

ELE130 Anvendt matematikk og fysikk i robotprogrammering

PROSJEKT - VÅREN 2022

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

Gruppenummer	2205		
Gruppens medlemmer	Navn	Studentnummer	Bilde
	Truls Gjerstad	265247	
	Aleksa Repanovic	267250	
	Tormod Klungland	267258	
	Sondre Lyngstad	256437	

Innhold

Innhold	i
Sammendrag	viii
0 Lego Eduaction	1
0.1 Deler i bruk	1
0.1.1 EV3 Kloss	1
0.1.2 Lyssensor	2
0.1.3 Trykksensor	3
0.1.4 Ultrasonisk sensor	3
0.1.5 Gyrosensor	4
0.1.6 Stor servomotor	4
0.1.7 Servomotor	5
1 Numerisk integrasjon 1 (utført av hele gruppen)	6
1.1 Numerisk Integrasjon	6

INNHOLD

1.2	Verifikasiing av kode - del 1	8
1.3	Verifikasiing av kode - del 2	10
1.4	Resultat	11
1.5	Sammendrag	12
2	Filtrering (utført av hele gruppen)	13
2.1	Problemstilling	13
2.2	FIR-filter	13
2.2.1	Kode	14
2.2.2	Resultat	15
2.3	IIR-filter	15
2.3.1	Kode	16
2.3.2	Resultat	17
2.4	FIR og IIR filtrering med målestøy	18
2.4.1	Kode	18
2.4.2	Resultat	19
2.5	FIR og IIR filtrering som funksjoner	20
2.6	Sammenligning	21
2.7	Sammendrag	22
3	Numerisk derivasjon (utført av hele gruppen)	23

INNHOLD

3.1	Problemstilling	23
3.2	Teori	24
3.3	Forslag til løsning	25
3.4	Resultat	26
3.4.1	Derivasjon av et lineært signal	26
3.4.2	Derivasjon av et sinussignal	28
3.4.3	Derivasjon av ulike typer signaler	31
3.4.4	G-krefter i Stupet".	33
3.5	Sammendrag	34
4	Manuell kjøring av Lego-robot (utført av hele gruppen)	35
4.1	Forslag til løsning	37
4.2	Resultat	41
4.3	Histogram og sammenligning	43
4.4	Sammendrag	44
5	Automatisk kjøring av Lego-robot (Utført av bare roboten)	46
5.1	Problemstilling	48
5.2	Forslag til løsning	48
5.3	Resultat	51
5.4	Sammenligning	54

INNHOLD

6 Cruise-control (utført av hele gruppen)	55
6.1 Introduksjon	57
6.2 Forslag til løsning	57
6.3 Resultat	59
6.4 Sammendrag	60
7 Parkeringsassistent	61
7.1 Problemstilling	63
7.2 Forslag til løsning	64
7.3 Resultat	66
7.4 Sammendrag	68
8 Samlebånd	69
8.1 Problemstilling	70
8.2 Forslag til løsning	70
8.3 Resultat	73
8.4 Sammendrag	74
9 Konklusjon	75
Bibliografi	77
Vedlegg	77

INNHOLD

A Timelister	78
B Programlisting prosjekt 01	79
B.1 Projekt01_Filtrering.m	79
C Programlisting prosjekt 02	80
C.1 Projekt02_filtrering.m	80
D Programlisting prosjekt 03	82
D.1 Projekt03_Numerisk_Derivasjon.m	82
E Programlisting prosjekt 04	84
E.1 Projekt04_Manuell_Kjoring.m	84
F Programlisting prosjekt 05	86
F.1 Projekt05_selvKjoring.m	86
G Programlisting prosjekt 06	89
G.1 Projekt06_CruiseControl.m	89
H Programlisting prosjekt 07	91
H.1 Projekt07_Parkeringsassistanse.m	91
I Programlisting prosjekt 08	95
I.1 Projekt08_Samlebaand.m	95

Sammendrag

Denne rapporten går gjennom diverse oppgaver angående programmering av en lego robot / EV3. Hver oppgave er et kapittel i rapporten. Det er en obligatorisk del (de 4 første oppgavene) og en kreativ del (de 4 siste oppgavene). Hva de 8 ulike oppgavene går ut på kan man lese i oversikten nedenfor.

Prosjekt 1 Numerisk Integrasjon

I delkapittelet numerisk integrasjon skal vi ta opp et signal ved hjelp av en lys-sensor. Dette signalet skal fremstille fylling/tapping av veske i en kopp. Vi skal lage en kode som beregner arealet under signalet, samt verifiserer at koden vår for integrasjonsregning stemmer. For å få til dette bruker vi både Eulerforover og Trapesmetoden.

Prosjekt 2 FIR og IIR filtrering

Vi har i dette delkapittelet dukket dypere i tematikken rundt filtrering. Vi har sett på hvordan man kan redusere målestøy ved bruk av et FIR eller et IIR filter og hvordan man implementerer dette i Matlab. Vi har sett på egenskapene til de to ulike filterene og hvordan de oppfører seg med ulike m og a verdier.

Prosjekt 3 Numerisk Derivasjon

I kapittelet om numerisk derivasjon har vi gjennomført to eksperimenter. I det første eksperimentet så vi på hvordan politiets fartsmålerer fungerte, nemlig ved å numerisk derivere avstand. I det andre eksperimentet så vi på avstander, fart og akselerasjon i en karusell kalt "Stupet". Avstandssignalet vi får ved å simulere

INNHOLD

bevegelsesmønsteret til Stupeter på sinus form.

Prosjekt 4 Manuell Kjøring

Vi har i dette delkapittelet for manuell kjøring, kjørt lego roboten gjennom en svart hvit skalert bane. Lego roboten har gjennom løypa tatt opp lysverdier, som vi har brukt videre i beregninger av kvalitetsmålinger som f.eks IAE (integral of absolute error) og MAE (mean of absolute error). Vi har utifra kvalitetsmålene drøftet rundt hvilken av gruppemedlemmene som har best kjøring.

Prosjekt 5 Automatisk Kjøring

I delkapittelet for automatisk kjøring har vi drøftet rundt robotens kjøreevne med *PID* regulering. Vi har sammenlignet ulike kvalitetsmål med manuell kjøring, samt sett på kvalitetsmål med enkelte deler av *PID* reguleringen.

Prosjekt 6 Cruise Controll

Ved hjelp av den adaptive cruise controll kan vi lene oss trygt tilbake i førerstolen ved bilkjøring. I delkapittel 6, cruise controll, er det akkurat dette vi ser på. Når man begynner å nærme seg en bil "i framkant, vil cruise kontrollen automatisk regulere farten, slik at den holder en fast avstand til bilenforan. Hvis det ikke er bil foran så vil roboten kjøre etter den satte cruise control farten. Vi har videre sammenliknet grafene med det som skjer i praksis og deretter verifisert koden.

Prosjekt 7 Parkerings Assistanse

I dette delkapittelet har vi sett på et konsept innført med de nyeste bilene, parkeringsassistanse. Vi har laget en kode som hjelper bilføreren å automatisk rygge ut av en vanskelig parkeringsplass. Dette gjorde vi ved å snu pådraget og dreie retning på motorene. Ettersom motorene ikke er ideelle får vi litt avvik og ved lengre simuleringer vil den slite mer med å returnere til startpunktet, som man kan lese mer om i dette delkapittelet.

Prosjekt 8 Samlebånd filtrering

INNHOLD

Vi har i dette delkapittelet gått nærmere inn på sorteringsautomatikk. Vi har ved hjelp av egenskrevet kode i Matlab og et egetlagd samlebånd, sortert to hvite og en svart boks. Vi har gått igjennom hvordan koden fungerer og hvordan grafene som viser motorpådrag og lysverdi er koblet til det som skjer i praksis.

Kapittel 0

Lego Eduaction

0.1 Deler i bruk

Dette delkapitlet er en kort beskrivelse av delene som blir tatt i bruk under dette prosjektet.

0.1.1 EV3 Kloss

EV3 klossen fungerer som en hjerne/cpu til delene og roboten som bygges. Den gjør det mulig å koble sammen opptil fire motorer og fire sensorer. For å bruke EV3 klossen sendes kode ved hjelp av enten en USB kabel mellom pc og EV3 kloss eller et SD-kort direkte i EV3 klossen, om man har tilgang til en WiFi eller bluetooth dongle kan det også benyttes. I dette prosjektet blir USB kobling tatt i bruk.

0.1 Deler i bruk



Figur 1: EV3 klossen. Figuren er hentet fra [1]

0.1.2 Lyssensor

Lyssensoren måler reflektert rødt lys, omgivelseslys, fra mørke og lyse områder. Den kan også skille mellom åtte farger. Sample rate skjer på 1kHz.



Figur 2: Lyssensor. Figuren er hentet fra [1]

0.1 Deler i bruk

0.1.3 Trykksensor

Trykksensoren er en presis sensor som registrerer hvis knappen blir trykket inn eller sluppet. Den kan telle en eller flere trykk.



Figur 3: Trykksensor. Figuren er hentet fra [1]

0.1.4 Ultrasonisk sensor

Ultralydsensoren genererer lydbølger og leser ekkoene for å registrere og måle avstanden til ett eller fler objekt. Den kan også fungere som enn sonar, den sender da enkle lydbølger. kan også lytte etter lydbølger som starter ett program. Den kan måle avstand mellom 1-250cm med enn feilmargin på +/-1cm.



Figur 4: Ultrasonisk sensor. Figuren er hentet fra [1]

0.1 Deler i bruk

0.1.5 Gyrosensor

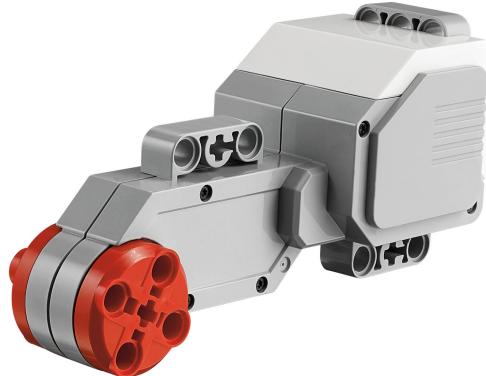
Gyrosensoren mäter rotasjonsbevegelse, den kan endre retning, med klokken er positiv og mot klokken er negativ. Gyrosensoren har en nøyaktighet på +/-3 grader, rotasjonshastighet maks ytelse på 440 grader/sekund. sample rate skjer på 1kHz.



Figur 5: Gyrosensor. Figuren er hentet fra [1]

0.1.6 Stor servomotor

Servomotoren brukes til å flytte konstruksjonen din, kan også brukes til å rotere sensorer.



Figur 6: Stor servomotor. Figuren er hentet fra [1]

0.1 Deler i bruk

0.1.7 Servomotor

Servomotoren brukes til å flytte konstruksjonen din, kan også brukes til å rotere sensorer.



Figur 7: Servomotor. Figuren er hentet fra [1]

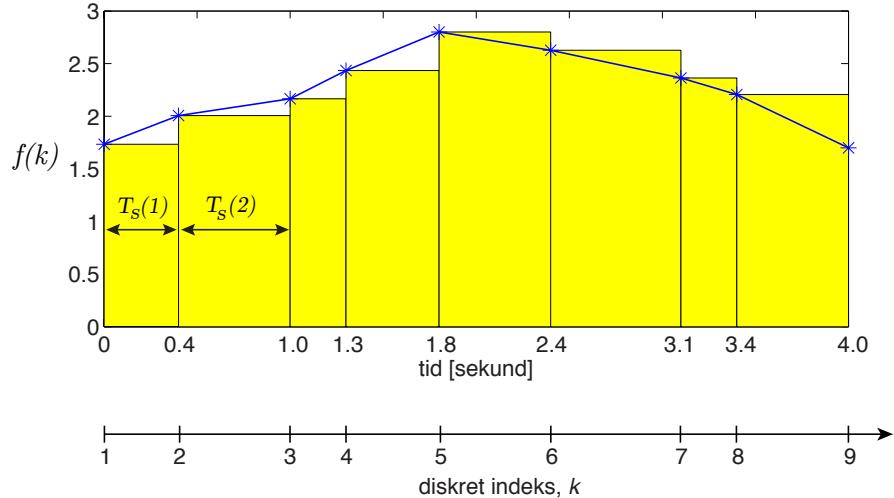
Kapittel 1

Numerisk integrasjon 1 (utført av hele gruppen)

1.1 Numerisk Integrasjon

I dette kapitlet skal vi gå nærmere innpå numerisk integrasjon. Vi skal vise hvordan vi kan få opp et signal og numerisk integrere signalet. Signalet vi skal jobbe rundt får vi ved bruk av en lyssensor. Vi skal numerisk integrere signalet, validere koden som utfører integrasjon og vise at det numerisk integrerte signalet viser volumet i glasset. For å kunne starte å vise dette trenger man å forstå hva numerisk integrasjon går ut på.

1.1 Numerisk Integrasjon



Figur 1.1: 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 [2].

Som forklart i figur 1.1, så vil summen av arealet til de gule rektanglene tilsvare integralverdien til enhver tid. Den generelle formelen for dette er:

$$volum(k) = volum(k - 1) + (Tid Skritt(k - 1) * Flow(k - 1)) \quad (1.1)$$

Her er $volum(k)$ det totale volumet, $volum(k-1)$ være det totale areal en indeks tidligere. Ettersom vi ikke har en måling for volumet når $k = 1$, så setter vi $Volum(1) = 0$. Vi begynner derfor ikke å måle volum før etter to tidsskritt. $Tid Skritt(k - 1)$ være tidskrittet en indeks tidligere og $Flow(k - 1)$ være hvor mye vann som blir sugd opp eller tilført glasset i en indeks tidligere. Med andre ord vil det totale volumet vann sugd opp/tlført tilført glasset være det totale volumet indeksen før addert med tidskrittet i forrige måling multiplisert med hvor mye vann som blir sugd opp eller tilført glasset i en indeks tidligere.

I formelen kommer bokstaven k opp. Denne bokstaven tilsvarer indeksen man er i til enhver tid. Eksempelvis vil $volum(k)$, hvor k er lik 7, være det totale volumet etter den 7 målingen er gjort.

Et av hovedproblemene med numerisk integrasjon er at volumet ikke vil bli 100 prosent nøyaktig. Som man kan se fra figur 1.1 vil det oppstå små hvite trekanner,

1.2 Verifisering av kode - del 1

ved tilførsel av vann (positiv flow), som ikke vil bli med i beregningene. Dette er fordi man ikke vet hvordan neste måling, $(k + 1)$, vil være. På den andre siden vil vi få et for stort areal når vann blir sugd ut av koppen (negativ flow). Siden vi får for lite når mengden øker, og for mye når mengden synker vil dette avviket bli tilnærmet lik 0. For å løse dette problemet ønsker man å ha minst mulig tidsskritt, slik at summen av unøyaktighetene vil bli mindre.

Nå som vi vet mer om numerisk integrasjon skal vi vise det i praksis. Det første vi gjør er å lage koden for numerisk integrasjon i Matlab som vist i kode 1.1 under.

Kode 1.1: Utdrag av funksjonen `NummeriskIntegrasjon.m`. Bilde av kode viser numerisk utregning. Bruker en if-setning når $k=1$ for å unngå feil. Definerer tidskrittet som T_s endring i signalet som flow. Bruker Eulers-forover metode for å numerisk integrere.

```
157     nullflow = Lys(1);  
158  
159     if k==1  
160         Flow(1) = 0;  
161         volum(1) = 20;  
162     else  
163         Ts(k-1) = Tid(k) - Tid(k-1) ;  
164         Flow(k) = Lys(k) - nullflow(1);  
165         %volum(k) = volum(k-1) + (Ts(k-1) * Flow(k-1));  
166         volum(k) = EulerForward(volum(k-1), Flow(k-1), Ts(k-1))  
167     end
```

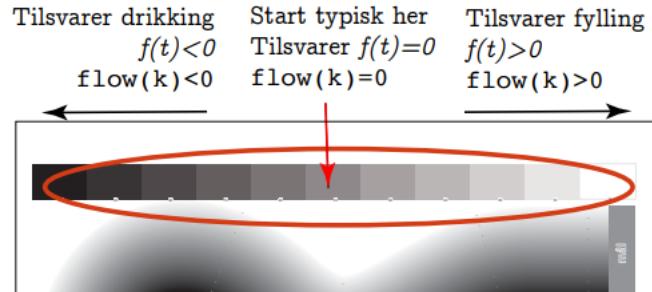
Som vist i Kode 1.1 definerer vi tidskrittet T_s som tiden målingen ble tatt minus tiden da forrige målingen ble tatt. Flowen definerer vi som lyset målt av lyssensoren minus nullflow. Nullflow setter vi som den første målte verdien av lys. Nå er vi klare for å ta opp signaler fra lysmåleren.

1.2 Verifisering av kode - del 1

For å verifisere at koden er riktig, må vi kontrollere et par ting. Vi må verifisere at differansen i funksjonsverdien til volums-signalet når tiden er t_1 og t_2 er lik arealet under flow-signalet når tiden er t_1 og t_2 .

For å verifisere er det nødvendig å ha et signal som skal verifiseres. Vi lager dette signalet ved å bevege lyssensoren i en rett linje på det fargeskalerte arket, som viste på figur 1.2.

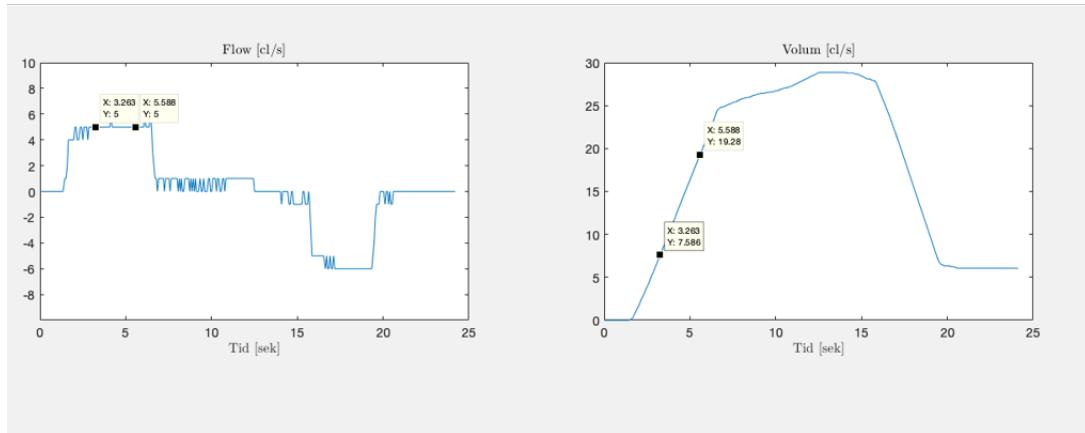
1.2 Verifisering av kode - del 1



Figur 1.2: Beveget lyssensoren langs rød linje vist på bildet

Dette signalet fremstiller fylling/tømming av vann i et glass. Ved å bevege lyssensoren mot den mørke delen av arket simulerte vi "tømming" av glasset. På samme måte simulerete vi "fylling" av vannglasset når vi beveget lyssensoren mot den lyse delen av arket.

Signalet som fremkommer, samt signalet numerisk integrert er vist i figur 1.3.



Figur 1.3: Flow - signal tatt opp av lyssensoren. Volum - signalet numerisk integrert

Fra flow signalet fra figur 1.3 kan vi se at arealet under grafen fra t_1 til t_2 er:
 $5 * (5.588 - 3.263) = 11.625$

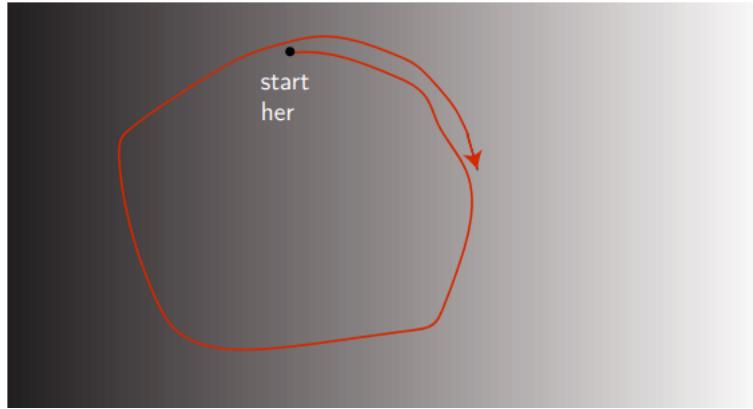
Denne verdien skal tilsvare differansen til funksjonsverdien i det nummerisk integrerte signalet i t_1 til t_2 . Dette er $19.28 - 7.586 = 11.694$.

1.3 Verifisering av kode - del 2

Vi ser at det stemmer og kan med det verifisere koden.

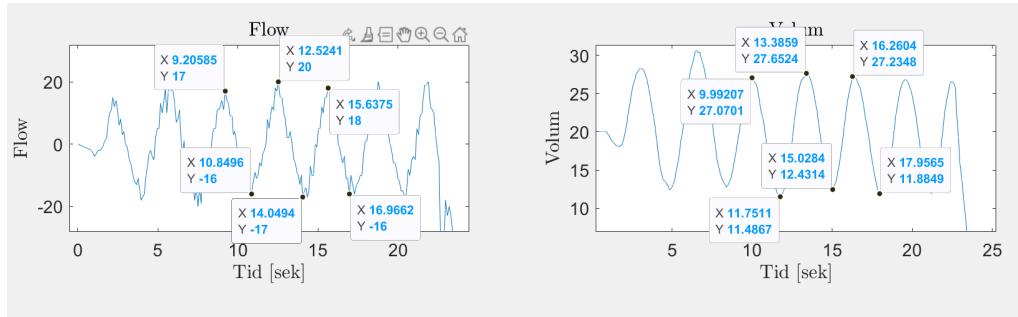
1.3 Verifisering av kode - del 2

I dette delkapittelet skal vi verifisere koden ved å se på et sinus formet signal. Dette gjør vi ved å rotere lyssensoren i en sirkelbevegelse på det svart-hvit skalerte arket vist i figuren 1.4.



Figur 1.4: Beveget lyssensoren langs rød sirkel vist på bildet

Signalet tatt opp ved å bevege lyssensoren som vist i figur 1.5 og dette signalet numerisk integrert, kan man se i figur 1.5.



Figur 1.5: Flow funksjon. Dette er lysmåling - nullflow til enhver tid.

1.4 Resultat

Når vi igjen skal verifisere koden for numerisk integrasjon for det sinusformede flow signalet vårt, må vi først finne et uttrykk for signalet. Den generelle formelen for en sinusfunksjon er:

$$a \sin(wt) \quad (1.2)$$

Ser at vi trenger amplitude og vinkelfrekvensen til flow signalet vårt. Leser av at amplituden ligger på 17.25. Bruker formelen $w = \frac{2\pi}{T}$ for å finne at vinkelfrekvensen er

$$w = \frac{2\pi}{T} w = \frac{2\pi}{12.52 - 9.21} = 1.90 \quad (1.3)$$

Uttrykket for det sinusformede flow signalet blir:

$$17.25 \sin(1.9t) \quad (1.4)$$

Ved å integrere dette uttrykket vil vi få et uttrykk for volumet. Utrykket integrert er vist i equation 1.5 under.

$$\int 17.25(1.9t)dt = -9.07895 \cos(1.9t) \quad (1.5)$$

Indefinite integral. funksjon for flow signalet integrert

Fra utrykket i equation 1.5 ser vi at amplituden er 9.08 og vinkelfrekvensen er 1.9. Ved å se på volum signalet vårt fra figur 1.5 kan vi lese av at den reelle amplituden er 8.1 og vinkelfrekvensen er 1.85. Fra dette er koden vår for numerisk integrasjon for et sinusformet signal verifisert.

1.4 Resultat

Nå som vi vet at koden vår fungerer som den skal, kan vi lage en generell funksjon for numerisk integrasjon i matlab. Da kan man kalle på denne funksjonen og den vil gjennomføre numerisk integrasjon for vilkårlige tall. Dette er vist i Kode 1.2.

1.5 Sammendrag

Kode 1.2: Utdrag av funksjonen `EulerForward.m`. Generell funksjon for numerisk integrasjon.

```
1 function [IntValueNew] = EulerForward(IntValueOld, ...
2                                     FunctionValue, TimeStep)
3 IntValueNew = IntValueOld + FunctionValue.*TimeStep;
4
5 end
```

I kode 1.3 er det fremvist hvordan man kaller på funksjonen og sender inn ønskelig tall.

Kode 1.3: Utdrag av funksjonen `NumeriskIntegrasjon.m`. Fremvisning av hvordan en definert funksjon kalles på tall.

```
159 if k==1
160     Flow(1) = 0;
161     volum(1) = 20;
162 else
163     Ts(k-1) = Tid(k) - Tid(k-1) ;
164     Flow(k) = Lys(k) - nullflow(1);
165     %volum(k) = volum(k-1) + (Ts(k-1) * Flow(k-1));
166     volum(k) = EulerForward(volum(k-1), Flow(k-1), Ts(k-1))
167 end
```

1.5 Sammendrag

Kapittel 2

Filtrering (utført av hele gruppen)

2.1 Problemstilling

I denne øvingen skal vi ved hjelp av FIR og IIR filtrering fjerne målestøy og glatte ut kurver. Dette skal vises med et kaffekopp og et termometer eksempel, som simuleres med en lyssensor. Vi skal se på de ulike egenskapene til FIR og IIR filtrering.

2.2 FIR-filter

Et FIR filter bruker de m antall siste målingene til å beregne en filtrert verdi i nå tidspunktet. Hvor mange m målinger man vil bruke, bestemmes av brukeren. Ønsker man å glatte ut kurven må man ha en høy m verdi. Ulempen med å ha en høy m verdi er at vi kan gå glipp av verdier og kodden kan bli treg/forsinket, fordi man er nødt til å lagre mye data. Derfor er det smart å velge en passende m , slik at kodden ikke blir treg eller forsinket, men har nok verdier til å få en glatt kurve. FIR-verdier regnes ut ved formelen:

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

2.2 FIR-filter

2.2.1 Kode

Her legger vi til en *randn* funksjon som skal simulerer målestøy. Når m er mindre enn k bruker vi antallet målinger vi har i tidspunktet k , som deles på måleverdiene. Hvis vi ikke hadde hatt med denne if-setningen, så hadde vi fått feil måling fram til m overstiger k .

Kode 2.1: Koden for FIR filtrering

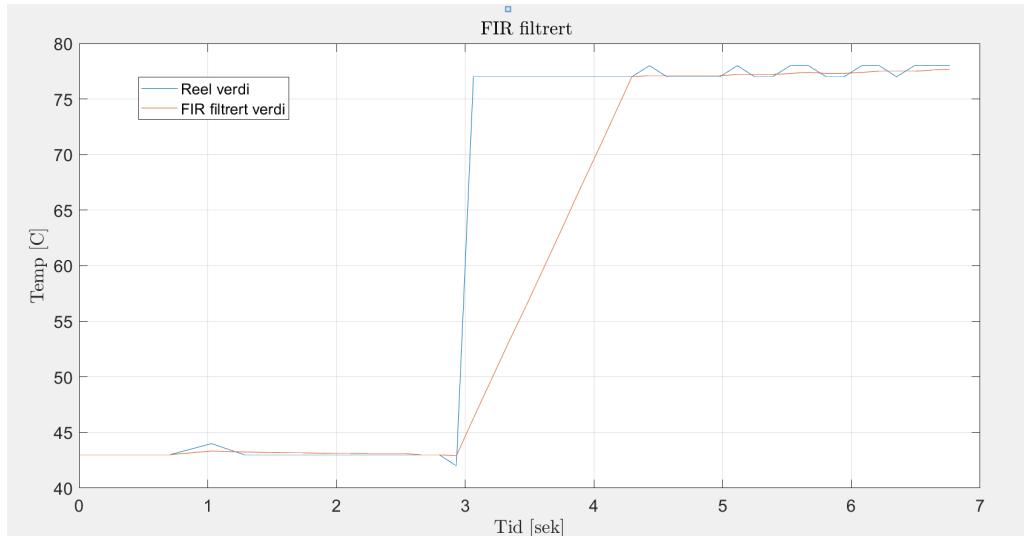
```
148      %FIR Filtrering%
149      %-----
150      Temp(k) = Lys(k);
151      %Temp_stoy(k) = Lys(k) + rand;
152      m1 = 3;
153      m2 = 10;
154      m3 = 50;
155      m4 = 100;
156      m5 = 200;
157
158      % FIR med m = 3
159      if k<m1
160          m1=k;
161      end
162
163      Temp_FIR1(k) = (1/m1)*sum(Temp(k-m1+1:k));
```

I kode 2.1, linje 150, så setter vi temp lik den målte lys verdien, denne skal simulere temperaturen. Videre i koden så setter vi de forskjellige m verdiene for *FIR* filtreringen.

Hvis vi setter m til å være 100 så vil summen av de 10 første målingene deles på 100. Da vil vi få en tiendel av den reelle verdien og et helt feil måleresultat. Derfor er vi nødt til å se på hvor mange målte verdier vi har til enhver tid. Dette gjør vi i kode 2.1,linje 159. Her sjekker vi om antall målinger er mindre enn den faste m verdien. Hvis vi har mindre målinger enn m , så vil programmet gjøre m til antall målinger. Da blir de filtrerte verdiene riktig, før vi har oppnådd m antall målinger.

2.3 IIR-filter

2.2.2 Resultat



Figur 2.1: FIR filtrert verdi (Rød) og Reel verdi (Blå)

Grafen viser reel verdi og filtrert verdi. Ser hvordan filtrert verdi stiger lineært. Ved å justere antallet målinger m som deles på måleverdiene vil en få en raskere eller tregere stigning av filtrert verdi.

2.3 IIR-filter

IIR filter er en alternativ måte å filtrere et signal på. Her brukes den gamle filtrerte verdien og den nye ufiltrerte verdien. Hvor mye av den gamle og nye verdien man vil bruke bestemmes med en variabel α . Dette kan vi se i bruk i ligning 2.3. α skal alltid ha totalverdi 1. For å få en fin og glatt kurve så bør man ha større del av den gamle verdi. Skrives på formel:

$$Temp_IIR(k) = \alpha * Temp(k) + (1 - \alpha) * Temp_IIR(k - 1) \quad (2.3)$$

2.3 IIR-filter

2.3.1 Kode

Koden som brukes for IIR filtrering er vist i figur 2.3. I starten av kodeutdrag 2.2 settes verdier for a , som sier hvor mye av den gamle og nye verdien som skal brukes.

Kode 2.2: Kode for IIR filtrering

```
194 % IIR-Filtrering
195 %-----
196 a1= 0.01;
197 a2= 0.1;
198 a3= 0.4;
199 a4= 0.7;
200 a5= 0.9;
201
202 if k==1
203     Temp_IIR1(k) = Lys(k);
204     Temp_IIR2(k) = Lys(k);
205     Temp_IIR3(k) = Lys(k);
206     Temp_IIR4(k) = Lys(k);
207     Temp_IIR5(k) = Lys(k);
208 else
209     %IIR filtrering med alfa = 0.01
210     Temp_IIR1(k)= a1*Temp(k)+(1-a1)*Temp_IIR1(k-1);
```

For utregning av filtrert temperatur så ser vi i kode 2.2, linje 210. Her multipliseres temperaturen med konstanten a , og adderes med den forrige filtrerte verdien som multipliseres med $(1-a)$. Ligning 2.4 er et eksempel hvor koden 2.3 blir brukt:

$$Temp_{IIR}(k) = 0.1 * 50 + (1 - 0.1) * 45 = 45.5 \quad (2.4)$$

Den målte temperatur verdien her er **50** som multipliseres med **a = 0.1**. Dette skal adderes med den forrige filtrerte verdien som blir multiplisert med **(1-a)**. Summen av den filtrerte verdien blir **45.5**, selv om den reelle verdien er **50**.

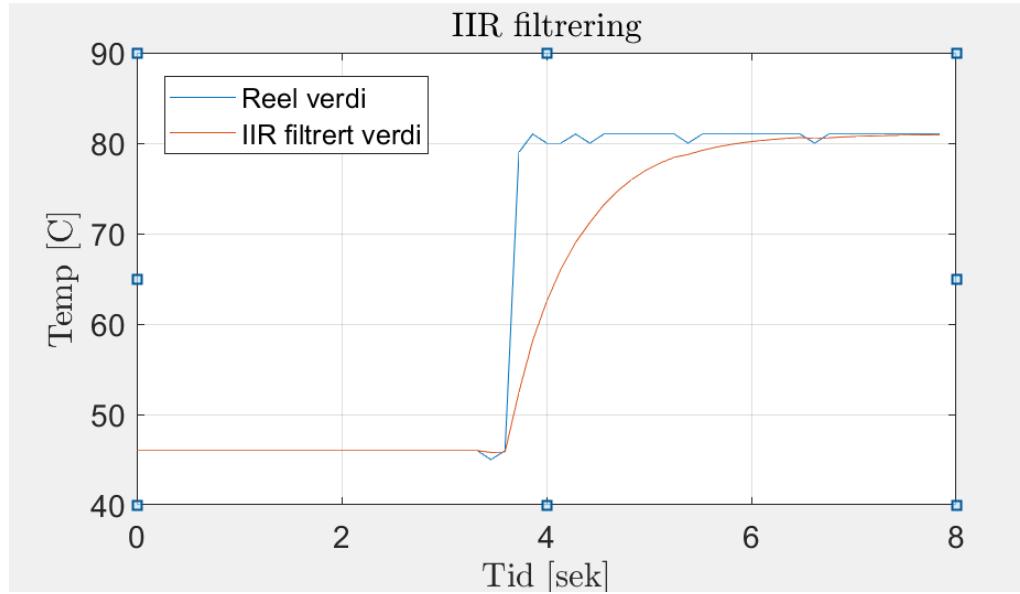
Hvis en bruker høyere a verdi så vil en få et annet resultat. Dette er vist i ligning 2.5.

$$Temp_{IIR}(k) = 0.5 * 50 + (1 - 0.5) * 45 = 47.5 \quad (2.5)$$

2.3 IIR-filter

2.3.2 Resultat

Fra figur 2.2 ser man at ved IIR filtrering vil man til kontrast fra FIR filtrering, få en ikke lineær stigning. Ved høy a verdi, så vil man få mer av den nye målingen, dette resulterer i at en fanger bedre opp de raske endringene.



Figur 2.2: IIR filtrering

2.4 FIR og IIR filtrering med målestøy

2.4 FIR og IIR filtrering med målestøy

Mange måleapparater har integrert filtrering for å kompansere for målestøy. Selv om vi har en del målestøy med bare lys sensoren så legger vi til ekstra for å fremheve hvordan filtreringen hjelper mot målestøy.

2.4.1 Kode

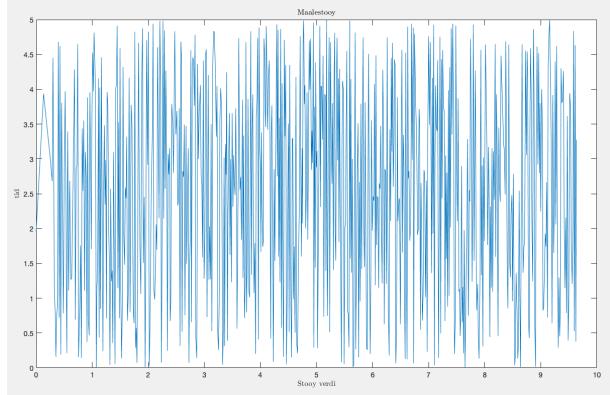
Kode 2.3: Kode for IIR og FIR filtrering med målestøy

```
150      Temp_stoy(k) = Lys(k) + 5*rand;
151      Temp(k) = Lys(k);
152      m = 10;
153      stoy(k) = Temp_stoy(k) - Temp(k);
154
155      % FIR med m = 10
156      if k<m
157          m=k;
158      end
159
160      Temp_FIR(k) = (1/m)*sum(Temp_stoy(k-m+1:k));
161
162      %-----%
163      % IIR-Filtrering
164      %
165      a= 0.2;
166
167      if k==1
168          Temp_IIR(k) = Temp_stoy(k);
169      else
170          %IIR filtrering med alfa = 0.2
171          Temp_IIR(k)= a*Temp_stoy(k)+(1-a)*Temp_IIR(k-1);
172
173      end
```

I kodeutdrag 2.3, linje 150 så måler vi Temp med lyssensoren. For å legge til målestøy så plussar vi på en innebygd funksjon, *rand*, multiplisert med 5. Ved å ikke multiplisere med 5 så vil vi få målestøy mellom 0-1, mens når vi multipliserer *rand* med 5 så vil få målestøy mellom 0-5. Støyet som blir lagt til ser vi i figur 2.3

FIR og IIR filtrerte verdier regnes ut som vist tidligere i delkapitelet.

2.4 FIR og IIR filtrering med målestøy



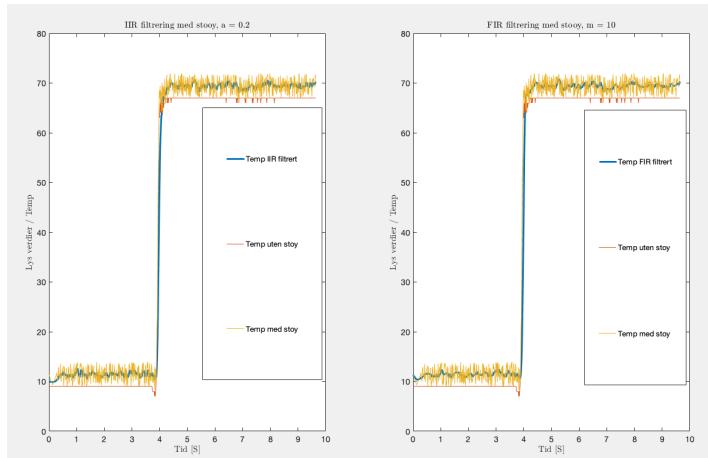
Figur 2.3: Støy lagt til måleverdi

2.4.2 Resultat

Når vi mäter lyssignal fra lyssensoren og legger til målestøy vil grafen se ut som i figur 2.4. Ettersom *rand* funksjonen ikke legger til negative tall vil avviket bare være positivt og filtrert verdi vil ikke nærme seg målt verdi.

Hvis *rand* funksjonen hadde lagd til både negative og positive verdier ville filtrert verdi vært tilnærmet lik målt verdi.

2.4.



Figur 2.4: FIR og IIR filtrering med målestøy

2.5 FIR og IIR filtrering som funksjoner

2.5 FIR og IIR filtrering som funksjoner

For videre bruk av filtrering til andre oppgaver, er det lettere å bruke en funksjon. Definerer derfor en funksjon for IIR filtrering og en funksjon for FIR filtrering, vist i 2.4 og 2.5.

Kode 2.4: FIR funksjon

```
1 function [FilteredValue] = FIR_filter(Measurments, NoOfMeas)
2     FilteredValue = (1/NoOfMeas) *sum(Measurments(k-m+1:k));
3 end
```

Kode 2.5: IIR funksjon

```
1 function [FilteredValue] = IIR_filter(OldFilteredValue, ...
2     Measurments, Para)
3     FilteredValue = Para * Measurments(k) + ...
4         (1-Para)*OldFilteredValue(k-1);
5 end
```

I koden 2.6 så kan man se hvordan man bruker/kaller på en tidligere definert funksjon.

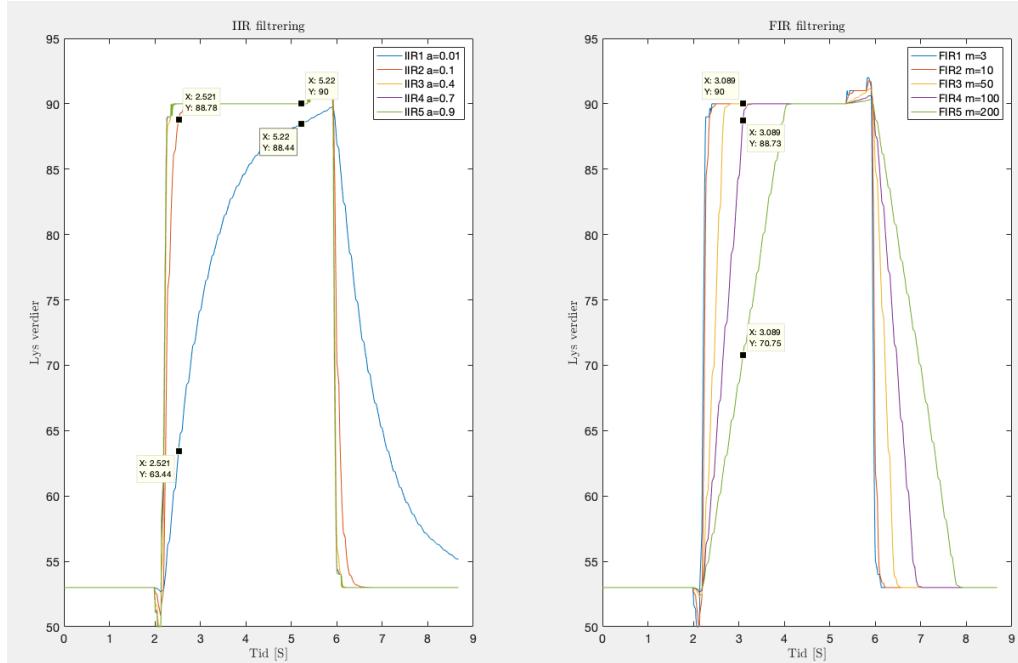
Kode 2.6: IIR funksjon brukt i kode

```
170 %IIR filtrering med alfa = 0.01
171 Temp_IIR(k) = IIR_filter(Temp_IIR(k-1), Temp_stoy(k), a)
```

2.6 Sammenligning

2.6 Sammenligning

Ved å sammenligne verdier, så ser man at ved IIR filtrering får man en raskere stigning i starten, men den bruker lengre tid til å komme opp til reell verdi. Kontra FIR filtrering som stiger med en lineær verdi.



Figur 2.5: FIR og IIR filtrering av lys med forskjellige verdier

Ser i figur 2.5 hvordan filtreringen påvirkes ved ulike verdier. Hvis man ser på IIR filtrering med lav a så vil ikke den filtrerte verdien nå sin reelle verdi før det skjer en ny endring.

Ved å observere FIR filtreringen i figur 2.5, så ser vi tydelig forskjell når vi tar gjennomsnittet av 200 verdier og 10 verdier. Ved høyerer m , desto langsommere blir endringen.

Hvilken filtreringsmetode som er best er avhengig av hvilken prosess som skal filtreres. Ønskes det en filtrering som når reell verdi raskt, så passer det bra med en FIR filtrering med lav m verdi.

2.7 Sammendrag

2.7 Sammendrag

Vi har i dette delkapittelet dukket dyperer i tematikken rundt filtrering. Vi har sett på hvordan man kan redusere målestøy ved bruk av et FIR eller et IIR filter og hvordan man implementerer dette i Matlab. Vi har sett på egenskapene til de to ulike filterene og hvordan de oppfører seg med ulike m og a verdier.

Kapittel 3

Numerisk derivasjon (utført av hele gruppen)

3.1 Problemstilling

Vi skal i dette delkapittelet gå nærmere inn på numerisk derivasjon. Ved å bruke lyssensoren og matlab skal det gjennomføres to eksperimenter. I det første eksperimentet skal vi vise hvordan politiets fartsmålere funker. Vi skal se på matematikken som ligger bak, implementerer en kode samt verifisere at kode funker.

I det andre eksperimentet skal vi se på avstander, fart og akselerasjon i en karusell kalt "Stupet". Karusellen beveger seg rett opp og ned langs en metallkonstruksjon. Denne bevegelsen vil gi et lyssignal på sinus-form. For å kunne dukke dypere inn i numerisk derivasjon vil vi først legge fram teorien bak.

Vi skal også vise at avstandsmålingene som blir tatt opp av lyssensoren må filtreres for å kunne brukes videre i derivasjonsbiten. Da bruker vi IIR-filteret som ble lagd i kapittel 2- filtrering.

3.2 Teori

3.2 Teori

Fra matematikken vet vi at den deriverte av en kontinuerlig funksjon er gitt ved:

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

Med andre ord så er den deriverte lik den øvre funksjonsverdien minus den nedre funksjonsverdien delt på endring i tid. Ved å la endring i tid gå mot 0 ($\lim_{\Delta t \rightarrow 0}$), vil vi få en tangent med tilnærmet lik stigningstall som punktet i funksjonen.

I praksis blir det litt annerledes. Vi kaller derivasjon i praksis, numerisk derivasjon. Når lysmåleren tar opp målinger, gjør den dette bare i gitte tidspunkt. Vi har derfor ikke data for alle t og må derfor sette Δt lik tidskrittet.

Siden vi underveis i målingene ikke har tilgang til informasjon som skal komme, må vi heller se bakover og bruke målinger vi allerede har. Med dette i bakhodet vil koden vår for numerisk integrasjon bli:

$$f'(k - 1) = \frac{f(k) - f(k - 1)}{T_s(k - 1)} \quad (3.2)$$

Formelen vi bruker for numerisk derivasjon

Her er T_s = Tidskrittet vårt.

3.3 Forslag til løsning

3.3 Forslag til løsning

Det første som må implementeres i koden er tidskrittet og avstanden. Dette er vist i kode 3.1. Siden vi bruker lyssensoren til å simulere avstand så lager vi en variabel $Avstand(k)$ som er lik som lysmålingene.

Kode 3.1: Utdrag av funksjonen `Prosjekt03_Numerisk_derivasjon.m`.

```
158      %Beregner tidsskrittet
159      Ts(k) = (Tid(k) - Tid(k-1));
160
161      %Setter Avstand lik lyset som blir reflektert fra ...
162          lysensoren
162      Avstand(k) = Lys(k);
```

Deretter er vi nødt til å filtrere avstanden. Konsekvensen av å ikke filtrere avstanden kommer vi tilbake i senere i kapitellet. Filtrering er vist i kode 3.2.

Kode 3.2: Utdrag av `Prosjekt03_Numerisk_derivasjon.m`. Filtrerer avstand. Para lik 0.2. Kaller på numerisk integrasjon formelen

```
164      %Filtrerer avstanden
165      Avstand_filtrert(k) = para*Avstand(k) + ...
165          ((1-para)*Avstand_filtrert(k-1));
```

Filtrer avstanden med para lik 0.2. Dette vil si at 20 prosent av den nye avstandsverdien og 80 prosent av den gamle avstandsverdien blir brukt. Dette gir et mye stødigere signal å jobbe med. Deretter kaller vi på numerisk derivasjonsfunksjonen. Dette er vist i kode 3.2.

Koden for numerisk derivasjon er vist i kode 3.3.

Kode 3.3: Utdrag av `derivasjon.m`. Numerisk derivasjon

```
1 %Funksjon numerisk derivasjon
2 function [sekant] = derivasjon(Ts, Avstand)
3 sekant = (Avstand(2)-Avstand(1))/Ts;
4 end
```

Her ser vi prinsippet for numerisk derivasjon bli tatt i bruk. Man tar avstanden i nåværende indeks minus avstanden i forrige indeks og deler på tidsskrittet.

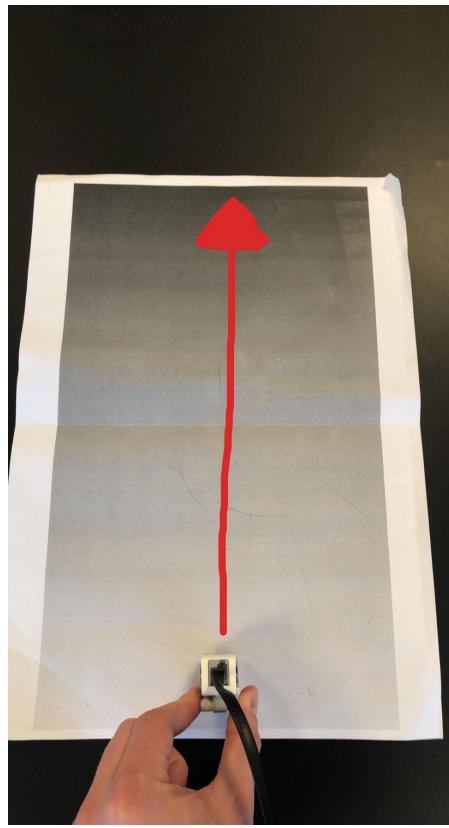
3.4 Resultat

3.4 Resultat

3.4.1 Derivasjon av et lineært signal

Nå som vi vet hvordan numerisk derivasjon funker og koden som tas i bruk er fremvist, gjenstår det å begynne selve eksperimentet og ta opp et signal. Vi skal vise hvordan politiets fartsmålere funker og trenger da et lineært signal. Dette er for å fremvise en stødig og konstant fart når signalet deriveres.

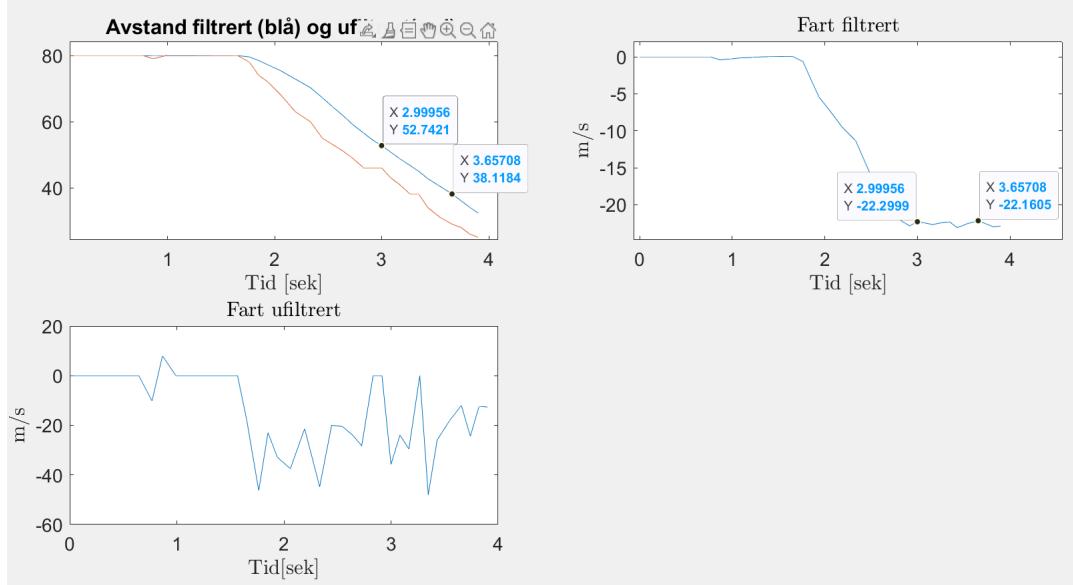
Avstandsmålingene som er definert som lys reflektert blir tatt opp ved at vi beveger lyssensoren med en jevn fart over et svart-hvitt skalert ark. Fremgangsmåten er vist i figur 3.1



Figur 3.1: Bevegelsesmønster for lyssensor

3.4 Resultat

Når vi beveger lyssensoren vist i figur 3.1 får vi opp 4 ulike signaler/funksjoner. Vi får avstand filtrert og ufiltrert, samt fart filtrert og ufiltrert. Disse signalene er vist i Figur 3.2.



Figur 3.2: Resultat for lineær bevegelse av lyssensor

Fra figur 3.2 ser man at signalet vi får for fart når vi bruker de ufiltrerte avstands-målingene er ubruklig. På andre siden ser man at signalet for fart når vi bruker den filtrerte avstanden er godt.

For å verifisere at koden for numerisk integrasjon stemmer, må vi regne ut stigningstallet i et gitt intervall fra avstandsmålingene og se om det deriverte uttrykket i det samme intervall har tilsvarende y-verdi.

Vi ser fra avstandsignalet at stigningstallet er:

$$stigningstall = \frac{38.12 - 52.74}{3.66 - 3.00} = -22.15 \quad (3.3)$$

Fra grafen for filtrert fart leser vi av en gjennomsnittlig verdi på -22.2. Med dette kan vi si at koden vår for numerisk derivasjon stemmer. Hadde dette vært en fartsmåler, så hadde vi målt at farten på bilen er lik 22.15 m/s. Dette er

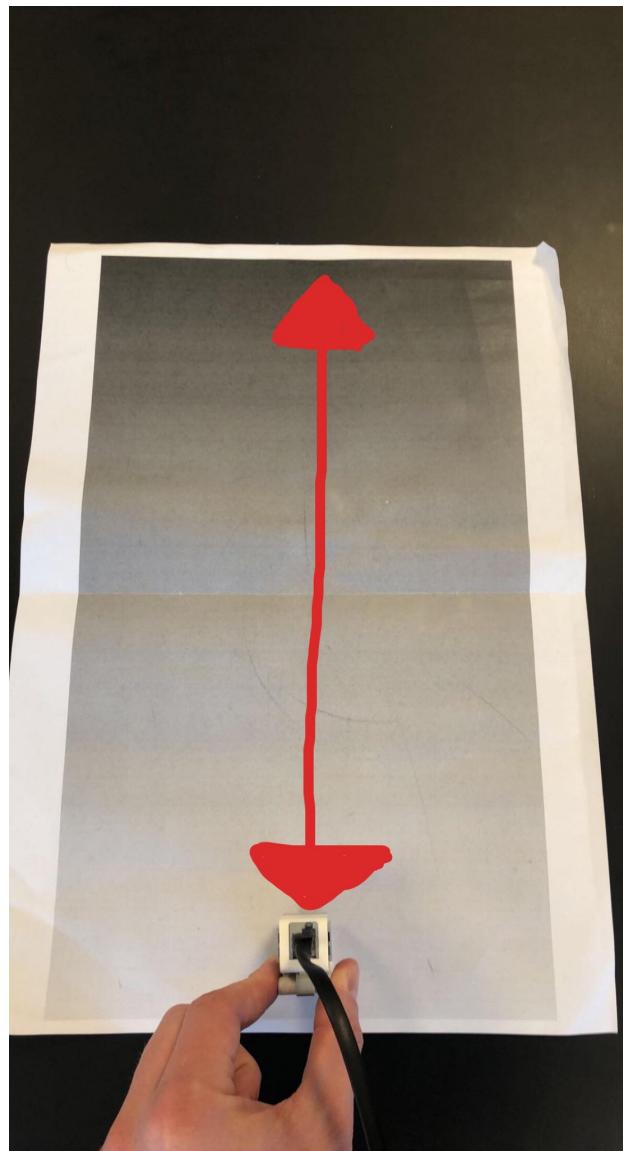
3.4 Resultat

79.74 km/t. Vi kan også konkludere med at bilen kjører mot fartsmåleren, siden vi får negativt fortegn siden avstanden minker. Dette samsvarer også godt med avstandsgrafen, hvor vi ser at avstanden minker og minker.

3.4.2 Derivasjon av et sinussignal

I det andre eksperimentet skal vi se på et sinusformet signal som kommer ved simulering av karusellen "Stupet". Stupet beveger seg rett opp og ned langs en akse. Dette fremstiller vi ved å bevege lyssensoren fram og tilbake på et svart-hvit skalert ark, vist i figur 3.3.

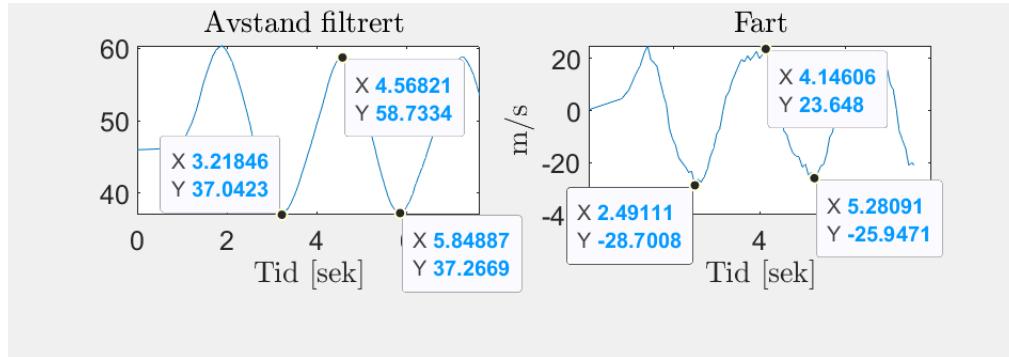
3.4 Resultat



Figur 3.3: Bevegelse for simulering av Stupet"

Ved å bevege lyssensoren som vist i figur 3.3 får vi følgende signaler, vist i figur 3.4.

3.4 Resultat



Figur 3.4: Signal tatt opp ved simulering av Stupet". Delfiguren til venstre viser avstanden til setene på karusellen. Delfiguren viser farta til setene i karusellen.

For å verifisere koden i dette eksperimentet skal vi deritere en korresponderende sinusfunksjon med konstantledd b , amplitude a og frekvens w [rad/s] som vist i ligning 3.4.

$$\frac{d}{db}(b + a \sin(wt)) \quad (3.4)$$

Vi kan lese av fra signalet som viser avstanden til setene på karusellen at konstantleddet b er 46. Amplituden er:

$$amplitude = \frac{y_{maks} - y_{min}}{2} = \frac{58.73 - 37.27}{2} = 10.73 \quad (3.5)$$

For å finne vinkelfrekvensen w må vi bruke formelen:

$$w = \frac{2\pi}{T} \quad (3.6)$$

hvor T er periodetiden. Da får vi:

$$w = \frac{2\pi}{5.85 - 3.22} = 2.39 \quad (3.7)$$

Da får vi det analytiske uttrykket vist i figur(eq?) 3.8.

$$46 + 10.73 \sin(2.39t) \quad (3.8)$$

3.4 Resultat

Ved å derivere denne funksjonen skal vi få et en funksjon som uttrykker fartssignalet. La oss sjekke at dette stemmer:

$$\frac{d}{dt}(46 + 10.73\sin(2.39t)) = 25.65\cos(2.39t) \quad (3.9)$$

Fra dette uttrykket kan vi lese av at konstantleddet b skal være 0, amplituden a skal være 25.65 og vinkelfrekvensen w skal være 2.39.

Ved å lese av funksjonen for fart i figur 3.8 kan vi se at konstantleddet b er lik 0. Amplituden a er:

$$a = \frac{23.65 - (-28.7)}{2} = 26.17. \quad (3.10)$$

og vinkelfrekvensen w :

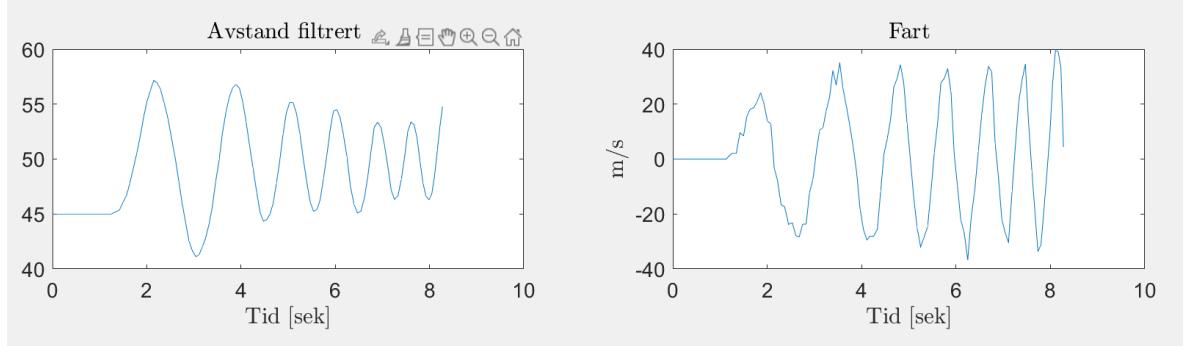
$$w = \frac{2\pi}{T} = \frac{2\pi}{5.28 - 2.49} = 2.25 \quad (3.11)$$

Ved å sammenligne grafen for fart i figur 3.4 og det matematiske uttryket i ligning 3.9, ser vi at verdiene for b , a og w er tilnærmet like og vi kan verifisere at koden for numerisk derivasjon stemmer. Grunnen til at vi får avvik er at vi selv skal lage et sinussignal med bruk av lyssensoren.

3.4.3 Derivasjon av ulike typer signaler

I dette delkapitlet skal vi se på hva som skjer med farten og avstanden til karusellen når vi har ulike vinkelfrekvenser og amplituder. Det første vi skal se nærmere på er chirp-signal. Et chirp signal er et signal hvor frekvensen til signalet øker med tiden. Vi vil få et signal som vist i figur 3.5.

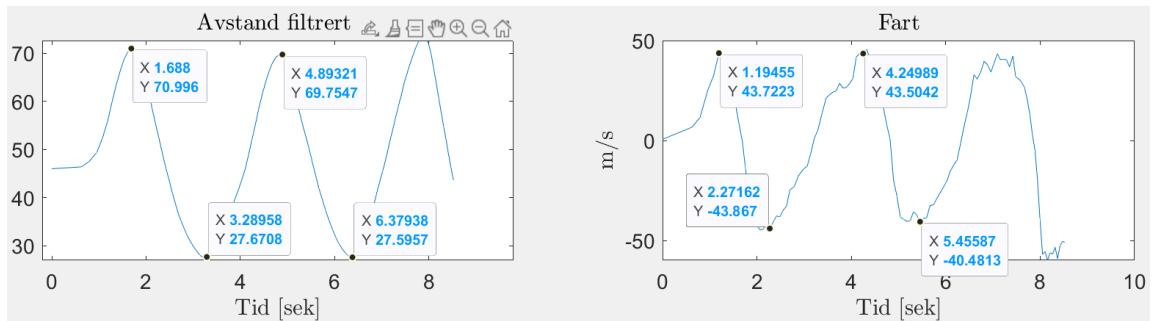
3.4 Resultat



Figur 3.5: Signal tatt opp ved simulering av Stupetpå chirp form.

Som man kan observere i figur 3.5 øker frekvens med tiden. Observerer også at amplituden til avstandssignalet minker med høyere frekvens. Dette er fordi "vognen" i stupet får mindre tid til forflytte seg opp og ned, fordi frekvensen øker. Hvis man ser på fartssignalet ser man at vognen holder tilnærmet lik toppfart gjennom økningen av frekvens på oppturen og nedturen gjennom eksperimentet.

Neste eksperiment er å simulere et signal med større amplituden. Med andre ord, at karusellen beveger seg lengre opp og ned langs aksen. Dette utføres ved å bevege lyssensor lengre over det hvit svart skalerte arket. Da får vi signalet vist i figur 3.6.



Figur 3.6: Signal tatt opp ved simulering av Stupet"med større bevegelse

Fra figur 3.6 kan man observere at vognen beveger 40 meter fra bunn til topp. i eksperimentet vil vinkelrekvensen synke, men ikke i takt med økningen av amplituden. Siden amplituden a i fartssignalet er gitt ved amplituden ganger vinkelrekvensen i avstandssignalet, så vil vi få en større fart. Dette kan vi verifisere ved

3.4 Resultat

å sammenligne med chirp signalet, hvor vi kan se at farten er lavere enn signalet fra figur 3.6.

3.4.4 G-krefter i Stupet".

På hjemmesiden til fornøyelsesparken reklameres det for at man opplever 3.2g i karusellen Stupet". Avslutningsvis i dette kapitelet skal vi se på g-kreftene man føler i karusellen.

Ved å derivere fart får man akselerasjon. Tyngdekraften på jorden er 1g og vi er derfor ute etter at amplitudeleddet til akselerasjonsfunksjonen. For å vise at tyngdekraft og akselerasjon henger sammen så kan vi se på benevningene. Bennevningen til tyngdekraften, g , og til akselerasjon kan skrives som m/s^2 . Ved å dobbelderivere ligning 3.4 får vi:

$$- A * w^2 * \sin(wt) \quad (3.12)$$

Siden vi ikke har noe fart å jobbe utifra, setter $A = 1$. Herifra isolerer vi amplitudeleddet og setter det lik 2.2g:

$$A * w^2 = 2.2 \quad (3.13)$$

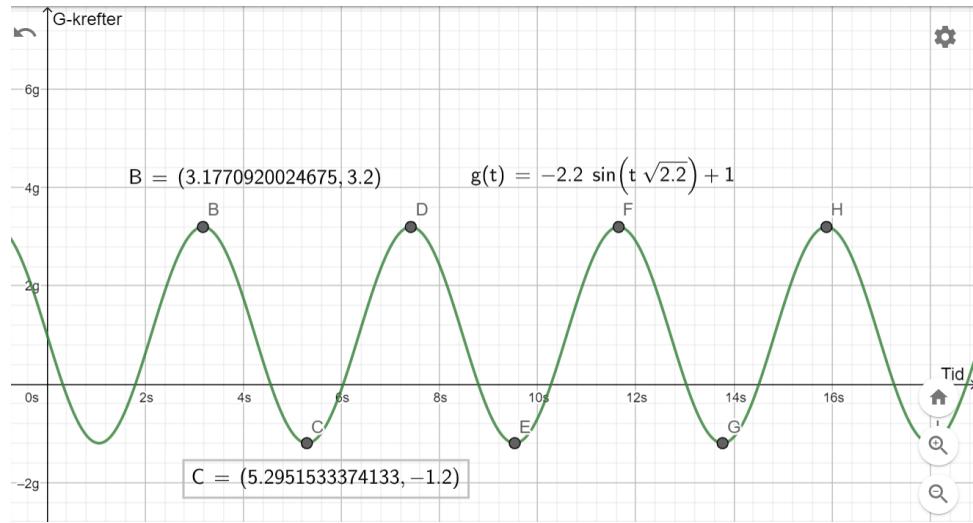
$$w = \sqrt{2.2} \quad (3.14)$$

Setter vi denne vinkelfrekvensen inn i ligning 3.12 og legger på 1g fra tyngdekraften på jorden får vi:

$$Akselerasjon(t) = -2.2\sin(\sqrt{2.2}t) + 1 \quad (3.15)$$

Funksjonen fra ligning 3.15 er vist i figur 3.7.

3.5 Sammendrag



Figur 3.7: G-kreftene i Stupetfremvist som en funksjon.

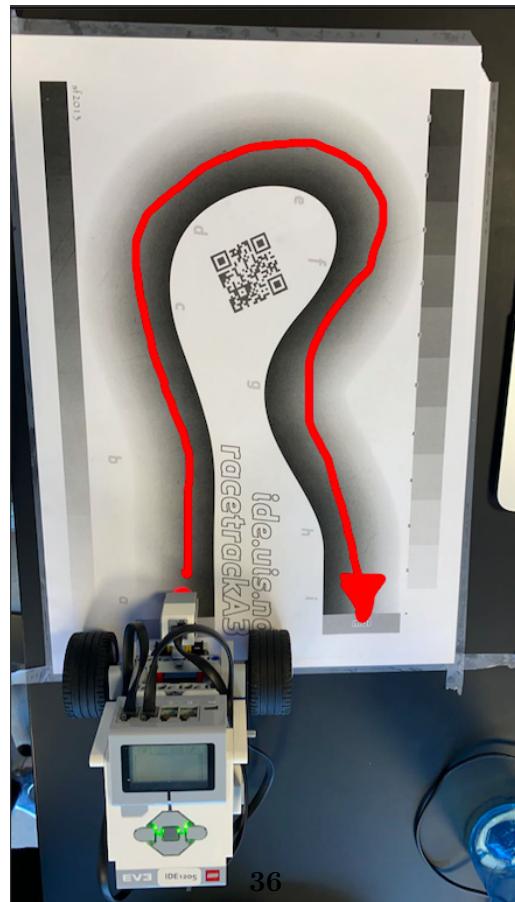
Med vinkelfrekvens $\sqrt{2.2}$, periodetid på 4.23 sekunder og amplitude -2.2, så vil man oppleve 3.2g ved tiden 3.18 sekunder.

3.5 Sammendrag

I kapittelet om numerisk derivasjon har vi gjennomført to eksperimenter. I det første eksperimentet så vi på hvordan politiets fartsmålerer fungerte, nemlig ved å numerisk derivere avstand. I det andre eksperimentet så vi på avstander, fart og akselerasjon i en karusell kalt "Stupet", som vil gi et lyssignal på sinus-form.

Kapittel 4

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



Figur 4.1: Banekort

4.1 Forslag til løsning

Nå har vi kommet så langt at vi kan begynne på manuell kjøring. Her skal vi først bygge roboten etter bruksanvisning, for å så kjøre den rundt en bane ved hjelp av en styrestikke. Kjørebanen er vist i figur 4.1. Kvaliteten på kjøringen vil bli vurdert av målinger tatt opp av en lys sensor. Den første lys verdien som blir registrert vil bli brukt som referanse, mens målingene etter blir vurdert mot referansen. Dette betyr i praksis at for å få minst mulig avvik, vil man gjennom hele kjøringen holde seg på samme gråtone som der man startet. Utifra dette vil vi regne ut forskjellige kvalitetsmålinger og konkurere innad i gruppen om hvem som er best til å kjøre.

4.1 Forslag til løsning

Når vi fikk bygd sammen EV3 etter bruksanvisning måtte vi finne hvordan vi kunne styre EV3 med styrestikken. En ferdiglagd fil, *Joytest_mac*, ble brukt. Vi bevegde styrestikken for å lese av verdier. Ettersom den bruker samme akse på framover/bakover bevegelse og en annen akse for sidelengs bevegelse, så konstruerte vi funksjonene vist i 4.1.

Kode 4.1: Styrestikken for *Prosjekt04_Kjoring.m*

```
122      % Data fra styrestikke. Utvid selv med andre knapper og ...
123          akser
124      if ismac
125          skalering = 100;           % konvertering fra 1 til 100%
126          JoyMainSwitch = button(joystick,1);
127          JoyForover(k) = -skalering*axis(joystick,2);
128          JoySideslengs(k) = skalering*axis(joystick,3);
```

Signalet fra styrestikken gir verdier fra 0 til 1. Ved å skalere dette signalet, så vil vi istedet få verdier fra 0 til 100. Når styrestikken beveges framover så vil signalet få negativ verdi. Derfor legger vi til et minus foran skaleringingen, ellers ville styrestikken fungert motsatt vei. Likt blir det for styring sidelengs.

4.1 Forslag til løsning

Kode 4.2: Motorstyring Prosjekt04_Kjoring.m

```
159 %Faktor for motorp?drag%
160 a=0.3;
161 b=0.3;
162
163 %P?drag sendt ut til motor A og B%
164 PowerA(k) = (a*JoyForover(k)) + (b*JoySideslengs(k));
165 PowerB(k) = (a*JoyForover(k)) - (b*JoySideslengs(k));
```

Bruker a og b som parametere for motorpådrag, ved a og $b = 1$ vil vi få en høy effekt og roboten vil kjøre raskt. Når vi har lavere a og b er det lettere å få kontrollert roboten.

Når vi skal sende ut pådrag til motorene så bruker vi Joyforover + Joysidelengs. Fra kode 4.2 ser vi at hvis Joysidelengs er + 100 og Joyforover er 0 vil bare motor a akselerere mens motor b vil stå i ro. Dette gjør at roboten vil svinge. Likt blir det motsatt vei, hvis Joysidelengs er - 100 og Joyforover er 0 så vil motor b akselerere mens motor a står i ro.

Kode 4.3: Stop ved hvit linje Prosjekt04_Kjoring.m

```
177 % Hvis lysensor treffer hvit ark stopper programmet%
178 if Lys(k) > 52
179 JoyMainSwitch = 1;
```

I kode utdrag 4.3, fremviser vi koden som stopper programmet når roboten kjører utfør banekartet og ut på det hvite. Hvis lys verdien blir høyere en 52, så vil koden sette JoyMainSwitch til 1. Da stopper programmet og vi får opp de kalkulerte verdiene til nå. Banekartet kan vi se igjen i figur 4.1.

Kode 4.4: Referansepunkt Prosjekt04_Kjoring.m

```
195 %Setter referansepunkt likt f?rste lysm?ling%
196 Referansepunkt(1:k) = Lys(1);
```

i kode 4.4 lages en variabel for referansen som er den første lysmålingen. Denne brukes til å videre beregne hvordan den som kjører ligger iforhold til fargelinjen.

4.1 Forslag til løsning

Kode 4.5: Beregning av avvik Prosjekt04_Kjoring.m

```
198      %Beregner avvik og absoluttverdien til avvik%
199      Avvik(k) = Referansepunkt(k) - Lys(k);
200      Abs_avvik(k)=abs(Avvik(k));
```

I koden 4.5, linje 199 beregner vi avviket. Tar referansepunkt - lys(k). Referansepunkt er en fast variabel mens lyset varierer. Når vi tar abs(avvik) så får vi absolutt verdien til avviket som vi bruker viderer for å regne ut *MAE* og *IAE*. Da vil det absolute avviket alltid adderes på verdien. Hadde vi brukt det vanlige avviket ville verdien synket når lysverdien ble mindre enn refereansen.

Kode 4.6: Beregning av MAE og IAE Prosjekt04_Kjoring.m

```
211      %Beregning av Mean Absolute Error (MAE)%
212      MAE(k) = (Abs_avvik(k) + sum(Abs_avvik(1:k-1))) / k;
213
214      %Beregning av Integral of Absolute Error (IAE)%
215      IAE(k) = IAE(k-1) + (Abs_avvik(k) * (Tid(k)-Tid(k-1)));
```

MAE (Mean of Absolute error) kalkuleres ved hjelp av formelen vist i ligning 4.1. Da finner vi summen av avvikene normalisert i forhold til antall målinger. MAE forteller oss i denne oppgaven gjennomsnittet av avviket per måling på den som har kjørt. Stor MAE verdi vil si at den som har kjørt har høyt avvik fordelt på antall målinger.

$$MAE = \frac{1}{k} \sum_{n=1}^k |e(n)| \quad (4.1)$$

IAE (Integral of Absolute Error) forteller oss summen av arealet under grafen til avviket. IAE summerer alle verdiene under veis og vil fortelle oss hvor mye feil som har blitt gjort totalt til slutt. Formelene for IAE er vist i ligning 4.2

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

TV (Total Variation) sier noe om kvantifisering av pådragsbruket. Ved høyt og variert pådragsbruk vil TV verdien bli høy og motoren vil bli utsatt for mer

4.1 Forslag til løsning

slitasje. I situasjoner der det skjer raske avvik så vil en prøve å få minsket avviket raskt. Da vil pådragsbruken bli høyt, og vil gi slitasje på utstyret. Ved å akseptere litt avvik vil en få mindre pådragsbruk og hindre slitasje på utstyr. For å regne ut TV så bruker vi summen av absolutt verdien til nåværende pådrag minus forrige pådrag. U_n er motorpådraget. Formelen for å regne ut dette er vist i 4.3.

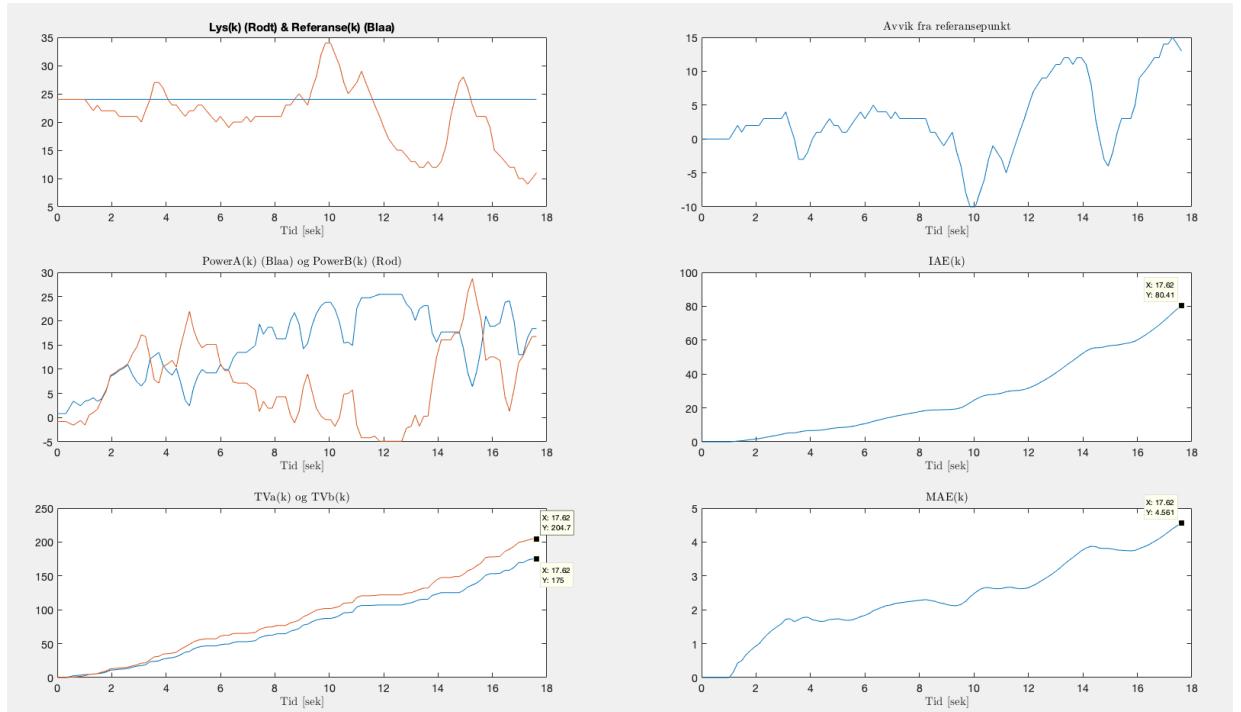
$$TV = \sum_{n=1}^k |u(n) - u(n-1)| \quad (4.3)$$

4.2 Resultat

4.2 Resultat

Etter at EV3 har kommet og programmet er ferdig, så får vi flere grafer. Fra disse kan vi lese alt av verdier som vi trenger for å analysere kjøringen. Lyssensoren er ikke av høyeste kvalitet så vi godtar avvik.

Observerer fra 4.2 hvordan *PowerA* og *PowerB* jobber mot hverandre i svinger og hvordan den samles igjen på rette strekninger. *PowerA* er venstre motor mens *PowerB* er høyre motor. Ved rundt 8 sekunder så kan vi se at svingen begynner. Da kan vi se på figur 4.2 at motor *A* vil akselerere mens motor *B* vil akselerere mye mindre og på et punkt snu dreieretning. Når roboten har kjørt i 14 sekund så kan vi se at den har kjørt ut av svingen og *PowerA* og *PowerB* samles igjen.



Figur 4.2: Verdier etter endt kjøring

De ulike gruppemedlemmers resultater fra kjøring gjennom løypa er presentert i figur .

4.2 Resultat

Tabell 4.1: Sluttresultater fra manuell kjøring til deltagere i gruppe 2205.

	Sondre	Tormod	Truls	Aleksa
Plattform	Mac	Mac	Mac	Mac
Strømkilde	Batteri	Batteri	Batteri	Batteri
Samtidig plotting	Nei	No	Nicht	Nein
Referanse	24	16	24	23
middelverdi μ	11	9	20	25
$ \text{Referanse}-\mu $	13	7	4	2
standardavvik σ	5.35	4.42	8.18	7.09
kjøretid [sek]	17.62	15.10	13.63	16.18
IAE	80.4	47.03	87.08	89.82
MAE	4.56	3.16	6.35	5.58
TV_A	174.99	246.63	196.14	414.07
TV_B	204.75	255.67	172.42	403.08
middelverdi av T_s [sek]	0.164	0.162	0.168	0.170
antall målinger (k)	108	94	81	96

Verdiene vi ønsker for god kjøring er et lavt standardavvik (σ), lav IAE, lav MAE, lav TVa og TVb. Den som kjører best avhenger av hvilke av måleverdiene vi ser på som viktigst.

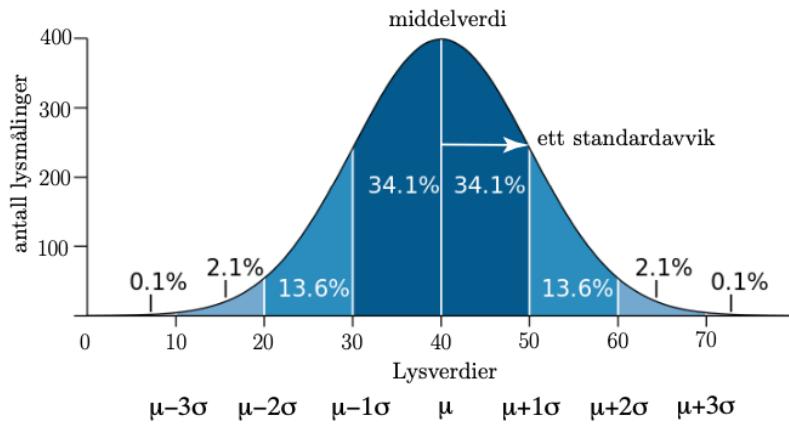
Tormod kjørte mest nøyaktig med minst standardavvik innenfor en god tid. Bruker litt mer motorpådrag for å holde EV3 på på ret lys-linje.

Truls brukte kortest tid og hadde minst pådrag på motoren, dette gir mening etter som han lot det bli mer avvik for å komme seg fortare i mål og korrigerte ikke fullt så mye på kjøringen når det oppsto avvik.

4.3 Histogram og sammenligning

4.3 Histogram og sammenligning

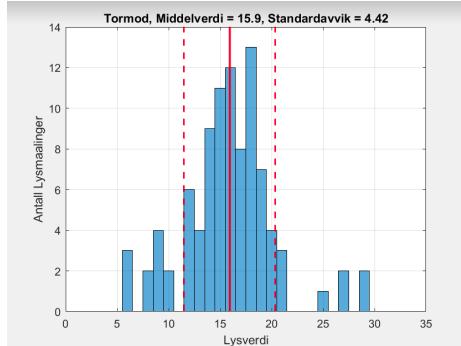
For å se resultatene etter kjøring på en lett måte brukes ofte en graf for normalfordeling. I figur 4.3 ser vi et eksempel på en normalfordeling. Her kan vi på en god måte observere avviket til den som har kjørt.



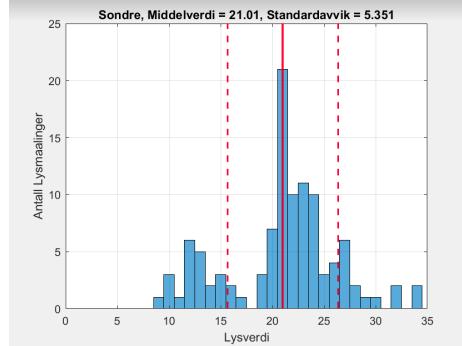
Figur 4.3: Eksempel på histogram med normalfordeling, ser hvordan 68,2% av alle målte verdier er innenfor et standardavvik

Ved å se på histogramene i figuren 4.4, 4.5, 4.6, 4.7 til alle som har kjørt kan en vurdere hvordan de har kjørt gjennom svingen. Ved lave lysverdier så har føreren kjørt mer mot den mørkerer delen av skalaen. For eksempel så har Truls mange lysverdier under referansen. Det vil si at han har kjørt på mørkere rute enn startpunktet og vært i innersvingen. Dette er vist i figur 4.6.

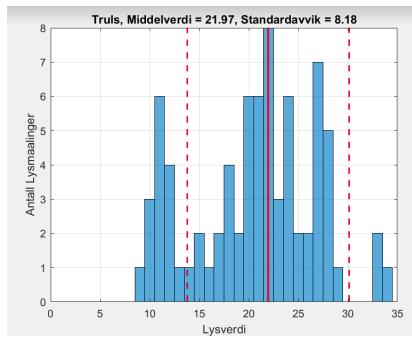
4.4 Sammendrag



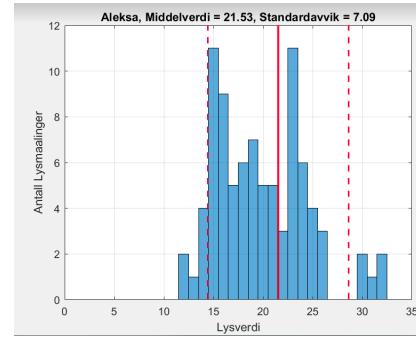
Figur 4.4: Histogram for Tormod, ikke normalfordelt, 84.6% er innenfor et standardavvik.



Figur 4.5: Histogram for Sondre, ikke normalfordelt, 73.2% er innenfor et standardavvik.



Figur 4.6: Histogram for Truls, ikke normalfordelt, 63.5% er innenfor et standardavvik.



Figur 4.7: Histogram for Aleksa, ikke normalfordelt, 88.6% er innenfor et standardavvik.

Etter vi har regnet ut normalfordelingen til alle som har kjørt kan vi se at ingen av grafene er normalfordelt(68.2%). Det vil si at kjøringene er bedre enn normal fordelt og ligger nærmere referansen enn en normalfordelt kjøring ville ha gjort. Ved å ha en større bane og lengre kjøretid hadde vi fått et annet resultat som kunne ha lignet mer på en normalfordeling.

4.4 Sammendrag

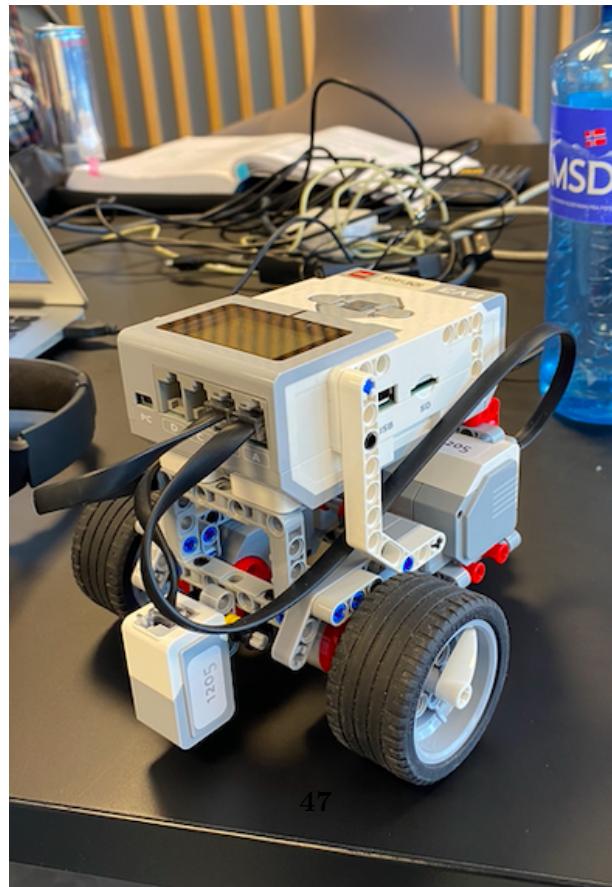
Vi har i dette delkapittelet for manuell kjøring, kjørt lego roboten gjennom en svart hvit skalert bane. Lego roboten har gjennom løypa tatt opp lysverdier, som vi har brukt videre i beregninger av kvalitetsmålinger som f.eks IAE (integral of

4.4 Sammendrag

absolute error) og MAE (mean of absolute error). Vi har utifra kvalitetsmålene drøftet rundt hvilken av gruppemedlemmene som har best kjøring.

Kapittel 5

Automatisk kjøring av Lego-robot (Utført av bare roboten)



5.1 Problemstilling

5.1 Problemstilling

Etter vi ble lei av å kjøre roboten ønsket vi at den kunne kjøre selv. Ved bruk av lyssensor og motorer, samme utstyr som i manuell kjøring så endret vi litt på koden slik at den kan kjøre av seg selv ved hjelp av *PID* regulering. Etter den har gjennomført løypen kan vi sammenligne resultater å se hvem som er best til å kjøre.

5.2 Forslag til løsning

Formelen som vi skal bruke i denne oppgaven er kalt PID- regulering, vi ser den nedenfor i equation 5.1

$$u(t) = u_0 + \underbrace{K_p * e(t)}_P + \underbrace{\int_0^1 K_I * e(\tau) d\tau}_I + \underbrace{\frac{d}{dt}(K_D * e_f(t))}_D \quad (5.1)$$

K_P , K_I K_D og U_0 er faste verdier som bestemmes av bruker (5.4). For at begge motorene ikke skal gå i samme retning så trekker vi fra *PID*-leddene på motorB og adderer på motorA. Dette kan vi se i kode 5.5

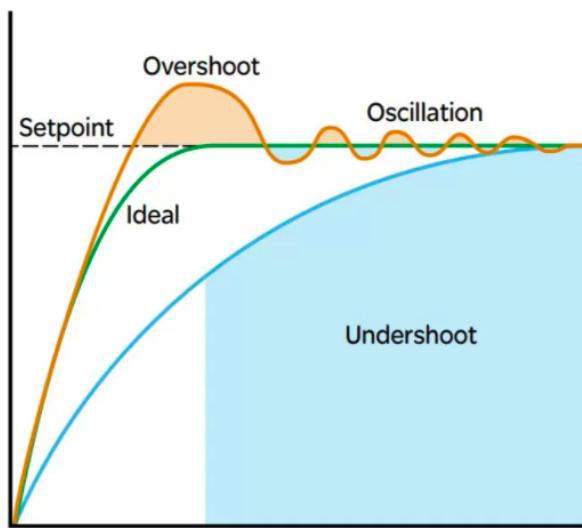
En begynner med å bare bruke P leddet, P leddet er konstanen K_P ganget med avviket $e(t)$. Kodden for P ser vi her 5.1. For å ikke få for mye pådrag på motoren slik at den spinner rundt, så setter vi en begrensning på P leddet.

Kode 5.1: *P*-leddet utregning *Prosjekt05* automatisk kjøring

```
222
223      %I-leddet, PID REGULERING%
224      Avvik_konstant(k) = Avvik(k-1) * Ki;
225      I(k) = EulerForward(I(k-1), Avvik_konstant(k), Ts(k-1));
226      if abs(I(k)) > 25
```

5.2 Forslag til løsning

Kan deretter legge til I leddet for å hjelpe til. I -leddet tar integralet til avviket ganget med konstanten K_I . Ettersom med bare P regulering så vil en aldri oppnå 0 avvik siden den regulerer på avvik, da den kommer til 0 avvik vil den stoppe å regulere. Da kommer I leddet og bruker integralet til avviket slik at pådraget ikke stopper når avviket nærmer seg 0. Ved høy K_I vil vi få en "overshoot", dette gir en kjapp regulering imot at målt verdi vil gå over sett verdien. Ved lav K_I verdi vil vi få "undershoot", som gir en sakte men fin regulering som ikke vil overstige settpunktet. Kan se dette i figur 5.2. Koden for I -leddet 5.2



Figur 5.2: "Overshoot" og "undershoot"

Kode 5.2: I -leddet utregning Prosjekt05 automatisk kjoring

```
228     end  
229  
230     %D-leddet, PID REGULERING%  
231     Avvik_filtrert(k) = (para*Avvik(k) + ...  
     ((1-para)*Avvik_filtrert(k-1)))*Kd;  
232     D(k) = derivasjon(Ts(k-1), Avvik_filtrert(k-1:k));
```

Til slutt kan en også legge til et D ledd for å optimalisere prosessen enda mer. D vil begrense optimaliseringen ifra å "overshoot", når målt verdi nærmer seg settpunktet. Da vil vi få en rask korreksjon på avviket men vil ikke overstige settpunktet.

5.2 Forslag til løsning

Kode 5.3: D-leddet utregning Prosjekt05 automatisk kjoring

```
235     PowerA(k) = a - P(k) - I(k) - D(k);  
236     PowerB(k) = a + P(k) + I(k) + D(k);
```

U_0 (kalt a her) er den konstante akselrasjon. Denne er vi nødt til å ha her for at roboten skal kjøre fremover å ikke bare stoppe opp og spinne i rundt.

Kode 5.4: Variabler for pådrag Prosjekt05 automatisk kjoring

```
215     para = 0.3;  
216  
217     %P-leddet, PID REGULERING%  
218     P(k) =(Kp * Avvik(k));  
219     if abs(P(k)) > 30  
220         P(k) = P(k-1);
```

For å finne de korrekte verdiene til PID -regulering bruker vi i dette prosjektet 'brute force', som er prøving og feiling til vi oppnår verdier som gir en bra regulering. Disse verdiene kan og regnes ut, men det får vi vite mer om i reguleringssteknikk.

Kode 5.5: Verdiene som styrer motorpådrag. Prosjekt05 automatisk kjoring

```
239     TVa(k) = sum(abs(PowerA(k)-PowerA(k-1)) + TVa(k-1));  
240     TVb(k) = sum(abs(PowerB(k)-PowerB(k-1)) + TVb(k-1));  
241     end
```

5.3 Resultat

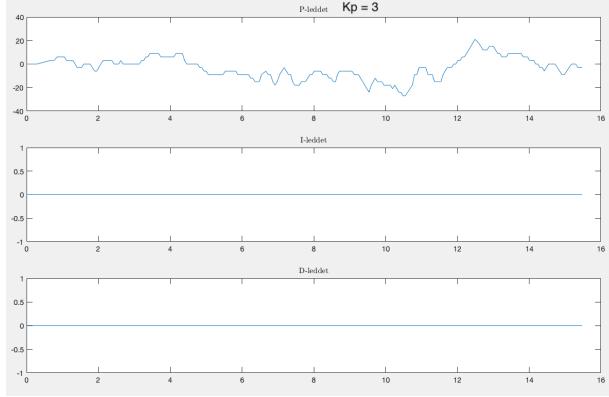
5.3 Resultat

	P-reg	P-I-reg	P-I-D-reg	Sondre
Plattform	Mac	Mac	Mac	Mac
Strømkilde	Batteri	Batteri	Batteri	Batteri
Samtidig plotting	Nei	Nei	Nei	Nei
Referanse	21	15	14	24
middelverdi μ	22	16	14	20
Referanse- μ	1	2	0	4
standardavvik σ	3.13	2.53	1.593	8.18
kjøretid [sek]	15.48	14.50	14.10	13.63
IAE	39.89	26.10	17.15	87.08
MAE	2.65	1.87	1.24	6.35
TV_A	372	324.31	648.85	196.14
TV_B	372	324.31	648.85	172.42
middelverdi av T_s [sek]	0.084	0.084	0.061	0.168
antall målinger (k)	246	173	230	81

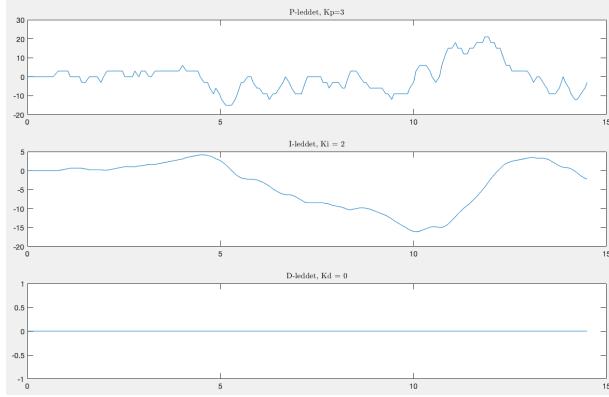
Tabell 5.1: Sluttresultater fra automatisk kjøring av roboten sammenligning mellom P-regulering, PI-regulering og PID-regulering og enkeltperson

Her kan vi se hvor bra kjøringen blir ved de forskjellige reguleringene. Ved bruk av bare P regulering vil en få en bedre kjøring enn det vi greide selv. Kan ha noe med at Sondre ikke kan kjøre i det hele tatt. Når en legger til flere optimaliseringsledd kan vi se at reguleringen blir bedre og bedre. IAE og MAE blir lavere mens TV_a og TV_b blir større ettersom roboten bruker mer pådrag for å holde roboten på rett lysverdi.

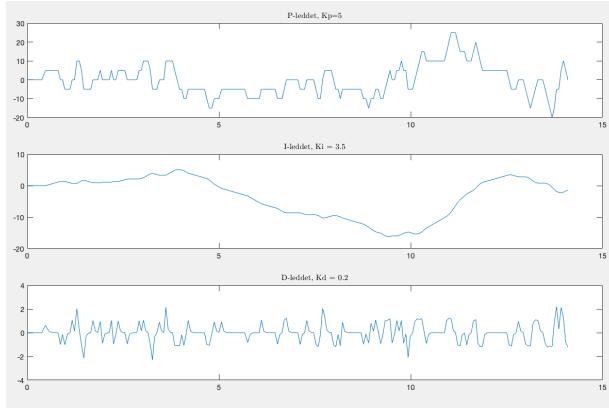
5.3 Resultat



Figur 5.3: P-regulering



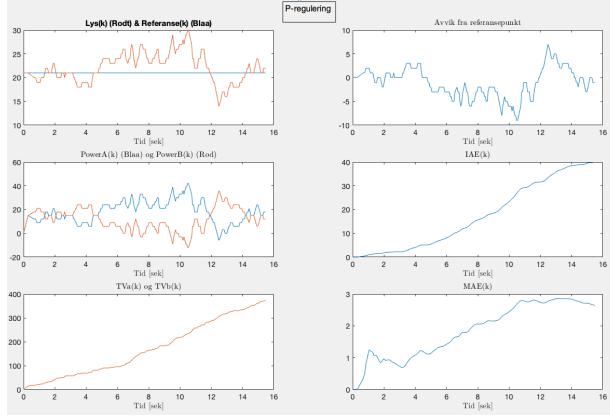
Figur 5.4: PI-regulering



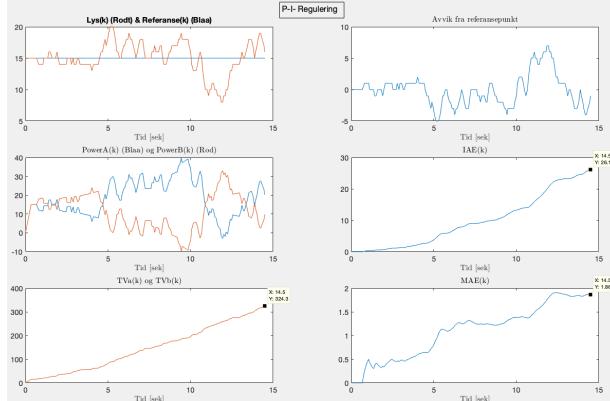
Figur 5.5: PID-regulering

Figur 5.6: De forskjellige PID verdiene til hver simulering, ser hvor stor påvirkning hvert enkelt bånd har på prosessen.

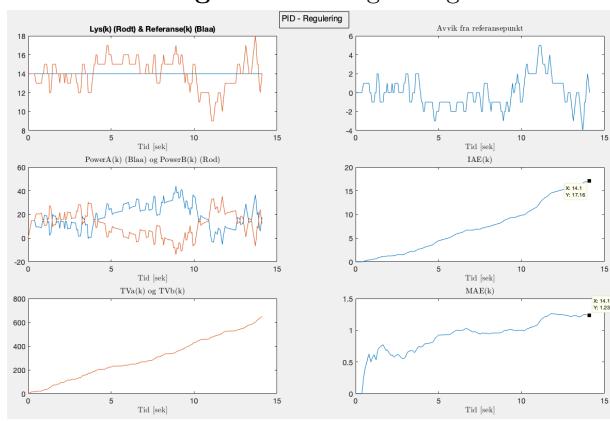
5.3 Resultat



Figur 5.7: P-regulering



Figur 5.8: PI-regulering



Figur 5.9: PID-regulering

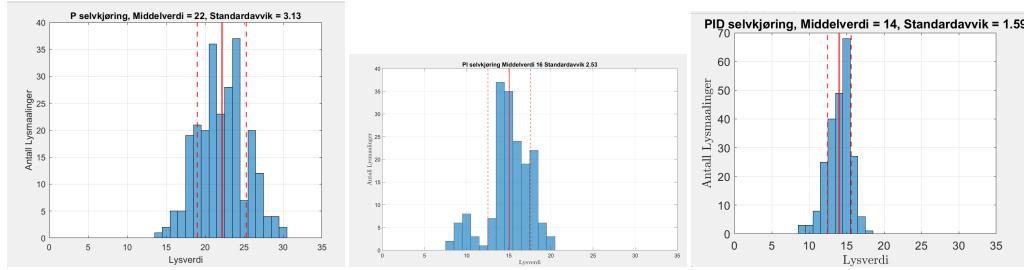
Figur 5.10: Ser hvordan avviket fra referansepunktet varierer i forhold til de forskjellige simuleringene med PID verdier

5.4 Sammenligning

I grafen 5.9 for PowerA og PowerB kan en se hvordan motorene jobber mot hverandre i svingen, og hvor den kjører på en strak linje.

Hvis vi ser på avviket fra referansepunktet til den første grafen 5.7 så kan vi se at det er veldig hakkete når avviket nær 0. Dette er fordi vi bare bruker P -leddet til å regulere roboten med. Når avviket blir 0 så vil den stoppe å regulere og gå rett frem. Da får vi et nytt avviket med en gang som resulterer i at vi får så hakkete grafer.

Ved å se på motorpådraget til de forskjellige ser vi hvordan ved PID -regulering at det er stabilt og ikke så høye verdier. I forkjell til PI -regulering hvor det er høyere og mer ustabilt pådrag.



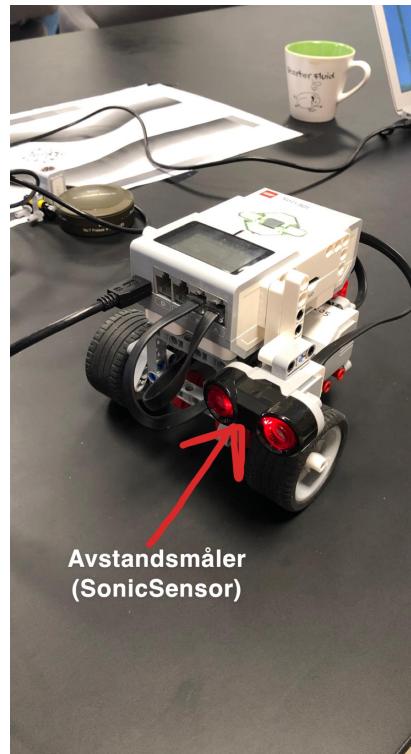
Figur 5.11: Kan se fra histogrammet hvordan flere lysmålinger er innforbi et standardavviket, fra når vi bruker bare P -leddet til å bruke PI -ledd og til slutt PID -ledd

5.4 Sammenligning

Etter vi har kjørt den manuelt og programert den til å kjøre selv med PID regulering kan vi klart se at den kjører best selv. Selv om vi kun bruker P regulering så vil den fremdeles få mindre avvik enn det vi klarer. Om en ønsker en perfekt regulering så er det mulig å utregne verdier for dette.

Kapittel 6

Cruise-control (utført av hele gruppen)



Figur 6.1: Robot brukt til prosjekt Cruisecontrol

6.1 Introduksjon

6.1 Introduksjon

I dette delkapittelet skal vi ved hjelp av matlab lage en kode som fungerer som adaptiv cruise kontroll. Lego roboten skal kunne kjøre i en gitt hastighet, dempe farten i forhold til 'biler' / gjenstander foran seg, for å så øke farten til den angitte hvis hindringene forsvinner.

På dette prosjektet er Lego roboten påmontert en avstandsmåler (SonicSensor), for å kunne måle avstanden foran seg. Dette er vist i Figur 6.1

6.2 Forslag til løsning

Informasjonen roboten trenger for å kunne løse denne oppgaven er avstand til bilen foran og en gitt utgangsfart. Den gitte utgangsfarten tilsvarer den satte verdien i cruisekontroll på en bil, denne kan vi se som a i kode 6.1 linje 159.

Kode 6.1: Bestemmer fart a og reguleringskonstant b `Prosjekt06_CruiseControl_kjoring`

```
158      %Bestemmer utgangsfart%
159      a=40;
160      %Bestemmer reguleringskonstanten%
161      b=500;
```

Reguleringskonstanten b fra Kode 6.1 linje 161, er en konstant som ganges med det aktuelle avviket. Dette vil gi større/mindre regulering av roboten, avhengig av størrelsen på reguleringskonstanten. Vi skal se både a og b bli brukt senere i delkapittelet.

Det neste som må bestemmes er område hvor roboten må begynne å regulere farten sin.

Kode 6.2: Setter referansepunkt `Prosjekt06_CruiseControl_kjoring`

```
182      %Setter avstand til bil foran hvor roboten må begynne å ...
183          regulere fart%
183      Referansepunkt(1:k) = 0.15;
```

6.2 Forslag til løsning

Som vist i kode 6.2 setter vi reguleringspunktetlik 15cm eller 0.15m.

Neste steg er å regne ut avviket fra referansepunktet vårt , regne ut pådraget som blir sendt ut til motorene og kode hvor roboten må begynne å regulere fart.

Kode 6.3: Regner ut Avviket, setter limit avviket lik 0 hvis avstand er over 15cm og bestemmer motorpådrag Prosjekt06_CruiseControl kjoring

```
198      %Beregner avvik og setter reaksjonspunktet til 15cm%
199      Avvik(k) = Referansepunkt(k) - Avstand(k);
200
201      if Avstand(k) > 0.15
202          Avvik(k) = 0;
203      end
204
205      %Bestemmer pådrag til motorene%
206      PowerA(k) = a + (b * -Avvik(k));
207      PowerB(k) = a + (b * -Avvik(k));
```

For å beregne avviket roboten har fra å ligge 15 cm unna bilen foran, setter vi avviket lik differansen mellom referansepunktet vårt (definert tidligere) og avstanden målt av avstandsmåleren, vist i Kode 6.3 linje 199.

For at roboten ikke skal ta hensyn til bilen foran når avstanden til bilen foran er mer enn 15cm bruker vi en if-setning vist i Kode 6.3 linje 201-203. Her sier vi at hvis avstanden til bilen er mer en 15cm så vil avviket bli satt til 0.

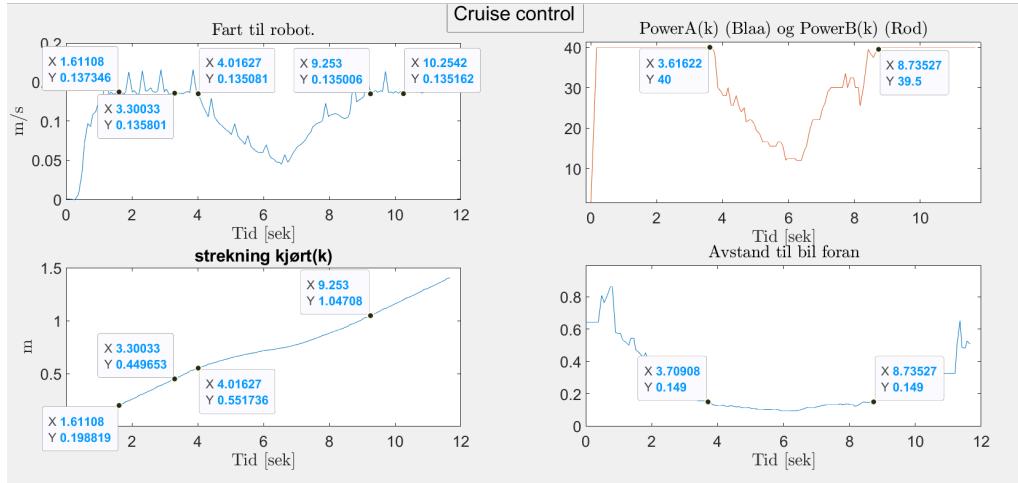
I linje 206-207 i Kode 6.3 bestemmer vi motorpådraget. Her kommer vi tilbake til konstantene a og b tidligere diskutert og definert i delkapitlet. Setter pådraget lik en konstant a, dette vil være farten vår ved fri veibane. Dette er fordi avstanden vil overstige 0.15m -> avviket blir deretter satt til 0 -> det bakerste ledet b*-Avvik(k) blir 0.

Derimot om avstanden går under 0.15m vil koden i Kode 6.3 regne ut et avvik og regulere farten. Her kommer reguleringskonstanten inn i bildet. Vist i linje 206-207, så vil avviket multipliseres med denne reguleringskonstanten. Man ser da at ved å justere på konstanten så vil roboten reagere mer eller mindre følsomt ovenfor avvik.

6.3 Resultat

6.3 Resultat

Ved å simulere en bil som kjører i konstant gitt fart, for å se regulere farten sin iforhold til en bil foran seg, for å så kjøre i samme konstant gitte fart, så får vi målingene vist i Figur 6.2.



Figur 6.2: Fart, strekning, motorpådrag

Fra fartsgrafen i Figur 6.2 kan man se at roboten sin fart begynner å regulere seg i samme område hvor avstanden til bilen foran seg blir mindre 0.15 m.

Kan også lese av at Roboten holder en konstant fart 0.136 m/s i området hvor den kjører med konstant fart uten hindring. Fra strekningsgrafen fra Figur 6.2 ser vi at farten mellom tiden 1.61s og 4.02 er:

$$\frac{\Delta y}{\Delta x} = \frac{0.552 - 0.199}{4.02 - 1.61} = 0.146 \text{ m/s} \quad (6.1)$$

Dette er tilnærmet lik den konstante farten 0.136. Avviket kommer høyst sannsynlig fra målestøy.

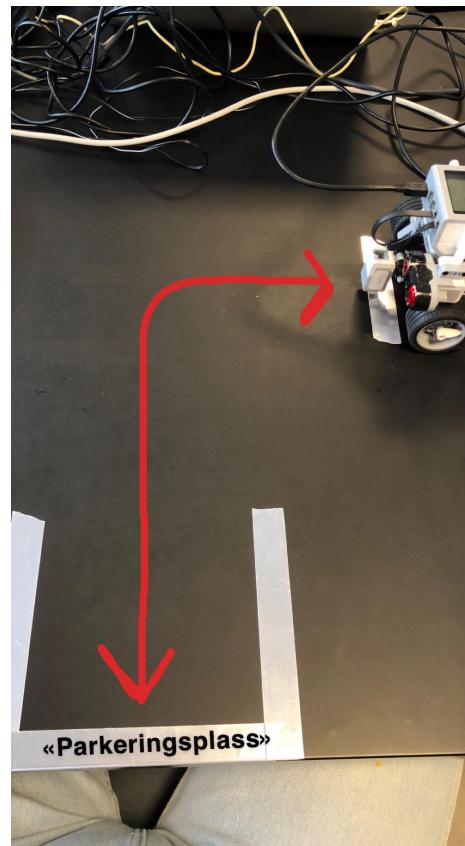
6.4 Sammendrag

6.4 Sammendrag

Ved hjelp av den adaptive cruise controll kan vi lene oss trygt tilbake i førerstolen når vi kjører bil. Når vi begynner å nærmer oss en bil så vil cruise kontrollen automatisk regulere farten på en rolig måte slik at den holder en fast avstand til bilen foran. Hvis det ikke er bil foran så vil roboten kjøre etter fartsgrensen. Ved å se avstanden som roboten har tilbakelagt kan vi regne ut farten til roboten og se at det stemmer med grafen 6.2.

Kapittel 7

Parkeringsassistent



Figur 7.1: Kjørebane for robot

7.1 Problemstilling

7.1 Problemstilling

Se for deg at du kjører inn i et trangt parkeringshus, og ikke er så flink til å rygge. På nymoderne biler kan en få parkeringsassistanse med så hjelper deg ut av denne situasjonen med å memorere rattbevegelsen og motorpådraget. Vi skal i dette delkapittel simulere denne parkeringsassistansen. Målet er at lego roboten skal parkere manuelt, for å så automatisk rygge i samme bane tilbake til start. Tenkt veibane hvor roboten er plassert i startpunktet er vist i 7.1.

7.2 Forslag til løsning

7.2 Forslag til løsning

Før roboten kan begynne å rygge trenger den en bane som den skal memorere. Siden vi kjører selv tar vi inspirasjon i koden som ble brukt i kapittel 4, manuell kjøring 4.

Kode 7.1: Kode for manuell kjøring i parkeringsassistanse.m

```
158     %Faktor for motorp?drag%
159     a=0.3;
160     b=0.3;
161
162     %P?drag sendt ut til motor A og B%
163     PowerA(k) = (a*JoyForover(k)) - (b*JoySideslengs(k));
164     PowerB(k) = (a*JoyForover(k)) + (b*JoySideslengs(k));
```

Likt som i manuell kjøring så bruker vi styrestikken. Denne er skalert opp til 100 så ved full pådrag så vil motoren gå i 100. Setter derfor konstant a og b til 0,3 for å senke farten på roboten.

For at roboten skal kunne rygge i en tilnærmet identisk veibane tilbake trenger den dataene for pådrag. Disse dataene lagres underveis i delen hvor bilen blir manuelt kjørt inn på parkeringsplassen. Koden brukt i dette delprosjektet kan man se i Kode 7.2.

7.2 Forslag til løsning

Kode 7.2: Kode for reverse kjøring i parkeringsassistanse.m

```
250    %Reverseringen starter når knappen blir trykket inn%
251    while JoyReverse == 1
252
253        %Flipper rekkefølgen og bytter fortegn for den lagrede ...
        %pådragsdataen%
254        ReverseA = fliplr(-PowerA);
255        ReverseB = fliplr(-PowerB);
256
257        %Flipper de lagrede verdiene for tidsskrittet%
258        ReverseTs = fliplr(Ts);
259
260        %Bruker en for-loop for å gå igjennom alle lagrede verdier%
261        for i = 1:k-2
262
263            %Setter pådraget lik de reversete verdiene%
264            motorA.Speed = ReverseA(i);
265            motorB.Speed = ReverseB(i);
266
267            %Leser av hvor langt hver motor har beveget seg%
268            VinkelPosReverseMotorA(i) = double(motorA.readRotation);
269            VinkelPosReverseMotorB(i) = double(motorB.readRotation);
270
271            %Viktig pause funksjon, denne gjør så robotten kjører ...
            %lengre nok%
272            pause(ReverseTs(i)-0.04)
273
274            %Regner ut hvor langt hjulet har kjørt%
275            VinkelPosDiffA(i) = (VinkelPosMotorA(k-1) + ...
            VinkelPosReverseMotorA(i));
276            VinkelPosDiffB(i) = (VinkelPosMotorB(k-1) + ...
            VinkelPosReverseMotorB(i));
277
278        end
279        %Sjekker om knappen fortsatt holdes inne, hvis ikke ...
        %avsluttes det
280        JoyReverse = button(joystick,12);
281    end
```

Etter at robotten blir kjørt manuelt inn på parkeringsplassen vil pådragsdataene for denne kjøringen være lagret. Det første vi gjør i koden er å flippe denne dataen, slik at siste måling vil bli den første målingene i variablene *ReverseA* og *ReverseB*. Funksjon som gjør dette for oss i Matlab heter *fipr*". Det samme må man gjøre med tidskrittet *ts*. Dette kan man se i Kode 7.2, linje 254-258.

For at alle de nye reverserte verdiene i *ReverseA* og *ReverseB* skal bli tatt i bruk av robotten bruker vi en forloop. Dette kan vi se i Kode 7.2, linje 261-265.

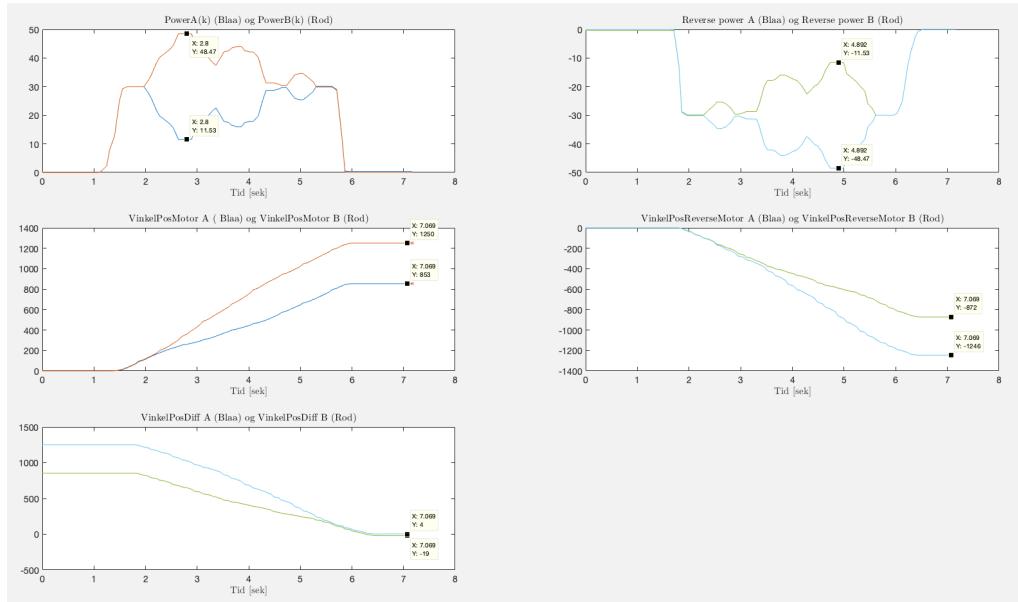
7.3 Resultat

I linje 272, kode 7.2, befinner en viktig funksjon seg. Her bruker vi en pause funksjon. Dette er fordi vi vil at motorene skal ha reversert pådrag i like lang tid som pådraget brukt i framoverkjøringen. For å få riktig lengde og på tidskrittet i denne pausen, må vi bruke *ReverseTs*.

Som man også kan observere i Kode 7.2, linje 272, er det også blitt subtrahert en konstant inne i pausevariablene. Denne konstanten kommer av selve tiden det tar å kjøre selve programmet. Hvis vi hadde vært foruten denne konstanten ville roboten reversert for langt, fordi den får litt lenger pådrag for hver runde i forløkka.

7.3 Resultat

Ved å kjøre roboten som vist i Figur 7.1 med bruk av parkeringsassistansen i revers delen, fikk vi resultatene vist i Figur 7.2.



Figur 7.2: Grafer parkeringsassistent

Ved å se på de to øverste grafene for motorpådraget i motor A og B (PowerA & PowerB) fra Figur 7.2, kan man observere at motorpådraget i revers-delen er

7.3 Resultat

identisk, bare speilvendt og med negative fortegn. Dette er god indikasjon på at programmet vårt fungerer som det skal, fordi målet med prosjektet var at lego-roboten skulle rygge tilbake til startposisjon automatisk.

De to midterste grafene vist Figur 7.2 viser vinkelposisjonen til motor A og motor B i den framover manuelle kjøringen og den automatiske reverse kjøringen. Den nederste grafen viser differansen mellom de to kjøredelene.

Optimalt sett skulle den nederste grafen *VinkelPosDiff* vært null. Det hadde i praksis betydd at hjulene har kjørt nøyaktig like langt fram som tilbake. Leser vi av fra sluttpunktet på grafen *VinkelPosDiff* ser vi at:

$$VinkelPosDiffA = 4 \quad (7.1)$$

$$VinkelPosDiffB = -19 \quad (7.2)$$

Siden radiusen i hjulet er 2.8 cm, så vil omkretsen av hjulet og lengden roboten kjører på en hjulrunde være:

$$omkrets = 2 * \pi * r = 0.176m \quad (7.3)$$

Motor A sitt avvik fra null var 4, mens motor B sitt avvik fra 0 var -19. Siden en hjulrunde er definert som 360, kan vi bruke forholdet mellom en runde, avviket fra null og omkretsen av hjulet til å regne ut hvor mange cm avvik hvert hjul har.

$$MotorAavvik = \frac{4}{360} * 17.6cm = 0.2cm \quad (7.4)$$

$$MotorBavvik = \frac{-19}{360} * 17.6cm = -0.93cm \quad (7.5)$$

Vi ser at motor A kjørte 0.2cm for langt, mens Motor B kjørte 0.93 cm for kort. Dette avviket er veldig lite og vi kan si oss fornøyd med resultatene.

7.4 Sammendrag

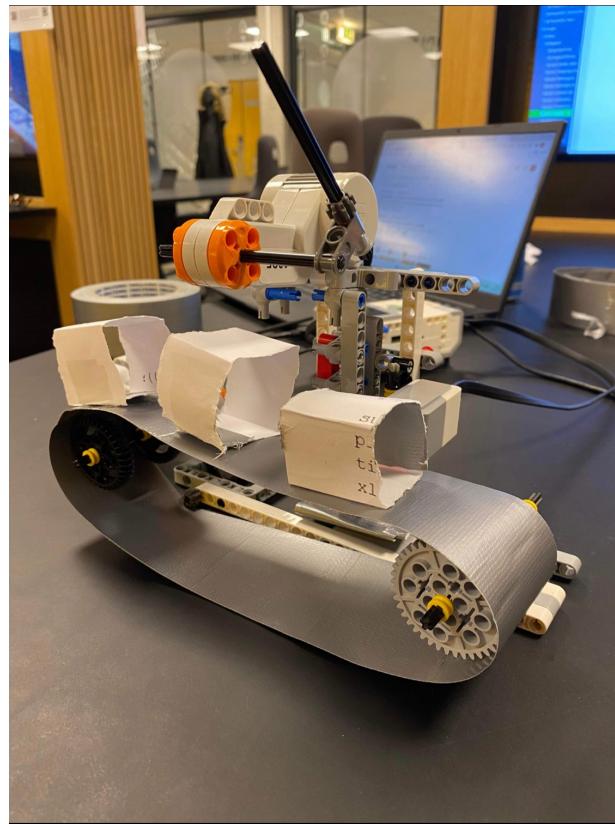
Grunner til dette avviket kan være mye forskjellig. Motorene forsyner ikke helt like pådrag framover som bakover. Det kan være noe ulik motstand i motorene når roboten svinger på framturen kontra tilbaketur. Samt pausefunksjonen i Kode 6.3, tidligere diskutert, kan være suboptimal.

7.4 Sammendrag

I dette delkapittelet har vi laget en funksjon for å gjøre det lettere å rygge ut av en vanskelig situasjon. Ved å snu pådraget og dreieretning på motorene ved hjelp av *fliplr* funksjonen så skal vi i teorien få en perfekt reverse funksjon. Ettersom motorene ikke er ideelle får vi litt avvik og ved lengre simuleringer vil den slite mer med å returnere til startpunktet.

Kapittel 8

Samlebånd



Figur 8.1: Bilde av samlebåndet vårt.

8.1 Problemstilling

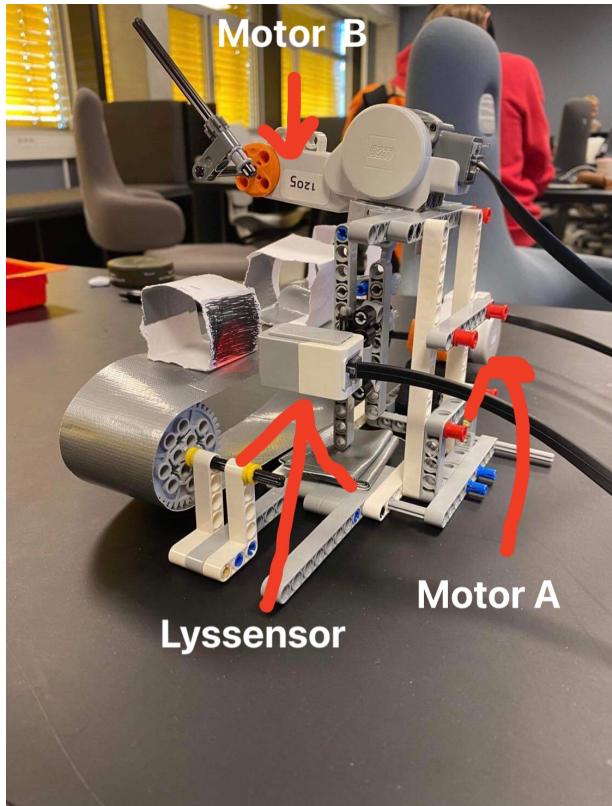
8.1 Problemstilling

I dette delkapittelet skal vi gå nærmere inn på sorteringsautomatikk. Vi skal ved hjelp av et samlebånd, en lyssensor og to motorer lage et system som kan sortere ut ønskede bokser. Målet med prosjektet er at systemet skal dytte av hvite bokser på langsiden av samlebåndet, men la svarte bokser slippe igjennom.

8.2 Forslag til løsning

Til å starte med må selve rammeverket bygges. Stabilitet og styrke er viktige faktorer i selve byggeprosessen. Det ferdige systemet er avbildet på forsiden av delkapittelet i Figur 8.1, samt et bilde i Figur 8.2, som viser mer av konstruksjonen bak og de brukte delene.

8.2 Forslag til løsning



Figur 8.2: Bilde av samlebåndet vårt.

Det er to motorer som driver det fysiske i prosessen, vist i Figur 8.2. Motor A driver selve samlebåndet, mens motor B driver dytte prosessen". Vi har valgt å basere selve sorteringprosessen vår på lysmålinger gjort av lyssensoren (0.1.2 i første delkapittel).

Nå over til koden som utfører de essensielle beregningene i sorteringssystemet. De er vist i Kode 8.1

8.2 Forslag til løsning

Kode 8.1: Kode samlebånd Prosjekt06_CruiseControl kjoring

```
166     PowerB(k) = 0;  
167     PowerA(k) = 0;  
168  
169     if k==1  
170         Lys(k)=0;  
171         Tid(k)=0;  
172     end  
173  
174     if Lys(k) < 25  
175         PowerA(k) = -20;  
176         motorA.Speed = PowerA(k);  
177     else  
178         PowerA(k) = 0;  
179         motorA.Speed = PowerA(k);  
180         pause(2)  
181         for c = 1:109  
182             PowerB(k) = -50;  
183             motorB.Speed = PowerB(k);  
184         end  
185         PowerB(k) = 0;  
186         motorB.Speed = PowerB(k);  
187         pause(2)  
188     end
```

Hvis ingen bokser, eller en svart boks står foran lyssensoren, vil det reflekterte lyset Lys(k) være under 25. Fra Kode 8.1, linje 174-176, ser man at motor A vil deretter få et pådrag som drar samlebåndet.

Hvis en hvit boks blir rullet foran lyssensoren vil det reflekterte lyset Lys(k) bli større enn 25 og else-setningen (Kode 8.1, linje 177-180) vil slå inn. Her settes pådraget til motor A lik 0 og den hvite boksen står nå stille, i posisjon til å bli dyttet av båndet.

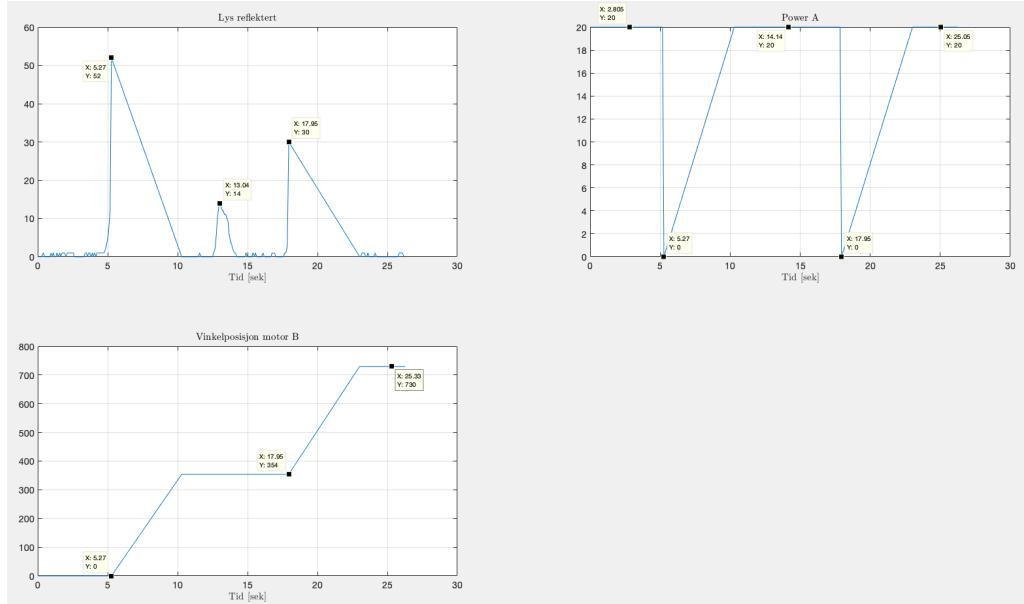
Deretter kommer en for-setning (Kode 8.1, linje 181-183) hvor pådraget til motor B settes til -50. Da vil dreiearmen påmontert til motor B bevege seg rundt og dytte den hvite boksen av samlebåndet. På denne måten kan vi sortere hvite og svarte bokser på et samlebånd.

I dette prosjektet skal sorteringsmaskinen dytte ut to hvit bokser og la en svart boks være. De kommer rekkefølgen: hvit - svart - hvit.

8.3 Resultat

8.3 Resultat

Ved å kjøre sorteringsystemet, får vi grafene vist i Figur 8.3.



Figur 8.3: Reflektert lys fra lyssensor, pådrag i motor A og vinkel posisjon motor B.

Det første kritiske punktet vi kan se på er ved $t = 5.27$ s. Fra lys grafen i Figur 8.3 ser vi at ved dette punktet får grafen en spike", på en verdi lik 52. Dette betyr i praksis at en hvit boks har inntatt posisjon foran lyssensoren.

Med den tidligere gjennomgåtte koden i bakhodet, vet vi at motor A som dreier selve samlebåndet skal stoppe. Derimot skal motor B få et pådrag, begynne å dreie, for å dytte av den hvite boksen.

Dette kan vi verifisere ved å se på grafene i Figur 8.3. Vi observerer at ved $t = 5.27$ s så går pådraget i motor A til 0. Observere også at vinkel posisjon til motor B ved $t = 5.27$ s starter å endre seg.

Neste kritiske punkt er ved $t=13.04$ s. Ved å observere lysgrafen fra Figur 8.3, ser vi at den får en betraktelig mindre spike. Det målte lys(k) i denne tiden ble 14. Dette betyr at en svart boks har blitt rullet inn foran lyssensoren.

8.4 Sammendrag

Som tidligere gjennomgått vil ikke if-setningen bli aktivert, siden lys-verdien ikke overstiger 25. Samlebåndet vil derfor fortsette å rulle og den svarte boksen fortsetter sin ferd forbi dreiaarmen.

Neste kritiske punkt er ved $t = 17.95\text{s}$. Hvis vi ser på grafen for lys, ser vi at det målte lyset er på 30. Da blir "ifsetningen igjen bli aktivert, samlebandet stopper og motor B starter. Dette resulterer i at den siste hvite boksen blir dyttet av samlebandet og vi har sortert de tre boksene etter farge.

8.4 Sammendrag

Vi har i dette delkapittelet gått nærmere innpå sorteringsautomatikk. Vi har ved hjelp av egenskrevet kode i Matlab og et egetlagd samlebånd sortert to hvite og en svart boks. Vi har gått igjennom hvordan koden fungerer og hvordan grafene som viser motorpådrag og lysverdi er koblet til det som skjer i praksis.

Kapittel 9

Konklusjon

Etter 16 uker med Lego prosjekt så har vi til slutt utført 8 øvinger, hvor 4 av de var obligatoriske og 4 frivillige. For å kode de forskjellige prosjektene har vi brukt Matlab, som var helt nytt for alle på gruppen. I starten så gav dette store utfordringer og vi brukte lang tid på å settes oss skikkelig inn i bruken av matlab. Etter mye prøving og feiling, så fikk vi til slutt utført de forskjellige øvingene på en god måte.

I tillegg til å lære oss matlab så skulle vi også skrive rapport fortløpene i overleaf. Her også ble det mye prøving, men vi innså omsider at for rapportskriving er overleaf det klart beste programmet å ta i bruk.

De obligatoriske prosjekten som vi skulle gjennomføre var integrasjon, filtrering, derivering og manuell kjøring av roboten. Med lite kunnskap om integrasjon og derivasjon så ble dette også en utfordring. Når vi omsider ble ferdig med disse prosjektene, fikk vi en større forståelse i hvor sentralt integrasjon og derivasjon er. Det var også motiverende å se koblingen mellom matematikken og det praktiske. Filtrering var en fin oppgave og fikk frem på en god måte hvorfor filtrering av signal er viktig i enkelte oppraskjoner, som f.eks plasser med mye målestøy. Til slutt så var det manuell kjøring. På dette tidspunktet hadde vi blitt bedre kjent med Matlab, samt allerede lagd viktige funksjoner som vi fikk bruk for, så det gikk forhåndsvist lett.

Konklusjon

Etter vi har fullført den obligatoriske delen kunne vi endelig starte på den frivillige. Første prosjektet var å se hvor bra vi kunne regulere roboten til å kjøre selv. Ved hjelp av en *PID* regulator fikk vi roboten til å kjøre gjennom banen uten noen problem, med bedre resultater enn oss andre.

Når roboten kjørte knirkefritt så beveget vi oss over til prosjekt Cruise Control. Her var målet å få roboten til å kjøre i en konstant hastighet, helt til en 'hindring' / bil ble plassert foran. Roboten regulerer hastighet slik at den har lik avstand til bilen foran hele tiden.

Videre til neste prosjekt så har vi laget en kode som skal hjelpe roboten til å rygge ut av vanskelige situasjoner. Ved hjelp av å memorere motorpådraget til hver motor kan vi på en enkel måte 'flippe' disse verdiene. Resultatet ble at den rygget tilnærmet likt som den kjørte inn.

Det siste prosjektet ble et samlebånd som sorterer esker etter fargen. Her ønsker vi at esker som er svarte skal gå gjennom uten problem mens hvite esker skal sorteres bort. I dette prosjektet ble det bygget en stor kreativ konstruksjon og ved begrenset utstyr fikk vi laget et samlebånd av gaffateip som roteres ved hjelp av lego motorene. Ved hjelp av en andre motor som roterer 360 grader å dytter av de hvite boksene, ble prosjektet veldig suksessfullt og gav stor mestringsfølelse.

Nå som vi er ferdig med lego prosjektet skulle vi ønsket at vi hadde bedre tid på prosjektet. Når vi ser tilbake på hvordan vi har løst de første oppgavene Det har vært et artig prosjekt som vi har lært masse om programmering, rapportskriving og problemløsning.

Konkluderer med at Sondre er gay. Tormod er transe, eller bare en veldig liten tiss.

Bibliografi

- [1] The LEGO Group. Lego Education Equipment. <https://www.lego.com/nb-no/search?q=mindstorm>. [Online; accessed 15-Mars-2022].
- [2] T. Drengstig. Prosjektdel av ELE130 Anvendt matematikk og fysikk i robotprogrammering. Lego Mindstorms og MATLAB/Python i skjønn forening. Technical report, Universitet i Stavanger, 2022. Utdelt materiale.

Vedlegg A

Timelister

Tabell A.1: Timelister for gruppe 22XX i faget ELE130.

Navn/uke	Sondre	Aleksa	Tormod	Truls
4	4	9	10	5
5	6	6	6	5
6	12	12	6	5
7	6		6	5
8	16		16	5
9	24		24	5
10	40		40	5
11	50		50	5
12	50		50	5
Sum				

Vedlegg B

Programlisting prosjekt 01

B.1 Prosjekt01_Filtrering.m

Kode B.1: Kode for Numerisk Integrasjon Prosjekt01_NumeriskIntegrasjon.m

```
147 % ...
148 %           CONDITIONS, CALCULATIONS AND SET MOTOR POWER
149 % Gjoer matematiske beregninger og motorkraftberegninger
150 % hvis motor er tilkoplet.
151 % Kaller IKKE paa en funksjon slik som i Python
152 %if numel(Tid) == 1
153 %SumDiffLys(1) = 0 ;
154 % DiffLys(1)=0;
155
156
157 nullflow = Lys(1);
158
159 if k==1
160     Flow(1) = 0;
161     volum(1) = 20;
162 else
163     Ts(k-1) = Tid(k) - Tid(k-1) ;
164     Flow(k) = Lys(k) - nullflow(1);
165     %volum(k) = volum(k-1) + (Ts(k-1) * Flow(k-1));
166     volum(k) = EulerForward(volum(k-1), Flow(k-1), Ts(k-1))
167 end
```

Vedlegg C

Programlisting prosjekt 02

C.1 Prosjekt02_filtrering.m

Kode C.1: Kode for FIR og IIR Prosjekt02_Filtrering.m

```
142 % ...
143 %           CONDITIONS, CALCULATIONS AND SET MOTOR POWER
144 % Gjoer matematiske beregninger og motorkraftberegninger
145 % hvis motor er tilkoplet.
146 % Kaller IKKE paa en funksjon slik som i Python
147
148 %FIR Filtrering%
149 %-----
150 Temp_stoy(k) = Lys(k) + 5*rand;
151 Temp(k) = Lys(k);
152 m = 10;
153 stoy(k) = Temp_stoy(k) - Temp(k);
154
155 % FIR med m = 3
156 if k<m
157     m=k;
158 end
159
160 Temp_FIR(k) = (1/m)*sum(Temp_stoy(k-m+1:k));
161
162 %-----
163 % IIR-Filtrering
164 %-----
```

C.1 Prosjekt02_filtrering.m

```
165     a= 0.2;
166
167     if k==1
168         Temp_IIR(k) = Temp_stoy(k);
169     else
170         %IIR filtrering med alfa = 0.01
171         Temp_IIR(k) = IIR_filter(Temp_IIR(k-1), Temp_stoy(k), a)
172
173 end
```

Kode C.2: Kode for FIR funksjon Prosjekt02_Filtrering.m

```
1 function [FilteredValue] = FIR_filter(Measurments, NoOfMeas)
2     FilteredValue = (1/NoOfMease)*sum(Measurments(k-m+1:k));
3 end
```

Kode C.3: Kode for IIR funksjon Prosjekt02_Filtrering.m

```
1 function [FilteredValue] = IIR_filter(OldFilteredValue, ...
2     Measurments, Para)
3     FilteredValue = Para * Measurments(k) + ...
4         (1-Para)*OldFilteredValue(k-1);;
5 end
```

Vedlegg D

Programlisting prosjekt 03

D.1 Prosjekt03_Numerisk_Derivasjon.m

Kode D.1: Kode for Numerisk Derivasjon Prosjekt03_Numerisk_derivasjon.m

```
139 % CONDITIONS, CALCULATIONS AND SET MOTOR POWER
140 % Gjoer matematiske beregninger og motorkraftberegninger
141 % hvis motor er tilkoplet.
142 % Kaller IKKE paa en funksjon slik som i Python
143
144 %Bestemmer para, som brukes i filtrering
145 para = 0.2;
146
147 %Setter initialbetingeler rundt problemområdet k=1 for ...
148 % feilfri kjøring
149 %av koden
150 if k == 1
151     Tid(k)=0;
152     Fart(k)=0;
153     Avstand(k)=Lys(k);
154     Avstand_filtrert(k) = Lys(k);
155     Aks(k) = 0;
156
157 else
158     %Beregner tidsskrittet
159     Ts(k) = (Tid(k) - Tid(k-1));
160
```

D.1 Projekt03_Numerisk_Derivasjon.m

```
161      %Setter Avstand lik lyset som blir reflektert fra ...
162      %lyssensoren
163      Avstand(k) = Lys(k);
164
165      %Filtrerer avstanden
166      Avstand_filtrert(k) = para*Avstand(k) + ...
167          ((1-para)*Avstand_filtrert(k-1));
168
169      %Ufilt_fart(k-1)= (Avstand(k)-Avstand(k-1))/Ts(k);
170
171      %Kaller på derivasjon funksjon og numerisk deriverer ...
172      %avstanden
173      Fart(k-1) = derivasjon(Ts(k), Avstand_filtrert(k-1:k)) ;
174
175
176      %Aks(k) = derivasjon(Ts(k), Fart(k-1:k));
177
178      %Fart_filtrert(k)= para*Fart(k) + ...
179          ((1-para)*Fart_filtrert(k-1));
180
181  end
```

Vedlegg E

Programlisting prosjekt 04

E.1 Prosjekt04_Manuell_Kjoring.m

Kode E.1: Kode for Manuell kjøring Prosjekt04_Kjoring.m

```
154 % CONDITIONS, CALCULATIONS AND SET MOTOR POWER
155 % Gjør matematiske beregninger og motorkraftberegninger
156 % hvis motor er tilkoplet.
157 % Kaller IKKE paa en funksjon slik som i Python
158
159 %Faktor for motorp?drag%
160 a=0.3;
161 b=0.3;
162
163 %P?drag sendt ut til motor A og B%
164 PowerA(k) = (a*JoyForover(k)) + (b*JoySideslengs(k));
165 PowerB(k) = (a*JoyForover(k)) - (b*JoySideslengs(k));
166
167 if online
168     % Setter powerdata mot EV3
169     % (slett de motorene du ikke bruker)
170     motorA.Speed = PowerA(k);
171     motorB.Speed = PowerB(k);
172
173     start(motorA)
174     start(motorB)
175 end
176
177 % Hvis lyssensor treffer hvit ark stopper programmet%
```

E.1 Prosjekt04_Manuell_Kjoring.m

```
178     if Lys(k) > 52
179         JoyMainSwitch = 1;
180     end
181
182     %Initsial betingelser for ? starte programet%
183     if k == 1
184         Gjennomsnitt(k) = 0;
185         Standardavvik(k) = 0;
186         Avvik(k)=0;
187         IAE(k)=0;
188         Tid(k)=0;
189         MAE(k)=0;
190         Abs_avvik(k)=0;
191         TVa(k)=0;
192         TVb(k)=0;
193
194     else
195         %Setter referansepunkt likt f?rste lysm?ling%
196         Referansepunkt(1:k) = Lys(1);
197
198         %Beregner avvik og absoluttverdien til avvik%
199         Avvik(k) = Referansepunkt(k) - Lys(k);
200         Abs_avvik(k)=abs(Avvik(k));
201
202
203         %Beregner gjennomsnitt og standardavvik av lys m?linger%
204         Middelverdi(k) = mean(Lys(k));
205         Standardavvik(k) = std(Lys(k));
206
207         %Beregner gjennomsnitt og standardavvik av avviket%
208         Gjennomsnitt(k) = mean(Avvik(k));
209         Standardavvik(k) = std(Avvik(1:k));
210
211         %Beregning av Mean Absolute Error (MAE)%
212         MAE(k) = (Abs_avvik(k) + sum(Abs_avvik(1:k-1))) / k;
213
214         %Beregning av Integral of Absolute Error (IAE)%
215         IAE(k) = IAE(k-1) + (Abs_avvik(k) * (Tid(k)-Tid(k-1)));
216
217         %Motorp?drag
218         TVa(k) = sum(abs(PowerA(k)-PowerA(k-1)) + TVa(k-1));
219         TVb(k) = sum(abs(PowerB(k)-PowerB(k-1)) + TVb(k-1));
220     end
```

Vedlegg F

Programlisting prosjekt 05

F.1 Prosjekt05_selvKjoring.m

Kode F.1: Kode for Automatisk kjøring Prosjekt05_selvKjoring.m

```
154 % CONDITIONS, CALCULATIONS AND SET MOTOR POWER
155 % Gjoer matematiske beregninger og motorkraftberegninger
156 % hvis motor er tilkoplet.
157 % Kaller IKKE paa en funksjon slik som i Python
158
159
160 % Hvis lysensor treffer hvit ark stopper programmet%
161 if Lys(k) > 50
162 JoyMainSwitch = 1;
163 end
164
165
166 %Initsial betingelser for ? starte programmet%
167 if k == 1
168     Gjennomsnitt(k) = 0;
169     Standardavvik(k) = 0;
170     Avvik(k)=0;
171     IAE(k)=0;
172     MAE(k)=0;
173     Abs_avvik(k)=0;
174     TVa(k)=0;
175     TVb(k)=0;
176     Referansepunkt(k) = Lys(1);
177     PowerA(k)=0;
```

F.1 Prosjekt05 _selvKjoring.m

```
178      PowerB(k)=0;
179      Ts(k) = 0;
180      Avvik_konstant(k)=0;
181      I(k)=0;
182      Avvik_filtrert(k)=0;
183
184  else
185      %Setter referansepunkt likt f?rste lysm?ling%
186      Referansepunkt(1:k) = Lys(1);
187
188      %Tidsskritt beregning%
189      Ts(k-1) = (Tid(k) - Tid(k-1));
190
191      %Beregner avvik og absoluttverdien til avvik%
192      Avvik(k) = Referansepunkt(k) - Lys(k);
193      Abs_avvik(k)=abs(Avvik(k));
194
195      %Beregner gjennomsnitt og standardavvik av lys m?linger%
196      Middelverdi(k) = mean(Lys(k));
197      Standardavvik(k) = std(Lys(k));
198
199      %Beregner gjennomsnitt og standardavvik av avviket%
200      Gjennomsnitt(k) = mean(Avvik(k));
201      Standardavvik(k) = std(Avvik(1:k));
202
203      %Beregning av Mean Absolute Error (MAE)%
204      MAE(k) = (sum(Abs_avvik(1:k))) / k;
205
206      %Beregning av Integral of Absolute Error (IAE)%
207      IAE(k) = IAE(k-1) + Abs_avvik(k) * Ts(k-1);
208
209
210      %Faktor for motorp?drag%
211      a=15;
212      Kp=3;
213      Ki=2;
214      Kd=0;
215      para = 0.3;
216
217      %P-leddet, PID REGULERING%
218      P(k) = (Kp * Avvik(k));
219      if abs(P(k)) > 30
220          P(k) = P(k-1);
221      end
222
223      %I-leddet, PID REGULERING%
224      Avvik_konstant(k) = Avvik(k-1) * Ki;
225      I(k) = EulerForward(I(k-1), Avvik_konstant(k), Ts(k-1));
226      if abs(I(k)) > 25
227          I(k) = I(k-1);
```

F.1 Prosjekt05_selvKjoring.m

```
228     end
229
230     %D-leddet, PID REGULERING%
231     Avvik_filtrert(k) = (para*Avvik(k) + ...
232                           ((1-para)*Avvik_filtrert(k-1)))*Kd;
233     D(k) = derivasjon(Ts(k-1), Avvik_filtrert(k-1:k));
234
235     %P?drag sendt ut til motor A og B%
236     PowerA(k) = a - P(k) - I(k) - D(k);
237     PowerB(k) = a + P(k) + I(k) + D(k);
238
239     %Motorp?drag
240     TVa(k) = sum(abs(PowerA(k)-PowerA(k-1)) + TVa(k-1));
241     TVb(k) = sum(abs(PowerB(k)-PowerB(k-1)) + TVb(k-1));
242
243 end
```

Vedlegg G

Programlisting prosjekt 06

G.1 Prosjekt06_CruiseControl.m

Kode G.1: Kode for Cruise Control Prosjekt06_CruiseControl.m

```
152 % CONDITIONS, CALCULATIONS AND SET MOTOR POWER
153 % Gjoer matematiske beregninger og motorkraftberegninger
154 % hvis motor er tilkoplet.
155 % Kaller IKKE paa en funksjon slik som i Python
156
157
158 %Bestemmer utgangsfart%
159 a=40;
160 %Bestemmer reguleringskonstanten%
161 b=500;
162
163 %Initsial betingelser for ? starte programet%
164 if k == 1
165     %Gjennomsnitt(k) = 0;
166     %Standardavvik(k) = 0;
167     Tid(k)=0;
168     TVa(k)=0;
169     TVb(k)=0;
170     PowerA(k)=0;
171     PowerB(k)=0;
172     VinkelPosMotorA(k) = 0;
173     DeltaVinkel(k) = 0;
174     Ts(k) = 0;
175     Rpm(k)=0;
```

G.1 Projekt06 _CruiseControl.m

```
176      Fart(k)=0;
177      strekning(k)=0;
178  else
179      %Beregner Ts%
180      Ts(k) = Tid(k)-Tid(k-1);
181
182      %Setter avstand til bil foran hvor roboten må begynne å ...
183      %regulere fart%
184      Referansepunkt(1:k) = 0.15;
185
186      %Regner ut endringen i Vinkelposisjon for motor A, for ...
187      %videre
188      DeltaVinkel(k-1)=VinkelPosMotorA(k)-VinkelPosMotorA(k-1);
189
190      %Beregner strekning kjørt i et tidsskritt
191      strekning(k) = (DeltaVinkel(k-1)/360)*0.175 ;
192
193      %Beregner farten roboten kjører%
194      Fart(k) = -(strekning(k)-strekning(k-1)/Ts(k-1));
195
196      %Beregner summen av strekningen
197
198      Sum_strekning(k) = sum(strekning(1:k));
199
200      %Beregner avvik og setter reaksjonspunktet til 15cm%
201      Avvik(k) = Referansepunkt(k) - Avstand(k);
202
203      if Avstand(k) > 0.15
204          Avvik(k) = 0;
205      end
206
207      %Bestemmer pådrag til motorene%
208      PowerA(k) = a+(b * -Avvik(k));
209      PowerB(k) = a+(b * -Avvik(k));
210
211      %Motorp?drag
212      %TVa(k) = sum(abs(PowerA(k)-PowerA(k-1)) + TVa(k-1));
213      %TVb(k) = sum(abs(PowerB(k)-PowerB(k-1)) + TVb(k-1));
214  end
```

Vedlegg H

Programlisting prosjekt 07

H.1 Prosjekt07_Parkeringsassistanse.m

Kode H.1: Kode for Parkeringsassistanse, kjøring fremover
Prosjekt07_parkeringsassistanse.m

```
153 % CONDITIONS, CALCULATIONS AND SET MOTOR POWER
154 % Gjør matematiske beregninger og motorkraftberegninger
155 % hvis motor er tilkoplet.
156 % Kaller IKKE paa en funksjon slik som i Python
157
158 %Faktor for motorp?drag%
159 a=0.3;
160 b=0.3;
161
162 %P?drag sendt ut til motor A og B%
163 PowerA(k) = (a*JoyForover(k)) - (b*JoySideslengs(k));
164 PowerB(k) = (a*JoyForover(k)) + (b*JoySideslengs(k));
165
166 if online
167     % Setter powerdata mot EV3
168     % (slett de motorene du ikke bruker)
169     motorA.Speed = PowerA(k);
170     motorB.Speed = PowerB(k);
171
172     start(motorA)
173     start(motorB)
174 end
175
```

H.1 Prosjekt07 _ Parkeringsassistanse.m

```
176 % Hvis lyssensor treffer hvit ark stopper programmet%
177 if Lys(k) > 9000
178 JoyMainSwitch = 1;
179 end
180
181 %Initsial betingelser for ? starte programet%
182 if k == 1
183     Gjennomsnitt(k) = 0;
184     Standardavvik(k) = 0;
185     Avvik(k)=0;
186     IAE(k)=0;
187     Tid(k)=0;
188     MAE(k)=0;
189     Abs_avvik(k)=0;
190     TVa(k)=0;
191     TVb(k)=0;
192     Ts(k)=0;
193
194 else
195     %Setter referansepunkt likt f?rste lysm?ling%
196     Referansepunkt(1:k) = Lys(1);
197
198     %Lager et tidsskritt
199     Ts(k-1) = (Tid(k) - Tid(k-1));
200
201     %Beregner avvik og absoluttverdien til avvik%
202     Avvik(k) = Referansepunkt(k) - Lys(k);
203     Abs_avvik(k)=abs(Avvik(k));
204
205
206     %Beregner gjennomsnitt og standardavvik av lys m?linger%
207     Middelverdi(k) = mean(Lys(k));
208     Standardavvik(k) = std(Lys(k));
209
210     %Beregner gjennomsnitt og standardavvik av avviket%
211     Gjennomsnitt(k) = mean(Avvik(k));
212     Standardavvik(k) = std(Avvik(1:k));
213
214     %Beregning av Mean Absolute Error (MAE)%
215     MAE(k) = (Abs_avvik(k) + sum(Abs_avvik(1:k-1))) / k;
216
217     %Beregning av Integral of Absolute Error (IAE)%
218     IAE(k) = IAE(k-1) + (Abs_avvik(k) * (Tid(k)-Tid(k-1)));
219
220     %Motorp?drag
221     TVa(k) = sum(abs(PowerA(k)-PowerA(k-1)) + TVa(k-1));
222     TVb(k) = sum(abs(PowerB(k)-PowerB(k-1)) + TVb(k-1));
223
224
225
```

H.1 Projekt07_Parkeringsassistanse.m

```
226     end
227         % tegn naa (viktig kommando)
228
229     drawnow
230     %-----
231
232     % For aa flytte PLOT DATA etter while-lokken, er det ...
233     % enklest aa
234     % flytte de neste 5 linjene (til og med "end") over PLOT DATA.
235     %
236     % Oppdaterer tellevariabel
237     k=k+1;
238
239     if JoyReverse == 1
240         JoyMainSwitch = 1;
241         motorA = motor(mylego, 'A');
242         motorA.resetRotation;
243         motorB = motor(mylego, 'B');
244         motorB.resetRotation;
245     end
```

Kode H.2: Kode for Parkeringsassistanse, kjøring bakover / reverse
Prosjekt07_parkeringsassistanse.m

```
250     %Reverseringen starter når knappen blir trykket inn%
251     while JoyReverse == 1
252
253         %Flipper rekkefølgen og bytter fortegn for den lagrede ...
254         %pådragsdataen%
255         ReverseA = fliplr(-PowerA);
256         ReverseB = fliplr(-PowerB);
257
258         %Flipper de lagrede verdiene for tidsskrittet%
259         ReverseTs = fliplr(Ts);
260
261         %Bruker en for-loop for å gå igjennom alle lagrede verdier%
262         for i = 1:k-2
263
264             %Setter pådraget lik de reversete verdiene%
265             motorA.Speed = ReverseA(i);
266             motorB.Speed = ReverseB(i);
267
268             %Leser av hvor langt hver motor har beveget seg%
269             VinkelPosReverseMotorA(i) = double(motorA.readRotation);
270             VinkelPosReverseMotorB(i) = double(motorB.readRotation);
271
272             %Viktig pause funksjon, denne gjør så robotten kjører ...
273             %lengre nok%
```

H.1 Prosjekt07 _ Parkeringsassistanse.m

```
272     pause(ReverseTs(i)-0.04)
273
274     %Regner ut hvor langt hjulet har kjørt%
275     VinkelPosDiffA(i) = (VinkelPosMotorA(k-1) + ...
276         VinkelPosReverseMotorA(i));
277     VinkelPosDiffB(i) = (VinkelPosMotorB(k-1) + ...
278         VinkelPosReverseMotorB(i));
279
280     JoyReverse = button(joystick,12);
281 end
```

Vedlegg I

Programlisting prosjekt 08

I.1 Prosjekt08_Samlebaand.m

Kode I.1: Kode for Samlebånd Prosjekt08_Samleband.m

```
153
154     end
155 %-----
156
157
158
159
160 % ...
161 %           ++++++ CONDITIONS, CALCULATIONS AND SET MOTOR POWER
162 % Gjoer matematiske beregninger og motorkraftberegninger
163 % hvis motor er tilkoplet.
164 % Kaller IKKE paa en funksjon slik som i Python
165
166 PowerB(k) = 0;
167 PowerA(k) = 0;
168
169 if k==1
170     Lys(k)=0;
171     Tid(k)=0;
172 end
173
174 if Lys(k) < 25
175     PowerA(k) = -20;
```

I.1 Prosjeekt08_Samlebaand.m

```
176         motorA.Speed = PowerA(k);
177     else
178         PowerA(k) = 0;
179         motorA.Speed = PowerA(k);
180         pause(2)
181         for c = 1:109
182             PowerB(k) = -50;
183             motorB.Speed = PowerB(k);
184         end
185         PowerB(k) = 0;
186         motorB.Speed = PowerB(k);
187         pause(2)
188     end
189
190
191 if online
192     % Setter powerdata mot EV3
193     % (slett de motorene du ikke bruker)
194     motorA.Speed = PowerA(k);
195     motorB.Speed = PowerB(k);
196     %motorC.Speed = PowerC(k);
197     %motorD.Speed = PowerD(k);
198
199     start(motorA)
200     start(motorB)
201     %start(motorC)
202     %start(motorD)
203 end
204 %-----
205
206
207 % For aa flytte PLOT DATA etter while-lokken, er det ...
208 % enklest aa
209 % flytte de neste 5 linjene (til og med "end") over PLOT DATA.
210 %
211 % Oppdaterer tellevariabel
212 k=k+1;
213 end
```