11/21/2014

Matlab & LEGO NXT

Anvendt matematikk & fysikk

Prosjekt i ING100 Ingeniørfaglig innføringsemne høsten 2014



|  |  |  |
| --- | --- | --- |
| Gruppenummer: | **1401** | |
| Medlemmer: | **Helge Johannessen Bjorland**  **Espen Ro Eliassen** | **228290**  **795934** |
| **Daniel Lovik** | **225039** |
| **Anders Svalestad** |  |
|  |  |



Innhold

[2 Sammendrag 3](#_Toc403773580)

[3 Prosjektgjennomføringen 7](#_Toc403773581)

[3.1 Mål 7](#_Toc403773582)

[3.2 Arbeidsform 7](#_Toc403773583)

[3.3 Metodikk 8](#_Toc403773584)

[3.3.1 Utstyr 8](#_Toc403773585)

[3.3.2 programmering 8](#_Toc403773586)

[4 Hoved Program 8](#_Toc403773587)

[4.1 Numerisk Integrasjon 8](#_Toc403773588)

[4.2 Numerisk Derivasjon 8](#_Toc403773589)

[4.3 Filtrering 8](#_Toc403773590)

[4.4 Manuell kjøring 9](#_Toc403773591)

[4.5 Automatisk kjøring 10](#_Toc403773592)

[4.6 Tillegg i obligatorisk del 10](#_Toc403773593)

[5 Kreative program 11](#_Toc403773594)

[5.1 Spill 11](#_Toc403773595)

[5.1.1 CANNON GAME 11](#_Toc403773596)

[5.1.2 Reaction 13](#_Toc403773597)

[5.2 Musikk 14](#_Toc403773598)

[5.2.1 Piano 14](#_Toc403773599)

[5.2.2 Gray Music 14](#_Toc403773600)

[5.2.3 Joy Music 14](#_Toc403773601)

[5.3 Robofun 15](#_Toc403773602)

[5.3.1 Lyd Sensor Clap 15](#_Toc403773603)

[5.3.2 Ultra Hand Follow 15](#_Toc403773604)

[5.3.3 Ultra Wall 15](#_Toc403773605)

[5.4 Matematiske funksjoner og plotting 16](#_Toc403773606)

[5.4.1 PLOT TIMER 16](#_Toc403773607)

[5.4.2 MATH SHOW¨ 17](#_Toc403773608)

[5.4.3 NUMJOY 19](#_Toc403773609)

[6 Konklusjon 21](#_Toc403773610)

[6.1 Manuell Kjøring 21](#_Toc403773611)

[6.2 Automatisk Kjøring 21](#_Toc403773612)

[6.3 Tillegg til obligatorisk del 21](#_Toc403773613)

[6.4 Cannon Game 21](#_Toc403773614)

[6.5 Reaction 21](#_Toc403773615)

[6.6 Piano 21](#_Toc403773616)

[6.7 Gray Music 21](#_Toc403773617)

[6.8 Joy Music 21](#_Toc403773618)

[6.9 Lyd Sensor Clap 21](#_Toc403773619)

[6.10 Ultra Hand Follow 21](#_Toc403773620)

[6.11 Ultra Wall 21](#_Toc403773621)

[6.12 Plot Timer 21](#_Toc403773622)

[6.13 Math Show 21](#_Toc403773623)

[6.14 NumJoy 21](#_Toc403773624)

[7 Referanser 22](#_Toc403773625)

[8 Appendix A - Illustrasjoner 23](#_Toc403773626)

[8.1 Obligatorisk del 23](#_Toc403773627)

[8.2 CANNON GAME 23](#_Toc403773628)

[8.3 mathShow 25](#_Toc403773629)

[8.4 NUMJOY 27](#_Toc403773630)

# Sammendrag

Skriv slik at leseren får lyst til å lese mer om programmene i selve rapporten.

1. Manuell Kjøring
2. Automatisk Kjøring
3. Tillegg til obligatorisk del
4. Cannon Game
5. Reaction
6. Piano
7. Gray Music
8. Joy Music
9. Lyd Sensor Clap
10. Ultra Hand Follow
11. Ultra Wall
12. Plot Timer
13. Math Show
14. NumJoy

DENNE DELEN ER KUN TIL INFO OG SKAL SLETTES Struktur på program beskrivelse:

**HENSIKT**

**OPPBYGNING**

**UTFORDRINGER**

**KONKLUSJON**

# Prosjektgjennomføringen

## 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.

## 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 en 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 tisplan 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.

## Metodikk

### 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.

### 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 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.

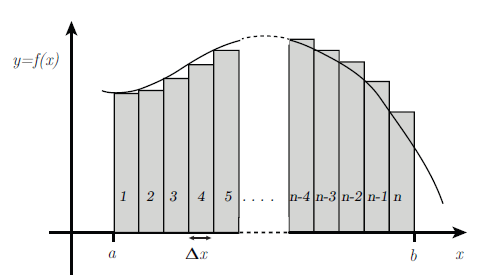
# Hoved Program - Main

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.

## Numerisk Integrasjon

For å finne ut hvor nøyaktig roboten kjørte brukte vi Eulers forovermetode 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:

1. function intOut = intFunk(x,y)
2. %integrer med hensyn på x vektorene x og y
3. dt=x(end)-x(end-1);
4. intOut=y(end-1)\*dt;
5. end

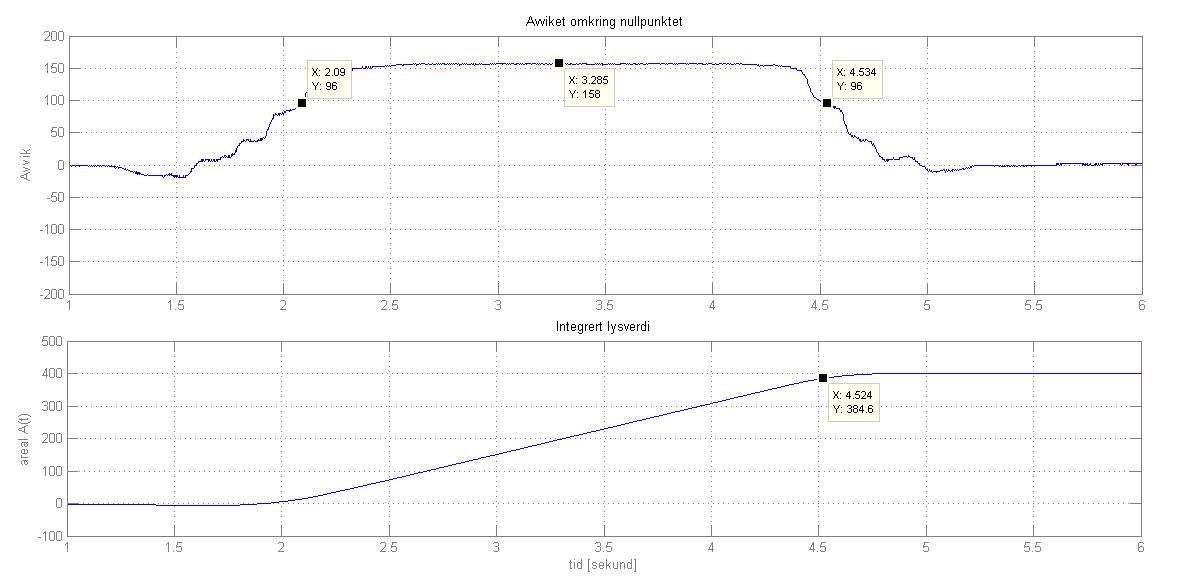
Man ser her at funksjonen intFunk lagrer siste endring i x verdi (tiden) i variabelen dt. Forrige y verdi blir multiplisert med dt for å gi integralet, eller arealet av en stolpe som vist i Figur 1.

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

1. avvikL(end+1)=lysFilt(end)-lysNp;
2. avvikA(end+1)=intFunk(tid,avvikL)+avvikA(end);
3. 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:

Endring i tid x = 4,53 – 2,09 = 2,44

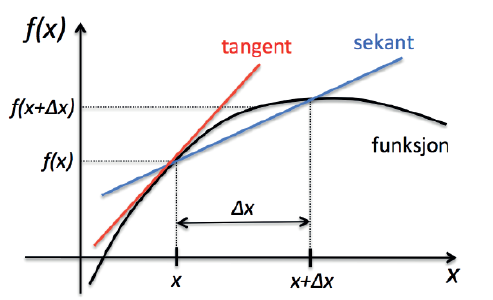
Avvik = 158

A (t) = 2,44 \* 158 = 385

Dette stemmer bra med integrert lysverdi som man kan se i Figur 2 er avrundet lik 385. Den komplette figuren som også viser lysmåling fra verifiseringen finnes i Figur 4 i Appendiks.

## 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

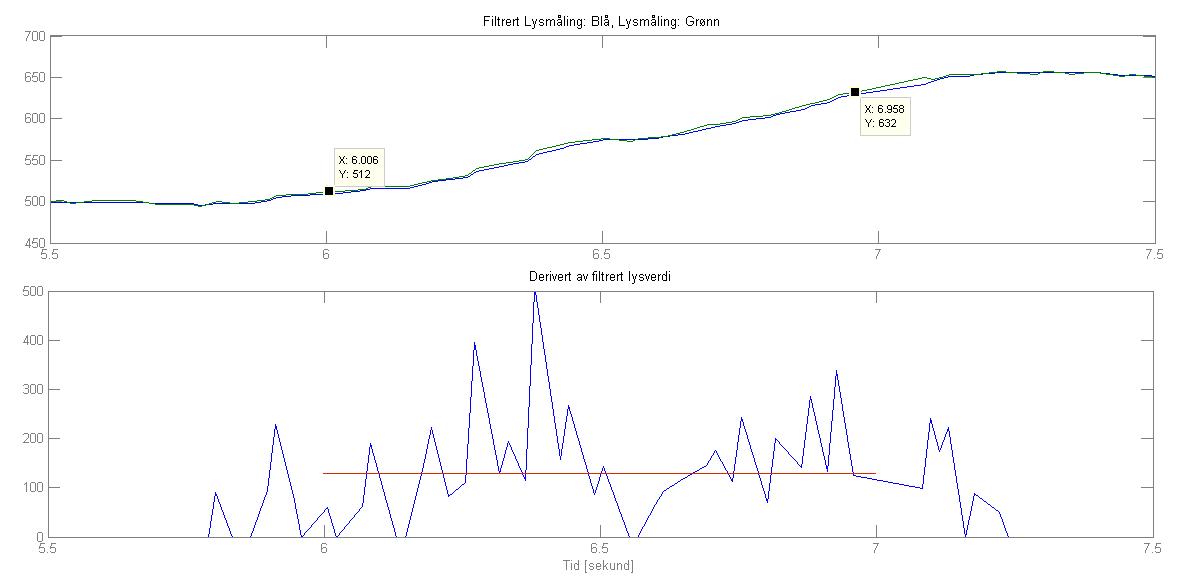
Funksjonen derivFunk brukte for å regne ut den deriverte og ble skrevet som følger:

1. function out = derivFunk(x,y)
2. %deriverer endring i y delt på endring i x
3. dy=y(end)-y(end-1);
4. dx=x(end)-x(end-1);
5. if dx==0
   1. dx=0.01;
6. end
7. out=dy/dx;
8. 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.

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

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



Figur 4 - derivasjons verifisering

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.

## Filtrering

Ta med kode og figurer som viser hvordan filtreringen fungerer. Her kan dere inkludere mange forskjellige filtervarianter og vise at dere har testet og forstått filtrering. Knytt gjerne denne filtreringen opp mot resultatene fra numerisk integrasjon og derivasjon ved å vise hva som skjer dersom man filtrerer *før* integrasjonen/derivasjonen.

1. function out = filtLys(in)
2. %Filtrerer en input vektor til et tall ut
3. %
4. temp = in(end)-in(end-1);
5. % if change is less than +-2 then take last value
6. if temp < 2 && temp > -2
7. out = in(end-1);
8. % filter joy input
9. else
10. out = 0.6\*in(end)+0.4\*in(end-1);
11. end
12. end
13. lys(end+1)=GetLight(SENSOR\_3);
14. lysFilt(end+1)=filtLys([lysFilt(end),lys(end)]);

## Manuell kjøring

Ta med kode og figurer som viser hvordan manuell kjøring av Legoroboten fungerer totalt sett.

Helt kjapt så er vel mitt sammendrag slik:

1. Det er lagt innkode for å velge plotfrekvens, dette for å gjøre det mulig å kjøre uforstyrret av data plot som er ressurskrevende og dermed øker løkketiden i hovedløkken.

2. Først opprettes kobling mot NXT og Joystick, variabler og figurer initialiseres.

3. tiden siden oppstart legges inn i siste element i tidsvektoren

4. en beregner tiden på forige løkkegjennomkjøring ved å ta tiden i starten av denne løkken å trekker fra tiden i starten av forige løkke. Denne skal brukes som delta tid (x) i kalkulasjoner.

5. lyset leses inn og filtreres basert på siste lysverdig og siste filtrerte lysverdig.

6. avviket for lyset fra nullpunkt regnes ut (denne kunne gjerne ligget under les lys sensor)

7. 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 sist verdi av integralet for å få hele integral summen.

8. tilsvarende integrasjon gjøres for en egen vektor, men denne bruker absolutt verdien av løkke integralet for dermed å få totalt avvik fra nullpunkt.

9. derivasjonen gjøres så i en egen funksjon ved bruk av en delta tid og delta y som regnes fra endringen i lyset verdi. (her ligger litt ekstra kode for å unngå ikke kalkulerbare verdier)

10. rettningen avgjøres i egen funksjon og legges i egen vektor, obs skulle være noe kode for at det må være en endring fra forige retning, dersom endringen gir 0 skal rettning beholdes lik den foregående.

11. verdien i konkurransen utregnes fra oppgitt formel og legges til slutten av en egen vektor.

12. joystickens verdier leses inn i egne vektorer og justeres for initiel nullstillingsverdi.

13. denne verdien kjøres gjennom et filter og oppdateres.

14. en bruker retningen til å sette variabler for visning av dette.

15. pådraget til hver av motorene regnes ut i egen funksjon basert på joystickens filtrerte innverdier og pådraget settes.

16. henter data fra hver av motorene for å avgjøre hastighet. dette gjøres ved å se på gjennomsnitts strekning motorene har beveget seg. som legges inn i egne vektorer ( dette kunne kanskje vært gjort til en egen funksjon)

17.plott frekvensen avgjør om det skal plottes (ved bruk av handles) eller om en bare skal oppdatere telleren som sjekker mot plott frekvens grensen.

18. dersom det plottes (se forslag til ny kode), plottes de 100 siste verdiene eller alle om det er ferre.

19. grafikk tvinges til å oppdateres

20. sjekker om koden skal avsluttes eller ny løkke skal kjøres, og prosessen startes eventuelt på nytt på 3.

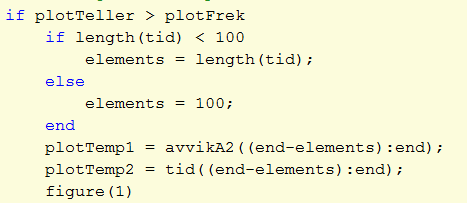
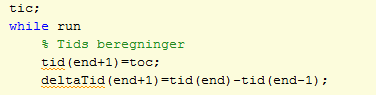
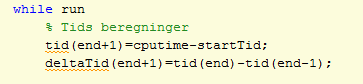
21. når en ikke ønsker å kjøre lengre (løkken avsluttes) stoppes motorene. sensor koblinger stegnes og joystick slettes.

22. figurer fra kjøring stenges og nye figurer basert på all data som har blitt tatt vare på.

23. 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.

## Automatisk kjøring

## Tillegg i obligatorisk del



## Konklusjon

# Kreative program

## Spill

### CANNON GAME

Ansvarlig: Espen Ro Eliassen

**HENSIKT**

**OPPBYGNING**

Programmet er skrevet i en funksjon som igjen inneholder under funksjoner. 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, gravitasjonsens akselerasjon og start fart til prosjektilet. Se seksjon “Constants”.

Etter konstanter følger initialisering av variabler. Disse inneholder både betingelses variabler for å vise forskjellige deler av programmet og variabler for mattematiske 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 skuud (“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.

FIGUR1 Appendix

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 flerspiller, 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 strekkers 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 under funksjoner. 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” variablen til 1.

Før selve hovedprogrammet/-løkken kommer det flere underfunksjoner som har forskjellige oppgaver. De to neste på listen er updateLongestShotHS og updateTotalLenghtHS. Disse funksjonene svært lik funksjonalitet. Den første oppdaterer lengste skudd listen, mens den siste oppdatere 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.

FIGUR2 Appendix

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°. Plaseringen til denne linjen når spiller trykker mellomrom knappen angir vinkelen prosjektilet 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 chartratio 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 variablen 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 prosjektilet 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 prosjektilet starter på bakken altså y=0. Lengden på skudded 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 prosjektilet animeres via to og to punkter i flyvebanen. 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. Logikkmessig er det lite spennende som skjer her. 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.

FIGUR 3 Appendix

Selve hovedløkken til programmet er egentlig 2 svært like løkker. Det er en for enspiller og en for flerspiller. Forskjellen ligger i noe ekstra logikk for å håndtere at det flere spillere. Derfor er det kun flerspiller 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 begyner 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 variablen 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.

FIGUR 4 APPENDIX

Deretter kjøres en løkke som går igjennom summene til alle spillerne, avgjør hvem som vant og setter opp spill detaljer for hver spiller. 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.

FIGUR 5-6 APPENDIX

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 ideel situasjon og kalle opp funksjonen inne i seg selv. Men i dette tilfellet anses muligheten for at noen klarer å sitte å spille sammenhengende nok til å klar å bruke opp minne på en moderne maskin som svært usannsynlig. Et alernativ 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**

**KONKLUSJON**

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.

### Reaction

**Ansvarlig: Anders Svalestad**

**HENSIKT**

**OPPBYGNING**

**UTFORDRINGER**

**KONKLUSJON**

## Musikk

### 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.

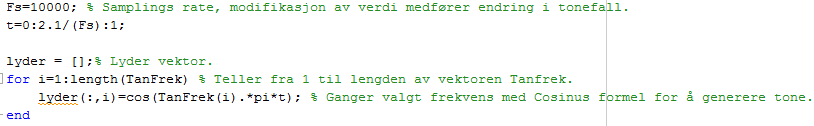
**OPPBYGNING**

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

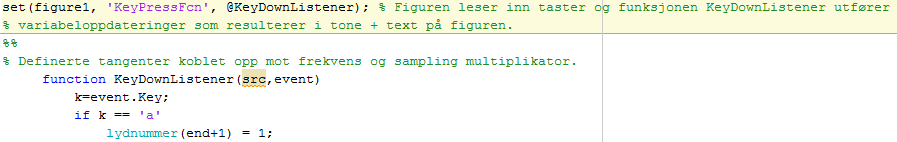


Frekvensene og tangentnavn er hentet i fra wikipedia sin artikkel om piano tangenter og frekvenser.

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.



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



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

**Dette brukes videre i følgende snutt:**



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

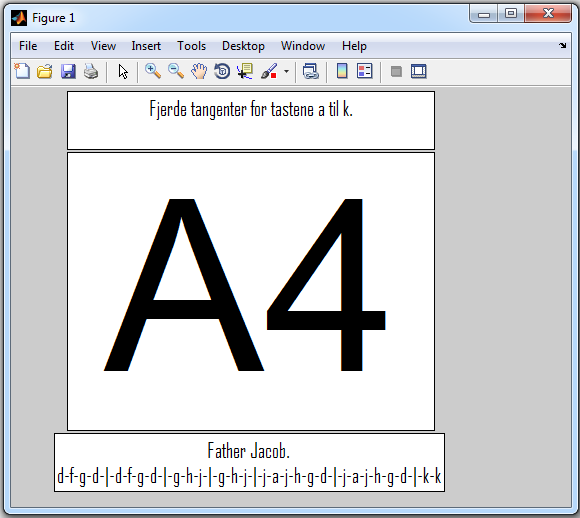
**Frekvensen ganges inn med cosinus formelen og tonen blir spilt av.**

**Tangentnavnet vises på figuren i et annotation vindu.**



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

**Under er en illustrasjon** Figur 5 **etter at tasten «a» blitt trykket på:**



Figur 5 - Inntastet 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 tangenten 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.**

**KONKLUSJON**

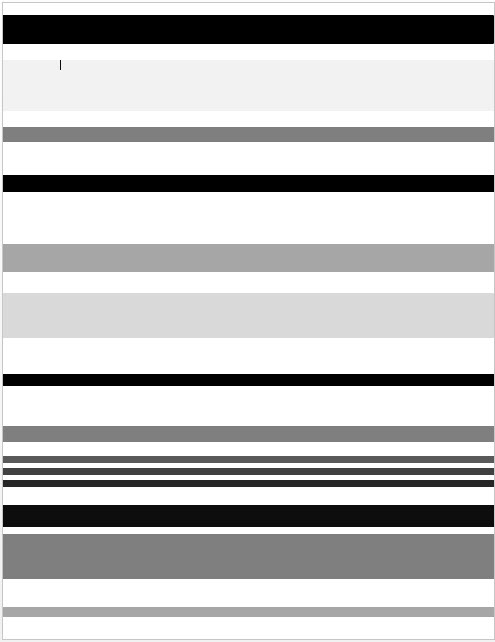
**Programmet virket etter hensikt og resultatet ble tilfredsstillende med tanke på tonefall og funksjonaliteten til «Pianoet». En videreutvikling av programmet til å støtte alle 88 tangenter og en mer piano lignende visualisering av figuren kunne vært en ide for et fremtidig prosjekt.**

### Gray Music

**Ansvarlig: Helge Bjorland**

**HENSIKT**

**Programmet bruker lysmåler for å lese gråtone striper separert med hvite striper på ark av valgfri størrelse (se** Figur 5**). Gråtonen på arket representerer en tone og den fysiske lengden av stripen på arket representerer tidslengden hver tone skal spilles.**



Figur 6 - 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.**

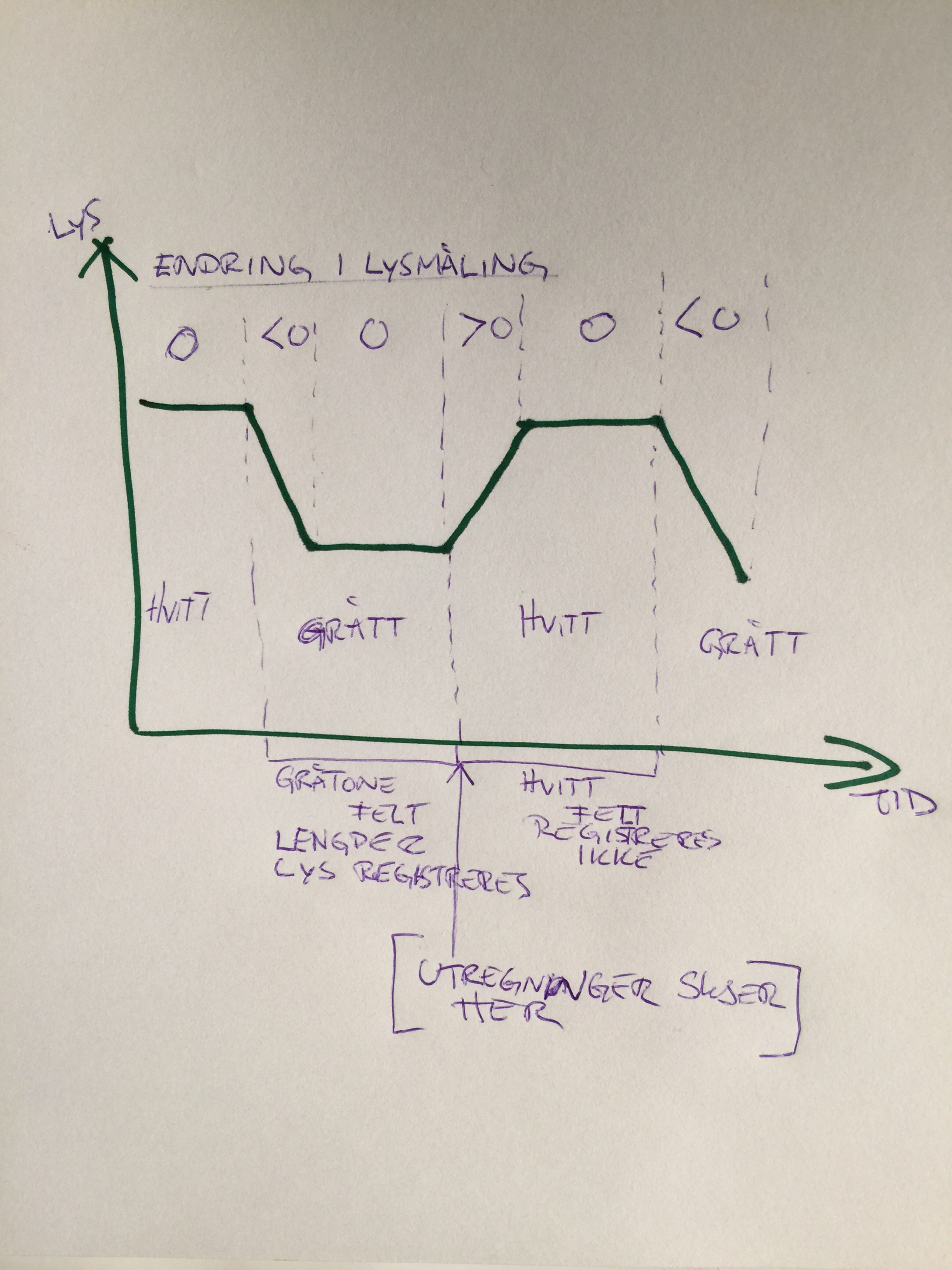
**Det første som ble gjort for å løse denne oppgaven var å tegne en illustrasjon av hvordan algoritmen skulle fungere. Denne er vis i** Figur 6 **under.**

4)

3)

2)

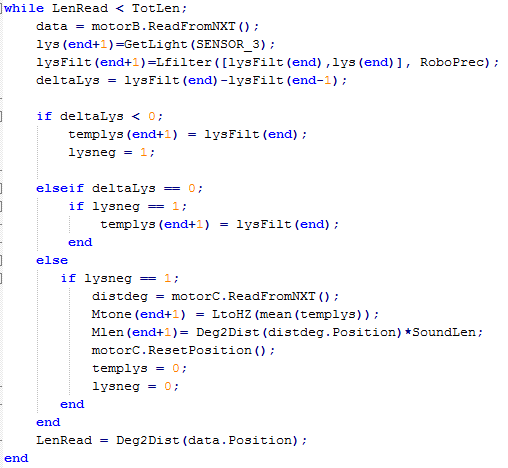
1)



Figur 7 - 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.**



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.

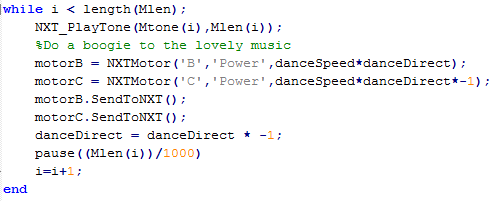
function [out] = Deg2Dist(deg)

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.



**UTFORDRINGER**

**KONKLUSJON**

### Joy Music

**Ansvarlig: Helge Bjorland**

**HENSIKT**

**OPPBYGNING**

**UTFORDRINGER**

**KONKLUSJON**

## Robofun

### 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 bremser den. Ultralydsensoren er også tatt i bruk slik at roboten stopper av seg selv når den kommer innenfor en forhåndsdefinert grense fra veggen.**

**OPPBYGNING**

**UTFORDRINGER**

**KONKLUSJON**

### 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:**



**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 vekke den er fra objektet.**

**For å rygge gjøres følgende:**



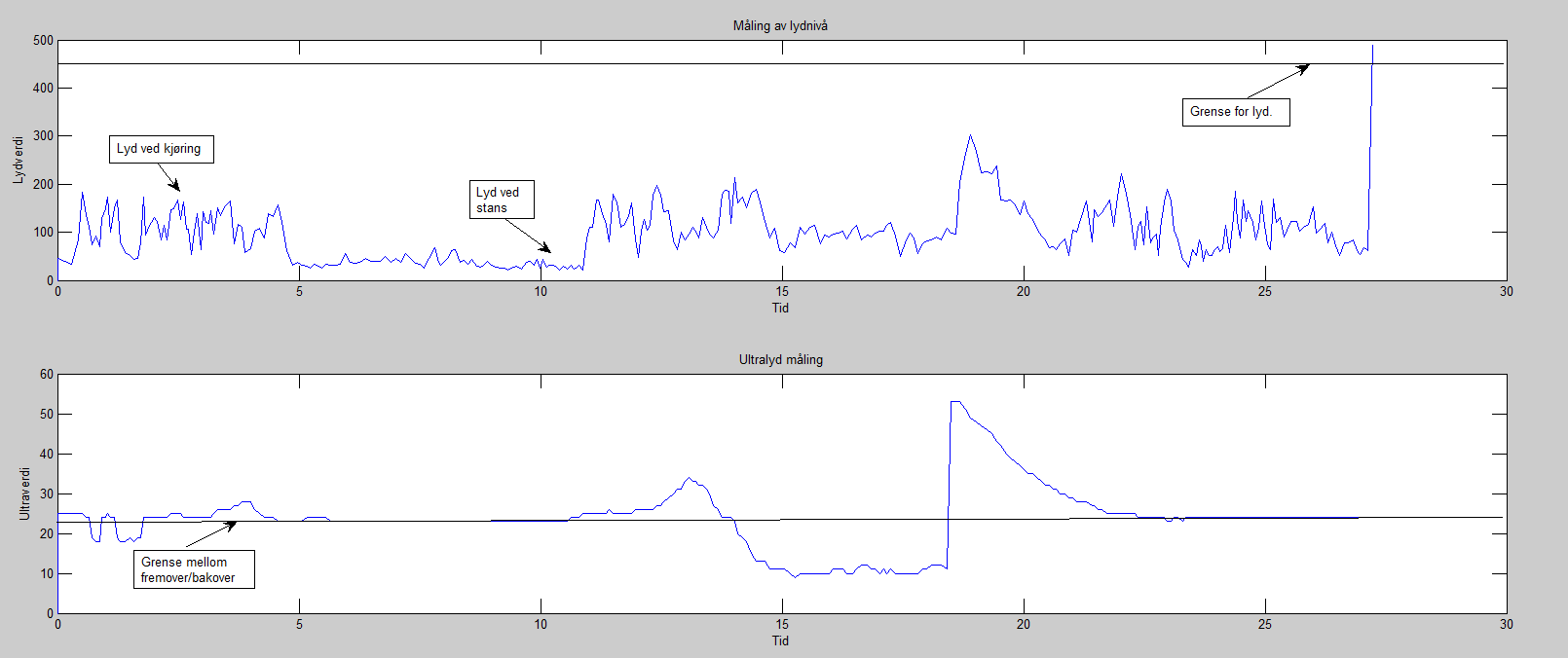
**Når objektet er under satt grense område på 23-25 cm så rygger roboten. Det er satt en minus verdi på 30 her for å få roboten til en negativ verdi ut fra grense området.**

**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(Figur 7) 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 begynnte 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 8 - Grafer under kjøring.

**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.**

**KONKLUSJON**

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.

### Ultra Wall

**Ansvarlig: Daniel Lovik**

**HENSIKT**

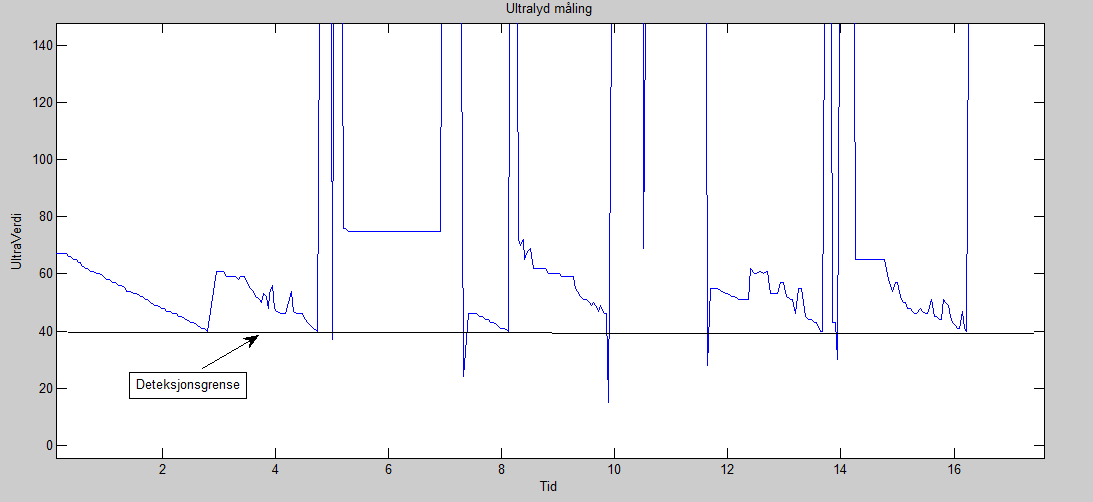
**Programmet tar i bruk ultralydsensoren til NXT’en for å finne en hindring, for eksempel en vegg eller person. Når sensoren finner hindringen innenfor angitt avstand vil roboten snu 90 grader og kjøre videre. For å stoppe programmet brukes lydsensoren og et høyt klapp.**

**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 8**) under ved deteksjonsgrensen:**



Figur 9 - Ultralyd graf.

**Når ultralydsensoren er under deteksjonsgrensen stopper fremover kjøringen og venstre motor gir nok kraft med forhåndsdefinert power ogTacholimit 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.**

**KONKLUSJON**

**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.**

## Matematiske funksjoner og plotting

### 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.

**OPPBYGNING**

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

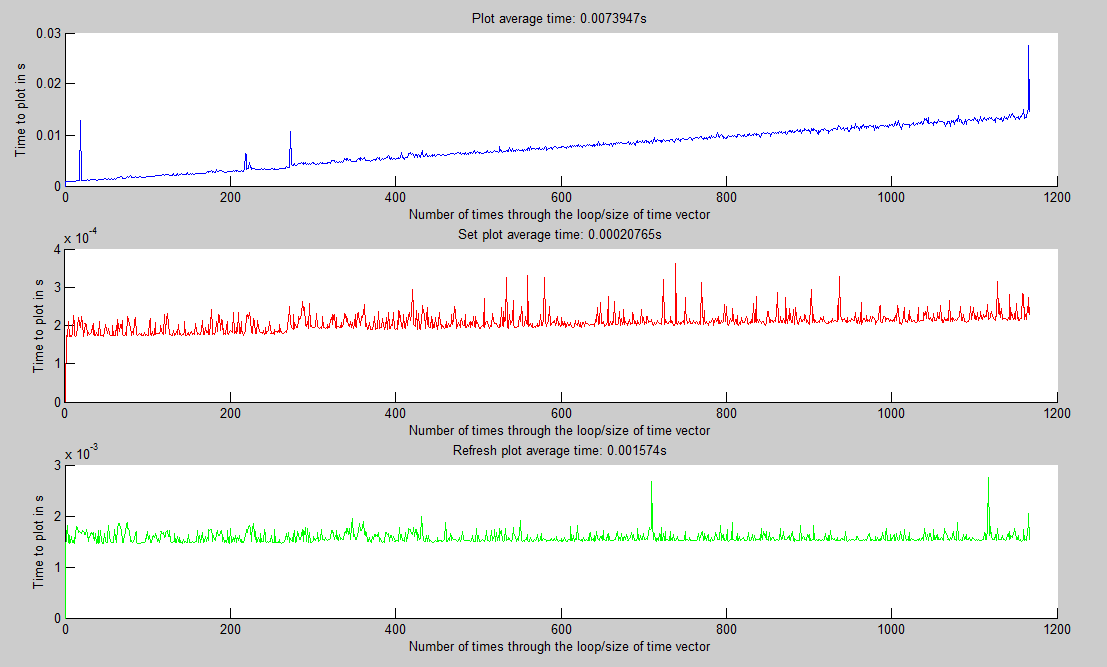
Deretter initialiseres programspesifikke data. Dette er vektorer for tid og gjennomsnitts tid for de forskellige måtene å plotte på. En vektor for antall ganger gjennom hoved løkka og variabel for styring av hoved løkka.

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 10 - 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**

x

**KONKLUSJON**

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.

### MATH SHOW¨

Ansvarlig: Espen Ro Eliassen

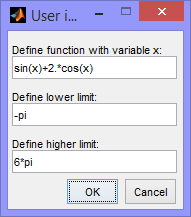
**HENSIKT**

**OPPBYGNING**

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

mathIntShow-funksjonen viser nummerisk 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 dynamsik ut fra skjermstørrelse og en keyDownListener funksjon registeres som figurens “KeyPressFcn”. Se cannongame for mer om dette. Instrukser 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 intiialiserings 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 definers 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 hovedløkken 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 innstrengen fra burker 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 nummerisk 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,alstå må det være 4 punkter. For X blir dette forige x, denne x, denne x og forige 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 resetes stoppeklokken.

Selve hovedløkken 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 avsluter 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 linæer approksimasjon og 4: alle linæer approksimasjoner.

For uten de matematiske forskjellene på de forskjellige plottene har funksjonene en forskjell i logikk. I stedenfor 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**

**KONKLUSJON**

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

### NUMJOY

Ansvarlig: Espen Ro Eliassen

**HENSIKT**

**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 nåævende linæer approksimasjonen, 7: alle linæer approksimasjoner og 8: forandringen i tid fra forige 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. Innhentingen av den eksterne dataen er også flyttet ut til en egen funksjon getJoy. Det er kun et sted denne brukes og den kan enkelt sikftes 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**

**KONKLUSJON**

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

# Konklusjon

Skriv noen setninger som oppsummerer programmene og hvordan de virker på formen:

Problemstilling:

Løsning:

Fungerte det:

## Manuell Kjøring

## Automatisk Kjøring

## Tillegg til obligatorisk del

## Cannon Game

## Reaction

## Piano

## Gray Music

## Joy Music

## Lyd Sensor Clap

## Ultra Hand Follow

## Ultra Wall

## Plot Timer

## Math Show

## NumJoy

# Referanser

[1] [www.mathworks.se](http://www.mathworks.se/)

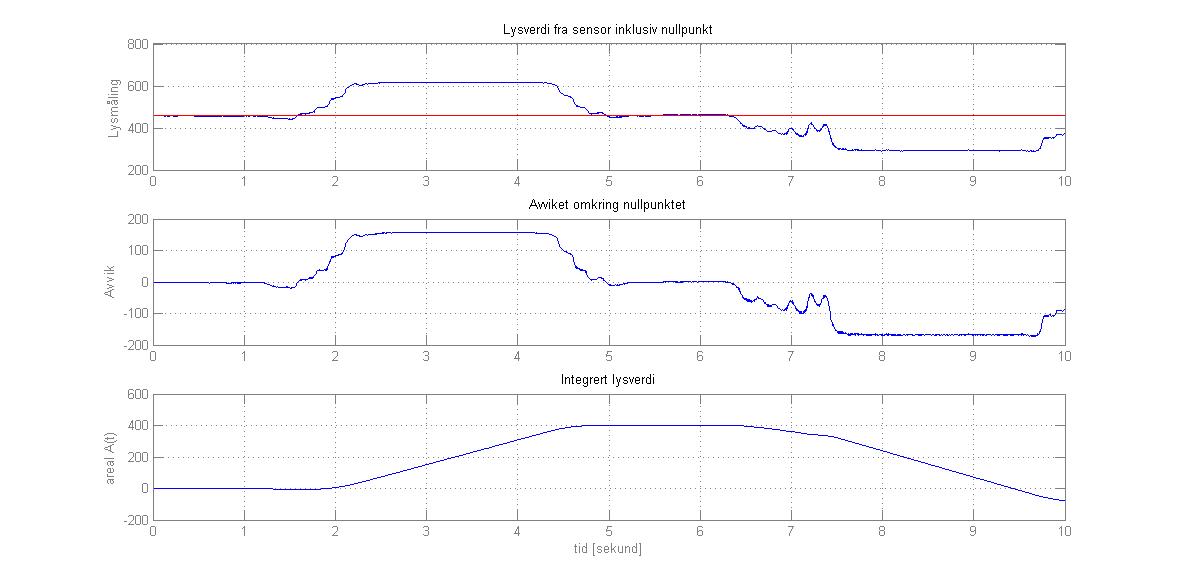
<http://en.wikipedia.org/wiki/Euler_method#mediaviewer/File:Euler_method.svg>

<http://upload.wikimedia.org/wikipedia/commons/b/b1/Composite_trapezoidal_rule_illustration.svg>

# Appendix A - Illustrasjoner

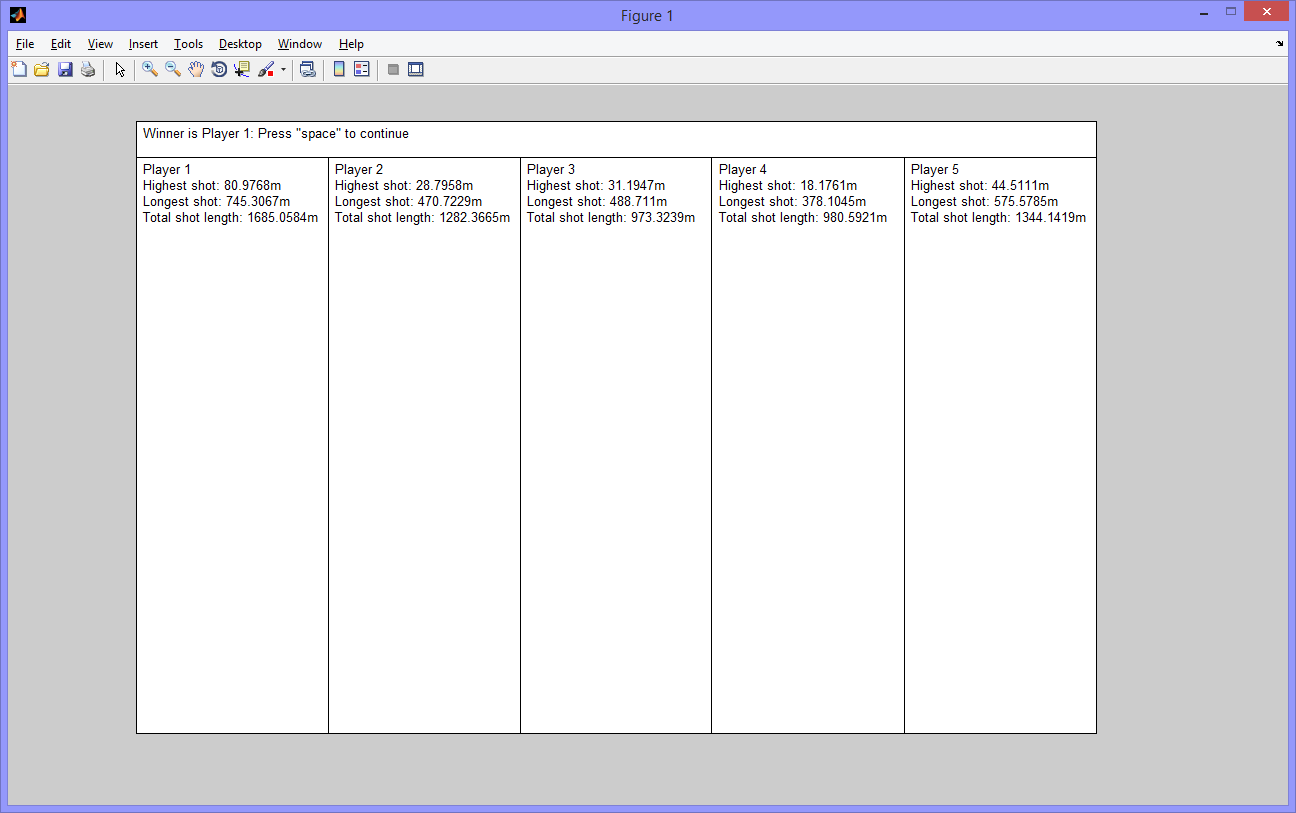
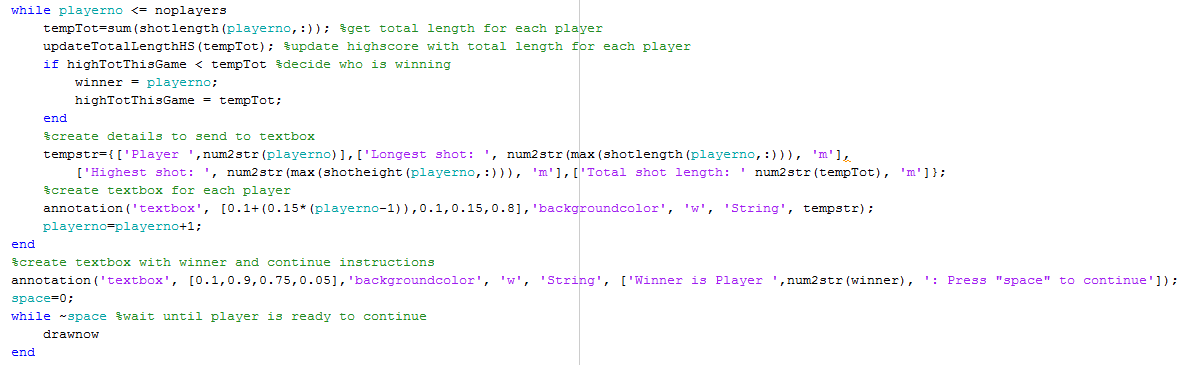
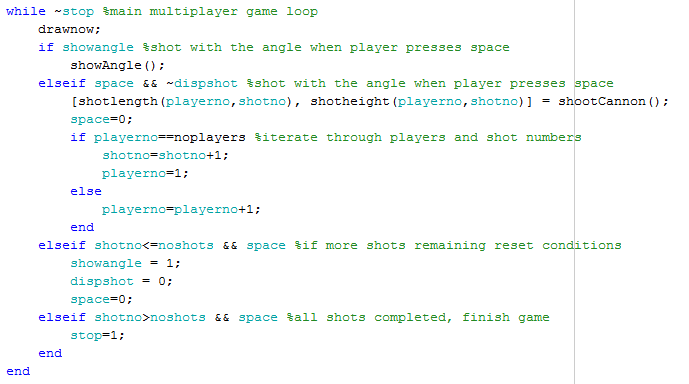
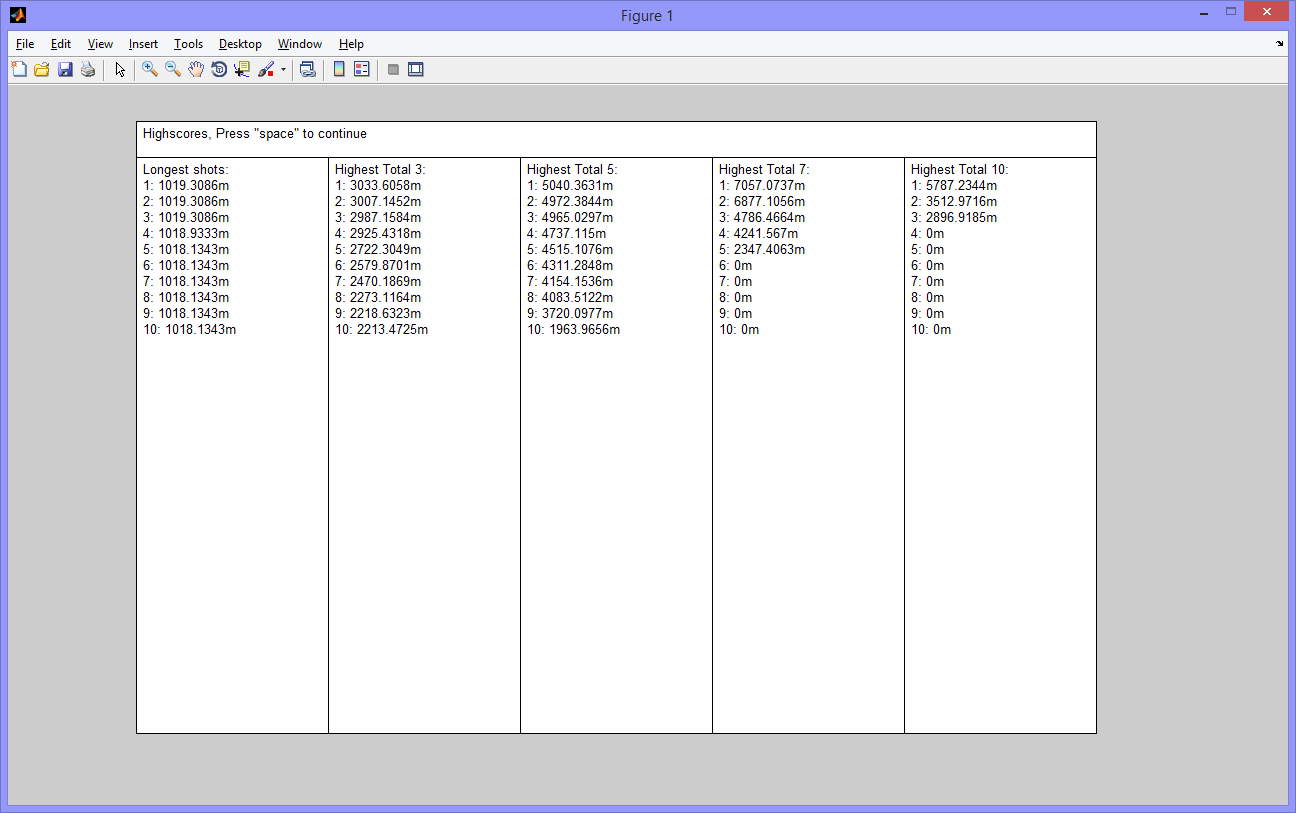
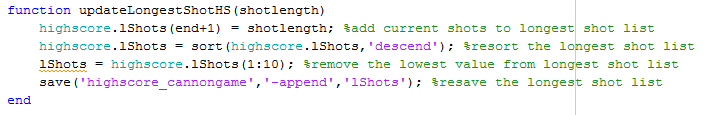
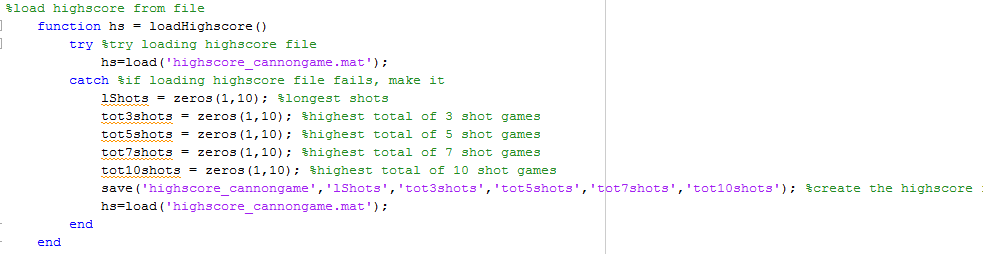
Alle Matlabfiler i skriftstørrelse 6.

## Obligatorisk del



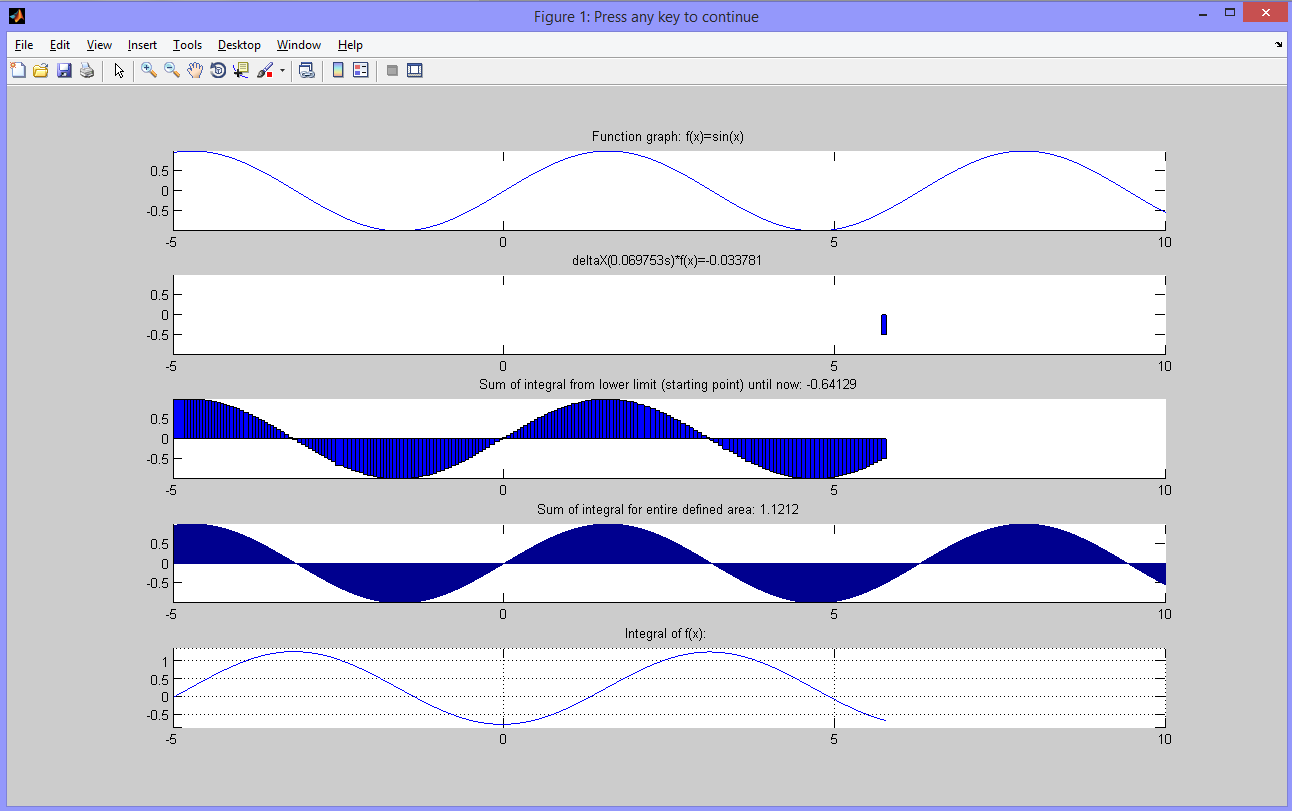
Figur 11 - Integrasjons verifisering alle grafer

## CANNON GAME

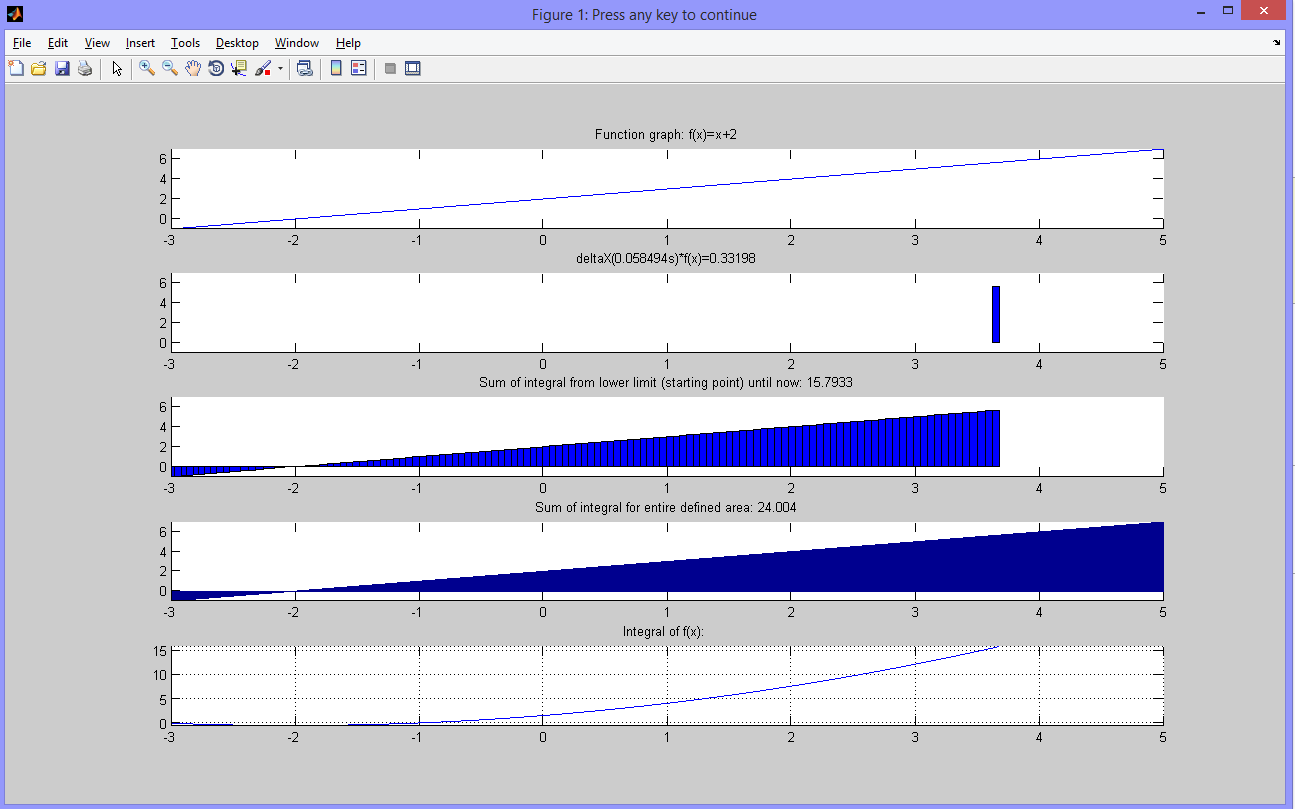


## mathShow

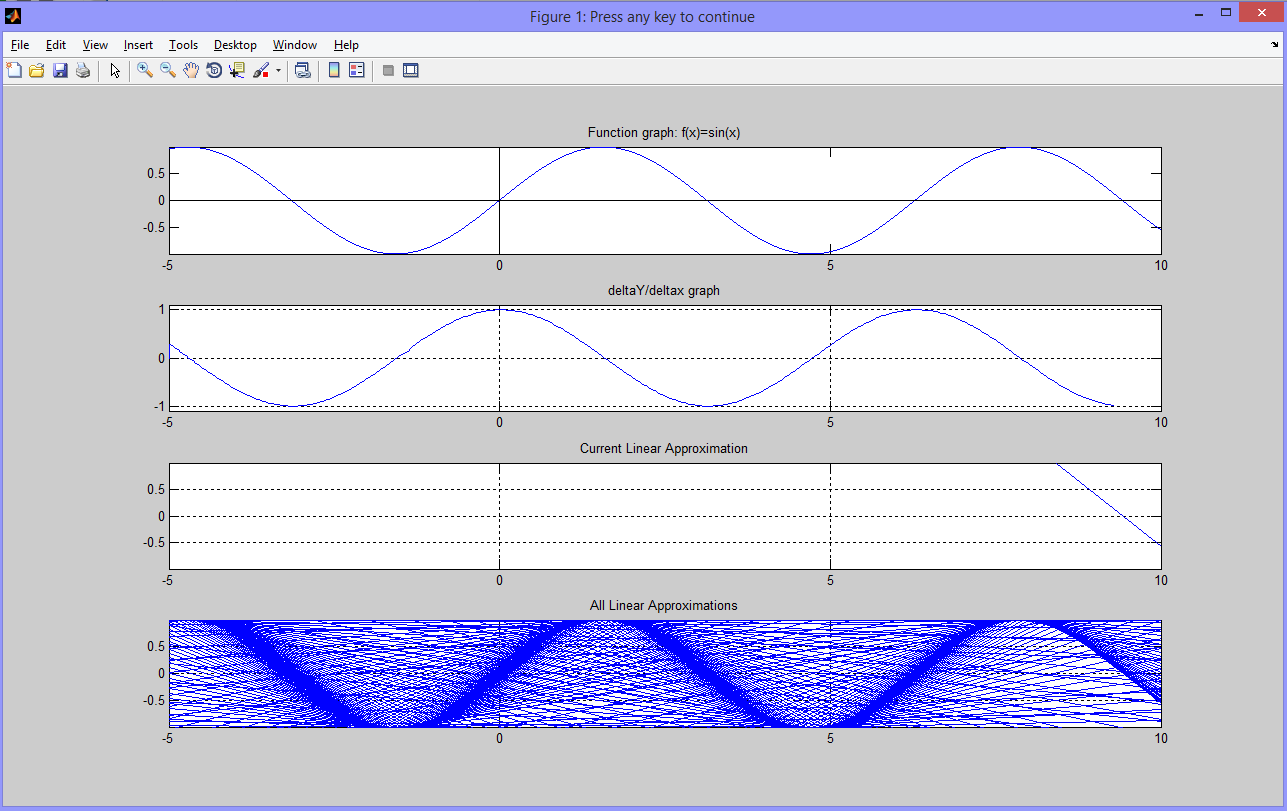
Nummerisk integrasjon av sin(x):



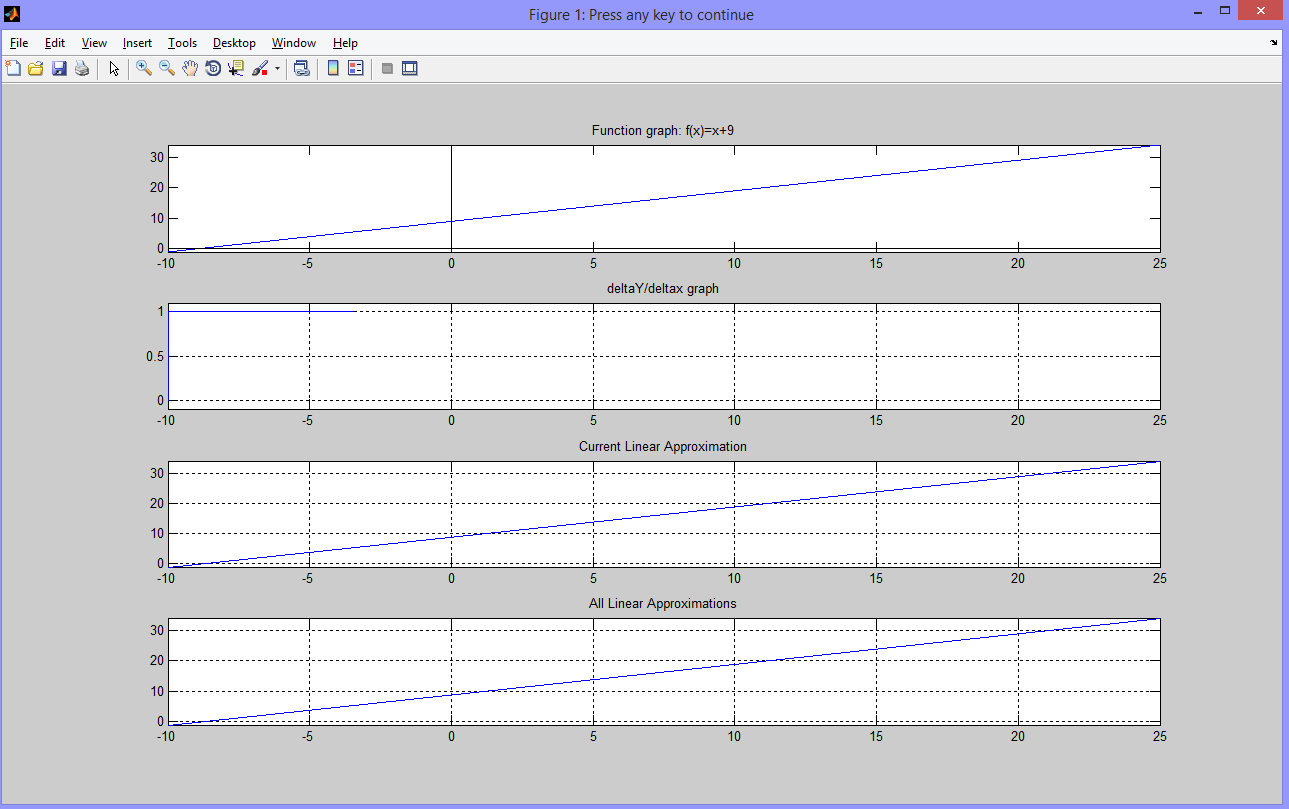
Nummerisk integrasjon av x+2:



Nummerisk derivasjon av sin(x):



Nummerisk derivasjon av x+9:



## NUMJOY

