

PROSJEKT - HØSTEN 2020

Prosjekt- oppgaven	LEGO Mindstorms og MATLAB; anvendt matematikk/ fysikk og programmering i skjønn forening
-----------------------	---

Gruppenummer	2067		
Gruppens medlemmer	Navn	Studentnummer	Bilde
	Marius Bjørnøy	260707	
	Thomas Forberg Haugvaldstad	247760	

Innhold

Innhold	ii
Sammendrag	viii
1 Numerisk integrasjon (utført av Marius)	1
1.1 Problemstilling	2
1.2 Teori	3
1.2.1 Lyssensor	3
1.2.2 Generell integrasjon	3
1.2.3 Nummerisk integrasjon	4
1.2.4 Trapesmetoden	5
1.3 Forslag til løsning	6
1.3.1 Funksjonen	6
1.3.2 Funksjonen	7
1.3.3 forslag til løsning sinuskurve	8

INNHOLD

1.4 Resultat	9
1.4.1 Integrasjon av en konstant	9
1.4.2 Integrasjon av et sinussignal	11
2 Filtrering (utført av hele gruppen)	15
2.1 Problemstilling	15
2.2 Løsning til FIR-filter	16
2.2.1 Kode	17
2.2.2 Resultat	18
2.3 IIR-filter	21
2.3.1 Kode	22
2.3.2 Resultat	23
2.4 Sammenligning	27
2.4.1 FIR-filter	27
2.4.2 IIR-filter	28
3 Numerisk derivasjon (utført av Thomas og Marius)	29
3.1 Problemstilling	31
3.2 Teori	31
3.2.1 Derivasjon definisjon	31
3.2.2 Praktisk bruk av numerisk derivasjon	32

INNHOLD

3.3	Forslag til Løsning	34
3.3.1	Forslag til løsning av et lineært signal	34
3.3.2	Forslag til løsning av et sinussignal	35
3.3.3	Funksjonen	37
3.4	Resultat	37
3.4.1	Derivasjon av et lineært signal	37
3.4.2	Derivasjon av et sinussignal	39
4	Manuell kjøring av Lego-robot (utført av hele gruppen)	42
4.1	Problemstilling	44
4.2	Sensorer og motorer	44
4.2.1	Motorer	44
4.2.2	Lyssensor	44
4.2.3	<i>Joystick</i>	44
4.3	Kvalitetsmål	45
4.3.1	Kvalitetsmål struktur	45
4.3.2	Avvik	45
4.3.3	Kvalitetsmål 1: IAE og MAE	46
4.3.4	Kvalitetsmål 2: Pådrag	48
4.3.5	Kvalitetsmål 3: Standardverdi og Middelverdi	49
4.4	Resultat	50

INNHOLD

5 Kreativ del Store prosjekter	55
5.1 PID kjøring	55
5.1.1 Problemstilling	55
5.1.2 Teori	55
5.1.3 Løsningsforslag	58
5.1.4 Resultat	60
5.2 Cruise Controll	65
5.2.1 Problemstilling	65
5.2.2 Teori	65
5.2.3 Løsningsforslag	66
5.2.4 Resultat	67
5.3 Sykkelcomputer	70
5.3.1 Problemstilling	70
5.3.2 Teori	71
5.3.3 Løsningsforslag	72
5.3.4 Resultat	73
6 Kreativ del Små prosjekter	75
6.1 Fart	75
6.1.1 Problemstilling	75
6.1.2 Teori	75

INNHOLD

6.1.3	Løsningsforslag	78
6.1.4	Resultat	80
6.2	Trapes metoden og <i>EulerForward</i> metoden	83
6.2.1	Problemstilling	83
6.2.2	Teori	83
6.2.3	Løsningsforslag	85
6.2.4	Resultat	85
7	Konklusjon	91
	Bibliografi	92
	Vedlegg	92
A	Timelister	93
B	Programlisting prosjekt 04	94
B.1	P04_F4_MathCalculations.m	94
B.2	P04_F5_CalculateAndSetMotorPower.m	95
C	Programlisting prosjekt 05	97
C.1	P05_F4_MathCalculations.m	97
C.2	P05_F5_CalculateAndSetMotorPower.m	98

INNHOLD

D Programlisting prosjekt 06	100
D.1 P06_F4_MathCalculations.m	100
D.2 P06_F5_CalculateAndSetMotorPower.m	101
 E Programlisting prosjekt 07	 103
E.1 P07_F4_MathCalculations.m	103
E.2 P07_F5_CalculateAndSetMotorPower.m	104
 F Programlisting prosjekt 08	 105
F.1 P08_F4_MathCalculations.m	105
F.2 P08_F5_CalculateAndSetMotorPower.m	106
 G Programlisting prosjekt 09	 108
G.1 P09_F4_MathCalculations.m	108
G.2 P09_F5_CalculateAndSetMotorPower.m	108

Sammendrag

I denne rapporten vil vi gå gjennom praktisk bruk av matematikk ved hjelp av en rekke sensorer. Vi skal for det meste ta for oss numerisk integrasjon, numerisk derivasjon og filtrering. Dette ønsker vi å vise ved simulering av dagligdagsbruk, og bruke disse matematisk utregningene til å utføre masse ulike funksjoner.

Vi kommer til å se på hvordan praktisk matte differensierer seg fra teoretisk matte. Dette kommer vi til å demonstrere med en rekke utregninger og målinger. Vi kommer også til å verifisere at funksjonene stemmer ved hjelp av generell matte kunskap.

Til slutt kommer vi til å ta i bruk det vi har lært gjennom utføringen av praktisk matematikk, til å gjennomføre prosjekter som *cruise controll* og full automatisk kjøring.

1.1 Problemstilling

Kapittel 1

Numerisk integrasjon (utført av Marius)

1.1 Problemstilling



Figur 1.1: Figuren ovenfor viser måleinstrumentet vårt. Den største maskinen med skjerm på i øverste bilde er en EV3. Dette er maskinen som plukker opp målingene våre. Denne mottar disse målingene fra sensorene. Til venstre nedenfor er det en lyssensor. I midten under er det en gyro og på siden til høyre er en ultrasonisksensor. Vi kommer til å gå mer i dybden om hvordan disse virker senere i denne rapporten.

1.2 Teori

Numerisk integrasjon er noe som skjer overalt uten at vi er klar over det. Det er lett å bygge opp en ren frustrasjon mens du sitter i mat100 og lærer om alle ulike måter å integrere på. Du lurte på når og hvor skal jeg få bruk for dette. I dette kapitellet skal vi gå gjennom praktisk bruk av integrasjon og nettopp hvorfor det er så viktig.

I dette kapitelet er målet å lage en funksjon ved å integrere ulike målinger fra EV3'en. Vi skal gjøre dette ved å simulere drikking ved bruk av en lyssensor, hvor vi først tar utgangspunkt i integrasjon av en konstant i del en, og ved å integrere en sinuskurve i del to. I dette kapittelet vil vi referere lys integrert som volum $V(t)$

1.2 Teori

1.2.1 Lyssensor

I denne delen av forsøket blir kun lyssensoren tatt i bruk. Denne måler reflektert lys. Lyssensoren får en høyere verdi dersom underflaten måler mye reflektert lys. Det vil si at på en sort underflate vil verdien være lavere enn på en hvit underflate, ettersom en svart farge reflekterer mindre lys en hvit farge. Dette ser du tydeligere på målingene fra figur 1.6, og rekkefølgen sensoren ble plassert i ser man i figur 1.4

1.2.2 Generell integrasjon

Målet med integrasjon er å regne ut arealet av en funksjon.

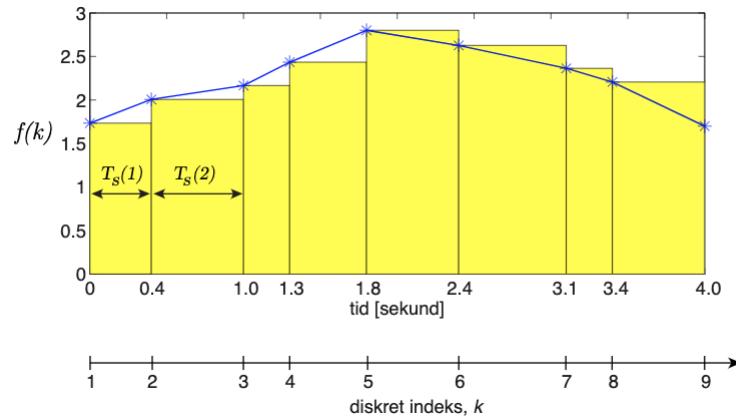
$$\int_a^b x dx = \frac{1}{2}b^2 - \frac{1}{2}a^2 \quad (1.1)$$

hvor a og b er grenseverdiene på x aksen i et bestemt integral. Problemet ved bruk av denne formelen er at det ikke er mulig å regne ut arealet i sanntid.

1.2 Teori

1.2.3 Nummerisk integrasjon

Eulers forovermetode er prinsippet hvor vi deler opp grafen i mange rektangler. Rektanglene går langs x aksen, og måler differansen mellom $X_1 - X_0$ også kjent som ΔX , multiplisert med høyden til rektanglene, som er definert av y-aksen. som vist i figuren under.



Figur 1.2: [3]Prinsippet for numerisk integrasjon av tidseriedata. For hver ny måling $f(k)$ legges det inn et rektangel med bredde $T_s(k - 1)$, og totalarealet av rektanglene tilsvarer integralverdien. Figuren er hentet fra [3].

Problemet nummerisk integrasjon er at dersom antall n rektangler ikke er uendelig vil man kun få et estimat av arealet. Hvis vi studerer figuren ovenfor ser vi at figuren ikke følger den blå linjen. Dette er siden vi bare antar at målingene er den samme til vi mottar en ny y verdi. Selv om dette er en ulempe, gjør det integrasjon i sanntid mye lettere. Nå kan vi regne ut arealet av grafene, det gjøres ved å lage en økende tidskonstant og multiplisere denne konstanten med y-verden.

Ev3-en gir oss målinger som vi kan plotte, men det er viktig å være obs på at hver måling ikke alltid kommer på samme tidspunkt. Sekundene kan variere mellom hver måling. For å gjøre denne prosessen lettere kaller vi differansen mellom hver måling for T_s , tidsskrittet mellom hver måling. Dette blir illustrert bedre i bildet ovenfor.

Vi bruker også k konstanten, som er en konstant som øker med 1 mellom hver måling. Det vil si at verdien $k - 1$ er målingen før den nåværende. Bruken av denne konstanten gir oss muligheten til å summere tidligere integral sammen med

1.2 Teori

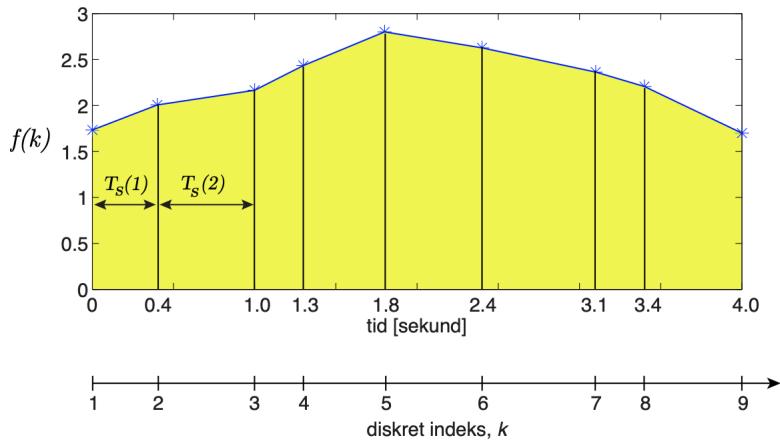
nåværende målinger. Vi kan også finne gamle målinger ved å gi k konstanten den x -verdien man ønsker å finne.

$$Volum(k) = Volum(k-1) + FunksjonsVerdi(k-1) \cdot TidsSkrift(k-1) \quad (1.2)$$

Volumet blir beregnet av den tidligere volumverdien vår som blir summert med arealet av den nye verdien vår. Den nye verdien består av tid multiplisert med måling.

1.2.4 Trapesmetoden

Trapesmetoden fyller opp større deler av grafen enn det Eulerforovermetoden gjør. Måten den virker på er at den tar utgangspunkt i en den første y -verdien til det neste trapeset. På den måten får du en lineær vekst for hvert enkelt trapes. Det kan ses på som å addere eller subtrahere en liten rettvinklet trekant på toppen av rektangelet fra Eulersforover metode. Dette blir bedre illustrert i grafen under.



Figur 1.3: [3] Dette er samme graf som i figur(1.2), men som man tydelig kan se er arealet mye nærmere den blå linjen. Dette fører til et mer nøyaktig areal.

Ulempen med denne metoden er at den er mer tungvindt å gjennomføre i lengden, noe som kan føre til en større differanse mellom k verdiene. Dette igjen vil påvirke

1.3 Forslag til løsning

styringen av roboten, siden ventetiden på dataoverføringen blir større. Derfor kan det være lurt å ikke gjennom føre trapes metoden i sanntid, men heller etterpå for å få et enda mer nøyaktig integral.

Formelen for trapesmetoden er:

$$IntValueNew = IntValueOld + (\text{sum}(FunctionValue)/2) \cdot \text{timestep} \quad (1.3)$$

1.3 Forslag til løsning

1.3.1 Funksjonen

Løsningsforslaget til oppgaven ligger i EulerForward som igjen baserer seg på ligningen 1.1. Likningen tar alle argumentene og summerer dem opp til et totalt areal.

Den nye integral verdien er gitt av det tidligere integralet, addert med lysverdien som er multiplisert med den diskre indeksen differansen av k og $k - 1$. På denne måten klarer vi å regne ut integralet i sanntid ved bruk av en diskre konstant.

Arealet av rektangelet under grafen blir da lettere beskrevet som timestep multiplisert med FunctionValue, hvor timestep er bredden og FunctionValue er høyden. Når vi multipliserer disse variablene vil vi få et rektangulært areal. Denne verdien blir deretter summert sammen med de tidligere arealene. Fordelen med dette er at det blir mindre tungvindt å regne arealet på, enn å lagre hvert eneste rektangel. På denne måten minimerer vi tidsskrittet mellom hver k verdi, og vil minimere ventetiden før EV3-en mottar signal fra joysticken.

Funksjonen nedenfor har to virkninger, den ene er EulerForward metoden og den andre metoden baserer seg på trapes metoden.

1.3 Forslag til løsning

1.3.2 Funksjonen

kode(1.1)

```
1 function [IntValueNew] = EulerForward(IntValueOld ...
    ,FunctionValue, TimeStep)
2 IntValueNew = IntValueOld + (FunctionValue * TimeStep);
3 end
```

Verifisering av funksjonen

I del en av oppgaven skal vi ta integralet av en konstant. $f(t) = [Cl/s]$. For å finne integralet regner vi alt fra $x = 0$ til $x = t$. Konstanten a er flow-raten og dette integralet blir da volumet i denne simuleringen. formelen blir da:

$$\int_0^t adt = a \cdot t + C \quad (1.4)$$

For å bekrefte dette lar vi konstanten a være lik $lys(k)$, og setter at tiden T_s er den diskre konstanten k . a kan være positiv, negativ og lik null.

For å teste dette praktisk ble det ført en lyssensor over ett sort hvit spektrum. Dette gjør vi for å simulere drikking og fylling i ett glass. mer refleksjon er tilsvarende fylling av glasset og lite refleksjon blir da drikking av glasset. For å kunne regne ut endring i refleksjon lar vi a være lik $Flow(k)$ subtrahert med $nullflow$. $nullflow$ er $Flow(1)$. Formelen blir da:

$$Flow(k) = lys(k) - nullflow \quad (1.5)$$

1.3 Forslag til løsning



Figur 1.4: [3](markert egne tall på)Sort hvit spekteret brukt. De røde tallene indikerer rekkefølgen over hvilken rekkefølge lyssensoren ble plassert. Det vil si at a er 0 når lyssensoren er på det midterste feltet.

Vi setter her at lysavvik a er 0 når lyssensoren befinner seg i felt 50 fra bilde ovenfor. Da er $flow(1)$ lik lysverdien vi mottar mens lyssensoren befinner seg i midterste felt.

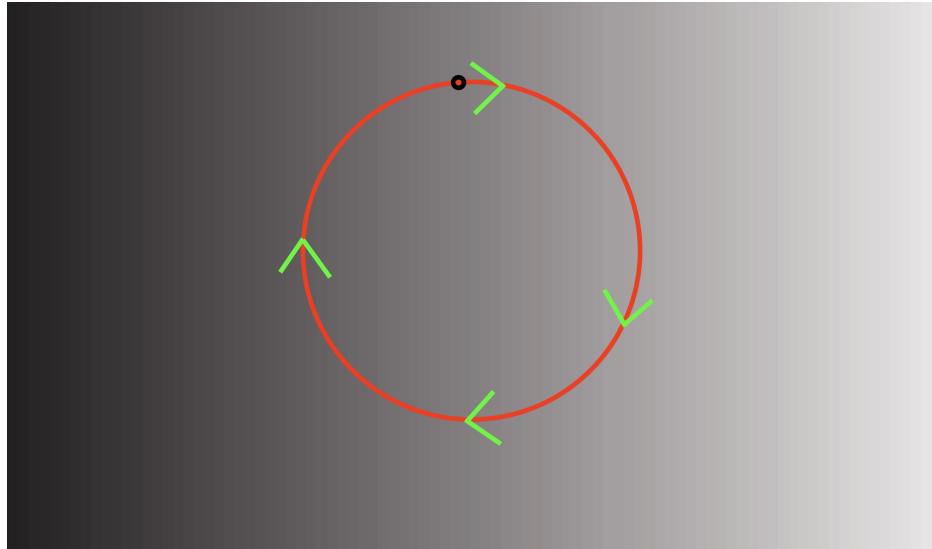
1.3.3 forslag til løsning sinuskurve

I del to av dette kapitelet tar vi for oss ubestemt integrasjon av en sinuskurve. I denne delen blir vi nødt til å bruke både en konstant a og frekvensen ω . Formelen for sinuskurven brukt i dette forsøket er vist under.

$$\int a * \sin(\omega * t) + ddt \quad (1.6)$$

Den beste måten å få en perfekt sinuskurve på er ved å rottere lyssensoren i en sirkulær bevegelse. Dette ble gjennomført ved å plasser avstands klosser foran lyssensoren for å ha konstant avstand mellom ark og sensoren. For å teste ulike kurver ble det gjennomført flere tester på ulike punkter av gråskala arket. Vi gjennomførte ulik fart på kurvene, og testet med ulike radius blant dem. Ved å gjennomføre disse testene, og ved å endre på amplituden kunne vi se sammenhengen mellom amplituden og frekvensen ω .

1.4 Resultat



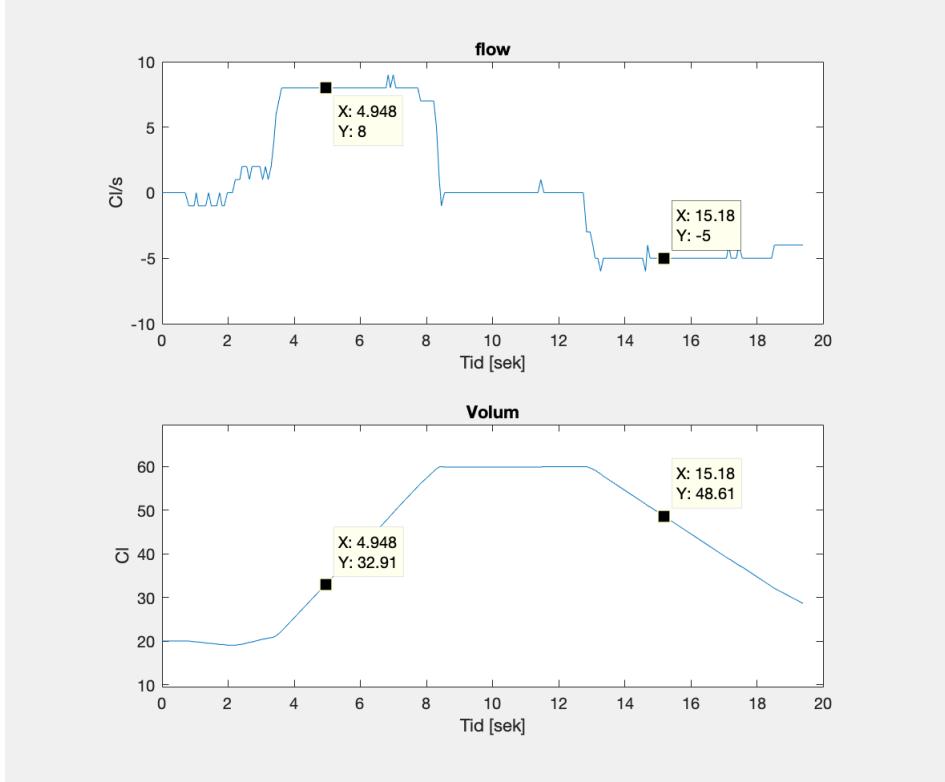
Figur 1.5: Gråskalaen vist ovenfor viser hvordan lyssensoren plukket opp informasjon. Den røde formen viser formen sensoren beveget seg i, hvor de grønne pilene viser retning. Det lille sorte punkt viser startpunktet til sensoren.

1.4 Resultat

1.4.1 Integrasjon av en konstant

Verifiseringen av integrasjonen er gitt ved ligningen... og er vist i figur1.6. Den øverste funksjonen er flow raten altså Centi liter per sekund, hvor den nederste grafen viser volum altså Centi liter. Begge grafene er målt mellom $t = 4.948$ og $t = 15.18$

1.4 Resultat



Figur 1.6: [3]Figur av graf etter numerisk integrasjon av en konstant. konseptet er hentet fra ligning...

Siden a er en konstant for lysavvik, og lysavviket varierer må vi dele opp integralet. $a_1 = 8$ som vist i grafen 1.6. Det vil si at lysavviket er positivt. $a_2 = 0$ i tidsperioden mellom 8.3 og 12.8 det vil si at volumet forblir konstant. mellom 12.8 og 15.2 er $a_3 = -5$ det vil si at lysavviket er negativt. For å løse dette tar vi å multipliseringer a_3 utrykket med -1 .

$$\text{Integralet} = \int_{4.9}^{8.3} a_1 dt + \int_{8.3}^{12.8} a_2 dt - \int_{12.8}^{15.2} a_3 dt \quad (1.7)$$

$$\begin{aligned}
 &= (8 * 8.3 - 8 * 4.9) + (0 * 12.8 - 0 * 8.3) - ((-5) * 15.2 - (-5) * 12.8) \\
 &= (66.4 - 39.2) + 0 - (-64 - (-76)) \\
 &= 27.2 - 12 = 15.2
 \end{aligned}$$

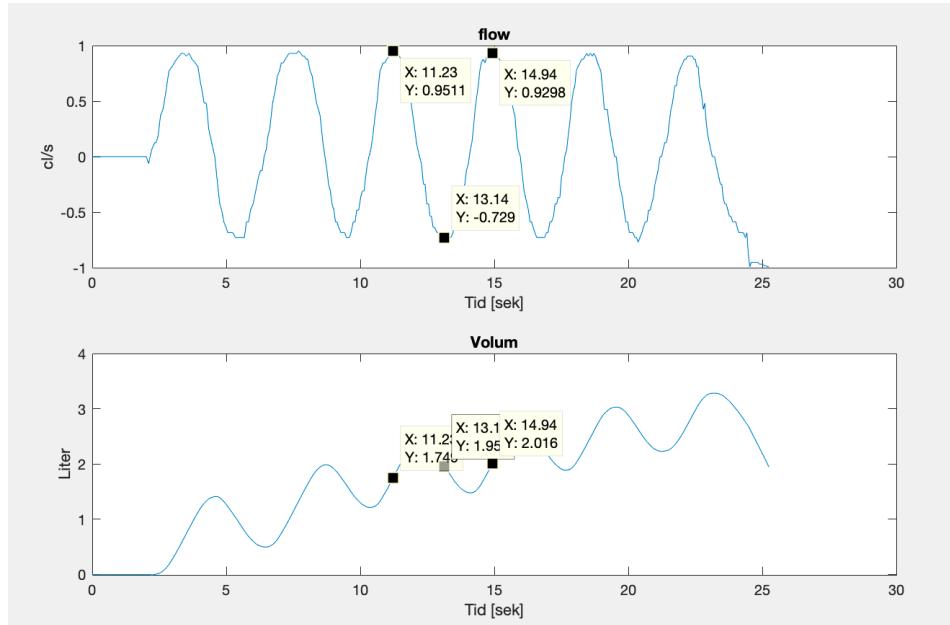
1.4 Resultat

$$15.2 + 32.9 = 48.1$$

Årsaken til at vi adderer med 32 på slutten, er siden det er utgangspunktet vårt, altså volumet vårt før vi har integrert, for å bekrefte at funksjonen stemmer. Som man kan se fra utregningene ovenfor kom vi nærmere, men ikke helt til 48.61, årsaken til dette er mange faktorer. Støy er en liten faktor, de små punktene som ikke følger den regelmessige verdien. Dette vil derfor ha en minimal effekt på grafen. En annen faktor er avrunding, hvor vi rundet av til nærmeste tidels plass i utregningen vår. Den største faktoren er at volumet i grafen fra matlab har brukt Eulersforover metode for å løse likningen.

1.4.2 Integrasjon av et sinussignal

Sinusfiguren under viser resultatet av lyssensoren som ble beveget i sirkler rundt gråskala arket. For å finne volumet av flowraten blir vi nødt til å integrere dette utrykket også. For å verifisere grafen må vi ta et sinusintegral.



Figur 1.7: Sinus kurve, og numerisk integrert resultat, fra ligning... Den er stigende siden grafen befinner seg lengre over 0 enn under. Når flowraten er 0 vil volumet være konstant stabilt.

1.4 Resultat

Formelen på Volumet blir da:

$$\int a \cdot \sin(w \cdot t) + dt = -\frac{a}{\omega} \cdot \cos(w \cdot t) + d \cdot t \quad (1.8)$$

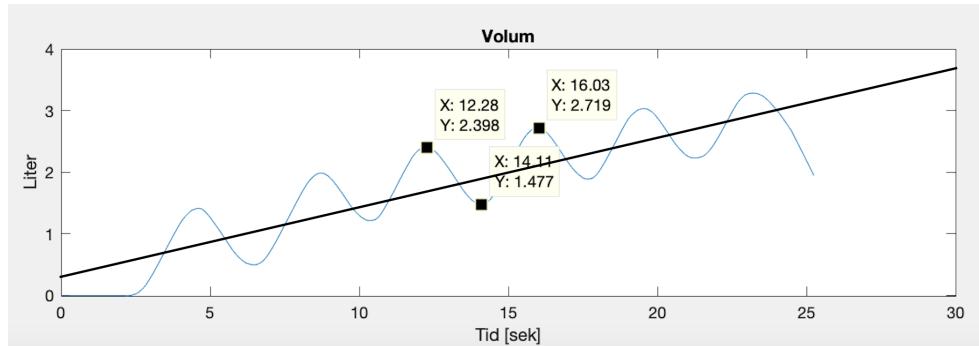
Finner deretter amplituden til sinusgrafen:

$$a_{\sin} = \frac{0.95 - (-0.73)}{2} = 0.84 \quad (1.9)$$

Vinkelfart ω er gitt ved ligningen:

$$\omega = \frac{2\pi}{T} = \frac{2\pi}{14.9 - 11.2} = \frac{2\pi}{3.7} = 1.7 \quad (1.10)$$

Hvor T er avstanden mellom to amplituder.



Figur 1.8: Siden grafen øker finner man en form for stigningstallet ved å ta to topp amplituder. Linjen i midten forteller den lineære veksten til grafen. Den er ikke alltid midt i cosinus funksjonen. Dette skyldes ikke konstante lysverdier, og Eulersforovermetode er ikke helt likt det reelle integralet.

Det vil si at vi blir nødt til å finne stigningstallet til den sorte linjen i grafen. En måte å gjøre dette på er ved å regne ut den lineære veksten gitt:

$$y = a \cdot x + b \quad (1.11)$$

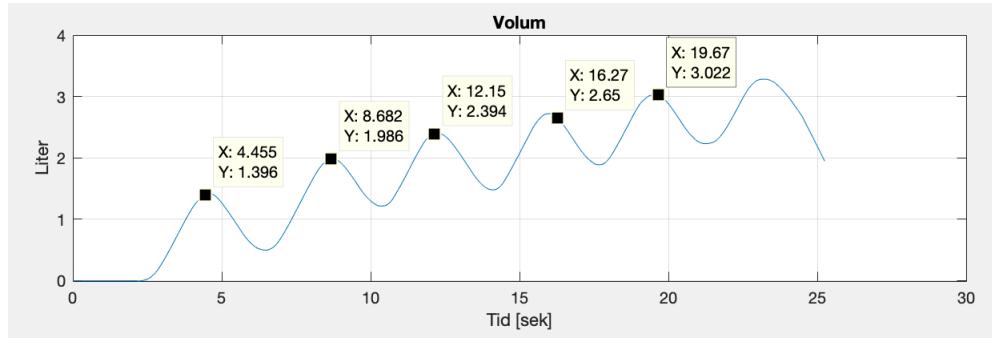
Stigningstallet er gitt ved:

1.4 Resultat

$$d = \frac{\Delta y}{\Delta x} = \frac{3.02 - 1.4}{19.67 - 4.45} = \frac{1.62}{15.22} = 0.11 \quad (1.12)$$

En annen måte å finne denne konstanten på er ved å finne likevektslinjen til sinusfunksjonen.

$$d = \frac{f_{maks} + f_{min}}{2} = \frac{0.95 - 0.73}{2} = 0.11 \quad (1.13)$$



Figur 1.9: Topp punkt på Cosinus kurven.

Så regner vi amplituden likt som i sinus kurven:

$$a_{cos} = \frac{2.4 - 1.5}{2} = 0.45 \quad (1.14)$$

Amplituden til cosinus signalet blir:

$$a_{cos} = \frac{a_{sin}}{\omega} \approx 0.5 \quad (1.15)$$

Siden Grafen vår er forkjøvet og begynner ikke før $x = 2.01$ må vi subtrahere grenseverdiene våre med 2.01. Det vil si at de nye grenseverdiene våre går fra 14.94 og 11.23 til 12.93 og 9.22.

$$\int_{9.22}^{12.93} 0.825 \cdot \sin(1.7 \cdot t) + 0.11 dt = -\frac{0.825}{1.7} \cos(1.7 \cdot t) + 0.11t \Big|_{9.22}^{12.93} \quad (1.16)$$

1.4 Resultat

$$-0.48\cos(1.7 \cdot 12.93) + 0.11 \cdot 12.93 - (-0.48\cos(1.7 \cdot 9.22) + 0.11 \cdot 9.22)$$

$$1.9 - 1.5 = 0.4$$

Den faktiske økningen i volum er 0.3, hvor vi fikk 0.4. Grunner til dette er blant annet at eulerforover metoden gir ikke et helt nøyaktig svar. En annen feilkilde er at vi antar at sinuskurven vår er helt perfekt. Det vil si at vi antar at amplitudene og likevektslinjen alltid har samme verdi, noe de ikke har. I utregningene våre har vi for det meste regnet ut fra tall med hundredels desimal, dette fører til et liten feilmargin.

Sinuskurven er verifisert på mange ulike grunnlag. Et av disse er blant annet at amplituden til cosinus funksjonen har lik verdi som sinus amplituden dividert med vinkelfarten. Stigningstallet til cosinus funksjonen er også tilnærmet ekvivalent likevektslinjen til sinussignalet. I tillegg til dette kan vi se ut ifra det visuelle aspektet at sinus grafen har en naturlig forskyvning.

Kapittel 2

Filtrering (utført av hele gruppen)

2.1 Problemstilling

I dette prosjektet skal vi lage to forskjellige filtrerings funksjoner. FIR filteret og IIR filteret

Filtrering er et kjent verktøy som er brukt i alt fra temperaturmålere til fartsmålere. Hovedprinsippet med filtrering er å gjøre utsynet tydelig og støyfritt datasett om til oversiktlig grafer vi kan letttere analysere og forstå, uten at grafen blir kontaminert med støy.

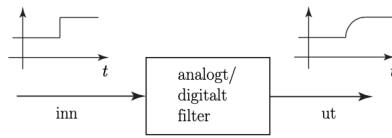
FIR-filteret (Finite Impulse Response) er et filter som tar for seg gjennomsnittet av en bestemt mengde tidligere målinger. Filteret blir jevnt og tydelig, men bruker mer datakraft ettersom den må regne igjennom flere tidligere verdier.

IIR-filteret (Infinite Impulse Response) er et filter som kun trenger den nye og en tidligere måling for å fungere, dette fører til at det er et mer foretrukket filter da det bruker lite datakraft, i tillegg til filteret også blir jevnt og tydelig.

2.2 Løsning til FIR-filter

2.2 Løsning til FIR-filter

Alle filter har som hensikt å jevne og glatte ut målingene, dette vil si at vi ikke får særlig skarpe verdiendringer som slår ut på grafen. I figur 2.1 ser vi en inn-verdi der verdiendringen er skarp(rådata), og en ut-verdi hvor endringen er glattet ut over en lengre periode(prosessert data). Verdiendringen er den samme på begge grafene, forskjellen ligger i hvor raskt endringen skjer.



Figur 2.1: Blokkskjema viser hvordan filteret gjør grafen mer glatt og lesbar.

FIR-filteret kan beskrives som et veldig intuitivt filter, ettersom man bare tar gjennomsnittet av et gitt antall forrige målinger. Dette vises ved:

$$\frac{1}{m} \sum_{n=0}^{m-1} Temp(k-n) \quad (2.1)$$

Temp refererer til temperaturen til målingen, *m* refererer til antallet forrige målinger vi vil ta gjennomsnittet av, og *n* er et variabel som refererer til verdien av forrige målinger.

Et eksempel der vi viser hvordan et FIR filter funker i praksis kan ses på denne måten: Vi setter $m=4$ og $k=50$

$$TempFIR(50) = \frac{1}{4} \cdot (Temp(50) + Temp(49) + Temp(47) + Temp(46)) \quad (2.2)$$

Utfordringen når man skal overføre denne formelen til kode dreier seg da rundt at *m*, ikke kan være større enn den nåværende *n* verdien, da dette fører til en indeks feil ettersom indeksen ikke kan være lavere enn 0 i matlab. Eksempel på dette: Vi setter $m=4$ og $k=2$

$$TempFIR(2) = \frac{1}{4} \cdot (Temp(2) + Temp(1) + Temp(0) + Temp(-1)) \quad (2.3)$$

2.2 Løsning til FIR-filter

For å fikse dette problemet, implementeres en *if* setning før utregnings-koden der vi setter $m=n$ når n er mindre enn m , slik at indeksen alltid holder seg over 0. Når denne koden er implementert vil regnestykket se slik ut når $m=4$ og $k=2$:

$$Temp_{FIR}(2) = \frac{1}{2} \cdot (Temp(2) + Temp(1)) \quad (2.4)$$

2.2.1 Kode

Før vi begynner med filteret så må vi ta imot målingene fra lyssensoren, i tillegg må vi også lage matrisene vi skal bruke:

Kode 2.1: Lysmålinger, og start verdi for matrisene.

```
24 % myColorSensor
25     Lys(1) = double(readLightIntensity(myColorSensor, 'reflected'));
26     %Legger første maalingene i matrisene
27     Temp(1) = Lys(1);
28     Temp_FIR(1) = Temp(1);
```

Lysmålingene skal representerer temperaturmålinger i oppgavene, vi velger derfor Temp som navn på matrisene.

Selve regningen er skrevet med hensyn til formelen, og det er brukt *sum* for å ta summen av verdiene. Koden vises her:

Kode 2.2: Kode for FIR-filter.

```
2 function [FilteredValue] = FIR_filter(Measurements, NoOfMeas)
3     if NoOfMeas ≥ length(Measurements)
4         NoOfMeas = length(Measurements);
5         FilteredValue= ...
6             (sum(Measurements(1:end)) - sum(Measurements(1:(end-NoOfMeas)))) / NoOfMeas;
6     else
7         FilteredValue= ...
8             (sum(Measurements(1:end)) - sum(Measurements(1:(end-NoOfMeas)))) / NoOfMeas;
8     end
9 end
```

Input i funksjonen er matrisen med målingene vi vil filtrere, m , som tilsvarer *NoOfMeans*. Desto høyere *NoOfMeas*, desto mer filtrerer *FIR* filteret.

2.2 Løsning til FIR-filter

For å finne den filtrerte nåverdien, summeres da hele temperatur matrisen. Dette subtraheres mot summen av matrisen opp mot indeksen til næværende måling minus m . Dette deles igjen på m . Vi vil da sitte igjen med gjennomsnittet av m antall målinger. Og målingen er filtrert.

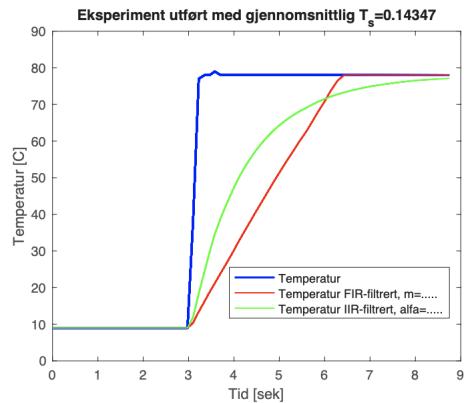
2.2.2 Resultat

I dette kapitelet blir resultatene fra eksperimentet 5.5.1 med hensyn til FIR-filteret presentert. I eksperiment 5.5.1 skulle vi simulere en temperaturmåler som går fra romtemperatur til en nylaget kaffe(nykokt vann).

Resultatet består av en ufiltrert graf og en filtrert graf der vi har brukt FIR-filteret. Målet med disse målingene er å komme oss nærmest grafen som ble gitt i oppgaven. Vi forandrer m for å komme oss nærmest mulig den gitte grafen, ved hjelp av prøv-og feilmetoden.

5.5.1 Eksperimentet

Før vi begynte å lage våre egne målinger og filtrere disse, måtte vi først analysere den gitte grafen vi skulle gjenskape.

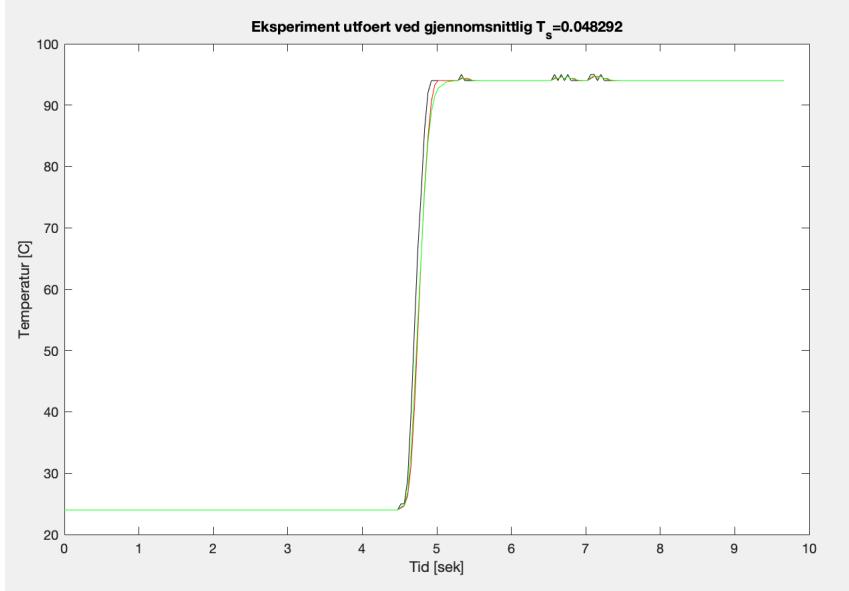


Figur 2.2: Denne grafen skal gjenskapes ved hjelp av prøv- og feilmetoden.(rød linje)

Det første vi la merke til var hvordan T_s (tidsskrittet fra hver måling) på vår graf

2.2 Løsning til FIR-filter

måtte være rundt det samme som i 2.2. Dersom vi målte uten å plotte endte vi opp med en graf som vist i 2.3

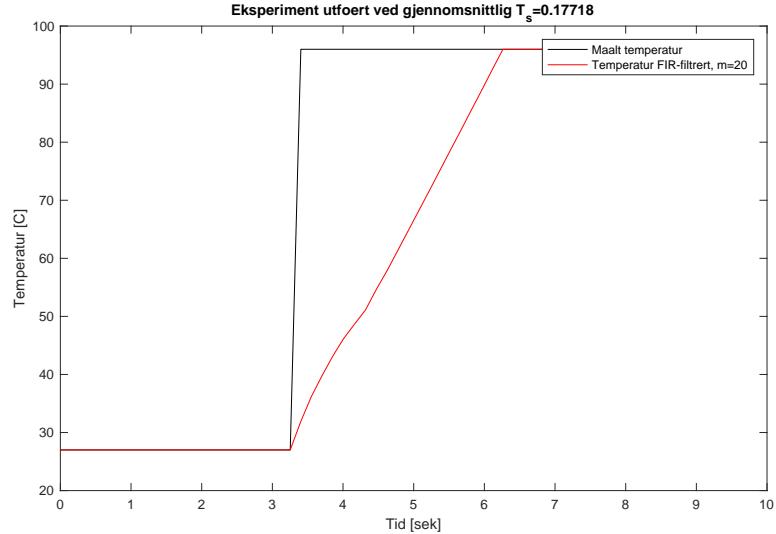


Figur 2.3: Bildet viser en graf der T_s er veldig liten, som fører til flere målinger der vi må ha en høyere m for å filtrere grafen skikkelig.

Vi konkluderte derfor at vi måtte plotte mens vi målte for å komme nærmest T_s brukt i 2.2.

Det neste steget var naturligvis å endre på m til grafen begynte å ligne på 2.2. Etter flere forsøk, endte vi på $m=20$. Vi merket også at desto høyere m vi brukte, desto mer bøyde grafen seg i begynnelsen. Vi tror dette kommer av *if* setningen, da den filtrerte grafen slutter å bøye seg når $k=m$, noe vi ser i 2.5. Dette kan da være en mulig feilkilde i koden.

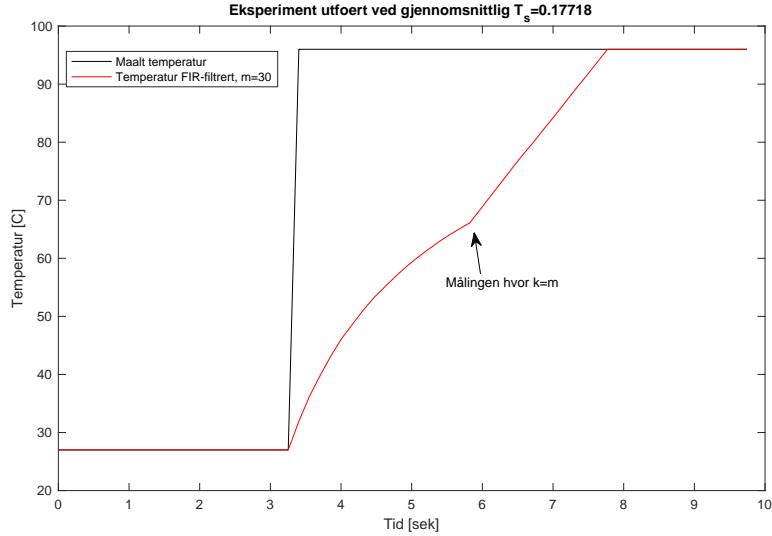
2.2 Løsning til FIR-filter



Figur 2.4: Grafen med målinger vi fikk som så nærmest ut som 2.2.

Vi kan si at vi har gjenskapt 2.2, ettersom vi ser at filteret varer like lenge. Vi ser på øyemål at begge grafene starter rundt $Tid = 3$ og at dem er ferdig filtrert rundt $Tid = 6$. Å få begge grafene til å være identiske ville nok ha vært tilnærmet umulig, ettersom vi ikke klarte å etterligne startmålingen der $Temp$ skal være rundt 10. Vi brukte den mørkeste verdien gitt på Gråskalaarket, og lysmåleren målte verdier rundt 27. Med tanke på at lysmåleren bare måler opp til 100, har vi en mindre verdiendring i 2.4 enn i 2.2. Dette fører videre til en raskere filtreringsprosess, som forklarer at 2.4 filtrerer litt raskere når vi ser på eksaktverdiene.

2.3 IIR-filter



Figur 2.5: Grafen hvor en mulig feilkilde blir vist.

2.3 IIR-filter

IIR-filteret kan anses som et litt mer avansert filter iforhold til FIR-filteret, men den er også veldig intuitivt. Hovedprinsippet og hovedsalgspunktet bak IIR-filteret ligger i at det trenger bare to målinger. Nemlig den forrigefiltrerte- og nåværende-målingen for at filteret skal fungere. Dette vises ved:

$$Temp_{IIR}(k) = \alpha \cdot Temp(k) + (1 - \alpha) \cdot Temp_{IIR}(k - 1) \quad (2.5)$$

Effektiviteten til dette filteret ligger ikke i gjennomsnittet av forrige målinger som FIR-filteret, men heller konstanten α . Filteret funker ved at den nye målingen multipliseres med α , og den forrige målingen multipliseres med $(1 - \alpha)$, for så å addere dem. Vi ser hvordan forskjellige verdier av alfa påvirker ut-verdien av filteret i disse eksempelene:

Vi setter α lik 1. Nåværende måling er 5, og forrige filtrerte måling er 1.

$$Temp_{IIR}(k) = 1 \cdot 5 + 0 \cdot 1 \quad (2.6)$$

$$Temp_{IIR}(k) = 5$$

2.3 IIR-filter

Vi setter α lik 0.5, med de samme målingene.

$$Temp_{IIR}(k) = 0.5 \cdot 5 + 0.5 \cdot 1 \quad (2.7)$$

$$Temp_{IIR}(k) = 2.5 + 0.5 = 3$$

Tilslutt setter vi α lik 0.1, med de samme målingene.

$$Temp_{IIR}(k) = 0.1 \cdot 5 + 0.9 \cdot 1 \quad (2.8)$$

$$Temp_{IIR}(k) = 0.5 + 0.9 = 1.4$$

Gjennomgående i eksemplene er verdiforandringen av konstanten α . Vi ser at hvor lengre α nærmer seg 0, så filtrerer IIR-filteret mer. Dette kommer av at jo større $(1-\alpha)$ blir, jo mer faktorerer vi inn de forrige verdiene istedet for de nyere verdiene, ergo en filtrering tar sted.

2.3.1 Kode

Likt som i ?? begynner vi først med å lage matrisene våre:

Kode 2.3: Målinger for IIR.

```
23     Lys(1) = double(readLightIntensity(myColorSensor, 'reflected'));
24
25 %Legger første målingene i matrisene
26 Temp(1) = Lys(1);
27 Temp_IIR(1) = Temp(1);
```

Lysmålingene representerer igjen temperatur, og vi setter lager derfor Temp matrisen.

Relativt til FIR-filterer, er koden for IIR-Filteret veldig simplistisk, ettersom det ikke er brukt noen kommandoer for å regne det ut da det bare trengs en utregning. Koden blir da som følger:

Kode 2.4: Kode for IIR-filter.

```
1 function [FilteredValue] = ...
    IIR_filter(OldFilteredValue, Measurements, alfa)
2 FilteredValue=alfa*Measurements+(1-alfa)*OldFilteredValue;
3 end
```

2.3 IIR-filter

Input i funksjonen er den forrige filtrerte verdien, den nåværende målingen, og α . Desto lavere α , desto mer filtrerte verdier får vi som *output* I Eksperiment 5.5.2 skal vi filtrere støy, vi legger derfor til linjen

Kode 2.5: Kode for kunstig laget støy

```
1 Temp(k) = Lys(k) + randn;  
2 end
```

2.3.2 Resultat

I dette kapitelet blir resultatene fra eksperiment 5.5.1 og 5.5.2 med hensyn til IIR-filteret presentert. I eksperiment 5.5.1 skulle vi simulere en temperaturstigning fra romtemperatur til nykøkt vann. I eksperiment 5.5.2 skal vi simulere en temepraturmåler med støy.

Resultatet i 5.5.1 består av en ufiltrert graf og en filtrert graf der vi har brukt IIR-filteret- Målet med disse målingene er å komme oss nærmest en graf som ble gitt i oppgaven. Vi forandrer α for å komme oss nærmest mulig den gitte filtrerte grafen, ved hjelp av prøv- og feilmетодen.

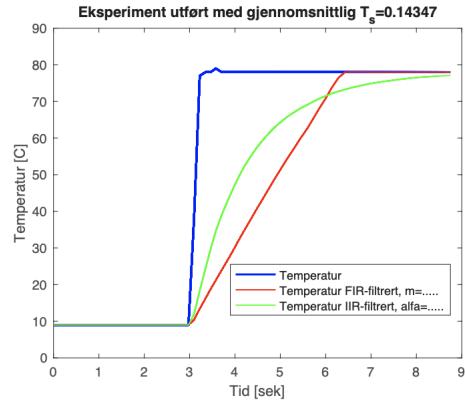
Resultatet i 5.5.2 består også av en ufiltrert- og filtrert graf, bare denne gangen skal vi se hvordan den filtrerte grafen forandrer seg ved ulike verdier av α , og hvordan dette hjelper oss når vi skal lese av grafen.

Eksperiment 5.5.1

, Fremgangsmåten her er helt lik som i FIR-filter delen, bare istedet for å bytte på m verdien, bytter vi α verdien.

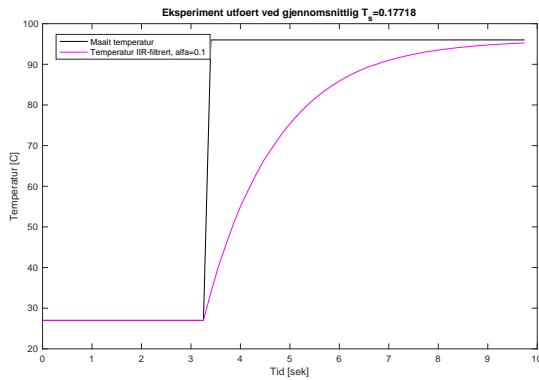
Den gitte grafen ser slik ut:

2.3 IIR-filter



Figur 2.6: Denne grafen skal gjenskapes ved hjelp av prøv- og feilmetoden.(Grønn linje)

Ulikt fra FIR-kapitelet derimot, fant vi α verdien på første forsøk. Grafen så slik ut:



Figur 2.7: Grafen vi fikk da vi satt α lik 0.1.

Verdien vi valgte for α var 0.1. Grunnen til vi kan si at grafen er den samme er fordi vi kan se at begge grafene begynner rundt $Tid = 3$ og vi ser at den filtrerte verdien er nærmest lik den ufiltrerte verdien når $Tid = 9$ i både 2.6 og 2.7 .

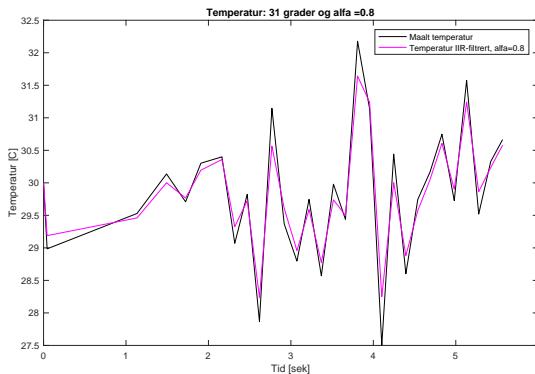
2.3 IIR-filter

Eksperiment 5.5.2

Hovedprinsippet i eksperiment 5.5.2 handler om hvordan filtrering forenkler grafer slik at man kan enklere forstå og bruke dem videre.

Fremgangsmåten vår var å legge til kunstige støyen, målet med lysmåleren, og se hvilke verdier av α filtrerer best. Vi repetepte dette 4 ganger med 4 forskjellige verdier av α .

Vår første måling satt vi α lik 0.8 og holdt lysmåleren til en konstant verdi, slik at vi kunne få et nærblikk på hvordan filteret håndterte støyen. Grafen så slik ut:

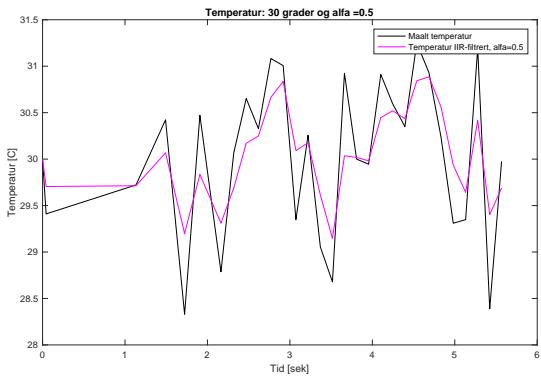


Figur 2.8: Grafen vi fikk da vi satt α lik 0.8.

Ut fra 2.8 kan vi se at de filtrerte verdiene fortsatt hoppe opp og ned som de ufiltrerte. Ettersom grafen kun skal simulere støy, er ikke $\alpha = 0.8$ en god verdi å filtrere på.

På den andre målingen satte vi ned α til 0.5. Grafen så slik ut:

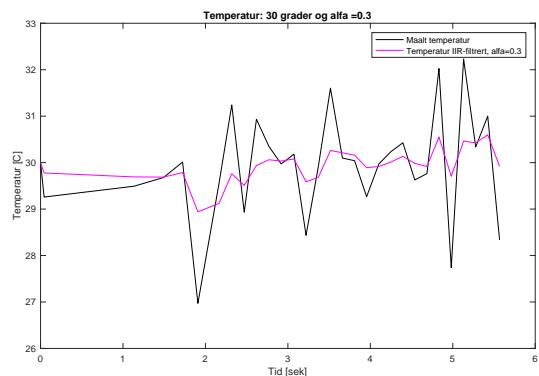
2.3 IIR-filter



Figur 2.9: Grafen vi fikk da vi satt α lik 0.5.

I forhold til 2.8 ser 2.9 litt mer filtrert ut, men det er fortsatt for store verdiendringer. α lik 0.5 er da heller ikke en optimal verdi å filtrere på.

På den tredje målingen satte vi α ned til 0.3. Grafen så slik ut:

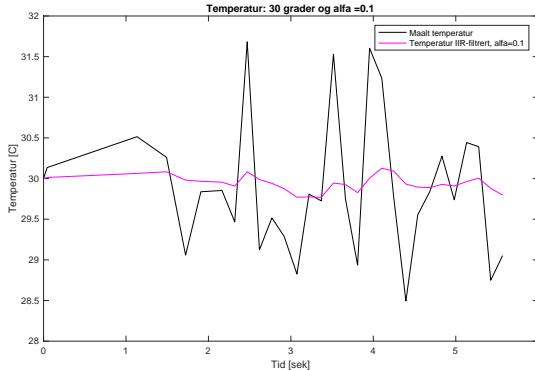


Figur 2.10: Grafen vi fikk da vi satt α lik 0.3.

Man kan se at denne grafen er mer filtrert. Verdiendringene i ?? er markant redusert, og man kan se at verdiene er mye mer lesbare og brukbare iforhold til f.eks 2.8 der den filtrerte grafen er veldig lik den ufiltrerte. α lik 0.3 er en brukbar verdi og filtrere denne typen støy på.

2.4 Sammenligning

På den fjerde grafen satte vi α ned til 0.1, og målte en siste gang. Grafen så slik ut:



Figur 2.11: Grafen vi fikk da vi satt α lik 0.1.

Etter å ha sett på 3 forskjellige filtrerte støygrafer, kan man raskt konkludere, når man ser denne grafen, at dette er den optimale verdien å filtrere denne typen støy på. Filteret reagerer fortsatt til verdiendringer, men den er mye glattere og brukbar enn alle de forrige grafene. Vi vet at den faktiske verdien til målingene er 30 grader. Verdiendringer til den filtrerte grafen er mindre enn 0.5. Dette betyr at hvis vi skal senere regne med disse verdiene, blir feilmarginene betydelig mindre.

2.4 Sammenligning

FIR- og IIR-filterene har begge sine positive og negative sider, hvor begge har sine unike bruksområder.

2.4.1 FIR-filter

FIR-filteret er et veldig enkelt filter å ta i bruk, og gir veldig pene lineære grafer, imotsettning til IIR-filteret gir det helt stabile grafer med høy stabilitet til mindre vedlikehold.

Det negative med FIR-filteret i denne situasjonen hvor vi vil regne og måle raskest

2.4 Sammenligning

mulig, er at den tar mye datakraft hvis vi vil at m skal være en høyere verdi. Bruk av mer datakraft øker tidsskrittet mellom hver måling og som et resultat av det så øker *latency*

2.4.2 IIR-filter

IIR-filteret er det fortrukne filteret i dette prosjektet ettersom den appelerer til våre trengsler. Den bruker lite datakraft, har en lavere *latency* og er bedre å filtrere støyete målinger.

På den andre siden kan vi si at det også er noen negative sider med IIR-filteret. Blant annet at den ikke kan bli en lineær graf om det kan være praktisk, den kan gjøre analyse av målingene litt vanskeligere, da den må ha en α som er bra kalibrert til oppgaven, og at den kan være litt mindre stabil.

Kapittel 3

Numerisk derivasjon (utført av Thomas og Marius)



Figur 3.1: I denne delen av prosjektet vil vi bruke de samme måle instrumentene som vi gjorde i nummerisk integrasjon kap1. Vi kommer kun til å bruke lyssensoren, som er avbildet nederst til venstre.

3.1 Problemstilling

3.1 Problemstilling

I dette kapitelet vil vi gå gjennom praktisk bruk av numerisk derivasjon.

Vi kommer til å simulere fart og avstand ved bruk av numerisk derivasjon. Dette kapitelet består av to oppgaver. Den første oppgaven er å derivere en konstant, hvor del 2 består av å derivere et sinussignal.

3.2 Teori

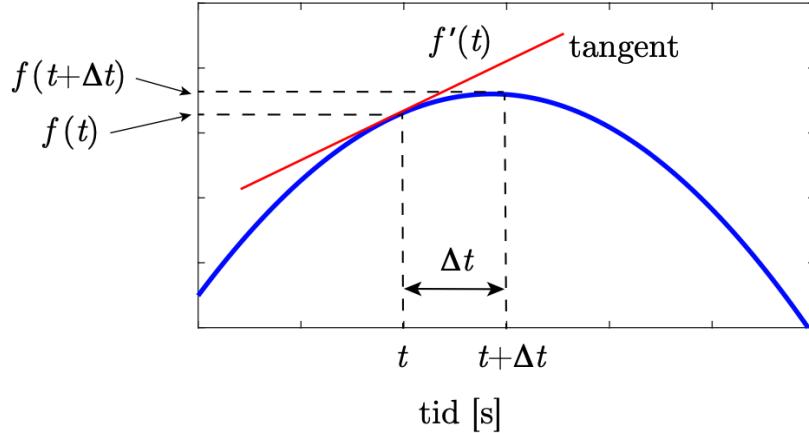
3.2.1 Derivasjon definisjon

For å kunne derivere praktisk er det viktig å ha forkunnskapene om hva derivasjon er, og hvordan man kan gjøre dette praktisk. Den generelle formelen for derivasjon er:

$$f'(t) = \lim_{\Delta t \rightarrow 0} \frac{f(t + \Delta t) - f(t)}{\Delta t} \quad (3.1)$$

Hvor Δt er endringen i tid. I dette kapitelet kommer vi til å skrive derivasjon slik $\frac{df(t)}{dt}$ i stedet for $f'(t)$. Under finner du en god illustrasjon som beskriver derivasjon grafisk.

3.2 Teori



Figur 3.2: [3] Illustrasjonen er hentet fra.... Og som du ser viser den endringen i tid, og verdiene den har på y-aksen.

Figuren ovenfor forklarer utrolig godt hvordan derivasjon virker. Dersom man skal splitte opp ligningen 3.1 ser vi at i telleren regner vi differansen mellom y verdiene, også kjent som Δy , og vi regner nevneren som differansen av tid(x). Derfor kan man også si at derivasjons tangenten er ekvivalent med forholde mellom x og y aksene.

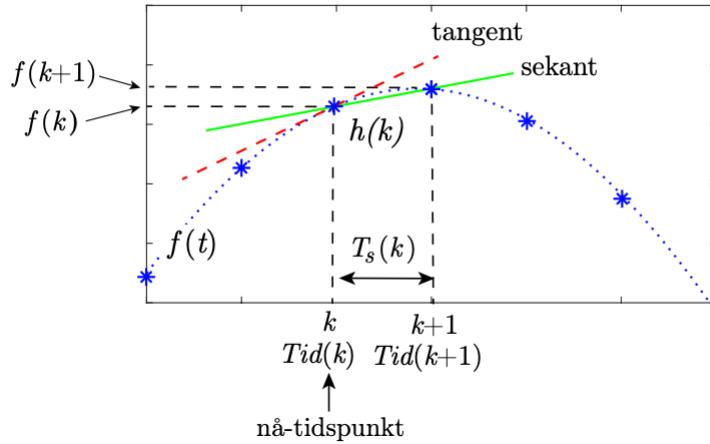
$$f'(t) = \frac{\Delta y}{\Delta x} \quad (3.2)$$

3.2.2 Praktisk bruk av numerisk derivasjon

Når vi skal regne ut derivasjon er det en fordel å kunne gjøre dette i sanntid. Det er alltid en fordel å kunne regne ut fart utifra hva den er der og da, framfor det den var over hele strekningen. For å klare dette må man utføre numerisk derivasjon.

Numerisk derivasjon lar oss utføre derivasjon selv om vi ikke har en gitt funksjon. Man husker fra kapitelet om numerisk integrasjon at måten vi integrerte på, var ved å måle et kvadratisk areal mellom hver måling. Numerisk derivasjon går også utpå å dele opp grafen, men nå ønsker vi bare punkter på grafen vår. Dette er bedre illustrert i bilde nedenfor.

3.2 Teori



Figur 3.3: hentet fra [3]

Bilde ovenfor viser hvordan tangenten blir til en sekant. En sekant er en linje som går gjennom to punkt, i vårt tilfellet gjennom to målinger. Først må vi se på hvordan man finner den til nærmeste verdien for den deriverte, og den kan skrives slik:

$$f'(t) \approx \frac{f(t + T_s) - f(t)}{T_s} \quad (3.3)$$

Denne funksjonen er nokså lik funksjonen 3.1. T_s står for tidsskritt og er differansen i tid mellom hver måling. Derfor kan vi bare bytte ut Δt verdien med T_s fra derivasjons definisjonen. T_s er altså:

$$T_s(k) = Tid(k + 1) - Tid(k) \quad (3.4)$$

For å bruke vår indeks k blir ligningen følgende:

$$h(k) = \frac{f(k + 1) - f(k)}{T_s(k)} \quad (3.5)$$

Dette er hvor k er målet på antall målinger. Dette er den samme indeksen vi

3.3 Forslag til Løsning

brukte i kapitel 1, og den lar oss bruke våre tidligere målinger når vi deriverer.

For å finne sekanten blir ligningen seende slik ut:

$$h(k-1) = \frac{f(k) - f(k-1)}{Ts(k-1)} \quad (3.6)$$

$Ts(k-1)$ vil da være

$$Ts(k-1) = Tid(k) - Tid(k-1) \quad (3.7)$$

Numerisk derivasjon gir ikke et konkret rett svar, med mindre antall målinger er evig. Man har som mål å få sekanten så nær tangenten som mulig. Det vil si at vi får kun et estimat over hva tangenten er i nærheten av.

3.3 Forslag til Løsning

3.3.1 Forslag til løsning av et lineært signal

I den første oppgaven i dette kapitelet har vi fått i oppgave å simulere at vi er politi og skal måle farten på forbipasserende biler. Dersom vi antar at vi bare har en konstant som er farten kan vi regne dette ut i fra avstanden.

Avstand er gitt ved

$$V * t = s \quad (3.8)$$

Farten V kommer vi til å referere som konstanten a videre i denne oppgaven.

Det vil si at dersom vi ønsker å finne farten til et legeme kan vi derivere avstanden:

$$\frac{d}{dt}(a \cdot t) = a \quad (3.9)$$

vi dereiverer her med hensyn på tiden t for å finne farten a i [m/s]

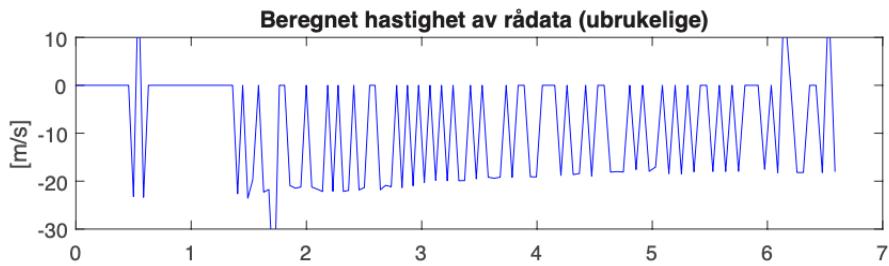
3.3 Forslag til Løsning

Måten vi skal verifisere ligningen på er ved å dra lyssensoren langs et sort hvit spektrum som vist under.



Figur 3.4: [3]Lyssensoren ble ført langs denne stripen mens vi mÅlte verdiene. Bildet er hentet fra...

Verdiene vi mottar må bli filtrerte slik at vi får et skikkelig overblikk over hvordan data-en ser ut. Uten filter vil det være tilnærmet umulig å få et skikkelig overblikk over beregnet hastighet. Et eksempel fra prosjekt beskrivelsen viser et ufiltrert signal under.



Figur 3.5: [3]På dette bilde ser man hvordan et ufiltrert derivasjons funksjon ser ut.hentet fra...

3.3.2 Forslag til løsning av et sinussignal

I oppgave to som omhandler derivasjon skal vi ta for oss numerisk derivasjon med et konstant ledd bias b . Vi skal simulere at vi står på bunden av en atraksjon som går opp og ned, og danner en perfekt sinus kurve. vi skal altså regne ut:

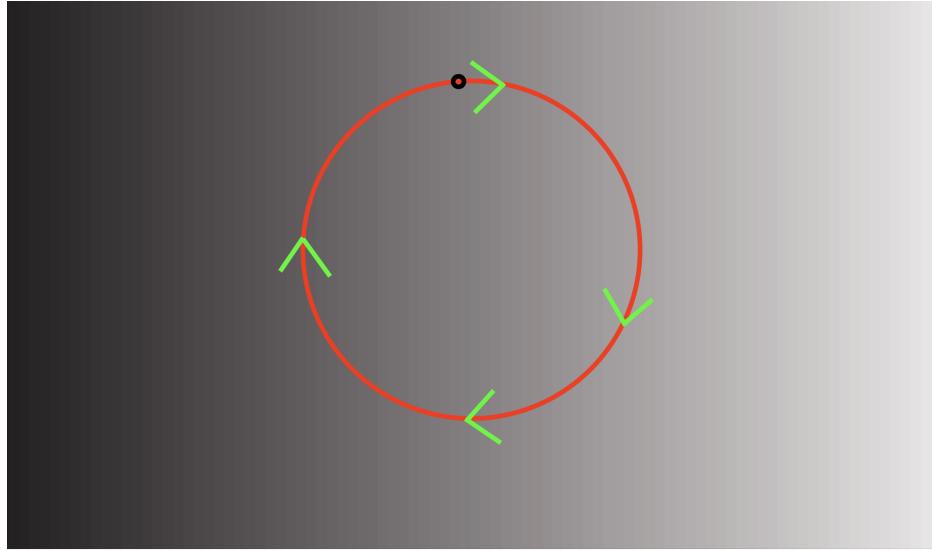
$$\frac{d}{dt}(b + a \cdot \sin(\omega \cdot t)) \quad (3.10)$$

Rent matematisk vil den deriverte være:

$$\frac{d}{dt}(b + a \cdot \sin(\omega \cdot t)) = a \cdot \omega \cdot \cos(\omega \cdot t) \quad (3.11)$$

3.3 Forslag til Løsning

Grafen ble laget ved å bevege lyssensoren i sirkler slik som man gjorde i integrasjonsoppgaven. Dette er slik at man får best mulig sinuskurve. Sinuskurven er fortsatt et mål på fart basert på attraksjonen.



Figur 3.6: [3]Dette er samme figur som ble brukt i kapitelet om numerisk integrasjon. Og det er siden vi valgte å gjøre det samme for numerisk derivasjon.

Når vi skal verifiserer sinus signalet vårt kan vi gjøre dette matematisk ved å finne enkelte verdier. vinkelfart ω er ett mål vi må ha når vi skal verifisere den deriverte funksjonen. Vinkelfart er det samme som en sirkel runde fordelt på tidsdifferansen mellom to toppunkt eller to bunnpunkt. Vinkelfart formelen er altså:

$$\omega = \frac{2\pi}{T} \quad (3.12)$$

Hvor T er:

$$T = Toppamplitude_2 - Toppamplitude_1 \quad (3.13)$$

I motsetning til numerisk integrasjon trenger vi ikke å finne ut b verdien til den deriverte. Det er siden dette leddet forsvinner, hvorav i kapitelet om numerisk integrasjon ble biasen/likevektslinjen et stigningstall.

3.4 Resultat

Den siste verdien vi mangler da er a verdien vår. Dette er amplituden til cosinus/sinus funksjonen. Vi kan determinere at dette er amplituden grunnet sinus og cosinus sin maks verdi er 1. Det betyr at det eneste som påvirker differansen mellom biasen og amplituden er konstanten a . Dette er da kun før man har derivert funksjonen, dette skyldes vinkelfarten som blir multiplisert med konstanten vår a .

Derfor ender vi opp med den deriverte som er lik:

$$a \cdot \omega \cdot \cos(\omega \cdot t) \quad (3.14)$$

3.3.3 Funksjonen

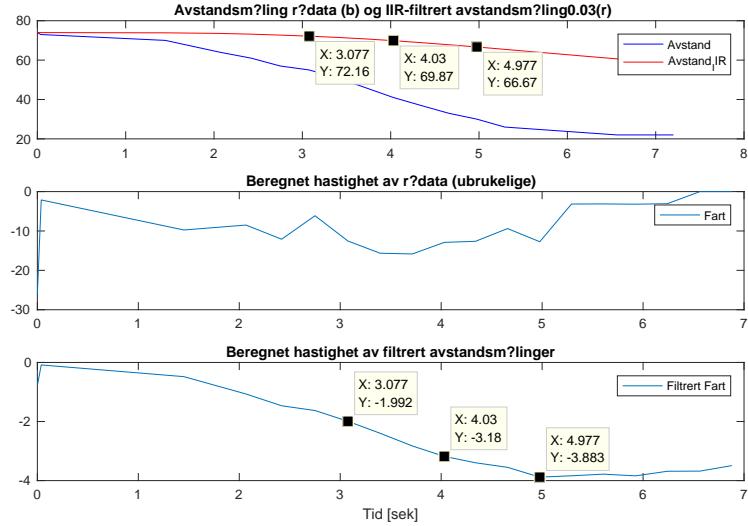
```
1 function [Secant] = Derivasjon(FunctionValue, TimeStep)
2
3 Secant = diff(FunctionValue) / TimeStep;
4
5 end
```

3.4 Resultat

3.4.1 Derivasjon av et lineært signal

I denne oppgaven er målet å derivere et lineært signal. Vi kommer til å ta utgangspunktet i den filtrerte funksjonen. Dette er siden et ufiltrert signal vil ikke være lesbart, noe man tydelig kan se i den midterste grafen.

3.4 Resultat



Figur 3.7: Den blå funksjonen øverst viser den ufiltrerte avstanden, hvor den røde funksjonen viser den filtrerte avstanden. Den midste grafen viser beregnet hastighet av et ufiltrert signal. Den nederste grafen viser den deriverte av den filtrerte funksjonen(røde).

for å finne gjennomsnittfarten, kan vi regne ut tangenten. For å gjøre dette vil vi bruke ligning 3.2.

$$Gjennomsnittsfart = \frac{\Delta y}{\Delta x} \quad (3.15)$$

Når vi implementerer verdiene fra grafen vår blir det da slik:

$$\begin{aligned} Gjennomsnittsfart &= \frac{66.7 - 72.2}{5.0 - 3.1} \\ &= \frac{-5.5}{1.9} = -2.9 \end{aligned} \quad (3.16)$$

Dette kan stemme siden vi ikke har en konstant fart. -2.9 er også mellom intervallet -2.0 og -3.9, som er grensen satt av vårt derivate uttrykk i nederste graf. For å være sikre regner vi også ut gjennomsnittsfarten mellom $x = 3.1$ og 4.0 . Deretter regner vi snitt farten mellom 4.0 og 5.0 .

x mellom 3.1 og 4.0

$$\frac{69.9 - 72.2}{4.0 - 3.1} = -2.56 \quad (3.17)$$

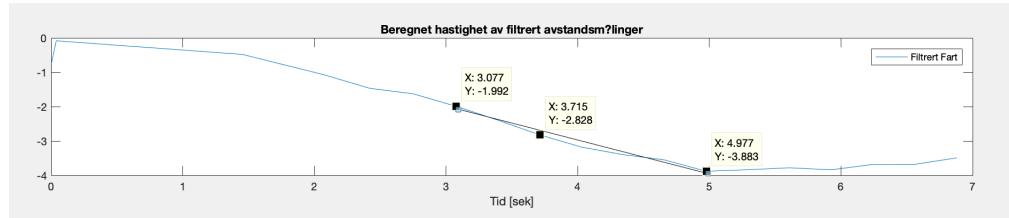
3.4 Resultat

-2.56 virker logisk siden det er mellom intervallet -2.0 og -3.2

x mellom 4.0 og 5.0

$$\frac{66.7 - 69.9}{5.0 - 4.0} = -3.2 \quad (3.18)$$

Dette er også en verdi som er mellom intervallet -3.2 og -3.9



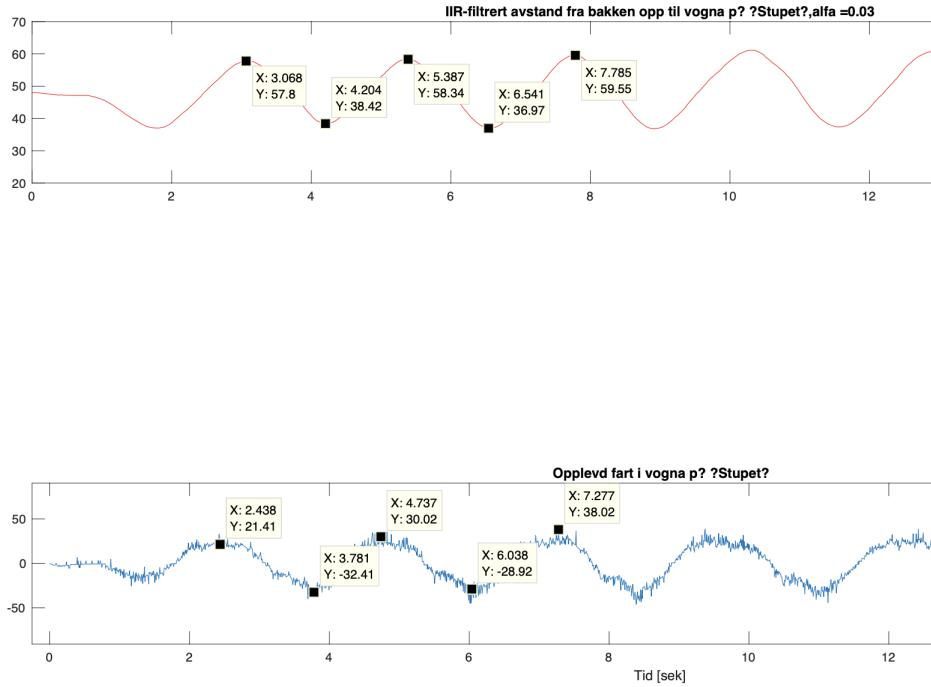
Figur 3.8: Man ser her at farten har en tilnærmet lineær minking, som illustrert av den sorte linjen.

Vi kan verifisere at derivasjons koden vår virker siden vi får lik gjennomsnittsfart, når vi regner det selv, som når vi bruker koden.

3.4.2 Derivasjon av et sinussignal

I del to av denne oppgaven skal vi ta en nærmere titt når vi deriverer en sinus funksjon. Vi skal verifisere derivasjonen ved å finne ulike verdier.

3.4 Resultat



Figur 3.9: Den øverste grafen viser avstanden til stupet, og den nederste grafen viser farten

Formelen for farten blir:

$$\frac{d}{dt}(a \cdot \sin(\omega \cdot t) + b) = a \cdot \omega \cdot \cos(\omega \cdot t) \quad (3.19)$$

Den første konstanten vi ønsker å finne for å verifisere funksjonen er å finne biasen, også kjent som likevektslinjen b .

$$b = \frac{f_{maks} + f_{min}}{2} = \frac{58 + 37}{2} = 48 \quad (3.20)$$

Nå som vi vet biasen kan vi regne ut a_{\sin}

$$a = f_{maks} - b = 58 - 48 = 10 \quad (3.21)$$

3.4 Resultat

Vi kan regne T ved bruk av ligning 3.13

$$T = 5.4 - 3.1 = 2.3 \quad (3.22)$$

Når vi har T kan vi regne ut vinkelfarten ω , for å gjøre dette bruker vi ligning 3.12

$$\omega = \frac{2\pi}{2.3} = 2.7 \quad (3.23)$$

Biasen forsvinner når vi deriverer uttrykket. Det er det motsatte fra kapitel 1 hvor biasen ble til en stigningskonstant. Det at vi ikke har en bias etter vi har derivert fører til at vår deriverte funksjon vil ha lik amplitud bare i det negative feltet. Det vil si at toppunktet vil ha samme verdi som absolutt verdien til bunnpunktet.

Vi kan regne ut a_{cos} slik:

$$a_{cos} = 30 - 0 = 30 \quad (3.24)$$

Fra formel 3.19 vet man at relasjonen mellom a_{cos} og a_{sin} er:

$$a_{cos} = a_{sin} \cdot \omega \quad (3.25)$$

Så implementerer vi våre verdier inn i ligningen:

$$a_{cos} = 10 * 2.7 = 27 \approx 30 \quad (3.26)$$

Grunnen til at vi får ulike verdier fra de reelle svarene er på grunn av all støyen i den deriverte funksjonen vår, og avrunding under regning.

Vi har verifisert sinus funksjonen ved å sjekke at a_{cos} deler sammenhengen mellom sinus amplituden og vinkelfart. Man kan også se den visuelle forskjyvningen som differensierer en cosinus graf fra en sinus graf, satt ved samme startpunkt.

Kapittel 4

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



Figur 4.1: EV3-en koblet sammen til motorene og lyssensoren. I bilde til høyre ser man undersiden av bilen og hvordan den kan lese verdier fra underflaten. Det midterste bilde er bilde av motoren som ble brukt, det blir da brukt to av disse. Nederst til venstre er en lyssensor.

4.1 Problemstilling

4.1 Problemstilling

I denne delen av prosjektet prøver vi å ta for oss praktisk bruk av sensorer og motorer, ved hjelp av EV-3en. Det konkrete målet er å lage en maskin man kan styre, og som kan måle verdier. Vi ønsker å måle og sammenlikne ulike målinger mellom oss i gruppen. EV3-en skal følge banen og måle avvik for å se hvem som er best til å kjøre.

4.2 Sensorer og motorer

4.2.1 Motorer

En av de viktigste komponentene brukt i dette prosjektet er motorene. Maskinen vår består av to motorer som styrer fart og retning. De befinner seg i portene A og B i EV3-en.

4.2.2 Lyssensor

Sensoren vi bruker i denne delen av prosjektet er lyssensoren. Den blir brukt for å måle verdier som vi kan bruke ti å måle hvordan styringen er og hvem som kjørte best.

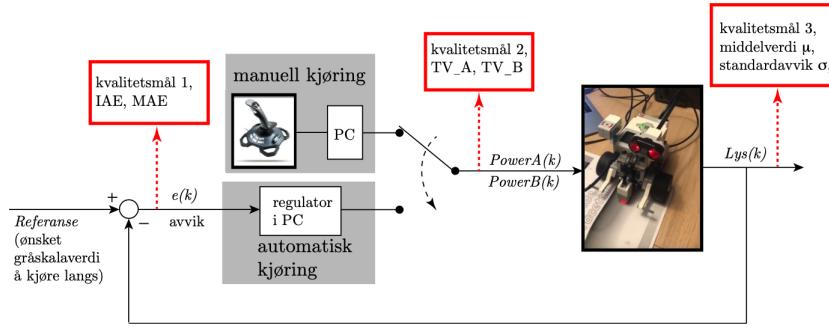
4.2.3 Joystick

Joystick-en er verktøyet vi skal bruke for å kontrollere motorene. Målet er at det skal føles best mulig ut å kunne kjøre bilen. For å kunne bruke joysticken må vi definere aksene, altså når den går framover/bakover og når den går til siden.

4.3 Kvalitetsmål

4.3 Kvalitetsmål

4.3.1 Kvalitetsmål struktur



Figur 4.2: Her ser vi strukturen bak kvalitetsmålingene. hentet fra [3]

Vi ser fra figur 4.2 at det første vi gjør er å finne en referanse. Ut ifra referansen finner vi de første kvalitetsmålingene *IAE* og *MAE*, mer om dette senere. I manuell kjøring kapittellet blir motorene styr av en *Joystick*. For å måle pådraget til motorene blir det gjort med TV_A og TV_B . Når vi har fått lysverdien vår, kan vi regne ut standardavvik σ og middelverdi μ . Dette er kvalitetsmål 3.

4.3.2 Avvik

For å måle hvem som er best til å kjøre målte vi opp ulike verdier fra lyssensoren. En av disse målingen er avviket. Vi måler avviket slik:

$$e(k) = Referanse - Lys(k) \quad (4.1)$$

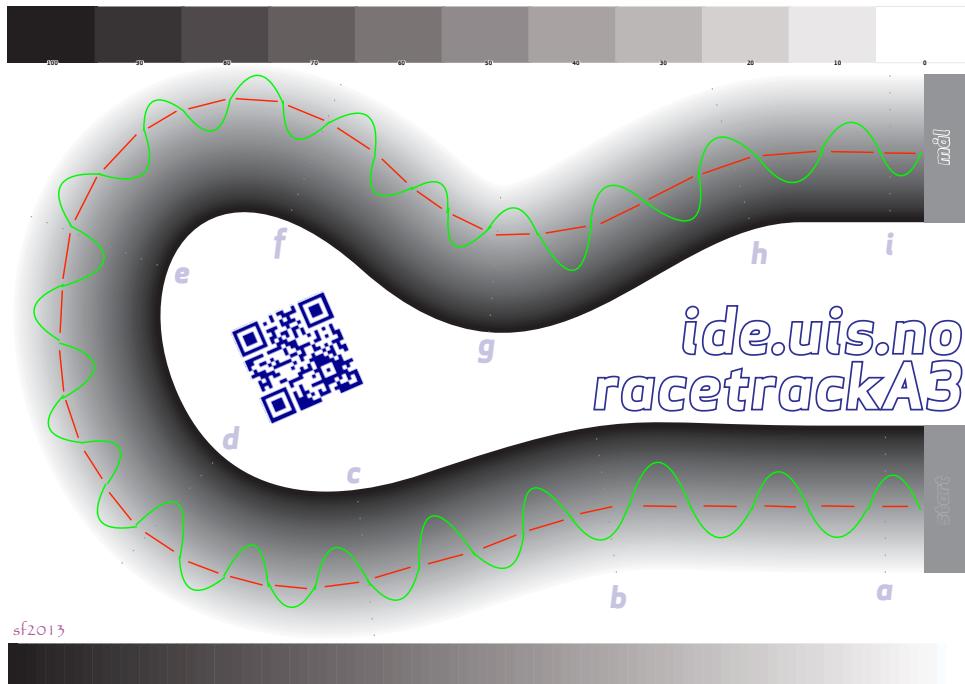
Hvor *Referansen* er $Lys(1)$ verdien vår.

Høyest avvik betyr verst kjøring, det vil si at målet er alltid å forblie på referansen. Dette er en måte vi definerer best kjøring på. Avvik er et av tre ulike kvalitetsmål.

4.3 Kvalitetsmål

4.3.3 Kvalitetsmål 1: IAE og MAE

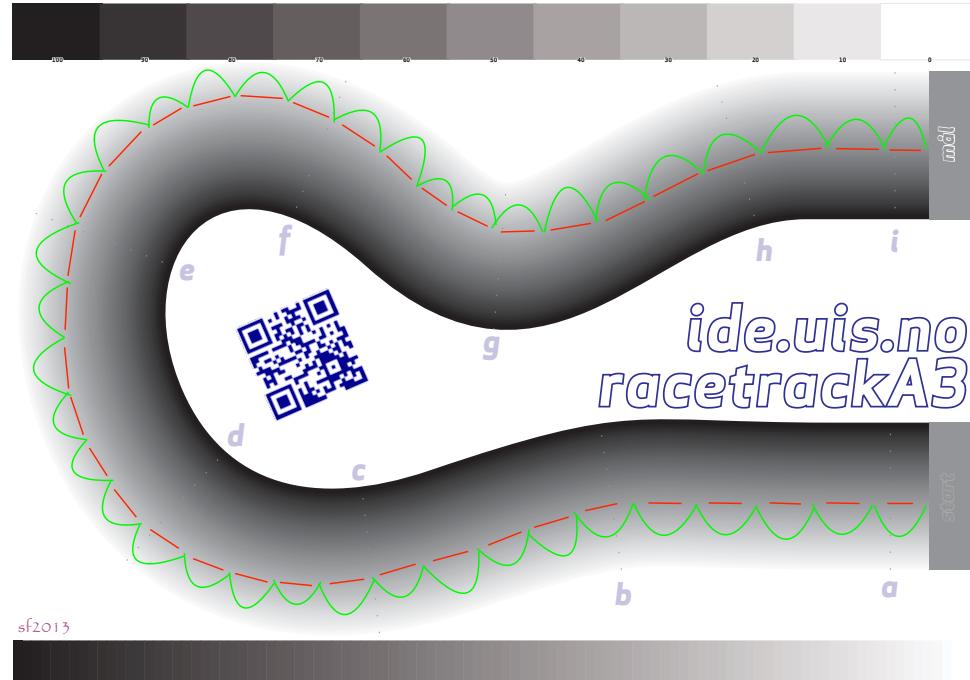
Problemet med avviket $e(k)$ er at det kan bli negativt. Det vil si at gjennomsnittsavviket kan være lavt selv om personen har kjørt helt forferdelig. Siden $e(k)$ kan være negativ vil det si at en person som kjører like mye på det sorte feltet som det hvite vil ha likt avvik som personen som klarte å følge banen perfekt. Dette ser man bedre illustrert i bilde nedenfor.



Figur 4.3: [3](Laget retning selv)Hvis vi antar at den grønne retningen er en perfekt sinuskurve i forhold til den røde linjen vil det si at de to sjåførene vil ha et likt avvik, Selv om det er klart at personen som har kjørt på den røde linjen har kjørt bedre.

En måte å fikse dette på er ved å alltid bruke absoluttverdien til $e(k)$. Da blir avviket seende ut som bilde under.

4.3 Kvalitetsmål



Figur 4.4: [3](Laget retning selv)Lik figur 4.3, men viser hvilke verdi $e(k)$ vil være dersom vi tar absoluttverdien.

Et logisk og enkelt kvalitetsmål som er avhengig av avviket er IAE . IAE står for *Integral of Absolute Error* eller Integral av absolutt feil på norsk. Denne verdien er arealet av avviket. Denne kvalitetsmålingen er avhengig av tid. Den generelle formelen blir da:

$$IAE = \int_0^t |e(t)| \quad (4.2)$$

Eller ved numerisk integrasjon i matlab slik:

$$IAE(k) = IAE(k - 1) + abs(e(k - 1)) * Ts(k - 1) \quad (4.3)$$

Som i prinsipp følger samme oppskrift som 1.2. Ved å bruke numerisk integrasjon kan vi se IAE verdien i sanntid. Slik som da vi regnet ut volumet i kapitel 1.

4.3 Kvalitetsmål

MAE er også et kvalitetsmål som er basert på avviket $e(k)$. *MAE* eller *Mean Absolute Error* er gjennomsnitt avviket per måling. Denne kvalitetsmålingen har en større vekt på antall målinger framfor tid. Den generelle formelen til *MAE*

$$MAE = \frac{1}{k} \sum_{n=1}^k |e(n)| \quad (4.4)$$

For å måle *MAE* verdiene våre i sanntid blir formelen slik i matlab:

$$MAE(k) = sum(abs(e(1:k)))/k \quad (4.5)$$

Ved bruk av både *MAE* og *IAE* kan vi måle kvaliteten ved bruk av avvik. Vi sjekker total avvik og gjennomsnittlig avvik per måling. Dette er en del av den første kvalitetsmålingen, som vist i illustrasjonen

4.3.4 Kvalitetsmål 2: Pådrag

Pådraget av motorene er også et av de tre kvalitetsmålene. De to Motorene er navngitt A og B. *TV* eller *Total Variation* er kvalitetsmål for kvantifisering av pådragsbruk. Et høyst aktivt pådragsbruk kan slite ut pådragsorganet. For å måle pådraget kan vi regne det ut slik:

$$TV = \sum_{n=1}^k |u(n) - u(n-1)| \quad (4.6)$$

Hvor $u(k)$ er $PowerA(k)$ og $PowerB(k)$, altså pådragssignalet.

I matlab blir ligningen ført opp slik:

$$TV_A(k) = abs(diff(PowerA(k-1:k))) + TV_A(k-1) \quad (4.7)$$

Pådraget for $PowerB$ er likt, bare at TV_A er TV_B og $PowerA$ er $PowerB$

4.3 Kvalitetsmål

4.3.5 Kvalitetsmål 3: Standardverdi og Middelverdi

Middelverdi er målingen som dukker opp flest ganger. I dette prosjektet μ det samme som middelverdier. For å finne middelverdien tar man summen av tallene og dividerer dem på antall målinger. Den generelle formelen er:

$$\mu = \frac{1}{k} \sum_{n=1}^k Lys(n) \quad (4.8)$$

Når vi fører dette opp i matlab blir det slik:

$$\mu = \text{sum}(lys(1:k))/k \quad (4.9)$$

Man kan også bruke *mean*-funksjonen i Matlab.

Standardavviket forteller hvor langt unna verdiene dine ligger fra middelverdien din. Man kan si at standardavvik er et mål på hvor spredt målingene er i forhold til middelverdi. Den generelle formelen på standardavvik er:

$$\sigma = \sqrt{\frac{1}{k} \sum_{n=1}^k (Lys(n) - \mu)^2} \quad (4.10)$$

I matlab vil funksjonen se slik ut:

$$\sigma = \text{sqrt}((\text{sum}((Lys(1:k) - \text{middelverdi}).^2))/k) \quad (4.11)$$

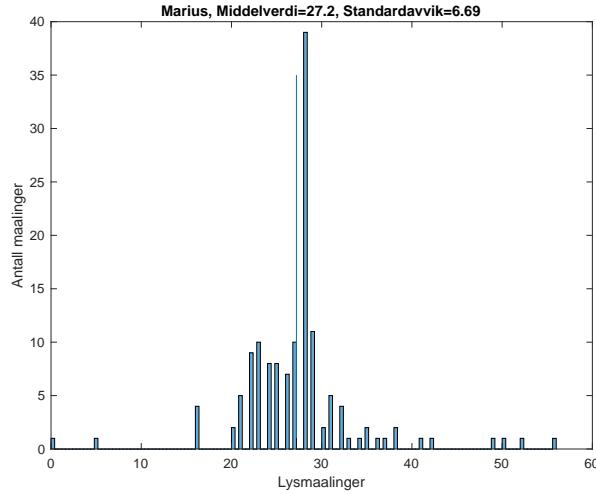
4.4 Resultat

4.4 Resultat

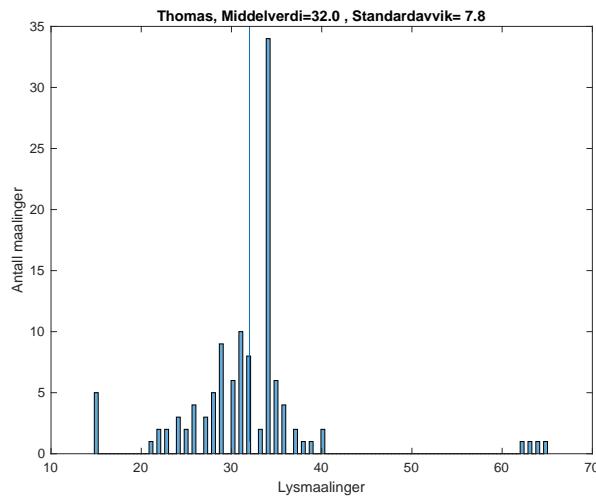
Tabell 4.1: Sluttresultater fra manuell kjøring til deltagere i gruppe 2067.

	Marius	Thomas
Plattform	MacBook Pro	MacBook Pro
Strømkilde	Batteri	Batteri
POX_F6_PlottData.m	Innenfor	Innenfor
Referanse	29	15
middelverdi μ	27.26	31.9
$ \text{Referanse}-\mu $	1.74	16.9
standardavvik σ	6.69	7.84
kjøretid [sek]	33.89	38.14
IAE	144.9*	563.2
MAE	4.5*	16.9
TV_A	502.7	302.7
TV_B	584.6	262.9
middelverdi av T_s [sek]	0.24	0.33
antall målinger (k)	140	116

4.4 Resultat



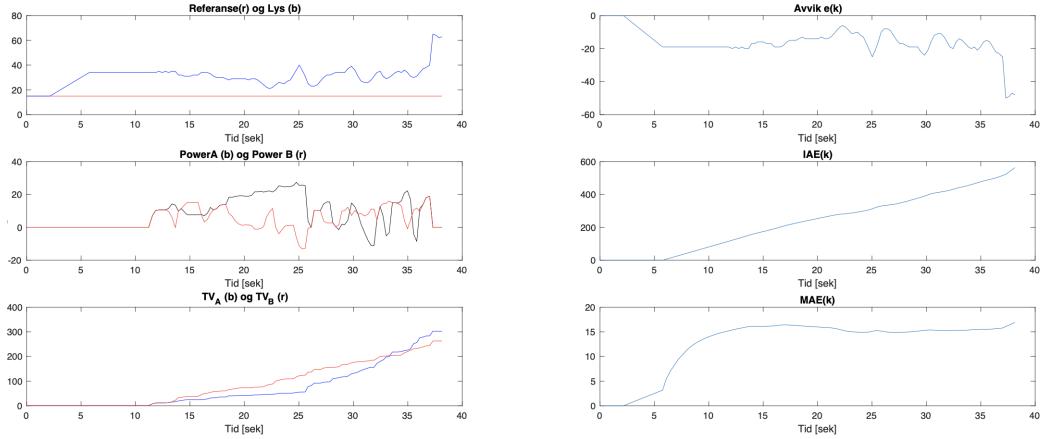
Figur 4.5: Histogrammet til Marius. Viser antall målinger og lysverdier. Linjen som kjører x aksen er middelverdien.



Figur 4.6: Histogrammet til Thomas. Viser antall målinger og lysverdier. Linjen som kjører x aksen er middelverdien.

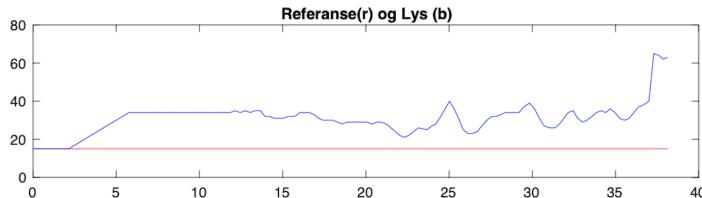
4.4 Resultat

Vi kan også se på kvalitetsmålene mens vi kjører. På denne måten kan vi se hvor pådraget var størst. Vi kan også se på hvorfor resultatene ble slik de gjorde. Thomas har blant annet en veldig høy IAE - verdi og dette kan man bedre illustrere ved å se på grafene.



Figur 4.7: Dette er de ulike verdiene til Thomas. Øverst til venstre har vi lysmålinger og referanse. Under der finner vi $PowerA$ og $PowerB$. Under $PowerA$ og $PowerB$ ser man pådragsbruket, TV_A og TV_B . På den høye siden finner vi avviket $e(k)$ på toppen. Under avviket ser vi IAE og derunder finner vi MAE

Det som er interessant når vi ser på målingene hans er spesielt grafen hans med referanse og lys.

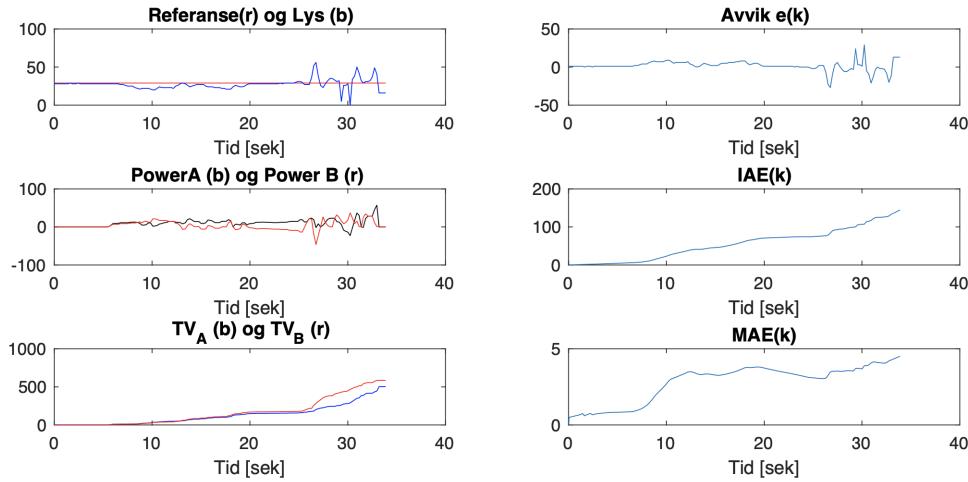


Figur 4.8: Den røde linjen er referansen, og målingene som er over er lysverdiene.

Vi ser at Thomas sine lysverdier går høyt over referansen. Thomas hadde en referanse som ligger 15 i lysverdi, noe som er en veldig lav lysverdi. Det vil si at lysverdien som ble plukket opp ganske tidlig, må ha vært på det mørke feltet i banen. Siden Thomas helt klart bare kjører over referansen forteller oss at Thomas har trodd at referansene verdien hans var mye høyere enn det den faktisk var.

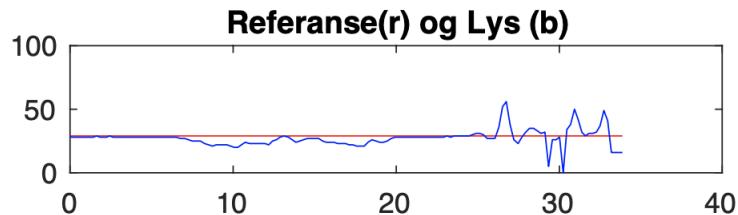
4.4 Resultat

Siden Thomas har så høy differanse mellom lysverdier og referanse vil dette føre til et større avvik. Det vil si at både IAE og MAE som er kvalitetsmålinger basert på avviket vil være vesentlig verre enn dersom han hadde hatt en høyere referanse verdi.



Figur 4.9: Dette er de ulike verdiene til Marius. Øverst til venstre har vi lysmålinger og referanse. Under der finner vi $PowerA$ og $PowerB$. Under $PowerA$ og $PowerB$ ser man pådragsbruket, TV_A og TV_B . På den høyre siden finner vi avviket $e(k)$ på toppen. Under avviket ser vi IAE og derunder finner vi MAE

Hvis vi ser på referansen og lysverdiene til Marius ser de slik ut.



Figur 4.10: Den røde linjen er referansen, og målingene som er over er lysverdiene.

I denne grafen kommer det tydeligere frem at Marius har et bedre referansepunkt

4.4 Resultat

enn det Thomas hadde. Vi kan se dette siden lys målingene går både over og under grafen noe som tyder på at Marius hadde bedre kontroll over hvor referansen var.

Setter vi alle verdiene sammen ser vi at Marius var flinkest til å kjøre. Det eneste punktet Marius har en verre verdi enn Thomas på, er pådraget. Det å ha høyt pådrag er ikke nødvendigvis gale, men det kan slite ut pådragsorganet. Når det kommer til de andre kvalitetsmålingene, vinner Marius soleklart.

Kapittel 5

Kreativ del Store prosjekter

5.1 PID kjøring

5.1.1 Problemstilling

I dette prosjektet skal vi fokusere på *PID*-kjøring, og bruke en *PID*-regulator for å kjøre automatisk igjennom banen som vi brukte i Kapittel 4 med minst mulig avvik fra referanse målingen.

5.1.2 Teori

I dette delkapittelet skal vi gå igjennom teorien rundt en *PID*-regulator.

PID-Regulator

En *PID*-regulator består av tre forskjellige ledd, som adderes sammen til å bli pådraget til motoren. Leddene er som navnet henviser til, navngitt *P* for proporsjonalitetsleddet, *I* for integrasjonsleddet, og *D* for derivasjonsleddet. Hvert ledd i seg selv, ville ikke ha fungert like bra i dette tilfellet med automatisk kjøring. Men sammen, dekker dem sine egne unøyaktigheter og gjør tilslutt at avviket blir

5.1 PID kjøring

kraftig redusert relativt til at vi bare skulle ha brukt ett av leddene.

Reguleringsavviket

Før vi kan regne, må vi først legge til grunn hvordan vi kommer fram til reguleringsavviket. Reguleringsavviket, lik som avviket vi regnte oss til i kapittel 4, er et mål på differansen mellom referanseverdien og nåværende måling.

$$e(t) = \text{referanse}(t) - m\text{ling}(t) \quad (5.1)$$

Den forteller oss hvor langt vekke vi er fra den ønskede målingen.

Proporsjonalitetsleddet

Proporsjonalitetsleddet er byggestenen til en *PID*-Regulator. Selve regningen bak leddet er svært enkel, og kan beskrives slik.

$$P(t) = K_p \cdot e(t) \quad (5.2)$$

I proporsjonalitetsleddet er hensikten bare å bruke avviket fra referansen og gange det med en konstant (proporsjonalitetskoeffisienten) som man varierer på, til man finner verdien som gir lavest avvik.

Integrasjonsleddet

Det er i integrasjonsleddet at *PID*-regulatoren virkelig skinner, og funksjonaliteten kommer frem. Vi ser i 5.1 når vi kun bruker proporsjonalitetesleddet så er det fremdeles forskjell på den ønskede verdien, og den reelle verdien. For å hjelpe å minske denne feilkilden, og redusere avviket enda mer, ganger vi den integrerte referanseverdien med en ny konstant K_i som vi varierer på samme måte som i proporsjonalitetskoeffisienten. Vi ender opp med mindre feil, og som resultat av dette minskes avviket. Integrasjonsleddet er gitt som dette:

$$I(t) = \int_0^t K_i \cdot e(t) dt \quad (5.3)$$

5.1 PID kjøring

Derivasjonsleddet

Dette leddet regner ut den fremtidige endringen til reguleringsavviket. Vi gjør det samme som i de forrige leddene når det gjelder Derivasjonskoeffisienten. Derivasjonsleddet bidrar med å øke stabiliteten ved at den reagerer raskere til endringer. Dette får vi bruk for i krappe svinger, eller andre situasjoner der det oppstår raske verdiendringer, der avviket ellers ville ha økt uten bruken av derivasjonsleddet. Derivasjonsleddet er gitt som:

$$\frac{d}{dt}(K_d \cdot e_f(t)) \quad (5.4)$$

Imotsetning til proporsjonalits- og integrasjonsleddet, er derivasjonsleddet ikke basert på $e(t)$ (reguleringsavviket), men e_f , som er den IIR-filtrerte versjonen av $e(t)$. Grunnen til at vi filtrerer $e(t)$ er på grunn av vi ikke vil ha en sterk verdiendring forårsaket av bakgrunnsstøy, da derivasjonsleddet enkelt kan tolke dette som en kommende sving, og påvirke pådraget slik at vi svinger vekk fra referanseverdien, og dermed øker avviket.

Basispådraget

Basispådraget er ikke alltid nødvendig å ha i en *PID*-regulator, men må tas i bruk i dette tilfellet. Regulatoren vår er laget slik at den kun kan svinge og har derfor ingen måte å kjøre framover på. Basispådraget u_0 er et mål på en konstant fart som roboten opprettholder iløpet av hele løpet. Men det er fremdeles viktig å være obs på at verdien til u_0 også må stilles inn. En u_0 verdi som er for høy gjør det vanskelig for *PID*-regulatoren å svinge med hensikt på lavest mulig avvik. u_0 må da mest sannsynlig være stilt inn til en lav verdi som skaper minst mulig avvik.

Kvalitetsmål

Videre skal det forklares hvilke typer kvalitetsmål som brukes for å vurdere hvilket innstillinger av de forskjellige koeffisientene gir det beste resultatet. Vi bruker akkurat de samme kvalitetsmålene som ble gitt i 4.2.

5.1 PID kjøring

5.1.3 Løsningsforslag

Å kode en PID-regulator blir relativt enkelt når man har forstått grunnprinsippene og matematikken bak dem. Vi tar i bruk funksjonene som ble laget i Kapittel 2,3 og 1, for å gjøre programmeringen betydelig enklere og oversiktlig. Før vi kan regne med verdiene, må vi først lage matrisene og variabelene som skal brukes, dette gjøres i GetFirstMeasurement filen.

Kode 5.1: Målinger for PID.

```
23     Lys(1) = double(readLightIntensity(myColorSensor, 'reflected'));
24     Referanse(1) = Lys(1);
25     e(1) = Referanse(1)-Lys(1);
26     IAE(1) = 0;
27     MAE(1) = 0;
28     TV_A(1) = 0;
29     TV_B(1) = 0;
30     P_K(1) = 0;
31     I(1)=0;
32     e_f(1)=0;
33     D(1)=0;
34     u(1)=0;
35     Avvik(1)=0;
```

De nye målingsverdiene er lagt til i GetNewMeasurement filen.

Kode 5.2: Nye målinger for PID.

```
28     e(k) = Referanse(1) - Lys(k);
29     Referanse(k) = Referanse(1);
30     Avvik(k) = abs(Referanse(1) - Lys(k))
```

Når alle matrisene og variabelene er tilrettelagt for, kan vi begynne med regningen i koden:

Kode 5.3: Regnekode for PID.

```
11 %Konstanter defineres her
12 Kp=1.7;
13 Ki=1.7;
14 Kd=0.9;
15 alfa=0.5;
16 u_0=10;
```

5.1 PID kjøring

```
17 %-----  
18 Ts(k-1)=Tid(k)-Tid(k-1);  
19  
20 IAE(k)= IAE(k-1)+abs(e(k-1))*Ts(k-1);  
21  
22 MAE(k) = sum(abs(e(1:k)))/k;  
23  
24 middelverdi = sum(Lys(1:k))/k;  
25  
26 standardavvik = sqrt((sum((Lys(1:k))-middelverdi).^2)/k);  
27  
28 P(k)=Kp*e(k);  
29  
30 I(k)= EulerForward(I(k-1),Ki*e(k-1),Ts(k-1));  
31  
32 e_f(k) = IIR_filter(e_f(k-1),e(k),alfa);  
33  
34 D(k) = Derivasjon(e_f(k-1:k),Ts(k-1));  
35  
36 u(k)=(P(k)+(Ki*I(k))+(Kd*D(k)));  
37  
38 AvvikGjennomsnitt = sum(Avvik(1:k))/k;
```

Regningen av *PID*-regulatoren er veldig rett fram iforhold til formelen. Alle ledene blir regnet ut ved hjelp av funksjonene som ble laget i de tidligere kapittelene. $u(k)$ er pådraget, og bestemmer motorfarten. Koden til motorene er:

Kode 5.4: Motor kode for PID.

```
19 PowerA(k) = u_0 - u(k);  
20 PowerB(k) = u_0 + u(k);  
21  
22 if online  
23     motorA.Speed = PowerA(k);  
24     motorB.Speed = PowerB(k);  
25  
26 start(motorA)  
27 start(motorB)
```

Det er nå vi adderer basispådraget med *PID* pådraget. Grunnen til at vi bytter fortegn foran $u(k)$ er for å ha en glatt sving. Man kunne bare ha hatt $u(k)$ som pådrag for en av motorene, men dette ville ha ført til veldig hakket og unøyaktig kjøring. Vi setter deretter de forskjellige *power*-ene som farten til motorene. Videre er det viktig å legge merke til hvilken av motorene man bytter fortegn til. Banen er laget slik at det er lave lysverdier til høyre som blir høyere jo mer man kommer

5.1 PID kjøring

til venstre. Dette betyr at roboten skal svinge til venstre, når lysverdiene er for lave, og høyre da dem er for høye. A motoren må derfor være motoren til høyre, og B motoren må være til venstre.

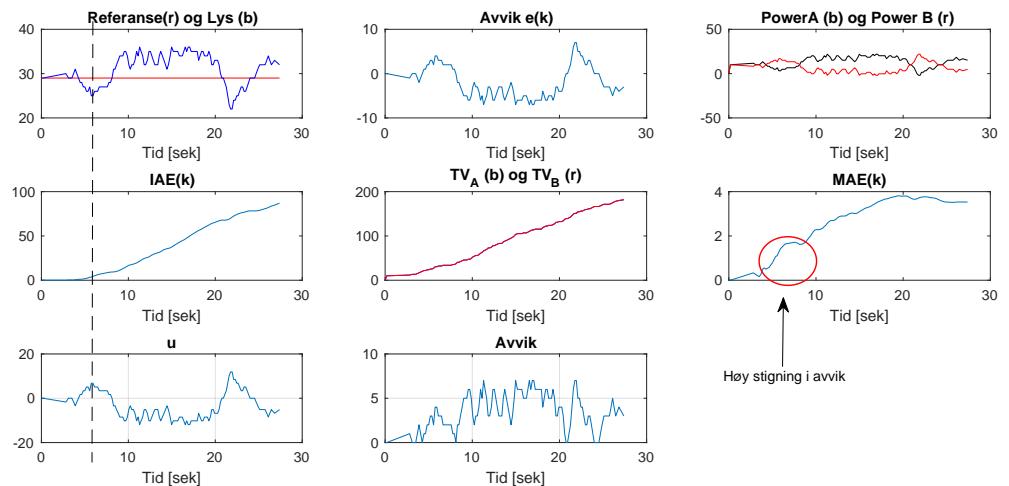
5.1.4 Resultat

I dette kapittelet blir resultatene fra testingen av *PID*-regulatoren presentert. Først blir *P*-regulatoren presenteret, hvor vi vil videre henvise til begrensingene ved å bare bruke *P*. Deretter vil vi presentere *PI*-regulatoren, og tilslutt vil vi konkludere med å vise *PID*-regulatoren og sammenligne resultatene av alle tre. Resultatene består av en rekke kvalitetsmål og målinger som beskriver nøyaktigheten av kjøringen. For å stille inn regulatoren brukte prøve- og feilemetoden.

Vi bruker $u_0 = 10$ i testingen av alle regulatorene.

P-regulatoren

P-regulatoren er det den enkleste regulatoren, hvor pådraget består kun av koden i linje 28 i 5.3. Resultatet ble slik:



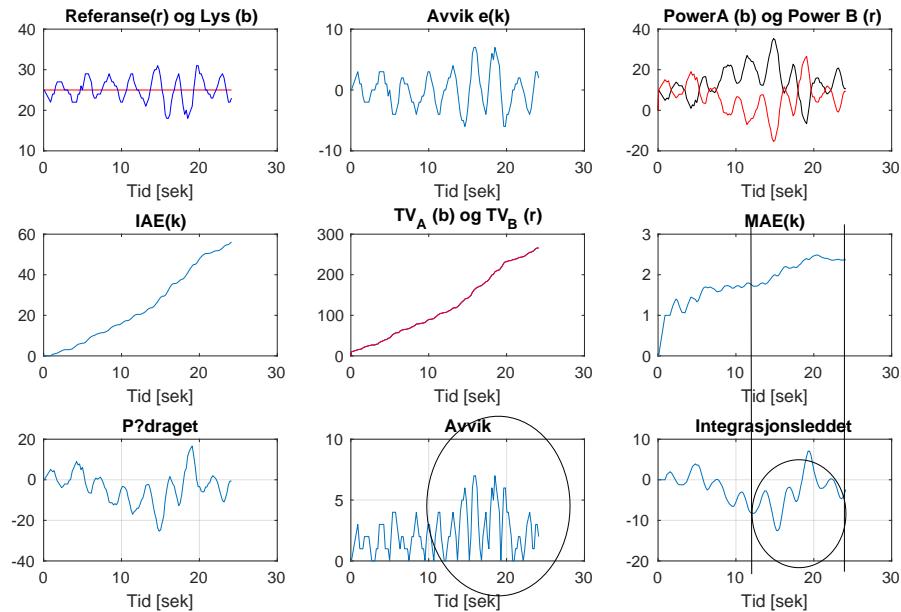
Figur 5.1: Målingene til *P*-regulatoren

5.1 PID kjøring

Når vi ser på målingene til P-regulatoren, kan vi raskt se at den gjorde en god jobb iforhold hvordan vi gjorde i kapittel 4, men vi ser også at det er rom for forbedring. Det besterresultatet vi fikk var når $K_p = 1.7$. Hvis vi ser på den stiplette linjen til høyre, ser vi en av svakheten til P. Ettersom den bare ser på reguleringssavviket i nåtid, vil små avvik ikke bli rettet raskt nok. Dette fører videre til et problem hvor avviket må være en passe størrelse før det er nok for P-regulatoren å rette det opp. Vi kan se dette der den stiplete linjen er. Det er først når avviket når et bunn- eller toppunkt at pådraget blir stort nok til å rette. Når vi ser på sirkelen rundt MAE verdien når dette skjer, er det også en sterk stigning, noe som indikerer en feilkilde.

PI-regulator

Ved å gjennomføre et likt eksperiment som i P-regulatoren fikk vi resultatene som er vist i 5.2. $K_p = 1.7$ og $K_i = 1.7$ Resultatet så slik ut:



Figur 5.2: Målingene til PI-regulatoren

5.1 PID kjøring

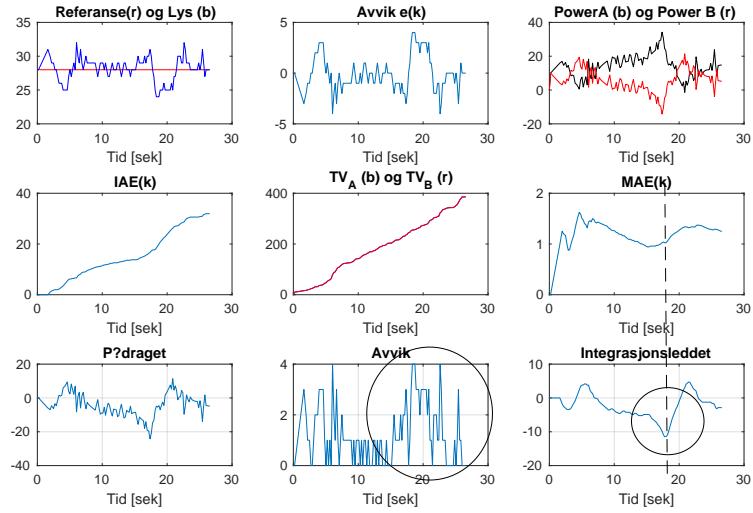
Utifra målingene til Kvalitetsmålingene gitt i 5.2 kan vi se at MAE er mindre enn i 5.1. Dette impliserer at implementeringen av integrasjonsleddet forbedret kjøringen.

Utover dette kan vi fortsatt se store avvik i målingene. Vi tar utgangspunkt i tidsintervallet $10 < Tid < 20$. I dette intervallet ser vi en sterk økning i MAE verdien, noe som indikerer feil i kjøring. Videre kan vi se på integrasjonsleddet og se hvor avviket kom fra. Tidsintervallet er nemlig satt i U-svingen i banen, og viser svakheten til integrasjonsleddet. Integrasjonsleddet takler svinger dårlig fordi det er en stor verdiendring. Når det kommer store verdiendringer, og integrasjonsleddet er stilt inn til små verdiendringer og er negativ(som vist ved målingene) må roboten komme seg på andre siden av referansemålingen for å komme seg nærmere null. Vi ville vanlig brukt det som kalles en integratorbegrensning(*anti windup*) for å stoppe dette, men vi brukte det ikke fordi derivasjonssleddet gjør opp for mye av dette i seg selv.

PID-regulator

Når vi legger til D-leddet, legges det til en verdi som tilsvarer endringen av avviket. Dette vil si at derivasjonssleddet forutsier endingen som vil skje og regulerer for den. Hvis vi igjen ser på intervallet $10 < Tid < 20$ i 5.3 sammenlignet med 5.2 ser man at avviket er mye mer stabilt. Grunnen til den økte stabiliteten er at derivasjonssleddet ser endringen som kommer til å komme i form av U-svingen, og regulerer for den. Dette motvirker svakheten til integrasjonsleddet, og minker MAE. Etter addisjonen av d-leddet, har MAE verdien minket med 50 %.

5.1 PID kjøring



Figur 5.3: Målingene til PID-regulatoren

Koeffisientene er her lik $K_p = 1.7$, $K_i = 1.7$ og $K_d = 0.85$. Vi kan se i de røde sirkelene som er laget ved at det fortsatt er rom for forbedring, men det er bare mer eksperimentering rundt verdiene man gir koeffisientene. MAE er her lik 1.1. Videre kan det informeres om at dette resultatet kan forbedres mer, ettersom vi tror at kalibreringen av koeffisientene kan fortsatt forbedres.

5.1 PID kjøring

Tabell 5.1: Sluttresultater fra manuell kjøring til deltagere i gruppe 2067. De beste verdiene er markert med fet tekst og en stjerne

	P	PI	PID
Plattform	MacBook Pro	MacBook Pro	MacBook Pro
Strømkilde	Batteri	Batteri	Batteri
POX_F6_PlottData.m	Innenfor	Innenfor	Innenfor
Referanse	29	25	28
middelverdi μ	31.0	25.1	28.03*
$ \text{Referanse}-\mu $	2.0	0.1	0.03*
standardavvik σ	$4.8 \cdot 10^{-15} *$	$1.6 \cdot 10^{-14}$	$1.8 \cdot 10^{-14}$
kjøretid [sek]	27.4	24.1*	26.6
IAE	87.0	56.1	31.9*
MAE	3.5	2.4	1.2*
TV_A	181.7	265.9	385.8
TV_B	181.7	265.9	385.8
middelverdi av T_s [sek]	0.2000*	0.33	0.2125
antall målinger (k)	138	123	126

Vi kan se utifra tabellen at det skjer forbedringer for hver gang vi legger til et av leddene. MAE-målet reduseres hver gang, noe som refererer til høyere kvalitet av kjøring. Selvsagt så er jo PID-regulatoren best i de fleste kvalitetsmålene, men det er også det mest stabile hver gang vi kjørte programmet.

5.2 Cruise Controll

5.2 Cruise Controll

5.2.1 Problemstilling

I dette prosjektet har målet vært å lage en *cruise controll*. Funksjonen til en *cruise controller* er å opprettholde av en gitt avstand fra bilen foran, og bremse når denne avstanden minker, og gasse når avstanden øker.

For å få til dette brukes en ultra-sonisksensor for å måle **avstand** fra bilen foran og en PID-regulator som skal regulere avstanden, ved å påvirke **pådraget** slik at avstanden mellom bilene alltid er lik den ønskede avstanden.

Prosjektet om som er beskrevet/dokumentert i dette kapittelet har gått ut på å lage en pådrags-regulator, dette gjelder kun med fram og tilbake bevegelser, og roboten kan ikke svinge hvis objektet foran svinger, ettersom dette krevde en eksra ultra-lydsensor, noe vi ikke hadde.

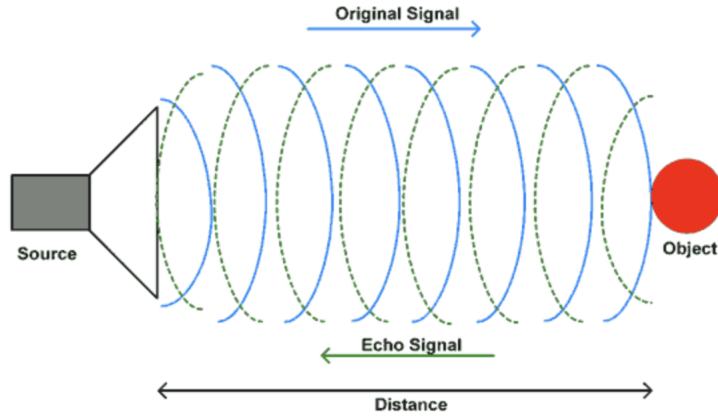
5.2.2 Teori

Teorien som brukes i dette kapittelet går mest ut på å finne avstanden til objektet foran, da vi allerede har gått igjennom teorien om PID-kontrolleren i forrige kapittel.

Ultralydsensor

En stor del av dette prosjektet handler jo om hvilken avstand roboten er ifra objektet den skal følge, det er derfor viktig å vite hvordan vi gjør det. Ultralydsensoren sender ut lydbølger som spretter av objektet foran og returnerer til sensoren som et ekko signal, den vil da regne ut avstanden til objektet foran ved å se på hvor sterkt lydsignalet er. Desto sterkere lydsignal, desto nærmere er objektet og vice versa.

5.2 Cruise Controll



Figur 5.4: Hvordan en ultralydsensor fungerer.

[2]

PID-regulator

En PID regulator er brukt for å regulere pådraget til motoren slik at vi kan endre avstanden fra roboten til objektet til et gitt ønsket avstand. Dette innebærer også at roboten rygger om den er for nærmee, noe som også er tatt med i resultatet.

5.2.3 Løsningsforslag

Løsningen er da som følger

- Vi setter en referanseavstanden som tilsvarer den ønskede avstanden fra objektet
- Vi lager et reguleringsavvik som tilsvarer differansen mellom referanseavstanden og den nåværende avstanden.
- Vi setter reguleringsavviket inn i en PID-regulator avgjør pådraget til motorene
- Motorene gis en verdi fra -100% til 100%

5.2 Cruise Controll

- Ny måling

Dette er vist med denne koden:

Kode 5.5: Cruise controller PID-kode.

```
12 Kp=10.0;
13 %-----
14 Ts(k-1)=Tid(k)-Tid(k-1);
15
16 IAE(k)= IAE(k-1)+abs(e(k-1))*Ts(k-1);
17
18 MAE(k) = sum(abs(e(1:k)))/k;
19
20 middelverdi = sum(Avstand(1:k))/k;
21
22 standardavvik = sqrt((sum((Avstand(1:k))-middelverdi).^2)/k);
23
24 P(k)=Kp*e(k);
25
26 u(k)=P(k);
```

Etter et raskt øyekast vil man raskt merke at det eneste leddet i PID som er brukt her er Proporsjonalitetsleddet, da implementasjonen av I og D, fikk roboten til å gå fram og tilbake veldig mye, istedet for å stå i ro når den hadde nådd det ønskede avstanden. Koden som tar opp målingene er gjort som dette:

Kode 5.6: Cruise controller Målinger.

```
22 Avstand(k) = (100/2.55)*double(readDistance(mySonicSensor));
23 e(k) = Referanse(1) - Avstand(k);
24 Referanse(k) = Referanse(1);
25 Avvik(k) = abs(Referanse(1) - Avstand(k))
```

Grunnen til vi skalererte avstanden til 1-100 er for å gjøre det enklere å kalibrere PID-regulatoren.

5.2.4 Resultat

I dette kapitellet blir resultatene av testingen av cruise controller-en presentert. Først presenteres resultatene fra testingen, for så å vurdere resultatene som ble

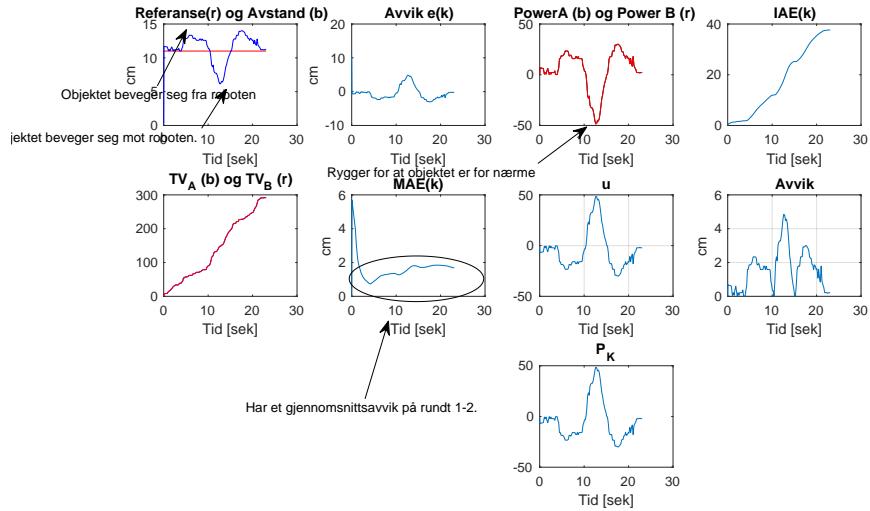
5.2 Cruise Controll

målt.

Resultatene består av tidsrespons kurver som viser hvordan regulatoren klarer å opprettholde den ønskede avstanden, i tillegg til kvalitetsmål som vurderer effektiviteten til regulatoren.

Målingene

I 5.5 kan man se i hvilken grad *cruise controll*-eren fungerer.



Figur 5.5: Resultatet fra *cruise controll*

Vi kan se via MAE målet at regulatoren kun har et gjennomsnittsavvik på 1-2 cm. I praksis så vi også at dette stemte. Videre ser vi at regulatoren reagerer veldig raskt til forandringer, dette er nok ettersom vi har satt $K_p = 10$. Raskere reaksjonstid fører til mindre dødtid og generelt lavere avvik. De store avvikene man ser, kommer av en rask verdiforandring som regulatoren ikke er rask nok til å oppfatte. Dette kommer av store tidsskritt mellom hver måling(Gjennomsnitts tidskrittet var på 0.1874 sekund) der objektet beveget seg nærmere uten av roboten fikk målt dette. Tidsskrittet var stort ettersom vi plottet mens vi kjørte programmet, dette tar mye datakraft og minker derfor antall målinger roboten

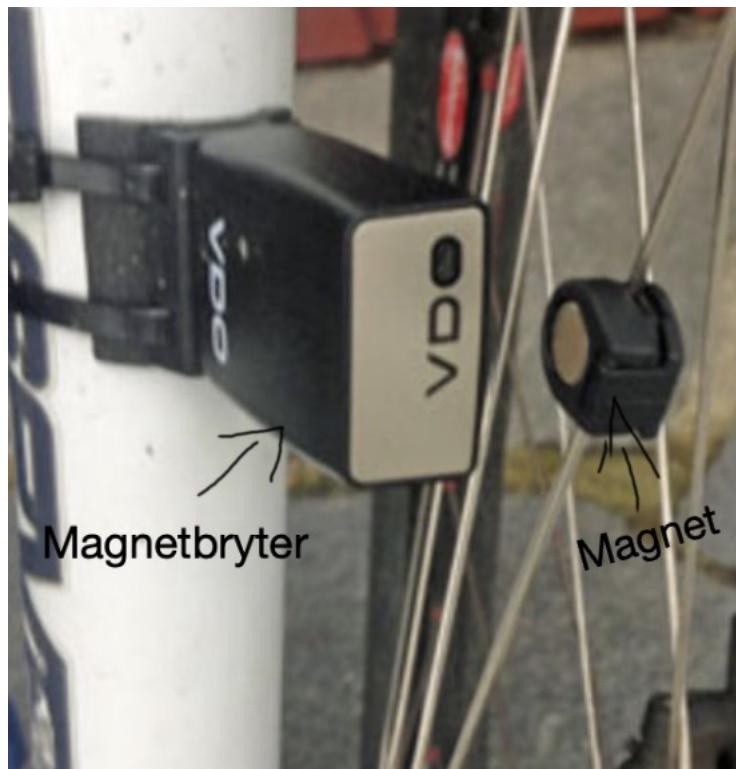
5.2 Cruise Controll

tar hvert sekund.

5.3 Sykkelcomputer

5.3.1 Problemstilling

I dette prosjektet har målet vært å lage en sykkelcomputer. Målet var å bruke en bryter for å registrere at hjulet hadde rotert en runde. Dette ville simulere hva som ellers ville ha vært en magnet montert på hjulet og en magnetbryter på gaffelen som registrerer når magneten passerer og dermed at hjulet har går rundt. Se bildet i figur 5.6.



Figur 5.6: Henvist på bildet ser vi hvordan en vanlig sykkelcomputer funker. Det kan være mer enn bare en magnet, da dette gir økt nøyaktighet.

[1]

Et trykk på bryteren skal da representere en rundgang på hjulet. Vi skal bruke dette for å regne ut **avstand**, **momentan-** og **gjennomsnittshastighet**(gitt ved en filtrert avstand), og **akselerasjon**.

5.3 Sykkelcomputer

5.3.2 Teori

I dette delkapittelet skal vi gå igjennom teorien rundt en sykkelcomputer.

Avstand

Måten vi regner ut avstanden i sykkelcomputeren, følger denne formelen.

$$\sum_{n=1} Bryter(n) \cdot Hjuldiameter \cdot \pi \quad (5.5)$$

Bryter matrisen består av 0-ere og 1-ere. Når vi summerer hele matrisen sammen, får vi da antall ganger hjulet har rotert. Når vi vet antallet rotasjoner siden begynnelsen, kan vi ta gange dette med hjulomkretsen ($Hjuldiameter \cdot \pi$) for å få avstanden.

Momentan- og gjennomsnittshastighet

Å finne farten, om det er momentan- eller gjennomsnittshastigheten, så handler det om å finne endringen i avstanden. Men i dette tilfellet hvor vi oppdaterer avstanden men en gitt verdi hver gang, så vil det jo være en stor endring for hver gang vi trykker på bryteren. Konsekvensen av dette vill da være at momentanfart grafen blir feil iforhold til den sanne farten. Vi bruker derfor et filter for at endringene ikke skal være så skarpe, som igjen fører til at farten blir mer sann. For å regne ut momentanfarten deriverer vi avstanden for å finne endringen. Eksempler på dette er gitt i kapittel 3 og 6.1, hvor vi deriverte avstanden for å finne farten. Momentanhastigheten er da gitt ved

$$\frac{d}{dt}(Avstand(t)) \quad (5.6)$$

Gjennomsnittshastigheten derimot er veldig enkel å komme fram til, og er gitt ved:

$$\frac{Avstand(1) + Avstand(t)}{Tid(t)} \quad (5.7)$$

5.3 Sykkelcomputer

Akselerasjon

Akselerasjonen til sykkel er endringen i farten. Vi skal finne momentan- og gjennomsnittsakselerasjonen. Vi deriverer derfor momentanfarten for å finne aksele- rasjonen, og tar gjennomsnittet av gjennomsnittsfarten. De er gitt slik:

$$\frac{Fart(1) + Fart(t)}{Tid(t)} \quad (5.8)$$

Dette gir gjennomsnittsakselerasjonen.

$$\frac{d}{dt}(Fart(t)) \quad (5.9)$$

Dette gir momentanakselerasjonen.

5.3.3 Løsningsforslag

Selve utfordringen med å kode en sykkelcomputer ligger ikke i hvordan man skal regne ut fart eller akselerasjon. Problemer ligger i hvordan man bare får et trykk på bryteren, til å bare telle som et trykk.

Når trykksensoren trykkes inn gir den 1 som output heilt til den presses ut. For at sykkelcomputeren skal fungere, må et trykk bare gi en verdi. Løsningen til problemet ble som visst i 5.7

Kode 5.7: Trykk knapp kode

```
26 while double(readTouch(myTouchSensor)) == 1
27     if double(readTouch(myTouchSensor)) == 0
28         break;
29     end
30 end
```

Tankegangen bak koden er å etterligne en *until* kommando. Dette vil si vi sier til koden at hvis vi finner en 1 verdi, så kan det ikke komme en ny 1 verdi før det har er registrert en ny 0 verdi. Dette vil si at den ikke tar opp signaler før knappen er presset ut.

Resten av koden er som følger:

5.3 Sykkelcomputer

Kode 5.8: Trykk knapp fartskode

```
30 Omkrets=3.14*0.740;
31 alfa=0.3;
32 Avstand(k) = sum(Bryter(1:k))*Omkrets;
33 Avstand_IIR(k) = IIR_filter(Avstand_IIR(k-1),Avstand(k),alfa);
34 Momentanfart(k) = Derivasjon(Avstand_IIR(k-1:k),Tid(k)-Tid(k-1));
35 Gjennomsnittsfart(k) = Avstand(k)/Tid(k);
36 Momentanakselerasjon(k) = ...
    Derivasjon(Momentanfart(k-1:k),Tid(k)-Tid(k-1));
37 Gjennomsnittsakselerasjon(k) = Momentanfart(k)/Tid(k);
38 Momentanfart_u(k) = Derivasjon(Avstand(k-1:k),Tid(k)-Tid(k-1));
```

I 5.8 gjør vi alle omregningene for å finne farten og akselerasjonen. Det er brukt de samme omregningsmetoden som visst i teori delen av kapittelet.

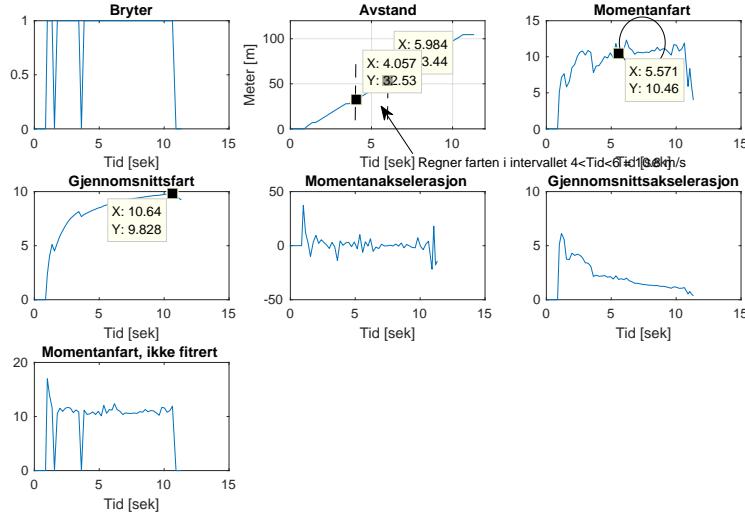
5.3.4 Resultat

I dette delkapittelet blir resultatene fra testing av sykkelcomputeren presentert. Først presenteres målingene og omregningene, følgt på en igjenomgang rundt betydningen av disse.

Målinger

En viktig ting som påvirker målingene og gjør dem presentable er filtreringen av avstanden. Men selv om målingene er filtrerte, er det fremdeles et hint av støy i grafen som kommer av den sterke verdiendring knappetrykket fører til. Dette vises i 5.7

5.3 Sykkelcomputer



Figur 5.7: Bildet viser til målingene gjort ved å bruke trykksensoren slik som å simulere en konstant fart på rundt 10m/s. Vi kan se filtreringen funke bra på gjennomsnittsfarten, men momentanfarten er noe volatil iforhold til den sanne farten.

I målingene kan vi se alle verdiene som kommer fra trykksensoren. Vi har valgt oss målingene i tidsintervallet $4 < Tid < 6$ som regne verdier for å se hvordan fartsgrafene er i forhold til den sanne farten. Som visst i grafen, ser vi at farten i det gitte tidsintervallet er ... m/s. Når vi sammenligner dette med verdiene vi får i momentan- og gjennomsnittsfart målingene, ser vi at det kan stemme med en viss feilmargin. Momentanfarten er som sagt litt volatil med tanke på at verdiendringen i avstand er bratt, men vi ser fremdeles at selve grafen er rundt $10m/s$ med en $+/- 1m/s$. Derimot når vi ser på gjennomsnittsfarten kan vi se filtreringen godt. Grafen kan sammenlignes med 2.7, og verdiene er nærmere den sanne farten enn momentanfarten.

Akselerasjonsgrafene skal være lik 0 desto lengre vi mäter, dette kan vi derimot se på grafene. Momentanakselerasjonen holder seg rundt $0m/s^2$, og gjennomsnittsfarten er på mot $0m/s^2$, men er litt tregere ettersom filteret drar ut verdiendringen.

Kapittel 6

Kreativ del Små prosjekter

6.1 Fart

6.1.1 Problemstilling

I dette delkapitelet ønsker vi å beregne ut farten. Dette skal vi gjøre ved bruk av lydsensor og motorrotasjon. Dette er en deloppgave fra prosjektbeskrivelsen..., oppgave 16 side 122. Målet med oppgaven er å få et så likt fartsresultat mellom motoren og lydsensoren.

6.1.2 Teori

Fart teoremet

For å kunne beregen ut hastighet er det viktig å ha litt kunnskap om relasjonen mellom avstand og fart. De fleste av oss vet at fart er det samme som strekning delt på tid. Men den deriverer med hensyn på tiden, av avstanden er også det samme som farten til et legeme. Det vil si:

$$\frac{d}{dt}(v * t) = v \quad (6.1)$$

6.1 Fart

Hvor v er målt i [cm/s].

Implementeringen av andre prosjekt

Utførelsen av dette prosjektet er relativt likt som konseptet i kapitel 3. Den reelle forskjellen er at vi nå ønsker å regne ut farten av fysiske målinger, i stedet for simulerte. Vi valgte også å implementere denne koden i kapitelet om manuell kjøring, dette er siden vi kan på denne måten styre hastigheten selv. Du finner mer om koden brukt i manuell kjøring i kapitel 4.

Ultrasonisksensor/Lydsensor



Figur 6.1: Bilde av en ultrasolisksensor

Lydsensoren er et av verktøyene våre i denne oppgaven. For å vite hvordan man kan anvende verdiene mottatt av lydsensoren må vi først vite hva de betyr. Måten lydsensoren virker på, er ved at den sender ut en gitt frekvens av lyd som den igjen plukker opp. Ved å gjøre dette klarer den ved bruk av lydens hastighet og det reflekterte lydsignalet å determinere avstanden. Det vil altså si, at vi vil ikke

6.1 Fart

kunne finne farten, ved hjelp av lydsensoren dersom det ikke er et objekt foran sensoren.

Sensoren har en begrensning i avstand, etter vår testing har vi kommet fram til at den måler kun avstanden opp til 2.55 meter. Alt utenfor denne avstanden klarer ikke lydsensoren å plukke opp. Vi testet også hvor god lydsensoren er til å plukke opp egne signal. Siden vi vet at sensoren reagerer på lyd, prøvde vi å interferere med lydsignalet. Dette gjorde vi med å lage mye støy rundt den. Dette påvirket ikke verdiene fra sensoren i våre forsøk.

Noe som vi merket derimot var at den ultrasoniske sensoren hadde en del *spikes*. For å rette opp i dette valgte vi å bruke ett *IIR*-filter for å få mer stabile verdier.

Motorverdier



Figur 6.2: Bilde av en motor

Motoren måler noen viktige målinger dersom vi ønsker å finne farten. Den viktigste målingen er motorrotasjon. Den forteller oss hvor mange grader hjulet har rotert. Det positive med å regne ut motor rotasjon er at vi slipper å filtrere verdiene våre.

6.1 Fart

6.1.3 Løsningsforslag

De første målingene vi trenger er avstanden. Når det kommer til den ultrasoniske lydsensoren måler den avstanden til et objekt innenfor 2.55 meter. Det vil si at avstanden vil minke desto nærmere maskinen beveger seg mot dette objektet.

Når vi regner fart ønsker vi å finne ut den tilbakelagte strekningen. For å finne den ut med lyssensoren blir det slik

$$0 - S = -S \quad (6.2)$$

hvor utgangspunktet ditt er 0 og S er strekningen til objektet. Det vil hovedsakelig bety at vi kan bare multiplisere ligningen med -1 for å finne tilbakelagt strekning.

For å finne farten nå må vi derivere vår tilbakelagte strekning. Heldigvis er dette lett, siden vi har laget en funksjon for derivasjon. Denne finner du her3.3.3. Formelen vår blir da slik.

$$-S = -v \cdot t \quad (6.3)$$

som blir derivert:

$$-S' = -\frac{d}{dt}(v \cdot t) = -V \quad (6.4)$$

For å finne farts verdien vi ønsker, tar vi da absolutt verdien av farten vår:

$$V = |-V| \quad (6.5)$$

Som nevnt tidligere i delkapitelet, lønner det seg å ha et IIR filter på. Ved hjelp av funksjonen fra kapitel 2.

Når vi bruker IIR filteret vårt i matlab vli funksjonen for strekning se slik ut:

$$Avstand_{filter}(k) = IIR_{filter}(Avstand_{filter}(k-1), Avstand(k), alfa) \quad (6.6)$$

Hvor alfa er filtreringskonstanten som bestemmer hvor mye filtrasjon du ønsker å gjennomføre på dataene dine.

6.1 Fart

Det er den filtrerte verdien vi ønsker å derivere for å finne farten: da blir uttrykket:

$$V_{Sonisk_f} = \text{abs}((\text{Derivasjon}(Avstand_{filter}(k - 1 : k), Ts(k - 1))) * 100) \quad (6.7)$$

Grunnen til at vi multipliserer absolutt verdien til farten vår, er siden vi ønsker svaret vårt i [Cm/s].

Når vi skal finne ut farten ved hjelp av motorene må dette gjøres ved å se på rotasjonene. Vi kjørte først en omdreining for å se hva verdien på rotasjonen var. Da fikk vi en verdi på 360, altså at en omdreigning er 360 grader.

For å finne ut distansen motoren beveger seg i, måtte vi regne ut omkretsen av dekket.

$$O = 2\pi r \quad (6.8)$$

Vi fant $2r$ ved å tegne en sirkel rundt et av hjulene. Deretter krysset vi en linje gjennom sirkelen og målte lengden dens. vi fant ut at diameteren, eller $2r$ er 5.2cm

det vil si at avstanden er gitt ved

$$S = 2\pi r \cdot \frac{1}{360} \cdot \text{graderrørt} \quad (6.9)$$

Føringen av dette i matlab blir slik:

$$Avstand_{motor}(k) = VinkelPosMotorA(k) \cdot \frac{5.6 \cdot \pi}{360} \quad (6.10)$$

Dette gir oss avstanden, men målet vårt er å finne farten, da gjør vi akkurat det samme som vi gjorde for å finne farten fra den ultrasoniske.

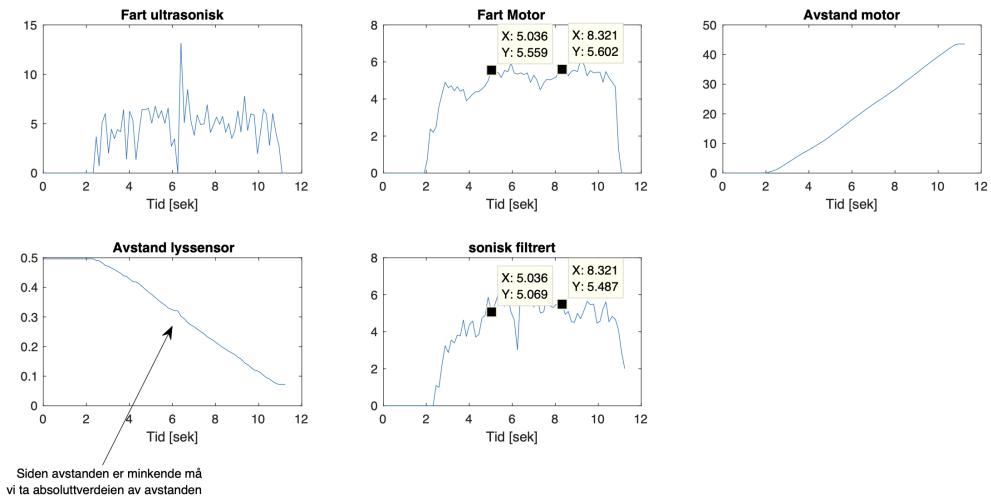
$$V_{motor} = \text{Derivasjon}(Avstand_{motor}(k - 1 : k), Ts(k - 1)) \quad (6.11)$$

Hvor Derivasjoner funksjonen vi lagte i kapitel 3, og Ts er enda tidsskrittet fra ligning 3.4

6.1 Fart

6.1.4 Resultat

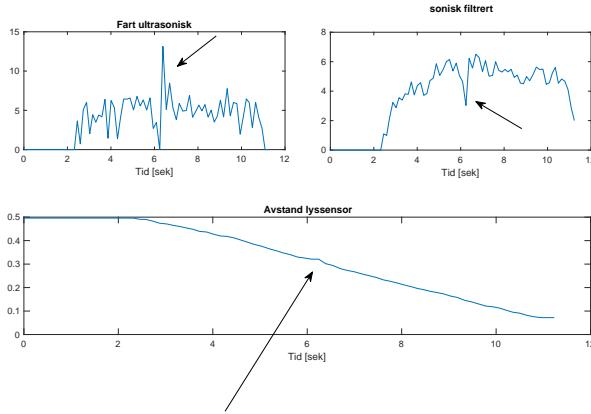
For å verifisere at ligningene våre er korrekte har vi plottet en graf som viser med filtrering, og en uten. Vi har også plottet avstanden til både motoren og den ultrasoniske.



Figur 6.3: Strekningen er gitt i Cm, og farten er gitt i [Cm/s]

Det første man legger merke til er hvor ujevn farten er når den er ufiltrert. Det er derfor vi filtrerer verdiene. For å gjøre dette implementerer vi et *IIR*-filter. Etter vi har gjort dette vil grafen se tilsvarende lik ut farten vi fikk fra motorene.

6.1 Fart



Figur 6.4: Pilene viser til hvor det er en *spike*. Dette kan være ytre forstuleser som påvirker frekvensen til lydsensoren. Avstand målt i meter, fart målt i [Cm/s]

Som vi ser fra grafene over har vi en relativt stor *spike*. Grunnen til at den er så stor i fartsgrafene men ser liten ut i avstand. Dette er siden akselerasjonen i dette partiet er ganske høyt. en annen faktor er at derivasjonen kan ha ført til en sekant som ble plassert litt ubeleilige. Heldigvis ser vi at mye av denne *spiken* blir filtrert bort, noe som demonstrerer viktigheten av filtrering.

Vi har satt to punkter i begge fartsgrafene. en på $x = 5.0$ og en på $x = 8.3$. Vi ser at selv om begge hastighetene er relativt like er de ikke identiske. Dette skyldes mange faktorer.

Vi kan ta utgangspunkt i motor farten som en reell fart, dersom vi har rett diameter til hjulene. Derfor er det mer interessant å se på hvordan og hvorfor det er et avvik i hastigheten vår fra den ultrasoniske.

Mye av avviket til den ultrasoniske skyldes filtreringen. Filtreringen retter opp mange uhamslige verdier, men den tar også å filtrere bort enkelte nøyaktige verdier.

6.1 Fart

Numerisk derivasjon, er heller aldri helt absolutt rett verdi, noe som fører til litt avvik.

6.2 Trapes metoden og *EulerForward* metoden

6.2 Trapes metoden og *EulerForward* metoden

6.2.1 Problemstilling

I kapitel 1 gikk vi litt overfladisk gjennom differansen mellom *EulerForward* og trapes metoden. I dette forsøket ønsker vi å demonstrere hvor mye differansen mellom trapes metoden og *EulerForward* metoden er på.

6.2.2 Teori

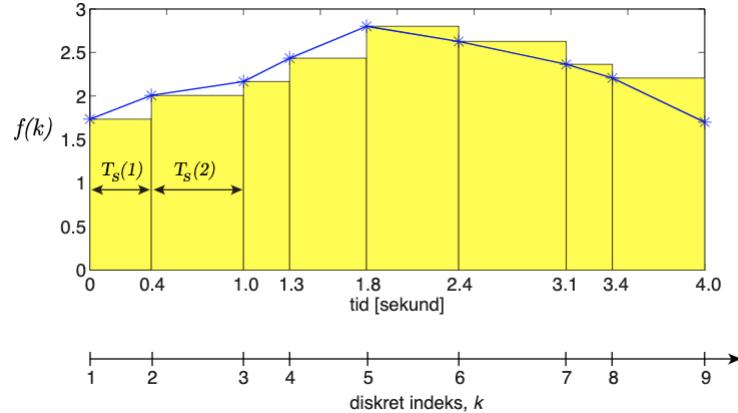
Både trapesmetoden og *EulerForward* er former av numerisk integrasjon. Numerisk integrasjon er den eneste måten å få et integral av en funksjon i sanntid. Nedturen med numerisk integrasjon er at den gir nærmere et estimat av integralet, framfor det reelle svaret. Den eneste måten vi kan få et reellt svar på, ved bruk av numerisk integrasjon er dersom antall punkter målt er uendelig. et eksakt integral av en konstant vil se slik ut:

$$\int x dx = \frac{1}{2} \cdot x^2 \quad (6.12)$$

Du finner mer om integrasjon i kapitel 1

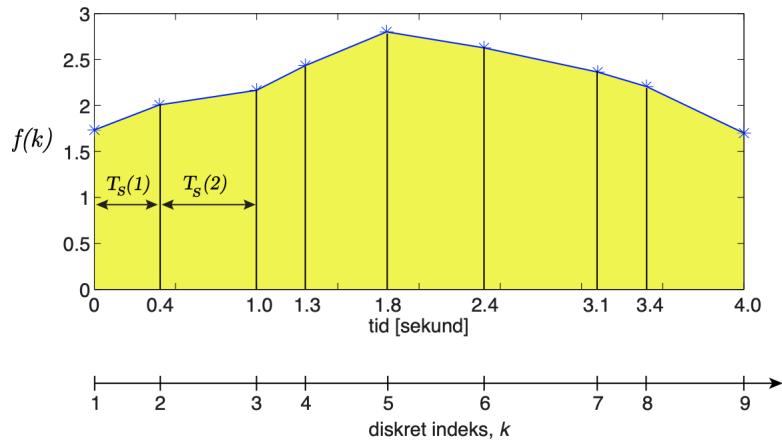
Forskjellen på trapes metoden og *EulerForward* er formen man velger å ta integralet fra. Nedenfor ser man de samme to figurene som ble brukt for å illustrere trapes metoden og *EulerForward*.

6.2 Trapes metoden og *EulerForward* metoden



Figur 6.5: *EulerForward* metoden. Figur hentet fra [3]

Dette er en fin figur som forklarer prinsippet bak *EulerForward* metoden. Man tar altså summen av de kule rektanglene, som man ser går ikke rektanglene helt opp til linjen. Dette fører til at vi vil ha et lite avvik fra det reelle integralet.



Figur 6.6: Trapesmetoden hentet fra [3]

Den største nedturen med bruk av *EulerForward* er at den tar kun i bruk gamle verdier. Det vil si at den antar alltid at målingen holder seg stabil, inntil den får en ny funksjonsverdi. Dersom man bruker trapesmetoden derimot, vil man kunne bruke vår nåværende måling, noe som gir et mer nøyaktig resultat. Nedturen med

6.2 Trapes metoden og *EulerForward* metoden

trapesmetoden er at den må gjennom flere handlinger, noe som påvirker *delay*-et.

Formelen for integrasjon ved hjelp av trapesmetoden er:

$$T = \frac{f(a) + f(b)}{2} + f(a + \Delta x) + f(a + 2\Delta x) \dots f(a + (n - 1)\Delta x) \quad (6.13)$$

Når vi skal regne ut integralet ved bruk av trapes metoden i sanntid derimot, regner vi ut ett og ett trapes hver for seg. Dette gjør vi slik

$$A_n = \frac{f(a) + f(b)}{2} \cdot \Delta x \quad (6.14)$$

Hvor $f(a)$ og $f(b)$ er funksjonsverdier. $f(a)$ er den forrige målingen, og $f(b)$ er den siste målingen. Δx er det samme som Ts verdien vår.

6.2.3 Løsningsforslag

I denne oppgaven kommer vi til å bruke samme utregning for *EulerForward* som vi gjorde i kapitel 1. Det vil si at vi kommer til å bruke ligningen:

$$\text{Integrert}(k) = \text{Integrert}(k - 1) + \text{FunksjonsVerdi}(k - 1) * \text{TidsSkrift}(k - 1) \quad (6.15)$$

Dette er den eksakte ligningen som i ligning 1.2, hvor den eneste forkjellen er at vi bruker benevningen ”*integrert*” som ”*Volum*”.

Når vi skal ta integralet ved bruk av trapesmetoden kan vi bruke formelen:

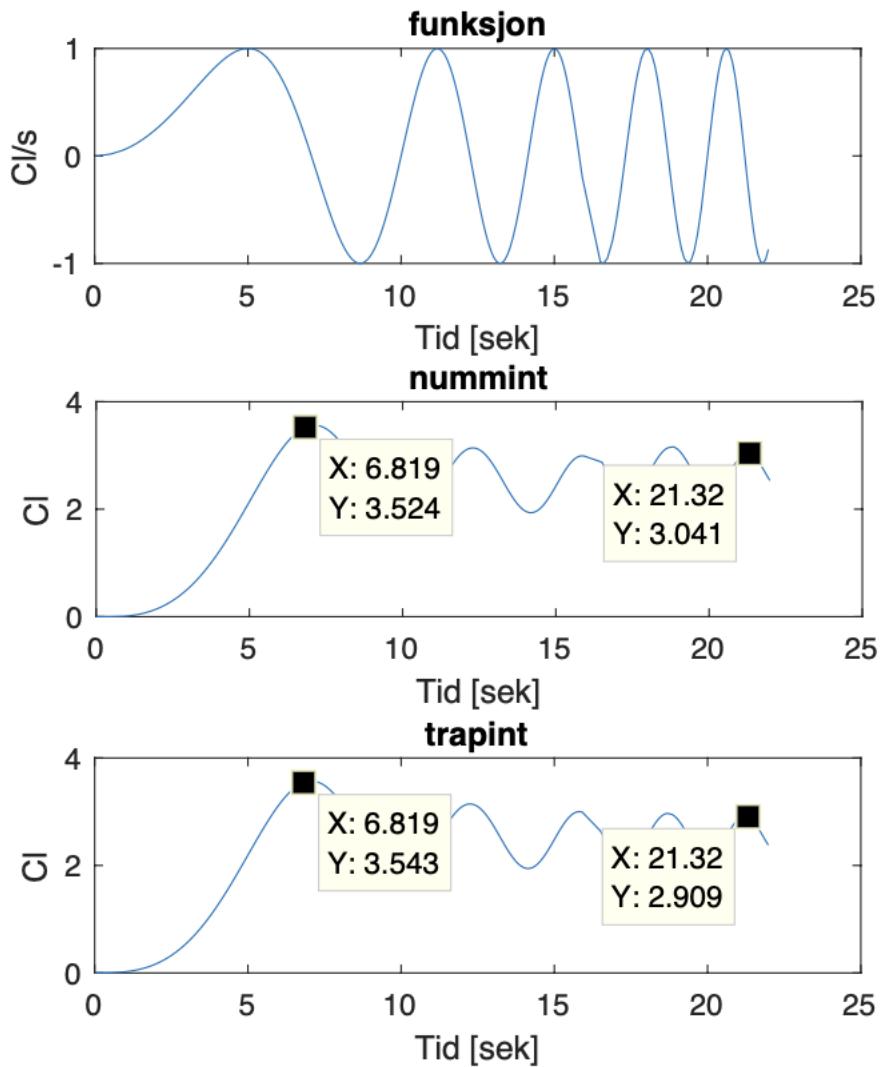
$$\text{IntValueNew} = (\text{IntValueNew}(k-1) + \text{sum}(\text{FunctionValue}(k-1 : k)) / 2) \cdot \text{Ts}(k-1) \quad (6.16)$$

6.2.4 Resultat

Vi implementerte først koden for numerisk integrasjon for i et ny mappe. Herfra la vi til formelen for trapes metoden inn, og plotter resultatene. For å teste at funksjonen vår faktisk virket, plotet vi en gitt graf i while løkken. Vi satt at

6.2 Trapes metoden og *EulerForward* metoden

$Lys(k) = Tid(k)^2$. Vi valgte å beholde ploittingen inne i while løkken, fordi det fører til færre målinger. Færre målinger gjør at differansen mellom *EulerForward* og Trapesmetoden blir vesentlig større.



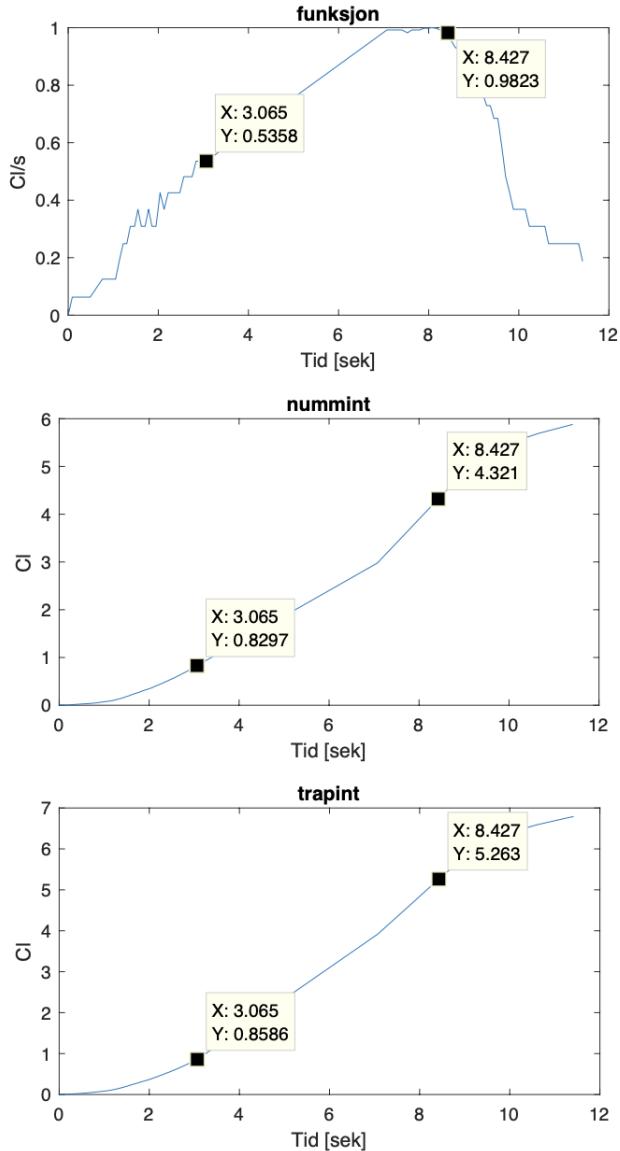
Figur 6.7: Hvordan *EulerForward* metoden virker på funksjonen vår. Reelt sett vil den ha enda flere rektangler enn det den har på bilde, men poenget er at denne metoden regner ikke ut hele arealet.

Som vi kan se ut i fra målingene gir trapes metoden en høyere funksjonsverdi,

6.2 Trapes metoden og *EulerForward* metoden

enn det *EulerForward* gjør. Dette virker realistisk siden trapesmetoden måler mer av integralet enn det *EulerForward* gjør. Grunnen til at differansen mellom funksjonsverdiene er såpass lave er grunnet antall målinger på kort tid. Vi hadde 40 målinger på 4.1 sekund, noe som gjør at *EulerForward* metoden i grunnprinsipp ikke har mye avvik uansett.

6.2 Trapes metoden og *EulerForward* metoden

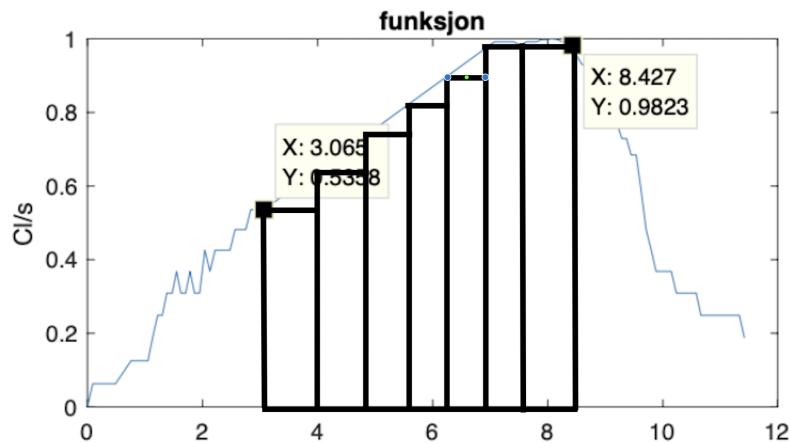


Figur 6.8: Selv om trapesmetoden ikke er perfekt, ser vi at den er mye næremere funksjonen, enn det *EulerForward* metoden er.

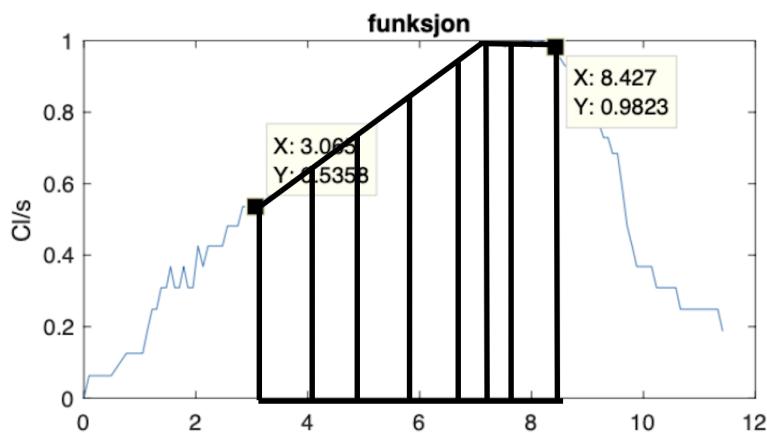
Vi ser at når $x = 3.1$ har den trapesintegrerte funksjonen en funksjonsverdi på på rundt 0.03 høyere enn det *EulerForward* har. Når vi ser på $x = 8.4$ har den

6.2 Trapes metoden og *EulerForward* metoden

trapesintegrerte en funksjonsverdi på 0.9 høyere enn det *EulerForward* funksjonen. Dette gir mening siden hvis vi ser på funksjonen øverst er det nesten en ren lineær vekst mellom $x = 3.1$ og $x = 8.4$. Det fører til at trapes metoden vil være tilnærmet det reelle arealet, hvor *EulerForward* vil mangle mellomverdien mellom målingene. Dette er best illustrert i figurene under.



Figur 6.9: Hvordan *EulerForward* integrerer funksjonen vår.



Figur 6.10: Figuren over viser hvordan integrasjon ved bruk av trapesmetoden virker.

Vi kan se ut i fra figurene over at trapesmetoden kommer nærmere funksjonen.

6.2 Trapes metoden og *EulerForward* metoden

EulerForward metoden mangler en del av arealet siden den ikke tar i betrakning den neste målingen. Det er også verdt å nevne at det er kun en illustrasjon lagt over grafen, Det er enda flere målinger enn det som vises. Grunnen til at vi viser få antall trapes/rektangler er siden vi ønsker å gi et klarere visuelt syn på hvordan grafene differensieres, og hvorfor trapesmetoden har en høyere funksjonsverdi.

Kapittel 7

Konklusjon

Det er lett å tenke at all matten vi lærer i MAT100 er matte vi ikke har bruk for senere i livet, hva er nytten bak integrasjon og derivasjon. Gjennom dette prosjektet har vi lært enormt mye om hvor vi kan bruke derivasjon og integrasjon til for å løse praktiske kompleksiteter.

Vi har sett på viktigheten av et filter demonstrert i en rekke ulike forsøk. Vi har lært ulikheten mellom et *IIR* filter og et *FIR* filter, og lært når vi skal bruke dem hver for seg

All denne kunnskapen har hjulpet oss til å gjennomføre automatisk kjøring og *cruise controll*. Vi har fått ett innblikk i hvordan matten løser problemene rundt oss på dagligbasis. Samtidig som vi kan forstå og reflektere rundt relevante problemstillinger i den moderne verden. Bensinpumper, fartsmåler og/eller temperaturmålere er eksempler på dagligdagse utfordringer dette prosjektet hjalp oss å forstå.

Vi lærte hvordan en PID-regulator kan opprettholde en ønsket verdi, ved at vi kalibrerer forskjellige koeffisienter til det gir oss et ønsket utfall. Dette prinsippet bruker vi i automatisk kjøring og *cruise controller* for å minimere avviket.

Tilslutt har vi lært ulike måter å anvende sensorer på for å finne samme resultat. Dette er fartsoppgaven vår et eksempel på, hvor vi var nødt til å ta i bruk derivasjon og filtrering for å kunne løse dette problemet som virker så alminnelig.

Bibliografi

- [1] Sigma bc 12.12 sts mot vdo x1dw. <https://www.dinside.no/fritid/sigma-bc-1212-sts-mot-vdo-x1dw/61270454>.
- [2] Ultrasonic module hc-sr04. <https://www.electronicwings.com/sensors-modules/ultrasonic-module-hc-sr04>.
- [3] T. Drengstig. Lego mindstorms og matlab; anvendt matematikk og fysikk i skjønn forening. Technical report, Universitet i Stavanger, 2019. Utdelt materiale.

Vedlegg A

Timelister

Tabell A.1: Timelister for gruppe 2067 i faget ING100.

Uke	Marius	Thomas
40	5	3
41	22	5
42	17	13
43	18	4
44	27	18
45	46	45
46	44	44
Sum	179	132

Vedlegg B

Programlisting prosjekt 04

B.1 P04_F4_MathCalculations.m

Kode B.1: *P04F4Mathcalculations1*

```
1 %%%%%%
2 % P0X_F4_MathCalculations.m
3 %
4 % Her programmerer du beregninger som skal gj?res
5 %
6 %%%%%%
7
8 %-----%
9 % Definer parametre som brukes i beregningene.
10 %-----%
11
12
13 Ts(k-1)=Tid(k)-Tid(k-1);
14
15 IAE(k)= IAE(k-1)+abs(e(k-1))*Ts(k-1);
16
17 MAE(k) = sum(abs(e(1:k)))/k;
18
19 middelverdi = sum(Lys(1:k))/k;
20
```

B.2 P04_F5_CalculateAndSetMotorPower.m

```
21 standardavvik = sqrt((sum((Lys(1:k))-middelverdi).^2))/k);
```

B.2 P04_F5_CalculateAndSetMotorPower.m

Kode B.2: *P04_F5_CalculateAndSetMotorPower.*

```
1 %%%%%%
2 % POX_F5_CalculateAndSetMotorPower.m
3 %
4 % Beregner hvordan motorene skal bevege seg
5 %
6 %%%%%%%%%%%%%%
7
8
9 %-----
10 % Definer parametre til ? beregne motorp?drag.
11 %
12 a = 0.5;
13 b = 0.3;
14
15 %-----
16 % Beregner motorp?drag og lagrer i datavektor
17 % (slett de motorene du ikke bruker og lag egen kode)
18 %
19 PowerA(k) = a*-JoyForover(k) + JoyLR(k);
20 PowerB(k) = a*-JoyForover(k) - JoyLR(k);
21
22
23 if online
24
25 %
26 % Setter powerdata mot EV3
27 % G?r ikke fortore enn -100 -> +100 selv
28 % om powerverdi er st?rre
29 % (slett de motorene du ikke bruker)
30 %
31
32
33 motorA.Speed = PowerA(k);
34 motorB.Speed = PowerB(k);
35
36 TV_A(k) = abs(diff(PowerA(k),PowerA(k-1)))+TV_A(k-1);
37 TV_B(k) = abs(diff(PowerB(k),PowerB(k-1)))+TV_B(k-1);
38
39 start(motorA)
```

B.2 P04_F5_CalculateAndSetMotorPower.m

```
40      start (motorB)
41  else
42      %-----
43      % simulerer EV3-Matlab kommunikasjon i online=0
44      %-----
45      pause(0.01)
46 end
```

Vedlegg C

Programlisting prosjekt 05

C.1 P05_F4_MathCalculations.m

Kode C.1: *P05F4Mathcalculations.*

```
1 %-----
2 % Definer parametre som brukes i beregningene.
3 %
4 %Konstanter defineres her
5 Kp=1.7;
6 Ki=1.7;
7 Kd=0.85;
8 alfa=0.5;
9 u_0=10;
10 %
11 Ts(k-1)=Tid(k)-Tid(k-1);
12
13 IAE(k) = IAE(k-1)+abs(e(k-1))*Ts(k-1);
14
15 MAE(k) = sum(abs(e(1:k)))/k;
16
17 middelverdi = sum(Lys(1:k))/k;
18
19 standardavvik = sqrt((sum((Lys(1:k))-middelverdi).^2)/k);
20
21 P(k)=Kp*e(k);
22
23 I(k)= EulerForward(I(k-1),Ki*e(k-1),Ts(k-1));
24
```

C.2 P05_F5_CalculateAndSetMotorPower.m

```
25 e_f(k) = IIR_filter(e_f(k-1), e(k), alfa);
26
27 D(k) = Derivasjon(e_f(k-1:k), Ts(k-1));
28
29 u(k) = (P(k)) + (Ki*I(k)) + (Kd*D(k));
30
31 AvvikGjennomsnitt = sum(Avvik(1:k))/k;
```

C.2 P05_F5_CalculateAndSetMotorPower.m

Kode C.2: P05_F5_CalculateAndSetMotorPower.

```
1 %%%%%%
2 % P05_F5_CalculateAndSetMotorPower.m
3 %
4 % Beregner hvordan motorene skal bevege seg
5 %
6 %%%%%%
7
8
9 %-----
10 % Definer parametre til ? beregne motorp?drag.
11 %-----
12 a = 0.5;
13 b = 2;
14
15 %-----
16 % Beregner motorp?drag og lagrer i datavektor
17 % (slett de motorene du ikke bruker og lag egen kode)
18 %
19 PowerA(k) = u_0 - u(k);
20 PowerB(k) = u_0 + u(k);
21
22
23 if online
24     motorA.Speed = PowerA(k);
25     motorB.Speed = PowerB(k);
26
27     TV_A(k) = abs(diff(PowerA(k-1:k)))+TV_A(k-1);
28     TV_B(k) = abs(diff(PowerB(k-1:k)))+TV_B(k-1);
29     start(motorA)
30     start(motorB)
31 else
32     %
33     % simulerer EV3-Matlab kommunikasjon i online=0
```

C.2 P05_F5_CalculateAndSetMotorPower.m

```
34 %-----  
35 pause(0.01)  
36 end
```

Vedlegg D

Programlisting prosjekt 06

D.1 P06_F4_MathCalculations.m

Kode D.1: *P06F4Mathcalculations.*

```
1 %%%%%%
2 % P0X_F4_MathCalculations.m
3 %
4 % Her programmerer du beregninger som skal gjøres
5 %
6 %%%%%%
7 %
8 %-----%
9 % Definer parametre som brukes i beregningene.
10 %-----%
11 %Konstanter defineres her
12 Kp=10.0;
13 %-----%
14 Ts(k-1)=Tid(k)-Tid(k-1);
15
16 IAE(k)= IAE(k-1)+abs(e(k-1))*Ts(k-1);
17
18 MAE(k) = sum(abs(e(1:k)))/k;
19
20 middelverdi = sum(Avstand(1:k))/k;
21
22 standardavvik = sqrt((sum((Avstand(1:k))-middelverdi).^2)/k);
23
24 P(k)=Kp*e(k);
```

D.2 P06_F5_CalculateAndSetMotorPower.m

```
25  
26 u(k)=P(k);
```

D.2 P06_F5_CalculateAndSetMotorPower.m

Kode D.2: P06_F5_CalculateAndSetMotorPower.

```
1 %%%%%%%  
2 % P0X_F5_CalculateAndSetMotorPower.m  
3 %  
4 % Beregner hvordan motorene skal bevege seg  
5 %  
6 %%%%%%%%%%%%%%  
7  
8  
9 %-----  
10 % Definer parametre til ? beregne motorp?drag.  
11 %-----  
12 a = 0.5;  
13 b = 2;  
14  
15 %-----  
16 % Beregner motorp?drag og lagrer i datavektor  
17 % (slett de motorene du ikke bruker og lag egen kode)  
18 %-----  
19 PowerA(k) = -u(k);  
20 PowerB(k) = -u(k);  
21  
22  
23 if online  
24  
25 %-----  
26 % Setter powerdata mot EV3  
27 % G?r ikke fort?re enn -100 -> +100 selv  
28 % om powerverdi er st?rre  
29 % (slett de motorene du ikke bruker)  
30 %-----  
31  
32  
33 motorA.Speed = PowerA(k);  
34 motorB.Speed = PowerB(k);  
35  
36 TV_A(k) = abs(diff(PowerA(k-1:k)))+TV_A(k-1);  
37 TV_B(k) = abs(diff(PowerB(k-1:k)))+TV_B(k-1);  
38 start(motorA)
```

D.2 P06_F5_CalculateAndSetMotorPower.m

```
39     start (motorB)
40 else
41 %-----
42 % simulerer EV3-Matlab kommunikasjon i online=0
43 %-----
44 pause(0.01)
45 end
```

Vedlegg E

Programlisting prosjekt 07

E.1 P07_F4_MathCalculations.m

Kode E.1: *P07F4Mathcalculations.*

```
1 %%%%%%%%%%%%%%%%
2 % P07_F4_MathCalculations.m
3 %-----
4 % Definer parametre som brukes i beregningene.
5 %-----
6
7 %Tar utgangspunkt i et vanlig sykkelhjul med diameter p? 29 ...
8 % tommer.Vi vil helst
9 % ha en metrisk verdi p? diametern og omgj?r tommer til cm.
10 %tommer omgjort til cm blir 73,66, som vi runder opp til 74 for ...
11 % f? penere
12 %tall.
13 while double(readTouch(myTouchSensor)) == 1
14     if double(readTouch(myTouchSensor)) == 0
15         break;
16     end
17 end
18 Omkrets=3.14*0.740;
19 alfa=0.3;
20 Avstand(k) = sum(Bryter(1:k))*Omkrets;
21 Avstand_IIR(k) = IIR_filter(Avstand_IIR(k-1),Avstand(k),alfa);
22 Momentanfart(k) = Derivasjon(Avstand_IIR(k-1:k),Tid(k)-Tid(k-1));
```

E.2 P07_F5_CalculateAndSetMotorPower.m

```
23 Gjennomsnittsfart(k) = Avstand(k)/Tid(k);  
24 Momentanakselerasjon(k) = ...  
    Derivasjon(Momentanfart(k-1:k), Tid(k)-Tid(k-1));  
25 Gjennomsnittsakselerasjon(k) = Momentanfart(k)/Tid(k);  
26 Momentanfart_u(k) = Derivasjon(Avstand(k-1:k), Tid(k)-Tid(k-1));
```

E.2 P07_F5_CalculateAndSetMotorPower.m

Kode E.2: *P07_F5_CalculateAndSetMotorPower.*

```
1 %%%%%%%%%%%%%%%  
2 % P0X_F5_CalculateAndSetMotorPower.m  
3 %  
4 % Beregner hvordan motorene skal bevege seg  
5 %  
6 %%%%%%%%%%%%%%%  
7  
8  
9 %-----  
10 % Definer parametre til ? beregne motorp?drag.  
11 %-----  
12 %-----  
13 % Beregner motorp?drag og lagrer i datavektor  
14 % (slett de motorene du ikke bruker og lag egen kode)  
15 %-----  
16  
17 if online  
18  
19 %-----  
20 % Setter powerdata mot EV3  
21 % G?r ikke fortore enn -100 -> +100 selv  
22 % om powerverdi er st?rre  
23 % (slett de motorene du ikke bruker)  
24 else  
25 %-----  
26 % simulerer EV3-Matlab kommunikasjon i online=0  
27 %-----  
28 pause(0.01)  
29 end
```

Vedlegg F

Programlisting prosjekt 08

F.1 P08_F4_MathCalculations.m

Kode F.1: *P08F4Mathcalculations.*

```
1 %%%%%%%%%%%%%%%%
2 % P0X_F4_MathCalculations.m
3 %
4 % Her programmerer du beregninger som skal gj?res
5 %
6 %%%%%%%%%%%%%%%%
7 %
8 %-----%
9 % Definer parametre som brukes i beregningene.
10 %-----
11 alfa = 0.3;
12
13 %Lys(k) = Lys(k) - nullflow;
14
15 Avstand_filter(k) = ...
    IIR_filter(Avstand_filter(k-1),Avstand(k),alfa);
16
17 V_Sonisk_f(k) = ...
    abs((Derivasjon(Avstand_filter(k-1:k),Ts(k-1)))*100);
18
19 V_Sonisk(k) = abs((Derivasjon(Avstand(k-1:k),Ts(k-1)))*100);
20
21 Avstand_motor(k) = VinkelPosMotorA(k)*((2*pi*2.8)/360); %17.6 ...
    er omkretsen, 360 er gradene. Det vil si at n?r hjulene har ...
```

F.2 P08_F5_CalculateAndSetMotorPower.m

```
    beveget seg en runde rundt vil det være ekvivalent 17.6 cm ...
    som er omkretsen av hjulene
22
23 V_motor(k) = Derivasjon(Avstand_motor(k-1:k), Ts(k-1));
24
25 Avstand_lys(1) = 0;
26
27 % if lys(k) > 50
28 %     Avstand_lys(k) = Avstand_lys(k) +2.7
29 % end
30 %
31 % if lys(k) < 50
32 %     Avstand_lys(k) = Avstand_lys(k) +2.7
33 % end
34
35 % a(k)=0;
36 % while Lys(k)<50;
37 %     if Lys(k)>50;
38 %         a(k)= 1;
39 %         break
40 %     end
41 % end           %Planen her er ? ta summen av a og deretter ...
        multiplisere dette med 5.4
42 %
43 %
44 % Avstand_lys(k)=sum(a(1:k))*5.4;
```

F.2 P08_F5_CalculateAndSetMotorPower.m

Kode F.2: P08F5_CalculateAndSetMotorPower.

```
1 %%%%%%
2 % P0X_F5_CalculateAndSetMotorPower.m
3 %
4 % Beregner hvordan motorene skal bevege seg
5 %
6 %%%%%%
7 %
8 %
9 %-----
10 % Definer parametre til ? beregne motorp?drag.
11 %-----
12 a = 0.5;
13 b = 0.3;
14
```

F.2 P08_F5_CalculateAndSetMotorPower.m

```
15 %-----
16 % Beregner motorp?drag og lagrer i datavektor
17 % (slett de motorene du ikke bruker og lag egen kode)
18 %-----
19 PowerA(k) = a*JoyForover(k) + b*JoyLR(k);
20 PowerB(k) = a*JoyForover(k) - b*JoyLR(k);
21
22 if online
23
24 %-----
25 % Setter powerdata mot EV3
26 % G?r ikke fortore enn -100 -> +100 selv
27 % om powerverdi er st?rre
28 % (slett de motorene du ikke bruker)
29 %-----
30 motorA.Speed = PowerA(k);
31 motorB.Speed = PowerB(k);
32 start(motorA)
33 start(motorB)
34 else
35 %-----
36 % simulerer EV3-Matlab kommunikasjon i online=0
37 %-----
38 pause(0.01)
39 end
```

Vedlegg G

Programlisting prosjekt 09

G.1 P09_F4_MathCalculations.m

Kode G.1: *P09F4Mathcalculations.*

```
1
2 %flow(k) = Lys(k) - nullflow;
3 flow(k) = 1*sin((Lys(k) - nullflow)*((2*pi)/100))
4 % ny "m?ling" av flow, f(k)Ts(k-1) = ....
5 % beregn tidsskritt i sekundvolum(k) = ....
6 % estimert volum i glasset basert p?% numerisk integrasjon av ...
    flowm?ling
7 Ts(k-1)=Tid(k)-Tid(k-1)
8 volum(k)= abs(volum(k-1)+flow(k-1)*Ts(k-1))
9 volum_trap(k) = abs((volum_trap(k-1) + ...
    ((sum(flow(k-1:k))./2))*Ts(k-1)))
```

G.2 P09_F5_CalculateAndSetMotorPower.m

Kode G.2: *P09F5CalculateAndSetMotorPower.*

```
1 %%%%%%
2 % P0X_F5_CalculateAndSetMotorPower.m
3 %
```

G.2 P09_F5_CalculateAndSetMotorPower.m

```
4 % Beregner hvordan motorene skal bevege seg
5 %
6 %%%%%%
7 %
8 %
9 %-----
10 % Definer parametre til ? beregne motorp?drag.
11 %
12 a = 1;
13 b = 1;
14 %
15 %
16 % Beregner motorp?drag og lagrer i datavektor
17 % (slett de motorene du ikke bruker og lag egen kode)
18 %
19 % PowerA(k) = a*JoyForover(k);
20 % PowerB(k) = a*b*Lys(k);
21 % PowerC(k) = b*Bryter(k);
22
23 if online
24
25 %
26 % Setter powerdata mot EV3
27 % G?r ikke fortore enn -100 -> +100 selv
28 % om powerverdi er st?rre
29 % (slett de motorene du ikke bruker)
30 %
31 % motorA.Speed = PowerA(k);
32 % motorB.Speed = PowerB(k);
33 % motorC.Speed = PowerC(k);
34 %
35 % start(motorA)
36 % start(motorB)
37 % start(motorC)
38
39 else
40 %
41 % simulerer EV3-Matlab kommunikasjon i online=0
42 %
43 pause(0.01)
44 end
```