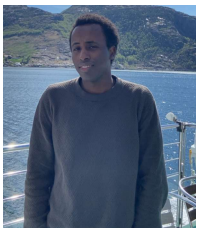
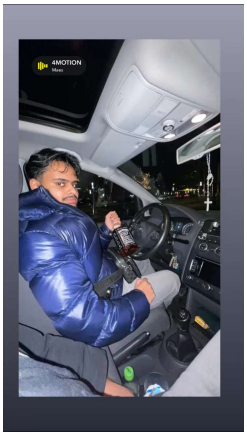
ELE130 Anvendt matematikk og fysikk i robotprogrammering

Universitetet i Stavanger

PROSJEKT - VÅREN 2025

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

Gruppenummer	2561		
Gruppens medlemmer	Navn	Studentnummer	Bilde
	Abdirahman Abdi Ibrahim	283989	
	Wijenthan Wijayaseelan	283995	

Innhold

Innhold **ii**

Sammendrag **vi**

1 Numerisk integrasjon **1**

1.1 Introduksjon 1

1.1.1 Trapesmetoden med variabelt tidssteg 2

1.2 Relasjon til det bestemte integral som en konstant 4

1.3 Relasjon til det bestemte integral som en sinusfunksjon 7

1.4 Numerisk integrasjon av *chirp*-signal 11

1.4.1 betydningen av høyere vinkelhastighet 11

2 Filtrering (utført av hele gruppen) **13**

2.1 Introduksjon 13

2.2 Første ordens lavpassfilter i termometer 15

2.3 Lavpass- og høypassfiltrering av støyfylt signal 20

3 Numerisk derivasjon (utført av hele gruppen) 24

3.1 Introduksjon 24

3.2 Relasjon til stigningstall som en tallverdi 26

3.3 Relasjon til den deriverte som en sinusfunksjon 29

3.4 Numerisk derivasjon av *chirp*-signal 34

3.4.1 Betydningen av høyere vinkelhastighet i derivasjon 34

4 PID regulatoren (utført av hele gruppen) 35

4.1 Introduksjon 35

4.2 Stryring av motor med P-del 37

4.3 Stryring av motor med I-del 40

4.4 Stryring av motor med D-del 42

4.5 Stryring av motor med PI-del 44

4.6 Stryring av motor med PID-del 46

4.7 Funksjon av PID-regulatoren 49

5 Manuell kjøring av Lego-robot 51

5.1 Introduksjon 51

5.2 Kode for styring av robot 53

5.3 Kvalitetsmål med kode 54

5.4 Kvalitetsmål under kjøring 55

5.5 Kvalitetsmål etter kjøring 57

5.6 Resultat 58

6 Automatisk kjøring med PID (utført av Abdirahman og Wijenthan) 62

6.1 Introduksjon 62

6.2 Beregninger, kode og flytskjema 63

6.3 Resultat 65

7 Støvsuger-robot 70

7.1 Introduksjon 70

7.1.1 fremgangsmåte 71

7.2 Beregninger og kode 72

7.2.1 Kode 72

7.2.2 Beregninger 74

7.3 Resultat 76

8 Konklusjon 77

Bibliografi 78

Vedlegg 78

A Timelister 79

B Programlisting prosjekt 06 **80**

B.1 Kode for prosjekt 01 - numerisk integrasjon av en konstant 80

B.2 Kode for prosjekt 01 - numerisk integrasjon et chirp-signal 83

B.3 Kode for vasket chirp-signal 86

B.4 Kode for prosjekt 02 - Lavpassfiltrering av temperatur . . 87

B.5 Kode for prosjekt 02 - høypassfiltrering av et sinussignal 89

B.6 Kode for prosjekt 03 - numerisk derivasjon av stigningstall 93

B.7 Kode for prosjekt 03 - numerisk derivasjon av et sinussignal 96

B.8 Kode for prosjekt 03 - numerisk derivasjon av et et chirp-signal 99

B.9 Kode for prosjekt 04 - P-delen 101

B.10 Kode for prosjekt 04 - I-delen 105

B.11 Kode for prosjekt 04 - D-delen 110

B.12 Kode for prosjekt 04 - PI-delen 114

B.13 Kode for prosjekt 04 - PID-delen 119

B.14 Kode for prosjekt 06 - automatisk kjøring 123

B.15 Kode for prosjekt 07 - støvsuger 129

KI-verktøyet ChatGPT [2] er benyttet til språklig kvalitetssikring og rettskriving av teksten i de fleste kapitlene i rapporten. Innholdet generert med ChatGPT er grundig gjennomgått og kontrollert for å sikre at det oppfyller våre krav til kvalitet og nøyaktighet.

Sammendrag

Denne rapporten beskriver et prosjekt hvor vi har kombinert lego mindstorms og Matlab for å lære anvendt matematikk og fysikk i robotprogrammering. Prosjektet ble utført av gruppe 61, bestående av Abdirahman Abdi Ibrahim og Wijenthan Wijayaseelan. Hensikten med prosjektet er å implementere og teste forskjellige metoder og anvende disse i fysiske eksperimenter. Rapporten består av åtte kapitler, numerisk integrasjon, filtrering, numerisk derivasjon, PID-regulator, manuell og automatisk kjøring av robot, støvsuger-robot, konklusjon og vedlegg med timelister og koder.

Vi har implementert traaesmetoden for å tilnærme bestemte integraler av diskrete målinger. Først demonstrerte vi konstant strømingsrate ved å måle, fylle, og drikke prosess i et vannglass med lyssensoren og et gråskalaark. For å redusere sensorstøy implementerte vi føruteordents IIR-filtere. Lavpassfilteret dempet høyfrekvente komponenter effektivt, mens høypassfilteret isolerte raske endringer. Numerisk bakoverderivasjon av ultralydsensoren ga hastighetsestimat. Videre utviklet vi en PID-regulator som kombinerer proporsjonale, integrerende og deriverende ledd. Vi testet regulatoren både i simulering og på EV3-roboten. I praktiske robotkjøringstester sammenlignet vi manuell og PID-styrt autokjøring. Til slutt konstruerte vi en automatisk støvsuger robot som rygger og roterer ved hinder.

Gjennom dette prosjektet har vi vist hvordan teori kan implementers I Matlab og anvendes praktisk ved hjelp av lego roboter. Eksperimentene ga svært gode resultater, og PID-styringen har forbereder robotens presisjon og fart

Kapittel 1

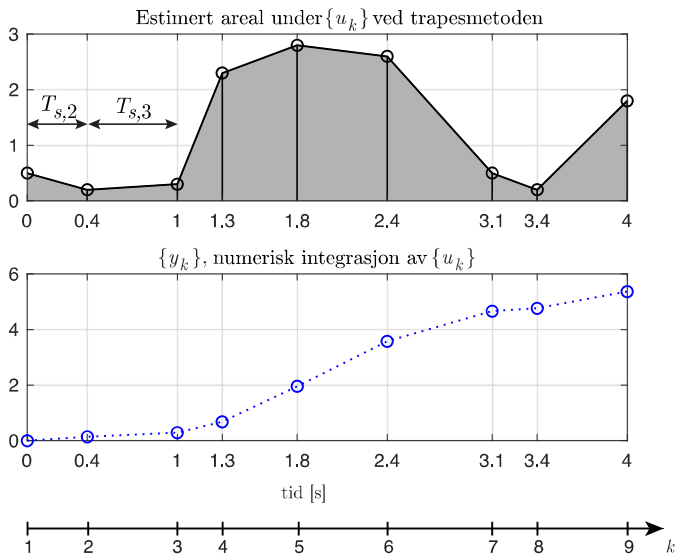
Numerisk integrasjon

1.1 Introduksjon

Numerisk integrasjon er en metode for å tilnærme arealet under en kurve ved å dele området i små rektangler eller trapeser. Fra Matematikken kan man finne et bestemt integral mellom to tidspunkter som vi ser fra uttrykket i formel 1.1

$$A = \int_a^b u(t)\ dt \tag{1.1}$$

I praktiske anvendelser mottas diskrete målinger fra en sensor som registrerer ulike verdier med varierende eller faste tidsintervaller. For å beregne arealet fortløpende uten et kjent funksjonsuttrykk, trenger vi et rekrusivt utrykk. Figur 1.1 viser et tidsseresignal som er hentet fra en praktisk applikasjon. Ved hvert tidssteg mottas en ny målt verdi på en størrelse u_k, hvor k er indeksen. Den nederste grafen y_k viser det samlede beregnede integralet under kurven, representert med de grå boksene mellom hvert tidssteg.



Figur 1.1: Prinsippet for numerisk integrasjon av tidseriedata. For hver ny måling u_k legges det inn et trapes med bredde $T_{s,k}$, og totalarealet av trapesene tilsvarer integralverdien. Figuren er hentet fra [1].

Lengden på tidssteget angir hvor ofte målingene tas. Kortere tidssteg innebærer tettere målinger, noe som gir en mer detaljert og nøyaktig estimering av arealet under kurven.

1.1.1 Trapesmetoden med variabelt tidssteg

For å estimere arealet under en kurve som representerer en tidsseriedata, benytter vi en tilnærming basert på Riemann-sum (ikke-rekursiv) som vist i ligning 1.2

$$A = \Delta t \sum_{k=1}^{n} \frac{1}{2}(u(t_{k-1}) + u((t_k)) \tag{1.2}$$

Her representerer $u(t_{k-1})$ og $u(t_k)$ henholdsvis funksjonsverdiene ved tidspunktene t_{k-1} og t_k. Ligningen under viser hvordan vi beregner tidskrittet Δt, hvor a og b er tidsintervallet og n er antall trapeser.

$$\Delta t = \frac{b-a}{n} \tag{1.3}$$

Riemann-sum egner seg ikke bra for praktiske anvendelser, derfor må vi skrive den om til en rekursiv form. En rekursiv formulering av riemann-sum lar oss oppdatere og samle data om arealet fortløpende for hver nye måling. Endringene innebærer at vi kan skrive om ligning 1.2 som:

$$A = T_s \sum_{k=1}^{n} \frac{1}{2}(u_{k-1} + u_k) \tag{1.4}$$

hvor T_s er tidskrittet og u_k er funksjonsverdien ved indeks k og u_{k-1} lar oss hente forrige funksjonsverdi. Ved å legge til et ledd y_{k-1} som lar oss hente verdien for forrige beregnet areal får vi trapesmetoden som vist i ligning 1.5

$$y_k = y_{k-1} + T_s \cdot (1/2 \cdot (u_{k-1} + u_k)) \tag{1.5}$$

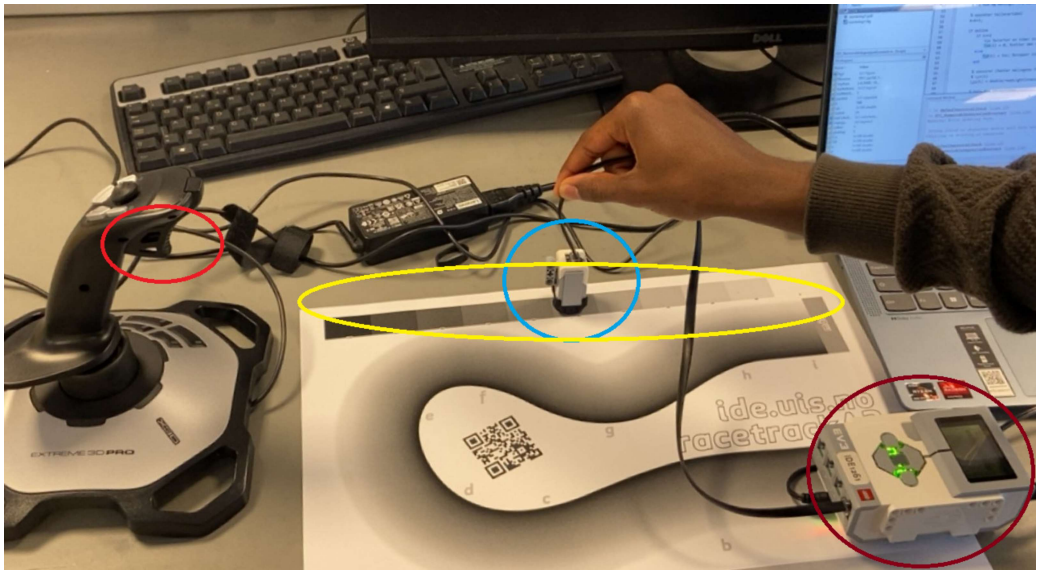
I kodeutdraget under implementerer vi trapesmetoden i kode. Siden tidskrittet mellom hver måling varierer med tiden på grunn av manglende synkronisering mellom datamaskinen og utsyret som er brukt, har vi tatt hensyn til dette i øverste linje av koden.

Kode 1.1: Kode hentet fra `P01_NumeriskIntegrasjonKonstant.m`

```
T_s(k) = Tid(k) - Tid(k - 1);
y(k) = y(k - 1) + T_s(k) * 1/2 * (u(k - 1) + u(k));
```

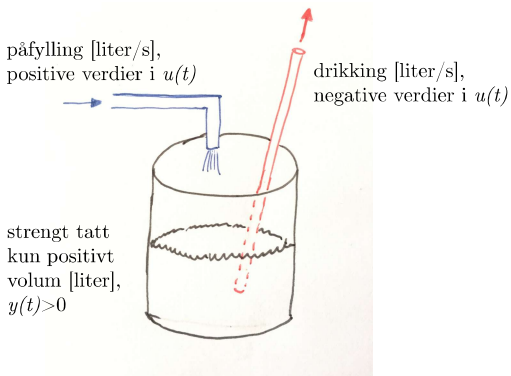
1.2 Relasjon til det bestemte integral som en konstant

I dette prosjektet har vi demonstrert bruken av trapesmetoden praktisk ved hjelp av en EV3 vist i figur 1.2 med brun ring. EV3-en er tilkoblet en lyssensor markert med blå ring, og dataene henter vi fra Matlab gjennom en tilkobling mellom EV3-bricken og PC-en. Lysmålingene fra roboten blir reflektert fra et gråskalaark markert med gul i figur 1.2. Målingene blir lagret som vektoren Lys(k), og tidspunktet for hver måling lagres som tidsvektoren Tid(k).



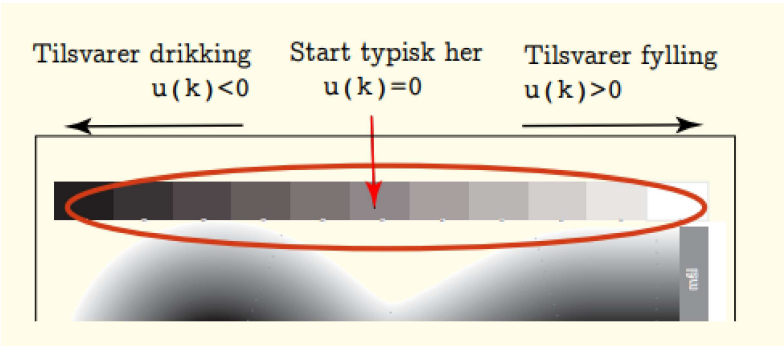
Figur 1.2: Utstyret som er brukt under eksperimentet: til venstre i bildet er det en Joystick, der skyteknappen er markert med rødt. Nederst i bildet er en EV3, markert med brun ring. Markert med gul er det de gråskalaverdiene. Lyssensoren er markert med blå.

I gjennomføringen av dette eksperimentet har vi gjenskapt eksempeloppgaven, ved å fylle opp et vannglass og drikke fra det med et sugerør, som vist i figur 1.3.



Figur 1.3: Glass som fylles med vann innebærer positive verdier på $u(t)$, mens drikking via sugerør innebærer negative verdier. Vannvolumet i glasset tilsvarer $y(t)$.

For å implementere dette praktisk har vi simulert fylling og tømming av vannglasset ved å plassere lyssensoren tett inntil og omtrent midt på gråskalaarket vist i figur 1.4. Etter noen sekunders ventetid flyttes lyssensoren raskt to gråskalaverdier til høyre. Dette representerer fylling av glasset. Deretter returneres sensoren til startposisjonen etter en liten pause. For å simulere tømming flyttes lyssensoren raskt to gråskalaverdier til venstre, før den igjen returneres til startpunktet.



Figur 1.4: Gråskalaark brukt for å simulere fylling og tømming av vannglasset. Figuren er hentet fra [1]

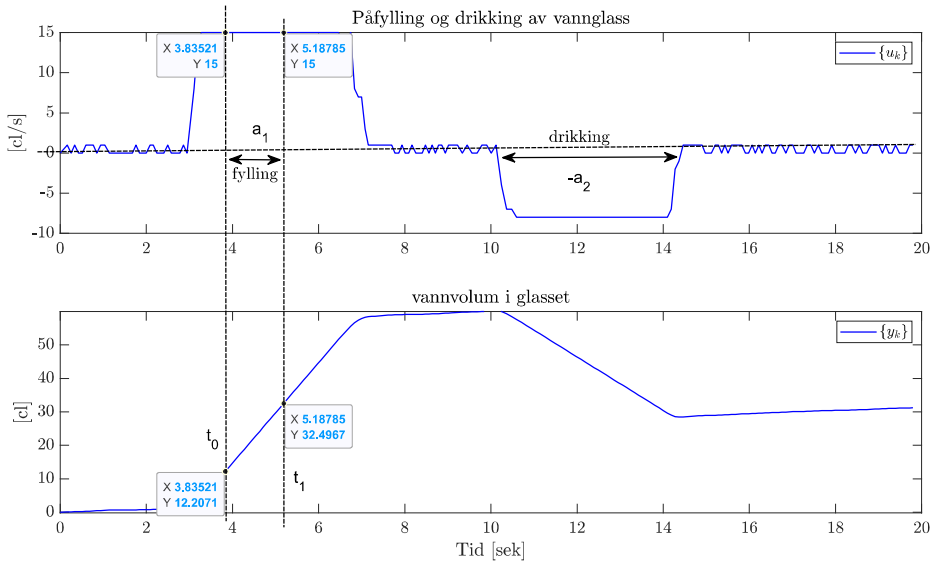
I kodeutdraget under implementerer vi eksempelprossessen i kode. Først henter vi målingene ved å sette de reflekterte verdiene fra lyssensoren til `Lys(k)`. Deretter setter vi den første målingen til å være `LysInit`, og trekker den fra lys-målingene som vi har gitt en ny variabel u(k). Dette for å hente riktig startvedi for målingene. I siste linje av koden har vi satt initialverdien til den integrerte y(k) til 0.

1.2 Relasjon til det bestemte integral som en konstant

Kode 1.2: Kode hentet fra `P01_NumeriskIntegrasjonKonstant.m`

```
Lys(k) = double(readLightIntensity(myColorSensor, 'reflected'));
LysInit = Lys(1); % den første målingen
u(k) = Lys(k) - LysInit;
y(1) = 0;
```

Resultatet fra eksperimentet vises i figur 1.5



Figur 1.5: Øverste delfigur viser strømnings-målingene fra lyssennsoren. Påfyllingsrate a_1=15 [cl/s] mellom tidsintervallet t_0 = 3.8s og t_1
=5.2s. I den nederste delfigur har vi integralet av strømningsmålingene som represennterer volumet. Dette er arealbetraktningen mellom de to stiplet linjene.

For å verifisere arealet av det området som repsenterer påfyllingsraten a_1 og tidsintervallet t_0 og t_1, bruker vi integrasjon slik det fremgår av ligning 1.6.

$$\begin{aligned} A &= \int_{t_0}^{t_1} u(t)dt = \int_{3.8}^{5.2} 15\,dt \\ &= 15 \cdot (5.2 - 3.84) = 15 \cdot 1.4 = \underline{\underline{20.4}} \end{aligned} \tag{1.6}$$

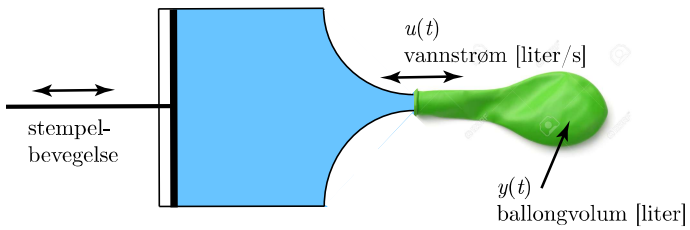
Til sammenligning beregner vi endringen i y(t) ved de gitte tidspunktene i figur 1.5 i ligningen under.

$$A = y(1) - y(0) = (32.5 - 12.2) = \underline{\underline{20.3}}\,\text{cl}$$

Vi ser at de matematiske utregningene av arealet er tilnærmet lik informasjonen vi kan observere fra figuren. Dette bekrefter at vår metode for numerisk integrasjon er pålitelig.

1.3 Relasjon til det bestemte integral som en sinusfunksjon

Målet med dette forsøket er å analysere et sinussignal ved hjelp av en ventilfri pumpe som pyller og tømmer en ballong, som illustrert i figur 1.6 Stemplet beveger seg slik at vannstrømmen følger formen $u(t) = U \cdot sin(\omega t)$

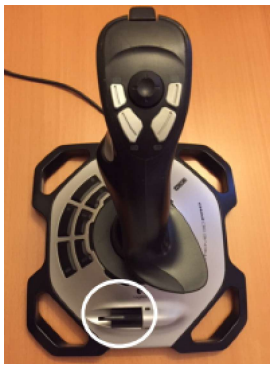


Figur 1.6: Spesialpumpe for å pumpe vann inn og ut av en ballong.

For å demonstrere dette i en praktisk anvendelse som representerer vannstrømmen $u(t)$, har vi laget en konstruksjon som ligner på den i figur 1.7. Ved å la potensiometeret markert med hvit ring i figur 1.8 være pådraget til motoren har vi god kontroll på motorpådraget og dermed også vinkelhastigheten (ω). EV3-en i konstruksjonen hjelper oss sørge for at det påmonterte gummihjulet roterer det gråskalaarket.

Figur 1.7: Konstruksjon som ved hjelp av friksjonen fra gummihjulet roterer et oppklipt A4 ark med gråskalaverdier. Bilde hentet fra [1]

.



Figur 1.8: Bilde av potensiometeret som vi bruker for å styre motorpådraget. Bilde hentet fra [1]

.

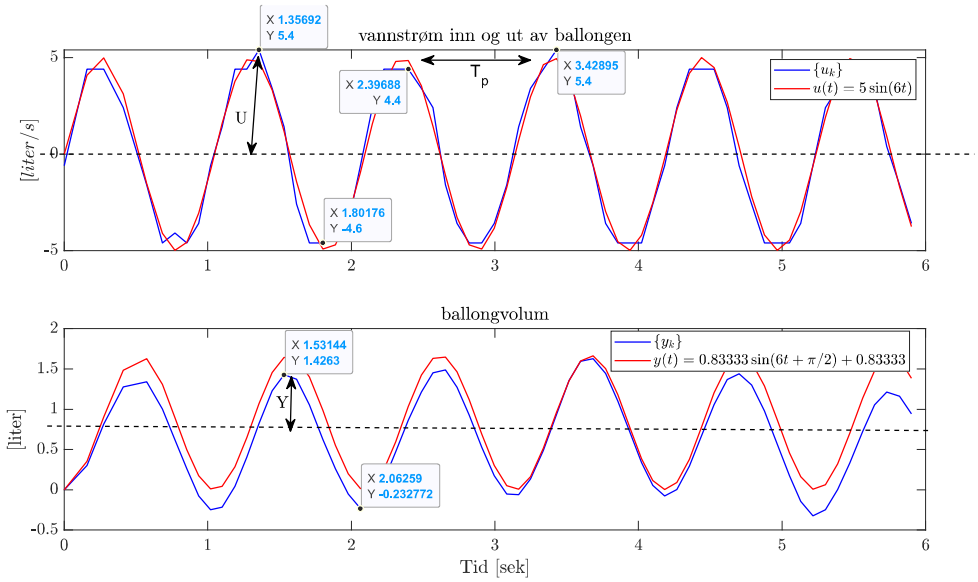
kodesnutten under i kodeutdrag 1.3 sørger vi for at friksjonen fra gummihjulet roterer arket. Den første linjen av koden benyttes for å styre motorpådraget med potensiometeret. Deretter setter vi pådragsdata mot EV3.

Kode 1.3: Kode hentet fra P01_NumeriskIntegrasjonSinus.m

```
% beregning av pådrag fra styrestikken
u_A(k) = JoyPot(k); %JoyPot(k) = JoyAxes(4) --> potensiometeret

if online
    motorA.Speed = u_A(k);
```

```
    start(motorA)
end
```

Dette gir oss resultatet i figuren under, hvor målingene og den integrerte er i blått, samt estimatet i rødt.



Figur 1.9: Målt vannstrømning $u(t)$ i liter/s, Ballongvolum $y(t)$ i liter. Faseforsyvningen mellom $u(t)$ og $y(t)$ er ca. 90°.

For å verifisere at den numerisk integrerte kurven i figur 1.9 stemmer overens med den teoretiske modellen, estimerer vi først amplituden, U og vinkelfrekvensen, ω.

Amplitudenen til en periodisk graf kan beregnes som vist i ligning 1.7

$$U = \frac{y_{maks} - y_{min}}{2} = \frac{5.4 - (-4.6)}{2} = 5 \tag{1.7}$$

Vinkelfrekvensen beregnes ved først å finne perioden, og deretter bruke denne til å beregne vinkelfrekvensen i henhold til ligningen nedenfor.

$$
\begin{aligned}
T_p = t_2 - t_1 = 3.43 - 2.4 = 1.03 \\
\omega = \frac{2\pi}{1.03} = 6.1[rad/s]
\end{aligned}
\tag{1.8}
$$

Dette betyr at sinussignalet fra målingene våre er gitt som

$$u(t) = 5 \cdot sin(6.1t)$$

Generelt kan vi finne det bestemte integralet av et sinussignal slik det fremgår i ligningen under.

$$
\begin{aligned}
y(t) &= \int_0^t U \cdot \sin(\omega\tau)\, d\tau + y(0) \\
&= -Y\cos(\omega t) + y_A, \quad \text{der } y_A = \frac{U}{\omega}, \quad \text{og } Y = \frac{U}{\omega}
\end{aligned}
\tag{1.9}
$$

Siden en cosnusfunksjon tilsvarer en faseforskjøvet sinusfunksjon kan vi skrive om integralet som

$$y(t) = -Y sin(\omega + \phi) + y_A \tag{1.10}$$

Med utgangspunkt i det generelle uttrykket kan vi finne integralet til signussignalet vårt som det følger av ligningen under.

$$
\begin{aligned}
y(t) = \int_0^t u(\tau)d\tau &= \int_0^t 5 \cdot sin(6.1\tau)d\tau \\
&= \frac{-5}{6.1} \cdot cos(6.5 \cdot t) + \frac{5}{6.1} \\
&= -0.82 \cdot cos(6.1 \cdot t) + 0.82 \\
&\approx \underline{\underline{-0.82 \cdot sin(6.1 + \pi/2) + 0.82}}
\end{aligned}
\tag{1.11}
$$

Til sammenligning sjekker vi om amplituden til den integrerte grafen i nederste delfigur stemmer overens med det vi har regnet ut matematisk. Dette gjør vi på samme måte som vi gjorde over, og ved å bruke punktene i nederste delfiguren.

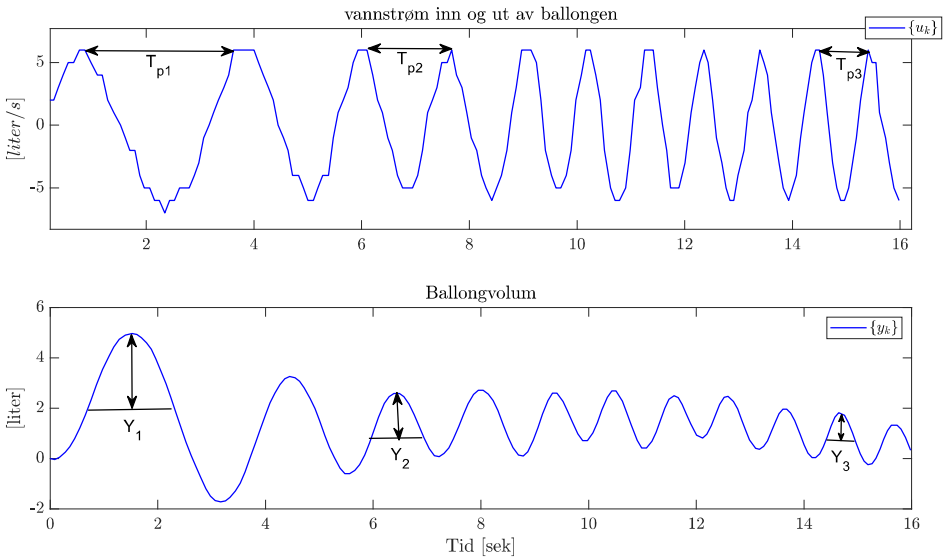
$$Y = \frac{1.4263 - (-0.23277)}{2} = \underline{\underline{0.8295}} \tag{1.12}$$

Resultatet fra utregningen viser at vi har estimert riktig og at vår metode for numerisk integrasjon er pålitelig. Likevel er det bemerkelig at det er litt avvik mellom den estimerte figuren og den uestimerte. Dette kommer også frem i de teoretiske utregningene.

1.4 Numerisk integrasjon av *chirp*-signal

1.4.1 betydningen av høyere vinkelhastighet

Når frekvensen til signalet $u(t)$ øker, reduseres amplituden til det integrerte signalet $y(t)$. Dette fremgår av det generelle uttrykket i ligning 1.7. For å illustrere dette har vi gjentatt det samme eksperimentet som for sinusoppgaven, men denne gangen roterte vi lyssensoren manuelt over arket for å oppnå et bedre resultat med en økning i frekvens over tid. Resultatet av demonstrasjonen er vist i figur 1.10.



Figur 1.10: Øverst delfigur: målt vannstrømning $u(t)$ i liter/s. Nederst delfigur: ballongvolum $y(t)$ i liter.

Ved å se på figuren kan det observeres at perioden i den øverste delfiguren avtar fra venstre, altså $T_{p1} > T_{p2} > T_{p3}$. Som følge av dette øker vinkelfrekvensen, i henhold til relasjonen gitt i ligning 1.8. Den økte vinkelfrekvensen påvirker amplituden Y til det integrerte signalet, slik at den reduseres fra venstre, og mot høyre, $Y_1 > Y_2 > Y_3$. Dette samsvarer med den generelle ligningen, og nederste delfiguren.

Kapittel 2

Filtrering (utført av hele gruppen)

2.1 Introduksjon

Filtrering blir brukt til en rekke ting. I praktiske anvendelser bruker vi filtrering til å filtrere måleresultatet før resultatet. Hensikten med dette er å fremheve viktig informasjon, samtidig som vi fjerner målestøy og andre faktorer som kan vanskeliggjøre avlesningene. I værstasjoner brukes det sensorer som måler for eksempel, temperatur, fuktighet eller trykk. Disse sensorene kan ta inn støy fra omgivelsene sine. Her brukes filtrering for å fjerne støyet.

Figur 2.1: Eksempler på praktiske annvelderser hvor resultatet filtreres før det presenteres.

Vi deler filtrering i to grupper, ikke-rekrusivt FIR-filter (finite impulse response) og rekursivt IIR-filter (infinite impulse response). I denne oppgaven har vi brukt et første ordens lavpassfilter og høypassfilter, som begge er et IIR-filter. Vi implementerer lavpassfilteret med formelen:

$$y_k = (1 - \alpha) \cdot y_{k-1} + \alpha \cdot u_k \tag{2.1}$$

Der u_k er målingene, y_{k-1} den forrige filtrerte verdien, og α parameteren bestemmer graden av filtrering. Dette betyr at en lav verdi for α tilsvarer en kraftig filtrering.

Et lavpassfilter bruker de siste ufiltrerte målingene og den forrige filtrerte målingen. Filteret fjerner de høyfrekvente signalene, og slipper de lavfrekvente signalene. Dette betyr at filteret reduserer støy, og viser frem signalet tydligere.

Et høypassfilter fjerner de lavfrekvente i signalene, og slipper gjennom de høyfrekvente signalene. Det fungerer dermed som det motsatt av et lavpassfilter. Et

høypassfilter fremhever støy i et signal heller enn å fjerne det. Den rekrusive formelen for et høypassfilter er:

$$h(k) = \alpha \cdot h(k-1) + \alpha(u(k) - u(k-1)) \tag{2.2}$$

Her representerer differansen $u(k) - u(k-1)$ forskjellen mellom to påfølgende målinger. Differansen er et mål på hvor fort signalet endrer seg. Når differansen er stor, betyr det at signalet endrer seg raskt. Filtret bruker sin forrige verdi $h(k-1)$ sammen med endringen $\alpha \cdot (u(k) - u(k-1))$ for å beregne den nye verdien $h(k)$

kodesnuten under viser hvordan vi implementerer et lavpass og høypassfilter i matlab for en bestemt verdi for τ:

Kode 2.1: Kode hentet fra `P02_FiltreringStoy.m`

```
alfa = 1 - exp(-T_s/tau);
% Implementasjon av første ordens lavpassfilter og høypassfilter
y(k) = alfa * u(k) + (1 - alfa) * y(k - 1);
h(k) = alfa * h(k - 1) + alfa * (u(k) - u(k - 1));
```

2.2 Første ordens lavpassfilter i termometer

I dette forsøket later vi som om vi måler temperaturen i en varm kaffekopp ved å ha temperaturmåleren opp i koppen. Til dette har vi brukt lyssensoren og EV3-en i figur 1.2, samt et hvitt ark og et svart bord. For å gjenskape eksempelprosessen plasserte vi lyssensoren først på det svarte bordet, før vi flyttet den lynraskt over til det hvite arket. Data fra Lyssensoren og simueleringstiden henter vi via Matlab. Den svarte fargen reflekterer lite lys, og tilsvarer dermed en lav temperatur, mens den hvite fargen reflekterer en mye lys, og tilsvarer dermed en høy temperatur.

Vi har gjennomført eksperimentet to ganger. En gang med sanntidsplotting og én gang uten. Sanntidsplotting innebærer at dataene fra lyssensoren plottes fortløpende mens programmet kjører. Uten sanntidsplotting samles dataene inn uten visning underveis, og grafen vises først etter at programmet er avsluttet. Dette styrte vi ved å sette variabelen plotting henholdsvis til true for sanntidsplotting og

false for ettervisning. Kodesnutten nedenfor viser hvordan dette er implementert i koden.

Kode 2.2: Kode hentet fra `P02_FiltreringStoy.m`

```

plotting = false;  % Skal det plottes mens forsøket kjøres
plotting = true;  % Skal det plottes etter at forsøket er ferdig
```

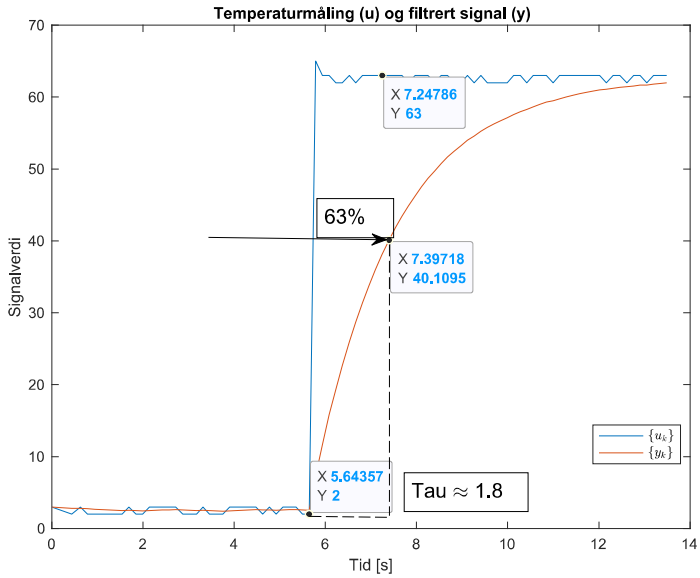
Årsaken til at vi kjører koden for eksperimentet to ganger, ligger i hvordan sanntidsplotting påvirker tidsbruken per måling. Når plotting er aktivert, brukes mer tid på grafisk visning, mens når plotting er deaktivert, kan flere målinger samles inn på kortere tid, noe som resulterer i en høyere samplingsfrekvens. I denne oppgaven har vi sett hvordan samplingstiden T_s påvirker α verdien. Koden under viser hvordan vi har satt opp parametene for første ordens lavpassfilter. Vi har benyttet samme verdier for samplingsfrekvens f_s og tidskonstant τ både i sanntidsplotting og uten sanntidsplotting.

Kode 2.3: Kode hentet fra P02_FiltreringTemperatur.m

```
fc = 0.5;
tau = 1.8;
T_s(k) = Tid(k) - Tid(k-1);
alfa(k) = 1 - exp(-T_s(k)/tau);
```

I koden har vi benyttet ligningen for et lavpassfilter til å lavpassfiltere lysmålingene. Som nevnt har vi kjørt koden to ganger. Resultat som er presentert i figur 2.2 er for et lavpassfilter som er plottet i sanntid. Resultatet i figurene viser en rød graf og blå graf, der den røde grafen representerer det filtrerte signalet y_k, og den blå viser det ufiltrerte signalet u_k.

Som figuren viser, inneholder u_k raske svingninger og tydelig støy, mens y_k er jevnere. Dette bekrefter at lavpassfilteret fungerer som forventet ved å dempe høyfrekvent støy og fremheve lavfrekvente endringer.



Figur 2.2: Grafene i figuren er plottet i sanntid (`plotting = true`). De viser temperaturmålingen u_k blått sammen med det lavpassfiltrerte signalet y_k i rødt.

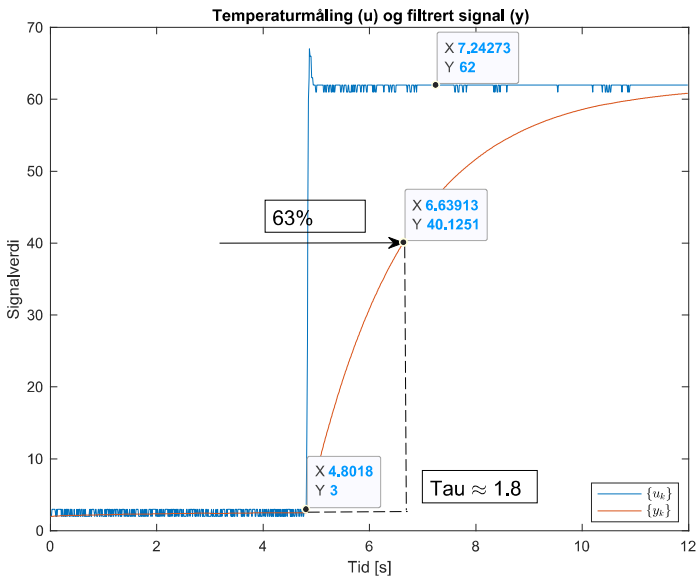
Tidskonstanten τ er definert som tiden det tar fra et sprang i temperaturen inntreffer, til den filtrerte temperaturen har nådd 63% av økningen. I ligningen nedenfor beregnes signalverdien ved dette punktet, basert på avlesningene fra figur 2.2:

$$\begin{aligned} y_{63\%} &= y_0 + 0.63 \cdot \Delta Y \\ &= 2 + 0.63 \cdot (63 - 2) \\ &= \underline{\underline{40.43}} \end{aligned} \tag{2.3}$$

Tidskonstanten τ beregnes på samme måte, men her bruker vi avlesninger fra x-aksen for å finne tiden fra spranget i temperaturen inntreffer, til signalet har nådd 63% av økningen i grafen.

$$\begin{aligned}\tau &= t_{63\%} - t_{sprang} \\ &= 7.39718 - 5.64357 \\ &\approx \underline{\underline{1.75}}\end{aligned} \tag{2.4}$$

Nedenfor vises et lavpassfiltrert signal, plottet uten sanntidsvisning. Vi gjennomfører de samme beregningene som tidligere for signalet med `plotting = true`, og sammenligner deretter resultatene.



Figur 2.3: Grafene i figuren er plottet uten sanntidsvisning (`plotting = false`). De viser temperaturmålingen u_k blått sammen med det lavpassfiltrerte signalet y_k i rødte

På samme måte som for sanntidsplotting av grafene beregner vi $y_{63\%}$ i ligningen nedenfor.

$$\begin{aligned}y_{63\%} &= y_{fr} + 0.63 \cdot \Delta Y \\ &= 3 + 0.63 \cdot 59 \\ &= \underline{\underline{40.17}}\end{aligned} \tag{2.5}$$

Deretter finner vi tidskontanten τ basert på avlesningene i figur 2.3

$$\begin{aligned}\tau &= t_{63\%} - t_{sprang} \\ &= 6.63913 - 4.8018 \\ &\approx \underline{\underline{1.84}}\end{aligned} \tag{2.6}$$

Til sammenligning ser vi at beregningene av tidskonstanten τ, basert på avlesningene fra grafen, er nesten identiske. Forskjellen mellom de to verdiene skyldes hovedsakelig små avvik, som for eksempel at vi ikke nødvendigvis har valgt de eksakte punktene på grafene. Hvordan påvirker Samplingstiden T_s og tidskonstanten τ, alfa-verdien?

Med en kjent tidskonstant τ kan vi beregne filterkoeffisienten α ved hjelp av følgende formel:

$$\alpha = 1 - e^{-T_s/\tau}, \tag{2.7}$$

Gitt at tidskonstanten τ er 1.8, kan α beregnes for ønsket samplingstid T_s.

For plotting i sanntid (`plotting = false`) har vi at samplingstiden $T_s = 0.0100$. Dette gir følgende verdi for filterkoeffisienten α:

$$\begin{aligned}\alpha &= 1 - e^{-0.0100/1.8} \\ &= 1 - e^{-0.00556} \\ &\approx 1 - 0.99446 \\ &\approx \underline{\underline{0.00554}}\end{aligned} \tag{2.8}$$

For plotting i sanntid (`plotting = true`) har vi at samplingstiden $T_s = 0.1488$. Dette gir følgende verdi for filterkoeffisienten α:

$$\begin{aligned}\alpha &= 1 - e^{-0.1488/1.8} \\ &= 1 - e^{-0.08267} \\ &\approx 1 - 0.09206 \\ &\approx \underline{\underline{0.0794}}\end{aligned} \tag{2.9}$$

For å undersøke hvordan α-verdien påvirkes av tidskonstanten τ og samplingstiden T_s, sammenligner vi de beregnede α-verdiene. Som vi vet, vil sanntidsplotting kreve mer tid til grafisk visning. Dette fører til en økning i samplingstiden, som igjen gir en litt høyere verdi for α sammenlignet med situasjonen uten sanntidsplotting.

Denne forskjellen skyldes hovedsakelig av eksponenten i uttrykket $e^{-T_s/\tau}$. Siden τ er konstant, styrer T_s direkte strørrelsen for denne brøken. Når T_s øker, blir forholdet T_s/τ større. Siden vi har et minus tegn foran, blir uttryket negeativt. Dermed minker selve eksponenten, og $e^{-T_s/\tau}$ blir mindre.

Når samplingstiden reduseres, blir forholdet T_s/τ mindre. Siden $-T_s/\tau$ er negatvit, men forholdet T_s/τ er allerede lavt, vil vi få et lite negativt tall i Eksponentialeddet

En større α verdi betyr at filteret gir mer vekt til den nye målingen $u(k)$ noe som gir en mer aggresiv respons. Filteret reagerer raskerer til endringer i signalet. Når vi har en lavere α verdi, beholder filteret mer av den forrige verdien $y(k-1)$ og endrer seg saktere. Dette resulterer til at filteret reagerer saktere på nye målinger

Dette illustrerer hvordan filteret tilpasser seg mot økt målehyppgighet. Selv med flere målinger, vil filtrerts struktur tilpasse α slik at vi får en jevn og stabil respons.

2.3 Lavpass- og høypassfiltrering av støyfylt signal

I denne oppgaven har vi lavpass og høypassfiltrert et støyfylt signal u_k. Vi har også spesifisert en vilkårlig knekkfrekvens f_c for u_k

For å skape et støyfylt signal har har vi simulert et sinus signal, ved hjelp av lyssensoren, sinus funksjonen, og randn funksjonen i matlab. Lyssensoren holdes

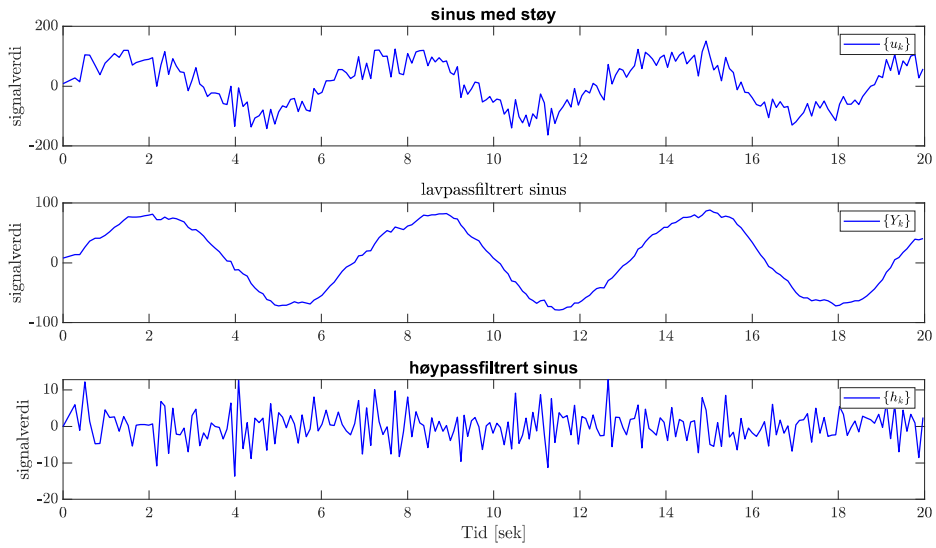
stille over bordet. På denne måten vil Lys(k) være tilnærmet konstant. Variasjonene i u_k kommer fra de to tillegsleddene. Sinusleddet legger til en periodisk sinusformet svigning med amplitude 100. Radn funksjonen legger til tilfeldig støy. Til sammen blir signalet definert som $u(k) = Lys(k)+100sin(Tid(k))+30randn()$ Her er lys(k) den stabile faktiske avlesningen fra lyssensoren, mens de to andre leddene legger til periodisk svingninger og støy.

Kodesnutten under viser hvordan vi har simulert et støyfylt signal.

Kode 2.4: Kode hentet fra P02_FiltreringTemperatur.m

```
u(k) = Lys(k) + 100*sin(Tid(k)) + 30*randn();
```

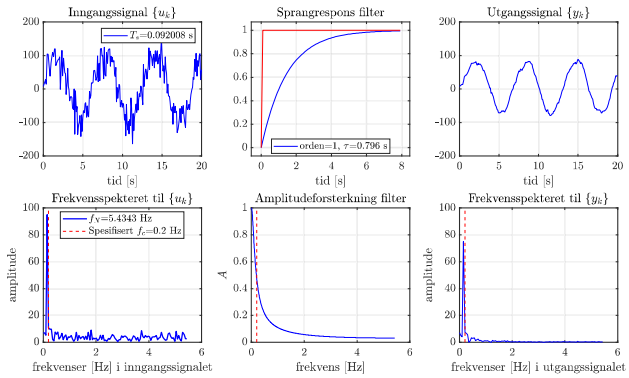
Og den gir oss resultatet nedenfor.



Figur 2.4: Bilde av sinusfunksjon med støy, som blir lavpass og høypass filtrert

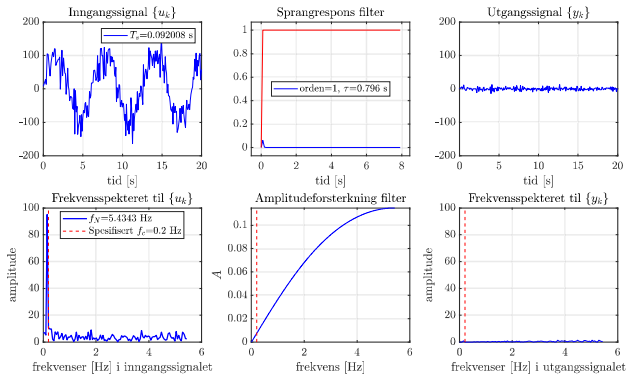
Første plottet viser u_k. Vi ser at grafen er en sinus funksjon, med amplitude 100 og med støy. Plot to viser u_k med lavpassfilteret. Signalet blir beregnet rekurisvt med likning2.1 Resultetet gir en glatt kurve der støyen blir redusert. Det siste plot viser samme signal, u_k men som beregnes rekurisvt med likning 2.2 Dette demper de lave variasjonene og fremhever de raske endringene, som vi kan se på

det siste plottet. Vi kan bruke PlotAltOmFilter funksjonen for å bekrefte resultatene våre. PlotAltOmFilter figurene består av seks underplott, inngangssignal u_k, sprangresponset, utgangssignalet y_k, frekvensspekteret, filtrert frkevensrespons, og frekvensspekteret til utgangssignalet.



Figur 2.5: Resultat av PlotAltOmFilter funksjonen for lavpass filteret

Utganssignalet fremstår som en jevn støydempet versjon av inngangssignalet. Rask varierende komponenter er fjernet, mens langsomme endringer følger sinusformen. Inngangsspekteret viser signalet over hele båndet, mens utgansspektret er dempet over knekkfrekvensen f_c Knekkfrekvensen ble valgt ved å kjøre koden én gang og deretter identifisere i frekvensspekteret det punktet hvor knekkene stabiliserer seg. Dette demonstrerer at filteret slipper lavfrekvente komponenter gjennom og demper høye.



Figur 2.6: Resultat av PlotAltOmFilter funksjonen for høypass filteret

Utgangssignalet består av det av kunn de rask endrende komponentene. Saktegående variasjoner er redusert og viser at de lave frekvensene er fjernet. Ingangsspekteret er bredt, mens utgangsspekteret demper alle frekvenskomponentene med $f < f_c$ slik at amplituden ligger nær $0Hz$ ligger på eller nærme null. Med andre ord, langsomme variasjoner er borte.

Kapittel 3

Numerisk derivasjon (utført av hele gruppen)

3.1 Introduksjon

Derivasjon i matematikk handler om endring. Den deriverte beskriver stigningstallet og hvordan en funksjon endrer seg. Mer presist handler det om momentan endring, altså hvordan funksjonsverdien endrer seg i et gitt punkt. Derivasjon har mange praktiske anvendelser i hverdagen. Et kjent eksempel er hvordan politiet måler farten til en bil. Ved å beregne endringen i posisjon over tid, finner de kjøretøyets hastighet, som er den deriverte av posisjonen. Innenfor modeller som beskriver populasjonsvekst eller spredning av sykdom, brukes derivasjon til å analysere hvordan disse fenomenene utvikler seg over tid. Den deriverte til en funksjon $f(t)$ i et punkt t defineres som:

$$f'(t) = \frac{df}{dt} = \lim_{h \to 0} \frac{f(t+h) - f(t)}{h} \tag{3.1}$$

Denne definisjonen forteller om endringsraten til $f(t)$ når vi beveger oss med veldig små steg h fra t Dersom grensen eksisterer gir den et eksatt resultat.

I noen tilfeller er det vanskelig, eller til og med umulig, å derivere en funksjon

analytisk. Da kan vi bruke numerisk derivasjon for å tilnærme den deriverte. Numerisk derivasjon estimerer endringsraten ved hjelp av diskrete punkter, og brukes ofte i situasjoner der vi kun har tilgang til måledata eller der funksjonen ikke lar seg derivere med vanlige metoder. En vanlig metode for numerisk derivasjon er framoverdifferansen, som beregnes slik:

$$f'(t) = \frac{df}{dt} \approx \frac{f(t+h) - f(t)}{h} \tag{3.2}$$

Der $f(t)$ er funksjonen vi vil derivere og h er avstanden mellom to punkter. Denne metoden kalles ofte for framoverdifferansen fordi vi tar utganspunkt i punktet t og neste punkt $t + h$. Et annet alternativ og like god likning er:

$$f'(t) = \frac{df}{dt} \approx \frac{f(t) - f(t-h)}{h} \tag{3.3}$$

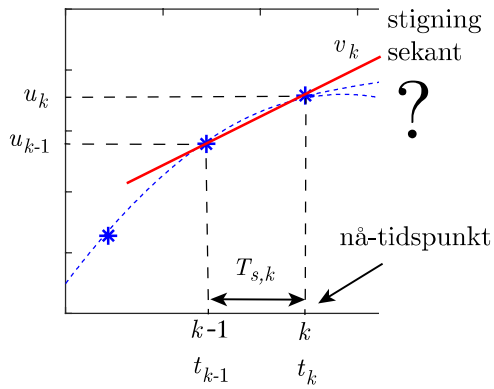
Denne metoden heter bakoverdifferansen. Her tar vi utgangspunkt i punktet t og det forrige punktet $t - h$. Når man arbeider med måledata eller tidsserier, er det ofte ikke mulig å hente ut verdien $f(t + h)$, ettersom fremtidige data ikke er tilgjengelige. I slike tilfeller er bakoverderivasjon et naturlig og praktisk valg for å estimere den deriverte.

Figur 3.1 illustrerer bakoverderivasjon ved å vise stigningen til funksjonen mellom de markerte punktene. I figuren bruker man punktet t_k og det foregående punktet t_{k-1} for å beregne endringen. Dette uttrykkes som:

$$v(t_k) \approx \frac{u_k - u_{k-1}}{t_k - t_{k-1}} \tag{3.4}$$

Denne ligningen kan også uttrykkes ved å representere tidsintervallet med indekser, der k betegner det nåværende tidspunktet og $k-1$ det foregående. Stigningen mellom disse punktene tilsvarer da den røde linjen i figuren.

$$v(t_k) \approx \frac{u_k - u_{k-1}}{k - (k-1)} \tag{3.5}$$



Figur 3.1: Sekanten beregnet med bakoverderivajon.

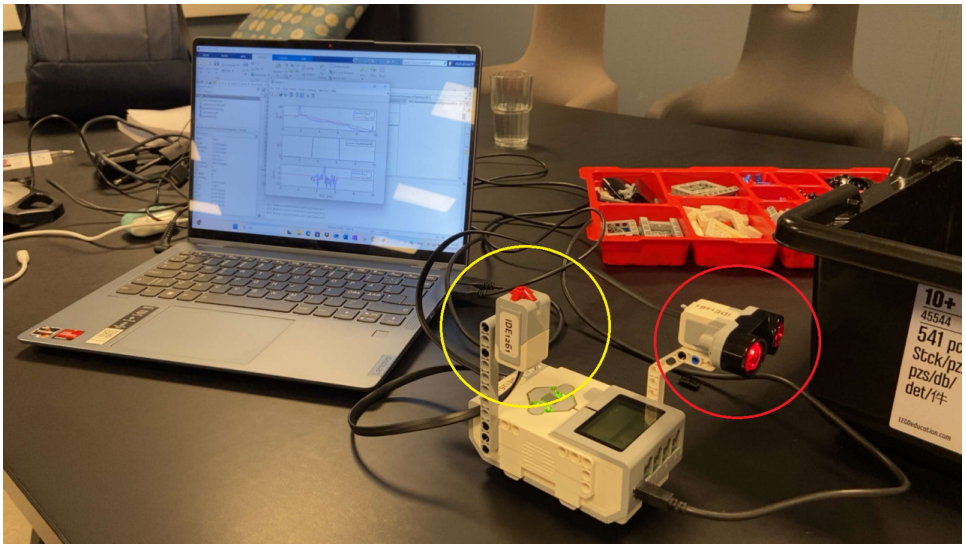
Bakoverderivasjon implementeres i kode slik:

Kode 3.1: Kode hentet fra `P03_NumeriskDerivasjonStigningstall.m`

```
v(k) = (u(k) - u(k-1)) / T_s(k);
```

3.2 Relasjon til stigningstall som en tallverdi

I arbeidet med dette prosjektet har vi demonstrert bruken av bakoverderivasjon ved hjelp av EV3-en. Den har en tilkoblet ultralydssensor markert med rød ring og en lego bryter markert med gul ring som vist i figur 3.4.



Figur 3.2: Utstyret brukt under eksperimentet: Ultralydssensor er markert med rød og lego bryteren med gul ring.

Ved hjelp av ultralydssensoren har vi gjenskapt et praktisk hverdags-eksempel, der vi simulerer hvordan politiet måler hastigheten til en møtende bil med en radarpistol. Dette illustrerte vi ved å bevege en gjenstand mot ultralydssensoren i konstant fart. Deretter trykket vi inn den røde knappen på LEGO-bryteren og holdt den inne i noen sekunder for å registrere farten.

For å implementere eksperimentet i kode definerer vi målingene fra ultralydssensoren som Avstand(k), og målingene fra bryteren som Bryter(k). Dette er vist i kodesnutten 3.2.

Kode 3.2: Kode hentet fra `P03_NumeriskDerivasjonStigningstall.m`

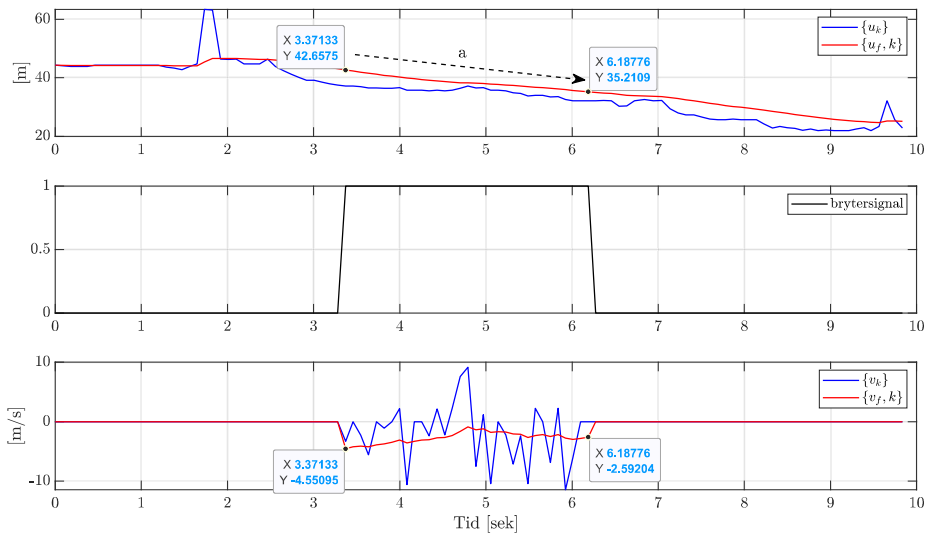
```
% sensorer
Avstand(k) = double(readDistance(mySonicSensor));
Bryter(k)  = double(readTouch(myTouchSensor));
```

Deretter redefinerer vi avstandsmålingene som $u(k)$, før vi filtrerer dem som $u_f(k)$ og deriverer dem numerisk for å beregne hastigheten i tidsrommet hvor bryteren er trykket inn, slik det fremgår av kodeutdraget under.

Kode 3.3: Kode hentet fra `P03_NumeriskDerivasjonStigningstall.m`

```
alfa(k) = 1 - exp(-T_s(k)/tau);
u_f(k) = alfa(k) * u(k) + (1 - alfa(k)) * u_f(k-1);
if Bryter(k)==1
    v(k) = (u(k) - u(k - 1))/T_s(k);
    v_f(k) = (u_f(k) - u_f(k - 1))/T_s(k);
else
    v_f(k)=0;
    v(k)=0;
end
```

Resultatet fra eksperimentet vises i figur 3.5.



Figur 3.3: Øverste delfigur viser en blå graf som representerer de ufiltrerte avstandsmålingene $u(k)$ og en rød graf som representerer de filtrerte avstandsmålingene $u_f(k)$. Midterste delfigur viser statusen til bryteren fra når det trykkes inn til når det slippes. Nederste delfigur viser den deriverte fra tidspunktet knappen er trykt inn til det slippes.

For å bekrefte om resultatet stemmer overens med det vi fant, velger vi to punkter på den røde grafen i øverste delfigur. Dette området representerer en lineært fallende kurve, og kan matematisk beregnes som vist i ligningen under.

$$a = \frac{y_2 - y_1}{t_2 - t_1} = \frac{35.2109 - 42.6575}{6.18776 - 3.37133}$$
$$= \underline{\underline{-2.644}} \tag{3.6}$$

Sammenligningsvis beregner vi gjennomsnittet av området mellom de to punktene i den nederste delfiguren. Dette gjorde vi i matlab for å få et bedre estimat av gjennomsnittsfarten akkurat i det området. Vi har brukt Matlabs innebygde funksjonen, mean for å beregne gjennomssnittet fra indeks 37 til 69 som vist i figuren.

```
>> mean(v_f(37:69))

ans =

   -2.7010
```

Vi ser at våre matematiske utregninger er tilnærmet like med resultatet fra eksperimentet. Derfor kan vi konkludere med at vår metode for numerisk derivasjon er pålitelig.

3.3 Relasjon til den deriverte som en sinusfunksjon

I denne eksempelprosessen har vi brukt samme sinussignal som fra integrasjon. Denne gangen deriverte vi signalet i stedet for å integrere det. Signalet gjenskaper et praktisk scenario der politiet står i bakken under stupet i kongeparken (figur 3.4), og måler hastigheten på vogna med radarpistolen mens den kjører opp og ned.

Figur 3.4: Attraksjon stupet i Kongeparken. Figur hentet fra [1].

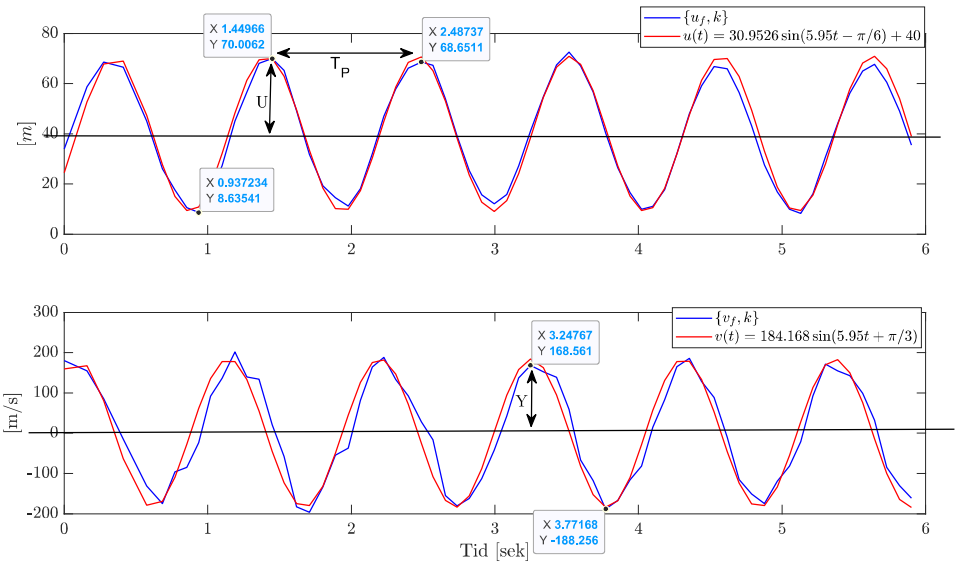
I gjennomføringen av dette eksperimentet har vi først filtrert vår avstandsmåling (u_f, k) med en initialverdi på 40 meter. Intitialverdien representerer at vi starter cirka midt på stupet når vi begynner å ta fartsmålingene. Deretter brukte vi numerisk derivasjon til å beregne hastigheten (v_k) til vogna. Dette kommer frem i kodeutdrag 3.4.

Kode 3.4: Kode hentet fra `estimering_derivasjon.m`

```
if k==1
    % Spesifisering av initialverdier og parametere
    T_s(1) = 0.01;  % nominell verdi
    fc = 0.68;    % ønsket knekkfrekvens i Hz
    tau = 1/(2*pi*fc); % tidskonstant
    v(1)= 0;
    u_f(1)=u(1);
    v_f(1)=0;
else
    T_s(k) = Tid(k) - Tid(k-1);
    alfa(k) = 1 - exp(-T_s(k)/tau);
    u_f(k) = alfa(k) * u(k) + (1 - alfa(k)) * u_f(k-1);
    if Bryter(k)==1
        v(k) = (u(k) - u(k - 1))/T_s(k);
        v_f(k) = (u_f(k) - u_f(k - 1))/T_s(k); %deriverer når ...
            bryteren er trykt inn
    else
        v_f(k)=0;
```

```
            v(k)=0;
        end
    end
```

I figuren under vises resultatet av eksperimentet. Det estimerte sinussignalet (i rødt) er blitt forskjøvet med $-\pi/6$. Dette for å få et riktig estimat av det originalet sinussignalet. Vi så oss nødt til å gjøre dette fordi den estimerte ikke starter på akkurat samme sted som den uestimerte.



Figur 3.5: Øverste delfigur viser en blå graf som representerer de filtrerte avstandsmålingene $u_f(k)$ og en rød graf som representerer den estimerte. Nederste delfigur viser den deriverte av de filtrerte avstandsmålingene og den estimerte av den.

For å verifisere resultatet regner vi ut ampltuden U og vinkelfrekvensen ω til det originale signalet slik vi har gjort i ligning 1.7 og 1.8.

$$U = \frac{70 - 8.63}{2} = 30.69 \tag{3.7}$$

Vinkelfrekvensen beregner vi ut fra perioden i figuren som vist i ligningen under.

$$T_p = 2.48737 - 1.44966 = 1.03771$$
$$\omega = \frac{2\pi}{1.11742} = 6.05 \tag{3.8}$$

Dermed estimeres avstandsmålingene $u(t)$ som:

$$u(t) = 30.69 \cdot sin(6.05 \cdot t - \frac{\pi}{6}) + 40 \tag{3.9}$$

Videre kan vi derivere signalet. Generelt er det deriverte av avstandsmålingene $u(t)$ gitt som:

$$v(t) = \frac{d}{dt} \cdot u(t) = , \quad \text{der}$$
$$u(t) = U \cdot sin(\omega t - \phi) + C \tag{3.10}$$

Dette gir oss at

$$v(t) = U \cdot \omega \cdot cos(\omega t - \phi) \tag{3.11}$$

Setter vi inn verdiene fra tidligere får vi:

$$v(t) = 30.69 \cdot 6.05 \cdot \cos(6.05 \cdot t - \frac{\pi}{6}) \tag{3.12}$$

En cosinusfunksjon kan omskrives som en sinusfunksjon med en faseforskyvning. Dette innebærer at vi kan skrive den deriverte av sinussignalet som:

$$v(t) = \underline{\underline{185.64 \cdot \sin(6.05 \cdot t + \frac{\pi}{3})}} \tag{3.13}$$

3.3 Relasjon til den deriverte som en sinusfunksjon

Vi kan kontrollere amplituden til den deriverte ved å bruke de to punktene i nederste delfigur.

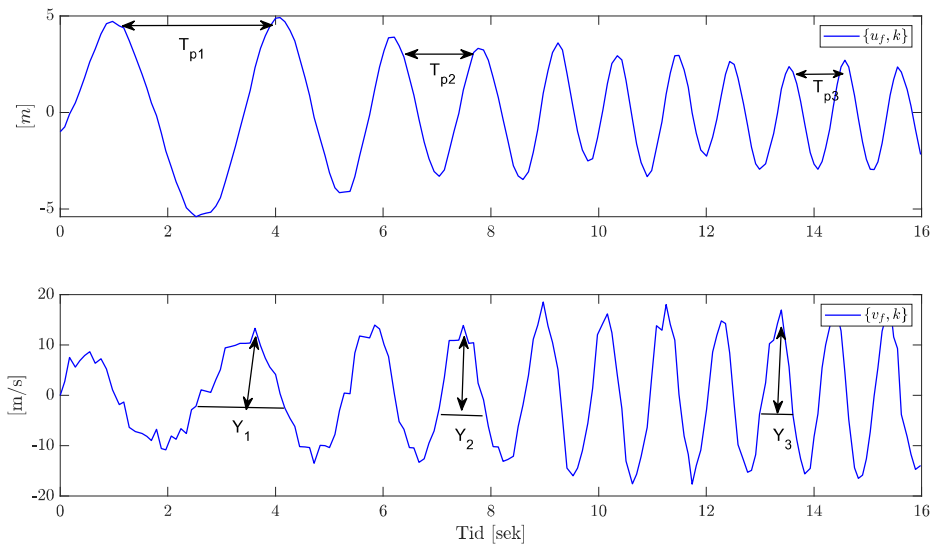
$$Y = \frac{168.561 - (-188.256)}{2} = \underline{\underline{178.41}} \tag{3.14}$$

Vi ser at det er litt avvik mellom de teoretiske beregningene og resulatet. Spesielt er dette gjenkjenbart i amplituden til den deriverte. Dette kan skyldes at det er avvik mellom den estimerte og uestimerte, samt de punktene vi har valgt for å komme frem til de utregningene. Alt i alt kan vi konkludere med at avvikene er akseptable og våre utergninger er pålitelige.

3.4 Numerisk derivasjon av *chirp*-signal

3.4.1 Betydningen av høyere vinkelhastighet i derivasjon

Når frekvensen til signalet $u(t)$ øker, øker også amplituden til det deriverte signalet $v(t)$. Dette fremgår av det generelle uttrykket i ligning 3.11. For å illustrere dette har vi brukt samme figur som ved integrasjon av chirpsignal, men i denne sammenhengen har vi derivert signalet i stedet for å integrere det. Dette resulterte i figur 3.6



Figur 3.6: Øverst delfigur: de filtrerte avstandsmålingene $u(t)$ i meter. Nederst delfigur: deriverte av avstandsmålingene $v(t)$ i meter/sekund, altså hastigheten.

Ut fra figuren kan vi se at perioden i øverste delfigur minker på samme måte som i integrasjon delen fra venstre $T_{p1} > T_{p2} > T_{p3}$. Dette gjør at vinkelfrekvensen øker som resultat av det (ligning 1.8), noe som fører til at amplituden til det deriverte signalet øker fra venstre $Y_1 < Y_2 < Y_3$.

Kapittel 4

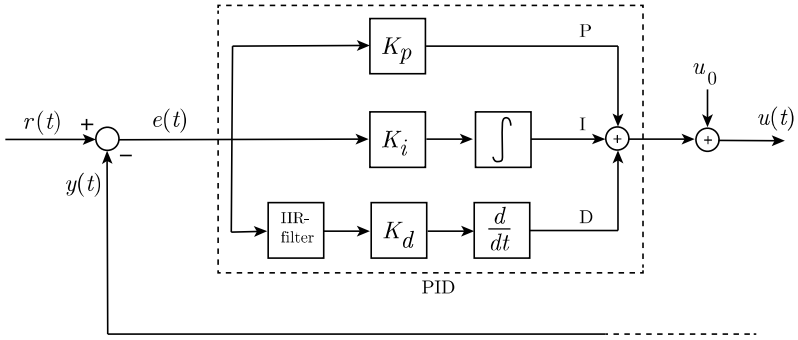
PID regulatoren (utført av hele gruppen)

4.1 Introduksjon

En PID-regulator er en regulator som beregner inngangssignalet til et system ved hjelp av tre matematiske operasjoner: proporsjonal forsterkning (P), integralvirkning (I) og derivatvirkning (D). Den brukes blant annet i reguleringsteknikk og cruisekontroll i biler for å kontrollere prosesser i både elektriske og mekaniske systemer, som motorer, pumper og varmeelementer. Hensikten med regulatoren er å oppnå og opprettholde en ønsket tilstand, for eksempel et bestemt nivå av en temperatur, trykk eller volum. PID uttrykkes matematisk på følgende måte:

$$u(t) = \underbrace{u_0}_{\text{Basispådrag}} + \underbrace{K_P \cdot e(t)}_{P\text{-leddet}} + \underbrace{\int_0^t K_I \cdot e(\tau)\, d\tau}_{I\text{-leddet}} + \underbrace{\frac{d}{dt}\left(K_D \cdot e_f(t)\right)}_{D\text{-leddet}} \tag{4.1}$$

Figur 4.1 viser et blokkskjema av ligning 4.1, hvor P-delen, I-delen og D-delen er presentert som parallelle parametere.



Figur 4.1: Blokkskjema over PID-regulatoren.

Matematikken kan forklares som det følger av i forklaringen under:

• $e(t)$ er reguleringsavviket, og kan bli beregnet ved å ta differansen mellom ønsket verdi $r(t)$ og den målte verdien $y(t)$.

• u_0 er basispådraget til motoren, og det er ikke alltid nødvendig å ha det med.

• K_P, K_I og K_D er reguleringsparameterne som justeres avhengig av prosess.

En fordel med PID-regulatoren er at de matematiske delene er uavhengige av hverandre. Dette gjør det mulig å teste de ulike operasjonene separat, og hver del kan analyseres og justeres individuelt.

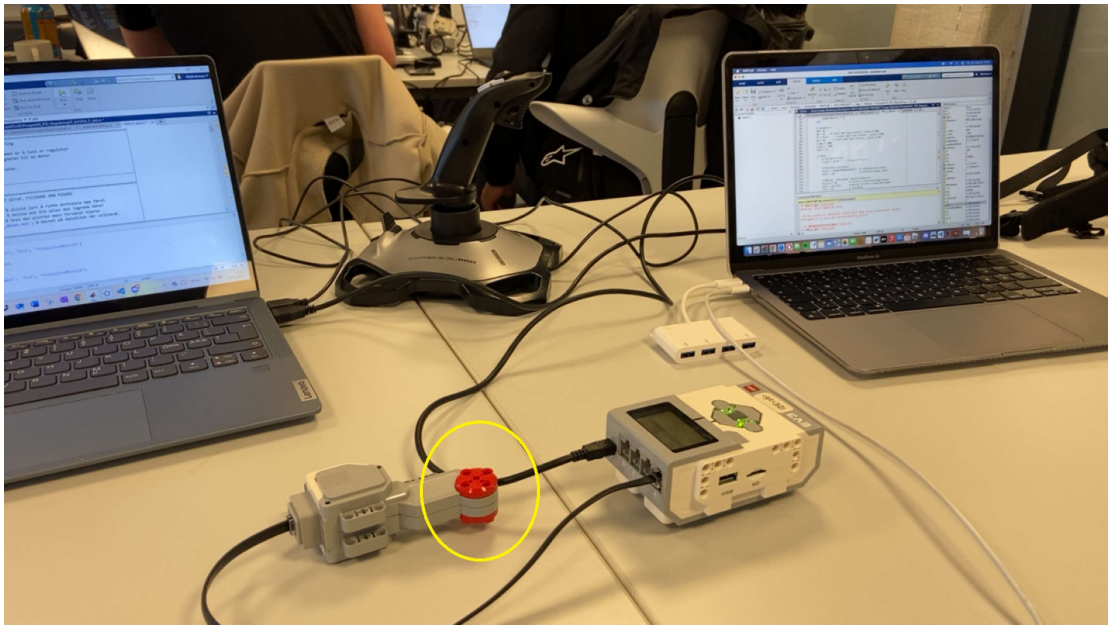
I kodesnutten under implementerer vi en PID-regulator. Implementasjonen følger en strukturert fremgangsmåte. Først beregnes reguleringsavviket, etterfulgt av den proporsjonale-delen og den integrale-delen. Deretter lavpassfiltreres avviket før det benyttes i beregningen av derivasjonsleddet. For I- og D-delene benyttes henholdsvis numerisk integrasjon og bakoverderivasjon, slik at de kan implementeres effektivt i kode.

Kode 4.1: P04-D-delen.m

```

e(k) = r(k)-y(k); % Reguleringssavvik
P(k) = Kp*e(k);
I(k) = I(k-1) + Ki*1/2*T_s(k)*(e(k) + e(k-1));
e_f(k) = (1-alfa(k))*e_f(k-1) + alfa(k)*e(k);
D(k) = Kd*((e_f(k)-e_f(k-1))/T_s(k));
u_A(k) = u0 + P(k) + I(k) + D(k);
```

4.2 Stryring av motor med P-del

For å analysere hvordan en PID-regulator oppfører seg når kun P-leddet benyttes, har vi koblet en motor til EV3-en, vist med gul markering i figur 4.2. I dette oppsettet er det kun det proporsjonale leddet som genererer pådraget $u(t)$, beskrevet i ligning 4.1.



Figur 4.2: Utstyr brukt under eksperimentet. Motoren er markert med gul ring, og data hentes i Matlab via EV3-en.

$$P(k) = K_P \cdot e(k) \tag{4.1}$$

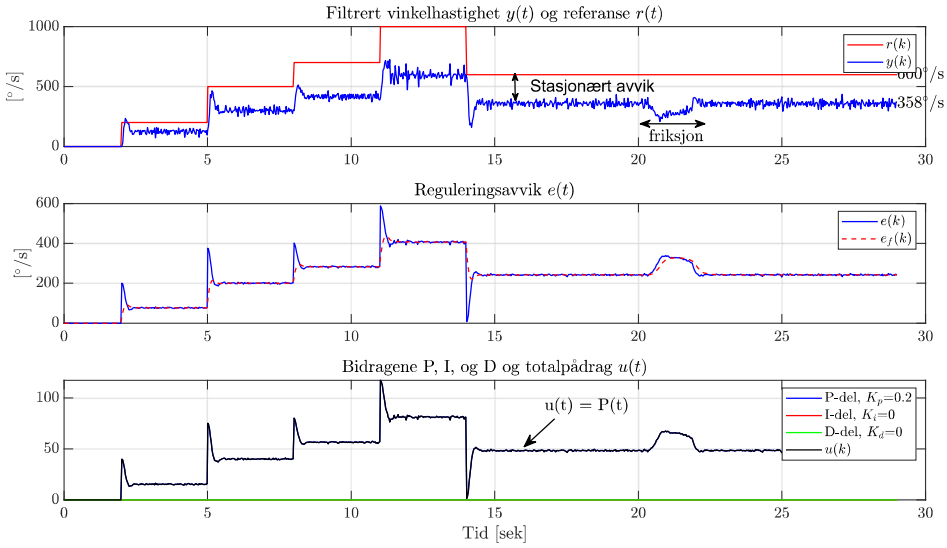
Som ligningen viser, er pådraget direkte proporsjonalt med reguleringsavviket $e(k)$. Dette betyr at et vedvarende avvik mellom referansen og faktisk verdi må alltid være til stede for at systemet skal generere pådrag. Når avviket går mot null, forsvinner også pådraget, noe som kan føre til at systemet stopper opp. Bruken av kun P-leddet fører derfor til et stasjonært avvik mellom målt verdi $y(t)$ og referanseverdi $r(t)$.

Eksperimentet er implementert i Matlab, som vist i kodeutdrag 4.2. Her settes I- og D-leddene til null, og ulike verdier for K_P testes til ønsket respons oppnås, og i det endelige oppsettet benyttes en K_P-verdi på 0.2. Det er viktig å holde K_P lavt og tilpasset systemet, ettersom en for høy verdi kan føre til oscillasjoner, der systemet beveger seg frem og tilbake. En høy K_P-verdi gir riktignok et kraftigere pådrag, og en eventuell dobling av K_P kan halvere reguleringsavviket, siden mindre avvik er nødvendig for å generere et pådrag. Likevel kan en slik økning svekke stabiliteten til systemet. Dette understreker viktigheten av å velge en balansert parameterverdi.

Kode 4.2: P04-P-delen.m

```
u0 = 0;
Kp = 0.2; %vårt valgte Kp-verdi
Ki =0;
Kd = 0;
u_A(k) = u0 + P(k) + I(k) + D(k);
```

Resultatene fra eksperimentet er presentert i figuren nedenfor. Som vi kan observere i det øverste plottet, oppstår det et stasjonært avvik der referanseverdien stabiliserer seg på omtrent 600°/s, mens den faktiske målte vinkelhastigheten kun når 358°/s.



Figur 4.3: I den øverste delfiguren vises referansesignalet $r(t)$ i rødt og den faktiske målte verdien $y(t)$ i blått. I midterste delfigur er det plottet reguleringsavviket $e(t)$ med blå farge, og det lavpassfiltrerte reguleringingsavviket med stiplet rød farge. I den nederste delfiguren er de ulike reguleringskomponentene plottet: P-leddet (proporsjonal), I-leddet (integral) og D-leddet (derivasjon), sammen med det totale pådraget $u(t)$ som blir generert

Et godt eksempel for å forstå resultatet presentert i figur 4.3, er cruisekontrollen i en bil. Her representerer referansen den ønskede hastigheten bilen skal holde, mens $y(t)$ beskriver den faktiske farten.

For å illustrere hvordan bilen påvirkes av motstand, har vi inkludert en friksjonseffekt som simulerer en motbakke, samtidig som motorens pådrag er konstant. Motstanden fra bakken fører naturligvis til en reduksjon i bilens faktiske hastighet. Siden systemet kun benytter P-leddet alene, klarer det ikke å kompensere fullt ut for denne motstanden. Dette medfører at reguleringsavviket øker, slik det fremkommer i figuren. Resultatet av denne motstanden blir at bilens faktiske hastighet er lengre unna ønsket hastighet under kjøringen oppover denne bakken.

4.3 Stryring av motor med I-del

I denne delen skal vi generere motorens pådrag, ved å benytte kun I-leddet i regulatoren. Det betyr at både P- og D-leddene settes til null. Fremgangsmåten følger samme prinsipp som for P-delen, beskrevet i kodeutdrag 4.2.

I-leddets oppgave er å akkumulere reguleringsavviket $e(t)$ over tid, slik at reguleringsavviket går gradvis mot null. For å finne en passende forsterkning for I-leddet justerer vi verdien for K_I gradvis i koden til vi oppnår ønsket respons. Verdien som er benyttet i dette prosjektet er 0.15, slik det fremgår av resultatene i figur 4.4 og 4.5. Høyere K_I verdi gir raksere akkumulering av pådrag, men for høy K_I verdi kan gi oscillering.

Matematisk kan det integrerende leddet skrives som:

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

Ligningen implementeres som en rekursiv sum i kode, som vist i kodeutdraget under.

Kode 4.3: Kode hentet fra `P04_I-delen.m`

```
I(k) = I(k-1) + Ki * 0.5 * T_s(k) * (e(k) + e(k-1));
```

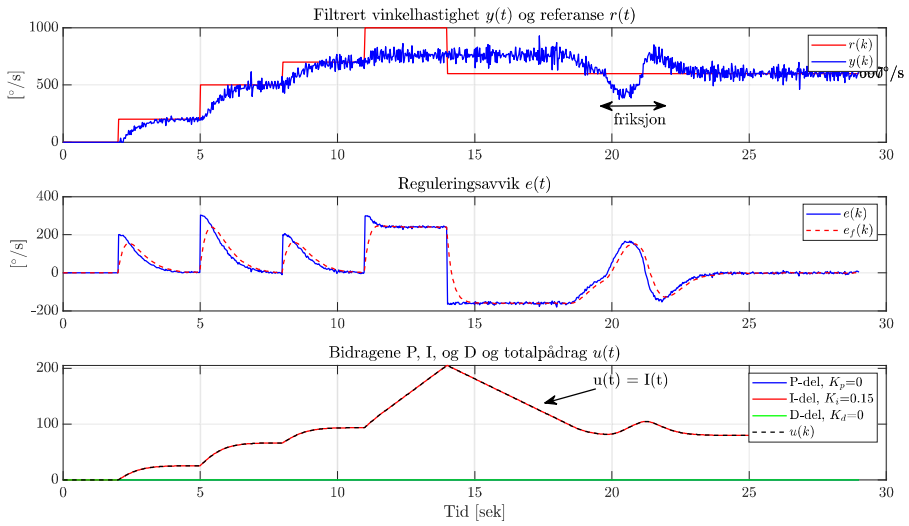
Vi har implementert I-leddet på to ulike måter: én med integratorbegrensning, og én uten. Kodeutdraget under viser hvordan begrensningen av integralet er satt opp:

Kode 4.4: Kode hentet fra `P04_I-delen.m`

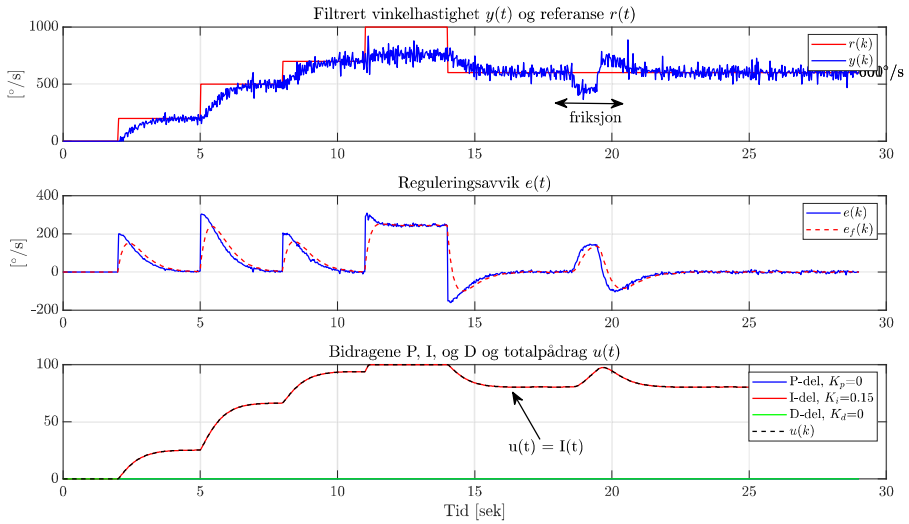
```
I_max = 100;
I_min = -100;

if I(k) > I_max
    I(k) = I_max;
elseif I(k) < I_min
    I(k) = I_min;
end
```

I-leddet kan vokse ukontrollert og bli uendelig stort. Som følge av dette kan pådraget $u(t)$ også bli uendelig stort, og til og med større enn det systemet fysisk kan levere. Ved å begrense størrelsen på I-leddet reduserer man risikoen for såkalt integrator-windup. Dette sikrer at I-leddet aldri overstiger de definerte grensene, noe som bidrar til et mer stabilt og forutsigbart reguleringssystem.

Resultatene av simuleringen er vist i de to figurene nedenfor. Den første viser simuleringen uten integratorbegrensning, mens den andre inkluderer en begrensning på mellom -100 og 100:



Figur 4.4: Simulering uten integratorbegrensning. Øverst delfigur: referansesignal $r(t)$ (rød) og målt verdi $y(t)$ (blå). Midterste delfigur: reguleringsavvik e(t) (blå) og lavpassfiltrert avvik (stiplet rød). Nederste delfigur: regulatorbidragene P (blå), I (rød) og D (grønn) samt totalpådraget u(t) (stiplet svart).



Figur 4.5: Simulering med integratorbegrensning på maksimum 100 og minimum -100.

For å tolke resultatene presentert i figurene kan vi benytte samme eksempel som i forrige oppgave, der vi brukte cruisekontroll i en bil. Referansen representerer fartsgrensen bilen skal holde, mens friksjonen representerer motstanden fra en oppoverbakke bilen må bestige. I motsetning til P-leddet, som er avhengig av reguleringsavviket for å generere et pådrag, genererer I-leddet pådrag ved å integrere reguleringsavviket over tid. Dette innebærer at I-leddet bruker arealbetraktninger for å håndtere vedvarende feil, spesielt store og langvarige avvik, slik at reguleringsavviket gradvis reduseres mot null. På denne måten kan bilens cruisekontroll redusere effekten fra bakken på bilen hastighet ved hjelp av I-leddet.

4.4 Stryring av motor med D-del

For å teste virkningen av D-leddet benytter vi utelukkende pådraget fra denne komponenten. I koden gjøres dette ved å sette de øvrige parameterne til null (kode 4.2). Deretter prøver vi ulike verdier for K_D inntil vi finner en passende verdi. Matematisk kan D-leddet uttrykkes som vist i ligning 4.3.

$$u(t) = \frac{d}{dt} \cdot (K_D \cdot e_f(t)) \tag{4.3}$$

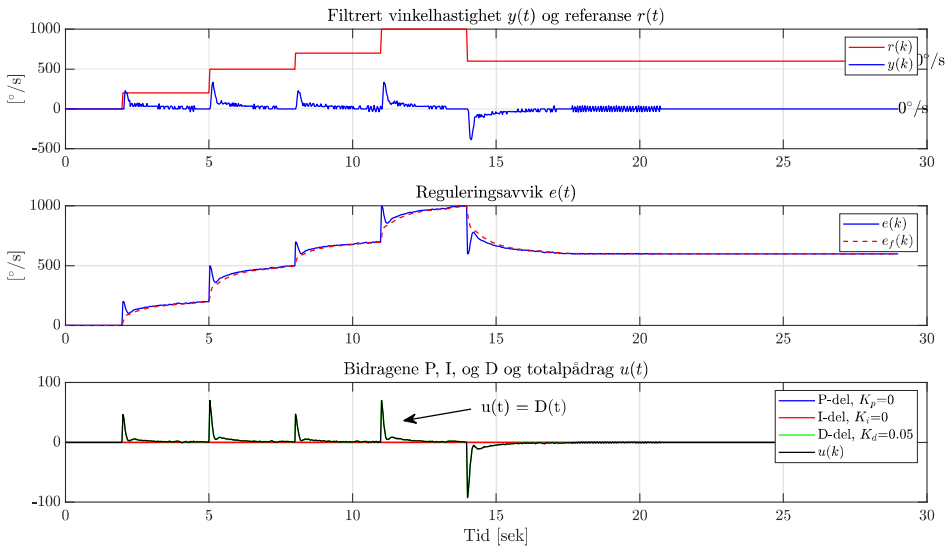
I ligningen kommer det frem at vi lavpassfiltrerer reguleringsavviket $e(t)$. Dette blir gjort for å redusere støy og forhindre at denne forårsaker uønskede og raske endringer i signalet.

I kodeutdrag 4.5 viser vi hvordan lavpassfiltreringen er implementert, samt hvilken verdi vi har valgt for tidskonstanten τ. I koden har vi benyttet en verdi på 0,05 for K_D, på samme måte som vi tidligere gjorde for P-leddet.

Kode 4.5: Kode hentet fra `P04_D-delen.m`

```
tau = 0.2;        % tidskonstant til filteret
alfa(k)   = 1-exp(-T_s(k)/tau);   % tidsavhengig alfa
e_f(k) = (1-alfa(k))*e_f(k-1) + alfa(k)*e(k);
D(k) = Kd*((e_f(k)-e_f(k-1))/T_s(k));
```

I figur 4.6 er det vist grafen D-leddet.



Figur 4.6: Simulering med kun D-leddet.

D-leddet består av den deriverte av det lavpassfiltrerte reguleringsavviket $e(t)$, slik det er vist i ligningen og implementert i kodeutdraget ved hjelp av bakoverderivasjon. Dette leddet analyserer hvordan reguleringsavviket endrer seg over tid, og bidrar til å forhindre raske endringer i signalet. Motoren roterer bare i øyeblikket feilsignalet endres. Mellom disse hendelsene fallder D-pådraget tilbake til 0 . Størrelsen på D-pådraget er proposjonal med K_D. En høy K_D verdi gir kraftigere demping av raske endringer, men kan også føre til at systemet oscillererer dersom K_D blir for stort

I eksempelet med bilen som kjører opp en oppoverbakke, vil en eventuell inkludering av D-leddet initiere en reaksjon i det øyeblikket endringen i fart begynner å skje. Det reagerer altså på selve tendensen til fartsendring, ikke på avviket i seg selv. Dette gjør at cruisekontrollen kan gripe inn tidligere og mer presist, noe som demper overskyting og reduserer svingninger i farten.

4.5 Stryring av motor med PI-del

I denne oppgaven har vi kombinert P og I leddene, for å se hva som skjer. I mange industrielle anvendelser benyttes PI-regulatoren framfor den fullstendige PID-regulatoren. Hovedårsaken til dette er at D-lettet er følsomt for raske endringer i referanseverdien. Oppgaven ble utført på samma måte som oppgavene ovenfor, og oppsettet er lik som i figur 4.2. Det samlet motorpådraget summerer P og I bidragene sammen, og vi får likningen

$$u_a(k) = P(k) + I(k) \tag{4.4}$$

der P-leddet $P(k)$ kommer fra likning 4.1 og I-leddet $I(k)$ kommer fra likning 4.2

P-delen gir en rask respons når det oppstår avvik mellom referanse verdeen og systemets utgang. Med en k_P verdi på 0.300 vil feilen medføre en proposjonal endring i pådraget. Dette gir systemet en rask startsrespos. Ulempen med bare en ren P-regulator er at den ofte etterlater noe som kalles for en steady-state error. Dette refererer til den konstante differansen mellom referanseverdien og systemets faktiske utgang. I dette tilfellet, vil en liten feil opprettholdes fordi P-delen alene ikke klarer å fjerne avviket.

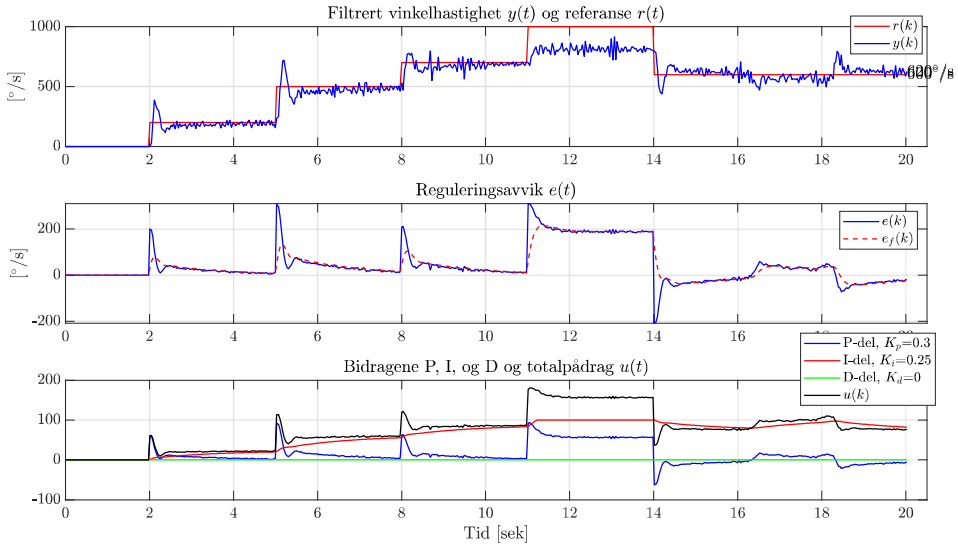
Som nevnt tidligere, er I-leddets oppgave å akkumulere reguleringsavviket $e(t)$ slik at avviket gradvis går mot null. Med en k_i verdi på 0.230, sørger vi for at små feil som vedvarer over tid blir retter på. I begynnelsen kan I-leddet virke mindre kraftfull enn P-leddet, men den vil etterhvert kompensere for avvik slik at feilen elimineres. Uten at I-delen kombineres med et P-ledd, som gir umiddebar respons, eller et D-ledd som demper og forutsier fremtidige svinginger, kan I-leddet fort bygge opp en stor akkumulasjon dersom feilen varer over lengre tid.

Ved å sette sammen P- og I- leddene oppnår vi en regulator som får det beste fra begge. P-delen gir en rask respons og bidrar til en rask oppstart i systemet. I-delen sørger for at avvik over tid akkumuleres og korrigeres, slik at systemet oppnår en nøyaktig samsvar mellom ønsket og faktisk verdi.

Kode 4.6: Kode hentet fra `P04_PI-delen.m`

```
u0 = 0;
Kp = 0.300;     % start med lave verdier, typisk 0.005
Ki = 0.250;  %0.135  %185   % start med lave verdier, typisk 0.005
Kd = 0.000;      % start med lave verdier, typisk 0.001

u_A(k) = u0 + P(k) + I(k) + D(k);
```

Resultatet av kodeutdraget gir oss figuren 4.7. Visuelt kan du se at P-leddet ofte gir et stort pådrag i starten ved et steg, mens I-leddet gradvis bygger opp og justerer den totale responsen. Dette fører til at det totale signalet til slutt oppnår ønsket verdi uten steady-state error, i motsetning til bruk av kun en P eller I regulator.



Figur 4.7: Simulering med P og I leddene.

Kort oppsummert fungerer et rent I-ledd når hovedmålet er å eliminere små feil, men ikke optimalt for systemer med rask respons og når demping mot svingninger er nødvendig. Derfor kombineres I-leddet ofte med P-leddet og eventuelt D-leddet slik at man får en god regulator som reagerer fort og over tid korrigerer feil. Det er denne kombinasjonen som gir en balanse mellom rask og langsiktig presisjon. Og det er nettop det du observerer i figur 4.7, der P-leddet umiddelbart tar fatt i feilen, mens I-leddet gradvis justerer styringen slik at systemets utgang til slutt samsvarer med referanen.

4.6 Stryring av motor med PID-del

I gjennomføringen av dette eksperimentet benyttet vi de tre delene i en PID-regulator for å regulere en turtallsregulator. Dette gjorde vi ved å teste parameterne, K_P, K_I og K_D ved hjelp av bruk-og-feil-metoden for å oppnå ønsket respons. Eksperimentet ble gjennomført med et basispådrag u_0 på 0.

Vi startet med å justere K_P-verdien til vi fant en passende respons fra systemet. Deretter justerte vi K_I for å redusere eventuelle vedvarende avvik mellom

referanse og faktisk verdi. Til slutt valgte vi en passende K_D-verdi, men holdt denne relativt lav for å unngå for aggressiv pådragsbruk. D-leddet ble vurdert som nødvendig for å dempe oversving og bidra til en mer stabil respons.

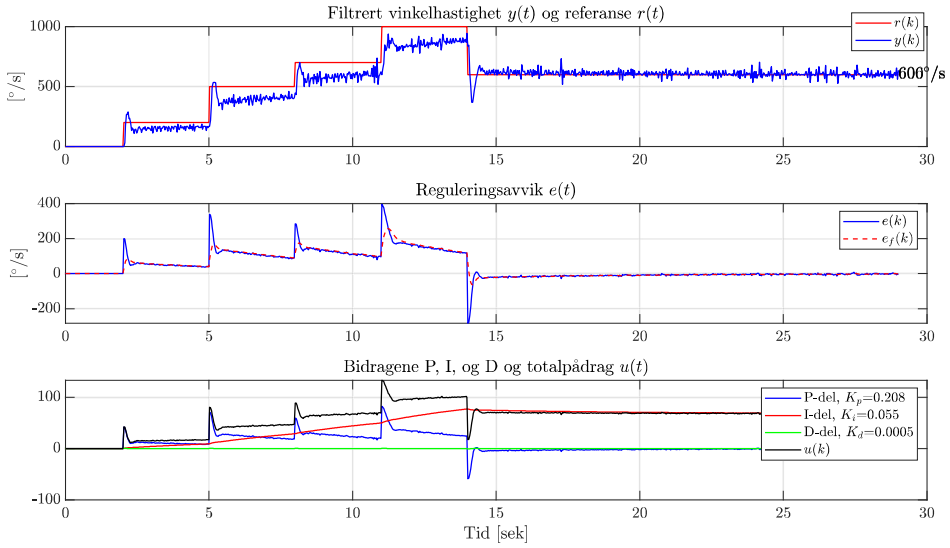
For å styre motoren med PID-regulering har vi implementert en egen funksjon som håndterer utregningene for PID-regulatoren. Funksjonen MinPID beregner P-leddet direkte, I-leddet ved hjelp av trapesmetoden, og D-leddet ved bruk av bakoverderivasjon. I tillegg er integratorbegrensingen for I-leddet implementert i funksjonen. Beregningene utføres ved å kalle funksjonen fra hovedkoden. Et utdrag av hovedkoden er vist i kodesnutten nedenfor.

Kode 4.7: Kode hentet fra `P04_PID-delen.m`

```

para = [Kp, Ki, Kd, I_max, I_min, alfa(k)];
[P(k),I(k),D(k),e_f(k)] = ...
    MinPID(I(k-1),e_f(k-1),e(k-1:k),T_s(k),para);
u(k) = u0 + P(k) + I(k) + D(k);
```

I kodesnutten samles alle parameterne for PID-regulatoren i variabelen para, som deretter brukes som input til MinPID-funksjonen. Etter at funksjonen er kalt, regnes pådraget ut ved å summere basispådraget med de beregnede PID-komponentene. Selve funksjonen for PID-regulatoren er vist i kodeutdrag 4.8. Blokkskjema 4.9 viser hvordan koden for PID-koden fungerer.

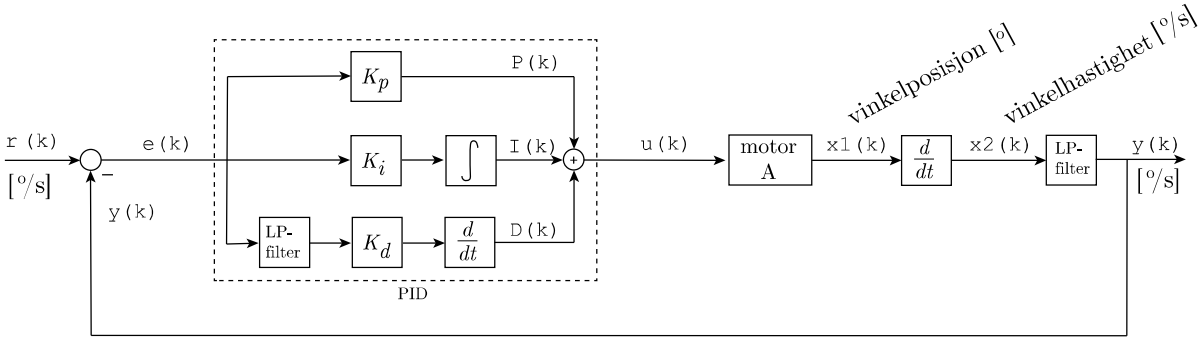
Resultatet fra eksperimentet er presentert i figuren nedenfor.



Figur 4.8: Simulering med P, I og D leddene.

I resultatet som vises i figur 4.8, har vi generert et utgangssignal basert på PID-regulatoren. Det er fortsatt litt avvik i resultatet, noe som kan skyldes flere faktorer. For det første vil en begrensning på integratoren kunne hindre I-leddet i å akkumulere nok reguleringsavvik over tid til å fullstendig eliminere det stasjonære avviket. I tillegg kan måleusikkerhet, eller forsinkelser i systemet påvirke reguleringens nøyaktighet.

4.7 Funksjon av PID-regulatoren



Figur 4.9: Blokkskjemaet viser hvordan en PID-regulator brukes for å styre vinkelhastigheten til en motor.

Kode 4.8: Kode hentet fra `P04_PID-delen.m`

```
% function [P, I_new, D, e_f_new] = MinPID(I_old, e_f_old, e, ...
    T_s, para)
%
%   PID-regulator som anvender trapesmetoden, bakoverderivasjon og
%   første ordens lavpassfilter. Inkluderer integratorbegrensing.
%
%   Funksjonsargumenter inn:
%   - I_old:    forrige integralverdi I(k-1)
%   - e_f_old:  forrige filtrerte reguleringsavvik e_f(k-1)
%   - e:        de to siste reguleringsavvik e(k-1:k)
%   - T_s:      tidsskritt T_s(k)
%   - para:     parametervektor [Kp, Ki, Kd, I_max, I_min, alfa(k)]
%
%   Returargumenter:
%   - P:        proporsjonaldel P(k)
%   - I_new:    integraldel I(k)
%   - D:        derivatdel D(k)
%   - e_f_new:  filtrert reguleringsavvik e_f(k)
%
%   Syntaks for bruk av funksjonen:
%   -------------------------------
%   fc = ..;                        % ønsket knekkfrekvens
%   tau = ...;                      % ønsket tidskonstant filter
%   alfa(k) = 1-exp(-Ts(k)/tau);    % alpha-verdi lavpassfilter
%   para = [Kp, Ki, Kd, I_max, I_min, alfa(k)];
%   [P(k),I(k),D(k),e_f(k)] = ...
    MinPID(I(k-1),e_f(k-1),e(k-1:k),T_s(k),para)
%   u(k) = u0 + P(k) + I(k) + D(k);
```

```
function [P, I_new, D, e_f_new] = MinPID(I_old, e_f_old, e, ...
    T_s, para)


% ------------------------------------------------------------
% Parametere
% ------------------------------------------------------------
Kp    = para(1);
Ki    = para(2);
Kd    = para(3);
I_max = para(4);
I_min = para(5);
alfa  = para(6);

% ------------------------------------------------------------
% Bidragene P, I og D
% ------------------------------------------------------------
P = Kp * e(end);
I_new = I_old + Ki * T_s/2 * (e(end) + e(end-1));
e_f_new = (1 - alfa) * e_f_old + alfa * e(end);
D = Kd * (e_f_new - e_f_old) / T_s;

% ------------------------------------------------------------
% Integratorbegrensing
% ------------------------------------------------------------
    if I_new > I_max
        I_new = I_max;
    elseif I_new < I_min
        I_new = I_min;
    end
end
```

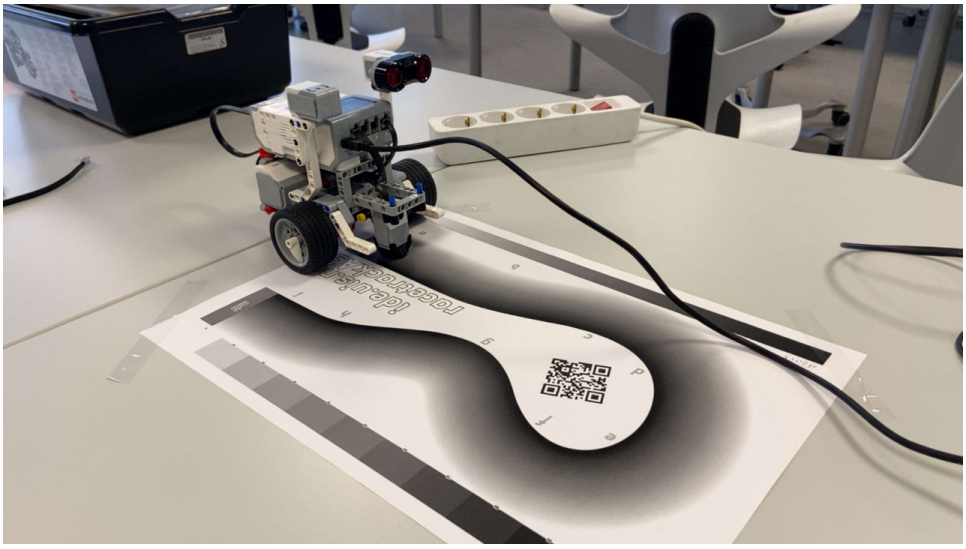
Kapittel 5

Manuell kjøring av Lego-robot

5.1 Introduksjon

I dette eksperimentet har vi laget en kode for å kjøre legoroboten i figur 5.1 manuelt med styrestikken. Hensikten var å samle inn måledata fra kjøringen, for deretter å analysere og sammenligne resultatet.

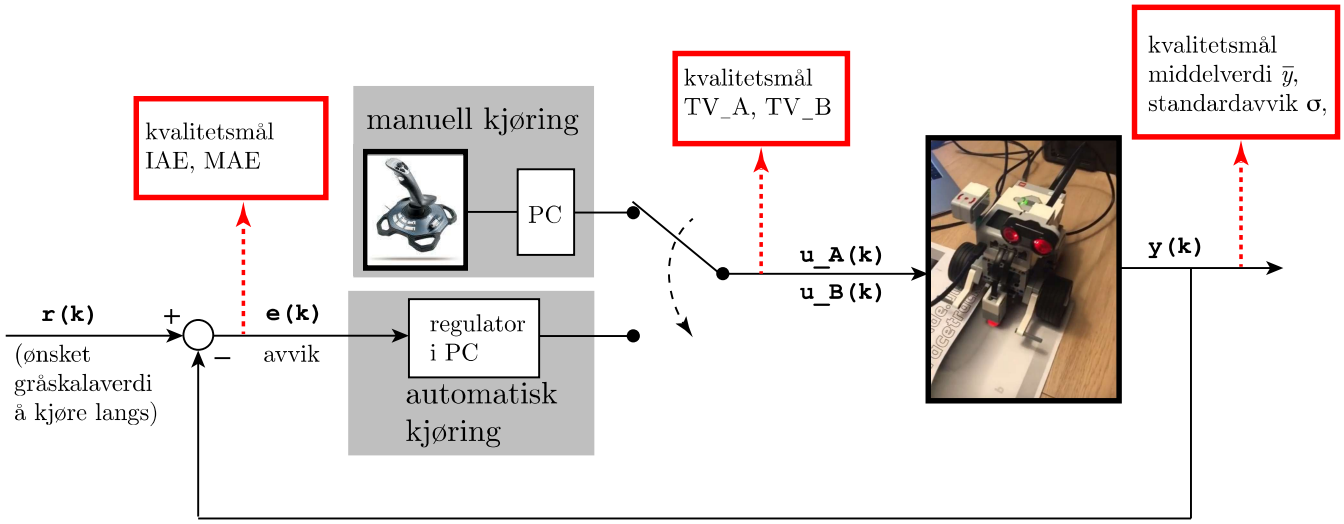
For å samle inn data brukte vi lyssenoren på den grå kjørebanen i figur 5.1, der mørke områder representerer lite reflektert lys, mens lyse områder representerer høy refleksjon. Det er disse målingene som danner grunnlaget for å evaluere kjøringen til hvert gruppemedlem.



Figur 5.1: Bilde av legoroboten på startpunket av kjørebanen. Kjørebanen er det grå område på arket.

For å evaulere kjøringen til hvert gruppe medlem har vi tatt i bruk noen industrielle kvalitetsmål. Vi har har brukt de tre signalene i figur 5.2, nemlig

- Reguleringsavviket $e(t)$
- Motorpådragene $u_A(k)$ og $u_D(k)$
- Lysmålingene $y(k)$



Figur 5.2: Prinsippskisse for manuell og automatisk kjøring av LEGO-robot med uthenting av data for beregning av kvalitetsmål for kjøringen.

5.2 Kode for styring av robot

For å lage en kode som lar LEGO-roboten styres med styrestikken, begynte vi med å implementere den delen av programmet som håndterer styrestikken. Denne koden gjør det mulig å kjøre roboten forover, bakover og svinge til sidene. Dette illustreres i kodeutdraget nedenfor.

Kode 5.1: Kode hentet fra `P05_manuellkjoring.m`

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

I starten av koden henter vi verdiene fra styrestikken ved hjelp av funksjonen HentJoystickVerdier, og lagrer dem i variablene JoyAxes og JoyButtons. Deretter tilordner vi skyteknappen variabelen JoyMainSwitch, som tilsvarer JoyButtons(1). Videre bruker vi aksene 1 og 2 til å hente verdier for henholdsvis svingbevegelse og fremover-/bakoverbevegelse.

Etterpå lagde vi en kode for pådragene til to de motorene, men siden de to motoren er er uavhengige av hverandre lager vi kode for hver av dem.

Kode 5.2: Kode hentet fra `P05_manuellkjoring.m`

```
% beregning av pådrag fra styrestikken
u_A(k) = foran(k) + sving(k);
u_D(k) = foran(k) - sving(k);
```

I kodeutdrag 5.2 er tanken at når vi kjører roboten rett frem eller bakover har de to motorene likt pådrag. Ved svingning til sidene derimot, har de to motorene motsatt pådrag. Det betyr at den ene motoren har positivt pådrag, mens den andre har negativt pådrag. Dette skaper rotasjonsbevegelsen vi bruker for å svinge roboten til sidene.

I tillegg inkluderte vi noen linjer i koden som gjør at programmet avsluttes automatisk når roboten kjører inn på et hvitt område. Først identifiserte vi lysverdien som sensoren måler på hvitt underlag. Deretter la vi inn en if-setning som aktiverer skyteknappen dersom Lys(k) overstiger denne verdien. Dette er vist i kodesnutten nedenfor.

Kode 5.3: Kode hentet fra `P05_manuellkjoring.m`

```
hvitt = 56;
if Lys(k) > hvitt %stopper programmet hvis hvitt omraade er ...
    avdekket
    JoyMainSwitch = 1;
end
```

5.3 Kvalitetsmål med kode

For å sammenligne kjøringen til hvert gruppemedlem har vi benyttet kvalitetsmål som Total Variation (TV), Mean of Absolute Error (MAE) og Integral of Absolute Error (IAE). Disse målene brukes blant annet i reguleringsteknikk for å kvalitetssikre målinger.

For å kunne gjøre disse beregningene benytter vi følgende målinger:

- Målinger fra lyssensoren, $Lys(k)$
- Reguleringssavviket $e(k)$, som vi finner ved å ta differansen mellom første målte verdi og lysmålingene.
- Pådraget fra motor A, u_A og pådraget fra motor D, u_D.

5.4 Kvalitetsmål under kjøring

MAE, IAE og TV er målinger som kan beregnes i sanntid under selve kjøringen. Dette er nyttig i en praktisk sammenheng, ettersom man kan kvalitetssikre samtidig som prosessen pågår. I dette tilfellet fikk vi se utviklingen av kjøringen.

- **Mean of Absolute Error** bruker vi til å beregne gjennomsnittet av reguleringsavviket. Dette gjør vi ved å summere absoluttverdiene av reguleringsavvikene, og normalisere dem i forhold til antall målinger. Vi benytter absoluttverdien fordi avviket $e(t)$ kan være både negativt og positivt. Uten dette kunne vingelete kjøring, der man kjører like mye på mørkt som på hvitt bli tolket som perfekt kjøring. Matematisk kan vi uttrykke Mean Of Absolute Error (MAE) som:

$$MAE = \frac{1}{k} \cdot \sum_{n=1}^{k} |e_n| \tag{5.1}$$

Vi implementerer summen i programmeringsspråket MATLAB ved å bruke den innebygde funksjonen, sum. Koden er vist i utdraget 5.4.

Kode 5.4: Kode hentet fra `P05_manuellkjoring.m`

```
sum_abs_e(k) = 0; %initialverdi
sum_abs_e(k) = 1/k * sum(abs(e(1:k)));
```

- **Integral Of Absolute Error** er et kvalitetsmål fra reguleringsteknikken som baserer seg på integrasjon. Kvalitetsmålet kan beregnes matematisk som:

$$IAE = \int_0^t |e(\tau)| \ d\tau \tag{5.2}$$

Dette er de absolutt verdiene av arealet under kurven $|e(t)|$, brukt av samme grunn som i MAE.

For å finne arealet bruker vi numerisk integrasjon i koden, som vist nedenfor.

Kode 5.5: Kode hentet fra `P05_manuellkjoring.m`

```
IAE(k) = IAE(k - 1) + T_s(k)*(1/2*(abs_e(k - 1) + ...
    abs_e(k)));
```

- **Total Variation** er et kvalitetsmål som brukes i reguleringsteknikk for å vurdere hvor aktivt pådragsorganet benyttes. Dette er spesielt nyttig i industrielle sammenhenger, der man ønsker å unngå unødvendig slitasje på komponenter som for eksempel ventiler. Derfor kan det i mange tilfeller være hensiktsmessig å akseptere et visst avvik, dersom det bidrar til redusert belastning på utstyret. Dette kvalitetsmålet formuleres slik:

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

Matematikken implementerer vi for begge de motorene vi har brukt under kjøringen av roboten. Et utdrag av dette er vist nedenfor:

Kode 5.6: Kode hentet fra `P05_manuellkjoring.m`

```
%initialverdier
TV_A(1) = 0;
TV_D(1) = 0;
% Berekning av TV
TV_A(k) = TV_A(k-1) + abs(u_A(k) - u_A(k-1));
TV_D(k) = TV_D(k-1) + abs(u_D(k) - u_D(k-1));
```

5.5 Kvalitetsmål etter kjøring

Etter at kjøringene er ferdige, utfører vi de siste kvalitetsmålingene. Her benytter vi middelverdi og standardavvik for å analysere stabilitet og variasjon i resultatene.

- **Middelverdi** ($\bar{y}$) er gjennomsnittet av målingene $y(k)$, altså summen av alle målingene delt på antall målinger. Det bruker vi for å finne hvilken verdi den overordnet kjøringen ligger på. En god kjøring ville ligget på omtrent referanseverdien. Middelveriden er matematisk uttrykt som:

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

- **Standardavvik** (σ) bruker vi for å vurdere hvor spredt målingene er i forhold til middelverdien. En liten verdi for standardavviket forteller oss at målinger (data) er samlet tett rundt middelverdien. Mens en stor verdi forteller oss at målingene er spredt rundt om middelverdien. Formelen for standardavviket er gitt som:

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

I koden benytter vi Matlabs innebygde funksjon, mean for å beregne middelverdien, og std for å beregne standardavviket. Dette illustreres i kodeutdraget nedenfor:

Kode 5.7: Kode hentet fra `P05_manuellkjoring.m`

```
1 middelverdi= mean(y);
2 std_y = std(y);
```

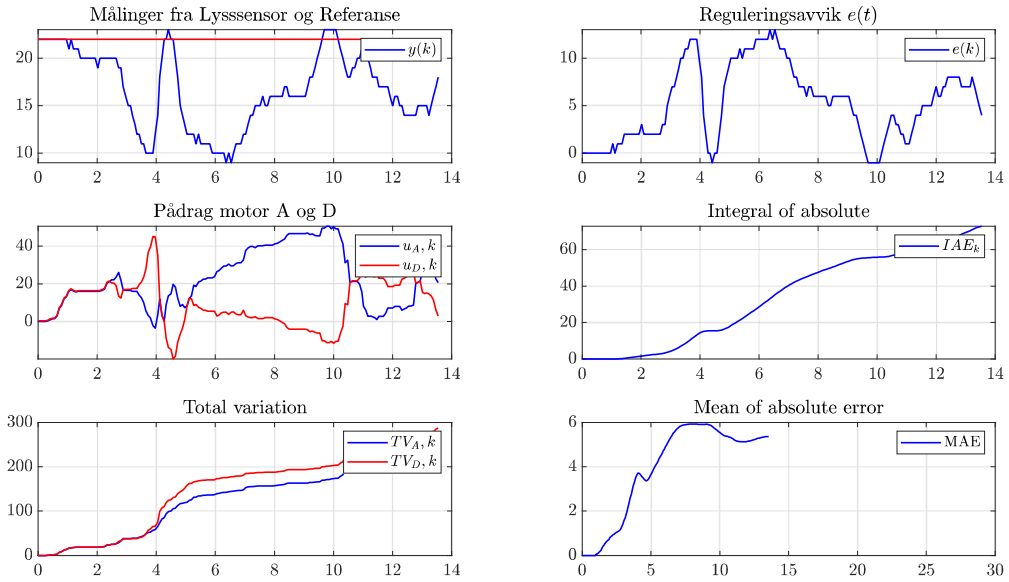
5.6 Resultat

Tabell 5.1: Sluttresultater fra manuell kjøring til deltagere i gruppe 2561. Den beste målingen er uthevd og det er satt en stjerne over det

	Abdirahman	Wijenthan
Plattform	Windows	Mac
Strømkilde	tilkoblet	tilkoblet
Samtidig plotting?	Ja	Nei
referanse $\{r_k\}$	26	22
middelverdi $\bar{y}$	20.63	**16.72***
$\lvert r - \bar{y} \rvert$	5.37	**5.28***
standardavvik σ	6.26	**3.98***
IAE	73.35	**72.87***
MAE	6.03	**5.36***
TV_A	307.12	**270.93***
TV_D	**279.70***	284.66
middelverdi av T_s [sek]	0.22	0.0786
kjøretid [sek]	12.36	13.55
antall målinger (k)	57	173

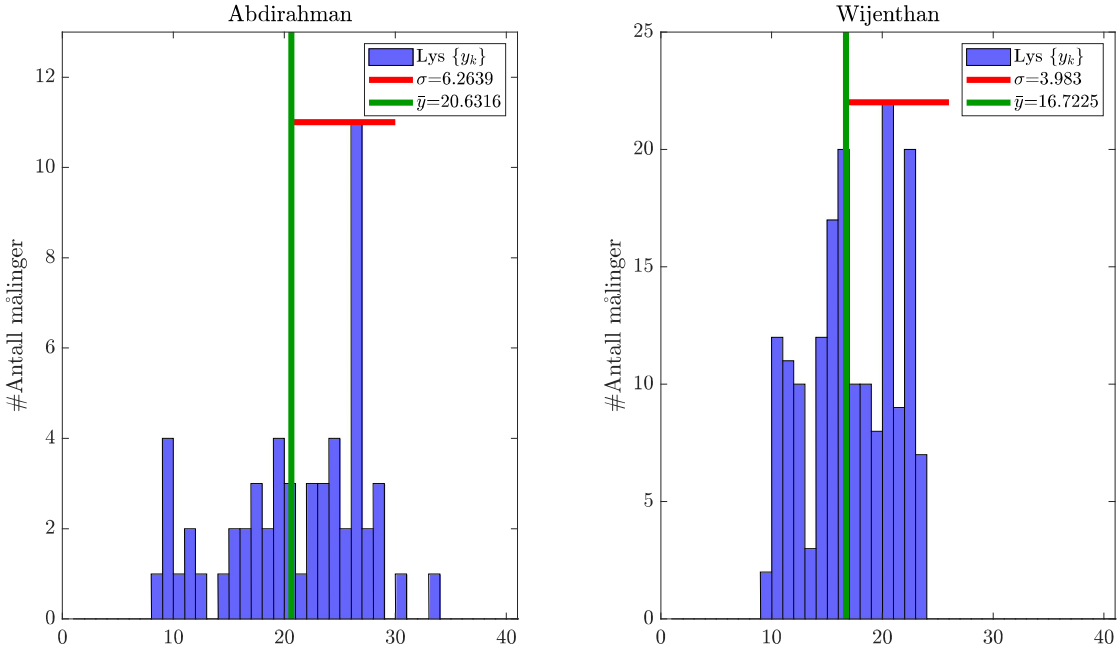
Resultatene fra kjøringen viser at Wijenthan oppnådde de beste målingene og dermed den beste kjøringen. Selv om kjøretiden hans var 1,19 sekunder lengre enn Abdirahmans, hadde han det laveste standardavviket, noe som indikerer tettere målinger, og dermed høyere presisjon. Wijenthan oppnådde også lavere verdier for både IAE og MAE. Differansen mellom middelverdien og referanseverdien er også lav, noe som indikerer at han fulgte referanseverdien bedre.

I figuren nedenfor viser vi til kvalitetsmål tatt under kjøringen til Wijenthan.



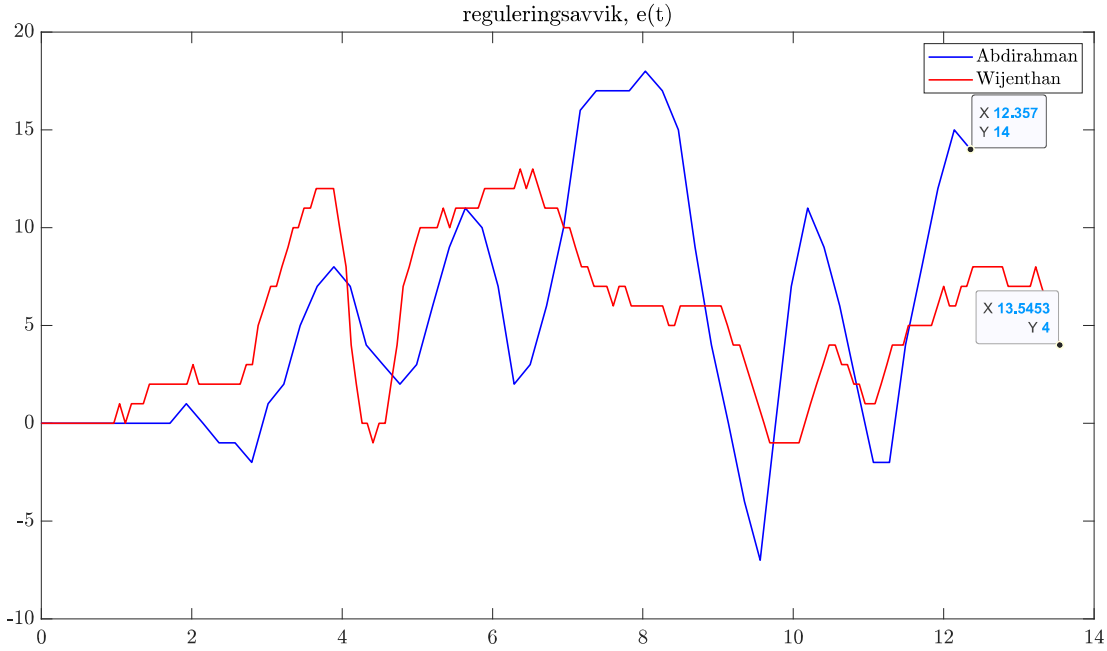
Figur 5.3: Figur som visualiserer målinger tatt under kjøring.

For å tydeligere illustrere resultatene som er diskutert, viser vi i figur 5.4 et histogram. Tettere søyler i histogrammet indikerer mer konsentrerte målinger. Dette fremhever at målingene til Wijenthan er mer konsentrerte enn målingene til Abdirahman, noe som resulterer i lavere standardavvik.



Figur 5.4: Histogram over målingene fra kjøringen til hvert gruppemedlem med standardavvik markert i rødt og middelverdi i grønt.

For å bekrefte yterligere at kjøringen til Wijenthan var bedre enn Abdirahmans, benytter vi reguleringsavviket fra begge kjøringene. Avvikene plotter vi i samme plott som det kommer frem i figur 5.5.



Figur 5.5: Figuren viser reguleringsavviket fra begge kjøringene, der avviket fra kjøringen til Abdirahm er i blått og avviket fra kjøringen til Wijenthan er i rødt.

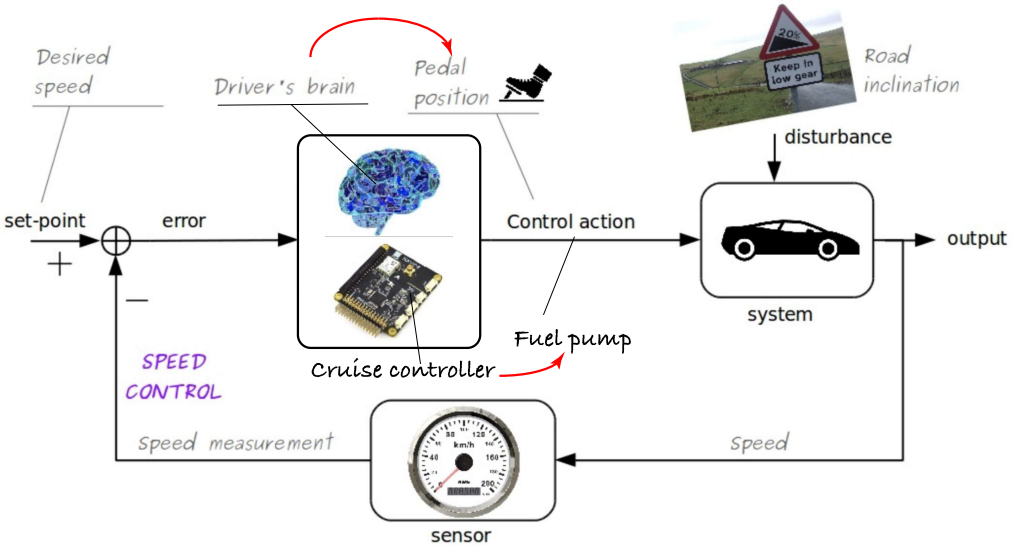
Som vi kan se i figuren, er det mindre avvik i Wijenthans kjøring i forhold til Abdirahmans kjøring. Dette tyder på at han har et mer stabilt kjøremønster. Det gir grunnlag for å si at hans kjøring er den mest presise og pålitelige sammenlignet med den andre, og dermed nærmest best.

Kapittel 6

Automatisk kjøring med PID (utført av Abdirahman og Wijenthan)

6.1 Introduksjon

I dette eksperimentet har vi gjennomført automatisk kjøring av roboten ved hjelp av de matematiske operasjonene i P-, I- og D-leddene, samt basispådraget vist i ligning 4.1. For å illustrere virkningen av PID-regulatoren kan vi sammenligne automatisk kjøring med manuell kjøring av roboten. Ved manuellkjøring er det hjernen vår som fungerer som regulator, øynene som måleinstrument og hånden som pådrag. Ved automatisk kjøring, blir utgangen $y(t)$, som vi ønsker å regulere, målt ved hjelp av en lyssensor. Målesignalet føres deretter tilbake og sammenlignes med referanseverdien ($r(1)$), og systemet blir justert etter dette. Figur 6.1 viser sammenhengen mellom manuellkjøring av en bil og det å bruke en cruisekontroll.



Figur 6.1: Figuren viser et blokkskjema som viser sammenhengen mellom manuellkjøring og automatisk kjøring med cruisekontroll. I manuellkjøring er det hjernen som tolker førerens ønsket hastighet og foten som regulerer pådraget, mens øynene fungerer som måleinstrument. Ved automatisk kjøring reguleres hastigheten ved hjelp av et måleinstrument og den målte hastigheten sammenlignes med referansen (ønsket hastighet), og systemet justeres etter dette. Figuren er hentet fra [1].

Den automatiske kjøringen av roboten gjennomføres på samme kjørebane som ved manuell kjøring. Resultatene kvalitetssikres på tilsvarende måte, og deretter sammenlignes de for å finne ut hvilken kjøring som var best.

6.2 Beregninger, kode og flytskjema

For å implementere koden for automatisk kjøring av LEGO-roboten, benytter vi pådraget fra PID-regulatoren som definert i ligning 4.1, og lyssensoren. På samme måte som ved manuell kjøring, lager vi en egen kode for pådraget til hver av de to motorene. Vi følger samme prinsipp her: motorene får likt pådrag ved kjøring forover/bakover, og motsatt pådrag ved svinging. Dette er illustrert i kodeutdrag 6.1.

Kode 6.1: Kode hentet fra `P06_autokjoring.m`

```

y(k)   = Lys(k);
r      = y(1);
e(k)   = r - y(k);

P(k) = Kp * e(k);
I(k) = I(k-1) + Ki * (e(k)+e(k-1))*T_s(k)/2;
D(k) = Kd * (e_f(k)-e_f(k-1)) / T_s(k);

PID(k) = P(k) + I(k) + D(k);
u_A(k) = u0 - PID(k);
u_D(k) = u0 + PID(k);

if I(k) > I_max
    I(k) = I_max;
elseif I(k) < I_min
    I(k) = I_min;
end
```

I koden har vi også lagt inn en integratorbegrensning på I-leddet for å forhindre oscillasjoner i reguleringen. Lego-roboten måler lysnivået ved hjelp av lyssensoren, og den første verdien som lyssensoren fanger opp er bestemt til å være referanseverdien $r(1)$. Ved å sammenligne løpende målinger, y_k med denne referansen avgjøres det om det reflekterte lyset er lavere eller høyere enn referanseverdien, og robotens bevegelser justeres deretter. PID-regulatoren styrer hvor mye hver motor må øke eller redusere hastigheten for å holde roboten innenfor kjørebanen. Nedenfor er det et bilde av et flytskjema som illustrer hvordan hele koden for automatisk kjøring av roboten er satt opp.

Figur 6.2: Flytskjema for koden som som brukes for automatisk kjøring av lego-robot med PID-regulering.

6.3 Resultat

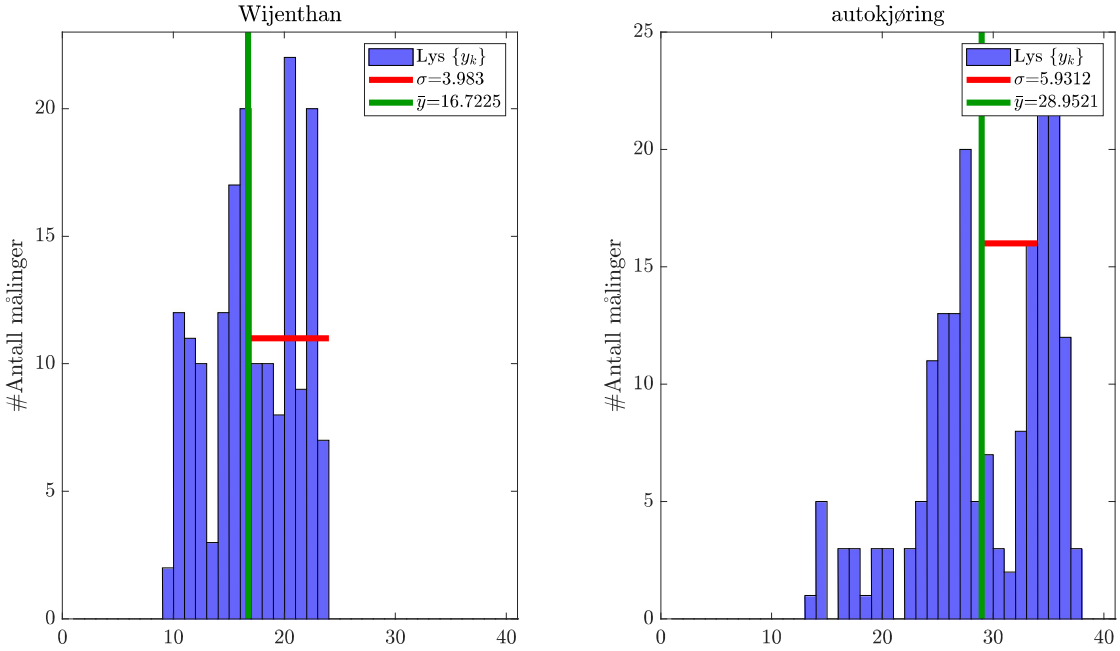
I resultatet sammenligner vi autokjøringen av roboten med det beste resultatet fra manuellkjøring. Sammenlikningen er gjort i tabellen nedenfor:

Tabell 6.1: Sluttresultater fra manuell kjøring av Lego-roboten og autokjøring av legoroboten. De beste målingene er uthevd og markert med stjerne.

	autokjøring	manuellkjøring
Plattform	Windows	Windows
Strømkilde	Tilkoblet	Tilkoblet
Samtidig plotting?	Nei	Nei
referanse $\{r_k\}$	27	22
middelverdi $\bar{y}$	**28.95***	16.72
$\lvert r - \bar{y} \rvert$	**1.95***	5.28
standardavvik σ	5.93	**3.98***
IAE	**63.63***	72.87
MAE	**5.18***	5.36
TV_A	480.73	**270.93***
TV_D	480.73	**284.66***
middelverdi av T_s [sek]	0.065	0.0786
kjøretid [sek]	12.26	13.55
antall målinger (k)	188	173

Tabellen viser at automatisk kjøring med PID-regulor ga det klart beste resultatet. Selv om Wijenthan oppnådde det laveste standardavviket, viser roboten betydelig bedre resultater på øvrige kvalitetsmål. Autokjøringen utmerker seg spesielt på mål som middelverdi, IAE og MAE. Differansen mellom referanseverdien og middelverdien er også lav, noe som indikerer at roboten fulgte referanseverdien svært godt.

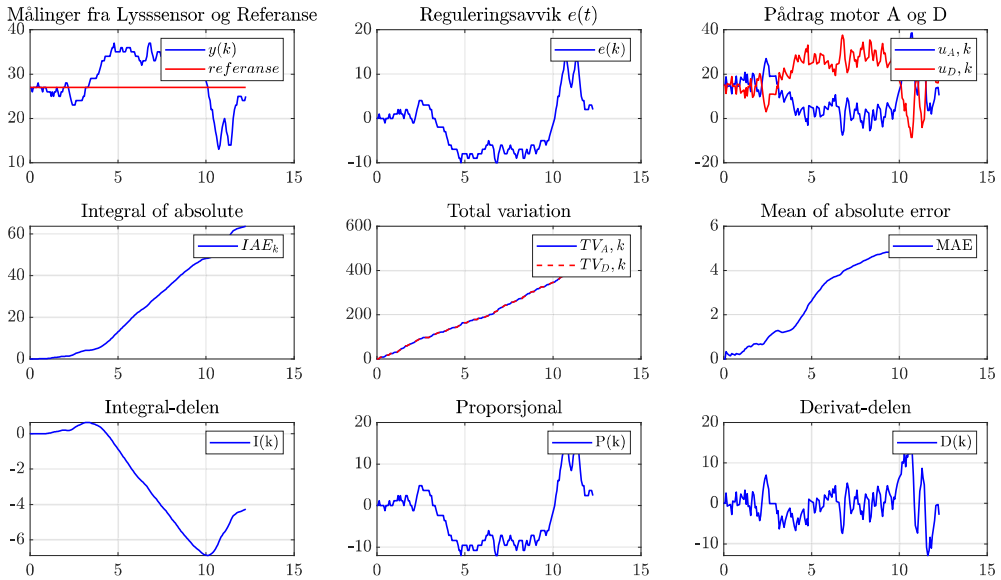
For å illustrere resultatet litt mer tydeligere, har vi lagd et histogram av målingene. Dette gjorde vi for å fremheve konsentrasjonen av lysmålingene $y(t)$ for å få et bedre bilde av hva resultatet i tabellen viser.



Figur 6.3: Histogram over målingene fra manuellkjøring og autokjøring sammen med standardavvik markert i rødt og middelverdi i grønt.

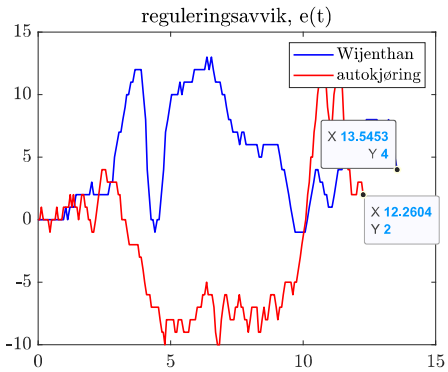
I histogrammene ser vi at målingene fra manuell kjøring er tettere samlet rundt middelverdien, og at de fleste verdiene ligger nær gjennomsnittet. For autokjøringen viser histogrammet en større spredning, der mange av målingene ligger lenger unna middelverdien. Dette tyder på mindre konsentrerte og mer varierende målinger, noe som fører til større standardavvik.

I figuren nedenfor presenterer vi kvalitetsmålene under automatisk kjøring av roboten. Figuren bekrefter at det vår konklusjon er riktig. Dette kan bemerkes ved å se på sluttresultatet for IAE og MAE.



Figur 6.4: Figuren viser målinger som er tatt under kjøringen.

For å yterligere verifisere resulatet har vi også plottet avvikene til begge kjøringene til sammenligning. Dette er vist i figur 6.5



Figur 6.5: Figuren viser reguleringsavviket for både autokjøring og manuell kjøring. Reguleringsavviket for autokjøringen er vist i rødt, mens avviket for Wijenthans kjøring er vist i blått.

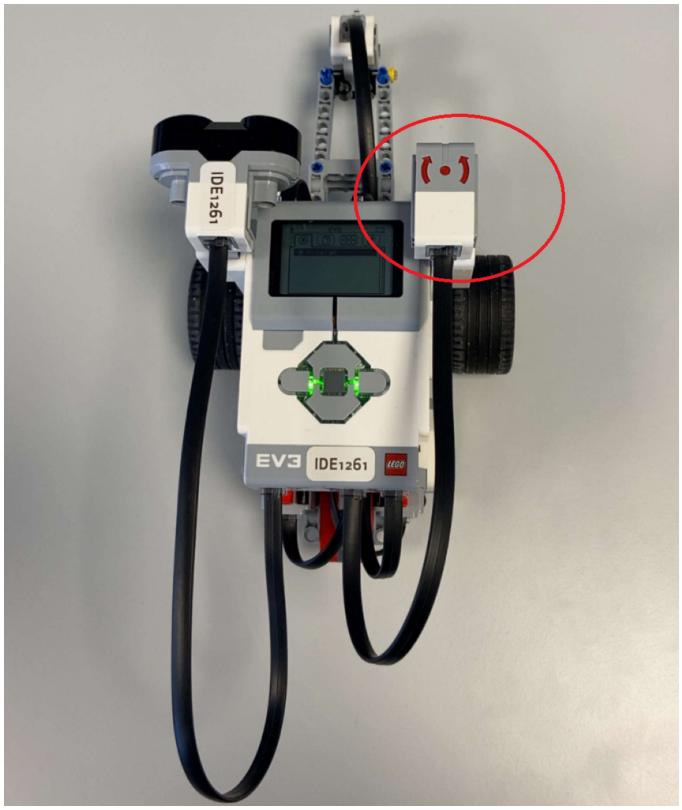
Ut fra figuren kan vi se at reguleringsavviket ved manuellkjøring har dobbelt så stort avvik som autokjøring. Dette påvirker målingene for IAE og MAE i den forstand at disse kvaltitetsmålene blir større under manuellkjøring av roboten. Dette bekrefter at vår metode for sammenligning og kvalitetssikring av de to kjøringene er pålitelig, og vi kan konkludere med at automatisk kjøring var best.

Kapittel 7

Støvsuger-robot

7.1 Introduksjon

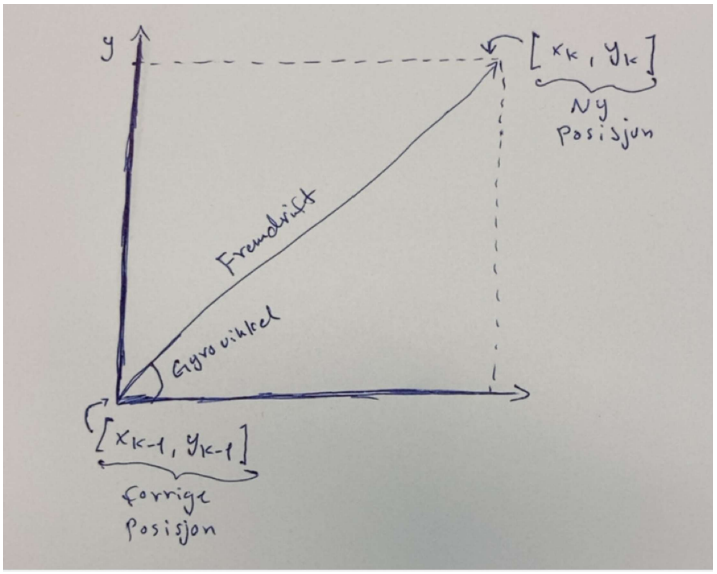
I dette prosjektet har vi bygget og programmert en LEGO-robot som fungerer som en robotstøvsuger, og som navigerer på et bord. Som en del av oppgaven har vi programmert roboten til å rygge og rotere et tilfeldig antall grader når den oppdager hindringer eller nærmer seg kanten av bordet. Til dette har vi benyttet både lyssensoren og ultralydsensoren og to motorer av samme type som i figur 4.2.I tillegg har vi kartlagt robotens kjørehistorikk ved hjelp av matematiske metoder og ved å anvende vinkelen som er funnet ved hjelp av gyrosensoren markert i rød ring i figur 7.1.



Figur 7.1: Figuren viser et bilde av roboten med tilkoblet utstyr, der gyrosensoren er markert med en rød ring.

7.1.1 fremgangsmåte

For å kartlegge kjøredataen til roboten har vi benyttet matematiske operasjoner som sinus og cosnus. I tillegg til dette har vi brukt Pytagoras' setningen til å finne robotens nåværende posisjon. Dette er illustrert i figur 7.2.



Figur 7.2: En skisse av hva prosjektet har gått ut på.

7.2 Beregninger og kode

7.2.1 Kode

For å implementere oppgaven i kode startet vi med å definere konstante hastigheter for robotens bevegelse forover og bakover. Disse hastighetene er lagret som variablene fremover og bakover i Matlab, som vist i kodesnutten under.

Kode 7.1: Kode hentet fra `P07_stovsuger.m`

```
fremover = 20;
bakover = -20;
```

Deretter bestemte vi verdier for når roboten skal rygge. Vi implementerte en timer som styrer hvor lenge roboten rygger før den begynner å rotere. Dette er illustrert i kodesnutten nedenfor.

Kode 7.2: Kode hentet fra `P07_stovsuger.m`

```
if Lys(k) <= 0 || s(k) < 0.5
pause(0.2);
rygge_tid = tic;

if online
    motorA.Speed = bakover;
    motorD.Speed = bakover;
    start(motorA);
    start(motorD);
end

while toc(rygge_tid) < 2
    pause(0.1);
end

if online
    stop(motorA);
    stop(motorD);
end
```

I kodeutdrag 7.2 kjører roboten bakover så lenge tiden fra timeren er mindre enn 2 sekunder. Deretter stopper roboten, før den begynner å rotere et tilfeldig antall grader. Dette er vist i kodeutdraget nedenfor.

Kode 7.3: Kode hentet fra `P07_stovsuger.m`

```
 retning(k) = round(rand());

if retning(k) == 1
    % høyre
    if online
        motorA.Speed = fremover;
        motorD.Speed = bakover;
        start(motorA);
        start(motorD);
    end
else
    % venstre
    if online
        motorA.Speed = bakover;
        motorD.Speed = fremover;
        start(motorA);
        start(motorD);
    end
end

```

```
    rotasjonstid = tic;

    while toc(rotasjonstid) < (1 + rand() * 5.5)
        pause(0.01);
    end

    if online
        stop(motorA);
        stop(motorD);
        motorA.Speed = fremover;
        motorD.Speed = fremover;
        start(motorA);
        start(motorD);
    end
```

I kodeutdrag 7.3 velger MATLAB en tilfeldig retning, der høyre tilsvarer verdien 1 og venstre verdien 0. Dersom det tilfeldige tallet er 1, roterer roboten ved at motorA får positiv hastighet og motorD får negativ hastighet. Hvis retningen er 0, skjer det motsatte: motorA får negativ hastighet og motorD får positiv hastighet.

For at roboten skal rotere et tilfeldig antall grader, har vi også benyttet en timer. Først målte vi hvor lang tid roboten brukte på å rotere en full sirkel (360°), som viste seg å være omtrent 6.5 sekunder. Basert på dette velger programmet et tilfeldig tall mellom 1 og 6.5, som deretter bestemmer hvor lenge roboten roterer.

7.2.2 Beregninger

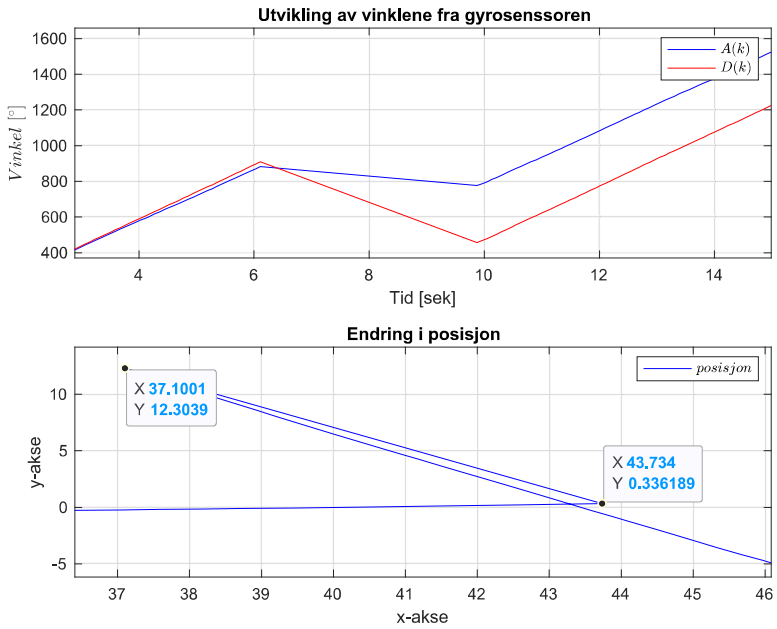
For å beregne posisjonen til roboten har vi først funnet den lineære rotasjonsendringen til hver motor. Rotasjonsendringene ble beregnet ved å ta differansen mellom motorens nåværende og forrige vinkelposisjon. Denne differansen delte vi på 360 for å finne antall hele omdreininger. Deretter multipliserte vi dette med hjulenes omkrets for å finne den tilbakelagte avstanden. For å bestemme fremdriften, som vist i figur 7.2, tok vi gjennomsnittet av den rotasjonsendringen for de to motorene. Ved å benytte fremdriften og vinkelen som er målt med gyrosensoren, markert med rød ring i figur 7.1, kan vi beregne robotens nåværende posisjon. For å finne robotens posisjon i x-retning bruker vi *cosnus*, mens for y-retningen benytter vi *sinus*. Kodeutdrag 7.4 viser hvordan vi har implementert dette i kode.

Kode 7.4: Kode hentet fra `P07_stovsuger.m`

```
    d = 5.6; % diameter i cm
```

```
    OmkretsHjul = pi*d; % i cm;

    if k > 1

        rotasjonsendringA = (VinkelPosMotorA(k) - ...
            VinkelPosMotorA(k-1))/360 * OmkretsHjul; % i cm;
        rotasjonsendringD = (VinkelPosMotorD(k) - ...
            VinkelPosMotorD(k-1))/ 360 * OmkretsHjul; % i cm;

        fremdrift = (rotasjonsendringA + rotasjonsendringD)/2; ...
            %fremdriften til hele roboten (snittet)


        x(k) = x(k-1) + fremdrift * cos(vinkel(k)/180*pi); ...
            %endring i posisjon i x-retning i cm
        y(k) = y(k-1) + fremdrift * sin(vinkel(k)/180*pi);  ...
            %endring i posisjon i y-retning

    end
```

7.3 Resultat



Figur 7.3: Figuren viser vinkelposisjonen til motorA og motorD i øverste subplot. I nederste subplot viser figuren posisjonsendringen til motoren i x og y-retning i centimeter.

Figuren viser to grafer som beskriver robotens bevegelse. Den første grafen viser vinkelposisjonen til hver motor over tid, mens den andre viser robotens posisjon i x og y-retning. I den øverste figuren ser vi at motorA (blå) og motorD (rød) øker jevnt fra 0 til rundt 6 sekunder, noe som indikerer at begge hjulene kjører fremover med nesten lik hastighet. Fra 6-8 sekunder flater kurven ut og synker litt, noe som viser at roboten rygger. Deretter splitter kurvene seg, en stiger og en synker, noe som betyr at roboten roterer. Etter 14 sekunder samkjøres motorene igjen med positiv stigning, og roboten kjører rett frem i sin nye retning.

Figur 2 viser posisjonsendring i xy-planet. Roboten starter i $(0, 0)$. Den rette linjen langs x-aksen representerer fremoverkjøring. Ved tidspunktet 6 sekunder er posisjonen $(43.13\text{cm}, 0.27\text{cm})$, mens ved tidspunktet 8 sekunder er den $(37.10\text{cm}, 12.30\text{cm})$. Linjen mellom disse punktene bekrefter at roboten har rygget i dette tidsintervallet, i samsvar med informasjonen fra figur 1.

Kapittel 8

Konklusjon

I dette prosjektet har vi gjennomført en rekke øvelser innen numerisk integrasjon, filtrering, numerisk derivasjon, og regulering. Vi startet med å utføre prosjektet i python. Vi tenkte det var en bedre ide i og med at vi går data, og python er et programmeringsspråk vi kommer til å bruke ofte, og mer enn Matlab. Vi møtte på flere utfordringer på starten, og bestemte oss for å bytte til Matlab.

De obligatoriske oppgavene gjorde vi sammen. Vi kjørte kodene på begge pcene for å sjekke om det fungerte. Vi valgte to prosjekter på den kreative delen siden det var det vi mente var fornuftig, med tanke på at vi bare er to i gruppa.

Prosjektet var litt krevende, fordi vi var bare to stykker på gruppa. Det ble brukt mye tid på å gjøre målingene og skrive det inn i latex. Men det var veldig lærerikt og prosjektet formidlet lærestoffet på en gøy måte. Det var mye enklere å forstå stoffet når man jobbet praktisk med det. Vi har kombinert teori og praksis anvendelse ved hjelp av lego mindstorms og Matlab.

Bibliografi

[1] T. Drengstig. Prosjektbeskrivelse. Lego Mindstorms og MATLAB/Python i skjønn forening, 2025. Utdelt materiale.

[2] OpenAI. Chatgpt (30. april versjon) [stor språkmodell]. `https://chat.openai.com/`, 2025. Tilgjengelig fra OpenAI.

Vedlegg A

Timelister

Tabell A.1: Timelister for gruppe 2561 i faget ELE130.

Navn/uke	Abdirahman	Wijenthan
12	23.75	23
13	24	24
14	14	14
15	25.25	22
16	27	23
17	21	16
18	29.5	26
Sum	164.5	148