

11/20/2014

ING100

PROSJEKT HØSTEN 2014

Matlab og LEGO NXT i anvendt matematikk og fysikk

Gruppenummer: **1401**

Medlemmer:	Helge Johannessen Bjorland	228290
	Espen Ro Eliassen	795934
	Daniel Lovik	225039
	Anders Svalestad	221627



1 INNHOLD

1	Innhold	1
2	Sammendrag	6
3	Prosjektgjennomføringen	7
3.1	Mål	7
3.2	Arbeidsform	7
3.3	Metodikk	7
3.3.1	Utstyr	7
3.3.2	Programmering	7
4	Hovedprogram - Main	9
4.1	Numerisk Integrasjon	9
4.2	Numerisk Derivasjon	10
4.3	Filtrering	12
4.4	Manuell kjøring	13
4.5	Automatisk kjøring	14
4.6	Tillegg til obligatorisk del i hovedprogram	17
4.7	Konklusjon	18
5	Kreative program	19
5.1	Grafisk Brukergrensesnitt (GUI)	19
5.2	Spill	20
5.2.1	Cannon Game	20
5.2.2	Reaction1	23
5.2.3	Reaction2	24
5.3	Musikk	26
5.3.1	KeyboardPiano	26
5.3.2	Gray Music	28
5.3.3	Joy Music	30
5.4	Robofun	31
5.4.1	Clap Drive	31

5.4.2	Ultra Drive	32
5.4.3	Ultra Wall.....	34
5.5	Matematiske funksjoner og plotting.....	36
5.5.1	Plot Timer	36
5.5.2	Math Show	37
5.5.3	Num Joy	39
6	Konklusjon	41
6.1	Manuell Kjøring	41
6.2	Automatisk Kjøring	41
6.3	Tillegg til obligatorisk del.....	41
6.4	Grafisk Brukergrensesnitt (GUI)	41
6.5	Cannon Game	41
6.6	Reaction1.....	41
6.7	Reaction2.....	41
6.8	KeyboardPiano	42
6.9	Gray Music.....	42
6.10	Joy Music	42
6.11	Clap Drive	42
6.12	Ultra Drive	42
6.13	Ultra Wall.....	42
6.14	Plot Timer	42
6.15	Math Show	43
6.16	NumJoy.....	43
7	Referanser.....	44
8	Appendiks A - Illustrasjoner.....	45
8.1	Obligatorisk del.....	45
8.2	Cannon Game	47
8.3	mathShow.....	47
8.4	NumJoy.....	49
8.5	KeyboardPiano	50
8.6	Clap Drive	50

8.7	Ultra Drive	51
8.8	Ultra Wall.....	51
8.9	Reaction1.....	52
8.10	Reaction2.....	53
9	Appendiks B – Matlab Program kode	54
9.1	Manuell Kjøring	54
9.2	Automatisk Kjøring	57
9.3	Grafisk Brukergrensesnitt (GUI)	60
9.4	Cannon Game	61
9.5	Reaction1.....	64
9.6	Reaction2.....	65
9.7	KeyboardPiano	67
9.8	Gray Music.....	68
9.9	Joy Music	70
9.10	Clap Drive	71
9.11	Ultra Drive	72
9.12	Ultra Wall.....	73
9.13	Plot Timer	73
9.14	Math Show	75
9.15	NumJoy.....	77
10	Appendiks C – Matlab funksjons kode	80
10.1	Autofunc	80
10.2	Deg2Dist	80
10.3	Dist2Deg	80
10.4	derivFunk.....	80
10.5	filtJoy	80
10.6	filtLys	81
10.7	initNXT	81
10.8	intFunk.....	81
10.9	Lfilter	81
10.10	LtoHZ	81

10.11	motorPaadrag.....	81
10.12	retFunk	82

This page has intentionally been left blank.

2 SAMMENDRAG

Vi er en litt spesiell gruppe siden alle gruppemedlemmene studerer deltid ved siden av jobb. Arbeidstider varierer for alle gruppemedlemmene og vi har derfor måttet organisere arbeidet forskjellig fra det som er tradisjonelt. Vi har også måttet ta i bruk mange typer teknologier for å koordinere arbeidet oss imellom. Dette er beskrevet i mer detalj i kapittel 3 som omhandler arbeidsform og metodikk for prosjektgruppen.

Tidligere erfaring blant gruppemedlemmene varierte en del, men vi er omforent om at alle har bidratt like mye i forhold til ferdighet. De som hadde mer erfaring å løste sine oppgaver raskt brukte tid på å hjelpe de andre. Hovedfokus har gjennom hele prosjektet vært at alle i gruppen skulle få en god forståelse av hvordan programmeringen ble gjort for å få størst mulig læringseffekt av prosjektet. Dette hadde særs stort fokus på den obligatoriske delen, hvor alle fikk oppgaver som gjorde at en ble godt kjent med hele koden. Måten vi løste den obligatorisk delen på er beskrevet i kapittel 4.

Før første møtet med foreleser hadde vi fullført det meste av den obligatoriske delen inkludert verifisering. Dette ble gjort individuelt av alle gruppemedlemmene og deretter samlet sammen til ett felles script av en person som også hadde ansvar for å utbedre denne koden med tilleggs element. Vi ble også enige om hvilke kreative oppgaver vi skulle løse og hvem som skulle gjøre hva. De kreative oppgavene delte vi inn i kategorier som var; *Spill, Musikk, Robofun og Matematisk funksjoner og plotting*. For å gjøre det enklere å navigerer mellom alle programmene ble vi enige om å strukturere de som funksjoner slik at vi kunne lage en Graphical User Interface (GUI) for å kalle programmene. Måten de kreative oppgavene ble løst på kan man lese om i kapittel 5.

Vi har valgt å samle konklusjon per program i eget avsnitt i kapittel 6. Etter dette finner man referanser i kapittel 7 og deretter 8 Appendiks A som inneholder en del illustrasjoner som er referert til i teksten. I 9 Appendiks B ligger alt av Matlab kode for programmene. 10 Appendiks C inneholder koden for alle funksjoner som er brukt i programmene (sortert etter funksjonsnavn).

Som en avsluttende kommentar kan det nevnes at alle i gruppen har brukt minst 15-20 timer i uken for å fullføre prosjektet, selv om det noen uker også var mer. Motivasjonen har vært stor og likeså læringseffekten med bruk av Matlab i kombinasjon med Lego NXT. Vi ser alle frem til videre studier og håper med dette leseren vil finne prosjektoppgaven interessant.

3 PROSJEKTGJENNOMFØRINGEN

3.1 MÅL

Den obligatoriske delen av dette prosjektet gikk ut på å kjøre en Lego NXT robot gjennom en løype. Banen hadde et gradert gråtone felt hvor målet var at roboten skulle holde seg i midten mens en kjørte. Roboten måtte programmeres slik at den kunne angi via lysmåler hvor på banen den var. Programmene skulle skrives i Matlab og roboten skulle styres via joystick.

Videre i prosjektet skulle gruppen lage egne kreative program med lego roboten for å demonstrere programmerings kunnskap i Matlab.

3.2 ARBEIDSFORM

Vi var 4 medlemmer på gruppen som alle studerer deltid ved siden av jobb, og har variert arbeidstid noe som førte til at vi måtte føre en litt annen arbeidsform enn andre grupper. Vi gjorde mye av arbeidet individuelt og koordinerte koden i gruppemøter. Vi rullerte roboten i mellom gruppemedlemmer og kommuniserte mye via epost/telefon og delte kode via Github.

Prosjektet startet med å avtale agenda for første gruppemøte og sette opp Github konto for å dele matlab kode med kildehåndtering. Første gruppemøte ble avholdt 16. September (2t). Ansvar ble fordelt og tidsplan under ble vedtatt:

- 10. okt - ferdig med obligatorisk del
- 24. okt - Ferdig med skisse av kreativ del
- 7. nov - Ferdig med kreativ del for å ferdigstille rapport frem mot innleveringsfrist
- 21. nov - innleveringsfrist, prosjekt ferdig

Obligatorisk del ble gjennomført av alle gruppemedlemmene i hovedsak individuelt, men også noe i grupper alt etter hva som passet med arbeidssituasjon. Fokus var at alle gruppemedlemmer skulle få en god forståelse av den obligatoriske delen. All kode ble deretter kombinert til en felles fil for gruppen. Ansvar ble fordelt på hver enkelt i gruppen og alle har jobbet minst 15-20 timer i uken med prosjektet.

3.3 METODIKK

3.3.1 UTSTYR

Gruppen ble utstyrt med en Lego Mindstorm robot med to motorer og en rekke sensorer (lys, ultralyd, lyd, etc.). Roboten ble bygget etter instruksjonene i prosjektbeskrivelsen og kun små modifikasjoner ble gjort i enkelte kreative prosjekt.

Matlab ble brukt for å programmere scriptene som ble brukt i prosjektet sammen med toolbox og joystick drivere som fulgte prosjektbeskrivelsen.

Alle gruppemedlemmene benyttet private EDB apparater i prosjektet.

3.3.2 PROGRAMMERING

Hovedfokus i prosjektet har vært å få et bra samarbeid rundt programmeringen slik at alle skulle forstå de forskjellige delene av prosjektet. Dette var spesielt viktig i siden alle gruppemedlemmene jobbet med varierende skift.

Vi gikk gjennom obligatorisk del av koden sammen etter at alle hadde løst det individuelt, for å bli enige om hvordan vi skulle kombinere det til ett script. For de kreative oppgavene gikk vi gjennom på gruppemøter hvordan det kunne være lurt å løse problemet. Vi begynte med å forstå oppgaven og gikk deretter videre med å prøve å skissere en løsning med flow diagram. Koding ble deretter håndtert individuelt og løsning diskutert i plenum i etterkant.

For å dele koden benyttet vi Github for å ha kildehåndtering slik at flere kunne jobbe med samme filer samtidig. Vi bestemte oss også for å velge kreative program i noen hovedkategorier (*Spill, Musikk, Robofun og Matematisk funksjoner og plotting*) å flere små program herunder. Vi laget også et graphical user interface (GUI) for å kunne velge hvilke program man skulle kjøre. Dette var for å gjøre det mer oversiktlig.

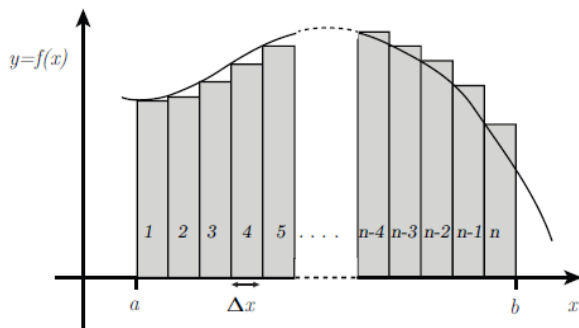
4 HOVEDPROGRAM - MAIN

Programmet Main sin hovedoppgave er å muliggjøre en manuell kjøring rundt en bane ved hjelp av joystick. Roboten har to motorer som er tilegnet de forskjellige aksene på joysticken. Motorene sitt motorpådrag følger joysticken sin bevegelse vekk fra nullpunktet mot et maks fremover punkt på 100 % motorkraft og maks bakover punkt på -100 % motorkraft. Ved bevegelse mot høyre/venstre vil roboten gå rundt i en sirkel med en motorkraft basert på hvor langt mot høyre/venstre joysticken er. Roboten leser av banen med lyssensoren og verdien på gråtonen som lyssensoren kjører over indikerer om den er på vei mot lys eller mørk side. Dette er illustrert under kjøring. For å programmere dette ble det brukt numerisk integrasjon og derivasjon av lysmåling over tid. Dette er forklart i avsnittene under samt hvordan automatisk kjøring ble gjort og andre spesialiteter som ble lagt til i hovedkoden.

4.1 NUMERISK INTEGRASJON

For å finne ut hvor nøyaktig roboten kjørte brukte vi Eulers forover metode for å integrere det tidsdiskret lysmålesignalet. Dette er numerisk integrasjon som vi måtte bruke siden vi ikke kan finne en eksakt løsning når målesignalet er tidsdiskret (målingene har varierende tidsmellomrom).

Metoden var beskrevet veldig nøye i prosjektbeskrivelsen og vi ble enige om at denne metoden var god nok for dette formålet, så vi prøvde ikke å benytte trapes-metoden selv om denne sannsynligvis ville gitt et mer nøyaktig resultat. Vi har tiden langs x-aksen og lysmåling på y-aksen. Integrasjonen blir brukt til å finne arealet under funksjonen ved å summere arealet til stolpene man får mellom hvert målepunkt og som har endring i tid som bredde og lysmåling som høyde illustrert i Figur 1 under.



Figur 1- illustrasjon av prinsippet for numerisk integrasjon hentet fra prosjektbeskrivelsen

Funksjonen `intFunk` for å integrere ble skrevet som følger:

```
function intOut = intFunk(x,y)
%integrer med hensyn på x vektorene x og y
dt=x(end)-x(end-1);
intOut=y(end-1)*dt;
end
```

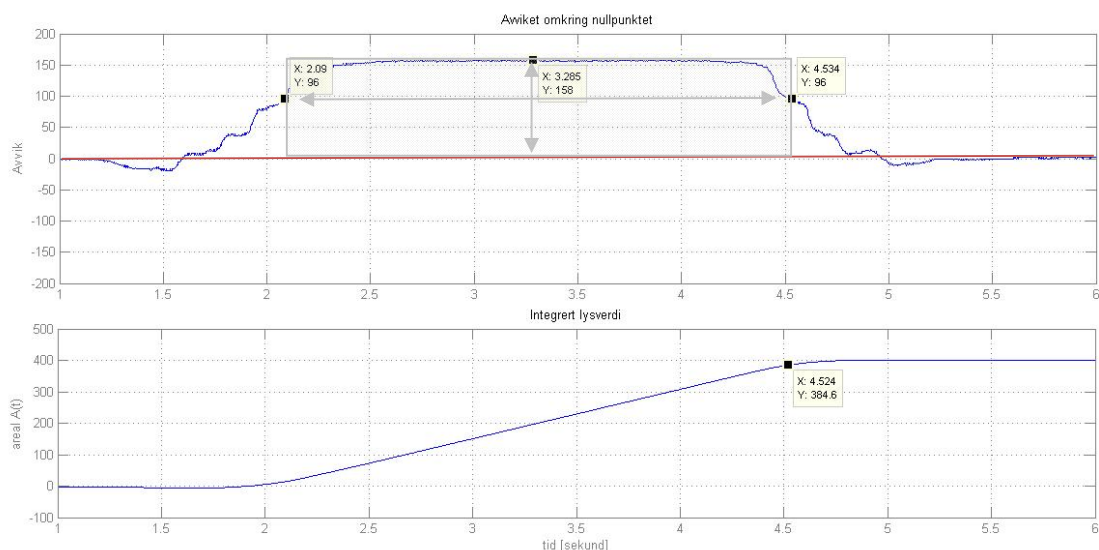
Man ser her at funksjonen `intFunk` lagrer siste endring i x verdi (tiden) i variabelen `dt`. Forrige y verdi blir multiplisert med `dt` for å gi integralet, eller arealet av en stolpe som vist i Figur 1.

Kodeuttrekk fra main koden hvor integreringsfunksjon ble brukt var som følger:

```
avvikL(end+1)=lysFilt(end)-lysNp;
avvikA(end+1)=intFunk(tid,avvikL)+avvikA(end);
avvikA2(end+1)=abs(intFunk(tid,avvikL))+avvikA2(end);
```

Her blir altså `avvikL` regnet ut først i linje 1 og er filtrert lysmåling minus nullpunkt for lysmåling `lysNp`. Integrasjonen blir altså regnet rundt nullpunktet. `avvikA` er vektor som inneholder kumulativt areal for stolpene. Her brukes funksjonen `intFunk` for å integrere og resultatet blir summert med forrige areal. Linje 3 over viser hvordan absolutt areal blir regnet ut for å gi et bedre bilde på totalt avvik. Det er denne verdien som blir brukt i konkurransen.

For å kontrollere at funksjonen ga riktig resultat verifiserte vi med utregning fra figuren som illustrert i Figur 2 under.



Figur 2 - Verifisering av integrasjonsrutine

Avviket ble beregnet på følgende måte:

$$\text{Endring i tid } x = 4,53 - 2,09 = \underline{2,44}$$

$$\text{Avvik} = 158$$

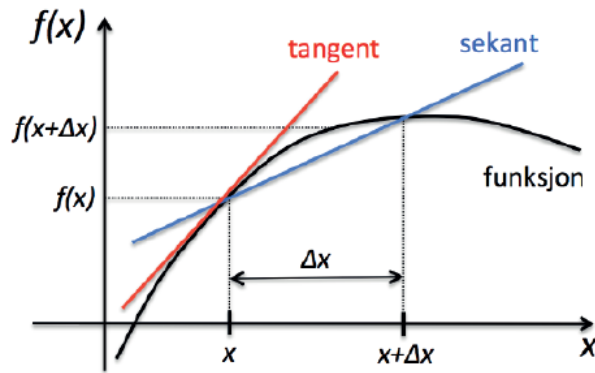
$$A(t) = 2,44 * 158 = \underline{385}$$

Dette stemmer bra med integrert lys verdi som man kan se i Figur 2 er avrundet lik 385. Den komplette figuren som også viser lysmåling fra verifisering finnes i Figur 17 i Appendiks.

4.2 NUMERISK DERIVASJON

For å finne ut hvor raskt målingen endrer seg regner vi ut den deriverte. Dette blir brukt i programmet for å indikere hvilken retning roboten kjører (mot lysere eller mørkere side). Dette kan vi ikke regne ut nøyaktig men brukersekanten for å gi et estimat når delta x ikke er tilnærmet null.

Dette er illustrert i Figur 3 under, som viser tangenten som er den nøyaktige utregningen når delta x går mot null.



Figur 3 - Illustrasjon av derivasjon hentet fra prosjektbeskrivelsen

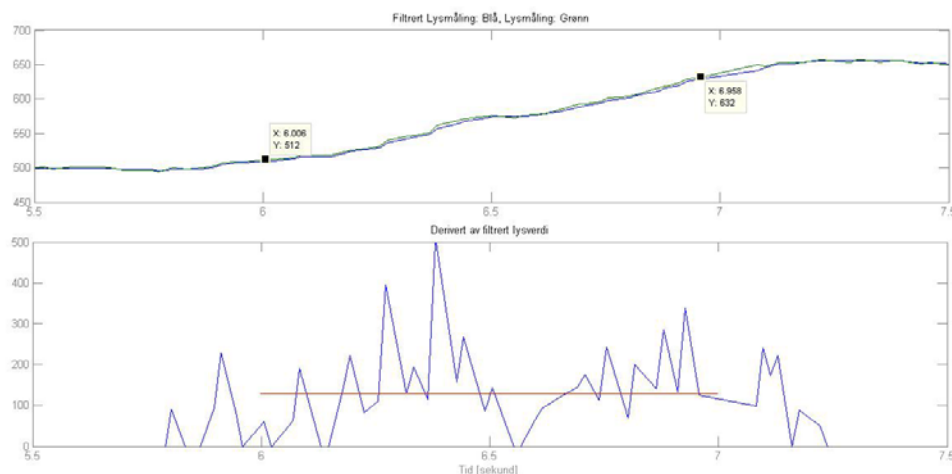
Funksjonen derivFunk brukte for å regne ut den deriverte og ble skrevet som følger:

```
function out = derivFunk(x,y)
%deriverer endring i y delt på endring i x
dy=y(end)-y(end-1);
dx=x(end)-x(end-1);
if dx==0
    dx=0.01;
end
out=dy/dx;
end
```

Denne funksjonen tar ganske enkelt siste verdi i y minus nest siste for å finne endring i y. Den finner så endring i x på samme måte, men sjekker i tillegg om denne endringen er null og justerer i så tilfelle verdien til 0.01 slik at man ikke får problemer med å dele på null. Endring i y blir delt på endring i x for å gi den deriverte ut. Måten dette ble brukt i programmet er hentet ut under hvor avvikL er filtrert lysmåling minus nullpunkt.

- deriv(end+1)=derivFunk(tid,avvikL)

Figur 4 under ble brukt for å verifisere at derivasjonsrutinen fungerte.



Figur 4 - derivasjonsverifisering

Man kan med bakgrunn i dette regne ut sekanten til målingen med datapunktene i grafen vist øverst i Figur 4, hvor beregningen blir som følger:

Endring i lysmåling $y = 632 - 512 = 120$

Endring i tid (x) = $7 - 6 = 1$ sekund

Derivert = $120 / 1 = 120$

Man ser dermed at den røde linjen som er tegnet inn på 120 ligger omtrent på gjennomsnittet for de deriverte verdiene og man kan konkludere med at derivasjonsrutinen fungerer til sin hensikt.

4.3 FILTERING

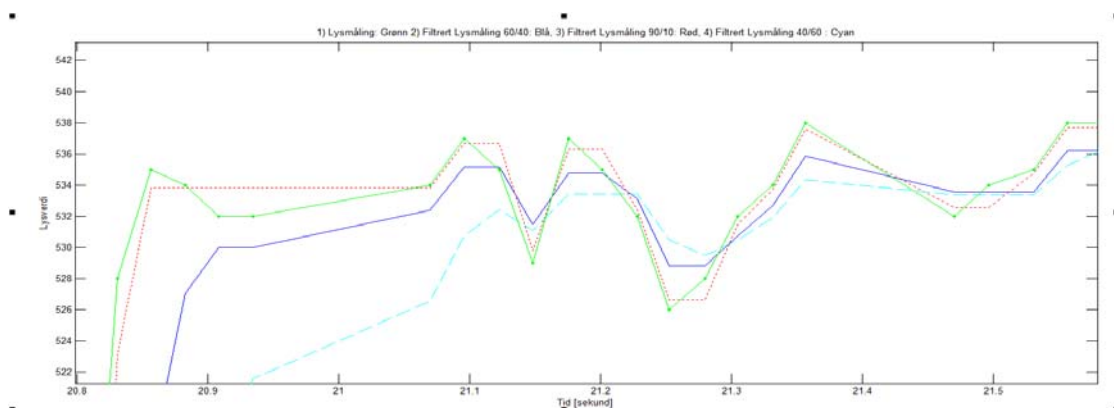
Vi valgte å lage funksjoner av filtreringen vi gjorde og filtrering av lysmåling ble gjort i funksjonen `filtLys` som vises under.

```
function out = filtLys(in)
%Filtrerer en input vektor til et tall ut
%
    temp = in(end)-in(end-1);
% if change is less than +-2 then take last value
    if temp < 2 && temp > -2
        out = in(end-1);
% filter joy input
    else
        out = 0.6*in(end)+0.4*in(end-1);
    end
end
```

I selve programmet så koden ut som under etter at en hadde lest inn lyssignalet.

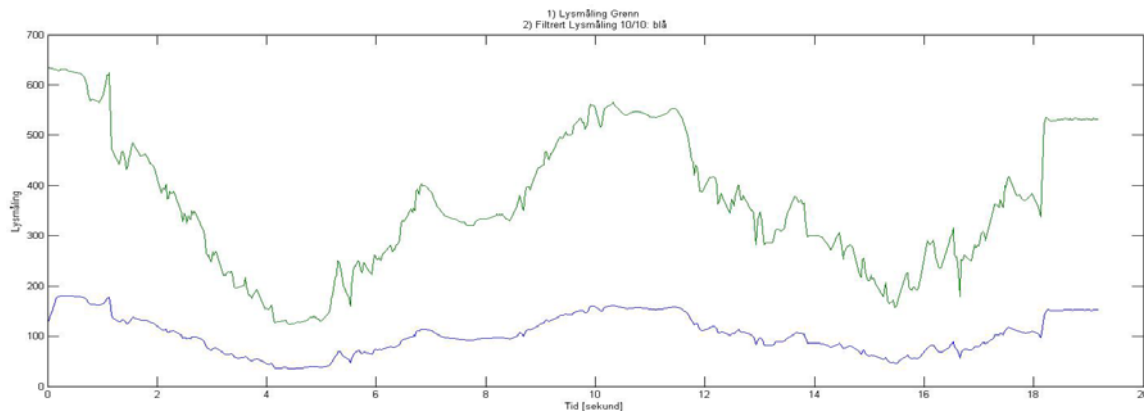
```
lys(end+1)=GetLight(SENSE_3);
lysFilt(end+1)=filtLys([lysFilt(end),lys(end)]);
```

Fra kode uttrekket ser man at vi valgte å bruke rekursiv filtrering (IIR-filtrer) for å filtrere lysmålingen da vi følte dette ga best resultat i forhold til å glatte måle verdiene med et FIR-filtrer. I Figur 5 under ser en eksempel fra en av testene vi gjennomførte for å komme frem til passe innstillinger for lysmåle filteret. Den grønne linjen viser ufiltrert lysmåling. Den blå heltrukne linjen viser filtrerings innstillingen vi valgte å bruke som var 60% fra ny måling og 40% fra forrige filtrerte verdi. Den prikkete røde linjen viser hvordan filteret fungerte om man brukte 90% fra ny måling og 10% fra forrige filtrerte måling. Den stripete linjen i cyan viser hvordan filteret fungerte om man brukte 40% fra ny måling og 60% fra forrige filtrerte måling. I figuren ser man hvordan den blå linjen gjenspeiler målesignalet greit men man unngår også de brå endringene. Dette filteret viste seg også å fungere bra i praksis.



Figur 5 - Test av innstilling på lysmålings filter

Vi testet også hva som skjedde om summen på vektene i filteret ikke summerte seg opp til 1. Forventningen her var at om den var under 1 ville filtrert målesignal hele tiden reflektere et for lavt nivå, og var den over ville det bli for store verdier i den filtrerte målingen. I Figur 6 under ser man hva resultatet blir om man legger inn kun 10% av ny lysmåling og 10% av forrige filtrerte lysmåling. Vi forventet da at filtrert måling bare vil være 20% av ufiltrert måling. Dette bekreftes også i figuren.



Figur 6 - IIR filter hvor vekting ikke = 1

4.4 MANUELL KJØRING

Først opprettes kobling mot NXT og Joystick, variabler og figurer initialiseres. Tiden siden oppstart legges inn i siste element i tidsvektoren og en beregner tiden på forrige løkkegjennomkjøring ved å ta tiden i starten av denne løkken å trekke fra tiden i starten av forrige løkke. Denne skal brukes som delta tid (x) i kalkulasjoner.

```
tid(end+1)=toc;
deltaTid(end+1)=tid(end)-tid(end-1);
```

Lyset leses inn og filtreres basert på siste lys verdi og siste filtrerte lys verdi, altså ved bruk av rekursiv filtrering.

```
lys(end+1)=GetLight(SENSOR_3);
lysFilt(end+1)=filtLys([lysFilt(end),lys(end)]);
```

Avviket for lyset fra nullpunkt regnes ut. Avviket integreres og legges inn i en egen vektor, dette gjøres i en egen funksjon ved bruk av en delta tid og avviket fra nullpunkt som blir en slags $f(t)$. En legger til siste verdi av integralet for å få hele integral summen. Tilsvarende integrasjon gjøres for en egen vektor, men denne bruker absolutt verdien av løkke integralet for dermed å få totalt avvik fra nullpunkt.

```
avvikL(end+1)=lysFilt(end)-lysNp;
avvikA(end+1)=intFunk(tid,avvikL)+avvikA(end);
avvikA2(end+1)=abs(intFunk(tid,avvikL))+avvikA2(end);
```

Derivasjonen gjøres så i en egen funksjon ved bruk av en delta tid og delta y som regnes fra endringen i lyset verdi. (sjekker også for å unngå ikke kalkulerbare verdier).

```
deriv(end+1)=derivFunk(tid,avvikL)
```

Retningen avgjøres i egen funksjon og legges i egen vektor, det må være en endring fra forrige retning, dersom endringen gir 0 skal retning beholdes lik den foregående. Verdien i konkurransen utregnes fra oppgitt formel og legges til slutten av en egen vektor.

```
rettning(end+1)=retFunk(deriv(end),rettning(end))
verdi(end+1)=tid(end)*100+avvikA2(end);
```

Joystickens verdier leses inn i egne vektorer og justeres for initiert nullverdi, noe som er beskrevet i egen seksjon lenger nede i rapporten. Denne verdien kjøres gjennom et filter og oppdateres for ikke å gi for «nappete» kjøring.

En bruker retningen til å sette variabler for visning av dette noe som også er beskrevet i egen seksjon under. Pådraget til hver av motorene regnes ut i egen funksjon basert på joystickens filtrerte inn verdier og pådraget settes.

```
[paadragB(end+1),paadragC(end+1)] = motorPaadrag(joyFB(end),joyS(end));
```

Henter data fra hver av motorene for å avgjøre hastighet. Dette gjøres ved å se på gjennomsnitt strekning motorene har beveget seg, beskrevet lenger nede i rapporten. Plott frekvensen avgjør om det skal plottes (ved bruk av handles) eller om en bare skal oppdatere telleren som sjekker mot plott frekvens grensen. Grafikk oppdateres eventuelt og sjekker om koden skal avsluttes eller ny løkke skal kjøres, og prosessen startes eventuelt på nytt på. Når en ikke ønsker å kjøre lengre (løkken avsluttes) stoppes motorene, sensor koblinger stegnes og joystick slettes. Figurer fra kjøring stenges og nye mer detaljerte figurer basert på all data som har blitt tatt vare på tegnes. En løkke for å vente på at bruker skal kunne se på figurer settes før en kjøres til bruker trykker en knapp, da avsluttes programmet.

4.5 AUTOMATISK KJØRING

Ansvarlig: Anders Svalestad

HENSIKT

Hensikten er å kjøre automatisk gjennom løype ved bruk av lyssensor.

OPPBYGNING

Programmet tar i bruk lys sensor for å kjøre robot automatisk gjennom utdelt løypeprofil. Løype banen er tegnet med gråskala som går fra hvit, mot sort. Vi ønsker å kjøre roboten i senter av banen. Hensikten med den automatiske kjøringen er at programmet skal styre roboten med minst mulig avvik fra banesenter.

Først opprettes kobling mot NXT og Joystick, variabler og figurer initialiseres. Tiden siden oppstart legges inn i siste element i tidsvektoren og en beregner tiden på forrige løkkegjennomkjøring ved å ta tiden i starten av denne løkken å trekke fra tiden i starten av forrige løkke. Denne skal brukes som delta tid (x) i kalkulasjoner.

```
tid(end+1)=toc;
deltaTid(end+1)=tid(end)-tid(end-1);
```

Lyset leses inn og filtreres basert på siste lys verdi og siste filtrerte lys verdi. Filtrert lysverdi bruker for eksempel 60% av inlest lysverdi sammen med 40% av siste verdi.

```
lys(end+1)=GetLight(SENSOR_3);
lysFilt(end+1)=filtLys([lysFilt(end),lys(end)]);
```

Avviket for lyset fra nullpunkt regnes ut. Avviket integreres og legges inn i en egen vektor, dette gjøres i en egen funksjon ved bruk av en delta tid og avviket fra nullpunkt som blir en slags $f(t)$. En legger til siste verdi av integralet for å få hele integral summen. Tilsvarende integrasjon gjøres for en egen vektor, men denne bruker absolutt verdien av løkke integralet for dermed å få totalt avvik fra nullpunkt.

```
avvikL(end+1)=lysFilt(end)-lysNp;
avvikA(end+1)=intFunk(tid,avvikL)+avvikA(end);
avvikA2(end+1)=abs(intFunk(tid,avvikL))+avvikA2(end);
```

Derivasjonen gjøres så i en egen funksjon ved bruk av en delta tid og delta y som regnes fra endringen i lyset verdi. (sjekker også for å unngå ikke kalkulerbare verdier).

```
deriv(end+1)=derivFunk(tid,avvikL)
```

Retningen avgjøres i egen funksjon og legges i egen vektor, det må være en endring fra forrige retning, dersom endringen gir 0 skal retning beholdes lik den foregående. Verdien i konkurransen utregnes fra oppgitt formel og legges til slutten av en egen vektor.

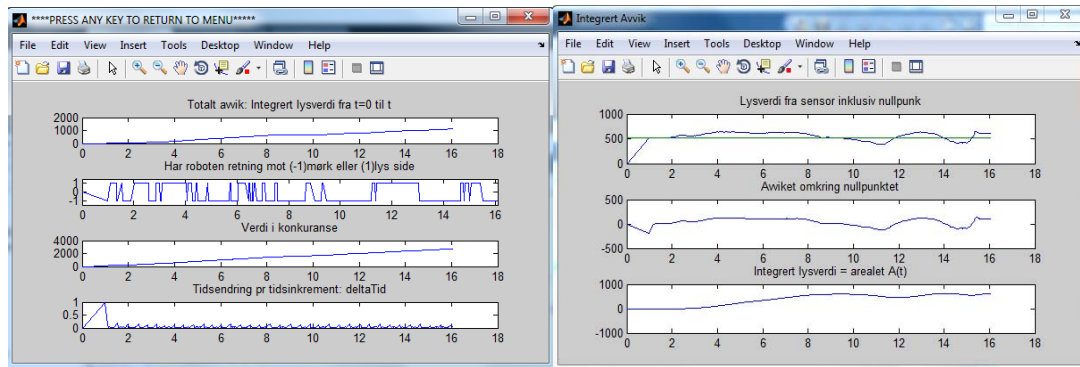
En bruker retningen til å sette variabler for visning av dette noe som også er beskrevet i egen seksjon under. Pådraget til hver av motorene regnes ut i egen funksjon basert på joystickens filtrerte inn verdier og pådraget settes.

```
rettning(end+1)=retFunk(deriv(end),rettning(end))
verdi(end+1)=tid(end)*100+avvikA2(end);
```

Pådrag til hver motor beregnes ut fra lysverdiens avvik fra nullpunkt. En definerer en fast fart fremover. Pådraget til de to motorene blir deretter regulert ved å legge til/trekke fra en prosent av lysavvik fra 0 punkt.

```
[paadragB(end+1),paadragC(end+1)] = Autofunc(avvikL(end));
```

Når robot har fullført løype banen å går over på hvit papir, går lys verdien over en presatt verdi. (løkken avsluttes) Da stoppes motorene og sensor koblinger stenges. Figurer fra kjøring stenges og nye mer detaljerte figurer basert på all data som har blitt tatt vare på tegnes. En løkke for å vente på at bruker skal kunne se på figurer settes før en kjøres til bruker trykker en knapp, da avsluttes programmet.



Figur 7 -Autokjøring avsluttende plot

UTFORDRINGER

Lyssensoren har begrensninger å miljøet rundt påvirket resultatene i stor grad. For eksempel lysforhold i rommet du kjører, påvirker operasjon. Små justeringer av parametere må påregnes når miljø endres.

KONKLUSJON

Automatisk kjøring virker greit. Negativt at koden har en presatt makshastighet.

Løsningen er en relativ enkel pådragsregulering. Det finnes bedre, mer avanserte måter å løse oppgaven på som ville latt robot regulere hastighet, og kjørt løypen raskere.

4.6 TILLEGG TIL OBLIGATORISK DEL I HOVEDPROGRAM

Ansvarlig: Helge Bjorland

I tillegg til det som er nevnt over har vi gjort mange forandringer for å forbedre det obligatoriske programmet og automatisk kjøring. Dette ble gjort med følgende tillegg:

1. Funksjoner

Det er brukt til sammen 8 funksjoner i manuell og auto kjøring. Når vi lærte å bruke dette oppdaget vi fort at dette var en veldig grei måte å gjøre koden ryddig. Det er også mulig å bruke funksjoner om igjen i andre program, så det var veldig greit. Det gikk med en del tid for å teste funksjoner, men det var nyttig å lære ordentlig.

2. Handles i grafer

Vi merket fort at det gikk veldig lang tid å oppdatere grafene i while loopen mens en kjørte roboten. Dette gjorde at vi fikk færre målepunkter fra roboten. Vi laget først en input parameter hvor en kunne velge hvor ofte grafene skulle oppdateres. Dette fungerte men ga til dels statiske grafer. Vi ønsket derfor å finne en bedre måte å gjøre dette på å begynnte å teste de forskjellige måtene å plote på. Dette kan en lese om i egen seksjon under kreative oppgaver. Konklusjonen var at handles var den raskeste måten å oppdatere grafer på. Vi valgte derfor å implementere denne metoden for å oppdatere grafer under kjøring. Dette tok en del tid å få satt opp, men når vi fikk det til å fungere virket det mye bedre. Koden for graf oppdatering under kjøring så ut som under:

```
figure(1)
set(plot1_1,'Ydata', avvikA2 , 'Xdata', tid);
```

Vi testet også med visning av kun de 100 siste verdiene under kjøring. Dette fungerte greit, men vi merket ikke så mye forskjell etter å ha implementert handles for å oppdatere grafene. Koden utsnitt fra testen kan ses under.

```
if plotTeller > plotFrek
    if length(tid) < 100
        elements = length(tid);
    else
        elements = 100;
    end
    plotTemp1 = avvikA2((end-elements):end);
    plotTemp2 = tid((end-elements):end);
    figure(1)
```

3. Fartsmåling

Vi la til fartsmåling som viste under kjøring. For å gjøre dette brukte vi posisjon fra hjul og konverterte til avstand som ble delt på tid for å få ut hastighet i mm/s. Måten dette ble implementert i koden er vist under:

```
% Gjennomsnitt mm avstand på begge motorer
LenRead(end+1) = (Deg2Dist(sBdata.Position)+ Deg2Dist(sCdata.Position))/2;
% delta avstand (mm) / delta tid (s)
speed(end+1) = round(((LenRead(end)-LenRead(end-1))) / deltaTid(end));
```

Vi prøvde også å bruke akselerator for å måle fart men denne sensoren viste seg å være defekt da den bare målte høyde over bakken. Vi brukte såpass mye tid på å bekrefte at den var defekt at vi fikk derfor ikke sammenlignet fetsmåling ved bruk av andre typer sensorer som planlagt.

4. Grafisk retnings visning

For å illustrere om roboten kjører mot lys eller mørk side ønsket vi istedenfor en graf å vise dette grafisk med bilder av piler som pekte i riktig retning. Vi importerte bildene i starten av koden og viste disse under kjøring alt etter hvilken side man kjører mot. Dette implementerte vi i programmet med kode uttrekket som er vist under.

```
% 1 angir retning mot lysere side (venstre)
if rettning(end) > 0;
    arrow=lArrow;
    map=lmap;
    % -1 angir retning mot mørkere side (høyre)
elseif rettning(end) < 0;
    arrow=rArrow;
    map=rmap;
end
```

5. Nullstilling av joystick

I begynnelsen begynte roboten å kjøre uten at man rørte på joystick. Denne «bugen» ønsket vi å fjerne så valgte derfor å inkludere en nullstilling av joystick i starten. Dette gjorde vi ved å lese av verdien joystick hadde når man startet programmet og deretter trekke fra denne verdien fra alle påfølgende joystick verdier. Måten dette ble gjort på er indikert i gult i kode uttrekket under.

```
joyFB(end+1) = (-joystick.axes(2)/327.68)-initFB; % henter joystick posisjon på "y"-aksen
joyS(end+1) = (joystick.axes(1)/327.68)-initS; % henter joystick posisjon på "x"-aksen
```

6. Tidsmåling

Vi brukte først funksjonen cputime for å angi tid i programmet. Når vi testet dette fant vi at siden den gir tiden som matlab prosessen har hatt hos cpu siden den ble startet, gir den alltid litt mindre tid en reelt. Dette er nok avhengig av hva man gjør men når vi testet så telte den bare 0.9s pr 1s. Vi valgte derfor å bruke tic/toc funksjonen for å forbedre dette. De to kode uttrekkene under viser hvordan vi endret koden.

```
while run
    % Tids beregninger
    tid(end+1)=cputime-startTid;
    deltaTid(end+1)=tid(end)-tid(end-1);
```

```
tic;
while run
    % Tids beregninger
    tid(end+1)=toc;
    deltaTid(end+1)=tid(end)-tid(end-1);
```

4.7 KONKLUSJON

Etter å ha jobbet mye med hovedprogrammet for manuell og automatisk kjøring fungerte disse bra og etter sin hensikt.

5 KREATIVE PROGRAM

5.1 GRAFISK BRUKERGRENSESNIFF (GUI)

For å gjøre det enklere å navigere mellom alle programmene vi etterhvert hadde laget vi en GUI. Her får man 3 valg til å starte med (Robot kjøring, Kreative oppgaver og Avslutt). Ved å velge her blir man tatt videre i nye menyer som illustrert under i Figur 8.



Figur 8 - GUI for alle program

For å lage GUI brukte vi while loop men switch/case sammen med menu funksjon. Dette er illustrert i kodesnutten under.

```
n=1
while n>0
    switch menu('Int100 prosjekt', 'Robot kjøring', 'Kreative oppgaver', 'Avslutt')
        case 1
            m1=1
            while m1>0
                switch menu('Robot kjøring', 'Manuell kjøring', 'Automatisk kjøring',
                    'Tilbake')
                    case 1
                        main()
                    case 2
                        auto()
                    case 3
                        m1 = 0; % Gå tilbake til forrige meny
                end
            end
        end
    end
```

5.2 SPILL

5.2.1 CANNON GAME

Ansvarlig: Espen Ro Eliassen

HENSIKT

Lage et “angry birds” liggende spill. Det betyr at det må simulere et skrått kast på en eller annen måte. Det er viktig i denne oppgaven at det ikke kun er en fysikk simulering, men at det også føles som et reelt spill.

OPPBYGNING

Programmet er skrevet i en funksjon som igjen inneholder underfunksjoner. Det startes med å definere konstante verdier som skal brukes i koden. Blant disse finner vi skjermstørrelse, et forhold mellom skjermstørrelse og plot størrelse, plott akser, gravitasjonens akselerasjon og start fart til prosjektillet. Se seksjon “Constants”.

Etter konstanter følger initialisering av variabler. Disse inneholder både betingelses variabler for å vise forskjellige deler av programmet og variabler for matematiske kalkulasjoner. Poengsum liste lastes her inn fra en underfunksjon “loadHighscore()”. Se ellers seksjon “Initiate variable”.

loadHighscore() funksjonen er en underfunksjon som returnerer en struct variabel. Denne hentes inn fra filen “highscore_cannongame.mat” og inneholder feltene lengste skudd (“lShots”), lengste total på 3, 5, 7 og 10 skudd, (“tot3shots”), (“tot5shots”), (“tot7shots”) og (“tot10shots”). Om ikke filen finnes skal den lages. Hvert felt inneholder en 1x10 matrise.

```
%load highscore from file
function hs = loadHighscore()
try %try loading highscore file
    hs=load('highscore_cannongame.mat');
catch %if loading highscore file fails, make it
    lShots = zeros(1,10); %longest shots
    tot3shots = zeros(1,10); %highest total of 3 shot games
    tot5shots = zeros(1,10); %highest total of 5 shot games
    tot7shots = zeros(1,10); %highest total of 7 shot games
    tot10shots = zeros(1,10); %highest total of 10 shot games
    save('highscore_cannongame','lShots','tot3shots','tot5shots','tot7shots','tot10shots'); %create the highscore :
    hs=load('highscore_cannongame.mat');
end
end
```

Når konstanter og variabler er satt opp er det på tide å starte selve spillet. Det gjøres først ved noen begynnende valg, se “Initial game options”. Her har en mulighet for å velge en- eller fler spiller, se høyeste poengsummer eller gå ut av spillet. Alt styrt ved bruk av menu funksjonen som gui. Etter en har valgt antall spillere får en mulighet å velge vanskelighets grad. Vanskelighetsgraden avgjør hvor mange radianer det er fra en vist vinkel til den neste. Altså utskytningsvinkelen endrer seg fortere og det blir vanskeligere å treffe den optimale vinkelen.

Hovedfiguren defineres i en egen seksjon, se “Initialize figure”, dynamisk ut fra skjermstørrelsen. Den starter 1/6 av høyde og bredde fra hjørnet og strekkes seg høyde og bredde delt på 1.5 (2/3 av de respektive størrelsene) fra start punktet. Figurens “KeyPressFcn” defineres som underfunksjonen “keyDownListener”. I tillegg så settes figuren slik at den ikke kan forandres størrelse på. Det kan nevnes at det kunne vært satt flere underfunksjoner. F.eks ville en funksjon som håndterte figurens “CloseRequestFcn”, være på sin plass i et reelt spill.

Underfunksjonen “keyDownListener” inneholder 2 muligheter. Ved at bruker trykker mellomrom setter den variablene “space” og “showangle” til 1 og 0. Denne vippen brukes for å kontrollere fremgang i selve hoved løkken. Når escape trykkes settes “stop” variabelen til 1.

Før selve hovedprogrammet/-løkken kommer det flere underfunksjoner som har forskjellige oppgaver. De to neste på listen er updateLongestShotHS og updateTotalLengthHS. Disse funksjonene svært lik funksjonalitet. Den første oppdaterer lengste skudd listen, mens den siste oppdaterer den riktige høyeste totaltsum listen basert på hvor mange runder er valgt. Grunnen til at disse er skilt i 2 funksjoner er fordi lengste skudd skal sjekkes/oppdateres hvert skudd, mens total sum oppdateres ved endt spill. Logikken for oppdatering er lik i begge funksjonene. Det gjøres ved at oppnådd lengde/sum legges til listen, så sorteres den og det siste elementet fjernes.

```
function updateLongestShotHS(shotlength)
    highscore.lShots(end+1) = shotlength; %add current shots to longest shot list
    highscore.lShots = sort(highscore.lShots,'descend'); %resort the longest shot list
    lShots = highscore.lShots(1:10); %remove the lowest value from longest shot list
    save('highscore_cannongame','-append','lShots'); %resave the longest shot list
end
```

showAngle er en av de mer spennende funksjonene i programmet. Det er en funksjon som styrer direkte i hoved funksjonaliteten til spillet. Den viser en linje som først øker fra 0° mot 90° grader for så å gå ned igjen mot 0°. Plasseringen til denne linjen når spiller trykker mellomrom knappen angir vinkelen prosjektillet skytes ut med. Linjen den tegner som angir vinkelen gis av et x,y punkt som regnes med trigonometriske funksjoner og origo. Her benyttes også konstanten chart ratio på y kalkulasjonen for å gi en illusjon av at linjen er tilnærmet like lang hele tiden, selv om skalaen på x og y aksene er forskjellige. Det er ingen returverdi fra denne funksjonen, istedenfor setter den angle variabelen direkte. Noe som er fordelen med å ha den som en egen underfunksjon, fremfor å trekke den ut som en enkeltstående funksjon. Se kode i appendiks og figur under.

Når en vinkel for skuddet er angitt er det under funksjonen shootCannon() som tar seg av både kalkulering av skudd og animasjon. Selve kalkuleringen gjøres ved å dekomponere utgangshastigheten, gitt av konstanten “initialSpeed”, i x og y deler ved vinkelen. Deretter kalkuleres det hvor lang tid det vil ta før prosjektillet treffer bakken igjen. Ut i fra denne kalkulasjonen lages en tidsvektor som igjen brukes til å animere selve skuddet. Det er vært å legge merke til at en antar at prosjektillet starter på bakken altså y=0. Lengden på skuddet regnes altså ut fra dekomponert x fart og tid i luften. Utregningen stemmer også overens med formelen for strekning gitt av . Dette kan bevises ved å manipulere koden til å kjøre med vinkel 45° som gir det lengste skuddet. Verdt å merke seg er at prosjektillet animeres via to og to punkter i flyve banen. Det hadde også vært mulig å gjøre det punktvis. Funksjonen returnerer både høyde og lengde på skuddet, selv om det bare er lengde som tas med i poengsummer. Det er fra denne funksjonene at lengste skudd oppdateringen for top listen oppdateres.

Siste under funksjon er showHighscores(), som rett og slett kalles fra de innledende valgene. Den viser alle høyeste poengsum / lengste lengde listene som vist i Figur 9. I forhold til logikk er det lite spennende som skjer. Det ligger noe enkel bearbeiding av variabler for å få det på riktig format for å vise dem med annotation funksjonen. Deretter kjøres en løkke for å vente til spiller er klar til å gå videre.

Highscores, Press "space" to continue				
Longest shots:	Highest Total 3:	Highest Total 5:	Highest Total 7:	Highest Total 10:
1. 1019.3086m	1. 3033.6058m	1. 5040.3631m	1. 7057.0737m	1. 5787.2344m
2. 1019.3086m	2. 3007.1452m	2. 4972.3844m	2. 6877.1066m	2. 3512.9716m
3. 1019.3086m	3. 2987.1584m	3. 4965.0297m	3. 4786.4664m	3. 2896.9189m
4. 1018.9333m	4. 2925.4318m	4. 4737.115m	4. 4241.567m	4. 0m
5. 1018.1343m	5. 2722.3049m	5. 4515.1076m	5. 2347.4063m	5. 0m
6. 1018.1343m	6. 2579.8701m	6. 4311.2848m	6. 0m	6. 0m
7. 1018.1343m	7. 2470.1869m	7. 4154.1536m	7. 0m	7. 0m
8. 1018.1343m	8. 2273.1164m	8. 4083.5122m	8. 0m	8. 0m
9. 1018.1343m	9. 2218.6323m	9. 3720.0977m	9. 0m	9. 0m
10. 1018.1343m	10. 2213.4725m	10. 1963.9656m	10. 0m	10. 0m

Figur 9 - Cannon Game HighScore table

Selve hoved løkken til programmet er egentlig 2 svært like løkker. Det er en for en spiller og en for flere spillere. Forskjellen ligger i noe ekstra logikk for å håndtere at det flere spillere. Derfor er det kun flere spiller løkken som beskrives her. Det settes opp matriser for å holde rede på skuddlengde (og høyde), hvor hver rad i i matrisen representerer en spiller og hver kolonne et skudd. En starter med spiller 1 tegner opp figuren og begynner med å kjøre showAngle(). Når spiller har trykket på på mellomrom for å skyte, flippes variablene via keyDownListener og space blir 1 og showangle 0. Dette plukkes opp i en if/elseif/else logikk. Dette sørger for at skuddet vises frem til spiller er klart til å gå videre og trykker mellomrom igjen. Dersom det er flere skudd igjen settes showangle variabelen til 1 igjen og hele prosessen gjentas. Dette gjøres om og om igjen til alle spillere har skutt alle sine skudd (eller escape knappen blir trykket). Da settes stop variabelen og løkken avsluttes.

```

while ~stop %main multiplayer game loop
    drawnow;
    if showangle %shot with the angle when player presses space
        showAngle();
    elseif space && ~dispshot %shot with the angle when player presses space
        [shotlength(playerno,shotno), shotheight(playerno,shotno)] = shootCannon();
        space=0;
        if playerno==nplayers %iterate through players and shot numbers
            shotno=shotno+1;
            playerno=1;
        else
            playerno=playerno+1;
        end
    elseif shotno<=nshots && space %if more shots remaining reset conditions
        showangle = 1;
        dispshot = 0;
        space=0;
    elseif shotno>nshots && space %all shots completed, finish game
        stop=1;
    end
end

```

Deretter kjøres en løkke som går igjennom summene til alle spillerne, avgjør hvem som vant og setter opp spill detaljer for hver spiller som vist i Figur 19 i appendiks. Når alle spillerne er blitt

vurdert settes en egen boks over detaljene med hvem som vant, og koden går inn i en løkke som venter på at spillere er klar til å fortsette.

```

while playerno <= noplayers
    tempTot=sum(shotlength(playerno,:)); %get total length for each player
    updateTotalLengthHS(tempTot); %update highscore with total length for each player
    if highTotThisGame < tempTot %decide who is winning
        winner = playerno;
        highTotThisGame = tempTot;
    end
    %create details to send to textbox
    tempstr=(['Player ',num2str(playerno)],['Longest shot: ', num2str(max(shotlength(playerno,:))), 'm'],
    ['Highest shot: ', num2str(max(shotheight(playerno,:))), 'm'],['Total shot length: ' num2str(tempTot)],
    %create textbox for each player
    annotation('textbox', [0.1+(0.15*(playerno-1)),0.1,0.15,0.8],'backgroundcolor', 'w', 'String', tempstr);
    playerno=playerno+1;
end
%create textbox with winner and continue instructions
annotation('textbox', [0.1,0.9,0.75,0.05],'backgroundcolor', 'w', 'String', ['Winner is Player ',num2str(winner),
space=0;
while ~space %wait until player is ready to continue
    drawnow
end

```

Avslutningsvis ligger det ligger det en liten logikk som kaller selve funksjonen selv igjen for å la spillere spille på nytt. Sett fra et minneperspektiv er det ikke noen ideell situasjon og kalle opp funksjonen inne i seg selv. Men i dette tilfellet anses muligheten for at noen klarer å sitte og spille sammenhengende nok til å klare og bruke opp minne på en moderne maskin som svært usannsynlig. Et alternativ hadde vært å flytte seksjonen "Initial game options" ut i en egen GUI funksjon som kontrollerte dette. Dersom exit blir valgt i menyen, så vil programkoden kjøres til ende og programmet / funksjonen er helt avsluttet.

UTFORDRINGER

De største utfordringene i denne oppgaven ligger helt klart i plottingen og grunnlaget for plottene. Spesielt streken som illustrerer utholdenhet vinkel. Her ligger ikke utfordringen først å fremst i matten, men i det faktumet at x og y aksene ikke har samme skala. I tillegg for utregningene må en altså kompensere for dette. Slik at spilleren får en lagt bedre opplevelse. Dette er gjerne det beste eksemplet på hvordan dette skiller seg ut som et spill og ikke en ren simulering. Utenom dette var selvfølgelig hver av animasjonene og grunnlagene for disse sine egne utfordringer.

5.2.2 REACTION1

Ansvarlig: Anders Svalestad

HENSIKT

Et spill med hensikt å teste reaksjon. Spiller får en vilkårlig retning (høyre\venstre) presentert i en figur. Retning vises som tekst. Spiller skal så fort som mulig trykke tilsvarende retnings-piltast på tastaturet.

OPPBYGNING

Figurer i Matlab har flere innebygde egenskaper. En av disse er «KeyPressFcn». Blir denne definert, vil figuren «lytte» etter tastetrykk. Jeg har brukte denne egenskapen i spillet for å slippe at bruker skal trykke bokstavgang + Enter.


```
function keyDownListener(src, event)
    resultat=event.Key;
```

Når du starter programmet, vil du få valget om å starte spill, eller avslutte.

Når du trykker start settes en vilkårlig pause mellom 0 og 5 sekunder. Dette for at bruker ikke skal vite når retning kommer.

Vilkårlig retning blir valgt ved bruk av Rand kommando. Retningen blir så presentert for bruker i en figur. Stoppeklokken blir da startet.

Når bruker har trykket korrekt retningspil på tastatur, hopper programmet videre, visker ut figur å presenterer retning og reaksjonstiden i figur. Man har så valget mellom å trykke «space» eller «escape» tast på tastatur.

«Space» vil gi deg muligheten for å teste en gang til, mens «escape» vil ta deg tilbake til GUI meny.

UTFORDRINGER

Program ble først laget med input funksjon. Bruker måtte taste «v» for venstre og «h» for høyre, og avslutte med Enter tast. Brukte en del tid på å finne ut av KeyPress funksjonen som er innebygget i Matlab figurer. Brukte <http://se.mathworks.com/> for å forstå hvordan den virket.

Hadde også utfordringer med feilmeldinger dersom man trykker feil piltast. Løste dette ved å bruke «try» og «catch» blokker. Dersom kode inni en try blokk feiler, hopper programmet til catch blokken og kjører handling. I dette tilfellet ingen handling, men program sjekker igjen for riktig piltast valg.

5.2.3 REACTION2

Ansvarlig: Anders Svalestad

HENSIKT

Et spill med hensikt å teste reaksjon og tastatur, "touch" ferdighet.

Spiller får vilkårlige bokstaver presentert i en figur. Spiller skal så finne og taste vist bokstav så fort som mulig. Antall runder og samlet tid blir presentert ved spill slutt.

OPPBYGNING

Tidsvektor initialiseres

Figurer i Matlab har flere innebygde egenskaper. En av disse er «KeyPressFcn». Blir denne definert, vil figuren «lytte» etter tastetrykk. Jeg har brukte denne egenskapen i spillet for å slippe at bruker skal trykke bokstavg + Enter.

```
function keyDownListener(src, event)
    resultat=event.Key;
    % Hvis bokstav er riktig, Stopp klokke og vis reultat.
    if resultat == bokstav;
        tid(end+1) = toc;
        disp = (['Riktig bokstav. Du brukte: ' num2str(tid(end)) ' sekund']);
        funnetBokstav = 1;
    elseif resultat == 'space'
        close all
        Reaction2()
    end
end
```

```
% Setter opp figur og viser villkårlig bokstav
figure(1)
axis([-2 2 -2 2]);
antallrunder = text(0,0,bokstav);
set(gcf,'name','trykk space for å avbryte')
set(gca,'visible','off');
set(antallrunder, 'fontsize',100);
set(gcf, 'KeyPressFcn', @keyDownListener)
```

En meny vil åpnes å du får velge hvor mange runder du vil kjøre. 3,5 eller 10.

Programmet går inn i en for løkke, som kjører runder basert på antall runder du valgte i meny. Kode velger en villkårlig bokstav, og presenterer den i en figur.

```
liste = 'abcdefghijklmnopqrstuvwxyz';
bokstav = liste(ceil(length(liste)* rand));
```

En stoppeklokke blir startet og figuren venter på at tast fra tastatur.

Kode sjekker så om tastatur input er den samme som vist bokstav. Hvis riktig bokstav blir tid lagret og neste villkårlige bokstav blir vist i figur og sekvensen gjentar seg lik antall runder valgt.

Etter rundene er gjennomført åpnes en figur som forteller deg hvor mange runder du kjørte, og hvor lang tid du brukte.

UTFORDRINGER

I starten ble det brukt «input» kommando. Dette medførte at bruker måtte taste riktig bokstav å deretter Enter tasten. Dette gjenspeiler ikke faktisk reaksjonstid.

Brukte en del tid på å finne ut av KeyPress funksjonen som er innebygget i Matlab figurer.

Brukte <http://se.mathworks.com/> for å forstå hvordan den virket.

KONKLUSJON

Programmet fungerer bra, og mer nøyaktig etter at keypressFcn i figur ble implementert.

5.3 MUSIKK

5.3.1 KEYBOARDPIANO

Ansvarlig: Daniel Lovik

HENSIKT

keyboardPiano er et program som simulerer tangentene til et piano ved hjelp av definerte taster på tastaturet. Tangentene spiller av en tone ved trykk og en figur viser hvilken tangent som spilles av i et annotation vindu.

OPPBYGNING

Programmet er laget slik at det er lett å legge til flere tangenter ved å oppdatere tangent frekvens og tangent navn vektorene:

```
%%      A4 A5 B4 B5  C4  C5  D4  E4  F4  G4 G3
TanFrek = [440,880,493.9,987.7,261.6,783.9,293.7,329.6,349.2,392,196]; % Frekvenser for tangenter i Hz
Tangnavn = {'A4','A5','B4','B5','C4','C5','D4','E4','F4','G4','G3',''}; % Tangent navn.
```

Frekvensene og tangentnavn er hentet i fra wikipedia sin artikkel om piano tangenter og frekvenser.

Frekvensen ganges med en formel som bruker cosinus til frekvensen ganget med π ganget med samplings rate. Formelen ble i utgangspunktet laget med sinus, men etter en del testing ble resultatene av lyden som ble generert bedre ved bruk av cosinus.

```
Fs=10000; % Samplings rate, modifikasjon av verdi medfører endring i tonefall.
t=0:2.1/(Fs):1;
```

```
lyder = []; % Lyder vektor.
for i=1:length(TanFrek) % Teller fra 1 til lengden av vektoren Tanfrek.
    lyder(:,i)=cos(TanFrek(i).*pi*t); % Ganger valgt frekvens med Cosinus formel for å generere tone.
end
```

Figuren leser av tastetrykkene ved hjelp av «KeyPressFcn» og funksjonen «KeyDownListener» oppdaterer lydnummer variabelen:

```
set(figure1, 'KeyPressFcn', @KeyDownListener); % Figuren leser inn taster og funksjonen KeyDownListener utfører
% variabeloppdateringer som resulterer i tone + text på figuren.
%%
% Definerte tangenter koblet opp mot frekvens og sampling multiplikator.
function KeyDownListener(src,event)
    k=event.Key;
    if k == 'a'
        lydnummer(end+1) = 1;
```

I snutten ovenfor registreres tastetrykket «a» og lydnummer settes til 1.

Dette brukes videre i følgende snutt:

```
sound(lyder(:,lydnummer(1))); % Lydnummer fra tastetrykk leses av og rett frekvens hentes inn og tone genereres.
```

Verdien generert av «a» korresponderer til frekvensen 440 Hz i TanFrek vektoren og tangentnavnet «A4» i Tangnavn vektoren.

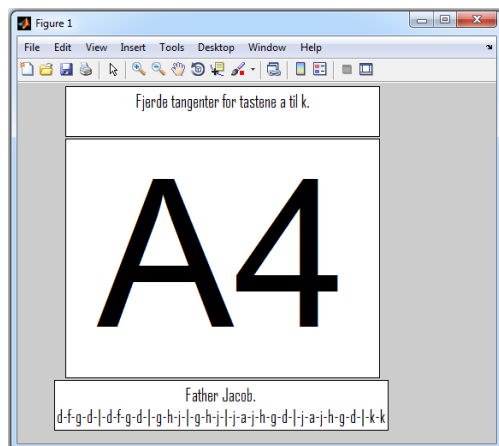
Frekvensen ganges inn med cosinus formelen og tonen blir spilt av.

Tangentnavnet vises på figuren i et annotation vindu.

```
anolyd = char(Tangnavn(lydnummer(1))); % Leser inn hvilken tast som blir trykket og gir det videre til tekstboxen.
```

Samme lydnummer blir brukt for å plukke frem rett tangentnavn fra vektoren og fremvise det i figuren.

Under er en illustrasjon av Figur 10 etter at tasten «a» blitt trykket på:



Figur 10- Aktiv tangent

Melodien for Fader Jakob er lagt med i figuren med rett fjerde tangenter i henhold til denne simulatoren.

Programmet kan avsluttes ved å trykke på «Esc» knappen, det vil da dukke opp en boks med modal funksjon som venter til bruker trykker på ok før programmet lukkes.

UTFORDRINGER

Programmet ble i første omgang laget med en «waitforbuttonpress» funksjon som ventet på et tastetrykk. Når tastetrykket kom ble det konvertert til ASCII for så å bli lest av i en if setning som hadde definerte ASCII verdier for de forskjellige tonene. For tasten «a» er ASCII verdien 97.

Dette fungerte veldig greit i første omgang da det gjorde akkurat det jeg ønsket, å spille av en tone. Problemene oppstod først når en melodi skulle spilles og tastetrykkene kom litt for fort etter hverandre. Resultatet ble at figuren lukket seg fordi det kom for mange inputs. Ikke definerte taster skapte også problemer i programmet ved at feilmeldinger oppstod da disse ikke var definert. Et annet problem her var at figuren måtte vente til tonen for den første tangenter ble ferdig spilt før neste kunne spilles. Dette var hovedgrunnen til at «KeyPressFcn» og «KeyDownListener» ble brukt for logging av definerte tastetrykk.

Figuren lukket seg ikke når det kom mange inputs, den hoppet rett til siste tastetrykk istedenfor å vente på at det forrige var spilt ferdig.

5.3.2 GRAY MUSIC

Ansvarlig: Helge Bjorland

HENSIKT

Programmet bruker lysmåler for å lese gråtone striper separert med hvite striper på ark av valgfri størrelse (Figur 11). Gråtonen på arket representerer en tone og den fysiske lengden av stripen på arket representerer tidslengden hver tone skal spilles.

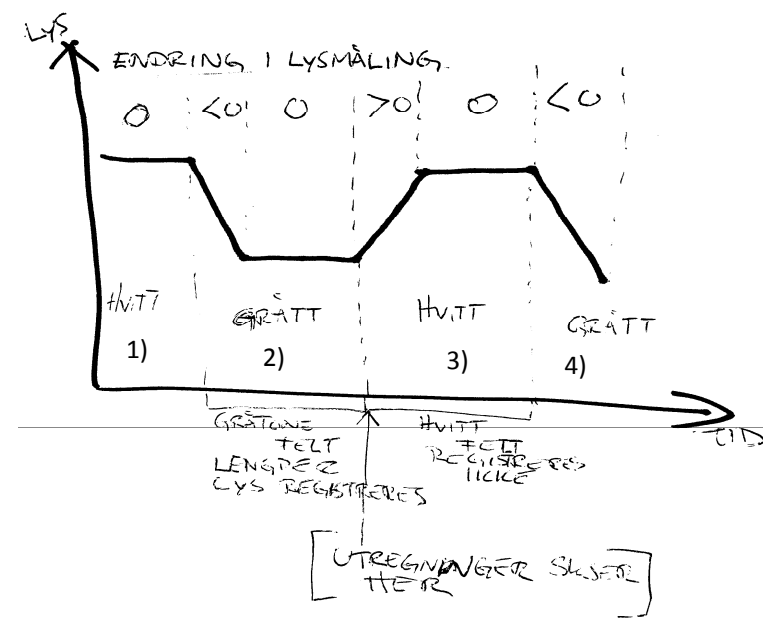


Figur 11 - Gray Music eksempel mønster på ark av valgfri størrelse

OPPBYGNING

Programmet begynner med å hente inn variabler som settes av brukeren. Koden for å innhente dette er samlet i en funksjon og bruker kombinasjon av `menu` funksjonen og `inputdlg`. Først må bruker velge størrelse på arket (A3, A4, A5). Deretter velger en multiplikator for tonelengde om man ønsker lengre toner per felt. Robot fart kan settes mellom 1 og 100 og presisjonen for lysmåling. Sistnevnte setter verdiene i funksjonen som filtrerer lysmålingen.

Det første som ble gjort for å løse denne oppgaven var å tegne en illustrasjon av hvordan algoritmen skulle fungere. Denne er vis i Figur 12 under.



Figur 12 - Graymusic illustrasjon av algoritmen

Illustrasjonen viser en graf av tenkt lysmåling når roboten kjører over arket og registrerer de grå feltene. 1) Her ser en at man begynner med at endringen er null. 2) Når roboten kjører over på et

grått område blir endringen negativ og deretter null til enden av feltet. Det er i denne perioden gråtone og lengde på felt skal registreres. 3) Første gang endringen blir positiv, som betyr at roboten har kjørt til et hvitt område igjen, skal verdiene fra det grå område registreres som en tone basert på lysmåling og lengde basert på fysisk lengde av det grå feltet. Etter første gang endringen ble positiv skal ingenting registreres heller ikke når endringen er null. 4) Ny registrering begynner igjen når endring blir negativ som betyr at et nytt grått felt har startet.

Under er utdrag fra programmet som viser illustrasjonen over konvertert til en algoritme i Matlab kode.

```
while LenRead < TotLen;
    data = motorB.ReadFromNXT();
    lys(end+1)=GetLight(SENSOR_3);
    lysFilt(end+1)=Lfilter([lysFilt(end),lys(end)], RoboPrec);
    deltaLys = lysFilt(end)-lysFilt(end-1);

    if deltaLys < 0;
        templys(end+1) = lysFilt(end);
        lysneg = 1;

    elseif deltaLys == 0;
        if lysneg == 1;
            templys(end+1) = lysFilt(end);
        end
    else
        if lysneg == 1;
            distdeg = motorC.ReadFromNXT();
            Mtone(end+1) = LtoHZ(mean(templys));
            Mlen(end+1)= Deg2Dist(distdeg.Position)*SoundLen;
            motorC.ResetPosition();
            templys = 0;
            lysneg = 0;
        end
    end
    LenRead = Deg2Dist(data.Position);
end
```

Her kan man se hvordan en først sjekker om endring i lysmåling er mindre en null. Da registreres lysmåling i temp variabel og variabelen lysneg settes til 1 for å vise at målingen har gått mot negativt. Neste del av if sjekker om deltaLys er lik null. Om dette stemmer og lysneg er lik null (man er på et grått område) da registreres lysmåling. Om endring er null og lysneg er 0 så registreres ingenting. Da er man på hvitt felt. Om man når siste del av if setning betyr det at endring i lysmåling er positiv og en sjekker da om lysneg er 1. Er den det betyr det at dette er første måling hvor endring er positiv. Da leses avstand fra motoren, gjennomsnitts temp lysmåling blir konvertert til hz og lagret i vektor og lengden på tone blir kalkulert fra avstandsmåling fra motor. Avstandsmåling på motor, templys og lysneg blir nullstilt. Posisjonsmåling fra motor kommer i grader og dette blir konvertert til avstand i millimeter med funksjonen Deg2Dist som vist under. Denne funksjonen regner ut hjulomkretsen og multipliserer denne med input gradene delt på 360.

```
function [out] = Deg2Dist(deg)
    WheelCirc = 56*pi; %56mm
    out = WheelCirc*(deg/360);
end
```

Når roboten er ferdig å kjøre over arket å registrere lysmålinger spilles melodien som er lagret i tone og lengde vektor. Funksjonen som brukes er NXT_Playtone og tonene spilles i en while loop som kjører til lengden av lengde vektoren. Det er i tillegg lagt inn kode slik at roboten danser til musikken. Det gjøres ved at motorene kjører mens det spilles og retning endres ved hver tone og kjøres like

lenge som tonen spiller. While loopen pauses ved hver gjennomkjøring for å gi tonen tid til å spille ferdig. Koden for avspilling er vist under.

```
while i < length(Mlen);
  NXT_PlayTone(Mtone(i),Mlen(i));
  %Do a boogie to the lovely music
  motorB = NXTMotor('B','Power',danceSpeed*danceDirect);
  motorC = NXTMotor('C','Power',danceSpeed*danceDirect*-1);
  motorB.SendToNXT();
  motorC.SendToNXT();
  danceDirect = danceDirect * -1;
  pause((Mlen(i))/1000)
  i=i+1;
end
```

UTFORDRINGER

Det ble litt utfordringer med denne koden i starten da startpunktet først var å bare begynne å kode. Det ble da en veldig spesiell melodi siden alle lysmålinger ble konvertert til toner og spilt. Etter dette var det tilbake til tegnebrettet for å lage en sketsj på hvordan algoritmen skulle oppføre seg. Etter dette fungerte målingsregistrering fint.

En annen utfordring var da melodien skulle spilles av. Her stoppet ikke while loopen for å vente til NXT_PlayTone funksjonen var ferdig å spille. Etter litt testing når det ble klart hva som var problemet var løsningen å legge inn en pause like lenge som tonen spilte. Dette fungerte greit.

En annen justering som ble gjort var på tonemappingen mot gråtone. Siden målingene som regel lå relativt høyt (>400) ble tonene veldig høye når disse ble mappet direkte mot frekvens-båndet til NXT. Det ble derfor testet med å dele resultat frekvens på to noe som ga en mye mer behagelig melodi. Den resulterende funksjonen kan sees under.

```
function [out] = LtoHZ(l)
  x= 13800/1023;
  out = ((x * l)+200)/2;
end
```

5.3.3 JOY MUSIC

Ansvarlig: Helge Bjorland

HENSIKT

Med dette programmet skulle man kunne spille en melodi på roboten med joysticken.

OPPBYGNING

Dette programmet tar input verdier fra joysticken i en while loop, konverterer dette til frekvens verdier som blir matet til NXT_PlayTone funksjonen. Programmet avsluttes når man trykker på hovedknappen. Under er utsnitt fra koden som viser hovedelementet i programmet.

```
if ((joyFB(end)+joyS(end))/2) > 0
  Mtone = LtoHZ((joyFB(end)+joyS(end))/2)
  NXT_PlayTone(Mtone, Mlen);
end
```

5.4 ROBOFUN

5.4.1 CLAP DRIVE

Ansvarlig: Daniel Lovik

HENSIKT

Programmet tar i bruk lydsensoren for å registrere klapping. Et klapp gjør at roboten kjører fremover og 2 klapp fort etter hverandre bremses den. Ultralydsensoren er også tatt i bruk slik at roboten stopper av seg selv når den kommer innenfor en forhåndsdefinert grense fra veggen.

OPPBYGNING

Programmet starter med å initialisere NXT'en, ultralydsensoren og lydsensoren. Diverse variabler for verdi håndtering og løkke håndtering settes i starten. Motorene defineres til å kjøre synkront med retning fremover.

Verdier fra lydsensoren og ultralydsensoren samt tid siden programstart plottes så lenge variabelen Progstart er satt til true.

Henting av lydsensorverdi er satt i en if løkke som aktiveres ved et klapp som genererer en verdi på større enn 600. Roboten vil da kjøre fremover med en motorkraft på 20 %.

```
if ~clapreg % Registrering av klapp + klappetid, sørger for at klapp bare
    % registreres en gang over satt nivå
    clapcount = clapcount +1
    claptime(end+1) = cputime;
    clapreg = true;
end
```

Snutten ovenfor sørger for at klappetelleren bare oppdateres en gang per klapp som overstiger grensen. Det gjør den ved at variabelen clapreg er satt til false i initialiseringen og etter at klappet registreres så settes den til true for å hindre at det telles flere klapp så lenge verdien er over 600 i et viss antall ms.

Følgende snutt viser hvordan to kjappe klapp innenfor 500ms håndteres. Klappeteller øker med 2 og den totale tellerverdien er nå 3. Dette setter motorene i brems.

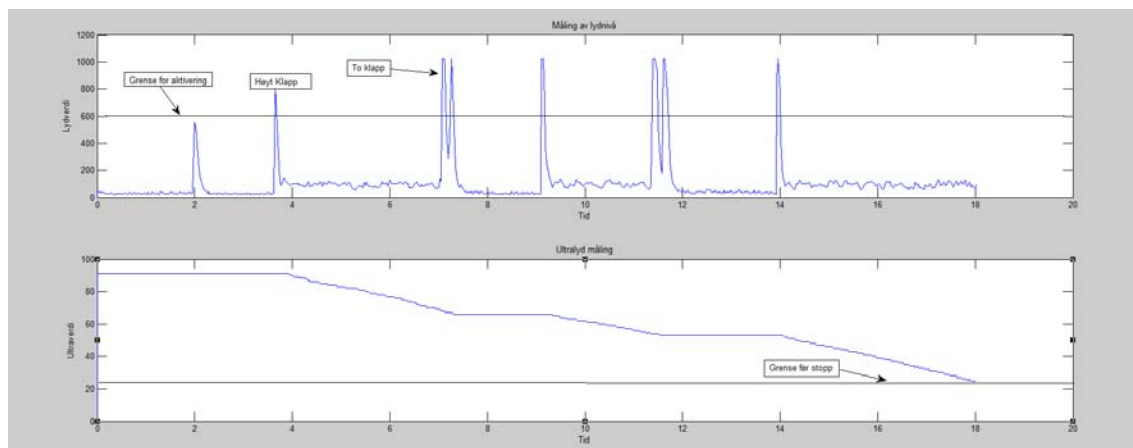
```
%Hvis det registreres to klapp innenfor 500ms øker klappeteller med 2
if claptime(end) - claptime(end-1) < 0.5
    clapcount = clapcount +2
end
```

Ved enda et enkelt klapp så vil roboten igjen kjøre fremover, og klappeteller settes til 1.

Ultralydsensoren overvåker hele tiden at avstanden fra veggen er over 25 cm og hvis den kommer under så stopper motorene rolig opp og Progstart variabelen settes til false. Dette gjør at hovedløkken avsluttes og programmet går videre til nedstenging av NXT sensorer og kommunikasjon.

Figur 13 under viser forskjellige typer klapping. Den første endingen ser ved vi ved et svakere klapp med en peak på ca 500. Dette klappet påvirker ikke roboten på noen måte. Det samme gjelder normal støy fra omgivelsene som ligger på ca 40. Ved klapp med peak over 600 starter roboten å kjøre her ser vi også i ultralydgrafen at den beveger seg nærmere veggen. Ved to kjappe klapp ser vi

at ultralydsensoren jevner seg ut fordi roboten står stille. Videre gjentar kjøre løpet seg helt til den når for nært vegg og stopper opp.



Figur 13- Lyd og Ultralyd graf under kjøring

UTFORDRINGER

Tidlig i kode fasen oppstod det et problem ved bruk av klappetelleren og det var at teller oppdaterte seg hele tiden så lenge ekkoet var over 600. Dvs ved et høyt klapp så ble teller oppdatert gjerne med 11 klapp istedenfor 1 slik den skulle. Dette ble løst ved hjelp av clapreg snutten som ble tipset av en av medlemmene i gruppen. Denne håndterte rett klapp telling og programmet oppførte seg som planlagt.

5.4.2 ULTRA DRIVE

Ansvarlig: Daniel Lovik

HENSIKT

Programmet tar i bruk ultralydsensoren for å kjøre mot en vegg/objekt helt til en forhåndsdefinert avstand er nådd. Jo nærmere roboten kommer hinderet jo tregere vil den kjøre. Når hinderet er innenfor grensen vil roboten stå stille, hvis hinderet beveger seg nærmere roboten vil den rygge for å holde seg til avstandsgrensen. Programmet avsluttes ved at lydsensoren registrerer et høyt rop/klapp.

OPPBYGNING

Programmet starter med å initialisere NXT'en, ultralydsensoren og lydsensoren. Diverse variabler for verdi håndtering og løkke håndtering settes i starten. Motorene defineres til å kjøre synkront fremover/bakover. Det er laget en variabel kalt breakcount som er satt til false. Denne variabelen styrer om roboten får lov til å bevege seg eller om den er i brems modus. Verdier fra ultralydsensor og lydsensor plottes fortløpende sammen med tiden siden programmet startet.

Hastigheten på roboten styres i en if-løkke som er aktiv så lenge roboten er 25 cm i fra hinderet:

```
Motorer.Power = GetUltrasonic(SENSOR_4)-10; % Nåværende avstandsverdi minus 10. Sendes som kraft % til NXT.  
Motorer.SendToNXT();
```

I snutten ovenfor får Motorer pådrag ved at ultralydverdien hele tiden blir trukket fra med 10 og sendt til motorene. Dette gjør at roboten reduserer farten jo nærmere objektet den kommer og øker farten jo lenger vekk den er fra objektet.

For å rygge gjøres følgende:

```
Motorer.Power = GetUltrasonic(SENSOR_4)-30; % Nåværende avstandsverdi minus 30. Sendes som kraft % til NXT.  
Motorer.SendToNXT();
```

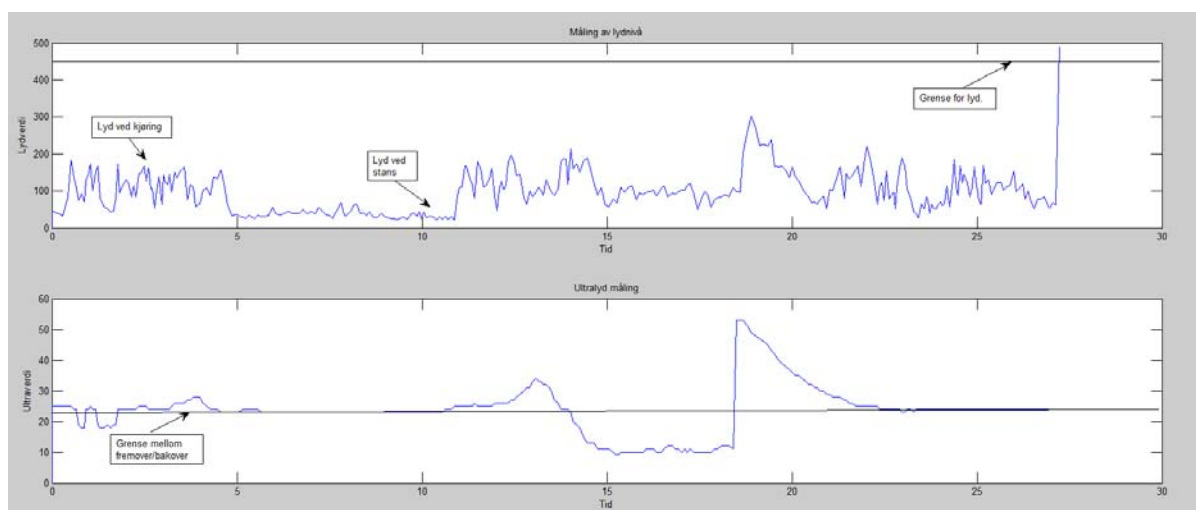
Når objektet er under satt grense område på 23-25 cm så rygger roboten. Det er satt en minus verdi på 30 her for å få roboten til en negativ verdi ut fra grense området.

Er roboten innenfor grenseområdet vil den sette motorene i brems. Dette gjøres ved hjelp av en enkel else og ('brake') kommando til NXT.

Roboten kan når som helst stoppes ved å rope/klappe høyt. Når lydsensoren registrerer dette så settes motorene til stans, breakcount settes til true og hoved løkken stenges. Dette gjør at programmet fortsetter til NXT shutdown prosedyren som lukker sensorene og tilkoblingen.

Under er et utsnitt av grafen under en 30 sekunders kjøreseanse. Det kommer tydelig frem at roboten står stille i grenseområdet ved den markerte streken på ultra grafen. Videre kjører roboten litt frem før objektet kommer under grenseområdet og roboten rygger før den går fremover igjen.

Under kjøringen generes det litt støy og det kommer frem på den øverste lydgrafen. I perioden der roboten var i ro så sank støynivået. Med en gang det begynte å kjøre så økte det. Terskelen for å stoppe roboten er markert med en strek øverst på lydgrafen og her ble den aktivert da lydsensoren registrerte et høyt rop.



Figur 14 - Grafer under kjøring med Ultralydsensor

UTFORDRINGER

Programmet ble i første omgang laget ved å bruke lyssensoren rettet mot en svart lommebok. Dette fungerte fint i første omgang, men når programmet skulle utvides og hastighets reguleres oppstod det store problemer. Lyssensoren var ikke god nok til å oppfatte objektene og miljøet rundt påvirket resultatene i stor grad. Med tips fra veilederen ble sensoren for avstandsregistrering endret til ultralyd. Her ble det mye lettere å kontrollere roboten ved å bruke en vegg eller bok som avlesnings objekt.

5.4.3 ULTRA WALL

Ansvarlig: Daniel Lovik

HENSIKT

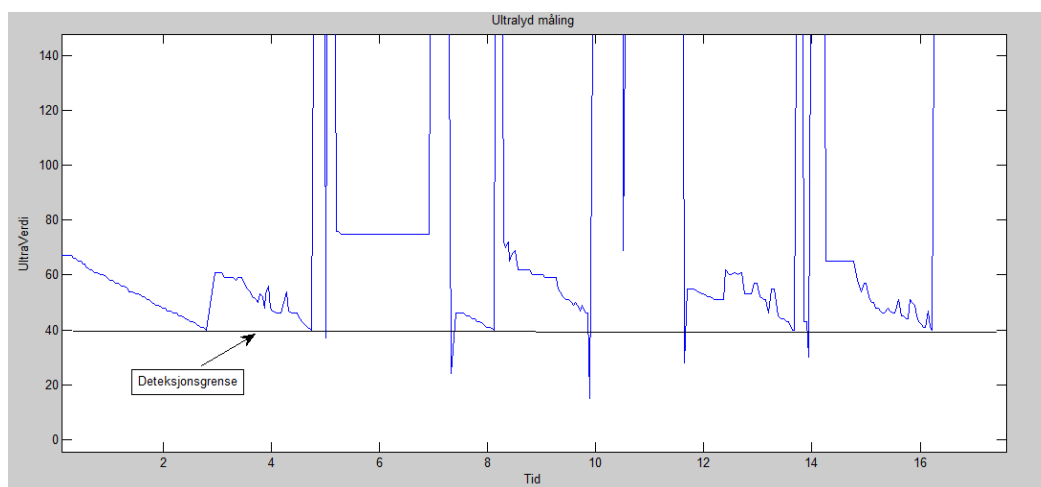
Programmet tar i bruk ultralydsensoren til NXT'en for å finne en hindring, for eksempel en vegg eller person. Når sensoren finner hindringen innenfor angitt avstand vil roboten snu 90 grader og kjøre videre. For å stoppe programmet brukes lydsensoren og et høyt klapp.

OPPBYGNING

Programmet starter med å initialisere NXT'en, ultralydsensoren og lydsensoren. Diverse variabler for verdi håndtering og løkke håndtering settes i starten. Motorene defineres, en variabel for lik pådrag på begge motorene og en for venstremotor som skal brukes til omdreining. Under omdreining settes det en kraft på 40 % av maksimal motorkraft på venstre motor og en tacholimit på 360 grader. Tacholimiten styrer hvor mange grader hjulet skal spinne.

Programmet kjører så lenge variabelen ProgStart er satt til true. Det er her hoved While løkken kjører. Her finner vi blant annet håndtering og logging av verdier til grafen samt logging av tid.

Videre er det satt opp enda en While løkke som bare er aktiv når ultralydsensoren registrerer et hinder innenfor 40 cm. Siden ultralydsensoren er litt tregere enn de andre sensorene tar det litt tid før den registrerer hinderet, dette ser man i grafen(Figur 15) under ved deteksjonsgrensen:



Figur 15- Graf for ultralyd under kjøring og omdreining

Når ultralydsensoren er under deteksjonsgrensen stopper fremover kjøringen og venstre motor gir nok kraft med forhåndsdefinert power og Tacholimit til å foreta en 90graders omdreining.

Så fortsetter programmet med en if løkke som sier at så lenge avstanden er større enn 40 cm så kjører roboten fremover med 30 % kraft.

Det er også lagt inn en «Stoppbryter», dette er en if løkke som registrerer lydnivå. Hvis lydsensoren hører en lyd tilsvarende et høyt klapp så skrus motorene rolig av og hoved While løkken endres til false. Programmet fortsetter videre etter While løkken og her stenges NXT ned.

UTFORDRINGER

Underveis var det en del utfordringer med hvordan ultralydsensoren registrerte hindringer og hvor lang tid det tok før disse ble registrert. Her var eksempel programmet «NextGenerationUltrasound» som lå med i prosjektfilene vi fikk utdelt til stor hjelp. Under kjøring av roboten så kom USB kabelen litt i veien til tider, dette førte til at roboten ikke alltid fikk til å snu seg helt korrekt. Her kunne en blåtann enhet vært til hjelp.

5.5 MATEMATISKE FUNKSJONER OG PLOTING

5.5.1 PLOT TIMER

Ansvarlig: Espen Ro Eliassen

HENSIKT

Programmet skal illustrere tre forskjellige måter å utføre plotting og presentere disse i tre sub plott i en figur i sann tid. Deretter skal et sammendrag vises i en slutt figur.

OPPBYGNING

Det startes ved initialisering av figuren, som sørger for en grei plassering og størrelse ut fra skjermens størrelse. Det registreres også en "keyDownListener" som figurens "KeyPressFcn". Denne defineres som en under funksjon og skal brukes for å gå videre i programmet.

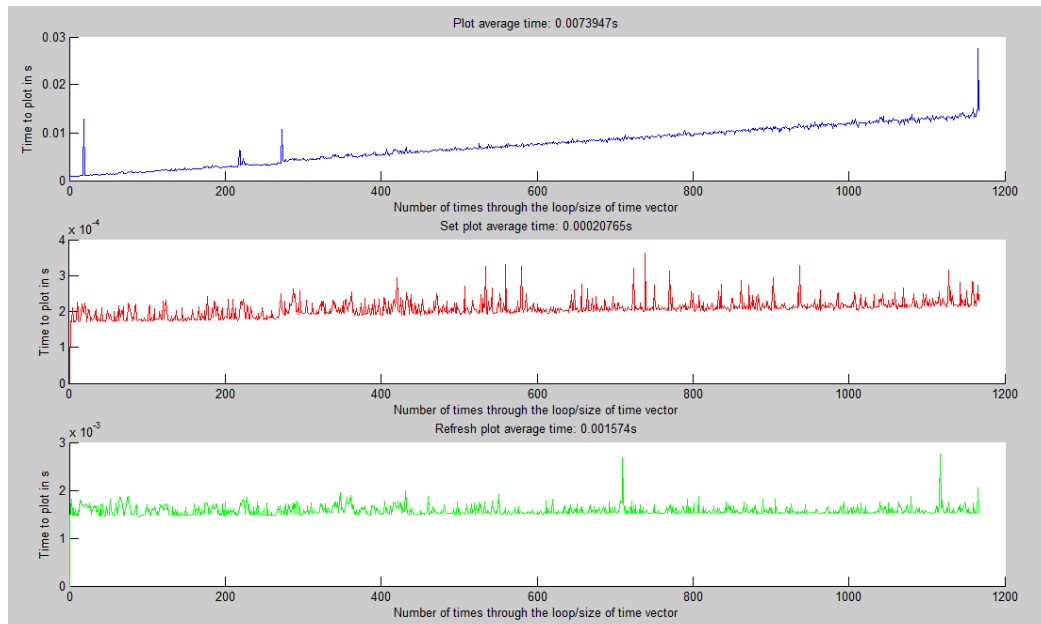
Deretter initialiseres programspesifikke data. Dette er vektorer for tid og gjennomsnitts tid for de forskjellige måtene å plote på. En vektor for antall ganger gjennom hoved løkka og variabel for styring av hoved løkka.

Neste steg er å initialisere plottene definere egne håndtak til hvert av plottene og gi aksene tekst. Plot(x,y) funksjonen gis også et håndtak, selv om det ikke benyttes til noe. Avsluttende i denne seksjonen tvinger en programmet til å tegne opp plottene.

Hoved løkka i programmet er neste seksjon, og stoppe klokka startes rett før en går inn i løkka. Deretter går en inn i hvert sub plott tar tiden rett før og rett etter hvert plot og lagrer det i den tilsvarende vektoren. En regner så gjennomsnittstid for den bestemte typen plott og viser den gjennomsnittstiden som gjelder for denne gjennomkjøringen i tittelen til plottet. Dette gjøres for alle tre plott. Rett før løkka avsluttes oppdaterer en hvor mange ganger løkken har blitt kjørt gjennom og tvinger grafikken til å oppdatere seg. Løkka avsluttes ved å trykke en vilkårlig knapp.

Når en er ferdig å se sanntidspresentasjonen skal programmet vise en oppsummerings figur. Nåværende figur renses og det defineres felles aksegrenser for alle 3 plottene. I dette oppsummeringsvinduet vises grafen for gjennomsnittstidene til de forskjellige plott metodene i tilsvarende sub plott. Dette gjøres alle ved bruk av plot(x,y) funksjonen da målingene nå er over. Til slutt lar en programmet gå i en løkke som tvinger grafikk oppdatering frem til bruker ønsker å gå videre ved å trykke hvilken som helst knapp.

Når programmet anses som ferdig stenger en nåværende figur.



Figur 16 - Plot 1 viser `plot(x,y)` funksjonen, plot 2 viser hvordan x- og y data settes direkte via håndtaket til plottet og plot 3 viser oppdatering av plott via kildedata.

UTFORDRINGER

Den eneste reelle utfordringen i dette programmet ligger i å forstå forskjellige måter en kan plote i matlab.

5.5.2 MATH SHOW

Ansvarlig: Espen Ro Eliassen

HENSIKT

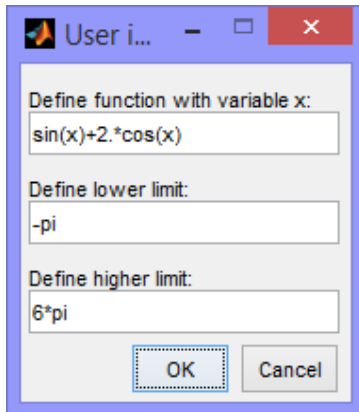
Lage et program som simulerer og illustrerer numerisk derivasjon og integrasjon. Programmet skal gjøre dette ved bruk av en matematisk funksjon og tiden det tar å kjøre en gang gjennom hovedløkken til programmet.

OPPBYGNING

Funksjonen `mathShowMain` er et initielt brukergrensesnitt for å velge om en ønsker å derivere eller integrere. Den kjører en løkke og kaller hver av de spesifikke funksjonene for seg. Noe som gjør at disse kan avslutte når de er ferdig fremfor og kalle seg selv for å holde programmet gående (denne metoden ble vist i `cannonGame` programmet). Se siste paragraf under oppbygning for `cannonGame` programmet.

mathIntShow-funksjonen viser numerisk integrasjon med 5 plot. 1: Selve funksjonsgrafen, 2: den nåværende løkkens bidrag til integralet, 3: integralet fra nedre grense til den nåværende løkken, 4: integralet for hele det definerte området og 5: grafen for funksjonen til integralet.

Funksjonen starter med at bruker angir funksjon, nedre og øvre grense. Dette forutsetter at bruker vet hvordan en skriver matematiske funksjoner i matlab. Inndata fra bruker lagres så til 3 variabler som strenger. Det er viktig at brukeren benytter x som variabel i funksjonen.



Deretter settes figuren dynamisk ut fra skjermstørrelse og en keyDownListener funksjon registreres som figurens "KeyPressFcn". Se cannongame for mer om dette. Instruks for å gå videre i programmet settes i figur navn, trykk hvilken som helst knapp for å få videre. Hele denne initialiseringen er gjort i en underfunksjon.

Forutenom kallet av intialiserings funksjonen for figuren er hele hovedprogrammet satt i en try/catch logikk. Dette fordi programmet er ikke satt opp til å håndtere komplekse tall eller grenser på 0, og dette lar programmet unngå å krasje helt.

Variablene som kommer inn fra bruker er på streng form, for grensene må denne konverteres til tall før de brukes videre. Deretter sjekkes det at øvre og nedre grenser er definert riktig i forhold til hverandre, om ikke så rettes dette.

Det lages en initiel x vektor fra nedre til øvre grense som øker med 0.001 pr x. Om dette gir en vektor med mer enn 10000 elementer økes dette hoppet fra pr x til vektoren har fått en størrelse som er 10000 elementer eller mindre. Deretter evalueres strengen som bruker har angitt som funksjon for å skape en y vektor basert på x vektoren. Felles grense for aksene på plottene defineres og integralet summeres for hele det definerte området.

Underplott 1 og 4 er faste hele tiden og settes nå opp ved bruk av plot(x,y) og bar(x,y) funksjonene. Restene av plottene initialiseres så langt de kan.

Før hovedløkken kjøres i gang settes x til å kun inneholde nedre grense som verdi. Det defineres en maxTime variabel som er hvor mange sekunder hele området skal være. En teller i, samt 0 verdier i andre variabler/vektorer som skal benyttes. Til slutt startes stoppeklokken.

Inne i selve hoved løkken leses først stoppeklokken av. Så lenge denne er under angitt maks tid, så legges den til som en ny verdi på slutten av x vektoren. Ved første gjennomkjøring av løkken vil alle vektorer ha 2 elementer dette er viktig for å kunne tegne rektanglene som integralene utgjør.

For hver gang gjennom løkken evalueres inn strengen fra bruker for å danne en y vektor som stemmer overens med x vektoren. Siden koden alltid vil ta litt tid å kjøre gjennom vil det alltid være

en endring i tid og det er denne endringen som benyttes i numerisk integrasjon. Det er ønskelig at denne endringen skal bli så liten som mulig.

Rektanglene som integralet utgjør tegnes ved å bruke fill funksjonen. Her skal linjene som lages av x, y punktene utgjøre et rektangel, altså må det være 4 punkter. For X blir dette forrige x , denne x , denne x og forrige x . Mens for y blir det $0, 0, y$ nå og y nå. Funksjonen til integralet beregnes ved å legge til summen av den nåværende løkkens integral til summen av integralet så langt.

Når tiden overstiger området nullstilles de grafene som skal det, og x verdien settes til nedre grense igjen. Til slutt resettes stoppeklokken.

Selve hoved løkken avsluttes med å tvinge grafikk oppdatering.

Programmet fanger feil som produseres av logikken i inne i try. Da viser den en feilmelding om feil med funksjon eller grenser. Dette er bare en standard feilmelding, og ikke en nøyaktig tilbakemelding på hva som gikk galt. Det lar bruker tenke et sekund så avslutter den og går tilbake til hoved brukergrensesnitt.

mathDerShow-funksjonen har nesten identisk logikk og kode som integral funksjonen. Plottene som vises her er. 1: selve funksjonsgrafen, 2: grafen for den deriverte fra nedre grense til nåværende løkke gjennomkjøring, 3: nåværende løkkes lineær approksimasjon og 4: alle lineær approksimasjoner.

For uten de matematiske forskjellene på de forskjellige plottene har funksjonene en forskjell i logikk. I istedenfor å benytte seg av en maks tid variabel så gjøres regnes tiden rett inn i x variablene og sammenligningen blir gjort direkte på den.

UTFORDRINGER

Det er to hoved problemer en her står ovenfor. På den ene siden må en forstå grunnlaget for numerisk integrasjon og derivasjon, regneteknisk. Når en har forstått dette, ligger utfordringen i å illustrere det. Fordi spesielt integrasjon representerer ikke bare en graf, men også rektangler/søyler.

5.5.3 NUM JOY

Ansvarlig: Espen Ro Eliassen

HENSIKT

Bruke tankemåten og logikken for hvordan math show simulerte og animerte en matematisk funksjon for og numerisk derivere og integrere en funksjon skapt av en joystick. I dette tilfellet en "xbox 360 controller". Skrive koden på en måte slik at joystick inputen enkelt kan erstattes med annen type input om ønskelig.

OPPBYGNING

Dette programmet bruker de samme matematiske prinsippene og logikken som i mathShow. Programmet har 8 plott som er. 1: Funksjons grafen, 2: nåværende løkke gjennomkjørings integral, 3: summen av integralene til nå, 4: funksjonen til integralet, 5: den deriverte funksjonen, 6: den

nåværende lineær approksimasjonen, 7: alle lineær approksimasjoner og 8: forandringen i tid fra forrige løkkegjennomkjøring til denne.

Koden i seg selv er noe mer strukturert enn i mathshow og variabler har blitt navngitt på en måte at en skal kunne benytte andre eksterne inndata og det fortsatt gir mening. Innhenting av den eksterne dataen er også flyttet ut til en egen funksjon `getJoy`. Det er kun et sted denne brukes og den kan enkelt skiftes til en annen funksjon om en ønsker annen inndata. All utregning av vektor verdier skjer i starten av hver løkke gjennomkjøring og så oppdateres alle plott.

UTFORDRINGER

Ta høyde for store unøyaktigheter i inngangssignalet fra joystick uten å la det dominere programmet. Ut over dette ligger de samme utfordringene i grunn som for simulering av matematisk funksjon.

KONKLUSJON

Programmet viser at logikken benyttet i simuleringen `mathShow` også virker på reelle eksterne inndata, her simulert av en joystick.

6 KONKLUSJON

6.1 MANUELL KJØRING

Problemstilling: Få roboten til å kjøre med joystick på bane med gråtone felt hvor lyssensor måler endring fra nullpunkt. Vi løste dette ved å la filtrerte joystick verdier styre robotens motorpådrag. Lysmåling med filtrering, integrering og derivering av denne blir håndtert i en while løkke som kjører til brukeren trykker på joystick knapp. Grafer blir tegnet mens man kjører og et mer detaljert sett blir tegnet når man avslutter kjøring. Alt så ut til å fungere slik det var tiltenkt.

6.2 AUTOMATISK KJØRING

Problemstilling: Få roboten til å kjøre med automatisk rundt baneprofil med gråtone felt hvor lyssensor måler endring fra nullpunkt. Dette ble løst ved å la avviket fra nullpunktet justere robotens motorpådrag.

Roboten kjørte banen å stoppet ved bane slutt.

6.3 TILLEGG TIL OBLIGATORISK DEL

Problemstilling: Vi ønsket å forbedre hoved program koden. Dette oppnådde vi ved å lage funksjoner for flere av delene, handles i grafer, fartsmåling, grafisk retningsvisning, nullstilling av joystick og tidsmåling. Alt dette fikk vi til å fungere på en tilfredsstillende måte.

6.4 GRAFISK BRUKERGRENSESNITT (GUI)

Vi trengte en måte å navigere mellom alle de forskjellige programmene og laget derfor en GUI som håndterte dette. Vi valgte å bruke innebygget Matlab funksjon menu for å lage meny sammen med switch/case rutine i en while loop.

6.5 CANNON GAME

Programmet er skrevet som et spill og derfor er noe av logikken tilpasset at det skal være et spill. F.eks. er animeringstiden av skuddene økt slik at spillere skal få en følelse av at de skyter noe virkelig. Samtidig er mange hensyns som en ville tatt i design av et slikt prosjekt utelatt, både av tids og relevans messige årsaker. Logikken i form av dekomponering av hastigheter gir samme resultat som regneregler for strekning. Programmet fungerer slik det var tiltenkt.

6.6 REACTION1

Programmet fungerer bra, og mer nøyaktig etter at keypressFcn for figur ble implementert.

6.7 REACTION2

Få figuren til og registrere tastatur trykk uten å måtte bekrefte med Enter tast.

Dette ble løst ved å aktivere KeyPressFcn som er innebygget i matlab figurer. Spillet virker til sin hensikt

6.8 KEYBOARDPIANO

Programmet virket etter hensikt og resultatet ble tilfredsstillende med tanke på tonefall og funksjonaliteten til «Pianoet». En videreutvikling av programmet til å støtte alle 88 tangenter og en mer piano lignende visualisering av figuren kunne vært en ide for et fremtidig prosjekt.

6.9 GRAY MUSIC

Problemstilling: Lag et program som kan lese gråtone striper (skilt med hvite striper) fra et ark. Alle variabler som blir brukt i koden blir hentet fra bruker via inputdlg og menu. Roboten kjører deretter over og registrerer gråtonen i feltene og den fysiske lengden av feltene. Gråtonen blir brukt for å representere en tone og den fysiske lengden representerer lengden på tonen. Melodien blir spilt av når roboten er ferdig å kjøre. Roboten tar seg også en dans i takt med melodien som blir spilt. Programmet fungerte til sin hensikt.

6.10 JOY MUSIC

Programmet fungerer slik det var tiltenkt men det var ikke lett å spille noen vakre melodier med joysticken. Her kunne man nok bygd på å kombinert med skytespill for eksempel, hvor man brukte joysticken for å sikte på toner på skjermen og spille på denne måten. Konklusjonen er vel at joysticken ikke er et veldig egnet input verktøy for å spille melodier.

6.11 CLAP DRIVE

Programmet virket etter hensikt når alt av klappetelling og lydhåndtering var på plass. Roboten reagerte kvikt på klappingen og ultralydsensoren stoppet roboten pent og rolig når den kom innenfor grenseområdet fra veggen. En mulig utvidelse her kunne vært å implementere flere retninger å kjøre basert på antall klapp/variasjon i klappe frekvens.

6.12 ULTRA DRIVE

Programmet virket etter hensikt etter noen modifikasjoner på opprinnelig kode. Roboten reduserte farten veldig fint ved å trekke fra 10 på nåværende ultralydverdi og stoppet pent opp der den skulle.

En mulig videreutvikling av programmet kunne vært å lage en slags labyrint med flere vegger som roboten leste av og beveget seg rundt.

6.13 ULTRA WALL

Programmet virket etter hensikt, med noen små utfordringer rundt USB kabelens forstyrrelser. En mulig videreutvikling med flere retninger å omdreie roboten samt varierende grad av omdreining basert på hinder ble utprøvd, men gav ikke forventede resultater. Kunne vært en ide å forske videre på.

6.14 PLOT TIMER

Det kommer helt klart å tydelig frem at å sette x- og ydata direkte via håndtak i et oppdaterende plott er minst resurskrevende/tar kortest tid. Oppdatering av x- og ydatakilde fungerer også bra, mens å bruke plot(x,y) funksjonen om og om igjen blir mer og mer ressurskrevende jo større vektorene blir.

6.15 MATH SHOW

Programmet gir en god visualisering av noe av hva derivasjon og integrasjon faktisk er. Det gir også en klar illustrasjon av at løkkesid har betydning, spesielt kommer dette godt frem for integralet hvor en kan se at rektanglene for hver løkke gjennomkjørings integral ikke er like store.

6.16 NUMJOY

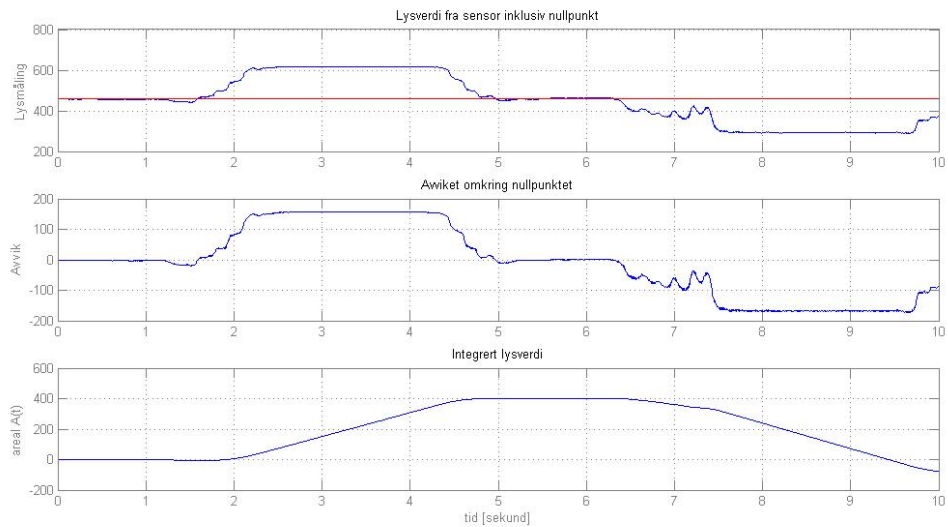
Programmet viser at logikken benyttet i simuleringen mathShow også virker på reelle eksterne inndata, her simulert av en joystick.

7 REFERANSER

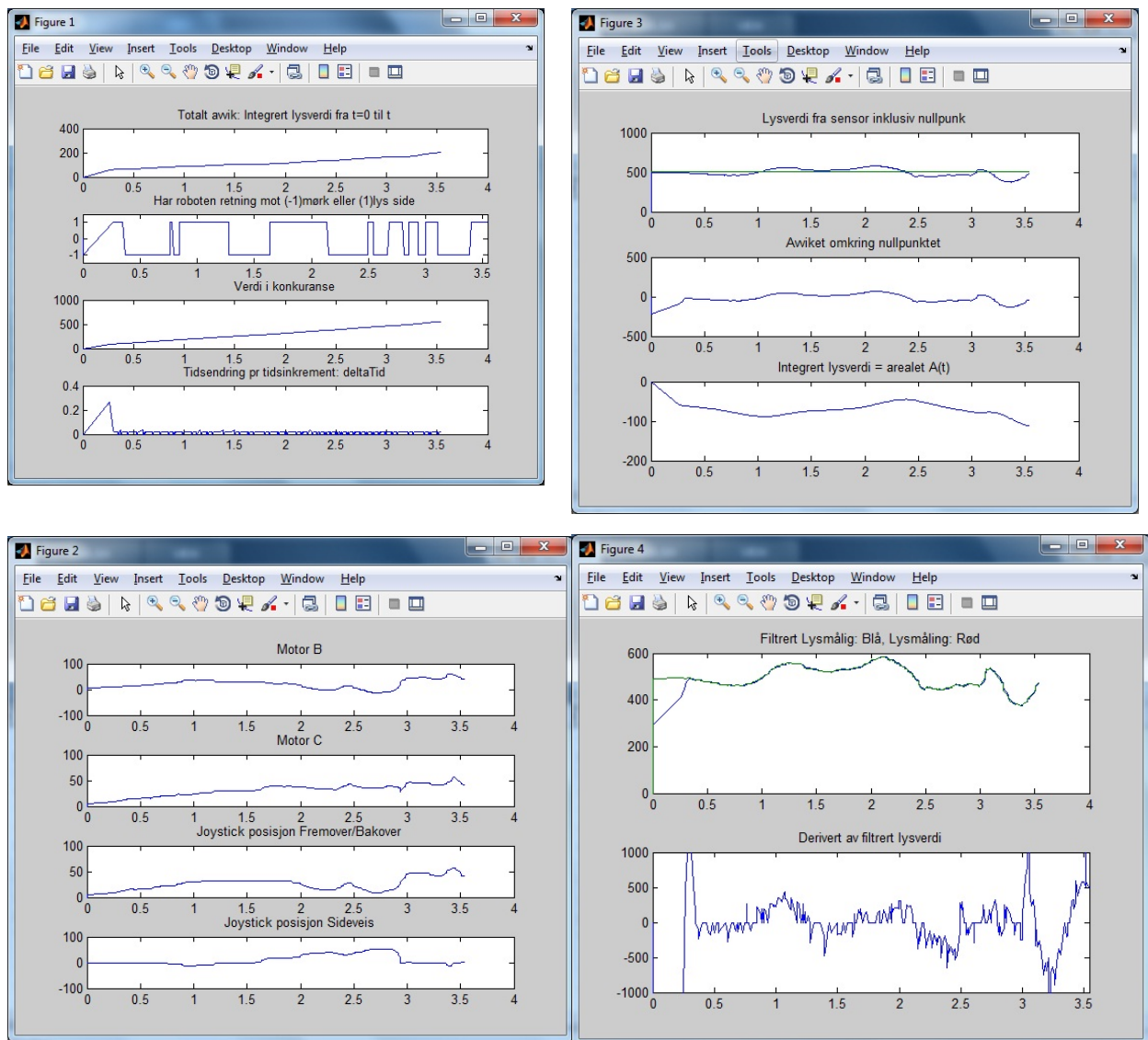
- [1] www.mathworks.se
- [2] <http://www.mindstorms.rwth-aachen.de/>
- [3] <http://www.mathworks.se/matlabcentral/fileexchange/26509-musical-notes>
- [4] https://en.wikipedia.org/wiki/Piano_key_frequencies
- [5] http://tabnabber.com/view_Tab.asp?tabID=Childrens+song&sName=Vader+Jacob

8 APPENDIKS A - ILLUSTRASJONER

8.1 OBLIGATORISK DEL

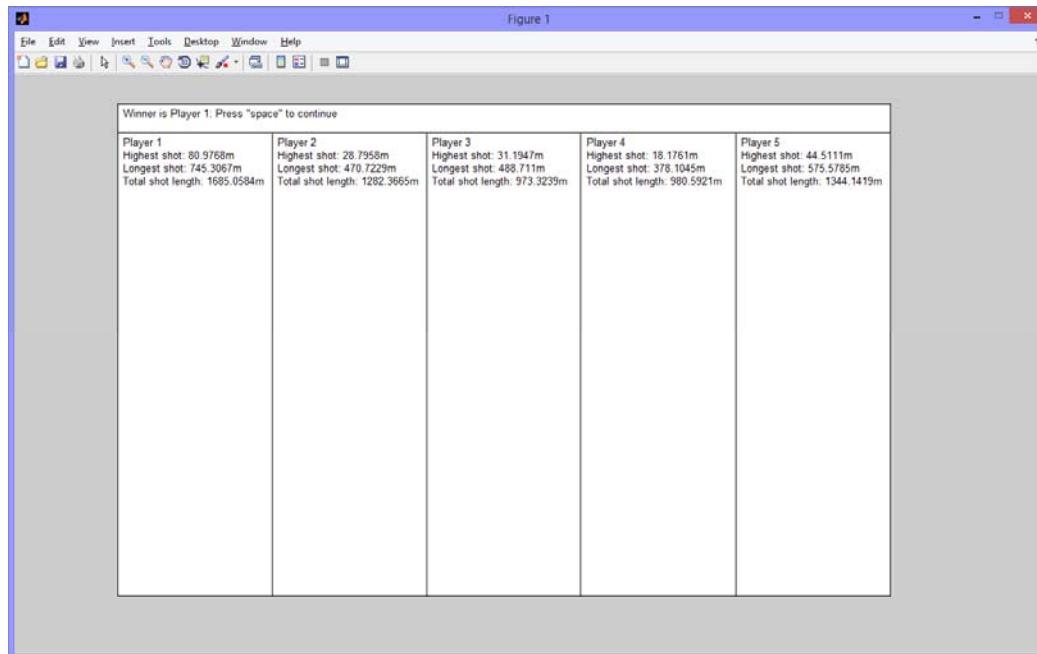


Figur 17 - Integrasjons verifisering alle grafer



Figur 18 - Figurer som plottes etter endt kjøring av main program

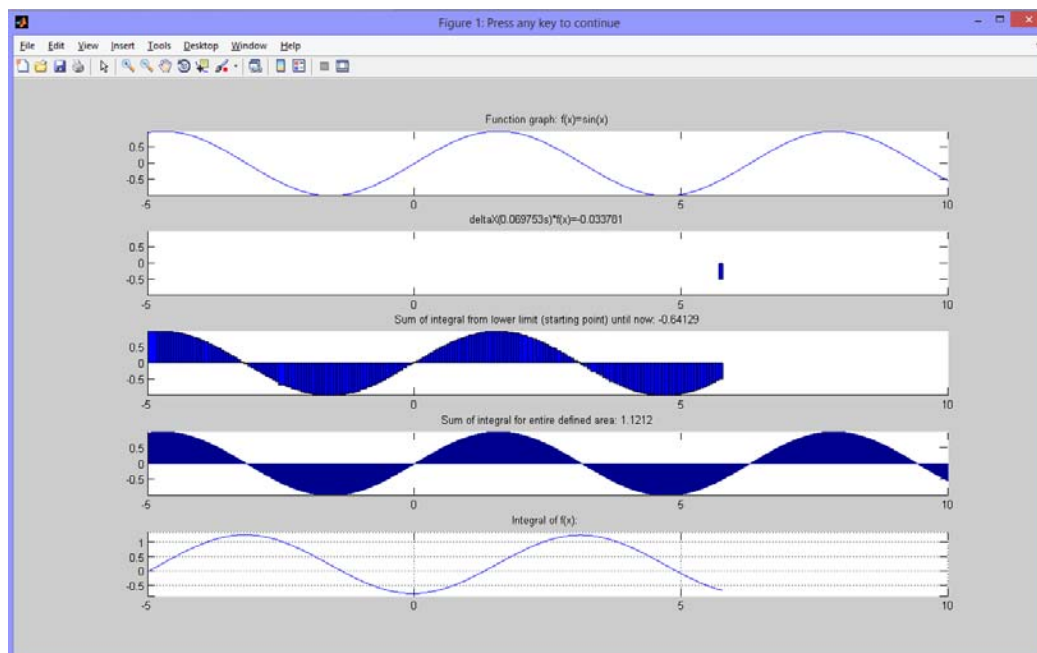
8.2 CANNON GAME



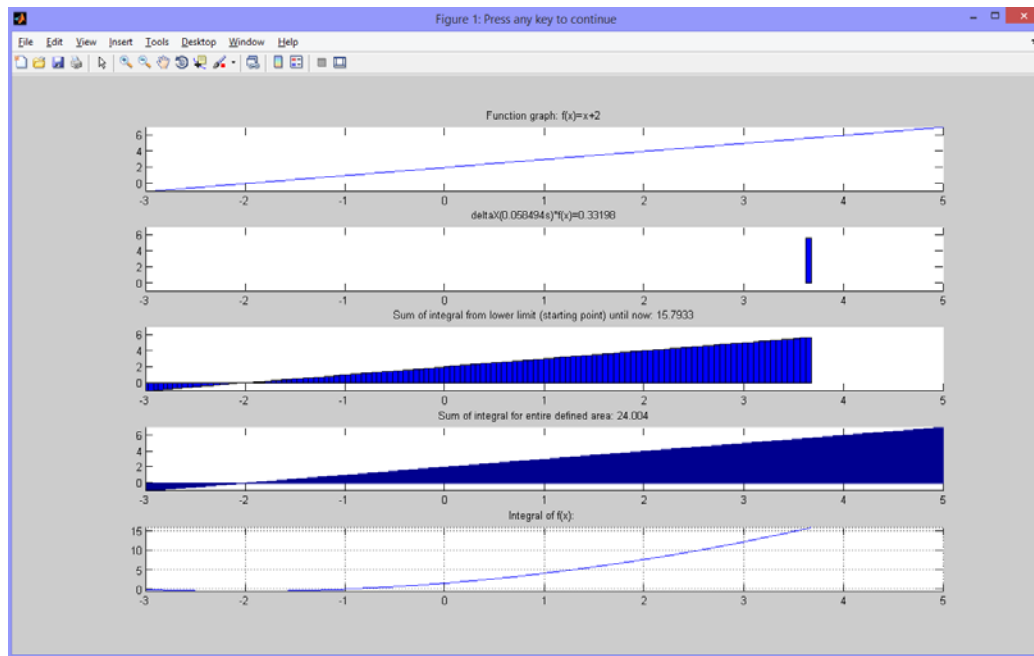
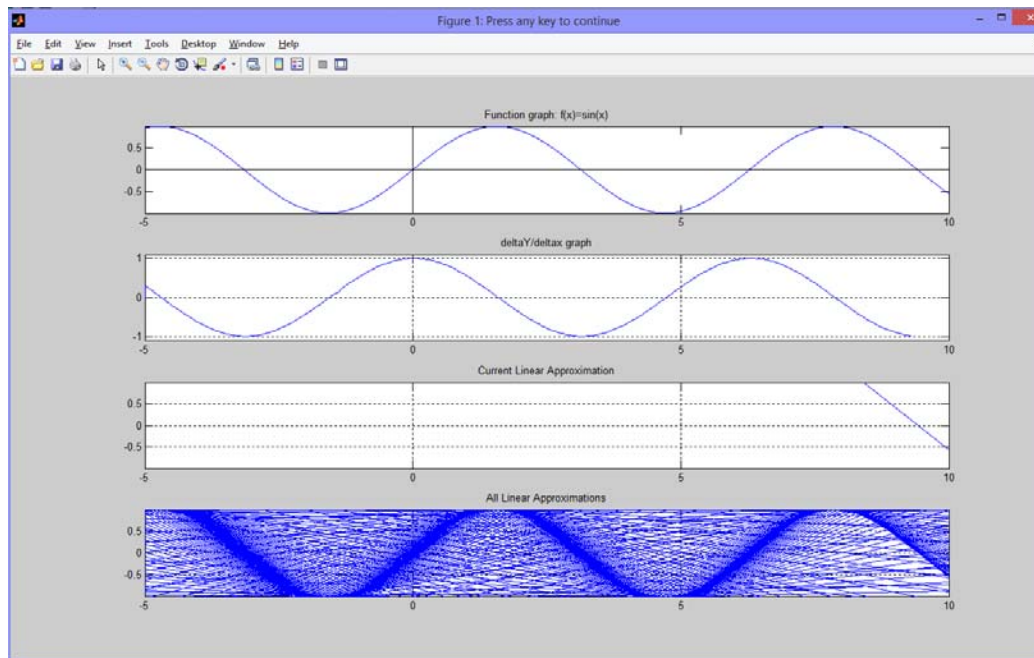
Winner is Player 1. Press "space" to continue				
Player 1 Highest shot: 80.9768m Longest shot: 745.3067m Total shot length: 1685.0584m	Player 2 Highest shot: 28.7958m Longest shot: 470.7229m Total shot length: 1282.3665m	Player 3 Highest shot: 31.1947m Longest shot: 488.711m Total shot length: 973.3239m	Player 4 Highest shot: 18.1761m Longest shot: 378.1045m Total shot length: 980.5921m	Player 5 Highest shot: 44.5111m Longest shot: 575.5785m Total shot length: 1344.1419m

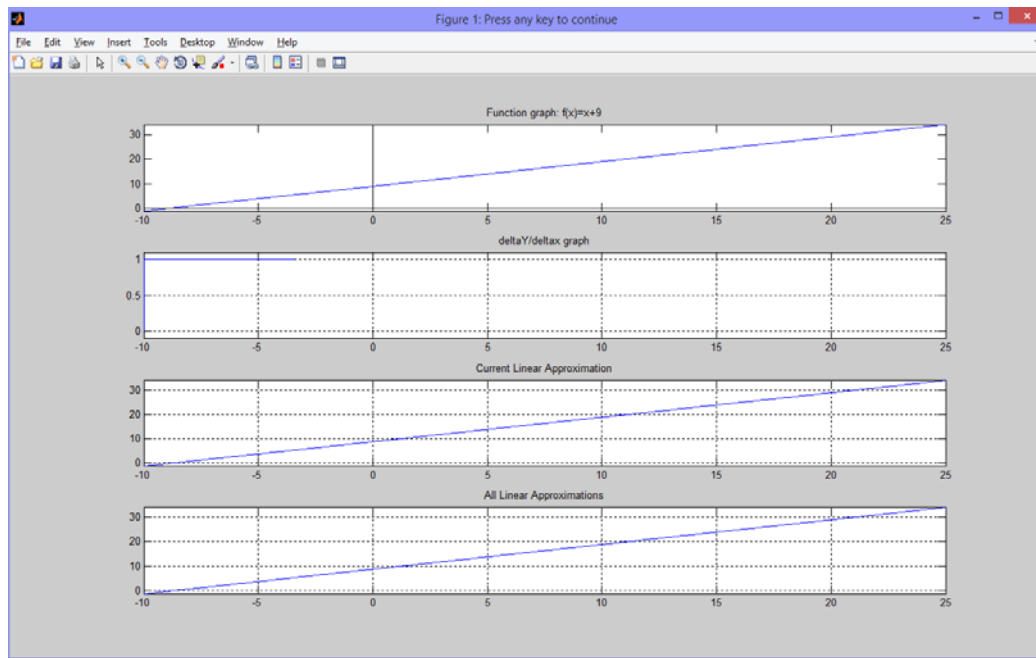
Figur 19 - Cannon Game Winner table

8.3 MATHSHOW

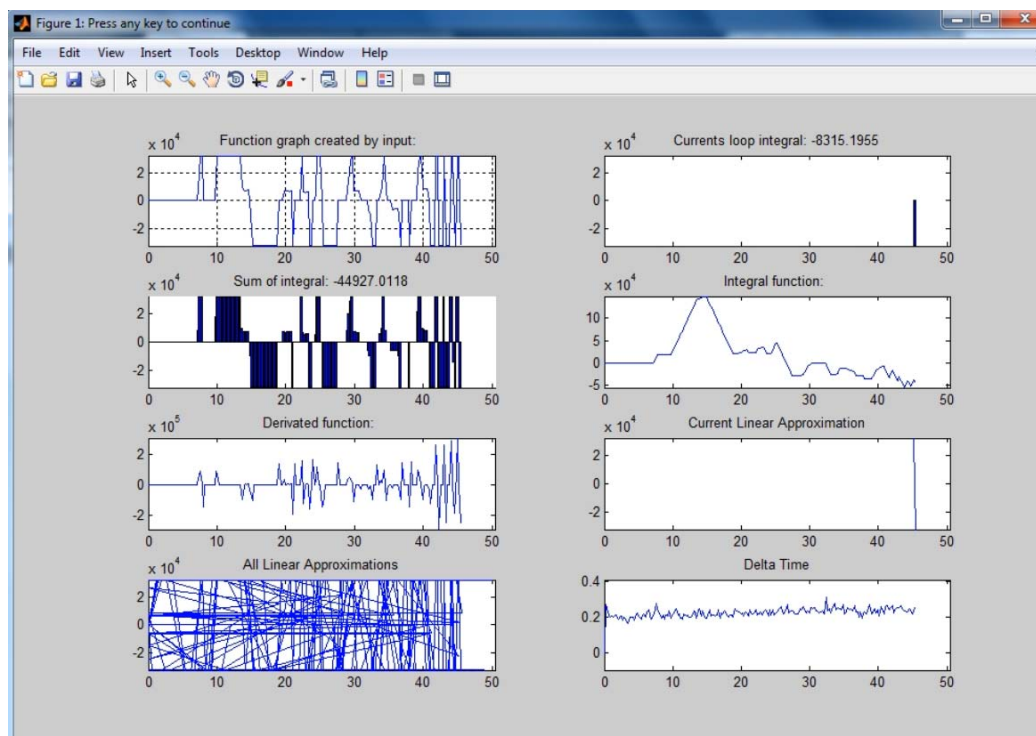


Figur 20 - Numerisk integrasjon av $\sin(x)$

Figur 21 - Numerisk integrasjon av $x+2$ Figur 22 - Numerisk derivasjon av $\sin(x)$

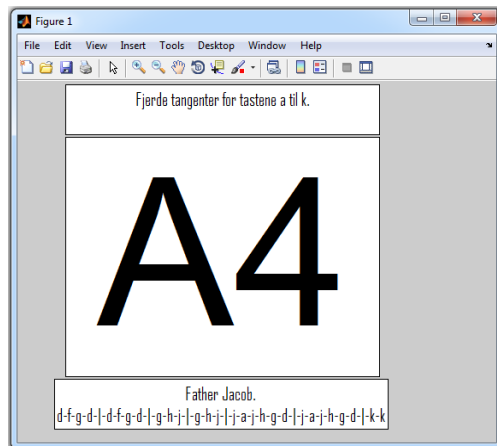
Figur 23 - Nummerisk derivasjon av $x+9$

8.4 NumJoy



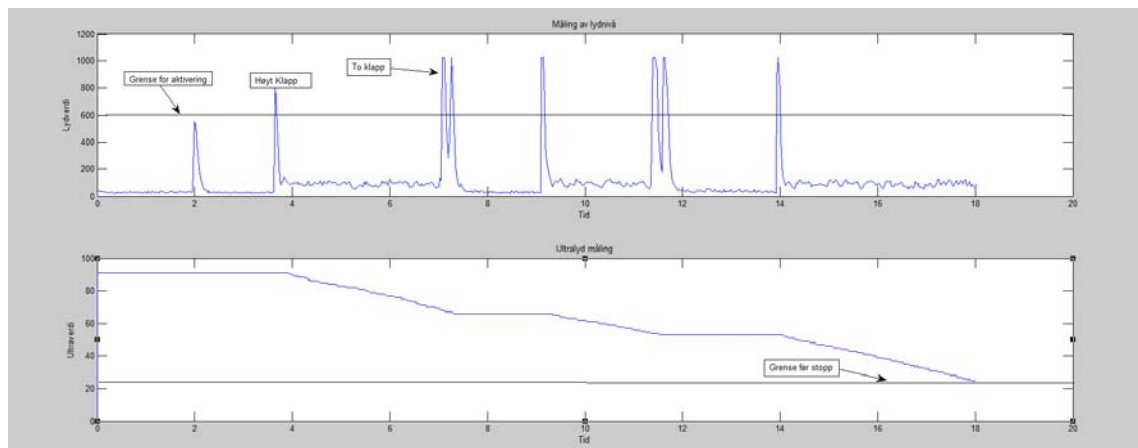
Figur 24 - NumJoy plot

8.5 KEYBOARDPIANO



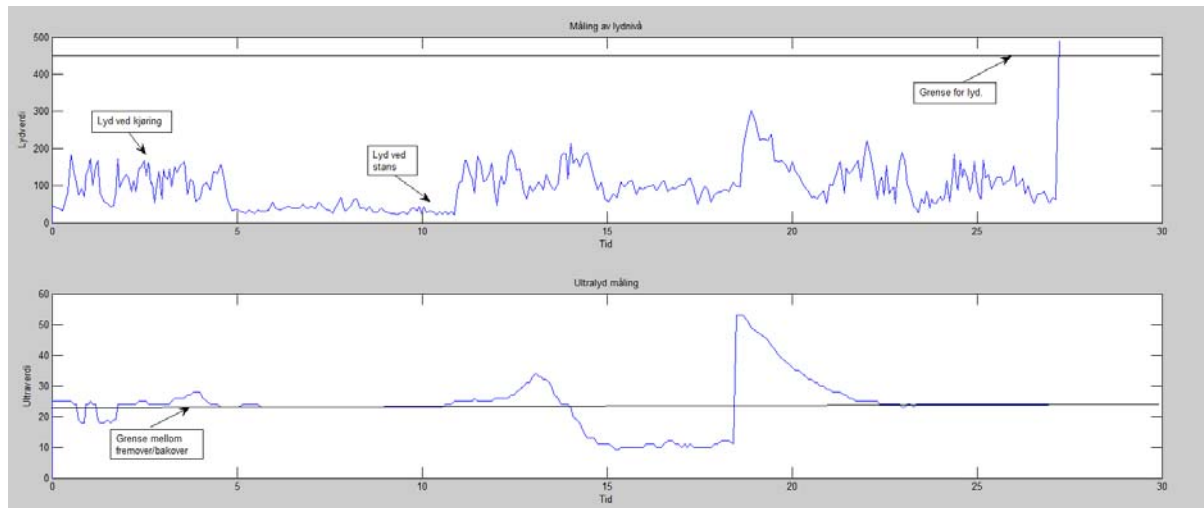
Figur 25- Aktiv tangent

8.6 CLAP DRIVE



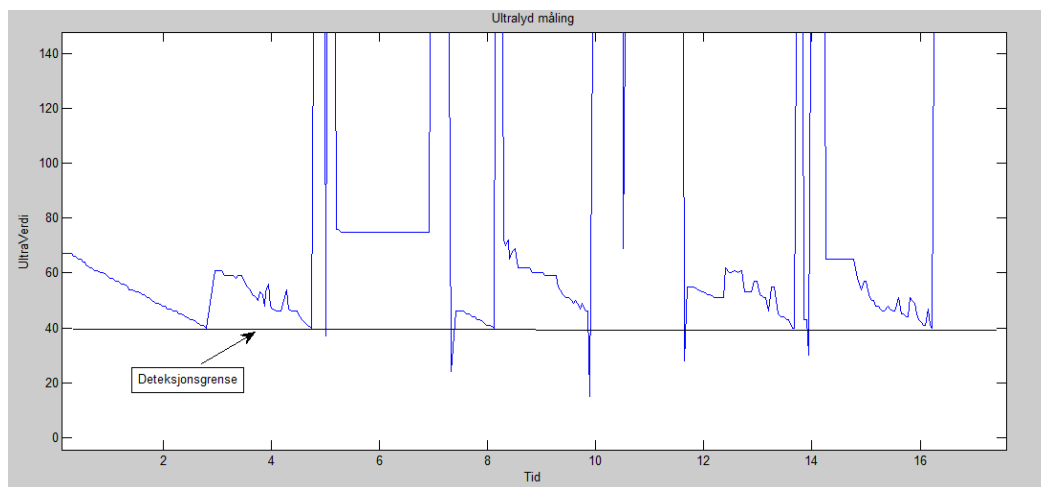
Figur 26- Lyd og Ultralyd graf under kjøring

8.7 ULTRA DRIVE



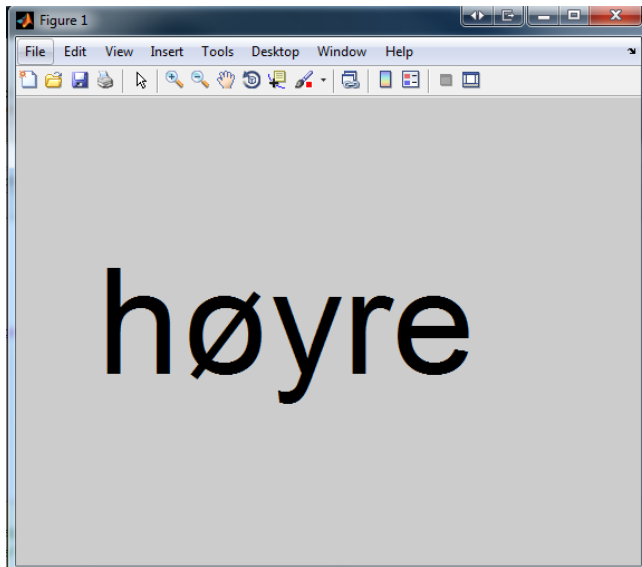
Figur 27 - Grafer under kjøring med Ultralydsensor

8.8 ULTRA WALL

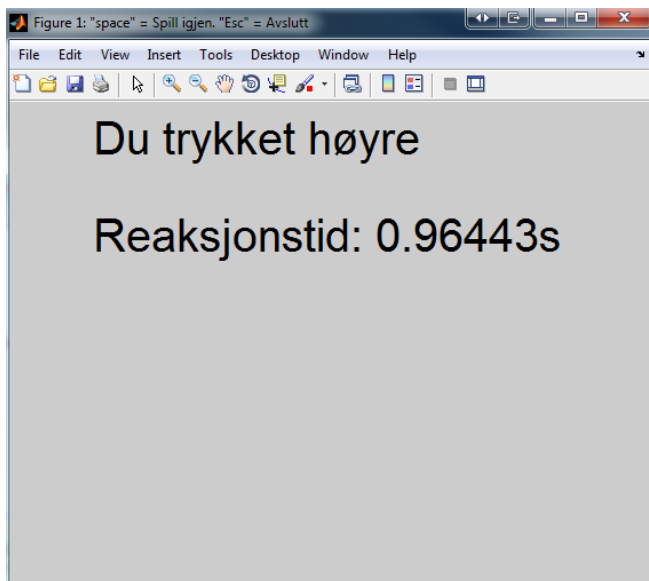


Figur 28- Graf for ultralyd under kjøring og omdreining

8.9 REACTION1

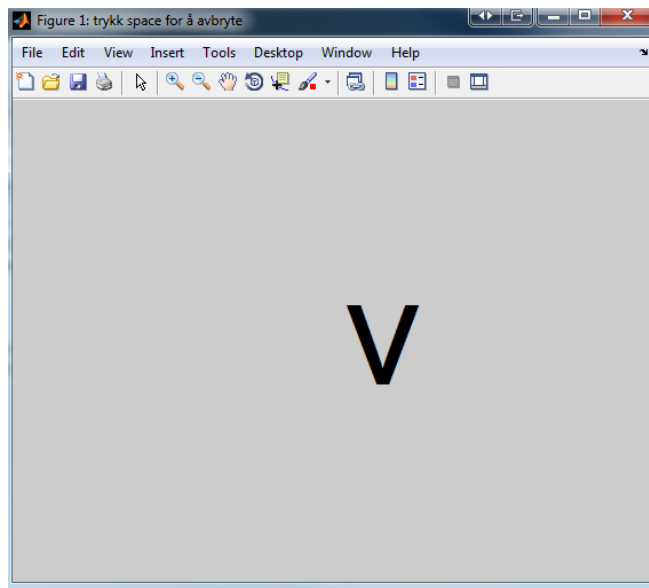


Figur 14- Vilkårlig retning, vist i figur

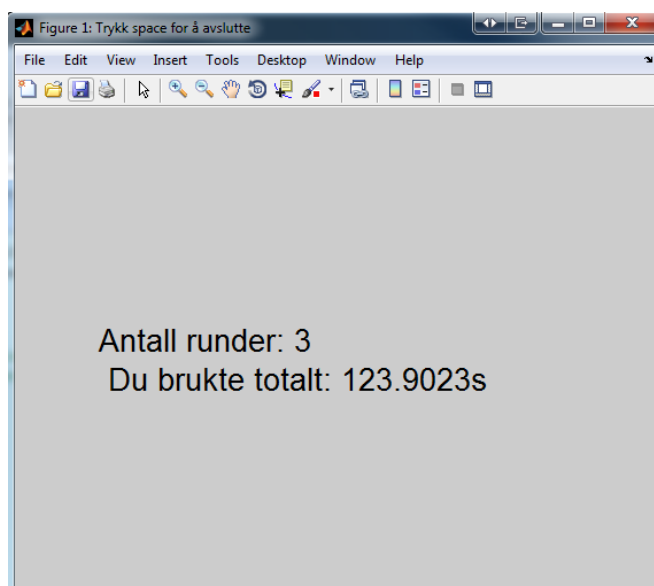


Figur 15- Vilkårlig retning, vist i figur

8.10 REACTION2



Figur 16- Vilkårlig bokstav vist i figur



Figur 17- Resultat

9 APPENDIKS B – MATLAB PROGRAM KODE

Alle Matlabfiler i skriftstørrelse 6.

9.1 MANUELL KJØRING

```
function [] = main()
    %% Main funksjon
    % 02.11.2014
    % Obligatorisk del av INT100 prosjekt
    % Gruppe 1401
    % *****
    % Det er gjort flere forbedringer til det som er obligatorisk
    % - Funksjoner er brukt gjennomgående
    % - Handles for oppdatering av grafer
    % - Fartmåling
    % - Grafisk visning av retning
    % - GUI meny
    % - Nullstilling av joystick ved initialisering
    % *****
    % Obligatorisk del ble først løst individuelt, koden ble deretter
    % kombinert til en og utbedringer ble gjort i den konsoliderte koden.
    % Alle har bidratt ihht avtalt arbeidsfordeling på main kode slik at
    % alle skulle få god kjennskap til denne delen.

    %% Initialiserer NXT
    initNXT();
    handle_NXT = COM_OpenNXT(); % etablerer nytt håndtak
    COM_SetDefaultNXT(handle_NXT); % setter globalt standard-håndtak

    %% Bruker input for hvor ofte grafene skal oppdateres
    prompt='Tast inn et tall fra 0 til 100000. Høyere tall gir bedre styring via joystick, mens lavere gir oftere
oppdatering av figurere og grafer: ';
    name='Graf oppdateringsrate';
    numlines=1;
    defaultanswer='10';
    answer=inputdlg(prompt,name,numlines,defaultanswer);
    plotFrek=str2double(answer);

    %% Initialiserer sensorer og motorer
    OpenLight(SENSOR_3,'ACTIVE'); % Lys sensor
    motorB = NXTMotor('B','SmoothStart',true); % Init motor b (høyre)
    motorC = NXTMotor('C','SmoothStart',true); % Init motor c (venstre)

    %% Initialiserer joystick
    joymex2('open',0); % Åpner joystick
    joystick = joymex2('query',0); % spør etter data fra joystick
    JoyMainSwitch = joystick.buttons(1); % Knapp 1, for å stoppe program
    initFB = -joystick.axes(2)/327.68; % henter joystick posisjon på "y"-aksen
    initS = joystick.axes(1)/327.68; % henter joystick posisjon på "x"-aksen

    %% Initialiserer variabler
    run = true; % loop variabel, settes til false for å avslutte programmet
    JoyFB = 0; % vektor for forover/bakover bevegelse av joystick
    JoyS = 0; % vektor for sideveis bevegelse av joystick
    tid=0; % tidsvektor
    deltaTid=0; % tidsendingsvektor
    paadragB = 0; % pådrag motor B
    paadragC = 0; % pådrag motor C
    lys = 0; % måling fra lyssensor
    lysFilt=0; % filtrert lysmåling
    lysNp=1023/2; % Maksverdien fra lyssensor
    avvikL=0; % vektor for filtrert lysverdi avviket fra nullpunkt
    avvikA=0; % vektor for integrert lysverdi = arealet A(t)
    avvikA2=0; % vektor for summen av arealet (integralet fra 0 til t)
    plotTeller = 0; % teller for hvor ofte det skal plottes
    verdi=0; % vektor for verdien som blir målt
    deriv=0; % vektor for de deriverte av avviket
    rettning=0; % vektor som beskriver rettning på avviket.
    startTid=cputime; % starttidspunkt
    speed= 0; % Fartsmåling
    screen = get(0,'screensize'); % Skjermstørrelse
    [rArrow, rmap] = imread('rarrow.jpg'); % Høyre pil for å angi retning
    [lArrow, lmap] = imread('larrow.jpg'); % Venstre pil for å angi retning
    [lrArrow, lrmap] = imread('lrarrow.png'); % Venstre/Høyre pil for å angi retning
    map=lrmap; % Variabel som angir map for retningspil
    arrow=lrArrow; % Angir hvilken retningspil som skal brukes
    w = 2; % Variabel til avsluttende meny
    h = 0; % MessageBox ved avslutning
    LenRead = 0; % Vector for avstandsmåling

    %% Initialiserer figurer med bruk av handles
    % Steng ned eksisterende figurer
    close all;

    % Main figur viser totalt avvik
    figure('Name','Hovedfigur','Position',[screen(3)/8, 4.5*screen(4)/8, 3*screen(3)/8,
2.8*screen(4)/8],'NumberTitle','off');
    plot1_1=plot(tid, avvikA2);
    set(plot1_1,'Ydata', avvikA2, 'Xdata', tid);
    hold on;
```

```

title('Totalt avvik: Integrert lysverdi fra t=0 til t');
xlabel('Tid i sekund');
ylabel('Totalt avvik');
legend('Totalt avvik som areal','Location','NorthWest');

% Figur som viser retning og fart i tittel
figure('Name','Retning','Position',[screen(3)/8, screen(4)/8,3*screen(3)/8, 2.8*screen(4)/8],'NumberTitle','off');
imshow(arrow,map);

%Integrert avvik figur, Lysverdi fra sensor inkl. nullpunkt, avvik rundt nullpunkt, integrert lysverdi
figure('Name','Integrert Avvik','Position',[4.5*screen(3)/8, 4.5*screen(4)/8,3*screen(3)/8,
2.8*screen(4)/8],'NumberTitle','off');
subplot(3,1,1)
plot3_1=plot(tid,lys,[0,tid(end)],[lysNp, lysNp]);
set(plot3_1,'Ydata', lys, 'Xdata', tid);
title('Lysverdi fra sensor inklusiv nullpunkt');
hold on;
subplot(3,1,2)
plot3_2=plot(tid,avvikL);
set(plot3_1,'Ydata', avvikL, 'Xdata', tid);
title('Avviket omkring nullpunktet');
hold on;
subplot(3,1,3)
plot3_3=plot(tid,avvikA);
set(plot3_1,'Ydata', avvikA, 'Xdata', tid);
title('Integrert lysverdi = arealet A(t)');
hold on;

%Derivert avvik figur, lysverdi, filtrert lysverdi, filtrert derivert
figure('Name','Derivert Avvik','Position',[4.5*screen(3)/8, screen(4)/8,3*screen(3)/8,
2.8*screen(4)/8],'NumberTitle','off');
%Lysverdi og Filtrert lysverdi
subplot(2,1,1)
plot4_1=plot(tid,lysFilt,tid,lys);
set(plot3_1,'Ydata', lysFilt, 'Xdata', tid);
title('Filtrert Lysmåling: Blå, Lysmåling: Rød');
hold on;
%Derivert av filtrert lysverdi
subplot(2,1,2)
plot4_2=plot(tid,deriv);
set(plot3_1,'Ydata', deriv, 'Xdata', tid);
title('Derivert av filtrert lysverdi');
%axis([0,tid(end),-1000,1000]);
hold on;

%% Main rutine som kjører roboten og oppdaterer grafer imens til knapp 1 blir trykket og run = false
tic;
while run
    % Tids beregninger
    tid(end+1)=toc;
    deltaTid(end+1)=tid(end)-tid(end-1);

    % les lys sensor
    lys(end+1)=GetLight(SENSOR_3);
    lysFilt(end+1)=filtLys([lysFilt(end),lys(end)]);

    % integrasjon av lyssignalet
    avvikL(end+1)=lysFilt(end)-lysNp;
    avvikA(end+1)=intFunk(tid,avvikL)+avvikA(end);
    avvikA2(end+1)=abs(intFunk(tid,avvikL))+avvikA2(end);

    % Derivasjon av lyssignalet
    deriv(end+1)=derivFunk(tid,avvikL)

    % Har roboten rettning mot lysere eller mørkere side
    rettning(end+1)=retFunk(deriv(end),rettning(end))

    % utregning av konkurranse poeng
    verdi(end+1)=tid(end)*100+avvikA2(end);

    % les joystick bevegelser
    joystick = joymex2('query',0);
    joyFB(end+1) = (-joystick.axes(2)/327.68)-initFB; % henter joystick posisjon på "y"-aksen
    joyS(end+1) = (joystick.axes(1)/327.68)-initS; % henter joystick posisjon på "x"-aksen

    % Filtrere joystick signalet
    joyFB(end) = filtJoy([joyFB(end-1),joyFB(end)]);
    joyS(end) = filtJoy([joyS(end-1),joyS(end)]);

    % Sett bilde som viser retning basert på derivert
    % 1 angir retning mot lysere side (venstre)
    if rettning(end) > 0;
        arrow=lArrow;
        map=lmap;
    % -1 angir retning mot mørkere side (høyre)
    elseif rettning(end) < 0;
        arrow=rArrow;
        map=rmap;
    end

    % Beregn motor pådrag
    [paadragB(end+1),paadragC(end+1)] = motorPaadrag(joyFB(end),joyS(end));

    % Send til motorene
    motorB.Power = paadragB(end);
    motorC.Power = paadragC(end);
    motorB.SendToNXT();
    motorC.SendToNXT();

    % Get speed data and calculate speed
    sBdata = motorB.ReadFromNXT(); % hent motorinformasjon
    sCdata = motorC.ReadFromNXT(); % hent motorinformasjon

```



```

        LenRead(end+1) = (Deg2Dist(sBdata.Position)+ Deg2Dist(sCdata.Position))/2; % Gjennomsnitt mm avstand på begge
motorer
speed(end+1) = round((((LenRead(end)-LenRead(end-1))) / deltaTid(end))); % delta avstand (mm) / delta tid (s)

% Plot figurer hvis plot frekvens er nådd
if plotTeller > plotFrek
    figure(1)
        set(plot1_1,'Ydata', avvikA2 , 'Xdata', tid);
        title(['Points in competition: ' num2str(verdi(end)) 's']);
    figure(2)
        imshow(arrow,map)
        title(['Part: ' num2str(speed(end)) ' mm/s'], 'FontSize', 16);
    figure(3)
        set(plot3_1,'Ydata', lys , 'Xdata', tid);
        set(plot3_2,'Ydata', avvikL , 'Xdata', tid);
        set(plot3_3,'Ydata', avvikA , 'Xdata', tid);
    figure(3)
        set(plot4_1,'Ydata', lysFilt , 'Xdata', tid);
        set(plot4_2,'Ydata', deriv , 'Xdata', tid);

    % reset plotcounter
    plotTeller = 0;
else
    plotTeller = plotTeller + 1;
end

% Tegn figurer
drawnow

%Sjekk om programmet skal avsluttes
JoyMainSwitch = joystick.buttons(1);
if JoyMainSwitch
    run = false;
end
end

%% Avslutt
% Stop motorer
motorB.Stop;
motorC.Stop;

% Steng kobling til sensorer
CloseSensor(SENSOR_3);

% Fjern joystick
clear joymex2

% Plott alt med de siste verdiene
% avsluttende tegning av alle figurer
close all;
% Plot figurer
% Plot figur 1
figure('Name','****PRESS ANY KEY TO RETURN TO MENU****','Position',[screen(3)/8, 4.5*screen(4)/8, 3*screen(3)/8,
2.8*screen(4)/8],'NumberTitle','off');
%totalt avvik som areal
subplot(4,1,1)
plot(tid,avvikA2);
title('Totalt avvik: Integrert lysverdi fra t=0 til t');
%Retting på avvik
subplot(4,1,2)
plot(tid,rettning);
title('Har roboten retning mot (-1)mørk eller (1)lys side');
axis([0,tid(end),-1.5,1.5]);
%Verdi i konkurranse ut fra gitt formel
subplot(4,1,3)
plot(tid,verdi);
title('Verdi i konkurranse');
%deltaTid
subplot(4,1,4)
plot(tid,deltaTid)
title('Tidsendring pr tidsinkrement: deltaTid');

% Plot figur 2
figure('Name','Retning','Position',[screen(3)/8, screen(4)/8,3*screen(3)/8, 2.8*screen(4)/8],'NumberTitle','off');
%Pådrag motor B
subplot(4,1,1)
plot(tid,paadragB)
title('Motor B');
%Pådrag motor C
subplot(4,1,2)
plot(tid,paadragC)
title('Motor C');
%Joystick posisjon i y-akse
subplot(4,1,3)
plot(tid,joyFB)
title('Joystick posisjon Fremover/Bakover');
%Joystick posisjon i x-akse
subplot(4,1,4)
plot(tid,joyS)
title('Joystick posisjon Sideveis');

%plot figur 3
figure('Name','Integrert Avvik','Position',[4.5*screen(3)/8, 4.5*screen(4)/8,3*screen(3)/8,
2.8*screen(4)/8],'NumberTitle','off');
%Lysverdi fra sensor og nullpunkt
subplot(3,1,1)
plot(tid,lys,[0,tid(end)],[lysNp , lysNp])
title('Lysverdi fra sensor inklusiv nullpunkt');
%Filtrert lysverdi avviket fra 0 punkt
subplot(3,1,2)
plot(tid,avvikL);
title('Avviket omkring nullpunktet');
%integrert lysverdi

```

```

subplot(3,1,3)
plot(tid,avvikA);
title('Integrert lysverdi = arealet A(t)')

%Plot figur 4
figure('Name','Derivert Avvik','Position',[4.5*screen(3)/8, screen(4)/8,3*screen(3)/8,
2.8*screen(4)/8],'NumberTitle','off');
%Lysverdi og Filtrert lysverdi
subplot(2,1,1)
plot(tid,lysFilt,tid,lys);
title('Filtrert Lysmåling: Blå, Lysmåling: Rød');
%Derviert av filtrert lysverdi
subplot(2,1,2)
plot(tid,deriv);
title('Derivert av filtrert lysverdi');
axis([0,tid(end),-1000,1000])

% Gi beskjed at bruker må taste key på tastatur for å stenge figurer.
if h == 0
    h = msgbox('Press any key on keyboard to close figures and return to menu','Message','help')
end

while w > 1
w = waitforbuttonpress;
    if w <= 1;
        % Steng NXT tilkobling
        initNXT();
        % Return to main menu
        %gui();
    end
end
end
end

```

9.2 AUTOMATISK KJØRING

```

function [] = auto()
%% Auto kjørs funksjon
% 02.11.2014
% kreativ del av INT100 prosjekt
% Gruppe 1401
% *****
% Dette er en kreativ oppgave hvor roboten skal kjøre gjennom lår,ypa
% automatisk. Her er joystick tatt ut og erstattet med kode som
% beregner motorpårdrag i forhold til derivert.

%% Initialiserer NXT
initNXT();
handle_NXT = COM_OpenNXT(); % etablerer nytt hÅvndtak
COM_SetDefaultNXT(handle_NXT); % setter globalt standard-hÅvndtak

%% Bruker input for hvor ofte grafene skal oppdateres
prompt=('Tast inn et tall fra 0 til 100000. HÅyere tall gir bedre styring via joystick, mens lavere gir oftere
oppdatering av figurer og grafer: ');
name='Graf oppdateringsrate';
numlines=1;
defaultanswer={'10'};
answer=inputdlg(prompt,name,numlines,defaultanswer);
plotPrek=str2double(answer);

%% Initialiserer sensorer og motorer
OpenLight(SENSOR_3,'ACTIVE'); % Lys sensor
motorB = NXTMotor('B','SmoothStart',true); % Init motor b (hÅyere)
motorC = NXTMotor('C','SmoothStart',true); % Init motor c (venste)

%% Initialiserer variabler
run = true; % loop variabel, settes til false for Å avslutte programmet
tid=0; % tidsvektor
deltaTid=0; % tidsendringsvektor
paadragB = 0; % påÅdrag motor B
paadragC = 0; % påÅdrag motor C
lys = 0; % mÅÅling fra lyssensor
lysFilt=0; % filtrert lysmÅÅling
lysNp=1023/2; % Maksverdien fra lyssensor
avvikL=0; % vektor for filtrert lysverdi avviket fra nullpunkt
avvikA=0; % vektor for integrert lysverdi = arealet A(t)
avvikA2=0; % vektor for summen av arealet (integralet fra 0 til t)
plotTeller = 0; % teller for hvor ofte det skal plottes
verdi=0; % vektor for verdien som blir mÅÅlt
deriv=0; % vektor for de deriverte av avviket
rettning=0; % vektor som beskriver rettning påÅ avviket.
startTid=cputime; % starttidspunkt
speed= 0; % FartsmÅÅling
screen = get(0,'screensize'); % SkjermstÅrrelse
[rArrow, rmap] = imread('rarrow.jpg'); % HÅyere pil for ÅÅ angi retning
[lArrow, lmap] = imread('larrow.jpg'); % Venstre pil for ÅÅ angi retning
[lrArrow, lrmap] = imread('lrarrow.png'); % Venstre/HÅyere pil for ÅÅ angi retning
map=lrmap; % Variabel som angir map for retningspil
arrow=lrArrow; % Angir hvilken retningspil som skal brukes
w = 2; % Variabel til avsluttende meny
h = 0; % MessageBox ved avslutning
LenRead = 0; % Vector for avstandsmÅÅling

%% Initialiserer figurer med bruk av handles
% Steng ned eksisterende figurer
close all;

% Main figur viser totalt avvik

```

```

figure('Name','Hovedfigur','Position',[screen(3)/8, 4.5*screen(4)/8, 3*screen(3)/8,
2.8*screen(4)/8],'NumberTitle','off');
plot1_1=plot(tid, avvikA2);
set(plot1_1,'Ydata', avvikA2, 'Xdata', tid);
hold on;
title('Totalt avvik: Integrert lysverdi fra t=0 til t');
xlabel('Tid i sekund');
ylabel('Totalt avvik');
legend('Totalt avvik som areal','Location','NorthWest');

% Figur som viser retning og fart i tittel
figure('Name','Retning','Position',[screen(3)/8, screen(4)/8,3*screen(3)/8, 2.8*screen(4)/8],'NumberTitle','off');
imshow(arrow,map);

%Integrert avvik figur, Lysverdi fra sensor inkl. nullpunkt, avvik rundt nullpunkt, integrert lysverdi
figure('Name','Integrert Avvik','Position',[4.5*screen(3)/8, 4.5*screen(4)/8,3*screen(3)/8,
2.8*screen(4)/8],'NumberTitle','off');
subplot(3,1,1)
plot3_1=plot(tid,lys,[0,tid(end)],[lysNp, lysNp]);
set(plot3_1,'Ydata', lys, 'Xdata', tid);
title('Lysverdi fra sensor inklusiv nullpunkt');
hold on;
subplot(3,1,2)
plot3_2=plot(tid,avvikL);
set(plot3_1,'Ydata', avvikL, 'Xdata', tid);
title('Avviket omkring nullpunktet');
hold on;
subplot(3,1,3)
plot3_3=plot(tid,avvikA);
set(plot3_1,'Ydata', avvikA, 'Xdata', tid);
title('Integrert lysverdi = arealet A(t)');
hold on;

%Derivert avvik figur, lysverdi, filtrert lysverdi, filtrert derivert
figure('Name','Derivert Avvik','Position',[4.5*screen(3)/8, screen(4)/8,3*screen(3)/8,
2.8*screen(4)/8],'NumberTitle','off');
%Lysverdi og Filtrert lysverdi
subplot(2,1,1)
plot4_1=plot(tid,lysFilt,tid,lys);
set(plot3_1,'Ydata', lysFilt, 'Xdata', tid);
title('Filtrert Lysmåling: BLÅ, Lysmåling: RÅ,d');
hold on;
%Derviert av filtrert lysverdi
subplot(2,1,2)
plot4_2=plot(tid,deriv);
set(plot3_1,'Ydata', deriv, 'Xdata', tid);
title('Derivert av filtrert lysverdi');
%axis([0,tid(end),-1000,1000]);
hold on;

%% Main rutine som kjører roboten og oppdaterer grafer imens til knapp 1 blir trykket og run = false
tic;
while run
    % Tids beregninger
    tid(end+1)=toc;
    deltaTid(end+1)=tid(end)-tid(end-1);

    % les lys sensor
    lys(end+1)=GetLight(SENSOR_3);
    lysFilt(end+1)=filtLys([lysFilt(end),lys(end)]);

    % integrasjon av lyssignalet
    avvikL(end+1)=lysFilt(end)-lysNp;
    avvikA(end+1)=intFunk(tid,avvikL)+avvikA(end);
    avvikA2(end+1)=abs(intFunk(tid,avvikL))+avvikA2(end);

    % Derivasjon av lyssignalet
    deriv(end+1)=derivFunk(tid,avvikL)

    % Har roboten retting mot lysere eller mårkere side
    retting(end+1)=retFunk(deriv(end),retting(end))

    % utregning av konkurranse poeng
    verdi(end+1)=tid(end)*100+avvikA2(end);

    % Sett bilde som viser retning basert på derivert
    % 1 angir retning mot lysere side (venstre)
    if retting(end) > 0;
        arrow=lArrow;
        map=lmap;
    % -1 angir retning mot mårkere side (høyre)
    elseif retting(end) < 0;
        arrow=rArrow;
        map=rmap;
    end

    % Beregn motor pådrag basert på lys avvik fra 0 punkt
    [paadragB(end+1),paadragC(end+1)] = Autofunc(avvikL(end)); % Legg inn pådrag ihht retning

    % Send til motorene
    motorB.Power = paadragB(end);
    motorC.Power = paadragC(end);
    motorB.SendToNXT();
    motorC.SendToNXT();

    % Get speed data and calculate speed
    sBdata = motorB.ReadFromNXT(); % hent motorinformasjon
    sCdata = motorC.ReadFromNXT(); % hent motorinformasjon
    LenRead(end+1) = (Deg2Dist(sBdata.Position)+ Deg2Dist(sCdata.Position))/2; % Gjennomsnitt avstand på begge motorer
    speed(end+1) = round((LenRead(end)-LenRead(end-1)) / deltaTid(end)); % delta avstand (mm) / delta tid (s)

    % Plot figurer hvis plot frekvens er nådd
    if plotTeller > plotFrek

```

```

figure(1)
set(plot1_1,'Ydata', avvikA2 , 'Xdata', tid);
title(['Points in competition: ' num2str(verdi(end)) 's']);
figure(2)
imshow(arrow,map)
title(['Fart: ' num2str(speed(end)) ' mm/s'], 'FontSize', 16);
figure(3)
set(plot3_1,'Ydata', lys , 'Xdata', tid);
set(plot3_2,'Ydata', avvikL , 'Xdata', tid);
set(plot3_3,'Ydata', avvikA , 'Xdata', tid);
figure(3)
set(plot4_1,'Ydata', lysFilt , 'Xdata', tid);
set(plot4_2,'Ydata', deriv , 'Xdata', tid);

% reset plotcounter
plotTeller = 0;
else
plotTeller = plotTeller + 1;
end

% Tegn figurer
drawnow

%Sjekk om programmet skal avsluttes. Det sjekkes om lysverdi
% overstiger en gitt verdi. Dette tolkes som robot har kjørt av
% banen.
if avvikL(end) >= 100 %
run = false;
end
end

%% Avslutt
% Stop motorer
motorB.Stop;
motorC.Stop;

% Steng kobling til sensorer
CloseSensor(SENSOR_3);

% Plott alt med de siste verdiene
% avsluttende tegning av alle figurer
close all;
% Plot figurer
% Plot figur 1
figure('Name','****PRESS ANY KEY TO RETURN TO MENU****','Position',[screen(3)/8, 4.5*screen(4)/8, 3*screen(3)/8,
2.8*screen(4)/8],'NumberTitle','off');
%totalt avvik som areal
subplot(4,1,1)
plot(tid,avvikA2);
title('Totalt avvik: Integrert lysverdi fra t=0 til t');
%Retting på Y avvik
subplot(4,1,2)
plot(tid,rettning);
title('Har roboten retning mot (-1)mÅ, rk eller (1)lys side');
axis([0,tid(end),-1.5,1.5]);
%Verdi i konkurransen ut fra gitt formel
subplot(4,1,3)
plot(tid,verdi);
title('Verdi i konkurranse');
%deltaTid
subplot(4,1,4)
plot(tid,deltaTid);
title('Tidsendring pr tidsinkrement: deltaTid');

% Plot figur 2
figure('Name','Retning','Position',[screen(3)/8, screen(4)/8,3*screen(3)/8, 2.8*screen(4)/8],'NumberTitle','off');
%PÅYdrag motor B
subplot(2,1,1)
plot(tid,paadragB);
title('Motor B');
%PÅYdrag motor C
subplot(2,1,2)
plot(tid,paadragC);
title('Motor C');

%plot figur 3
figure('Name','Integrert Avvik','Position',[4.5*screen(3)/8, 4.5*screen(4)/8,3*screen(3)/8,
2.8*screen(4)/8],'NumberTitle','off');
%Lysverdi fra sensor og nullpunkt
subplot(3,1,1)
plot(tid,lys,[0,tid(end)],[lysNp , lysNp]);
title('Lysverdi fra sensor inklusiv nullpunkt');
%Filtrert lysverdi avviket fra 0 punkt
subplot(3,1,2)
plot(tid,avvikL);
title('Avviket omkring nullpunktet');
%integrert lysverdi
subplot(3,1,3)
plot(tid,avvikA);
title('Integrert lysverdi = arealet A(t)');

%Plot figur 4
figure('Name','Derivert Avvik','Position',[4.5*screen(3)/8, screen(4)/8,3*screen(3)/8,
2.8*screen(4)/8],'NumberTitle','off');
%Lysverdi og Filtrert lysverdi
subplot(2,1,1)
plot(tid,lysFilt,tid,lys);
title('Filtrert lysmåling: BlÅV, Lysmåling: RÅ,d');
%Derivert av filtrert lysverdi
subplot(2,1,2)
plot(tid,deriv);
title('Derivert av filtrert lysverdi');
axis([0,tid(end),-1000,1000])

```

```

% Gi beskjed at bruker må trykke på tastatur for å stenge figurer.
if h == 0
    h = msgbox('Press any key on keyboard to close figures and return to menu','Message','help')
end

while w > 1
    w = waitforbuttonpress;
    if w <= 1;
        % Steng NXT tilkobling
        initNXT();
        % Return to main menu
        %gui();
    end
end
end
end

```

9.3 GRAFISK BRUKERGRENSESNIFF (GUI)

```

function [] = gui()
%% GUI to create choices to select the various projects
n=1
while n>0
    switch menu('Ing100 prosjekt', 'Robot kjøring', 'Kreative oppgaver', 'Avslutt')
    case 1
        m1=1
        while m1>0
            switch menu('Robot kjøring', 'Manuell kjøring', 'Automatisk kjøring', 'Tilbake')
            case 1
                main()
            case 2
                auto()
            case 3
                m1 = 0; % Gå tilbake til forrige meny
            end
        end
    case 2
        m2 = 1
        while m2 > 0
            switch menu('Kreative oppgaver', 'Spill', 'Musikk', 'Robo Fun', 'Plot test', 'Tilbake')
            case 1
                m2_1 = 1
                while m2_1 > 0
                    switch menu('Spill', 'CannonGame', 'Reaksjonstest1', 'Reaksjonstest2', 'Tilbake')
                    case 1
                        cannonGame()
                    case 2
                        Reaction1()
                    case 3
                        Reaction2()
                    case 4
                        m2_1 = 0; % Gå tilbake til forrige meny
                    end
                end
            case 2
                m2_2 = 1
                while m2_2 > 0
                    switch menu('Musikk', 'Robo Read 2Music', 'Piano', 'Joystick Play', 'Tilbake')
                    case 1
                        graymusic()
                    case 2
                        piano()
                    case 3
                        joymusic()
                    case 4
                        m2_2 = 0; % Gå tilbake til forrige meny
                    end
                end
            case 3
                m2_3 = 1
                while m2_3 > 0
                    switch menu('Robo Fun', 'Robo Clap', 'Robo Follow', 'Robo Wall', 'Tilbake')
                    case 1
                        lydsensorclap()
                    case 2
                        ultrahandfollow()
                    case 3
                        ultrawall()
                    case 4
                        m2_3 = 0; % Gå tilbake til forrige meny
                    end
                end
            case 4
                plotTime();
            case 5
                m2 = 0; % Avslutt has been pressed n is set to 0 to end while loop and terminate
            end
        end
    case 3
        n = 0; % Avslutt has been pressed n is set to 0 to end while loop and terminate
    end
end
end
end

```

9.4 CANNON GAME

```
function [] = cannonGame()

% A simple game that simulates a projectile being shot from a cannon at an
% angle between 0° and 90°.
% Game can be played alone or in multiplayer up to five player
% Players should see the shots they are making
% One game consist of 3, 5, 7 or 10 shots pr player
% A summary should be shown at the end
% There is a highscore list

%%
% Constants
screen = get(0,'screensize');
chartratio=(550/1050)*screen(3)/screen(4); %ratio between plot axis times ratio between screen size
chartaxis=[0,1050,0,550]; %Contant to lock the displayed axis slightly larger than possible height and length with an initial
speed of 100 m/s
g=-9.81; %Gravitational acceleration
initialSpeed = 100; %Shots intial speed m/s

%%
% Initiate variables
showangle=1; %Condition to run showangle function
space = 0; %Continue variable
stop = 0; %main loops variable
angle = 0; %angle of shot
shotno = 1; %shoot number
noshots = 0; %number of shots this round
dispshot=0; %display shot condition
showhighscore=false; %show highscore
noplayers=0; %number of players
playerno=1; %current players number
highscore=loadHighscore(); %structur variable used for highscores, and initially loading highscores from file

%%
% Initial game options
%such as number of players, difficulty, highscore and exit
switch menu('Choose: ', 'Single player', 'Multiplayer', 'Highscore','Exit')
case 1
    noplayers=1;
case 2
    noplayers=menu('Number of players: ', '2', '3', '4', '5')+1;
case 3
    noplayers=0;
    showhighscore=true;
case 4
    noplayers=0;
end

if noplayers>0
    switch menu('Choose difficulty:', 'easy', 'normal', 'hard'); % Game difficulty
    case 1
        difficulty = 0.01; %change from one angle to the next in radians
    case 2
        difficulty = 0.02; %change from one angle to the next in radians
    case 3
        difficulty = 0.03; %change from one angle to the next in radians
    end

    switch menu('Number of shots:', '3', '5', '7', '10') % Number of shots pr game
    case 1
        noshots = 3;
    case 2
        noshots = 5;
    case 3
        noshots = 7;
    case 4
        noshots = 10;
    end
end

%%
% Initialize figure
% By defining position on screen, the keydownlistnere and removing the possibility to resize the figure.
figure('Position',[screen(3)/6, screen(4)/6,screen(3)/1.5, screen(4)/1.5]);
set(gcf, 'KeyPressFcn', @keyDownListener, 'Resize', 'off');

%%
%keyDownListener
% to know when "space" has been hit, also escape to stop main loops
function keyDownListener(src, event)
    switch event.Key
    case 'space'
        space=1;
        showangle=0;
    case 'escape'
        stop=1;
    end
end

%%
%load highscore from file
```

```

function hs = loadHighscore()
    try %try loading highscore file
        hs=load('highscore_cannongame.mat');
    catch %if loading highscore file fails, make it
        lShots = zeros(1,10); %longest shots
        tot3shots = zeros(1,10); %highest total of 3 shot games
        tot5shots = zeros(1,10); %highest total of 5 shot games
        tot7shots = zeros(1,10); %highest total of 7 shot games
        tot10shots = zeros(1,10); %highest total of 10 shot games
        save('highscore_cannongame', 'lShots', 'tot3shots', 'tot5shots', 'tot7shots', 'tot10shots'); %create the highscore
    end
end

file
    hs=load('highscore_cannongame.mat');
end

%%
%update longest shot highscore
function updateLongestShotHS(shotlength)
    highscore.lShots(end+1) = shotlength; %add current shots to longest shot list
    highscore.lShots = sort(highscore.lShots, 'descend'); %resort the longest shot list
    lShots = highscore.lShots(1:10); %remove the lowest value from longest shot list
    save('highscore_cannongame', '-append', 'lShots'); %resave the longest shot list
end

%%
%update total length highscore see comments for update longest shot
%highscore, it follows exact same logic only for the rest of the lists.
function updateTotalLengthHS(totallength)
    if noshots == 3
        highscore.tot3shots(end+1) = totallength;
        highscore.tot3shots = sort(highscore.tot3shots, 'descend');
        tot3shots=highscore.tot3shots(1:10);
        save('highscore_cannongame', '-append', 'tot3shots');
    elseif noshots == 5
        highscore.tot5shots(end+1) = totallength;
        highscore.tot5shots = sort(highscore.tot5shots, 'descend');
        tot5shots=highscore.tot5shots(1:10);
        save('highscore_cannongame', '-append', 'tot5shots');
    elseif noshots == 7
        highscore.tot7shots(end+1) = totallength;
        highscore.tot7shots = sort(highscore.tot7shots, 'descend');
        tot7shots=highscore.tot7shots(1:10);
        save('highscore_cannongame', '-append', 'tot7shots');
    elseif noshots == 10
        highscore.tot10shots(end+1) = totallength;
        highscore.tot10shots = sort(highscore.tot10shots, 'descend');
        tot10shots=highscore.tot10shots(1:10);
        save('highscore_cannongame', '-append', 'tot10shots');
    end
end

%%
%showAngle function, displaying a line that moves from 0° towards 90° then back
%towards 0°
function showAngle()
    cannonLength = 100; %lengt of line to illustrate the angle with
    while ~space && angle<pi/2 && ~stop %show line from 0° towards 90°
        x=cannonLength*cos(angle); %furthest x value of line
        y=cannonLength*sin(angle)*chartratio; %highest y value of line times chartratio display a more even size on
    screen
        plot([0,x],[0,y]); %plot the line using point [0,0] and [x,y]
        axis(chartaxis); %set the axis that is to be locked
        title(['Player ', num2str(playerno), ': Shot no: ', num2str(shotno), '. Press "space" to fire at desired
    angle']); %give instructions in title
        drawnow; %make sure it gets drawn
        angle = angle + difficulty; %increase angle for next iteration by the value set by the selected difficulty
    end

    while ~space && angle>0 && ~stop %show line from 90° towards 0°
        x=cannonLength*cos(angle); %furthest x value of line
        y=cannonLength*sin(angle)*chartratio; %highest y value of line times chartratio display a more even size on
    screen
        plot([0,x],[0,y]); %plot the line using point [0,0] and [x,y]
        axis(chartaxis); %set the axis that is to be locked
        title(['Player ', num2str(playerno), ': Shot no: ', num2str(shotno), '. Press "space" to fire at desired
    angle']); %give instructions in title
        drawnow; %make sure it gets drawn
        angle = angle - difficulty; %decrease angle for next iteration by the value set by the selected difficulty
    end
end

%%
%calculate and show shot
function [shotlength,shotheight] = shootCannon()
    clf;
    v0=initialSpeed; %initial speed
    v0x=v0*cos(angle); %initial speed in x direction
    v0y=v0*sin(angle); %initial speed in y direction
    t=-(2*v0y/g); %calculate time until shot hits ground again
    tid=0:0.1:t; %make a time vector with steps 0.1 sec
    tid(end+1)=t; %add the exact time of shot hitting the ground as last time in timevector
    x=v0x.*tid; %calculate each x value by time
    y=v0y.*tid+0.5*g.*tid.^2; %calculate each y value by time
    for i=2:length(x) % simulate shot by drawing small lines
        plot([x(i-1) x(i)], [y(i-1) y(i)]); %plot the small lines
    end
end

```

```

        axis(chartaxis); %set the axis that is to be locked
        title(['Shot current length: ' num2str(x(i)) 'm. Shot current height: ' num2str(y(i)) 'm.']); %display details
    during shot
        drawnow; %make sure it gets drawn
        pause(0.01); %short pause to make sure it is a smooth illustration
    end
    title(['Shot length: ' num2str(x(end)) 'm. Shot height: ' num2str(max(y)) 'm. Press "space" to continue.']); % give
    details about shot, such as shot length
    dispshot=1; % set shot to be displayed until continued
    angle=0; %reset angle
    shotlength = x(end); %define shotlength output from function
    shotheight = max(y); %define shotheight output form function
    updateLongestShotHS(shotlength); %run function to update longest shot highscore
end

%%
%show highscores in figure
function showHighscores()
    HiStL = {'Longest shots: '}; %highscore structure string longest shot
    HiStT3 = {'Highest Total 3: '}; %highscore structure string sum of 3 shots
    HiStT5 = {'Highest Total 5: '}; %highscore structure string sum of 5 shots
    HiStT7 = {'Highest Total 7: '}; %highscore structure string sum of 7 shots
    HiStT10 = {'Highest Total 10: '}; %highscore structure string sum of 10 shots
    for i=1:10 % load the highscore lists into the correct structures
        HiStL(end+1) = {[num2str(i) ' ' num2str(highscore.lShots(i)) 'm']}];
        HiStT3(end+1) = {[num2str(i) ' ' num2str(highscore.tot3shots(i)) 'm']}];
        HiStT5(end+1) = {[num2str(i) ' ' num2str(highscore.tot5shots(i)) 'm']}];
        HiStT7(end+1) = {[num2str(i) ' ' num2str(highscore.tot7shots(i)) 'm']}];
        HiStT10(end+1) = {[num2str(i) ' ' num2str(highscore.tot10shots(i)) 'm']}];
    end
    %write highscore lists to textboxes
    annotation('textbox', [0.1,0.9,0.75,0.05],'backgroundcolor', 'w', 'String', 'Highscores, Press "space" to continue.');
```

```

    annotation('textbox', [0.1,0.1,0.15,0.8],'backgroundcolor', 'w', 'String', HiStL);
    annotation('textbox', [0.25,0.1,0.15,0.8],'backgroundcolor', 'w', 'String', HiStT3);
    annotation('textbox', [0.4,0.1,0.15,0.8],'backgroundcolor', 'w', 'String', HiStT5);
    annotation('textbox', [0.55,0.1,0.15,0.8],'backgroundcolor', 'w', 'String', HiStT7);
    annotation('textbox', [0.7,0.1,0.15,0.8],'backgroundcolor', 'w', 'String', HiStT10);
    while ~space %draw and wait until player is ready to continue
        drawnow;
    end
end

%%
%main single player script
if noplayers ==1;
    shotlength=zeros(1,noshots); %define vector for putting the length of the shots into
    shotheight=zeros(1,noshots); %define vector for putting the height of the shots into
    while ~stop %main single player game loop
        drawnow;
        if showangle %show the angle to the player
            showAngle();
        elseif space && ~dispshot %shot with the angle when player presses space
            [shotlength(shotno), shotheight(shotno)] = shootCannon();
            space=0;
            shotno=shotno+1;
        elseif shotno<=noshots && space %if more shots remaining reset conditions
            showangle = 1;
            dispshot = 0;
            space=0;
        elseif shotno>noshots && space %all shots completed, finish game
            stop=1;
        end
    end
    updateTotalLengthHS(sum(shotlength)); %update total length highscores
    clf;
    %display game details
    tempstr = {'Longest shot: ', num2str(max(shotlength)), 'm'},
        {'Highest shot: ', num2str(max(shotheight)), 'm'},
        {'Total shot length: ' num2str(sum(shotlength)), 'm'},
        'Press "Space" to continue';
    annotation('textbox', [0.1,0.1,0.8,0.8],'backgroundcolor', 'w', 'String', tempstr);
    space=0;
    while ~space %wait until player is ready to continue
        drawnow
    end
    close(gcf;
end

%%
%main multiplayer
if noplayers>1;
    shotlength=zeros(noplayers,noshots); %define vector for putting the length of the shots into
    shotheight=zeros(noplayers,noshots); %define vector for putting the height of the shots into
    playerno=1;
    while ~stop %main multiplayer game loop
        drawnow;
        if showangle %shot with the angle when player presses space
            showAngle();
        elseif space && ~dispshot %shot with the angle when player presses space
            [shotlength(playerno,shotno), shotheight(playerno,shotno)] = shootCannon();
            space=0;
            if playerno==noplayers %iterate through players and shot numbers
                shotno=shotno+1;
                playerno=1;
            else

```



```

        playerno=playerno+1;
    end
    elseif shotno<=noshots && space %if more shots remaining reset conditions
        showangle = 1;
        dispshot = 0;
        space=0;
    elseif shotno>noshots && space %all shots completed, finish game
        stop=1;
    end
end
clf;
winner = 0;
highTotThisGame = 0;
while playerno <= noplayers
    tempTot=sum(shotlength(playerno,:)); %get total length for each player
    updateTotalLengthHS(tempTot); %update highscore with total length for each player
    if highTotThisGame < tempTot %decide who is winning
        winner = playerno;
        highTotThisGame = tempTot;
    end
    %create details to send to textbox
    tempstr=[['Player ',num2str(playerno)],['Longest shot: ', num2str(max(shotlength(playerno,:))), 'm'],
        ['Highest shot: ', num2str(max(shotheight(playerno,:))), 'm'],['Total shot length: ' num2str(tempTot), 'm']];
    %create textbox for each player
    annotation('textbox', [0.1+(0.15*(playerno-1)),0.1,0.15,0.8],'backgroundcolor', 'w', 'String', tempstr);
    playerno=playerno+1;
end
%create textbox with winner and continue instructions
annotation('textbox', [0.1,0.9,0.75,0.05],'backgroundcolor', 'w', 'String', ['Winner is Player ',num2str(winner), '
Press "space" to continue']);
space=0;
while ~space %wait until player is ready to continue
    drawnow
end
close(gcf);
end

%%
% end
if noplayers>0 %if game just ended, give intial options again
    cannonGame();
elseif noplayers == 0 && showhighscore %show highscore, then show initial options again
    showHighscores();
    close(gcf);
    cannonGame();
end
close(gcf);
end

```

9.5 REACTION1

```

function [] = Reaction1()
%% Høyre / Venstre reaksjon

% Anders Svalestad
% 10.11.2014

% Beskrivelse: Et spill som lar deg teste reaksjonstiden.
% Høyre eller venstre blir vist i en figur. Bruker må taste høyre eller
% venstre piltast på tastatur ut fra vist retning.
%%

clear
clc

%%
% keyDownListener. Brukes til å detektere tastatur input i figur. Laget for
% å slippe input + Enter tast

function keyDownListener(src, event)
    resultat=event.Key;

% Hvis riktig tastatur piltast ble trykket, stoppes klokke, og
% figur med resultat åpnes.
    try
        if resultat == rett
            steg = 0;
            tid = toc;
        end
    catch
    end
end

%%
% keyDownListener. Brukes til å detektere tastatur input i figur. Laget for
% å slippe input + Enter tast

function keyDownListener2(src, event)
    event.Key

% Etter figur med resultat har blitt åpnet. Kan du trykke "space" - tast
% for å returnere til menu, eller "escape" tast for å avslutte spill.
    try

```

```

        if event.Key == 'space'
            steg2 = 0;
            spilligjen = 1;
        end
    catch
        try
            if event.Key == 'escape'
                spilligjen = 0 ;
                steg2 = 0;
            end
        catch
            end
        end
    end
end

%%
% Meny åpnes for å starte / avslutte spill. Når spill startes, går det
% mellom 0 og 5 sec før "høyre/venstre" retning blir vist.

switch menu('Start/Avbryt','Start','Avslutt')
    case 1
        steg = 1;
        steg2=1;
        pause(rand*5);
    case 2
        steg = 0;
        steg2 = 0;
end
%%
% Hvis spill blir startet, åpnes en figur som skriver ut vilkårlig retning,
% "venstre/høyre" i en figur.
if steg
    clf
    figure(1)
    % Skalering av figur.
    % Riktig piltast blir definert ut fra hvilekn retning som blir generert.
    axis([-10 10 -10 10]);
    if rand < 0.5
        A = 'venstre'
        rett = 'leftarrow';
    else
        A = 'høyre'
        rett = 'rightarrow';
    end
    retning = text(-10,0,A);
    set(gca,'visible','off');
    set(retning,'fontsize',100);
    set(gcf,'KeyPressFcn',@keyDownListener)
    tic;
    while steg == 1;
        drawnow;
    end
end

%%
% Figur for resultat
if steg2
    clf
    figure(1)
    set(gcf,'Name','"space" = Spill igjen. "Esc" = Avslutt ');
    axis([-10 10 -10 10]);
    tekst1 = text(-10,10,['Du trykket ' A]);
    tekst2 = text(-10,5,['Reaksjonstid: ' num2str(tid) ' s']);
    set(gca,'visible','off');
    set(tekst1,'fontsize',30);
    set(tekst2,'fontsize',30);
    set(gcf,'KeyPressFcn',@keyDownListener2);
    while steg2 == 1;
        drawnow;
    end

    if spilligjen
        close(gcf);
        Reaction1();
    end
end
close(gcf);

end

```

9.6 REACTION2

```

function [] = Reaction2()
%% Reaksjon test.

% Anders Svalestad
% 23.10.2014

```

```

% Programmet har til hensikt å teste reaksjonstid sammen med
% tastatur "touch" ferdigheter til bruker

%%
clear
clc

%%
% Init
tid=[];

%%
%keyDownListener. Brukes til å detektere når riktig tast blir tastet på
%keyboard.
function keyDownListener(src, event)
    resultat=event.Key;
    % Hvis bokstav er riktig, Stopp klokke og vis reultat.
    if resultat == bokstav;
        tid(end+1) = toc;
        disp = ([ 'Riktig bokstav. Du brukte: ' num2str(tid(end)) ' sekund' ]);
        funnetBokstav = 1;
    elseif resultat == 'space'
        close all
        Reaction2()
    end
end

%%

% valg = menu('Hvor mange runder vil du prøve?','3','5','10')
switch menu('Hvor mange runder vil du prøve?','3','5','10','Avslutt')
    case 1
        runder=3;
    case 2
        runder=5
    case 3
        runder=10;
    case 4
        runder = 0

end

for n = 1:runder
    % lukker alle figurer
    clf;
    % Velger en vilkårlig bokstav mellom A og Z
    liste = 'abcdefghijklmnopqrstuvwxyz';
    bokstav = liste(ceil(length(liste)* rand));

    % Setter opp figur og viser villkårlig bokstav
    figure(1)
    axis([-2 2 -2 2]);
    antallrunder = text(0,0,bokstav);
    set(gcf,'name','trykk space for å avbryte')
    set(gca,'visible','off');
    set(antallrunder, 'fontsize',100);
    set(gcf, 'KeyPressFcn', @keyDownListener)

    % Starter stoppeklokke
    tic;
    % Finn vist bokstav. Tast bokstaven via tastatur og trykk enter
    %resultat = input('tast in vist bokstav: ','s');
    funnetBokstav = 0;
    while ~funnetBokstav
        drawnow;
    end

end

% Lukker alle figurer
close all

% Setter opp figur for resultat visning
figure(1)

    axis([-20 00 -20 20]);
    antallrunder = text(-20,0,['Antall runder: ' num2str(runder)]);
    Tiden = text(-20,-4,[' Du brukte totalt: ' num2str(sum(tid)) ' s']);
    set(gcf, 'name', 'Trykk space for å avslutte')
    set(gca,'visible','off');
    set(antallrunder, 'fontsize',20);
    set(Tiden, 'fontsize',20);

% Hvis avslutt knapp blir trykket. Lukk figur og returner
if runder==0
    close(gcf;
    gui();
end
end

```

9.7 KEYBOARDPIANO

```
function keyboardPiano()

%% Oppgave 14 "Piano" fra "Forslag til kreative oppgaver"
% ING100 Gruppe 1401 "Daniel Løvik" 2014

% Beskrivelse av program:
% Dette programmet simulerer noen tangenter slik som på et piano,
% Utvalgte tangenter for melodien "Fader Jacob" er definert.
% Bokstavene a til k brukes for Fjerde tangenter.
% Hvilken tangent som spilles av vises i figuren.

%% Frekvenser for taster hentet fra: https://en.wikipedia.org/wiki/Piano_key_frequencies
%%
%%      A4  A5  B4  B5      C4  C5  D4  E4  F4  G4  G5
TanFrek = [440,880,493.9,987.7,261.6,783.9,293.7,329.6,349.2,392.1,96]; % Frekvenser for tangenter i Hz
Tangnavn = {'A4','A5','B4','B5','C4','C5','D4','E4','F4','G4','G5'}; % Tangent navn.
figure1 = figure;
    annotation(figure1,'textbox',... % Tekstbox på figuren med informasjon om tangenter.
    [0.101616954474097 0.852380952380952 0.65541601255887 0.137891421159905],...
    'String',{'Fjerde tangenter for tastene a til k.'},...
    'HorizontalAlignment','center',...
    'FontSize',16,...
    'FontName','Agency FB',...
    'FitBoxToText','off',...
    'BackgroundColor',[1 1 1]);

    annotation(figure1,'textbox',... % Tekstbox på figuren med tangenter for "Father Jacob"
    [0.0767857142857143 0.0364168268204139 0.698214285714285 0.137891421159905],...
    'String',{'Father Jacob.', 'd-f-g-d-|-d-f-g-d-|-g-h-j-|-g-h-j-|-j-a-j-h-g-d-|-j-a-j-h-g-d-|-k-k'},...
    'HorizontalAlignment','center',...
    'FontSize',16,...
    'FontName','Agency FB',...
    'FitBoxToText','off',...
    'BackgroundColor',[1 1 1]);

set(figure1, 'KeyPressFcn', @KeyDownListener); % Figuren leser inn taster og funksjonen KeyDownListener utfører
% variabeloppdateringer som resulterer i tone + text på figuren.

%%
% Definerte tangenter koblet opp mot frekvens og sampling multiplikator.
function KeyDownListener(src,event)
    k=event.Key;
    if k == 'a'
        lydnnummer(end+1) = 1;
    elseif k=='q';
        lydnnummer(end+1) = 2;

    elseif k=='s'
        lydnnummer(end+1) = 3;

    elseif k=='w'
        lydnnummer(end+1) = 4;

    elseif k=='d'
        lydnnummer(end+1) = 5;

    elseif k=='e'
        lydnnummer(end+1) = 6;

    elseif k=='f'
        lydnnummer(end+1) = 7;

    elseif k=='g'
        lydnnummer(end+1) = 8;

    elseif k=='h'
        lydnnummer(end+1) = 9;

    elseif k=='j'
        lydnnummer(end+1) = 10;

    elseif k=='k'
        lydnnummer(end+1) = 11;

    elseif k=='escape' % Ved trykk på Esc knapp.
        Plyd = 0; % Lukker figuren ned.
    end
end

%% Kode snutt under hentet og modifisert fra: http://www.mathworks.se/matlabcentral/fileexchange/26509-musical-notes
% Snutten under brukes til å generere forskjellige lyder basert på ASCII
Fs=10000; % Samplings rate, modifikasjon av verdi medfører endring i tonefall.
t=0:2.1/(Fs):1;

lyder = []; % Lyder vektor.
for i=1:length(TanFrek) % Teller fra 1 til lengden av vektoren TanFrek.
    lyder(:,i)=cos(TanFrek(i).*pi*t); % Ganger valgt frekvens med Cosinus formel for å generere tone.
end

%% Initialisering
Plyd=1; % Setter variabel til "True" for While loop.
lydnnummer = []; % Vektor for tangentnummer.
while Plyd % True
```

```

drawnow; % Oppdaterer figuren.
if length(lydnummer) > 0 && Plyd % Gyldig så lenge lydnummer er større enn 0 og Plyd true.
    % Korresponderende taster generer tone+ tekst på figur.
    if lydnummer(end) > 0 && lydnummer(end) < 12; % Gyldig så lenge lydnummer verdien er mellom 0 og 12.
        anolyd = char(Tangnavn(lydnummer(1))); % Leser inn hvilken tast som blir trykket og gir det videre til
tekstboxen.
        annotation('textbox',...% Tekstbox for visning av tangent trykket.
            [0.100470957613815 0.182716049382716 0.656200941915228 0.662111536824181], 'String',{anolyd},...
            'HorizontalAlignment','center',...
            'FontSize',180,...
            'FitBoxToText','off',...
            'BackgroundColor',[1 1 1]);
        sound(lyder(:,lydnummer(1))); % Lydnummer fra tastetrykk leses av og rett frekvens hentes inn og tone genereres.
    end
    try % Exception handler, prøver koden under.
        lydnummer = lydnummer(2:end); % Hvis element 2 ikke eksisterer pga mangel på elementer i vektor fanges dette opp.
    catch % Koden under setter vektoren tom uten at bruker får en feilmelding på ovenvent kode.
        lydnummer = [];
    end
end
end

uiwait(msgbox('Takk for at du brukte dette programmet!','Avslutter','modal')); % Meldingsboks som venter på brukerinput.
close(gcf); % Stenger ned figuren.
end

```

9.8 GRAY MUSIC

```

function [] = graymusic()
%% Kreativ oppgave av Helge Bjorland

%% Bruk lysm?ler til ? spille musikk
% Bruker lysm?ler til ? lese gr?toner separatert av hvitt felt p? ett ark, hvor gr?tone
% repr. en tone, og den fysiske lengden av gr?tonene p? arket repr.
% tidslengde som hver tone skal spilles. Man velger:
% + st?rrrelse p? arket
% + Lyd lengde multiplierer
% + Robot fart under m?ling
% + Presisjon p? lysm?ling
% Roboten kj?rer over arket og spiller deretter melodien

%% Initialiserer NXT
initNXT();
handle_NXT = COM_OpenNXT(); % etablerer nytt h?ndtak
COM_SetDefaultNXT(handle_NXT); % setter globalt standard-h?ndtak

%% Initialiserer sensorer
OpenLight(SENSOR_3,'ACTIVE'); % ?pner lys sensor

%% Initialiserer motorer

motorB = NXTMotor('B','SmoothStart',true); % Initialiser motor b (h?yre)
motorC = NXTMotor('C','SmoothStart',true); % Initialiser motor c (venste)

%% Initialiser variabler
lys = 0; % m?ling fra lyssensor
lysFilt=0; % filtrert lysm?ling
RoboSpeed = 10; % Robot speed m/s
TotLen = 300; % Total length robot should drive
LenRead=5; % M?ler lengden robot har kj?rt
SoundLen= 10; % Lengden p? tonen i forhold til distanse
RoboPrec= 5; % Presisjons faktor for lysm?ler
Mlen= 0; % Tone lengde
Mtone= 200; % Tone frekvens
templys = 0; % lysm?ling temp variabel
lysneg = 0; % lysm?ling derivert positiv eller negativ
i=1; % teller i while loop
danceDirect=1; % Danse retning
danceSpeed=15; % Danse fart

function [TotLen,SoundLen,RoboSpeed,RoboPrec]=menuCh()
% Meny for ? sette papirst?rrrelse

switch menu('Choose length robot should drive to measure: ', 'A3', 'A4', 'A5','Back')
case 1
    TotLen=420; %papersize in mm
case 2
    TotLen=300; %papersize in mm
case 3
    TotLen=210; %papersize in mm
case 4
    main();
end

% Meny for ? sette variabler fra bruker input
prompt={'Sound length multiplier? Default is 1 (1mm=1ms). Higher number give longer tones',...
    'Input robot speed when measuring? Default is 10 and maks is 100',...
    'Input robot light reading precision between 0 - 100? Lower number is more accurate. Default is 5'};
name='Input for light to music function';
numlines=1;
defaultanswer={'1','10','5'};
answer=inputdlg(prompt,name,numlines,defaultanswer);
    SoundLen = str2double(answer(1))*10;
    if isempty(SoundLen)
        SoundLen = 10;
    end

    RoboSpeed = str2double(answer(2));

```

```

    if or(isempty(RoboSpeed),RoboSpeed > 100)
        RoboSpeed = 10;
    end

    RoboPrec = str2double(answer(3));
    if or(isempty(RoboPrec),RoboPrec > 360)
        RoboPrec = 5;
    end
    RoboPrec = ceil(Dist2Deg(RoboPrec))
end

%% Getting everything ready
[TotLen, SoundLen, RoboSpeed,RoboPrec] = menuCh(); % get user input
motorB.ResetPosition(); % nullstill vinkelteller
motorC.ResetPosition(); % nullstill vinkelteller
lys = GetLight(SENSOR_3); % Get first light reading
screen = get(0,'screensize'); % hent skjermstørrelse for ? stille grafene

% Figur for ? vise tonelengde
figure('Name','Hovedfigur - Press en tast for ? komme tilbake til meny','Position',[screen(3)/8, screen(4)/8,3*screen(3)/8,
2.8*screen(4)/8],'NumberTitle','off');
subplot(3,1,1)
plotl_1=plot((1:(length(Mlen))), Mlen);
set(plotl_1,'Ydata', Mlen , 'Xdata', (1:(length(Mlen))) );
hold on;
title('Lyd data');
ylabel('Lydlengde');

% Figur for ? vise frekvens
subplot(3,1,2)
plotl_2=plot((1:(length(Mtone))), Mtone);
set(plotl_2,'Ydata', Mtone , 'Xdata', (1:(length(Mtone))) );
ylabel('Frekvens');
hold on;

% Figur for ? vise lysm?lingsdata
subplot(3,1,3)
plotl_3=plot((1:(length(lysFilt))), lysFilt);
set(plotl_3,'Ydata', lysFilt , 'Xdata', (1:(length(lysFilt))) );
xlabel('M?lepunkt');
ylabel('Lys');
hold on;

% Move forward with robospeed
motorB = NXTMotor('B','Power',RoboSpeed);
motorC = NXTMotor('C','Power',RoboSpeed);
motorB.SendToNXT();
motorC.SendToNXT();

%% Drive and collect measurments
while LenRead < TotLen;
    data = motorB.ReadFromNXT(); % hent motor B informasjon
    lys(end+1)=GetLight(SENSOR_3); % hent lysm?ling
    lysFilt(end+1)=Lfilter([lysFilt(end),lys(end)], RoboPrec); % filtrer lysm?ling med input presisjon
    deltaLys = lysFilt(end)-lysFilt(end-1); % Finn endring i lysm?ling

    if deltaLys < 0; % Hvis endring i lysm?ling er negativ (et m?rkere parti
starter)
        tempLys(end+1) = lysFilt(end); % registrer da lysm?ling i temp variabel
        lysneg = 1; % sett lysneg til 1 for ? vite at m?ling av gr?tt omr?de
er startet

    elseif deltaLys == 0; % Hvis det ikke er endring i lysm?ling
        if lysneg == 1; % Men man er p? et gr?tt felt
            tempLys(end+1) = lysFilt(end); % registrer lysm?ling
        end
    else
        if lysneg == 1; % Endring er positiv og dette er f?rste gang (hvitt parti
starter)
            distdeg = motorC.ReadFromNXT(); % Les data fra motor C
            Mtone(end+1) = LtoHZ(mean(tempLys)); % Sett lyd tone til gjennomsnitt lysm?linger som er gjort
p? gr?tt parti
            Mlen(end+1)= Deg2Dist(distdeg.Position)*SoundLen; % Sett lengden p? tone til fysisk lengde av gr?tt p?
arket fra motor c m?ling
            motorC.ResetPosition(); % nullstill motorC posisjons m?ler
            tempLys = 0; % T?m temp lysm?lingsvariabel
            lysneg = 0; % Forteller at robot er p? hvitt parti og gr?tt omr?de er
allerede registrert
        end
    end
    LenRead = Deg2Dist(data.Position); % Sjekk total kj?relengde for ? se om det er p? tide ?
stoppe
end

% Stop motorer n?r ark lengde er n?dd
motorB.Stop;
motorC.Stop;

%% Spill av sangen og dans til de herlige tonene
while i < length(Mlen);
    NXT_PlayTone(Mtone(i),Mlen(i)); % Kj?r til alle toner er spilt av
    %Do a boogie to the lovely music % Spill toner p? n?xt
    motorB = NXTMotor('B','Power',danceSpeed*danceDirect); % kj?r motor b i en retning
    motorC = NXTMotor('C','Power',danceSpeed*danceDirect*-1); % kj?r motor c i andre retning
    motorB.SendToNXT(); % send motorinfo til n?xt
    motorC.SendToNXT(); % send motorinfo til n?xt
    danceDirect = danceDirect * -1; % snu motor retninger
    pause((Mlen(i))/1000) % pause i tonelengde for ? la roboten spille ferdig tone
    i=i+1; % ?k teller
end

%% Avslutt n?r sang og dans er ferdig

```

```

% Stop motorer
motorB.Stop;
motorC.Stop;

%% Oppdater figurer
figure(1)
set(plot1_1,'Ydata', Mlen , 'Xdata', (1:(length(Mlen))));
set(plot1_2,'Ydata', Mtone , 'Xdata', (1:(length(Mtone))));
set(plot1_3,'Ydata', lysFilt , 'Xdata', (1:(length(lysFilt))) );

%% Steng koblinger til sensorer og avslutt n?r knapp trykkes p? keyboard
CloseSensor(SENSOR_3);

w = 2;
h=0;
if h==0
    h= msgbox('Press any key to close figures and return to menu','Message','help')
end

while w > 1
    w = waitforbuttonpress;
    if w <= 1;
        % Steng NXT tilkobling
        initNXT();
        % Return to main menu
        % main();
    end
end

end
end

```

9.9 Joy Music

```

function [] = joymusic()
%JOYMUSIC spiller musikk med joystick

%% Initialiserer variabler
run = true; % loop variabel, settes til false for ? avslutte programmet
Mlen = 50; % ms the tone should last for
joyFB = 0; % vektor for forover/bakover bevegelse av joystick
joyS = 0; % vektor for sideveis bevegelse av joystick
Mtone = 0;
%% Initialiserer NXT
initNXT();
handle_NXT = COM_OpenNXT(); % etablerer nytt h?ndtak
COM_SetDefaultNXT(handle_NXT); % setter globalt standard-h?ndtak

%% Initialiserer joystick
joymex2('open',0); % ?pner joystick
joystick = joymex2('query',0); % sp?r etter data fra joystick
JoyMainSwitch = joystick.buttons(1); % Knapp 1, for ? stoppe program

while run
    joystick = joymex2('query',0); % sp?r etter data fra joystick
    joyFB(end+1) = (-joystick.axes(2))/(32.768); % henter joystick posisjon p? "y"-aksen som angir lave toner
    joyS(end+1) = ((joystick.axes(1))/(32.768))*2; % henter joystick posisjon p? "x"-aksen som angir h?ye toner
    if ((joyFB(end)+joyS(end))/2) > 0; % Hvis joystick posisjon st?rre enn 0
        Mtone = LtoHZ((joyFB(end)+joyS(end))/2); % Konverter joystick signal til hz
        NXT_PlayTone(Mtone, Mlen); % Spill tone
    end
    JoyMainSwitch = joystick.buttons(1); % Definer avsluttings program
    if JoyMainSwitch
        run = false; % Hvis knapp trykket avsluttes programmet
    end
end

end

```

9.10 CLAP DRIVE

```
function []=ClapDrive()
%% Oppgave 8 "NXT reaksjon på klapp" fra "Forslag til kreative oppgaver"
%% ING100 Gruppe 1401 "Daniel Løvik" 2014

% Beskrivelse av program:
% Roboten starter å kjøre fremover ved 1 kraftig klapp.
% Lyd og ultralyd verdier plottes fortløpende.
% Ved 2 hurtige kraftige klapp stopper roboten.
% Ved nytt klapp resettes klapptelleren og settes til 1 mens koden kjører
% runden på nytt.

%% Initialiserer NXT
%
COM_CloseNXT all % lukker alle NXT-håndtak
close all % lukker alle figurer
clear all % sletter alle variable
handle_NXT = COM_OpenNXT(); % Ser etter NXT USB enheter
COM_SetDefaultNXT(handle_NXT); % setter globalt standard-håndtak

%% Initialiserer Lyd sensor
OpenUltrasonic(SENSOR_4); % Åpner ultralydsensor..
OpenSound(SENSOR_2, 'DB' ); % Åpner lydsensor og leser data i DB.

%% Div variabler
slyd = [0];% Lydvektor
sUltra = [0];% Ultravektor
tid = [0]; % Tidsvektor
Progstart = true; % Variable for While kontroll
Fremover = NXTMotor('BC'); % Kjører frem
clapcount = 0; % Klappteller
clapreg = false; % Klapregistrering
claptime = [1]; % Tidsvektor for bruk i cputime kode til måling mellom klapp
Start = cputime; % Starttid

while Progstart
    % Plotter nåværende lyd verdier og ultraverdier samt tid ved
    % hjelp av cputime.
    tid(end+1)=cputime-Start;
    slyd(end+1) = GetSound(SENSOR_2);
    subplot(2,1,1)
    plot(tid,slyd);
    title('Måling av lydnivå')
    ylabel({'Lydverdi'})
    xlabel({'Tid'});
    sUltra(end+1) = GetUltrasonic(SENSOR_4);
    subplot(2,1,2)
    plot(tid,sUltra);
    title('Ultralyd måling');
    ylabel({'Ultraverdi'});
    xlabel({'Tid'});

    if GetSound(SENSOR_2) > 600 % Lydverdier i et stille rom er rundt 60.
        % Ved godt Klapp øker verdien til over 600
        Fremover.Power = 20; % Kjører frem med 20% kraft
        Fremover.SendToNXT(); % Sender til NXT

        if ~clapreg % Registrering av klapp + klappetid, sørger for at klapp bare
            % registreres en gang over satt nivå
            clapcount = clapcount +1
            claptime(end+1) = cputime;
            clapreg = true;
        end
        %Hvis det registreres to klapp innenfor 500ms øker klappteller med 2
        if claptime(end) - claptime(end-1) < 0.5
            clapcount = clapcount +2
        end
    else
        if clapreg % Restter clapreg etter et registrert klapp.
            clapreg = false;
        end

        end
        % Etter 1 vanlig klapp og 2 kjappe klapp settes NXT i brems.
        if clapcount == 3
            Fremover.Stop('brake')

            % Ved nytt klapp etter 3 resettes klappteller og starter på 1.
        elseif clapcount > 3
            clapcount = (clapcount - clapcount) +1
        end
        if GetUltrasonic(SENSOR_4) < 25 % Grense for auto stopp av robot ved under 25 cm i fra vegg.
            Fremover.Stop('off');
            Progstart = false; % Hovedløkke stoppes.
        end
    end
end
%%
```



```
% Steng ned sensorer
CloseSensor(SENSOR_2);
CloseSensor(SENSOR_4);

% Close NXT connection.
COM_CloseNXT(handle_NXT);
```

9.11 ULTRA DRIVE

```
function []=UltraDrive()
%% Oppgave 9 "Følg objekt" fra "Forslag til kreative oppgaver"
% ING100 Gruppe 1401 "Daniel Løvik" 2014

% Beskrivelse av program:
% NXT kjører mot et objekt til den er innenfor grenseområdet mellom 23 og
% 25 cm.
% NXT vil da bremse og stå stille så lenge den er i grenseområdet, øker
% avstanden kjører den framover, minker den rygger NXT.
% Kommer objektet under "smertegrensen på 23cm" så rygger roboten.

%% Initialiserer NXT
%
COM_CloseNXT all % lukker alle NXT-håndtak
close all % lukker alle figurer
clear all % sletter alle variable
handle_NXT = COM_OpenNXT(); % Ser etter NXT USB enheter
COM_SetDefaultNXT(handle_NXT); % setter globalt standard-håndtak

%% Sensorer aktiveres.
OpenUltrasonic(SENSOR_4); % Åpner Ultralydsensoren.
OpenSound(SENSOR_2, 'DB' ); % Åpner lydsensoren og leser data i DB

% Diverse oppstarts variabler
slyd = [0]; %Lydvektor
sUltra = [0]; % Ultravektor
tid = [0]; % Tidsvektor
ultrastart = 1; % Kontroll løkke for lys registrering/endring
ProgStart = cputime; % Logger tiden programmet starter
Motorer = NXTMotor('BC' ); % Bruker begge motorene synkront
breakcount = 0; % Hvis 1 = Tvungen stopp av NXT og While løkke

while ultrastart == 1
    % Plotter Ultralyd, lyd og tids verdier underveis.
    sUltra(end+1) = GetUltrasonic(SENSOR_4);
    tid(end+1)=cputime-ProgStart;
    subplot(2,1,2)
    plot(tid,sUltra);
    title('Ultralyd måling');
    ylabel({'Ultraverdi'});
    xlabel({'Tid'});
    slyd(end+1) = GetSound(SENSOR_2);
    figure(1)
    subplot(2,1,1)
    plot(tid,slyd);
    title('Måling av lydnivå');
    xlabel({'Tid'});
    ylabel({'Lydverdi'});

% Ultrastart On
if breakcount == 0;

    if GetUltrasonic(SENSOR_4) >= 80 % Avstand større enn 80cm.
        Motorer.Power = 80; % Kjører med 80% kraft.
        Motorer.SendToNXT();

    elseif (GetUltrasonic(SENSOR_4)>= 25) && (GetUltrasonic(SENSOR_4)<80) % Avstand mellom 25 og 80cm.
        Motorer.Power = GetUltrasonic(SENSOR_4)-10; % Nåværende avstandsverdi minus 10. Sendes som kraft % til NXT.
        Motorer.SendToNXT();
    elseif GetUltrasonic(SENSOR_4)<= 23 %Avstand under 23cm, kjører bakover.
        Motorer.Power = GetUltrasonic(SENSOR_4)-30; % Nåværende avstandsverdi minus 30. Sendes som kraft % til NXT.
        Motorer.SendToNXT();
    else
        Motorer.Stop('brake'); % Bremses

    end
    if GetSound(SENSOR_2) >900; % Ved høy lyd fra rop eller klapp aktiveres "Force stop".
        Motorer.Stop('off'); % Skurr av motor
        breakcount = breakcount +1; % Stopper Ultrastart
        ultrastart = false; % Stopper While løkke
        disp('Force stop aktivert')
    end
end
end

%%
% Steng ned sensorer
CloseSensor(SENSOR_2);
CloseSensor(SENSOR_4);

% Close NXT connection.
COM_CloseNXT(handle_NXT);
end
```

9.12 ULTRA WALL

```
function [] = UltraWall()
%% Egen kreativ oppgave " UltraWall"
% ING100 Gruppe 1401 "Daniel Løvik" 2014

% Beskrivelse av program:

% Roboten kjører fremover helt til Ultralydsensoren finner veggen innenfor
% 40cm, roboten vil da snu 90 grader og kjøre videre og gjenta prosedyren
% ved ny vegg.

%% Initialiserer NXT
%
COM_CloseNXT all           % lukker alle NXT-håndtak
close all                 % lukker alle figurer
clear all                 % sletter alle variable
handle_NXT = COM_OpenNXT(); % Ser etter NXT USB enheter
COM_SetDefaultNXT(handle_NXT); % setter globalt standard-håndtak

%% Initialiserer sensorer
OpenUltrasonic(SENSOR_4); % Starter Ultrasensor.
OpenSound(SENSOR_2, 'DB' ); % Starter Lydsensor.

%%
sUltra = [0]; % Ultravektor
tid = [0]; % Tidsvektor
ultraStart = 1 ;
ProgStart = 1; % Variabel for While kontroll
tStart = cputime; % Logger tiden programmet starter
Motorer = NXTMotor('BC') ;
Venstre = NXTMotor('C', 'Power', 40, 'Tacholimit', 420); % 40% kraft på venstre motor med 90 graders grense for hjul spin.
% Hjulene sin Diameter = 3cm, omkrets = 9.42
Omkrets = 9.42 ;

while ProgStart == 1

    % Plotter ultralydverdier og tid underveis.
    sUltra(end+1) = GetUltrasonic(SENSOR_4);
    tid(end+1)=cputime-tStart;
    plot(tid,sUltra);
    title('Ultralyd måling');

    if GetUltrasonic(SENSOR_4) <= 40 % Vegg under 40 cm i fra, aktiver brems og snu robot 90 grader mot høyre.
        Motorer.Stop('brake');
        Venstre.Power;
        Venstre.SendToNXT();
        Venstre.WaitFor();

    end

    if GetUltrasonic(SENSOR_4) > 40 % Ingen vegg deteksjon, kjører med 30% motor kraft på begge motorer synkront.
        Motorer.Power = 30;
        Motorer.SendToNXT();

    end
    if GetSound(SENSOR_2) > 900; % Ved høy lyd fra rop eller klapp aktiveres "Force stop".
        Motorer.Stop('off'); % Skurr av motor
        ProgStart = ProgStart+1; % Stopper While løkke
        disp('Force stop aktivert')

    end

end

%%
% Steng ned sensorer
CloseSensor(SENSOR_2);
CloseSensor(SENSOR_4);

% Close NXT connection.
COM_CloseNXT(handle_NXT);

end
```

9.13 PLOT TIMER

```
function plotTime()
%Illustrate how much time different ways of plotting takes pr run through the loop.
% Subplot 1: plot() call each time
% Subplot 2: set with X and Y data
% Subplot 3: refreshdata
% End with summary display

%%
% Initiate figure
```

```

screen = get(0,'screensize');
figure('Position',[screen(3)/6, screen(4)/6,screen(3)/1.5, screen(4)/1.5]);
set(gcf, 'KeyPressFcn', @keyDownListener, 'Name', 'Press any key to continue');

%%
% KeyDownListener to stop program any button
function keyDownListener(src, event)
    run = 0;
end

%%
%Initialte variables
run = 1;
plotTime = [0]; % time vector for plot
setTime = [0]; % time vector for set
refreshTime = [0]; % time vector for refresh
loops = [0]; % vector for number of loops
avgPT = [0]; % vector for average plot time
avgST = [0]; % vector for average set time
avgRT = [0]; % vector for average refresh time

%%
%initiate plots
subplot(3,1,1) % set up subplot 1 for plot
hold on;
plotplot = plot(loops,plotTime, '-b');
xlabel('Number of times through the loop/size of time vector');
ylabel('Time to plot in s'); % end setup of subplot 1
subplot(3,1,2) % set up subplot 2 for set
hold on;
setPlot = plot(setTime, loops, '-r');
set(setPlot,'Ydata', setTime, 'Xdata', loops);
xlabel('Number of times through the loop/size of time vector');
ylabel('Time to plot in s'); % end setup of subplot 2
subplot(3,1,3) % set up subplot 3 for refresh
hold on
refreshPlot = plot(refreshTime, loops, '-g');
set(refreshPlot, 'YDataSource', 'refreshTime', 'XDataSource', 'loops');
xlabel('Number of times through the loop/size of time vector');
ylabel('Time to plot in s'); % end setup of subplot 3
drawnow; % force the plots to draw

%%
%main loop, displaying the plot time realtime
tic; % start stop clock
while run
    subplot(3,1,1) % plot function
    time = toc; %time before plot
    plotplot = plot(loops,plotTime, '-b'); % do the plot
    plotTime(end+1) = toc - time; % time after plot
    avgPT(end+1) = sum(plotTime)/(length(plotTime)-1); % calculate average
    title(['Plot average time: ' num2str(avgPT(end)) 's']);
    subplot(3,1,2) % set plots
    time = toc; %time before plot
    set(setPlot,'Ydata', setTime, 'Xdata', loops); % do the plot
    setTime(end+1) = toc - time; % time after plot
    avgST(end+1) = sum(setTime)/(length(setTime)-1); % calculate average
    title(['Set plot average time: ' num2str(avgST(end)) 's']);
    subplot(3,1,3) % refresh plot
    time = toc; % time before plot
    refreshdata(refreshPlot, 'caller') % do the plot
    refreshTime(end+1) = toc - time; % time after plot
    avgRT(end+1) = sum(refreshTime)/(length(refreshTime)-1); % calculate average
    title(['Refresh plot average time: ' num2str(avgRT(end)) 's']);
    loops(end+1) = loops(end) + 1; % update number of loops
    drawnow; % force plots to draw
end

%%
%Summary display
clf; % clear current figure
comAxis = [0 loops(end) -0.0001 (max([avgPT avgST avgRT])+0.0001)]; % set common axis
subplot(3,1,1) % plot average time for plot
plot(loops,avgPT, '-b');
xlabel('Number of times through the loop/size of time vector');
ylabel('Time in seconds');
title(['Average time to plot using plot function: ' num2str(avgPT(end)) 's']);
axis(comAxis);
subplot(3,1,2) % plot average time for set
plot(loops,avgST, '-r');
xlabel('Number of times through the loop/size of time vector');
ylabel('Time in seconds');
title(['Average time to plot using set function: ' num2str(avgST(end)) 's']);
axis(comAxis);
subplot(3,1,3) % plot average time for refresh
plot(loops,avgRT, '-g');
xlabel('Number of times through the loop/size of time vector');
ylabel('Time in seconds');
title(['Average time to plot using refreshdata function: ' num2str(avgRT(end)) 's']);
axis(comAxis);
run = 1;
while run % show until user is ready to continue, controlled by keydownlistener
    drawnow;
end

```

```
%%
% end the program
close(gcf);
end
```

9.14 MATH SHOW

```
function mathShowMain()
% Main function for mathshow program
% Program 1: show numeric integration
% Program 2: show numeric derivation

%%
% Main program
run = 1; %Loop variable
while run
    choice = menu('Choose math function to show: ', 'Integration', 'Derivation', 'Exit');
    switch choice
        case 1 %Show numeric integration
            mathIntShow();
        case 2 %Show numeric derivation
            mathDerShow();
        case 3 %Exit program
            run = 0;
    end
end
end

function mathIntShow()
%Animate numeric integration
% Allow user to define a function of x
% User defines lower and higher limits of the integral
% Subplot 1: Show the function graph
% Subplot 2: Show the current loops addition to the integral
% Subplot 3: Show the integral from the lower limit until the current
% loop
% Subplot 4: Show the integral for the entire defined area
% Subplot 5: Graph for integral of f(x)

%%
%user defines function
userInputs = inputdlg('Define function with variable x: ', 'Define lower limit: ', 'Define higher limit: ', 'User inputs',
1, {' ',' ',' '});
y = char(userInputs(1)); %function to be integrated
lowLim = char(userInputs(2)); %lower limit of integral (in string form)
highLim = char(userInputs(3)); %higher limit of integral (in string form)

%%
% Initiate figure, getting screensize and defining size based upon this.
% In addition add a keydownlistener to the figure and a title with
% instructions
function initialFig()
    screen = get(0,'screensize');
    figure('Position',[screen(3)/6, screen(4)/6,screen(3)/1.5, screen(4)/1.5]);
    set(gcf, 'KeyPressFcn', @keyDownListener, 'Name', 'Press any key to continue');
end

%%
% KeyDownListener to stop program by pressing any button button
function keyDownListener(src, event)
    run = 0;
end

%%
% main part of program
initialFig(); %initiate the figure
try %try setting up and running the program
    hLim = str2num(highLim); %convert higher limit to number
    lLim = str2num(lowLim); %convert lower limit to number
    if hLim < lLim %make sure higher limit is higher than lower limit
        temp = hLim;
        hLim = lLim;
        lLim = temp;
    end
    increment = 0.001; %increment for the initial x vector
    x=lLim:increment:hLim; %define an x vector with steps of 0.001, to be used in initial calculations
    while length(x) > 10000 %code to ensure the x vector does not get to big for the program
        increment = increment + 0.001;
        x=lLim:increment:hLim;
    end
    FofX=eval(y); %convert function from string to matlab expression thus creating a vector based upon the defined x vector
    comAxis=[lLim hLim min(FofX) max(FofX)]; %create a common axis for all plots
    sumInt = 0; %Sum of integral variable, to be used both for showing the sum of the entire integral and from startingpoint
until now
    for i = 2:length(x) %create a sum for the entire integral. (numeric integration)
        sumInt = sumInt + FofX(i)*(x(i)-x(i-1));
    end
    subplot(5,1,1); % set up subplot 1 to show the graph of the function
    plot(x,FofX);
    title(['Function graph: f(x)= ' y]);
    axis(comAxis); %end of subplot 1
    subplot(5,1,4); %set up subplot 4 to show the integral of the entire function
    bar(x,FofX);
    title(['Sum of integral for entire defined area: ' num2str(sumInt)])
```

```

axis(comAxis); %end of subplot 4
subplot(5,1,3); %set up subplot 3 to show the integral as far as the program has come
title('Sum of integral from lower limit (starting point) until now:');
hold on
axis(comAxis); %end of subplot 3
IntofF = 0;
x=lLim; %set x vector to start at lower limit
subplot(5,1,5); % set up subplot 5
intPlot = plot(x,IntofF);
grid on;
title('Integral of f(x): '); %end set up subplot 5
maxTime = hLim - lLim; %define the max for the x vector
sumInt = 0; %reset sum of integral variable to use for starting point until now sum.
run = 1; %variable to run the main loop, stops by keydownlistner
i=1; %counter
tic; %start stop clock
while run %main loop
    t=toc; %get current time on stop clock
    if t <= maxTime %if current time within the defined limits, update plots
        x(end+1) = x(1)+t; %update time (x-vector)
        i=i+1; %update counter
        FofX=eval(y); % recreate the function vector based upon the updated x vector
        currentInt = FofX(i)*(x(i)-x(i-1)); %calculate the current loops integration value
        IntofF(end+1) = IntofF(end) + currentInt; %add the current loops integration value to the sum from starting point
    until now
        subplot(5,1,2); %subplot 2 showing a bar of the current loops integration value
        fill([x(i-1) x(i) x(i) x(i-1)], [0 0 FofX(i) FofX(i)], 'b')
        axis(comAxis);
        title(['deltaX(' num2str(x(end)-x(end-1)) 's)*f(x)= ' num2str(currentInt)]); %end of subplot 2
        subplot(5,1,3); %subplot 3 add the current loops integral bar to the sum from starting point until now
        fill([x(i-1) x(i) x(i) x(i-1)], [0 0 FofX(i) FofX(i)], 'b')
        title(['Sum of integral from lower limit (starting point) until now: ' num2str(IntofF(end))]); %end subplot 3
        subplot(5,1,5) %update subplot 5
        set(intPlot, 'Ydata', IntofF, 'Xdata', x);
        axis([comAxis(1) comAxis(2) (min(IntofF)-0.1) (max(IntofF)+0.1)]); % end update subplot 5
    else %x value exceeds the limits of the integral, reset and start at the beginning
        i=1; % counter reset
        x=lLim; % x reset
        sumInt = 0; % sum reset
        IntofF = 0; % reset integral of f(x)
        subplot(5,1,3) % subplot 3 reset
        cla; %end subplot 3 reset
        tic; % reset stop clock
    end
    drawnow; % force graphics to update
end
catch %setting up/running program failed, give user error message and terminate
    errorMessage=msgbox('Error with function or limits');
    pause(1); % pause for 1 sec to let user see error message
    try % try to delete msgbox if it has not been removed by user
        delete(errorMessage);
    catch
    end
end

%%
close(gcf) %close current figure and end program
end

function mathDerShow()
%Animate numeric derivation
% Allow user to define a function of x
% User defines a starting x and ending x
% Subplot 1: Show the function graph, show the x an y axis in the plot
% Subplot 2: Show the graph of the derivate (deltaY/deltaX)
% Subplot 3: Show the current linear approximation (tangent line)
% Subplot 4: Show all linear approximations

%%
%user defines function
userInputs = inputdlg('Define function with variable x: ', 'Define starting x: ', 'Define highest x: ', 'User inputs', 1, {'','',''});
y = char(userInputs(1)); %function
startXinput = char(userInputs(2)); %Starting x
endXinput = char(userInputs(3)); %highest x

%%
% Initiate figure, getting screensize and defining size based upon this.
% In addition add a keydownlistener to the figure and a title with
% instructions
function initialFig()
    screen = get(0,'screensize');
    figure('Position',[screen(3)/6, screen(4)/6,screen(3)/1.5, screen(4)/1.5]);
    set(gcf, 'KeyPressFcn', @keyDownListener, 'Name', 'Press any key to continue');
end

%%
% KeyDownListener to stop program by pressing any button
function keyDownListener(src, event)
    run = 0;
end

%%
% main part of program

```

```

initialFig(); %initiate the figure
try %try setting up and running program
    startX=str2num(startXinput);
    endX=str2num(endXinput);
    if endX<startX %make sure endX is higher than startX
        temp=endX;
        endX=startX;
        startX=temp;
    end
    increment = 0.001;%increment for the initial x vector
    x=startX:increment:endX; %initial x vector to plot function graph
    while length(x) > 10000 %code to ensure the x vector does not get to big for the program
        increment = increment + 0.001;
        x=startX:increment:endX;
    end
    FofX=eval(y); % convert the input function to a vector using initial x vector
    comAxis=[startX endX min(FofX) max(FofX)]; %create a common axis for all plots
    subplot(4,1,1) %subplot 1: set up to show function graph
    plot(x,FofX,'b',[startX endX], [0 0], 'k', [0 0], [comAxis(3) comAxis(4)], 'k');
    title(['Function graph: f(x)= ' y])
    axis(comAxis); %end subplot 1
    dfdx=0; %vector for the derivated function
    x=startX; %set x vector to start at startX
    subplot(4,1,2) %set up subplot 2
    dfdxPlot = plot(x,dfdx);
    grid on
    hold on
    title('deltaY/deltaX graph') %end se up of subplot 2
    run = 1; % variable to run the main loop, stops by keydownlistener
    i=1; %counter
    tic; %start stop clock
    while run
        x(end+1) = x(1)+toc; % add to x by the time spent by program
        if x(end)<endX % check that within defined x area
            FofX=eval(y); % recreate the function vecotr based upon the updated x vector;
            i = i+1; %update counter
            dfdx(end+1)= (FofX(i)-FofX(i-1))/(x(i)-x(i-1)); %current loop value of the derivation
            subplot(4,1,2) % subplot2 update deltaY/deltaX grap
            set(dfdxPlot, 'Ydata', dfdx, 'Xdata', x);
            axis([comAxis(1) comAxis(2) (min(dfdx)-0.1) (max(dfdx)+0.1)]); % end subplot 2 update
            y1=((dfdx(end)*startX)-(dfdx(end)*x(end)))+FofX(end); % calculate low x using linear approximation
            y2=((dfdx(end)*endX)-(dfdx(end)*x(end)))+FofX(end); % calculate high x using linear approximation
            subplot(4,1,3) %set up / update subplot 3
            plot([startX endX], [y1 y2]);
            grid on
            title('Current Linear Approximation')
            axis(comAxis) % end subplot 3
            subplot(4,1,4) %set up / update subplot 4
            plot([startX endX], [y1 y2]);
            grid on
            hold on
            title('All Linear Approximations')
            axis(comAxis) % end subplot 4
        else
            i=1; %counter reset
            x=startX; %x reset
            dfdx=0; % dfdx reset
            subplot(4,1,2); %subplot 2 reset
            cla;
            dfdxPlot = plot(x,dfdx);%end subplot 2 reset
            subplot(4,1,4); %subplot 4 reset
            cla;
            tic; %reset stop clock
        end
        drawnow;
    end
catch %setting up/running program failed, give user error message and terminate
    errorMessage=msgbox('Error with function or limits');
    pause(1); % pause for 1 sec to let user see error message
    try % try to delete msgbox if it has not been removed by user
        delete(errorMessage);
    catch
    end
end

%%
close(gcf) % close current figure and end program
end

```

9.15 NumJOY

```

function numJoy()
% A function that show numeric integration and derivation from the joystick
%input.
% Subplot 1: Function graph created by joystick input

```

```

% Subplot 2: The current loop numeric integral
% Subplot 3: The sum of the integral including this loop
% Subplot 4: The numeric integral function graph
% Subplot 5: The numeric derivation of the function.
% Subplot 6: The current linear approximation of the function
% Subplot 7: All linear approximation of the function
% Subplot 8: Delta Time

%%
% Initiate figure, getting screensize and defining size based upon this.
% In addition add a keydownlistener to the figure and a title with
% instructions
function initialFig()
    screen = get(0,'screensize');
    figure('Position',[screen(3)/8, screen(4)/8,screen(3)/1.3, screen(4)/1.3]);
    set(gcf, 'KeyPressFcn', @keyDownListener, 'Name', 'Press Esc to end program');
end

%%
% KeyDownListener to stop program by pressing escape button
function keyDownListener(src, event)
    if event.Key == 'escape'
        run = 0;
    end
end

%%
% subfunction to calculate commonAxis
function commonAxis = comAxis()
    commonAxis = [0 (max(timeVec)+5) (min(inputVec)-10) (max(inputVec)+10)];
end

%%
% Initialization
initialFig(); %initiate the figure
inputVec = [0]; % input vector
timeVec = [0]; % time vector
cIntVec = [0]; % current integral vector
sIntVec = [0]; % sum of ingegral vector
derVec = [0]; % derivated vector
deltaT = [0]; % Loop time vector
linAllVec = []; % Vector to display all linear approximations, Y-values
linAllXvec = []; % Vector to display all linear approximations, x-values
subplot(4,2,1) % Initial set up of subplot 1
sub1 = plot(timeVec,inputVec);
title('Function graph created by input: ');
hold on;
grid on; % End of initial setup subplot 1
subplot(4,2,3) % Initial set up of subplot 3
hold on; % End of initial setup subplot 3
subplot(4,2,4) % Initial set up of subplot 4
sub4 = plot(timeVec, cIntVec);
title('Integral function: ');
hold on; % End of initial setup subplot 4
subplot(4,2,5) % Initial set up of subplot 5
sub5 = plot(timeVec, derVec);
title('Derivated function: ');
hold on; % End of initial setup subplot 5
subplot(4,2,6) % Initial set up of subplot 6
sub6 = plot([0,0],[0,0]); % End of initial setup subplot 6
subplot(4,2,7) % Initial set up of subplot 7
sub7 = plot([0,0],[0,0]);
hold on;
title('All Linear Approximations') % End of initial setup subplot 7
subplot(4,2,8) % Initial set up of subplot 8
sub8 = plot([0,0],[0,0]);
hold on;
title('Delta Time') % End of initial setup subplot 8

%%
%main program loop
run = 1;
tic; % Starting stop watch
while run
    inputVec(end+1) = getJoy(); % Get joystick input from getJoy() function
    timeVec(end+1) = toc; % Getting stop watch current time
    cAxis = comAxis(); % create/update a common axis
    deltaT(end+1)=timeVec(end)-timeVec(end-1); % calculate looptime from start of last loop until start of this loop
    cIntVec(end+1) = inputVec(end)*(deltaT(end)); % calculate current integrals value
    sIntVec(end+1) = sIntVec(end) + cIntVec(end); % calculate the sum of the integral
    derVec(end+1) = (inputVec(end)-inputVec(end-1))/(deltaT(end)); % The current derivated
    linVec(1)=(-derVec(end)*timeVec(end))+inputVec(end); % lower y-value for current linear approximation
    linVec(2)=(derVec(end)*cAxis(2)-derVec(end)*timeVec(end))+inputVec(end); % higher y-value for current linear
approximation
    linXvec = [0 cAxis(2)]; % setting an x vector for the current linear approximation
    linAllVec = [linAllVec linVec]; % setting y values for all linear approximations
    linAllXvec = [linAllXvec linXvec]; % setting x values for all linear approximations

    subplot(4,2,1) % Loop update of subplot 1
    set(sub1,'Ydata', inputVec,'Xdata',timeVec);
    axis(cAxis); % set current common axis
    subplot(4,2,2) % Loop update of subplot 2
    fill([timeVec(end-1) timeVec(end) timeVec(end) timeVec(end-1)], [0 0 inputVec(end) inputVec(end)], 'b')
    axis(cAxis) % set current common axis

```

```

title(['Current loop integral: ' num2str(cIntVec(end))]); % update title including the current integral
subplot(4,2,3) % Loop update of subplot 3
fill([timeVec(end-1) timeVec(end) timeVec(end) timeVec(end-1)], [0 0 inputVec(end) inputVec(end)], 'b')
axis(cAxis) % set current common axis
title(['Sum of integral: ' num2str(sIntVec(end))]); % update the title including the sum of the integral
subplot(4,2,4) % Loop update of subplot 4
set(sub4, 'Ydata', sIntVec, 'Xdata', timeVec);
axis([cAxis(1) cAxis(2) (min(sIntVec)-1) (max(sIntVec)+1)]); % set current common x-axis and plot specific y-axis
subplot(4,2,5) % Loop update of subplot 5
set(sub5, 'Ydata', derVec, 'Xdata', timeVec);
axis([cAxis(1) cAxis(2) (min(derVec)-1) (max(derVec)+1)]); % set current common x-axis and plot specific y-axis
subplot(4,2,6) % Loop update of subplot 6
set(sub6, 'Ydata', linVec, 'Xdata', linXvec);
title('Current Linear Approximation')
axis(cAxis) % set current common axis
subplot(4,2,7) % Loop update of subplot 7
set(sub7, 'Ydata', linAllVec, 'Xdata', linAllXvec);
axis(cAxis) % set current common axis
subplot(4,2,8) % Loop update of subplot 8
set(sub8, 'Ydata', deltaT, 'Xdata', timeVec);
axis([cAxis(1) cAxis(2) (min(deltaT)-0.1) (max(deltaT)+0.1)]); % set current common x-axis and plot specific y-axis
drawnow; % force graphics to update
end

close(gcf) % close current figure before exiting program
end

function joyPos = getJoy()
%The sum of joystick axis 2 and return it as output
% Initialize joystick, query, set output value as sum and close
% connection
% Initialize joystick
joymex2('open',0);
joyPos = 0;
for i=1:3
% Query joystick
a = joymex2('query',0);
% Set output value as axis 2
joyPos = joyPos + a.axes(2);
end
joyPos/3;
if joyPos < 5000 && joyPos > -5000
joyPos = 0;
end
% close connection
clear joymex2
end

```


10 APPENDIKS C – MATLAB FUNKSJONS KODE

10.1 AUTOFUNC

```
function [motorB,motorC] = Autofunc(avvikL)
%Beregner pådrage til hver motor ut fra lysverdi avvik fra nullpunkt.
% Gir ut motorB og motorC pådrag som vektor
temp = 0.23*abs(avvikL); % bruker 22% av absolutt verdi til avviket.
Autospeed = 13;
TopValue = 50;
BotValue = -70;

if avvikL > 0
    motorB = Autospeed - temp;
    motorC = Autospeed + temp;
elseif avvikL < 0
    motorB = Autospeed + temp;
    motorC = Autospeed - temp;
else
    motorB = Autospeed;
    motorC = Autospeed;
end

if motorB > TopValue
    motorB = TopValue;
elseif motorB < BotValue;
    motorB = BotValue;
end

if motorC > TopValue
    motorC = TopValue;
elseif motorC < BotValue;
    motorC = BotValue;
end
motorB = floor(motorB);
motorC = floor(motorC);
```

10.2 DEG2DIST

```
function [out] = Deg2Dist(deg)
WheelCirc = 56*pi; %56mm
out = WheelCirc*(deg/360);
end
```

10.3 DIST2DEG

```
function [out] = Dist2Deg(dist)
WheelCirc = 56*pi;
out = (dist/WheelCirc)*360;
end
```

10.4 DERIVFUNK

```
function out = derivFunk(x,y)
%deriverer endring i y delt på endring i x
% y: vektor
% dy: endring mellom siste og nest siste verdi i y
% x: vektor
% dx: endring mellom siste og nest siste verdi i x
dy=y(end)-y(end-1);
dx=x(end)-x(end-1);
if dx==0
    dx=0.01;
end
out=dy/dx;
end
```

10.5 FILTJOY

```
function out = filtJoy(in)
%Filter joystick input
% less than 2 = 0
% 40% from new value, 60% from old
if length(in) > 1
    if in(end) < 2 && in(end) > -2
        out = 0;
    else
        out = 0.4*in(end)+0.6*in(end-1);
    end
else
    out = 0;
end
```

```
end
```

10.6 FILTLYS

```
function out = filtLys(in)
%Filtrerer en input vektor til et tall ut
%
temp = in(end)-in(end-1);
% if change is less than +-2 then take last value
if temp < 2 && temp > -2
    out = in(end-1);
% filter joy input
else
    out = 0.6*in(end)+0.4*in(end-1);
end
end
```

10.7 INITNXT

```
function [] = initNXT()
COM_CloseNXT all           % lukker alle NXT-h?ndtak
close all                  % lukker alle figurer
clear all                  % sletter alle variable
end
```

10.8 INTFUNK

```
function intOut = intFunk(x,y)
%integrer med hensyn på x vektorene x og y rundt nullpunkt np.
% x: tid
% y: verdi
% sum: summen intergralet x-1.
dt=x(end)-x(end-1);
intOut=y(end-1)*dt;
end
```

10.9 LFILTER

```
function [out] = Lfilter(in, Prec)
%Filtrerer en input vektor til et tall ut
temp = in(end)-in(end-1);
% if change is less than +-Prec then take last value instead
if temp < Prec && temp > -Prec
    out = in(end-1);
% else take the new light reading
else
    out = in(end);
end
end
```

10.10 LtoHZ

```
function [out] = LtoHZ(l)
x= 13800/1023;
out = ((x * l)+200)/2;
if out > 14000;
    out = 14000;
end

if out < 200;
    out = 200;
end
end
```

10.11 MOTORPAADRAG

```
function [motorB,motorC] = motorPaadrag(joyFB,joyS)
%Beregner pådrage til hver motor ut fra y og x akse posisjon til joystick
% Gir ut motorB og motorC pådrag som vektor
temp = 0.5*abs(joyS); % halve abseluttverdien til sideveis posisjon
if joyS > 0
    motorB = joyFB - 0.5*temp; %Bidraget fra sideveis pos ytterligere redusert
    motorC = joyFB + temp;
elseif joyS < 0
    motorB = joyFB + temp;
    motorC = joyFB - 0.5*temp; %Bidraget fra sideveis pos ytterligere redusert
else
    motorB = joyFB;
    motorC = joyFB;
end
```

```
if motorB > 100
    motorB = 100;
elseif motorB < -100;
    motorB = -100;
end

if motorC > 100
    motorC = 100;
elseif motorC < -100;
    motorC = -100;
end
motorB = floor(motorB);
motorC = floor(motorC);
end
```

10.12 RETFUNK

```
function out = retFunk(in,p)
%Finner rettningsendringer dersom ingen endring
if in<0
    out=-1;
elseif in>0
    out=1;
else
    out=p;
end
end
```