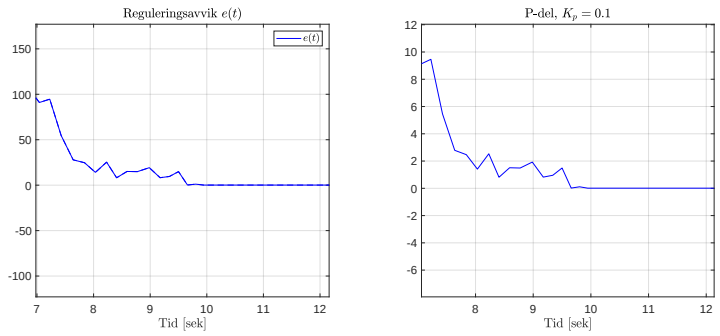
I forsøkene vil vi konfigurere en PID-regulatoren for å styre en legomotor. Denne motoren, som er koblet til et hjul, vil vi gi motstand. Vi justerer ønsket hastighet ved å bruke "throttle-spaken" på joysticken (figur 6.3). For å simulere motstand, presser vi hjulets mot underlaget, dette skaper friksjon. Vi kan da observere hvordan PID-regulatoren responderer og justerer pådraget basert på endringer i motstanden.

Først kalibrerer vi P-delen, så I-delen og D-delen til slutt. På hvert ledd så starter vi med en lav verdi. Dersom regulatoren gir tilfredsstillende resultat så dobles verdien. Når verdien blir så høy at regulatoren gir for mye utslag, så reduseres verdien med 20% til regulatoren gir ønsket effekt.

4.4.1 Kalibrering av P-ledd (proporsjonal)

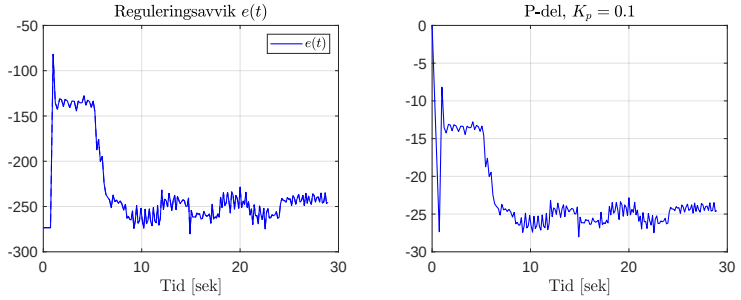
Vi starter med å konfigurere proporsjonal-delen (P-delen) av regulatoren. P-delen vil kun regulere pådraget når vi har et reguleringsavvik, $e(t)$, som er differansen mellom referanseverdien $r(t)$ og systemets respons $y(t)$. Vi starter med å se på et par viktige egenskaper ved P-leddet. Den første egenskapen er at P-delen kun

gir regulering når avviket $e(t)$ er større enn 0. I figur 4.4 har vi et utdrag fra en kjøring der $e(t)$ rett før 10 sekunder går mot null. Vi ser da at reguleringen fra P-leddet opphører.



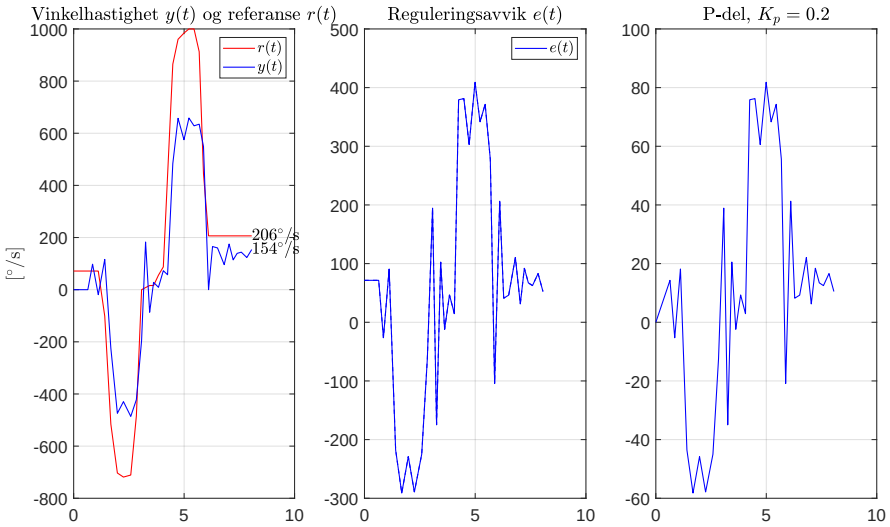
Figur 4.4: ved $e(t) = 0$ så vil pådraget $u(t) = 0$

Den neste egenskapen vi skal se på er reaksjonen fra P-leddet når motstanden er stabil og at dermed $e(t)$ er stabilt. Så lenge $e(t)$ er stabilt vil også reguleringen være stabil. I figur 4.5 ser vi denne effekten da



Figur 4.5: I figuren ser vi at et stabilt reguleringsavvik gir en stabil reaksjon fra P-delen av regulatoren. Etter ca 5 sekunder har vi et sprang i reguleringsavviket, og utover fra ca 7 sekund har vi en stabil reaksjon

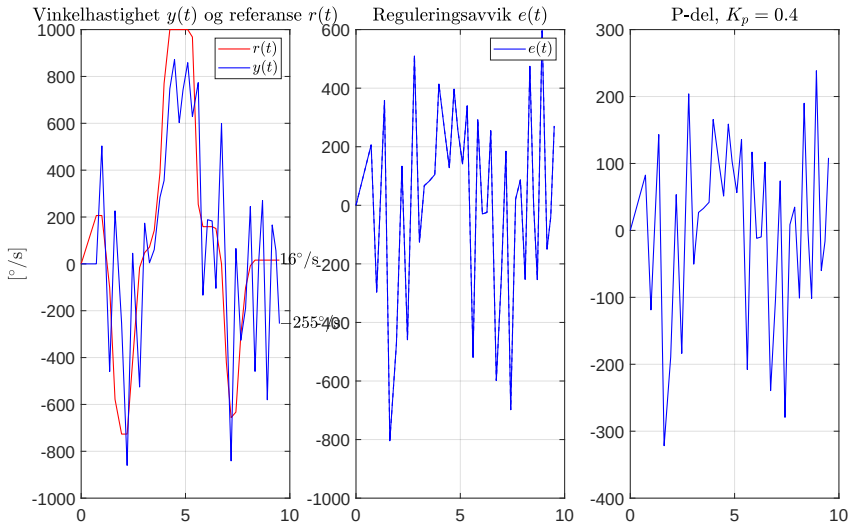
Videre vil vi gi referanse verdi med throttlen på joysticken, og vi vil stille inn regulatoren slik at den justerer farten til referanseverdien. Verdien på K_p vil avgjøre hvor raskt motorens turtall reagerer på endringer i referanseverdien. Vi starter testingen med K_p på 0.2 (4.6).



Figur 4.6: Når vi endrer referanseverdien ser vi hvordan P-leddet justeres.

Vi fortsetter å øke K_p med dobbel verdi helt til vi ser stående svingninger i målingen $y(k)$.

$K_p = 0.4$ resulterer i stående svingninger, se figur 4.7.

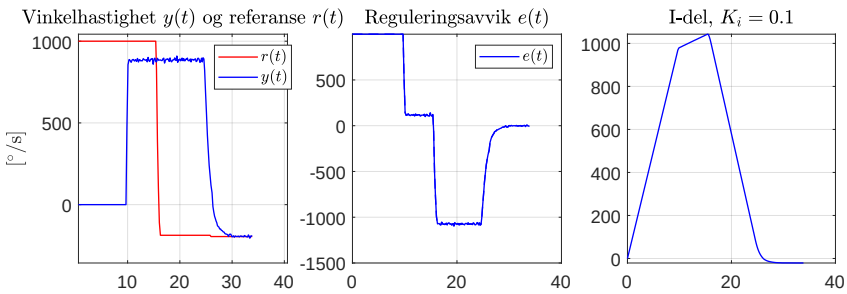


Figur 4.7: Eksempel med for stor K_p der vi får stående svingninger

Vi reduserte K_p med omtrent 20 prosent helt tilbake til svingningene stopper. Av disse forsøkene er det først tilbake på $K_p = 0.2$ at det igjen er en stabil reaksjon, se figur 4.6.

4.4.2 Kalibrering av I-ledd (integral)

Vi starter med å se på en egenskap ved I-leddet som er viktig å være oppmerksom på. I-leddet akkumulerer avviket, og dersom referanseverdien over tid ligger over faktisk verdi, kan vi få en "wind up". I figur 4.8 har vi konstruert et eksempel der motoren ikke starter før etter 10 sekund, mens vi har en høy referanseverdi. Vi ser at etter ca 15 sekund når referanseverdien reduseres, så fortsetter I-leddet å gi pådrag så motoren går til max turtall. En effekt av dette er at først etter ca 25 sekunder reduseres pådraget slik at vinkelhastigheten går mot referanseverdien. For å unngå denne type forsinkelse i pådraget kan det legges inn en begrensning på integratoren i matlabkoden, se kode 4.3 nedenfor.

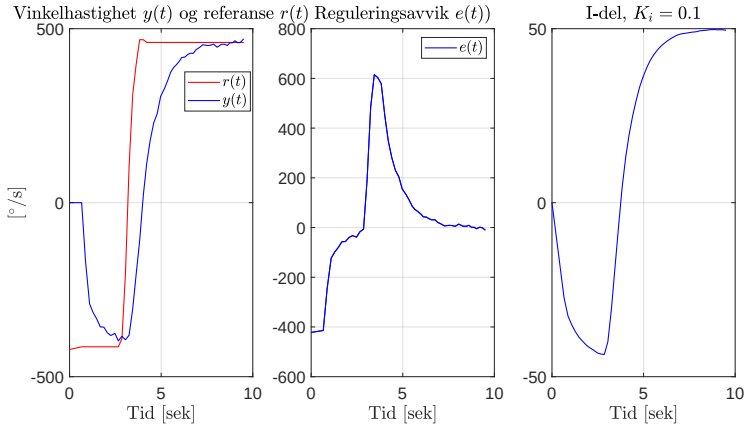


Figur 4.8: Eksempel på wind up der vi får en forsinket reaksjon på endringen i referanseverdi

Kode 4.3: Eksempel på MatLab-kode for integratorbegrensning til I-leddet i PID-regulatoren.

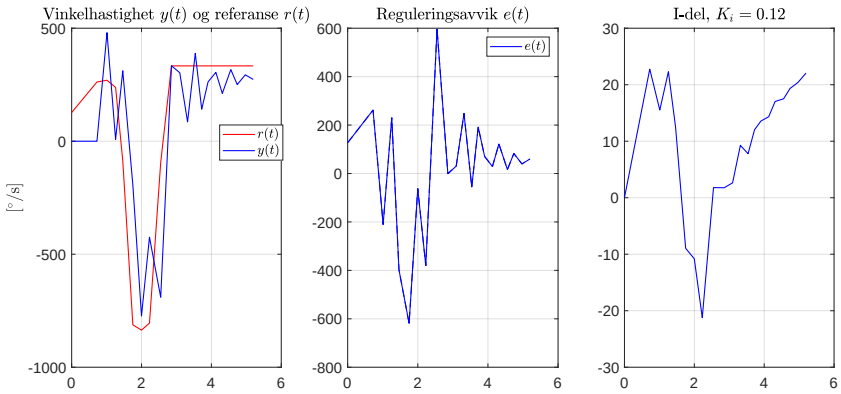
```
I_max(k) = 100;
I_min(k) = -100;

if I(k) > I_max(k)
    I(k) = I_max(k);
end

if I(k) < I_min(k)
   I(k) = I_min(k);
end
```

Andre viktige egenskaper ved I-leddet er at på grunn av leddet har "minne"har vi pådrag selv om $e(t) = 0$, se figur 4.9 der $e(t) = 0$ rundt 7 sekunder. Vi observerer også at når referanse $r(t)$ verdien er negativ, så gir I-leddet et negativt pådrag, og når $r(t)$ er positiv får vi et positivt pådrag.



Figur 4.9: Vinkelhastigheten følger synker når $e(t) < 0$ og stiger når $e(t) > 0$, og I-leddet forsetter å gi pådrag også etter at $e(t)$ nærmer seg 0

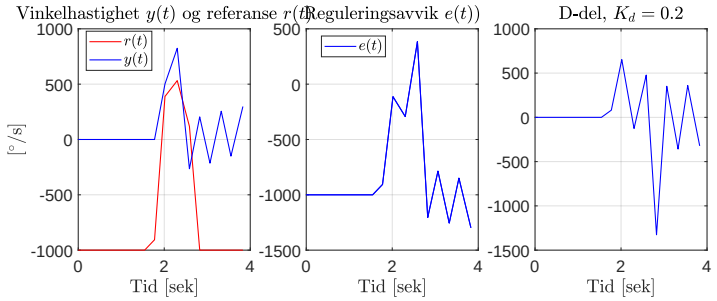
For justering av integralforsterkningskoeffisienten K_i, startet vi med en verdi på 0.1, som gir en forventet respons. Ved K_i på 0.2, opplevde vi en hissig respons som indikerer en overreaksjon. Vi justerte deretter K_i ned med 20% til 0.16, noe som ga bedre resultat, men fortsatt en overreaksjon. Etter videre justering til K_i-verdi på 0.12 fikk vi en stabil stabil respons.



Figur 4.10: Når vi endrer referanseverdien ser vi at I-leddet følger samme mønster som avviket

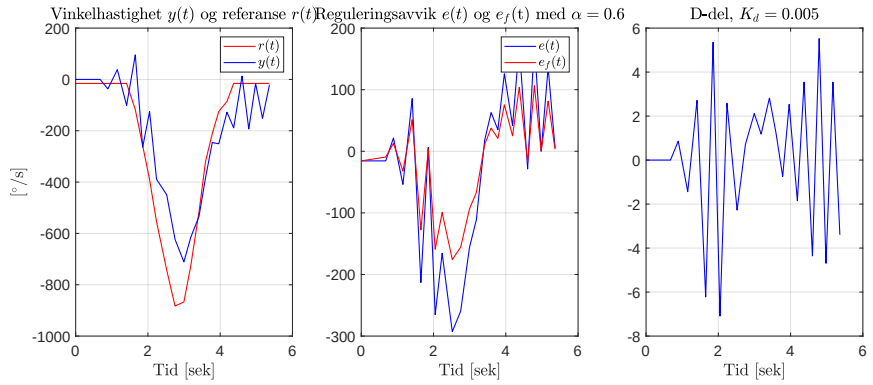
4.4.3 Kalibrering av D-ledd (derivativ)

Spesielle egenskaper med D-leddet i en regulator er at leddet ikke reagerer på avvik $e(t)$ i selv, kun endringer i avvik. For eksempel ser vi i figur 4.11 at D-leddet ikke gir noen regulering mellom 0 opptil nesten 2 sekunder selv om det er et reguleringsavvik, det kommer kun en reaksjon når avviket $e(t)$ endres. I hvor stor grad reguleringen ser vi i ligning 4.3 at vil avhenge av størrelsen på K_p. Siden reguleringen tar utgangspunkt i filtrerte verdier av $e(t)$, vil effekten av leddet også bli justert av verdien av α. En α nærmere 1 vil gi bråere regulering enn en α nærmere 0.



Figur 4.11: Når vi endrer referanseverdien ser vi at D-leddet følger samme mønster som avviket

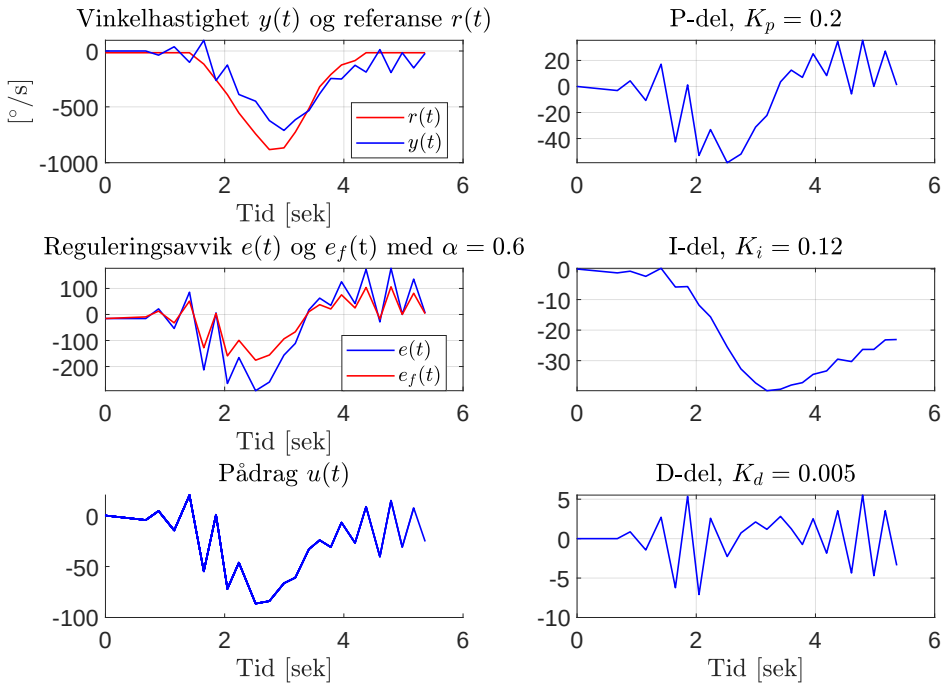
For derivativ delen forsøkte vi ulike verdier av α, og justerte derivatforsterknings-koeffisienten K_d. Vi så at K_d måtte settes til en lav verdi, før vi kunne fortsette med metoden med dobling, og så reduksjon på 20 %. Etter å ha testet forskjellige kombinasjoner av K_d og α, konkluderte vi med at en innstilling av K_d til 0.005 og α på 0.6, ga best resultat, selv om bidraget er lite og det er usikkert om bidraget i det hele tatt er positivt for reguleringen.



Figur 4.12: Når vi endrer referanseverdien ser vi at D-leddet følger samme mønster som avviket

4.4.4 Samlet regulering

Til slutt ser vi på resultatet av den samlede reguleringen i figur 4.13. Vi ser at den fungerer etter hensikten, og at alle leddene gir bidrag til reguleringen. Reguleringen skjer ikke helt jevnt, for eksempel ser vi det noen mindre innslag stående svingninger (pådraget hopper og ned). Selv om regulatoren fungerer et bra resultat, kan svingningene antyde at det er rom for ytterligere forbedringer.



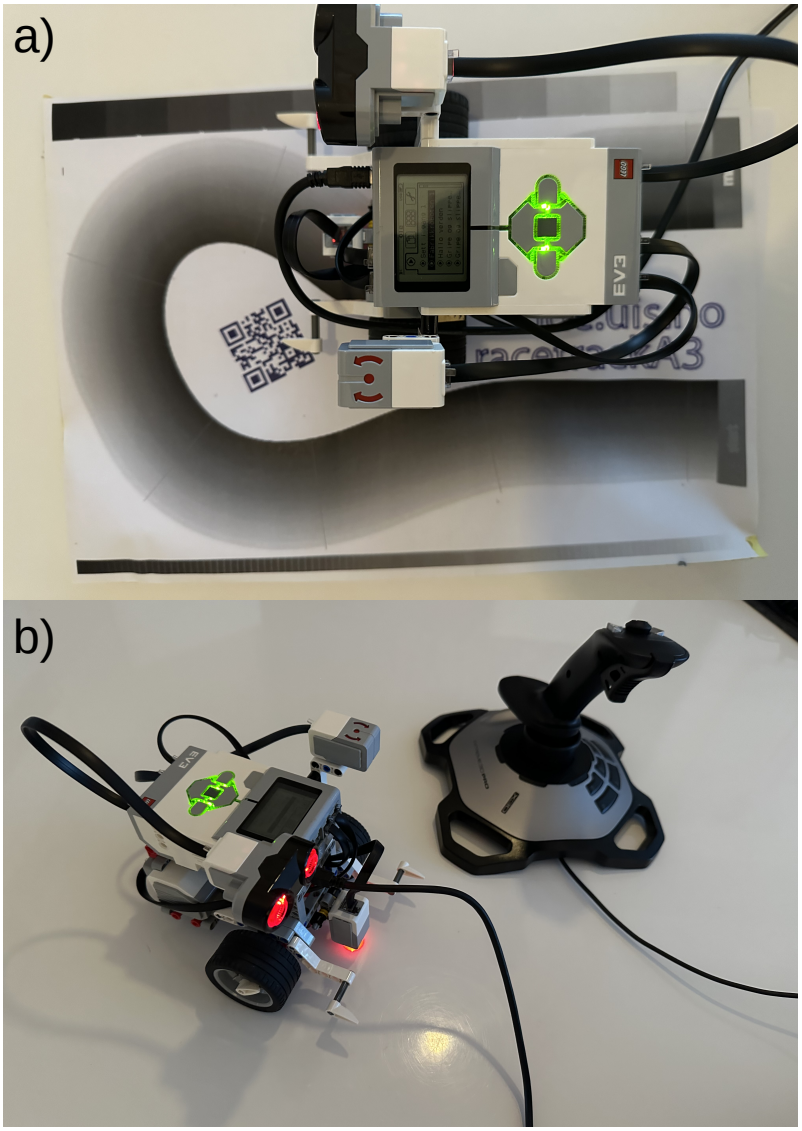
Figur 4.13: Regulering med PID-regulatoren

Kapittel 5

Manuell kjøring av Lego-robot (utført av hele gruppen)

5.1 Introduksjon

I dette kapittelet går vi gjennom et forsøk hvor alle i gruppen manuelt kjørte en LEGO-robot på en bane. På denne måten demonstrerte vi regulering og pådrag av motorer i praksis, hvor vi opptrådde som regulatoren. Ved hjelp av en lyssensor koblet til roboten, registrerte vi en valgt gråskalaverdi på en gråskalabane, vist i figur 5.1, som vår referanseverdi $r(k)$, og fulgte denne til vår beste evne ved å gi pådrag til motorene med styrestikken.



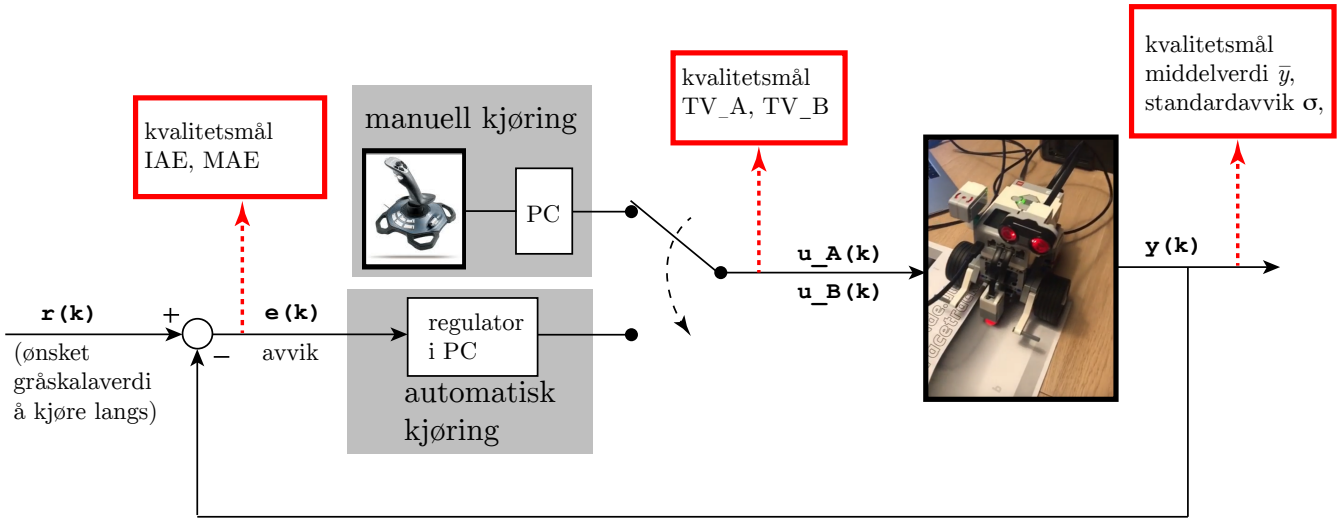
Figur 5.1: LEGO-robot med ulike sensorer, blant annet lyssensor montert midt foran, pekende nedover. Lyssensoren benyttes til å registrere første verdi som referanseverdien $r(k)$ og registrere avviket fra denne etter hvert som roboten styres gjennom løypen. **a)** Roboten og gråskalabanen som benyttes ved manuell og automatisk kjøring. **b)** Roboten og styrestikken som benyttes til å styre roboten.

Målingene fra lyssensoren ble deretter brukt til å finne avviket $e(k)$. I løpet av kjøringen beregnet vi en rekke kvalitetsmål som for eksempel MAE (*Mean Absolute*

Error), pådragsbruk TV (*Total Variation*) og standardavvik σ. Kvalitetsmålene blir benyttet til å sammenligne resultatene fra kjøringene til de ulike gruppemedlemmene senere i dette kapittelet, og automatisk kjøring av LEGO-roboten med PID-regulator som gjennomgås i kapittel 7.

5.2 Teori

Som nevnt benyttes ulike kvalitetsmål for å sammenligne resultatene fra kjøringen. Disse kommer fra industrien og fra reguleringsteknikk. For å beregne kvalitetsmålene brukte vi reguleringsavviket `e(k)`, pådragene til motorene og utgangen `y(k)`. Kvalitetsmålene og hvor i prosessen de beregnes er oppsummert i figur 5.2, som også skisserer oppbyggingen av reguleringen ved manuell og automatisk kjøring av LEGO-roboten.



Figur 5.2: Prinsippskisse for manuell og automatisk kjøring av LEGO-robot med uthenting av data for beregning av kvalitetsmål for kjøringen. Figuren er hentet fra [1].

Under kjøring ble følgende kvalitetsmål beregnet [1]:

- IAE (*Integral of Absolute Error*): IAE benyttes i reguleringsteknikk som kvalitetsmål og tar i bruk integrasjon for å bestemme avviket over en tidsperiode:

$$\text{IAE} = \int_0^t |e(\tau)| d\tau \tag{5.1}$$

Ettersom $e(k)$ kan være både positiv og negativ, benyttes absoluttverdien $|e(k)|$.

- MAE (*Mean Absolute Error*): MAE brukes i statistikk og gir gjennomsnittlig avvik per måling basert på summen av absolutt avvik:

$$\text{MAE} = \frac{1}{k} \sum_{n=1}^{k} |e_n| \tag{5.2}$$

- TV (*Total Variation*): TV brukes i reguleringsteknikk og er en kvantifisering av pådragsbruket, som sier noe om hvor aktivt pådraget brukes for å regulere avviket $e(k)$. Aktivt bruk av pådraget for å minske $e(k)$ mest mulig, kan føre til økt slitasje på delene. TV beregnes fra pådragssignalet $u(k)$:

$$\text{TV} = \sum_{n=1}^{k} |u_n - u_{n-1}| \tag{5.3}$$

Etter kjøring ble følgende kvalitetsmål beregnet:

- Middelverdi $\bar{y}$: Middelverdien er mye brukt i statistikk og gir gjennomsnittet av målingene:

$$\bar{y} = \frac{1}{k} \sum_{n=1}^{k} y_n \tag{5.4}$$

- Standardavviket σ: Standardavviket er også et vanlig kvalitetsmål i statistikk og angir spredningen i måleverdiene, dvs. avstanden fra middelverdien $\bar{y}$:

$$\sigma = \sqrt{\frac{1}{k} \sum_{n=1}^{k} (y_n - \bar{y})^2} \tag{5.5}$$

I tillegg benytter vi fordelingen av målingene i et histogram som kvalitetsmål til sammenligningen.

5.3 Kode for manuell kjøring

For å kunne styre roboten manuelt med styrestikken (*joystick*), måtte vi hente ut verdiene for aksene og knappene, som vist i kodeutdrag 5.1.

Kode 5.1: Kode for å hente verdier for styrestikken (*joystick*).

```
[JoyAxes,JoyButtons] = HentJoystickVerdier(joystick);
JoyMainSwitch = JoyButtons(1);
JoyForover(k) = JoyAxes(2);
JoySving(k) = JoyAxes(1);
```

Etter prøving og feiling kom vi fram til koden i kodeutdrag 5.2 for implementering av kjøringen, dvs. beregning av pådraget til motor A og B utifra styrestikkverdiene og to skalafaktorer, a og b.

Kode 5.2: Kode for implementering av manuell kjøring av LEGO-roboten.

```
% Parametre
a = 0.7;  % skalafaktor for fart
b = 0.5; % skalafaktor for sving

% Oppdater pådrag for hver motor
PowerA(k) = a * JoyForover(k) - b * JoySving(k)
PowerB(k) = a * JoyForover(k) + b * JoySving(k)
```

Basert på måleverdiene $y(k)$ som da er de registrerte verdiene `Lys(k)`, samt ligningene 5.1-5.3, kan vi beregne avviket $e(k)$ og kvalitetsmålene fra koden vist i kodeutdrag 5.3. Merk at TV må beregnes for hver motor, `TVA` og `TVB`, og at for middelverdien og standardavviket har Matlab egne innebygde funksjoner, `mean` og `std`.

Kode 5.3: Kode for beregning kvalitetsmål

```
% Avviket e(k)
e(k) = Lys(1) - Lys(k);
if k == 1
    MAE(1) = 0;
    IAE(1) = 0;
end

if k > 1
```

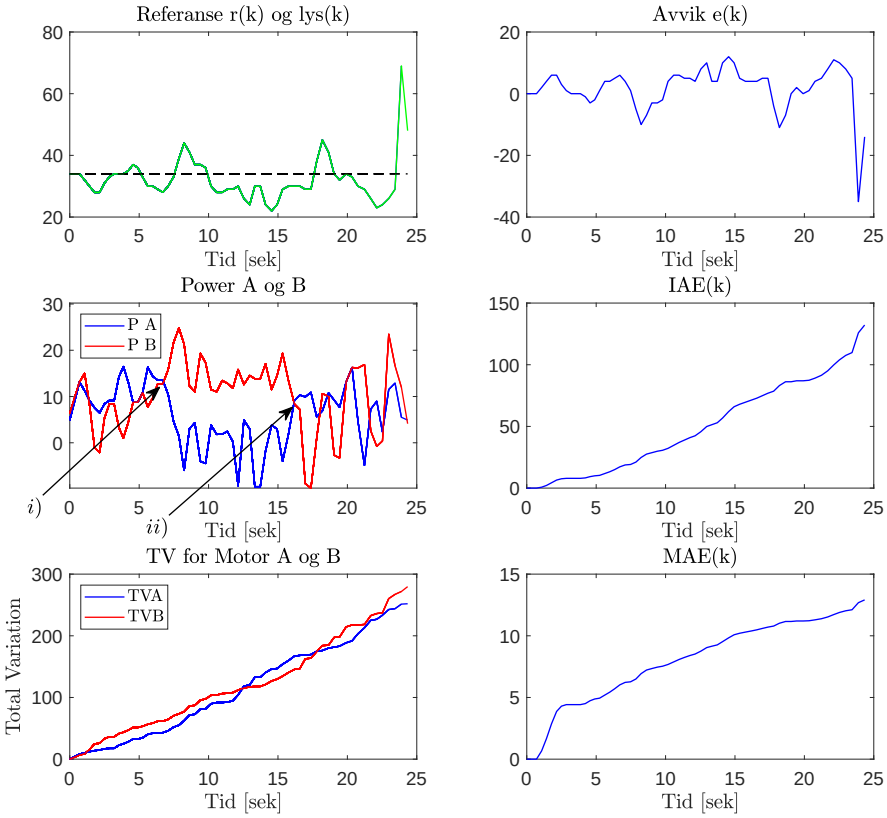
```
    % Numerisk integrasjon av avviket |e(k)| (IAE)
    IAE(k) = Eulerbakover(IAE(k-1), Ts(k), abs(e(k)));

    % Beregning MAE
    MAE(k) = MAE(k-1) + abs(e(k)) / k;

    % Beregning Total Variation for hver motor
    TVA(k) = TVA(k-1) + abs(PowerA(k) - PowerA(k-1));
    TVB(k) = TVB(k-1) + abs(PowerB(k) - PowerB(k-1));
end

% Middelverdi og standardavvik
middelverdi = mean(Lys);
standardavvik = std(Lys);
```

5.4 Resultater fra forsøket

Først skal vi se på resultatet til en av gruppemedlemmene før vi går videre med å sammenligne resultatene.

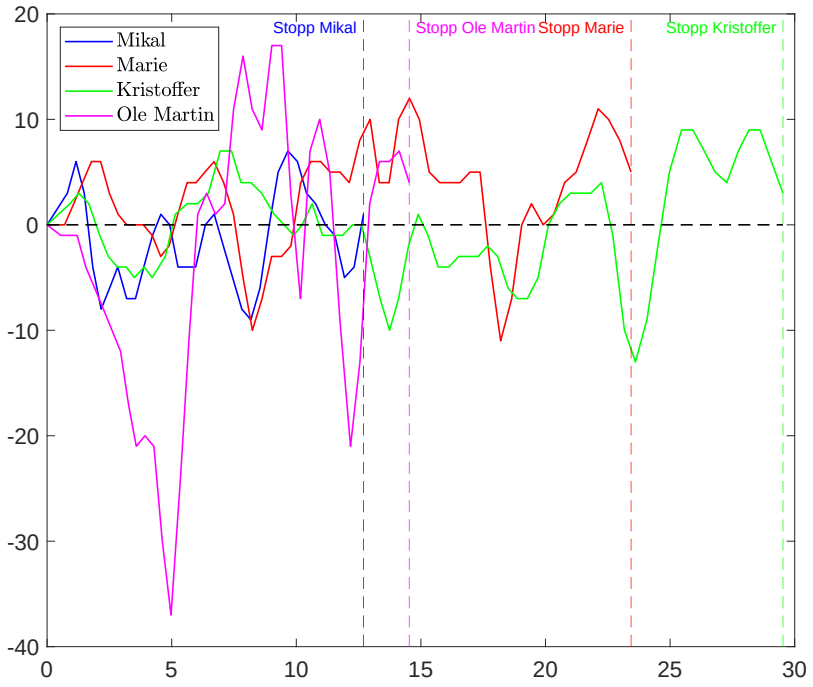
Figur 5.3 viser et av gruppemedlemmene sitt resultat, her kan vi se med hjelp av grafene for *Referanse* $r(k)$ *og* $lys(k)$ og *Avvik* $e(k)$ kan vi se hvor nøye roboten har kjørt på på samme gråskala på banen vist i figur 5.1. Dette vises best i grafen for *Referanse* $r(k)$ *og* $lys(k)$ da denne viser en referanse linje for den første avleste gråskala verdien. Med hjelp av den referanselinjen kan man lett se at roboten kjører ikke helt bent, den vingler litt frem og tilbake. Grafen for *Avvik* $e(k)$ viser det samme, men her har vi ikke referanselinjen, men da avviket skal helst være $e = 0$ så kan man se at roboten kjører noe vinglete gjennom løypen.

Videre kan vi se på grafen *Power A og B* den viser pådraget til motor A og motor B. Når roboten kjører rett frem så skal pådraget til motorene være like. Man kan se her og at roboten vingler mens den kjører da pådragene sjeldent er like. Det er merket av to punker i pådragene, i) og ii) disse punktene markerer starten og slutten på den største svingen som man kan se på figur 5.1



Figur 5.3: Eksempel på resultat fra kjøring, med referanse $r(k)$, måleverdiene $lys(k)$, avvik $e(k)$ og ulike kvalitetsmål.

For å starte å sammenligne resultatene skal vi starte å se på avviket $e(k)$ til hvert medlem, for å gjøre dette har vi lagt avviket for hvert medlem inn i figur 5.4. Det første man kan se er at alle gruppemedlemmene slet med å holde seg til referanse verdien for hver sin tur. Det kommer ikke helt tydelig frem i figur 5.4 hvem som kjørte best, det ser ut til å være mellom Marie og Kristoffer.



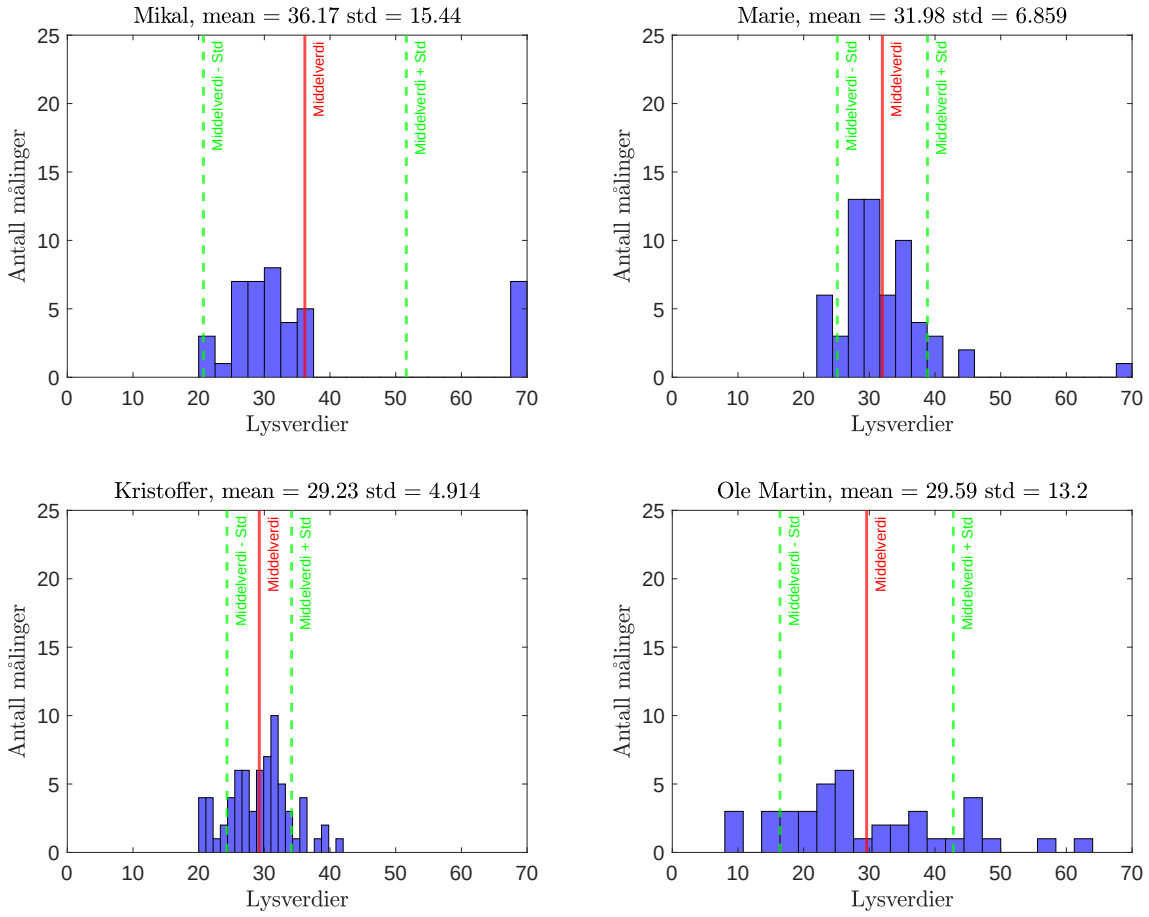
Figur 5.4: Reguleringsavviket under kjøring til hvert gruppemedlem. Blå: Mikal, rød: Marie, grønn: Kristoffer, rosa: Ole Martin

Videre så har vi lagt inn flere resultater fra kjøringene i tabell 5.1 nedenfor, for å gjøre det lettere å sammenligne resultatene. Her kan vi se hvor lang tid hvert medlem har brukt, referanse verdien til hver enkelt, middelverdien til lysmålingene, standardavvik, gjennomsnittlig avvik per måling (MAE), integral av absolutt avvik (IAE), samt den totale variasjonen i pådrag til motorene, antall målinger, og middelverdi av T_s.
Resultatene som er vurdert til å være best er uthevet.

Tabell 5.1: Sluttresultater fra manuell kjøring til deltagere i gruppe 2423.

	Ole M	Mikal	Marie	Kristoffer
Plattform	PC	PC	PC	PC
Strømkilde	230V	Batteri	Batteri	Batteri
Samtidig plotting?	Ja	Ja	Ja	Ja
Referanse `r`	26	28	34	29
middelverdi $\bar{y}$	29.59	36.17	31.98	**29.23**
$\lvert$`r`-$\bar{y}\rvert$	3.59	8.17	2.02	**0.23**
standardavvik σ	13.20	15.44	6.86	**4.91**
kjøretid [sek]	**14.5**	16	24.3	29.5
IAE	172.7	192.3	132.1	**118**
MAE	28.1	21.4	12.9	**12**
TV_A	375.6	**182.1**	251.9	233.6
TV_B	**156.9**	174.6	279.6	265.9
middelverdi av T_s [sek]	0.47	**0.37**	0.39	0.42
antall målinger (k)	39	41	60	69

Videre har vi lagt inn data for lysmålingene som er gjort underveis når roboten kjører i figur 5.5. Figur 5.5 viser spredningen av lysmålingene, og hvor mange målinger som er gjort for lysverdiene.



Figur 5.5: Histogram for sammenligning av resultat

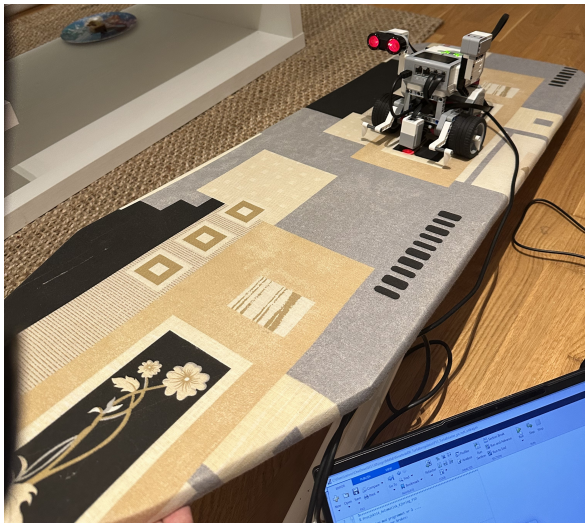
Vinneren kan vurderes på flere måter, hvis vi skal ta den som kjørte gjennom løypen raskest så blir det Ole Martin, om vi skal ta laveste middelverdi på T_s så blir det Mikal. Men jevnt over så kan det se ut til at Kristoffer kommer ut på topp til slutt.

Kapittel 6

Kjøring med turtallsregulator (utført av Ole Martin og Marie)

6.1 Introduksjon

I mange systemer er det viktig å opprettholde et stabilt turtall i motorer, også ved varierende motstand. Formålet med dette forsøket var å demonstrere hvordan vi kan stabilisere turtallet til en motor som møter motstand. For å simulere dette, lot vi først roboten kjøre på et flatt underlag. Deretter økte vi motstanden ved å løfte den ene enden av underlaget, som vist i figur 6.1. Dette scenariet etterligner forhold hvor motorer må tilpasse seg økt motstand for å stabilisere turtallet, som for eksempel *cruise*-kontrolleren i bilen i møte med motbakker.

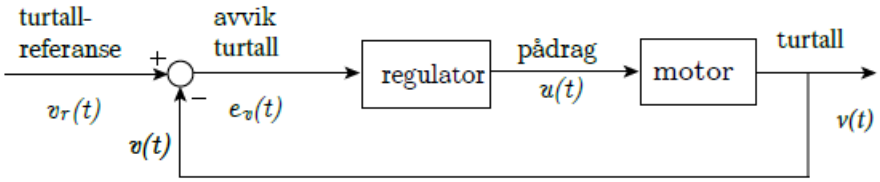


Figur 6.1: Roboten kjører på et underlag der vi kan løfte enden av underlaget i kjøreretning for å øke motstand.

6.2 Teori

Regulatorer er typisk brukt for å stabilisere turtallet i motorer i møte med motstand, for eksempel motbakker. Den teoretiske bakgrunnen for regulatorer ble gjennomgått i kapittel 4. Til dette forsøket benyttet vi en P-regulator, dvs. en regulator med kun proporsjonalleddet. Vi brukte følgende notasjon, som vist i den skjematiske fremstillingen av turtallsregulatoren i figur 6.2:

- $v(t)$ for måleverdiene av turtallet
- $v_r(t)$ for referanseverdien for turtallet
- $e_v(t)$ for avviket
- $u(t)$ for pådraget



Figur 6.2: Skjematisk fremstilling av turtallsregulatoren brukt i forsøket.

6.3 Kode

For å kunne gi et jevnt og stabilt pådrag til motoren, brukte vi *throttle*-kontrolleren på styrestikken vist i figur 6.3. Verdiene fra denne kontrolleren kan hentes i Matlab som vist i kodeutdrag 6.1

Figur 6.3: *Throttle*-kontroller på styrestikken som er brukt for å gi en stabil referanse-verdi.

Kode 6.1: Kode for å hente verdi fra *throttle*-kontrolleren på styrestikken.

```
% Data fra styrestikke.
[JoyAxes,JoyButtons] = HentJoystickVerdier(joystick);
JoyMainSwitch = JoyButtons(1);
JoyPot(k) = 0.2 * JoyAxes(4); % throttle-kontroller
```

Turtallsregulatoren ble videreutviklet fra teori og kode fra kapittel 4 og er vist i kodeutdrag 6.2. Referansen `vr` ble bestemt ut fra verdien registrert med *throttle*-regulatoren, `JoyPot(k)`, og et maksimalt motorpådrag, `maxPower`, mens turtallene til motorene, `v_A` og `v_B`, ble beregnet fra vinkelposisjonen til motorene. Regulatorparameteren K_p ble satt til 0.2, som var den optimale verdien vi fant i kapittel 4.

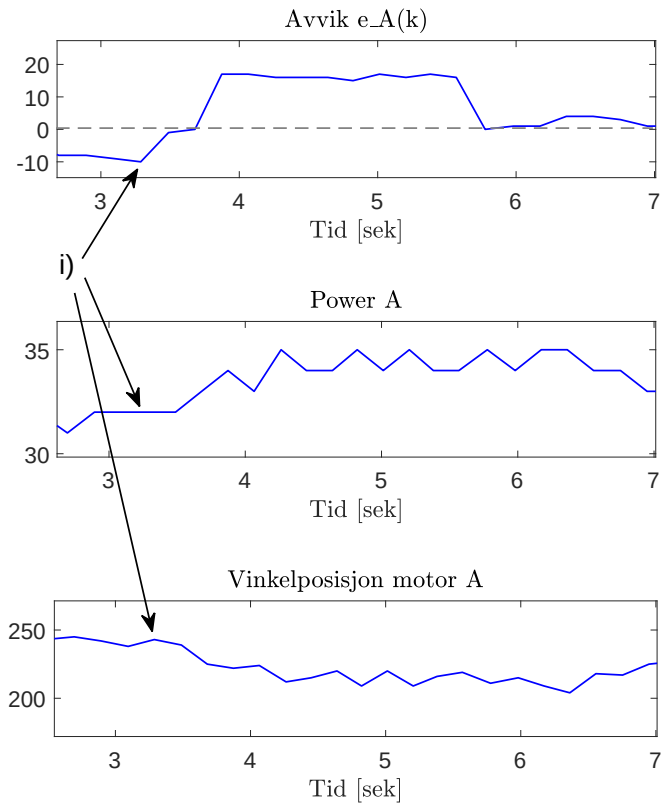
Kode 6.2: Kode for turtallsregulatoren (P-regulator).

```
% Initialverdier
Kp = 0.2;
maxPower = 10;

% Motorrotasjon
VinkelPosMotorA(k) = readRotation(motorA);
VinkelPosMotorB(k) = readRotation(motorB);

% JoyPot endrer ønsket hastighet lineært
vr = JoyPot(k) * maxPower; % Juster dette etter behov

% Beregn faktisk hastighet (turtall) som endring i ...
    vinkelposisjon delt på tid
v_A = (VinkelPosMotorA(k) - VinkelPosMotorA(k-1)) / Ts;
v_B = (VinkelPosMotorB(k) - VinkelPosMotorB(k-1)) / Ts;

% Beregn avviket (e_A/B) mellom ønsket (vr) og faktisk (A/B) ...
    hastighet
ev_A = vr - v_A;
ev_B = vr - v_B;

% Beregn P-pådrag for hver motor
U_A = Kp * ev_A
U_B = Kp * ev_B
```

6.4 Resultat

Resultatet av kjøringen med turtallsregulator er vist under i figur 6.4. Roboten kjører på flatt underlag de første cirka 3.5 sekundene før underlaget løftes i ene enden. Som vi ser i figuren, synker da turtallet (gitt som vinkelposisjon i figuren) samtidig som avviket `e_A(k)` for motor A øker. For å kompensere for fallet i turtall, øker pådraget til motoren (`Power A`). Etter hvert minker avviket igjen og stabiliserer seg rundt 0. Dette indikerer at turtallsregulatoren fungerer som

forventet.



Figur 6.4: Resultat av forsøket med turtallsregulator. Ved *i)* løftes ene enden av underlaget slik at roboten kjører i motbakke. Som resultat øker avviket mens turtallet, gitt som vinkelposisjon, minker, og regulatoren øker pådraget for å kompensere.

Kapittel 7

Kjøring med PID (utført av Ole Martin, Marie og Mikal)

7.1 Introduksjon

Vi ønsket å fortsette arbeidet med å demonstrere bruken av regulatorer i praksis. Vi gjorde dermed et forsøk hvor vi programmerte LEGO-roboten til å kjøre automatisk etter en gitt referanseverdi på gråskalabanen vi brukte til manuell kjøring i kapittel 5. Vi brukte ulike regulatorer for å beregne blant annet hvordan roboten oppførte seg i svinger og hvor fort den kom seg gjennom banen. I dette kapittelet tar vi for oss dette forsøket og sammenligner resultatene fra de ulike regulatorene og den manuelle kjøringen.

7.2 Teori

Den teoretiske bakgrunnen for dette forsøket er beskrevet i kapittel 4 for regulatoren, og i kapittel 5 for kjøring av roboten og de kvalitetsmålene vi bruker for sammenligning. I forsøket har vi testet både P-, PI- og PID-regulator.

7.3 Kode for regulator og automatisk kjøring

Vi implementerte regulatoren for å justerte pådraget til motorene for at roboten kunne svinge. Dette justeringen, `u_A`, ble beregnet basert på ligning 4.4 som vist i kodeutdrag 7.1. Vi lot da basispådraget til regulatoren $u_0 = 0$ og brukte i stedet en starthastighet for hver motor, `grunnfart`, som sørget for at roboten kjørte framover med en gitt hastighet.

Kode 7.1: Utdrag fra koden for PID-regulator i løkke for å beregne justering av pådrag på motorene for å svinge.

```
%Leser posisjon på motorer
VinkelPosMotorA(k) = readRotation(motorA);
VinkelPosMotorB(k) = readRotation(motorB);

if k > 1
    y = Lys(k);
    r = Lys(1);
    e(k) = r - y;
    Ts(k) = Tid(k) - Tid(k-1);
    P(k) = Kp * e(k);
    I(k) = min(max(I(k-1) + Ki * e(k) * Ts(k), I_min), I_max);
    D(k) = Kd * (e(k) - e(k-1)) / Ts(k);
    u_A = u0 + P(k) + I(k) + D(k);
    ...
end
```

Beregningen av det endelige pådraget til hver motor, `PowerA` og `PowerB`, basert på grunnfarten og justeringen fra regulatoren, er vist i kodeutdrag 7.2. Pådraget ble begrenset av et maksimalt motorpådrag, `maxPower`.

Kode 7.2: Justering av pådrag i løkken for å gi motorpådrag.

```
% Oppdater pådrag for hver motor
PID_padrag = max(min(u_A, maxPower), -maxPower);
PowerA(k) = max(min(grunnfart + PID_padrag, maxPower), -maxPower);
PowerB(k) = max(min(grunnfart - PID_padrag, maxPower), -maxPower);
motorA.Speed = PowerA(k);
motorB.Speed = PowerB(k);
if online
    start(motorA);
    start(motorB);
end
```

7.4 Tuning av regulatorparametre og grunnfart

Vi bygde videre på arbeidet i kapittel 4 og justerte regulatorparametrene ytterligere for å gi det mest optimale resultatet for kjøringen. Dette gjorde vi ved å doble verdiene fram til vi ikke lenger observerte merkbar forbedring, hvorpå vi reduserte verdiene med 20 % for å finjustere ytelsen. Det optimale resultatet ble da $K_p = 2$ og $K_i = 0.02$. Vi fant ingen forbedringer i ytelse ved justeringer av K_d med forsøk i intervallet $0.1 - 0.001$. Vi valgte derfor å utelate D-leddet fra regulatoren i sluttresultatet.

Videre testet vi hvor høy hastighet roboten var i stand til å håndtere ved å øke grunnfarten. For sluttresultatene var denne opprinnelig satt lik 20. Da vi økte grunnfarten til 25, observerte vi at roboten lyktes med å fullføre løypen først etter flere forsøk. Dette tyder på at et motorpådrag på 25 som grunnfart, ligger nær grensen for hva roboten kan håndtere.

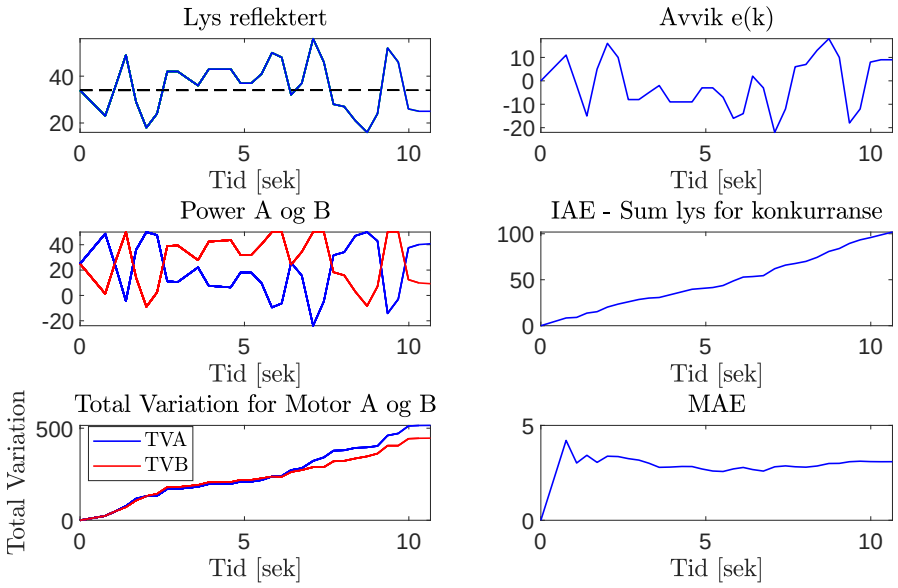
7.5 Resultat

Resultatene av kjøringene med P- og PI-regulator, samt en sammenligning med beste resultat fra manuell kjøring (Kristoffer sin kjøring), er oppsummert i tabell 7.1. Under disse testene observerte vi at standardavviket σ, IAE og MAE ble betydelig forbedret da integralleddet ble implementert, gitt en grunnfart på 20. Vi ser da igjen effekten av dette leddets evne til å dempe feil og forbedre ytelsen til systemet. Derimot øker totalvariasjonen TV, hvilket antyder at integralleder fører til mer regulering av pådraget, hvilket igjen kan føre til mer slitasje på utstyret. Kostnaden ved økt kvalitet kan altså bli mer slitasje. Videre observerer vi at å øke grunnfarten til 25 reduserer kvalitetsmålene. Den økte hastigheten gjorde at roboten kjørte raskere utenfor referanseverdien, og avvikene fra referanseverdien blir da større, som vi observerer i figur 7.1. Større avvik fører til at det må gjøres flere justeringer for å følge løypen, og dermed øker også TV.

Tabell 7.1: Sluttresultater fra automatisk kjøring.

Regulator	P	PI	PI	Manuell (Kristoffer)
Plattform	PC	PC	PC	PC
Strømkilde	Strøm	Batteri	Batteri	Batteri
Samtidig plotting?	Ja	Ja	Ja	Ja
Grunnfart	20	20	25	N/A
K_p	2	2	2	N/A
K_i	N/A	0.02	0.02	N/A
Referanse `r`	28	28	29	29
Middelverdi $\bar{y}$	32.32	30.64	32.55	29.23
$\lvert$`r`-$\bar{y}\rvert$	4.32	2.64	3.55	0.23
Standardavvik σ	9.14	8.62	11.31	4.91
Kjøretid [sek]	10.91	10.81	9.40	29.5
IAE	60.02	54.5	74.2	118
MAE	13.74	12.75	17.37	12
`TV_A`	324	379	521	233.6
`TV_B`	318	367	403	265.9
Middelverdi av T_s [sek]	0.24	0.22	0.21	0.42
Antall målinger (k)	47	49	44	69

Sammenlignet med manuell kjøring resulterte den automatiske kjøringen i raskere gjennomkjøring i alle forsøk. Kvalitetsmålet IAE er i alle tilfeller bedre for den automatiske kjøringen. MAE er tilsynelatende bedre for manuell kjøring, men kjøretiden er også nesten det tredobbelte. Sammenlignet med MAE med kjøretider nærmere kjøretidende ved automatisk kjøring (se tabell 5.1), er resultatet for automatisk kjøring betraktelig bedre. For TV er resultatene for det meste bedre for manuell kjøring. Vi ser at TV ligger høyere for automatisk kjøring enn for den beste manuelle kjøringen, også med hensyn på tid. En høy TV indikerer at det er høyere slitasje på utstyret, så en god manuell pilot kan i dette tilfelle gjøre kjøringen mer skånsomt enn programmet.



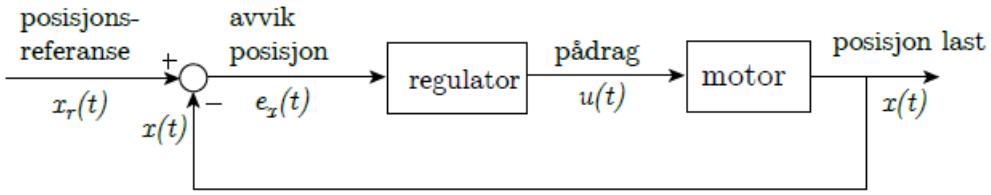
Figur 7.1: Resultat av automatisk kjøring med PI-regulator og økt fart (`grunnfart = 25;`.

Kapittel 8

Hengende last (utført av Ole Martin)

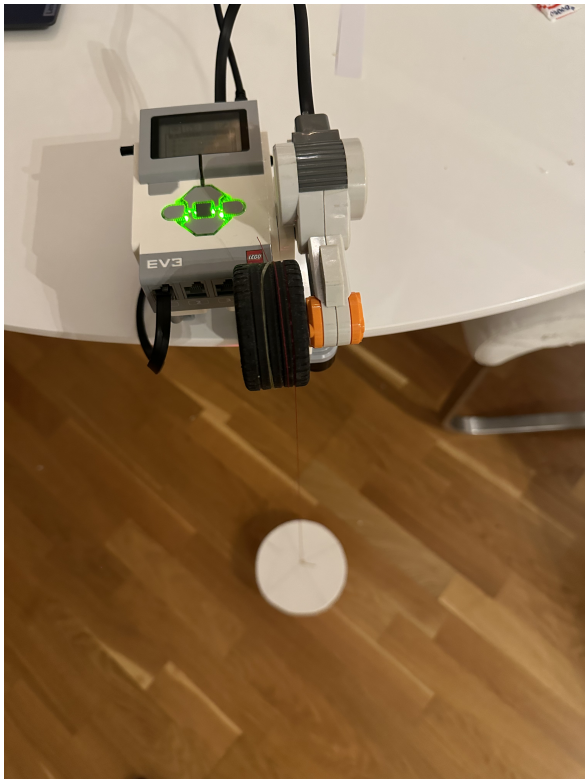
8.1 Introduksjon

I dette kapittelet programmerer vi legoutstyret til å simulere en krane med last. I programmet vil brukeren oppgi startposisjon og en ønsket sluttposisjon, og vi bruker en ultralydmåler til å måle avstanden for referanse. I en faktisk praksis så ville det i mange situasjoner vært mest praktisk om slutttposisjonen ble lagret som ny startposisjon, men det anser vi ikke som hensiktsmessig når vi utfører forsøkene.

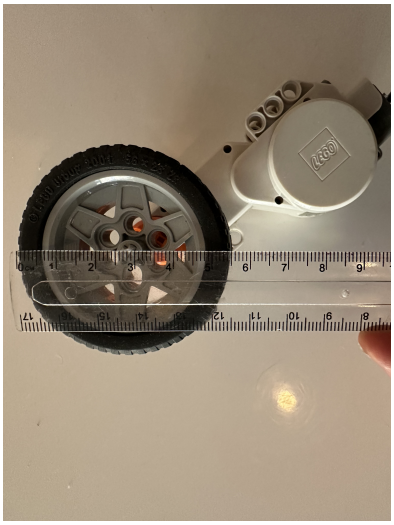


Figur 8.1: Flytdiagram for å beregne pådrag for løft, plukket fra oppgave tekst. Figuren er hentet fra [1].

Vi måler diameteren til hjulet for å beregne omkretsen.



Figur 8.2: Legoutstyr med tråd festet til hjulet



Figur 8.3: Måler diameter

8.2 Teori

Forsøkene og resultatene bygger på teorien gitt i kapittel 4

8.3 Utdrag av kode

Kode 8.1: Input fra bruker og beregning av trommel

```

%Spør burkeren om input for startposisjon og ønsket posisjon
    StartPosisjon = input('Oppgi faktisk startposisjon i meter: ');
    SluttPosisjon = input('Oppgi ønsket sluttposisjon i meter: ');

%Bergening av trommeldiameter
    trommelDiameter = 0.055; % Diameter i meter
    trommelOmkrets = pi * trommelDiameter; % Beregner omkretsen
```

Kode 8.2: Løkke

```
while true
    rotasjon(k) = motorA.readRotation;
    x(k) = ((double(rotasjon(k)) / 360) * trommelOmkrets + ...
        faktiskStartPosisjon);
    Tid(k) = toc(startTid);
    ex(k) = onsketSluttPosisjon - x(k);

    % Posisjons P-regulator
    vr(k) = Kp * ex(k);
    vr(k) = min(vr(k), maksFart);

    % Faktisk hastighet
    if k > 1
        v(k) = double(x(k) - forrigeX) ./ double(Tid(k) - ...
            forrigeTid) ./ trommelOmkrets;  % Hastighet i ...
            rotasjon per sekund
    else
        v(k) = 0;
    end

    ev(k) = vr(k) - v(k);  % Hastighetsavvik

    % Hastighets PI
    if k > 1
```

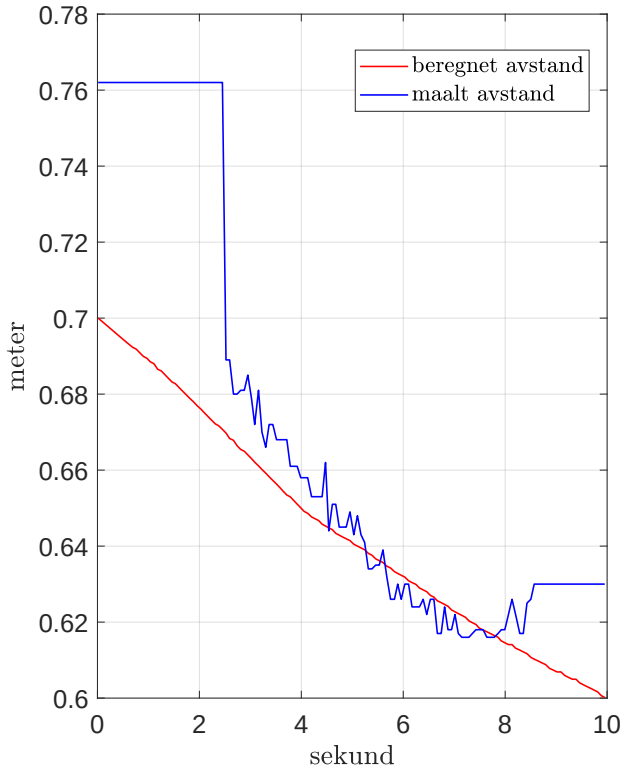
```
    integrator_v(k) = integrator_v(k-1) + ev(k);
    else
    integrator_v(k) = 0;
    end

    u(k) = Kp_v * ev(k) + Ki_v * integrator_v(k);  % Pådrag

    if Tid(k) < 2 %leggger inn en begrensing på fart i 2 sekund ...
        for å unngå rykk
    maksMotorspeed = 50 * Tid(k);
    u(k) = min(u(k),maksMotorspeed);
    end

    motorA.Speed = u(k);
    motorA.start;

    if abs(ex(k)) < 0.01  % Stopper når avviket fra ønsket ...
        avstand er mindre enn 1 cm.
        disp('Nådd ønsket posisjon.');
        break;
    end

    if Tid(k) > 10  % Timeout for sikkerhet
        disp('Timeout, stopper løftet.');
        break;
    end

    % Lagre forrige verdier
    forrigeFeil = ex(k); forrigeEv = ev(k);
    forrigeX = x(k); forrigeTid = Tid(k);
    k = k + 1;
end
```

8.4 Gjennomføring og resultat

Vi starter forsøket ved å sette startposisjonen for lasten til 0.7 meter og sluttposisjonen til 0.6 meter. Utfra dataene fra ultralydmåleren ser vi at sensoren ikke alltid fanger opp lasten, noe som fører målingene i starten og slutten ikke er gyldige. Derimot, i de tilfellene hvor målinger er tilgjengelige, korresponderer disse godt med den beregnede posisjonen. Dette indikerer at avstandsberegningen, som tar hensyn til trommelens bevegelser, er nøyaktig.

Eksperimentet ble utført ved bruk av en PI-regulator som regulerer fart ut fra

avstanden mellom ønsket og beregnet avstand, og gjennomført med en gjennomsnittlig lav hastighet på 1 cm/s. Vi ser i figur 8.4 regulatoren gir pådrag helt til den beregnede avstanden er lik den ønskede avstanden på 0.6 m, med noe høyere hastighet i begynnelsen.



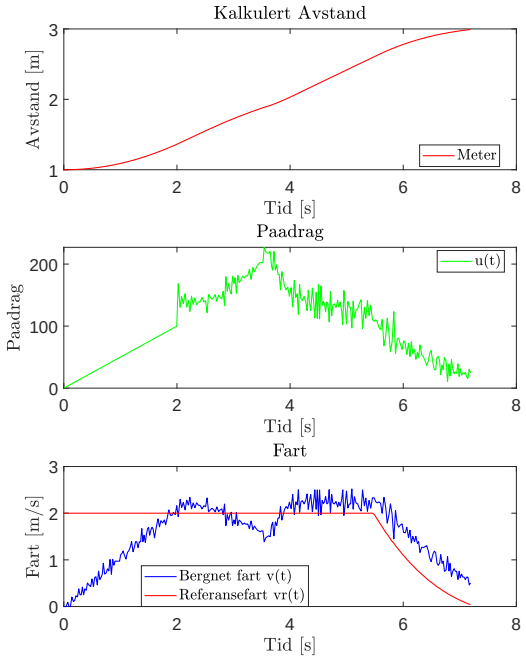
Figur 8.4: Løft med en enkel PI-regulator og avstand målt med ultralyssensor.

I neste forsøket vil vi legge inn dobbel regulator tar hensyn til variasjon i motstanden i en kran. Dette for eksempel værforhold eller en vinsj som trekker en last langs bakken, hvor motstanden kan variere.

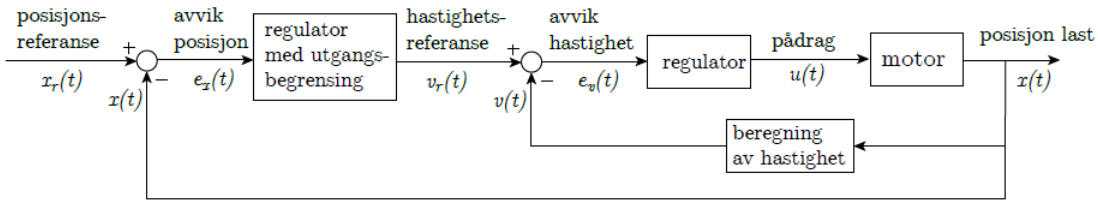
Vi forenkler bruken av regulatoren som regulerer bergner fart en P-regulator som gir referanse hastighet $v_r(t)$.. Den andre regulatoren er en PI-regulator som justerer pådraget dersom det blir avvik mellom referansefarten og den målte farten.

Pådraget justeres for å oppnå en hastighet nærmere en gitt hastighetsreferanse. I tillegg er det implementert en fartsgrense i de første to sekundene av eksperimentet for å minimere risiko for rykk ved start.

Forsøket, i motsetning til det med enkel regulering, utføres uten last. For å simulere motstand, plasseres en finger på hjulet. I løpet av de første to sekundene observeres en gradvis økning i den beregnede farten $v(t)$ på grunn av hastighetsbegrensningen. Hastigheten stabiliserer seg rundt referansehastigheten $v_r(t)$. Etter omtrent tre sekunder påføres motstand på hjulet. Når den beregnede farten synker, øker hastighetsregulatoren pådraget $u(t)$ til motstanden slippes rett før det fjerde sekundet. Hastigheten $v(t)$ går da tilbake til området rundt referansefarten $v_r(t)$. Mot slutten av det simulerte løftet følger den beregnede farten samme bane som referansefarten, helt til løftet avsluttes.



Figur 8.5: Resultat fra løft med dobbel regulator



Figur 8.6: Figuren er en bearbeidet figur hentet fra [1].

8.4.1 Oppsummert

Det første forsøket viste at vi ved hjelp av en PI-regulator kan styre en krane med å føre inn ønsket høyde, og verifisere at lasten har kommet til høyden med hjelp av avstandsmålinger.

Det andre forsøket viste at vi ved bruk av en sekundær hastighetsregulator kunne justere pådraget, slik at farten på kranen kunne tilnærme seg den ønskede hastighetsreferansen $v_r(t)$ selv under forstyrrelser som simuleres av en finger på hjulet. Den innførte hastighetsbegrensningen i de første to sekundene var også effektiv for å forhindre rykk og gi en jevnere start på bevegelsen.

Kapittel 9

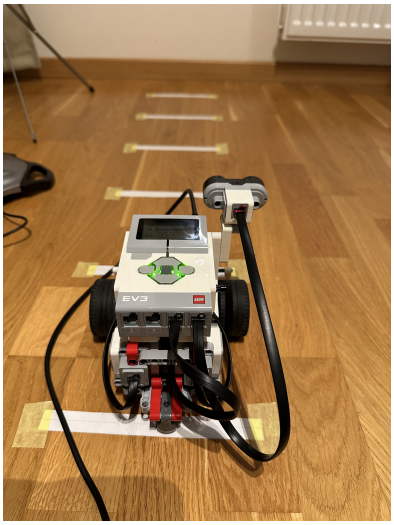
Estimert hastighet (utført av Ole Martin og Mikal)

9.1 Introduksjon

I dette forsøket skal vi gjennomføre hastighetsmålinger ved hjelp av sensorer fra LEGO-utstyret. Vi har plassert lyse teipstriper på underlaget med 10cm eller 20 cm mellom hver stripe. En lyssensor, som er rettet ned mot gulvet, vil registrere refleksjonene fra disse stripene.

I tillegg vil en ultralydsensor bli benyttet for å måle avstanden til en vegg. For å beregne farten vil vi bruke avstandsmålingene, den numerisk deriverte av avstandsmålingene, og dataene fra lyssensoren. Vi forventer at de ulike beregningene skal gi samme resultat. Vi gjør flere forsøk, der første forsøk er med 20 cm mellom fartstripene, andre forsøk med 10 cm mellom stripene og 3. forsøk med økende fart.

Vi utvider med å bygge på prosjektet videre der vi øker hastigheten til roboten over tid. Dette for å beregne akselerasjon og finne ut om vi kan tilbakeregne til posisjons endringer.



Figur 9.1: Lyssensoren som peker ned mot gulvet gir utslag når lego-utstyret kjører over de lyse teipstripene

9.2 Teori

Teori for de innværende delene i dette forsøket er beskrevet i kapittel 2 Filtrering og 3 Derivasjon.

9.3 Matlab-kode

Nedenfor er utdrag fra de viktigste punktene i MATLAB-koden for å registrere målinger og beregne volum basert på lysintensitetsmålinger:

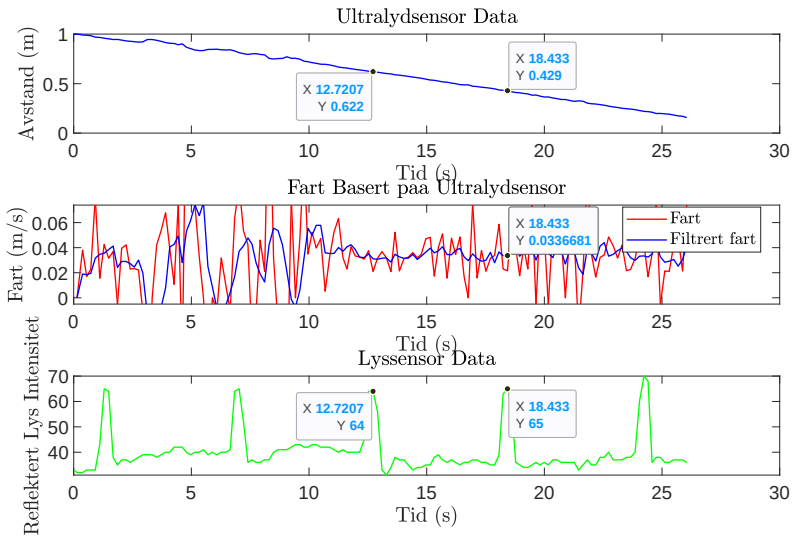
```

% Legger inn sensorer og motorer
    myUltrasonicSensor = sonicSensor(myev3);
    myColorSensor = colorSensor(myev3);

    MotorA = motor(myev3, 'A');
    MotorB = motor(myev3, 'B');

% kode for å sette fart og økning i fart
```

```
startSpeed = 10;
okning = 0;

%Viktigste kode i løkke:
    % Datainnsamling
    if online
    avstand(k) = double(readDistance(myUltrasonicSensor));
    Lys(k) =double(readLightIntensity(myColorSensor, 'reflected'));
    tid(k) = toc;
    end


    % Beregning av fart, og inkludert kode for IRR- og FIR-filter
    fart(k) = (avstand(k) - avstand(k-1)) / (tid(k) - tid(k-1));
    filtrert_fart_IRR(k) = IIR_filter(OldFilteredValue, ...
        fart(k), alfa);
    filtrert_fart_FIR(k) = FIR_filter(fart(2:k), M); % ...
        Eksempel, erstatt med faktisk logikk
    OldFilteredValue = filtrert_fart_IRR(k);


    %Beregning av den deriverte av fart og filtrert fart:
    fart_derivert(k) = BakoverDerivasjon([fart(k-1), fart(k)], ...
        tid(k)-tid(k-1));
    fart_filtrert_derivert(k) = ...
        BakoverDerivasjon([filtrert_fart_IRR(k-1),filtrert_fart_IRR(k)], ...
        tid(k)-tid(k-1));

    %Kode for å gi pådrag til motorene
    MotorA.Speed = MotorA.Speed + (okning * tid(k));
    MotorB.Speed = MotorB.Speed + (okning * tid(k));
```

9.4 Gjennomføring og resultat

9.4.1 Forsøk nr. 1

I første forsøk hadde vi 20cm avstand mellom fartstripene. Som vi ser ut fra målingene i figur 9.2 under har vi valgt ut 2 punkter i grafene for avstand og reflektert lysintensitet vi skal bevise. Dette for å vise at koden våres utfører de rette beregningene. For å finne hastigheten v bruker vi uttrykket nedenfor (9.1) og setter inn for punktene vi har valgt oss ut.



Figur 9.2: Resultat fra første kjøring med 20cm avstand mellom fartstriper.

Formel for utregning av hastighet (Endring i posisjon over endring i tid):

$$v = \frac{\Delta x}{\Delta t} = \frac{x_2 - x_1}{t_2 - t_1} \tag{9.1}$$

- Setter inn verdiene fra de utvalgte punktene i avstandsmålingen fra Ultralydsensoren og regner ut hastigheten:

$$v = \frac{0.425\,\mathrm{m} - 0.622\,\mathrm{m}}{18.433\,\mathrm{s} - 12.72\,\mathrm{s}} = \frac{-0.197\,\mathrm{m}}{6.163\,\mathrm{s}} = \frac{-0.197\,\mathrm{m}}{6.163\,\mathrm{s}} \approx -0.032\,\frac{\mathrm{m}}{\mathrm{s}}$$

- Setter inn punktverdiene fra Lyssensor Data og regner ut hastigheten: For å beregne hastigheten ved bruk av lyssensoren bruker vi avstanden mellom fartstripene og når disse blir målt som referanse til posisjonen. Siden avstanden er 20cm eller 0.2m vil utregningen se slik ut:

$$v = \frac{0.2\,\mathrm{m} - 0.0\,\mathrm{m}}{18.433\,\mathrm{s} - 12.27\,\mathrm{s}} = \frac{0.2\,\mathrm{m}}{6.163\,\mathrm{s}} \approx 0.032\,\frac{\mathrm{m}}{\mathrm{s}}$$

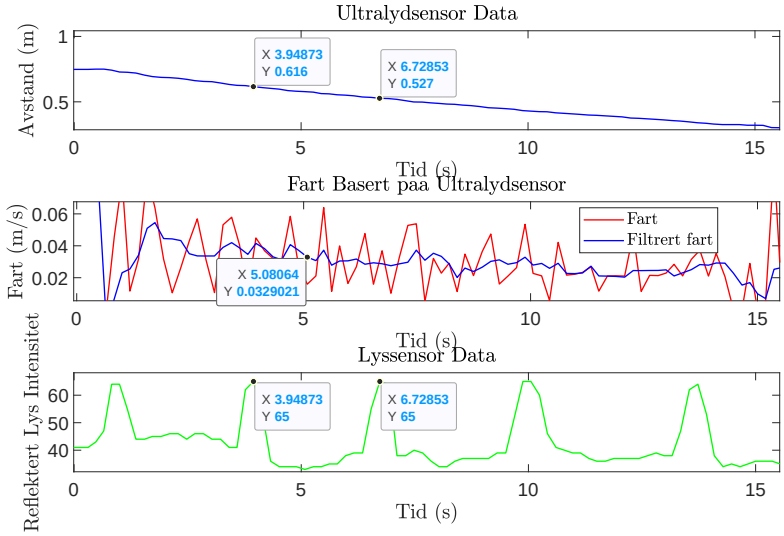
- Filtrert og numerisk derivert fartsutregning basert på ultralydsensoren ved t = 18.4:

$$v \approx 0.034\,\frac{\mathrm{m}}{\mathrm{s}}$$

Vi ser derfor at utregningen av hastigheten blir den samme som koden våres. Dette beviser at målingene vi har gjort gir resultater vi kan stole på.

9.4.2 Forsøk nr. 2

I andre forsøk reduserte vi avstanden mellom fartstripene til 10cm og gjorde samme målinger, se figur 9.3 og utregningene nedenfor.



Figur 9.3: Resultat fra andre kjøring med halvert avstand mellom teipstripene på 10 cm

- Setter inn verdiene fra de utvalgte punktene i avstandsmålingen fra Ultralydsensoren og regner ut hastigheten:

$$v = \frac{0.527\text{m} - 0.616\text{m}}{6.73\text{s} - 3.95\text{s}} = \frac{-0.089\text{m}}{2.78\text{s}} \approx -0.0320\,\frac{\text{m}}{\text{s}} \approx -0.0320\,\text{m/s} \quad (9.2)$$

- Setter inn punktverdiene fra Lyssensor Data og regner ut hastigheten:

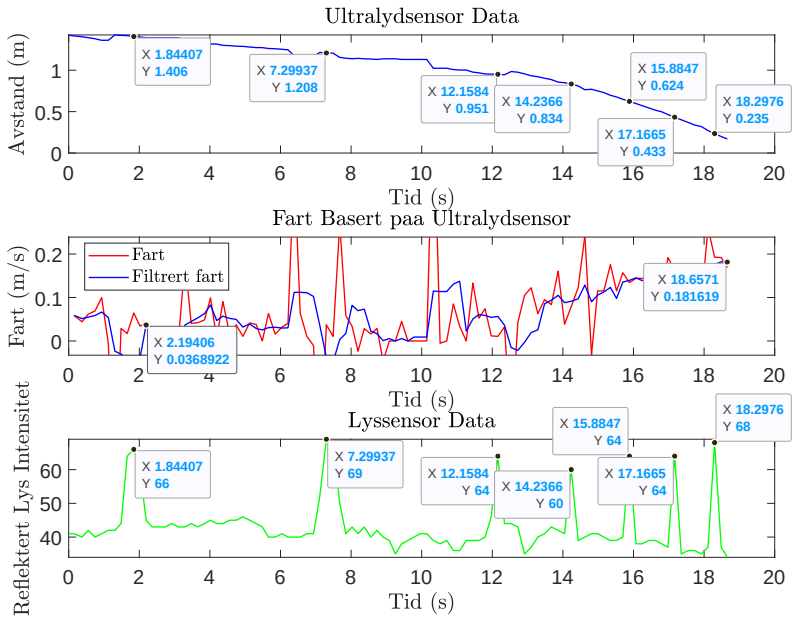
$$v = \frac{0.1\text{m} - 0\text{m}}{6.73\text{s} - 3.95\text{s}} = \frac{0.1\text{m}}{2.78\text{s}} \approx 0.036\,\text{m/s} \quad (9.3)$$

- Filtrert og numerisk derivert fartsutregning basert på ultralydsensoren ved t = 5.08:

$$v = 0.0329\,\mathrm{m/s}$$

Vi ser at de ulike beregningene av fart gir en fart på samme nivå.

9.4.3 Forsøk nr. 3



Figur 9.4: Resultat fra tredje kjøring med økende hastighet på roboten

I forsøket har roboten fått et økende pådrag etter hvert som tiden går. Vi kan bergegne akselerasjonen ved å velge et par punkter i plottet som viser fart.

Vi har valgt punktene $v_1 = 0.0368\,\mathrm{m/s}$ ved $t_1 = 2.19\,\mathrm{s}$ og $v_2 = 0.182\,\mathrm{m/s}$ ved $t_2 = 18.66\,\mathrm{s}$.

Akselerasjon a defineres som endringen i hastighet delt på tidsintervallet:

$$a(t) = \frac{\Delta v}{\Delta t} = \frac{v_2 - v_1}{t_2 - t_1} = \frac{0.182\,\mathrm{m/s} - 0.0368\,\mathrm{m/s}}{18.66\,\mathrm{s} - 2.19\,\mathrm{s}} \approx 0.0088\,\mathrm{m/s}^2$$

Med kjent akselerasjon kan vi beregne den opprinnelige hastigheten v_0 ved $t = 0$:

$$v_0 = v_1 - a \cdot t_1 = 0.0368\,\mathrm{m/s} - 0.0088\,\mathrm{m/s}^2 \cdot 2.19\,\mathrm{s} \approx 0.0175\,\mathrm{m/s}$$

Farten kan nå beregnes som integralet av akselerasjonen

$$v(t) = v_0 + a \cdot t = 0.0175\,\mathrm{m/s} + 0.0088\,\mathrm{m/s}^2 \cdot t$$

Videre kan vi beregne tilbakelagt avstand $x(t)$ som integralet av farten:

$$x(t) = v_0 \cdot t + \frac{1}{2} \cdot a \cdot t^2 = 0.0175\,\mathrm{m/s} \cdot t + \frac{1}{2} \cdot 0.0088\,\mathrm{m/s}^2 \cdot t^2$$

Vi har nå tre ulike utgangspunkt for tilbakelagt avstand. Vi har ultralydmåling, lysmålingene fra teipstripene og dobbeltintegralet av akselerasjoen. Resultatene er samlet i tabellen nedenfor.

Avstandsmålinger

Tid i sekund	1,84	7,30	12,16	14,24	15,88	17,17	18,30
Avstand med beregnet med utgangspunkt deriverte av fart i meter	0,01	0,33	0,85	1,14	1,39	1,60	1,80
Avstand ultralysmåling i meter	0,00	0,20	0,46	0,57	0,78	0,97	1,17
Avstand Reflektert lys i meter	0,00	0,20	0,40	0,60	0,80	1,00	1,20

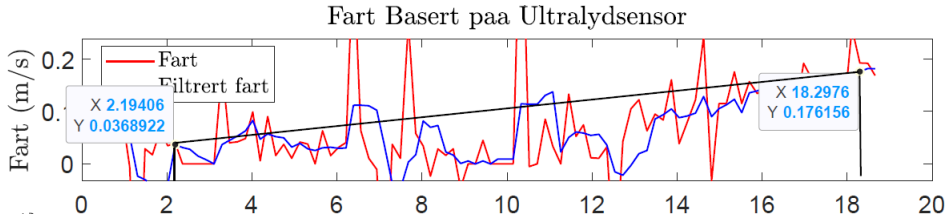
Figur 9.5: Resultater fra avstandmålinger

Vi ser at ultraltsmålingene og beregningene fra teipstripene gir resultater som harmonerer. Beregningen av dobbeltintegralet av akselerasjonen øker som forventet med ca 20 cm på de tre siste målingene, men ellers er ikke målingene i tråd med forventing. Videre analyse og sammenligning med akselerajonen beregnet fra ultralys og lysmålinger, indikerer at årsaken kan være at akselerasjon øker ujevnt, og ikke linært som forutsatt i være beregninger.

Tid i sekund	1,84	7,30	12,16	14,24	15,88	17,17	18,30
Fart UL m/s		0,036	0,053	0,056	0,128	0,148	0,175
Fart lys m/s		0,037	0,041	0,096	0,122	0,155	0,177
Akselerasjon UL m/s^2			0,003	0,001	0,035	0,012	0,021
Akselerasjon lys m/s^2			0,001	0,011	0,012	0,020	0,017

Figur 9.6: Beregnet fart og akselerasjon

Vi tar noen siste analyser av resultat, arealet under en linje trukket mellom punktet vi har brukt for den numeriske derivasjonen av farten skal gi samme resultat. Vi har ulike beregning av startpunktet, men dette vil ikke ha noen signifikant betydning for vårt formål.



Figur 9.7: Resultater fra avstandmålinger

Arealet under linjen er beregnet til omtrent 1.80 m2, som er samme resultat som avstandsmålingen basert på akselerasjonen, noe som indikerer at avstandsmålingene basert på akselerasjon er korrekte. Videre får vi en ny indikasjon på at akselerasjonen er varierende gjennom målingene.

Kapittel 10

Konklusjon

I rapporten har vi sett på og anvendt sentrale teorier innen robotprogrammering gjennom en rekke ulike forsøk med LEGO-utstyr og MATLAB-programmering. Via numerisk integrasjon, derivasjon, filtrering, og PID-regulering, har vi vist praktisk anvendelse av matematisk og fysisk teori.

Numerisk integrasjon ble gjennomgått ved simulering av væskestrøm, hvor vi med bra nøyaktighet kunne måle volumendringer over tid. For filtrering ble filtreringsteknikker anvendt på ulike data for å redusere støy og gjøre signalene signalene mer anvendbare, f.eks gjennom filtrering av lydsignal.Både FIR- og IIR-filtre er blitt brukt for å vise hvordan ulike filterkonfigurasjoner påvirker signalbehandlingen. I forsøkene med numerisk integrasjon fikk vi også vist en anvendelse av teorien i en mer praktisk situasjon der det kan være nødvendig med justeringer av den grunn. PID-reguleringen er blitt testet gjennom flere ulike forsøk med variabel hastighet og belastning, der vi har finjusterte parametere for optimal regulering. Dette resulterte i en mer stabil og kontrollert drift i de ulike situasjonene.

Vi har brukt kompetansen fra de første 5 kapittelene for å gjennomføre de mer kreative oppgavene da disse bygger naturlig på hverandre. En regulator blir for eksempel brukt videre i det kreative prosjektet om automatisk kjøring av legoroboten, turtallsregulatoren og hengende last.

Eksperimentene har ikke bare bekreftet gyldigheten av de anvendte teoriene, men også belyst områder hvor ytterligere justeringer og forbedringer kan hentes. Slike innsikter er viktige for å kunne programmere roboter til praktiske situasjoner.

Konklusjon

Rapporten viser hvordan forståelse og anvendelse av matematisk og fysisk teori kan brukes i praksis gjennom programmering.

Bibliografi

[1] T. Drengstig. Prosjektdel av ELE130 Anvendt matematikk og fysikk i robotprogrammering. Lego Mindstorms og MATLAB/Python i skjønn forening. Technical report, Universitet i Stavanger, 2023. Utdelt materiale.

[2] T. Drengstig. Kompendium i ELE130 Anvendt matematikk og fysikk i robotprogrammering. Utdelt pensumsmateriale, 2024.

[3] S. W. Smith. *The Scientist and Engineer's Guide to Digital Signal Processing*. California Technical Publishing, 2002. Hentet fra https://www.dspguide.com/.

Vedlegg A

Timelister

Tabell A.1: Timelister for gruppe 2423 i faget ELE130.

Navn/uke	Marie	Kristoffer	Ole Martin	Mikal
11	6	0	9.5	
12	7	0	9	
13	3	0	14	
14	6	10	18	
15	10	20	21	19
16	6	10	24	17
17	21	20	19	49
Sum	59	60	114.5	85