NTNU
Det skapende universitet

Samarbeidende Legoroboter

Sigurd Hannaas

Master i teknisk kybernetikk
Oppgaven levert: Juli 2011
Hovedveileder: Tor Engebret Onshus, ITK

Norges teknisk-naturvitenskapelige universitet

Masteroppgave NTNU

Kandidatens navn: Sigurd Hannaas

Fag: Teknisk kybernetikk

Oppgavens tittel: Samarbeidende legoroboter

Oppgavens innhold:

Med utgangspunkt i tidligere prosjekt- og masteroppgaver skal et system for automatisk kartlegging av et ukjent område videreutvikles. Utgangspunktet er to roboter, hvor den ene benytter infrarød avstandsmåling som måleprinsipp, mens den andre benytter bildebehandling av bildet fra et påmontert webkamera, sammen med en ultrasonisk sensor. Det er tidligere laget et grensesnitt for IR-roboten i Matlab. Følgende punkter inngår i oppgaven:

- Hastigheten på de to motorene til IR-roboten skal kontrolleres og reguleres ved hjelp av målinger fra optokoblere på drivhjulene (eller eventuelt nye slepehjul).
- IR-roboten skal bygges om for å få bedre vektfordeling.
- En simuleringsmodell av IR-robotens batteri skal implementeres.
- Kamerarobotens kartleggingssystem skal inkluderes i IR-robotens grensesnitt, slik at grensesnittets kartgenerering fungerer med måledata fra kameraroboten.
- Usikkerhet i målingene bør benyttes som parameter i kartgenereringen.
- Metodikk for å kombinere måledata fra de to robotene skal implementeres.
- Grensesnittet bør utvides til å håndtere to eller flere roboter samtidig.

Oppgaven gitt: 1. februar 2011

Besvarelse leveres innen: 4. juli 2011

Utført ved Institutt for teknisk kybernetikk, Norges teknisk-naturvitenskapelige universitet (NTNU)

Veileder: Tor Onshus

Forord

Denne mastergradsoppgaven er en videreføring av en rekke tidligere oppgaver som dreier seg om legoroboter. Rapporten dokumenterer forbedringsarbeidet som er gjort på en robot utviklet ved Institutt for teknisk kybernetikk gjennom flerfoldige tidligere prosjekt- og mastergradsoppgaver. I tillegg dokumenteres utvidelsen av det tilhørende styringssystemet i Matlab som gjør det i stand til å håndtere en annen robot.

Arbeidet som er lagt ned har vært både spennende, utfordrende og lærerikt. Å gjennomføre et slikt prosjekt har til tider vært vanskelig, men prosessen rundt har vært svært tankevekkende, og jeg har fått mange nyttige erfaringer underveis.

Uten følgende personer er det lite trolig prosjektet ville kommet i havn. Jeg er dem stor takk skyldig:

- Professor Tor Onshus som kunne tilby denne masteroppgaven, og som har vært en svært hjelpsom og tilpasningsdyktig veileder underveis.
- Marianne Ingeborg Karlsen fra Studentservice, for uvurderlig prosessveiledning og motivasjonstrening.
- Resten av gjengen på kontor D252, for å gjøre hverdagen til en fest!

Sammendrag

Denne oppgavebesvarelsen beskriver forbedringsprosessen av en robot, samt prosessen rundt å utvide et eksisterende system for generering av kart over ukjente områder. Utvidelsen består i at systemet nå kan håndtere to ulike roboter, der det i utgangspunktet kun håndterte den ene.

Den roboten som har blitt forbedret består i all hovedsak av to hjul med hver sin motor, slik at den kan bevege seg i bakkeplanet, og et sensortårn med fire infrarøde sensorer til avstandsmåling. Sensortårnet er plassert på en servomotor, slik at det kan roteres, og sensorene kan måle avstanden til omgivelsene 360° rundt roboten. For å holde orden på sin egen posisjon, er roboten utstyrt med to optiske sensorer som registrerer hjulenes omdreining, og dermed hvor langt, og i hvilken retning roboten har beveget seg. Alt dette styres av en mikrokontroller, som også holder styr på kommunikasjonen mot et brukergrensesnitt over bluetooth. Dette grensesnittet er utviklet i Matlab, og genererer et kart over omgivelsene basert på målinger fra roboten. I tillegg er det dette Matlab-systemet som forteller roboten hvor den skal bevege seg.

IR-roboten led i utgangspunktet av hjulspinn, grunnet feil vektbalanse og dårlige hjul. Ved å flytte massemiddelpunktet og bytte hjul har dette problemet blitt mer eller mindre eliminert. Sammen med en ny reguleringsprosess som sammenligner de to drivhjulenes bevegelser, gjør dette at robotens bevegelsesnøyaktighet har blitt kraftig forbedret.

For å bedre robotens autonome egenskaper, har den fått implementert batterimodellering. Batterimodellen simulerer robotens strømbruk, og benyttes i forbindelse med allerede eksisterende ladeautomatikk, slik at systemet selv vet når roboten trenger å lades. Denne utvidelsen av systemet har fjernet det siste ikke-autonome momentet i kartleggingsprosessen, slik at det nå i prinsippet bare er å trykke på start-knappen i brukergrensesnittet, så holder roboten på til kartleggingen er ferdig.

Totalt sett har IR-roboten gjennomgått mange små forbedringer, som til sammen utgjør en vesentlig forbedring, og oppgavetekstens mål om forbedringer av IR-roboten kan sies å være nådd.

Den andre roboten er konstruert på tilsvarende vis som IR-roboten. Den har to motorer med hvert sitt hjul, i tillegg til et sensortårn som kan roteres. Tårnet består i dette tilfellet av et trådløst webkamera, samt en avstandssensor basert på ultrasonisk lyd. Roboten observerer sine omgivelser ved ved å undersøke bilder fra kameraet, og sjekker avstanden til det den finner med ultralydsensoren.

Oppgavebesvarelsen viser videre hvordan kameraroboten har blitt inkludert i IR-robotens grensesnitt, slik at dette grensesnitte kan kontrollere begge robotene. Systemet har blitt tilpasset slik at måledata fra kamera-roboten også fungerer i kartgenereringsprosessen.

I utgangspunktet var meningen å la grensesnittet operere begge robotene samtidig, men dette er ikke gjort. Dermed vil et naturlig neste trinn være å fortsette utvidelsen av systemet til å fungere med to eller flere roboter parallelt. Arbeidet som er gjort for å utvide systemet til to roboter skal ha lagt et godt grunnlag for å kunne ta utvidelsen et nytt steg videre.

Innholdsfortegnelse

Figurer

Tabeller

1. Innledning

Flere små og enkle autonome roboter som kan i mange tilfeller være bedre egnet enn mer kompliserte roboter som opererer alene. For eksempel gjelder dette kartlegging av et ukjent område. Her vil en gruppe enkle roboter i de fleste tilfeller kunne utføre jobben på mye kortere tid enn hva en enkelt kompleks robot ville klart.

En liten og billig robot vil – både av økonomiske årsaker og plasshensyn – ikke ha like mange eller nøyaktige sensorer som en stor og dyr robot. En metode for å kompensere for denne ulempen, er å bruke flere små roboter sammen, og korrelere deres individuelt genererte representasjoner av omgivelsene. En flokk av små autonome roboter kan ha mange bruksområder, for eksempel søk i sammenraste bygninger, hvor det er begrenset fremkommelighet og liten størrelse er en fordel, eller rask kartlegging og overvåkning av andre områder hvor det er for farlig å oppholde seg for mennesker.

Små autonome roboter er alt på markedet, for eksempel i form av gressklippere og støvsugere, men de er særlig utbredt i militær sammenheng, der små (og store) autonome fly stadig tar over nye oppgaver fra sine pilotkontrollerte kolleger. Det er nok ikke lenge til vi vil se bruk av flokk-roboter både i sivil og militær sammenheng, så arbeidet med disse problemstillingene er definitivt nyttig.

I denne oppgaven skal en legorobot med IR-detektorer og en med påmontert webkamera kartlegge et ukjent område. Et eksisterende brukergrensesnitt utviklet for IR-roboten brukes som utgangspunkt. Formålet er kort fortalt å få begge robotene til å fungere i det samme grensesnittet.

Løsningen av oppgaven er her delt i to separate prosesser. Den første dreier seg om å forbedre IR-roboten, da denne har hatt en del problemer med fremkommeligheten. Ved hjelp av en rekke mindre forbedringer har nøyaktigheten i kartet som produseres på grunnlag av data fra denne roboten blitt betraktelig bedre.

Den andre delen dreier seg om å sy sammen et brukergrensesnitt som fungerer med begge robotene. Det eksisterende bruker

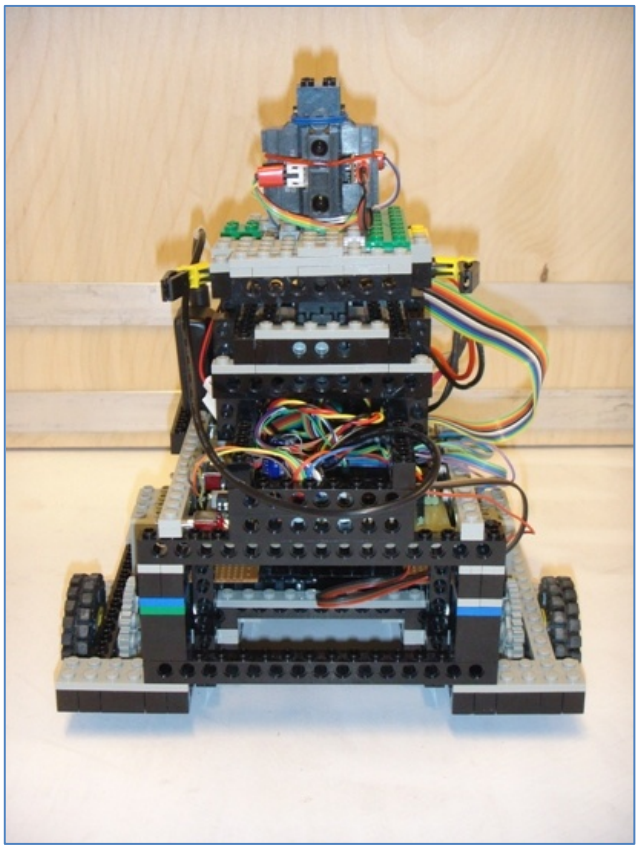
Ettersom denne oppgaven baserer seg på mange tidligere prosjekt- og masteroppgaver om de samme robotene, er det innledningsvis tatt med et kapittel om utgangspunktet. Der er de to robotene beskrevet, og nødvendig informasjon for å koble dem opp og sette i gang systemene er gjennomgått. Videre kommer kapitlene om forbedring av IR-roboten, og sammensying av grensesnittet, før en utfyllende liste med mulige fremtidige forbedringer av systemet er med til slutt.

2. Utgangspunkt

2.1. Oppsett

I utgangspunktet er de to robotene som omtales utviklet individuelt, og er derforsatt opp på forskjellig måte. Under følger en beskrivelse av utgangsoppsettet for robotene og deres kontrollsystem.

2.1.1. Oppsett for IR-roboten



Figur 1: IR-robot. Bilde fra (Tusvik, 2009)

Det samlede systemet for IR-roboten er godt dokumentert av Tusvik (Tusvik, 2009). Roboten og den tilhørende programkoden er utviklet i flere trinn i forskjellige prosjekt- og diplomoppgaver, og hun har derfor laget en helhetsbeskrivelse av systemet, slik at videre arbeid enklere kan gjennomføres.

Kommunikasjonen med roboten foregår via RS-232 trådløst over Bluetooth. Roboten er utstyrt med en enhet som sørger for dette, kalt Free2move, som må initieres før Matlab-grensesnittet kan fungere. Dette ble forsøkt gjennomført i henhold til instruksene i (Tusvik, 2009), men enkelte endringer måtte foretas, da det ikke fungerte som beskrevet. Grunnen er etter alt å dømme at Tusvik benytter operativsystemet Windows XP, mens datamaskinen i dag er utstyrt med Windows 7. Her følger en punktvis beskrivelse av en tilkoblingsprosedyre som fungerer i Windows 7, basert på prosedyren i (Tusvik, 2009):

2.1.1.1. Prosedyre for kommunikasjon mellom IR-robot og PC

1. Sett Bluetooth-donglen i en av PC'ens USB-porter, og vent til driverne er installert
2. Free2move-enheten kobles til PC'ens RS-232 seriellport (her COM1) og forsynes med strøm – gjerne fra roboten (den må da være ladet og påslått)
3. Åpne Free2move configuration software, kan lasts ned fra (Free2Move, 2011)
4. Velg riktig COM-port (her COM1)
5. Velg fanen "Endpoint"
6. Velg "Endpoint (Bluetooth slave) under "Device mode" og "Required paired device" under "Security mode". Trykk så "Pair device…"
7. Velg "Remote device initiates pairing" og trykk "Next"
8. Skriv inn ønsket passord og trykk "Pair". Programmet blir nå stående å vente, tilsynelatende uten å svare.
9. Trykk på Bluetooth-logoen nede til høyre på Windows-oppgavelinjen, og velg "Add a device". Det søkes nå etter enheter, og etter en stund skal "Free2move" dukke opp. Velg denne og trykk "Next". Skriv så inn det samme passordet som i Free2move configuration software og trykk "Next". Nå konfigureres tilkoblingen automatisk. Vent til alle drivere er installert.
10. Kontrollér hvilken COM-port som er utpekt ved å velge "Devices and printers" fra Windows' start-meny. Dobbeltklikk på Free2move-ikonet og velg fanen "Services". Hvis alt er konfigurert riktig, skal det være en linje som heter "Serial port (SPP) 'SerialPort'". Det er den tilhørende COM-porten som skal benyttes under tilkoblingsprosedyren i brukergrensesnittet [avsnitt 2.1.1.2] (her COM9).
11. Fjern strømtilførselen fra Free2move-enheten og koble den fra PC'en og inn i robotens RS-232-port. Koble til strømmen igjen. Roboten er nå klar for tilkobling fra brukergrensesnittet i Matlab.

2.1.1.2. Prosedyre for kommunikasjon mellom brukergrensesnitt og IR-robot

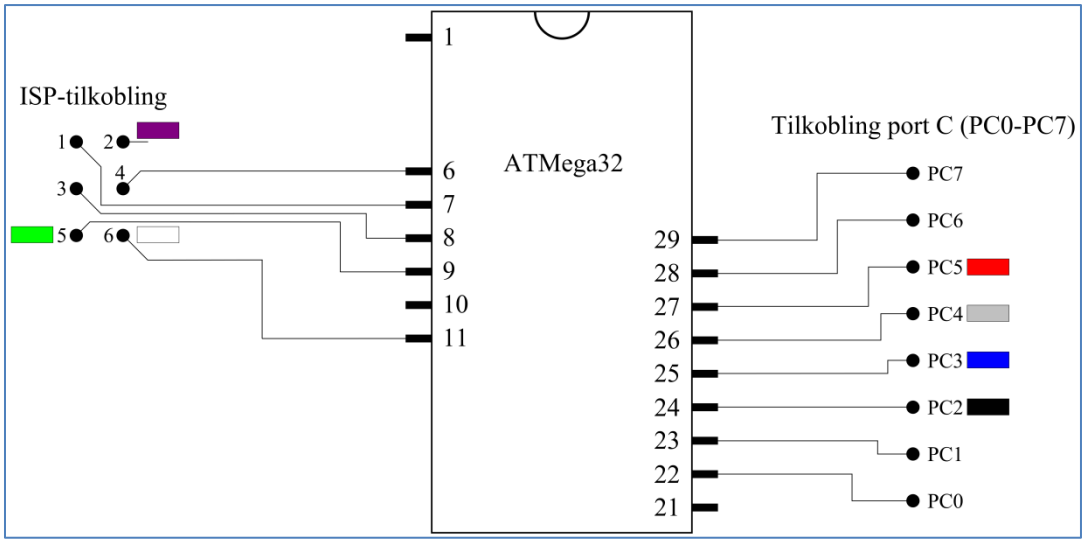
Veiledningen for bruk av Matlab-grensesnittet er tilstrekkelig beskrevet av Tusvik (Tusvik, 2009), og hennes veiledning er derfor gjengitt i sin helhet i vedlegg E.

2.1.1.3. Prosedyre for kommunikasjon mellom PC og IR-robotens mikrokontroller

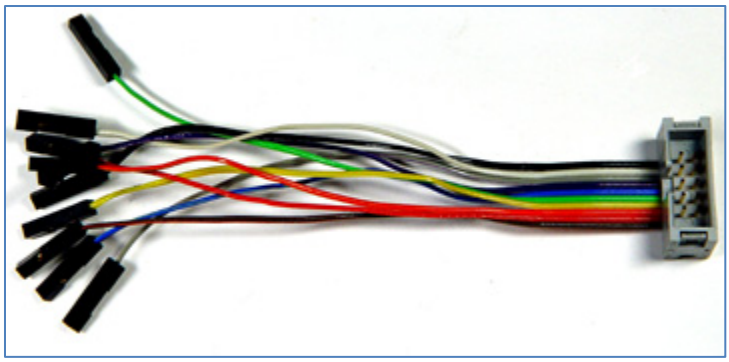
Roboten er kontrollert av en mikrokontroller av typen AVR ATMega32. For å kunne endre robotens oppførsel og kommunikasjon med grensesnittet, må mikrokontrolleren

omprogrammeres. Dette kan enklest gjøres ved å bruke AVR Studio som er fritt tilgjengelig og kan lastes ned fra Atmel (Atmel, 2011b). I skrivende stund er siste tilgjengelige versjon 5.0 beta, og det er denne prosedyren er basert på. Programvarepakken kan kommunisere med en rekke verktøy til bruk sammen med mikrokontrollere, for eksempel AVR JTAGICE mkii, som ble brukt i (Tusvik, 2009), eller AVR Dragon som denne prosedyren baserer seg på. Begge kommuniserer med mikrokontrolleren via JTAG, og med AVR Studio via USB. Prosedyren er basert på en tilsvarende prosedyre i (Tusvik, 2009), men den er noe endret som følge av ny programvare og annet tilkoblingsverktøy, samt utvidet for å gjøre den enklere å følge.

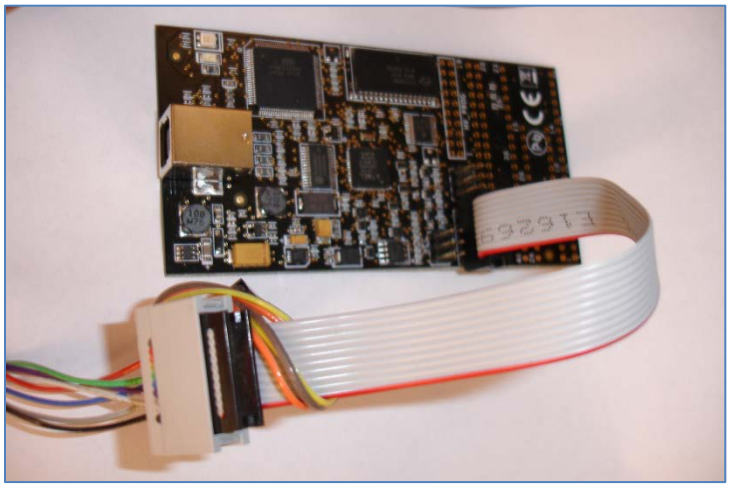
- Åpne AVR Studio og riktig prosjektfil/"Solution" (her kalt *Hovedoppgave_Hannaas.avrsln*)
- Utfør eventuelle endringer og tilpasninger i C-koden
- Bygg koden ved å trykke F7 eller knappen *Build Solution* ()
- Dersom koden bygges uten feilmeldinger, er den klar til å skrives til mikrokontrolleren. Før dette kan skje, må roboten være tilkoblet AVR Dragon via JTAG etter følgende oppsett:



Figur 2: Topografisk riktig utsnitt av robotens hovedkort. Basert på Kalleviks kretskortlayout (Kallevik, 2007). For å forenkle tilkoblingen, er tilkoblingspunktene som skal benyttes fargekodet tilsvarende JTAG-kabelen (se Figur 3).



Figur 3: Fargekodet tilkolingskabel



Figur 4: Fargekodet kabel koblet til AVR Dragons JTAG-header via en vanlig flatkabel

Tabell 1: Oversikt over JTAG-tilkobling mellom robot of AVR Dragon

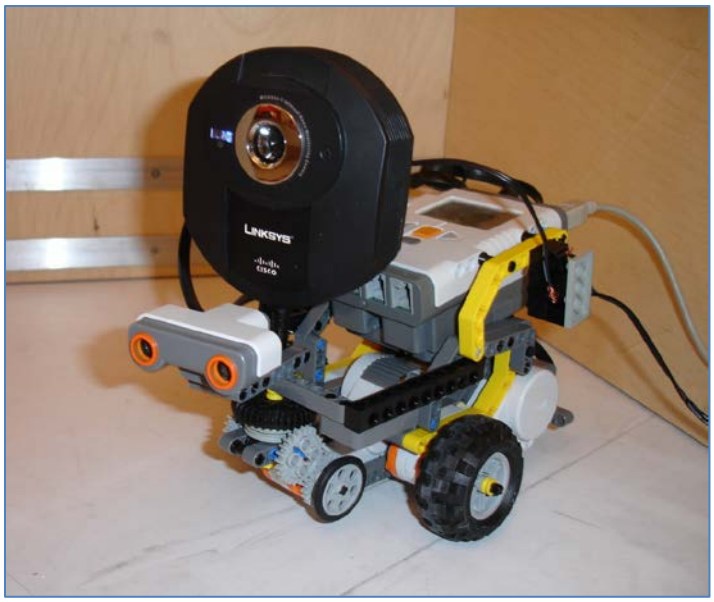
Tilkobling AVR Dragon JTAG (farget kabel)	ISP-tilkobling pin nr.	Tilkobling port C	Signal	ATMega32 pin nr.
Pin 1 (svart)	-	PC2 (TCK)	**TCK** (test clock-signal fra AVR Dragon til ATMega32)	24
Pin 2 (hvit)	6 (GND)	-	**GND** (jord)	11
Pin 3 (grå)	-	PC4 (TDO)	**TDO** (test data output-signal fra ATMega32 til AVR Dragon)	26
Pin 4 (lilla)	2 (VTG)	-	**VTref** (target reference voltage, kontrollerer level-converter på AVR Dragon)	-
Pin 5 (blå)	-	PC3 (TMS)	**TMS** (test mode select-signal fra AVR Dragon til ATMega32)	25
Pin 6 (grønn)	5 (RESET)	-	**nSRST** (open collector output fra AVR Dragon til invertert RESET på ATMega32)	9
Pin 9 (rød)	-	PC5 (TDI)	**TDI** (test data input-signal fra AVR Dragon til ATMega32)	27

- Koble AVR Dragon til PC'en via USB, og slå på roboten. Vent til Windows er ferdig med eventuelle driverinstallasjonsprosesser.
- Nå skal AVR Dragon ha to lysende dioder; en grønn og en rød. Dette betyr at den er riktig tilkoblet, men ingen kommunikasjonskanal er åpen.
- Trykk på knappen *AVR Programming* ( ). I vinduet som kommer opp velges *AVR Dragon* under *Tool*, *ATMega32* under *Device* og *JTAG* under *Interface*. Trykk *Apply*.
- Dersom tilkoblingen er vellykket skal AVR Dragon kun ha en lysende grønn diode. Dette betyr at kommunikasjonskanalen er åpen mellom PC og AVR Dragon. For å verifisere dette, kan Dragonens *Device ID* avleses ved å trykke på *Read*-knappen i *AVR Programming*-vinduet.
- I samme vindu kan det velges *Memories* fra listen til venstre. Her kan eksisterende robotprogram avleses og lagres som backup ved å trykke *Read*. For å programmere roboten, må riktig *.hex-fil velges (her *Hovedoppgave_Hannaas.hex*) i *flash*-feltet. Trykk så *Program*.

Figur 5: Opprinnelig oppsatt kamera-robot. Bilde fra (Tøraasen, 2009)

2.1.2. Oppsett for kameraroboten

Denne roboten baserer seg på LEGO Mindstorms NXT, som er et system basert på en sentral kontrollmodul (heretter kalt NXT-enhet) og diverse sensorer og aktuatorer som kobles til denne, og som enkelt lar seg konfigurere. Roboten er prinsipielt lik Tøraasens (Tøraasen, 2009) robot, men den mekaniske oppbyggingen er noe annerledes, da Tøraasens robot ble demontert i forbindelse med et annet prosjekt, og følgelig måtte bygges opp igjen. Roboten som benyttes her er altså ikke fullstendig lik den tidligere roboten, men består av tilsvarende komponenter, og oppfører seg likt.



Figur 6: Nytt oppsett av kamera-roboten

Systemet består av NXT-enheten, tilkoblet tre motorer, og én ultrasonisk avstandssensor. To av motorene driver hvert sitt hjul, mens den tredje styrer vinkelen til et sensortårn hvor avstandssensoren er montert, og hvor det i tillegg står påmontert et trådløst webkamera. På denne måten kan roboten vri kamera og sensor uten å måtte bruke drivhjulene. Videre støttes roboten opp av en støtte i bakkant, noe som fungerer godt på jevnt underlag, selv om det potensielt kunne medført problemer dersom roboten skulle beveget seg i mer ulendt terreng.

I tidligere prosjekter med denne roboten, har kommunikasjonen mellom PC og NXT-enhet foregått via bluetooth. Det har av uvisse årsaker ikke latt seg gjøre å oppnå kontakt med roboten over dette grensesnittet, trolig er det noe galt i NXT-enheten. Derfor har det blitt benyttet en USB-kabel, noe som i utgangspunktet vil begrense robotens bevegelsesfrihet, men som i praksis har liten betydning i denne omgang, da roboten beveger seg i et begrenset område. I tillegg trenger det påmonterte kameraet strøm, som det per i dag får via ledning, så denne bevegelsesbegrensningen er tilstede uansett. En innlysende forbedring blir altså å bygge om roboten med batteri til kameraet og tilhørende ladekrets, samt reparere bluetooth-grensesnittet, slik at roboten blir trådløs. Dette er ført opp som mulige forbedringer i kapittel 6 Videre arbeid.

For å styre roboten direkte fra Matlab har RWTH Aachen University utviklet en *toolbox* for Matlab (RWTH Aachen University, 2011), med en rekke ferdigimplementerte metoder som brukes direkte mot NXT-enheten. Tøraasen (Tøraasen, 2009) har benyttet dette grensesnittet i sitt arbeid med kameraroboten, som denne rapporten bygger videre på, og det vil følgelig også brukes her.

Følgende prosedyrer er basert på prosedyren utarbeidet av Tøraasen (Tøraasen, 2009), men endret til å benytte USB i stedet for bluetooth.

2.1.2.1. Tilkobling av NXT-enheten

- Last ned og installer siste USB-driver fra (LEGO, 2011)
- Koble NXT-enheten til PC'en via USB-kabel, og slå på roboten.

Det er flere mulige løsninger for å få konfigurert kameraet. Den prosedyren som følges her, har vist seg å være relativt enkel, og bør fungere uavhengig av hvilket (om noe) nettverk PC'en er koblet til.

2.1.2.2. Tilkobling og konfigurering av trådløst kamera

- Koble PC'en til en trådløs ruter (her ble det gjort med kabel, men trådløs tilkobling bør også fungere. Poenget her, er at PC'en og kameraet er på det samme lokale nettverket.
- Koble en nettverkskabel i ruteren, slik at den kan kobles til kameraet i konfigureringsfasen.
- Sett inn CD'en som fulgte med kameraet, og følg installasjons- og konfigurasjonsveiledningen.
- For å kontrollere at kameraet vil levere bilder til Matlab, gå til http://[kameraets ip-adresse]/img/snapshot.cgi. (her http://192.168.1.150/img/snapshot.cgi) i en nettleser. Det skal da vises et fotografi tatt av kameraet.

Nå skal roboten kunne benyttes i Matlab.

2.1.3. Laboppsett

Systemet som er benyttet består av følgende komponenter i tillegg til de to robotene:

- PC med nødvendig programvare:
  - Windows 7 Enterprise (kontinuerlig oppdatert)
  - Matlab R2010a
  - AVR Studio 5
  - USB-driver fra LEGO (LEGO, 2011)
- AVR Dragon for programmering av og feilsøking på IR-roboten (se Figur 4)
- Testbane for robotene, der forskjellige omgivelser kan settes opp og kartlegges
- Ladestasjon som kan plasseres hvor som helst på testbanen

Det er ikke utarbeidet noen prosedyre for laboppsettet totalt sett, da dette er nokså intuitivt, eller beskrevet i andre prosedyrer, der dette vil være naturlig.

2.1.4. Forbedringspotensiale

Det følger her en oversikt over områder med åpenbart forbedringspotensiale i det opprinnelige oppsettet, i tillegg til forbedringer som har dukket opp under utarbeidelsen av oppgaveteksten, og som følgelig vil bli behandlet i kommende kapitler.

Tusvik (Tusvik, 2009) nevner flere mulige forbedringsområder for IR-roboten, hvorav noen går utenfor denne oppgavens formål, og derfor er gjentatt i kap. 6 ”Videre arbeid”, mens de følgende vil bli behandlet her (se for øvrig kapittel 3.1 for detaljer):

- Hastighetsregulering på drivhjulene ved hjelp av tilbakekobling fra optokoblerne, for å kompensere for eventuelle ulikheter mellom de to motorene som resulterer i dårligere posisjonering. Enkel feildeteksjon på optokoblerne er også ønskelig, og bør implementeres samtidig.

- Ombygging/omstrukturering av roboten for å forbedre vektbalansen og sikre godt veigrep, og på den måten begrense problemer som dukker opp som konsekvens av hjulspinn. I all hovedsak dreier det seg om at robotens posisjonsberegning raskt blir svært unøyaktig, ettersom en i utgangspunktet liten feil, for eksempel i robotens vinkel til omgivelsene, raskt blir stor. Dette skjer fordi avviket i robotens posisjon da vil øke proporsjonalt med distansen den har beveget seg etter at feilen inntraff.

(Dette er ikke slik punktene fremstår hos Tusvik, men hun nevner det som står i disse punktene).

Ettersom det alt er utarbeidet rutiner for retur til ladestasjonen (se (Kristiansen, 2009) for detaljer) er det ønskelig å kunne overvåke IR-robotens batterinivå, for å kunne lade automatisk. Dette kan gjennomføres ved å lage lage en simuleringsmodell av batterikapasitet og -bruk.

Kamera-roboten vil av omfangshensyn ikke bli forbedret i denne oppgaven, men mulige forbedringsområder er listet opp i kap. 6 ”Videre arbeid”.

Matlab-grensesnittet vil gjennomgå en del forbedringer og utvidelser, jf. oppgaveteksten. Først og fremst dreier det seg om å la det håndtere kamera-roboten i tillegg til IR-roboten. Følgende punkter dukker opp i den sammenheng:

- Måledata fra de to eksisterende robotene må være tilgjengelig samtidig, for at grensesnittet skal kunne generere det globale kartet basert på begge robotenes målinger.

- En algoritme for kombinasjon av de forskjellige måledataene til ett globalt kart må utarbeides. Samtidig bør det implementeres en usikkerhetsparameter i kartmålingene, for å kunne effektivisere kartleggingsprosessen ved å la usikre områder bli gransket på nytt av en robot, før de områdene som er kartlagt med større sikkerhet blir gransket.

- Robotene må vite om hverandre for å unngå at de detekterer hverandre som deler av omgivelsene.
- Dersom en robot befinner seg i et område som tidligere er kartlagt med stor sikkerhet (for eksempel ladestasjonen), men får målinger som ligger langt utenfor forventet intervall, bør systemet oppdatere dette, og i stedet for å oppdatere kartet, korrigere robotens posisjon, slik at den stemmer overens med de omgivelsene den befinner seg i. Slike situasjoner kan oppstå når det forekommer feil i robotens odometri, for eksempel ved hjulspinn.

3. Forbedringer

3.1. IR-robot

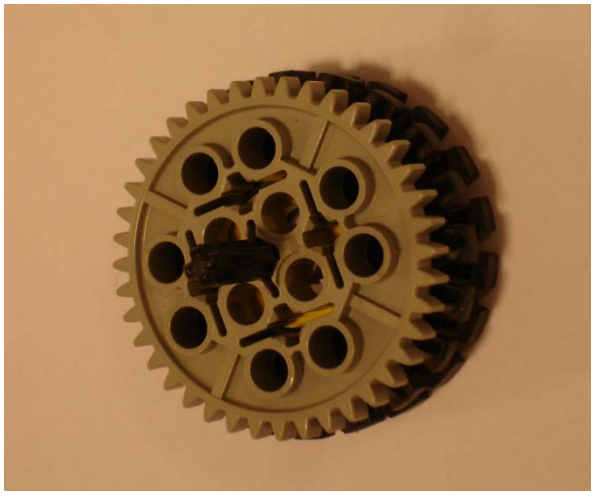
3.1.1. Inndeling av forbedringsprosesser

Som nevnt i kapittel 2.1.4 kan roboten forbedres på en rekke områder. Av praktiske hensyn er utbedringsarbeidet delt inn i fire separate prosesser; *Mekanisk ombygging, Omprogrammering av mikrokontrolleren, Oppdatering av grensesnitt* og *Batteriberegninger*. Detaljene i utbedringene er gjennomgått i de tilsvarende delkapitlene.

3.1.2. Mekanisk ombygging

I utgangspunktet var det fire hovedproblemer med robotens mekaniske konstruksjon:

1. For liten diameter på drivhjul i forhold til tannhjulene på samme aksel førte til hjulspinn ved at tannhjulet kom i kontakt med underlaget. Tannhjulenes lavere friksjon mot underlaget, sammenlignet med drivhjulenes gummidekk medførte at roboten begynte å spinne ved en rekke tilfeller.



Figur 7: Opprinnelig drivhjul på aksel med tannhjul fra giret

2. Et annet bidrag til hjulspinn var selve drivhjulene. Disse hadde dekk av gammel og hard plast, som medførte lav friksjonskoeffisient så snart de hadde fått et lite støvlag. Dekk som ikke tåler å bli litt skitne regnes ofte som sub-optimale.

Figur 8: Opprinnelig drivhjul

3. Odometriberegningene ble tatt direkte på motorakslingen før nedgiringen til drivhjulene, i stedet for på et eget system (for eksempel slepehjul). Dette betyr at målingene representerte omdreiningen på hjulene godt, men ettersom roboten led så kraftig av hjulspinn, ble ikke målt distanse riktig i forhold til omgivelsene.

4. Roboten var svært baktung, noe som representerte et problem på grunn av flere faktorer. Ettersom roboten er konstruert med trekk på to forhjul, og kun et støttehjul bak, vil det være mest hensiktsmessig å ha robotens tyngdepunkt sentrert over forhjulene. Bakhjulet rullet dårlig under den store vekten, og dette, sammen med lav vekt over drivhjulene utgjorde nok et bidrag til hjulspinn.

Punkt 1 og 2 i listen over ble løst ved å bytte drivhjulene til en bredere modell med marginalt større diameter, og dekk laget av luftfylt gummi. Dette bedret problemene med hjulspinn betraktelig.

Figur 9: Nytt og gammelt drivhjul

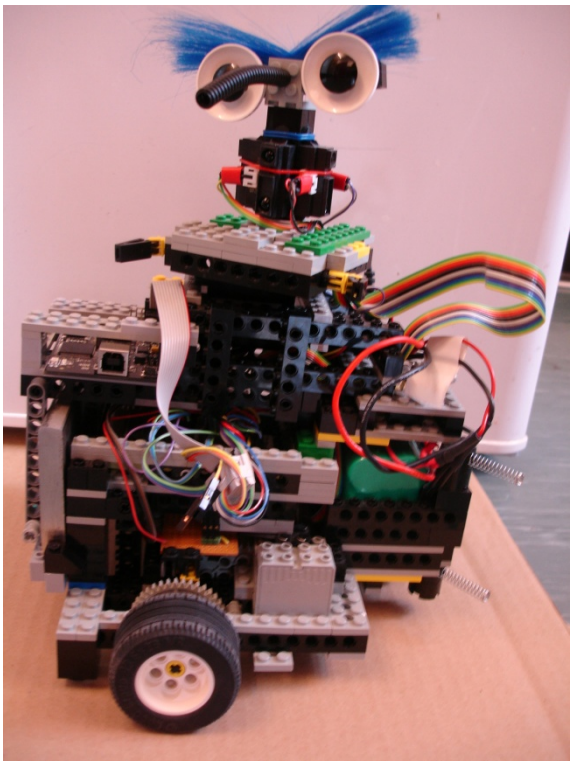
Figur 10: Større hjuldiameter sikrer tilstrekkelig klaring mellom tannhjul og bakken

Det ble gjort forsøk på å lage et par separate slepehjul for å løse problem 3. For å sikre tilstrekkelig oppløsning måtte det lages et gir, slik at tannhjulet som optokoblerne benyttet

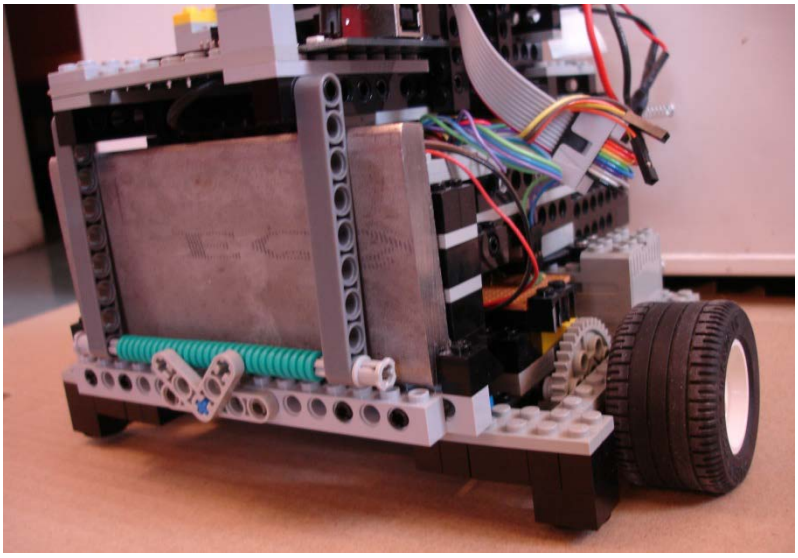
Figur 11: Utkast til slepehjul med gir, der optokobleren ville talt "tikk" på det store tannhjulet øverst

hadde høy nok omdreiningshastighet. Ettersom roboten i all hovedsak er laget av lego, ble giret bygget av standard lego-tannhjul av plast. Det viste seg å gjøre konstruksjonen ganske stor, som ville gitt problemer med plasseringen i selve roboten. I tillegg viste det seg at tannhjulene ikke var stive eller presise nok, noe som medførte variasjoner i omdreiningshastigheten på måletannhjulet, selv om translasjonshastigheten mot underlaget ble holdt konstant. Konklusjonen ble at dette ikke var en god løsning i dette tilfellet, men ved å bruke mer nøyaktige tannhjul, samt omstrukturere roboten kan dette vise seg å bli en god løsning i fremtiden. En slik løsning ville blitt for omfangsrik for denne oppgaven, og er derfor flyttet til kapittel 6 *Videre arbeid.* Det ble konkludert med at i denne omgang ville fokus ligge på å forbedre odometrinøyaktigheten ved å hindre hjulspinn.

For å løse problem 4, så det ut til at en massiv ombygging av roboten var nødvendig. For å spare tid, ble det i stedet valgt en løsning der en motvekt ble montert i front av roboten, slik at tyngdepunktet plasserte seg bedre over hjulakslingen, og bakhjulet ble avlastet. Denne ad-hoc-løsningen fungerte greit, men er ikke optimal, ettersom den tilfører roboten unødvendig vekt. En større ombygging er derfor foreslått i kapittel 6 *Videre arbeid.*En heldig konsekvens av denne vektøkningen er at roboten nå er for tung til at motorene klarer å spinne hjulene, roboten stopper i stedet. Dette gjør at odometriberegningene i det minste ikke blir skadelidende, dersom roboten for eksempel skulle kjøre seg fast.



Figur 12: Roboten sett fra siden. Motvekten skimtes til venstre i bildet.



Figur 13: Nærbilde av motvekten i front av roboten

Alle disse løsningene medførte endret bevegelseskarakteristikk, og følgelig måtte robotens mikrokontroller justeres tilsvarende. Dette er beskrevet i kapittel 3.1.3 *Omprogrammering av mikrokontrolleren.*

3.1.3. Omprogrammering av mikrokontrolleren

3.1.3.1. Parameterendringer som følge av mekaniske endringer

I prinsippet er det kun én endring som burde foretas i koden, utfra de mekaniske forbedringene som ble foretatt i kapittel 3.1.2, nemlig nustering av hjuldiameteren. Det kan vise seg at flere parametre må justeres, da roboten tidligere har vist seg å måtte kalibreres med forskjellige konstanter til forskjellige bevegelser, jf. sitat fra filen *encoder.h* (se vedlegg A):

```
//#define X_FACTOR 0.675          //Circumference(135mm)/Tics pr
                                  rotation(200) SHOULD WORK. How far the
                                  robot travles per tick
#define X_FACTOR 0.770            //Found by testing. Why must this be
                                  increased?
#define X_FACTOR_ROTATE 0.675;
#define X_FACTOR_DRIVE 0.775;
```

Som det fremkommer i kodeutdraget, oppfører roboten seg ulikt når den kjører og når den svinger (det er også en tredje konstant, som kun heter X_FACTOR, men denne er ikke lenger i bruk under bevegelsestilstander, og har dermed ingen effekt). Det er trolig at begge disse tallene må endres, også i forholdet til hverandre. Dette vil bli avgjort ved testing, og med utgangspunkt i at alle tallene kalt X_FACTOR skal være det samme, nemlig $\frac{\text{hjulomkrets [mm]}}{\text{"tikk" per hjulomdreining}}$. De nye hjulene på roboten har påskrevet 49,6 × 28, som tilsier en omkrets på ca. 156 mm, men dekkgummien er myk og blir komprimert noe, så den effektive omkretsen vil være noe mindre. Utvekslingen mellom tannhjulet som trigger optokobleren, og drivhjulet er 5:1, og nevnte tannhjul har 40 tenner, altså registrerer optokobleren 200 ”tikk” per hjulomdreining. Dette gir oss utgangspunktet for testingen, nemlig at X_FACTOR = 0,78 uavhengig av robotens tilstand.

Ved å la roboten gå gjennom en iterativ testprosedyre, kan X_FACTOR-konstantene kalibreres individuelt. I Tabell 2 og Tabell 3 er enkeltforsøkene listet opp.

Ved kalibrering av X_FACTOR_DRIVE aksepteres et slingringsmonn på ± 2 mm, da dette omtrent tilsvarer dødgangen i girene.

Tabell 2: Kalibrering av X_FACTOR_DRIVE

X_FACTOR _DRIVE	Kommandert distanse [mm]	Teoretisk antall "tikk" = $\frac{\text{distanse}}{\text{X\_FACTOR}}$	Målt distanse [mm]	Korreksjon: Ny X_FACTOR = $\frac{\text{målt distanse}}{\text{antall "tikk"}}$
0,780	500	641,0256	535	X_FACTOR = 0,835
0,835	500	598,8024	496	X_FACTOR = 0,828
0,828	500	603,8647	504	X_FACTOR = 0,835
0,835	500	598,8024	496	Konvergerer ikke. Lar ny X_FACTOR = $\frac{0,828+0,835}{2}$ = 0,8315
0,8315	500	601,3229	500	OK

Rotasjonskonstanten beregnes ved en tilsvarende prosedyre, men med rotasjon i stedet for translasjon. Antall "tikk" registrert på høyre og venstre hjul skal være lik hverandre, men med motsatt fortegn, så her benyttes fortegn etter vanlig norm, med positiv vinkel mot klokken, altså venstresving. Ettersom vinkelen må beregnes på bakgrunn av tilbakelagt buelengde per hjul, kommer akselbredden inn som faktor i ligningene. Denne forekommer for øvrig utelukkende i uttrykk sammen med X_FACTOR_ROTATE, der disse er proporsjonale størrelser, så i praksis er det ikke så farlig hva bredden er satt til når rotasjonskonstanten uansett kalibreres. I prinsippet kunne X_FACTOR_ROTATE blitt holdt konstant mens akselbredden ble justert, men det blir en smakssak. Ettersom det er enklere å få nøyaktige mål av akselbredden, ble X_FACTOR_ROTATE valgt som optimaliseringsvariabel.

Akselbredden, D, ble målt til ca. 210 mm.

Ved kalibrering av X_FACTOR_ROTATE aksepteres et slingringsmonn på ± 2° ≈ 0,0349 radianer per omdreining (altså ca. 0,009 radianer på en vinkel av $\frac{\pi}{2}$ som er brukt i forsøkene).

Tabell 3: Kalibrering av X_FACTOR_ROTATE

X_FACTOR _ROTATE	Kommandert rotasjon, θ_{ref} [rad]	teoretisk antall "tikk" = $\frac{\theta_{ref} \cdot D}{2 \cdot X\_FACTOR}$	Målt rotasjon, θ [rad]	Korreksjon: Ny X_FACTOR = $\frac{\theta \cdot D}{2 \cdot antall\ "tikk"}$
0,780	$\frac{\pi}{2} \approx$ 1,5708	211,4538	1,52797	X_FACTOR = 0,7587
0,7587	$\frac{\pi}{2} \approx$ 1,5708	217,3903	1,57794	OK, tester rotasjon andre veien.
0,7587	$-\frac{\pi}{2} \approx$ -1,5708	217,3903	-1,57556	OK

Tabell 2 viser at X_FACTOR_DRIVE måtte justeres opp i forhold til beregnet verdi, stikk i strid med antakelsen om at hjuldiameteren i realiteten vil være mindre enn målt/påtrykket grunnet komprimering av dekkene. Dette kan være forårsaket av forskjellige faktorer:

1. Hjuldiameteren kan være målt/avlest for liten
2. Antall ”tikk” blir ikke talt riktig

Alternativ 1 kan enkelt avvises, ved å beregne hjulets omkrets utfra kalibrert X_FACTOR_DRIVE. Denne blir da 0,8315 * 200 = 166,3 mm, som gir en diameter på 52,9 mm. Dette er 3,3 mm *mer* enn påtrykket verdi (verifisert med kontrollmålinger av både diameter og omkrets), og kan altså ikke være feilkilden.

Alternativ 2 kan ha sitt utspring i at optokoblerne fungerer dårlig, og ikke får med seg alle tannhjulets tenner. Dette vil medføre at systemet får med seg færre ”tikk” enn det skulle, og roboten må kjøre lengre for å komme opp i riktig antall ”tikk”. Dette kan stemme, og at det prinsipielt kan fungere slik ble vist med et forsøk der den ene optokobleren ble løsnet, slik at tikk kun ble registrert på den ene siden. Roboten fikk så beskjed om å snu seg 90°. På grunn av måten roboten beregner vinkelen den har dreid[1], vil roboten nå måtte rotere dobbelt så langt, ettersom summen av absoluttverdiene til høyre og venstre antall ”tikk” vokser halvparten så fort. I eksperimentet endte også roboten med å snu seg ca. 180°.

Problemet med denne forklaringsmodellen er at resultatene er svært konsistente, altså må systemet gå glipp av nøyaktig like mange tikk per omdreining hver gang. Dette ville vært tilfellet dersom ett eller begge av tannhjulene optokoblerne teller på hadde manglet en eller flere tenner. Den enkle måten å finne ut dette på, er å se på tannhjulene, og de er like hele, dermed er ikke dette problemet. Det kan for øvrig hende at det er smuss eller noe annet på tannhjulene som forårsaker samme symptomer. Hvis det er her problemet ligger, vil dette gi seg utslag i at antall tikk per hjulomdreining, $X = \frac{hjulomkrets}{X\_FACTOR}$ vil ligge under 200 med en faktor av 0,5 (altså bommer tellesystemet med minst 1 tann på minst ett av to tannhjul). Utfra siste resultat i Tabell 2, og avlest hjuldiameter, blir svaret: $X = \frac{49{,}6 \cdot \pi}{0{,}8315} = 187{,}3999$, som avviker fra 200 med 12,6001. Dette stemmer for dårlig med premisset (rundt 12,5 hadde passet bedre) til å bekrefte denne feilkilden, men samtidig er reell hjuldiameter ganske usikker, så feilkilden kan ikke avvises.

Ved å bruke data fra siste test i Tabell 3 fremkommer svaret $X = \frac{49{,}6 \cdot \pi}{0{,}7587} = 205{,}3816$. Dette ligger godt over 200, som skulle tilsi at optokoblerne i dette tilfellet oppfatter *mer* enn 200 tenner per hjulomdreining (ca. 11 mer). Dette er nok ikke tilfelle; det store tallet skyldes

[1] Vinkelendringen, $\delta\theta = \frac{s_h - s_v}{D}$,
der D = akselbredde, $s_i = \delta T_i \cdot X\_FACTOR\_ROTATE$ er buelengden tilbakelagt av høyre ($i = h$) og venstre ($i = v$) hjul, mens δT_i er antall registrerte tikk på hjul i siden rotasjonen startet.

sannsynligvis heller at hjulene spinner litt når roboten roterer. I dette tilfellet trekker altså feilen i motsatt retning.

Konklusjonen blir: For å sørge for best mulig resultat er det fortsatt er nødvendig med ulik X_FACTOR for translasjon og rotasjon.

3.1.3.2. Kompensering av feil som følge av individuelle drivlinjekarakteristikker

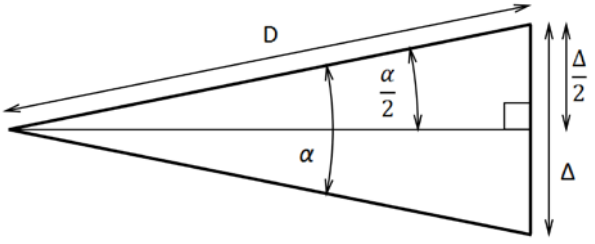
Roboten har i prinsippet to uavhengige drivlinjer; fra signalene går fra mikrokontrolleren, via motorstyringskortet, og til motorene, gir og drivhjul. Som nevnt i kapittel 2.1.4 *Forbedringspotensiale* er dette problematisk ettersom motorene etter alt å dømme er produsert med relativt store toleranseverdier, og følgelig har tydelige individuelle forskjeller. Dette fører til at roboten driver av kurs når den beveger seg over større distanser.

Ettersom motorene drives individuelt helt fra mikrokontrolleren, bør det la seg gjøre å implementere en enkel regulator i mikrokontrolleren for å kompensere for denne feilen. Ved å benytte data fra optokoblerne, som allerede er tilgjengelig til bruk i navigasjonsberegningene, kan feildeteksjonen implementeres med en oppløsning ned mot ca. 0,2° [2]. Den forholdsvis gode oppløsningen tilsier at en slik regulator kan bedre robotens bevegelsesnøyaktighet betraktelig.

Slik mikrokontrollerens programvare er satt opp per dags dato, beveger roboten seg på én av fire forskjellige måter hvis den ikke står i ro; rett fremover, rett bakover, roterer mot høyre eller mot venstre. Dette gjør at regulatoren bare har fire forskjellige tilstander å ta hensyn til. I Tabell 4 er disse tilstandene listet opp sammen med det regulatoren må undersøke, og påfølgende aksjoner. T_h og T_v viser til antall ”tikk” registrert på henholdsvis høyre og venstre hjul, etter at roboten startet den gjeldende bevegelsen. Det er lagt til et slingringsmonn på 1, da det er sannsynlig at høyre og venstre optokobler ikke observerer et ”tikk” nøyaktig samtidig, så en forskjell på 1 mellom $|T_h|$ og $|T_v|$ kan oppstå ved et gitt samplingspunkt, selv om hjulene har beveget seg nøyaktig like langt i sin respektive retning.

---

[2] Oppløsningen er beregnet utfra følgende ligning: $\alpha = 2\sin^{-1}\left(\frac{\Delta}{2D}\right)$
der α = minste observerbare avvik i grader, Δ = distanse per ”tikk”=$\frac{hjulomkrets}{\text{"tikk" per hjulomdreining}}$ og D = akselbredde.
Et ”tikk” oppstår når en tann fra et tannhjul passerer en optokobler, og med gjeldende girutveksling og omtrentlige dimensjoner blir tallene
Δ = $\frac{156\ mm}{200}$ = 0,78 mm, D = 210 mm ⇒ $\alpha = 2\sin^{-1}\left(\frac{0{,}78}{420}\right) \approx 0{,}213°$.



Tabell 4: Regulator

Tilstand	Kontroll	Aksjon
Kjører fremover	$T_h > T_v + 1$	Demp hastigheten på høyre motor
	$T_v > T_h + 1$	Demp hastigheten på venstre motor
Kjører bakover	$T_h < T_v - 1$	Demp hastigheten på høyre motor
	$T_v < T_h - 1$	Demp hastigheten på venstre motor
Roterer mot høyre	$-T_h > T_v + 1$	Demp hastigheten på høyre motor
	$T_v > -T_h + 1$	Demp hastigheten på venstre motor
Roterer mot venstre	$T_h > -T_v + 1$	Demp hastigheten på høyre motor
	$-T_v > T_h + 1$	Demp hastigheten på venstre motor

Det er hensiktsmessig å implementere denne regulatoren i mikrokontrolleren på samme måte som funksjonene i filen *position.c* (se vedlegg B). Disse funksjonene blir kalt fra hovedløkken i programvaren, og sørger dermed for å oppdatere posisjonsestimatet kontinuerlig. Regulatoren vil være avhengig av en raskest mulig oppdateringsfrekvens, slik at avvikene ikke vokser seg store, men kan korrigeres raskt, og bør derfor implementeres på en tilsvarende måte. En rask oppdateringsfrekvens vil også medføre at selve reguleringsfunksjonen kan gjøres enkel og rask, da den ikke trenger å håndtere store avvik.

Opprinnelig fungerte bevegelsessystemet til roboten ved at den mottok en ønsket posisjon og sammenlignet denne med sin egen posisjon, for så å snu seg i riktig retning og kjøre til destinasjonen. I det en bevegelse var igangsatt, ville den bli fullført uten noen form for dynamisk regulering av robotens trajektor, slik at jo større distanse roboten skulle bevege seg, desto større ble feilen i robotens sluttposisjon i forhold til destinasjonen.

Det er viktig å understreke at denne feilen *ikke* påvirker nøyaktigheten i robotens kartlegging, men kan være et problem dersom roboten skal bevege seg i trange omgivelser, eller på andre måter er avhengig av å være presist posisjonert i forhold til destinasjonen. Posisjonsestimatet beregnes separat fra robotens bevegelser, så ved fullført bevegelse vet roboten hvor den er, uavhengig av om den har beveget seg unøyaktig og havnet et annet sted enn destinasjonen.

For å kunne regulere robotens bevegelser underveis, ble en enkel regulator implementert i filen *navigation.c* (se vedlegg A). I utgangspunktet ble funksjonen *robotDrive* kalt fra hovedløkken dersom roboten skulle bevege seg fremover, altså ble denne funksjonen kalt med rask frekvens. *robotDrive* sjekket så om bevegelseskommandoen var ny, og i så fall ba den funksjonen *lego_drive* sette retning fremover på begge motorene og slå dem på. Dersom kommandoen ikke var ny, ville *robotDrive* sjekke om den hadde beveget seg langt nok, og i så fall be *lego_drive* slå av motorene. Tilsvarende ble funksjonen *robotRotate* kalt fra hovedløkken, for så å gjøre sine beregninger og be *lego_drive* sette retning på motorene og slå dem av eller på.

Programflyten blir dermed slik:

Hovedløkke → *robotDrive* eller *robotRotate* → *lego_drive*

Det ble besluttet å implementere regulatoren *robotControl* som en egen funksjon før *lego_drive*, slik:

Hovedløkke → *robotDrive* eller *robotRotate* → *robotControl* → *lego_drive*

I tillegg ble *robotDrive* og *robotRotate* modifisert til å kalle på *robotControl* hver gang de selv ble kalt på, i stedet for *lego_drive* ved start eller slutt av en bevegelse. På denne måten vil *robotControl* bli kjørt med en tilstrekkelig høy frekvens. Som et ekstra sikkerhetstiltak – i tilfelle mikrokontrollerens hovedprosessor skulle bli opphengt med noe annet, og ikke være i stand til å kjøre hovedløkken med høy nok frekvens – vil en av mikrokontrollerens avbruddstimere kalle på *robotControl* ca. 20 ganger i sekundet.

robotControl fungerer på enklest mulige måte, nemlig ved å slå av eller på motorene individuelt, basert på testene i Tabell 4.

Et enkelt eksperiment, der roboten ble bedt om å bevege seg rett fremover én meter med og uten regulering ble gjennomført fem ganger, for å se om regulatoren hadde noen virkning. Posisjonsestimatet ble så notert ned i Tabell 5 og Tabell 6.

Tabell 5: Data fra regulatoreksperiment uten regulering

Tur nr.	X [mm]	Y [mm]	Θ [°]
1	999	-33	355,8641
2	998	-50	354,7755
3	998	-58	353,4004
4	999	-48	354,7755
5	999	-19	356,3225
Gjennomsnitt	998,6	-41,6	355,0276
Gjennomsnittlig avvik	**-1,4**	**-41,6**	**-4,9724**

Det kommer tydelig frem at venstre motor roterer noe raskere enn høyre ved samme spenning. Dette gir seg utslag i feilposisjonering, der roboten gjennomsnittlig har beveget seg over fire cm for langt til høyre. Samtidig teller posisjonsberegningen ”tikk” på hjulene, og stopper roboten når den har beveget seg én meter. Med avviket sidelengs kommer den seg altså ikke langt nok fremover, selv om dette avviket er så lite at det kan neglisjeres.

Tabell 6: Data fra regulatoreksperiment med regulator

Tur nr.	X [mm]	Y [mm]	Θ [°]
1	1000	-4	0
2	1000	-4	359,531
3	1000	-3	359,7029
4	1000	-4	359,531
5	1000	-4	359,531
Gjennomsnitt	1000	-3,8	359,6592
Gjennomsnittlig avvik	**0**	**-3,8**	**-0,34082**
Forbedring	**100 %**	**91 %**	**93 %**

Det kommer frem at regulatoren forbedrer ytelsen betraktelig. Det er ingen overraskelse, da regulatoren og posisjonsestimatoren benytter seg av samme datasett, nemlig antall ”tikk” fra optokoblerne. På tross av tydelig forbedring, er det fremdeles en feil på rett under fire mm mot høyre. Dette er fordi regulatoren ikke oppdager og kompenserer en feil før det er forskjell i antall ”tikk” registrert på høyre og venstre side. Når roboten konsekvent vrir seg mot høyre for så å bli vridd i riktig retning igjen, vil den i praksis kjøre i sikksakk skrått mot høyre. Dette kunne vært forhindret ved å la regulatoren følge en gitt trajektor i stedet for å bare sammenligne ”tikk”, men en slik regulator ville blitt mer komplisert, og følgelig kostet kjøretid på mikrokontrolleren. Roboten vil svært sjelden bevege seg så langt som en hel meter før den stanser eller svinger (og dermed får et oppdatert posisjonsestimat), og med et avvik på under fire mm til siden per meter bevegelse blir konklusjonen at regulatoren er god nok for oppgaven.

3.1.4. Oppdatering av grensesnitt

Grensesnittet har en innebygget mekanisme som holder orden på akkumulert feil, basert på robotens hjuloppsett (se vedlegg C, LineBeaconSLAM.m). Dette fungerer ved at systemet antar en konstant feilkoeffisient (*error growth coefficient, EGC*) med enhet [1/m]. Denne var i utgangspunktet satt til 0,03, altså et avvik på 30 mm per meter tilbakelagt, som er i riktig størrelsesorden, om enn litt lavt sammenlignet med eksperimentdata i Tabell 5 (gjennomsnittlig 41,6 mm).

Ettersom roboten er bygget om, må konstantene for hjuloppsett endres. Derfor har matlabkodens hovedløkke blitt endret til å reflektere dette, samtidig som *EGC* har blitt satt betraktelig lavere. Dette kan rettferdiggjøres med at systemet nå har aktiv feilkorrigering på et lavere nivå, og dermed får bedre sikkerhet i posisjonsestimatet. *ECG* ble satt til 0,004 basert på resultatene i Tabell 6 (se vedlegg C, LineBeaconSLAM.m).

3.1.5. Batteriberegninger

Automatikk for tilbakevending til ladestasjonen er implementert av (Kristiansen, 2009). Det er for øvrig ingen automatikk i når roboten bestemmer seg for å returnere, dette må settes av operatøren. For å øke nytteverdien av denne funksjonaliteten er det ønskelig å finne en løsning som lar roboten selv informere systemet om at den trenger lading. Dette kan gjøres på hovedsakelig to forskjellige måter.

Den beste løsningen vil være å la robotens mikrokontroller aktivt overvåke batteriet. Dette kan gjøres ved å benytte mikrokontrollerens innebygde ADC. Denne er allerede i bruk til IR-sensorene, men det er flere ledige kanaler som kan tas i bruk (Atmel, 2011a). For å gjøre beregningen av gjenværende batterikapasitet nøyaktigst mulig, vil en sanntidsovervåking av både strømbruk og batterispenning være ønskelig. En enklere løsning er å kun overvåke batterispenningen, og sammenligne med spenningen fra en matematisk modell for gjenværende batterikapasitet. Problemet med disse løsningene er at batterispenningen ligger på ca. 11,1 V, mens referansespenningen er på 5 V slik hovedkortet er designet nå (se vedlegg D, Figur 17 og Figur 18). Målt spenning alltid vil ligge mellom 0 og referansespenningen, noe som fordrer enten redesign av hovedkortet, eller i det minste en spenningsdeling før avlesningen av batterispenningen. Dette vil uansett bli en for omfattende utvidelse, og er derfor ikke benyttet her, men foreslått i kapittel 6 Videre arbeid.

En annen løsning er å lage en simuleringsmodell av batteriet, og la denne kjøre samtidig med roboten. Det er i hovedsak to problemer med denne måten å gjøre det på: For det første vil batteriets karakteristikker utvikle seg over tid, så modellen må rekalibreres med jevne mellomrom eller ”huske” hvor gammelt batteriet er, og så modellere langtidsdynamikken på en nøyaktig måte, noe som vil kreve en nokså avansert modell. Det andre problemet oppstår når roboten slås på uten å være fullt oppladet. Da må simuleringen ta hensyn til dette og bruke andre initialverdier enn ellers, som krever at modellen må vite hvor mye strøm som har blitt brukt siden sist lading, selv om roboten har blitt slått av, og simuleringsprogramvaren avsluttet. Dette problemet gjentar seg ved ufullstendig opplading, eller lading med roboten avslått.

Det er også nødvendig å velge et fornuftig miljø å simulere batteriet i. Dersom en avansert modell er ønskelig, er det lite gunstig å implementere denne i robotens mikrokontroller, da det vil koste mye kjøretid. Ettersom robotens grensesnitt kjøres fra Matlab, er det en mulighet å simulere batteriet her. Utvidelsen *SimPowerSystems* er kommersielt tilgjengelig, og har en ferdig, avansert batterimodell tilgjengelig (Mathworks, 2011). Dette vil nok kunne gi en god matematisk modell, men er en lite prisgunstig løsning. En egenprodusert modell vil i så måte være bedre, enten den er implementert i Matlab eller i robotens mikrokontroller. Fordelen med å gjøre det i Matlab, er at man her har simuleringsredskapet *Simulink* tilgjengelig, som gjør det lettvint å lage dynamiske modeller. Fordelen med en implementasjon i mikrokontrollerens programkode, er at batterisimuleringen da blir en integrert del av roboten, uavhengig av grensesnittet. Dersom det på et senere tidspunkt

skulle være ønskelig med sanntidsovervåking i stedet for simulering, er dette noe som må gjøres fra mikrokontrolleren, og da vil en modell implementert her gjøre endringen mindre omfattende. Samtidig vil en modell implementert i mikrokontrolleren ha umiddelbar tilgang til alle data fra roboten, uten å måtte få det tilsendt via Bluetooth.

Summen av disse momentene tilsier at en enkel simuleringsmodell implementert på robotens mikrokontroller vil være den mest hensiktsmessige løsningen i denne omgang.

3.1.5.1. Implementasjon

Næss (Næss, 2008), implementerte i sin tid den nåværende batteri- og ladekonfigurasjonen. Følgende data for batteripakken er oppgitt:

Tabell 7: Spesifikasjoner for batteripakke (tabellen er et utdrag av (Næss, 2008), tabell 1)

Type	Li-Ion
Spenning	11,1 V
Kapasitet	5200 mAh
Antall batterier	2 * 3 (serieparallell)
Ladetid	2-3 timer[3]

I tillegg har han gjennomført eksperimenter på robotens strømbruk:

Tabell 8: Strømbruk (hentet fra (Næss, 2008), tabell 2)

"Tomgang" (elektronikk, optokoblere)	60 mA
Fremdriftsmotorer (vanlig drift)	2 * 20 mA = 40 mA
IR-sensorer	4 * 30 mA = 120 mA

Dette gir en teoretisk driftstid på 86,7 timers "tomgang", 52 timers kjøring uten IR-skanning, 29 timers skanning uten kjøring, eller 23,6 timer med både skanning og kjøring. Det er sannsynlig at disse tallene er for store i forhold til virkeligheten, både fordi Næss foretok målingene med et upresist instrument og påpeker dette som en feilkilde (se(Næss, 2008), s. 7), og særlig fordi Li-Ion-batterier mister kapasitet ettersom tiden går (Cadex Electronics, 2011)[4].

[3] Dette stemmer ikke. Reell ladetid er ca. fem timer, som vist senere i teksten.
[4] Denne kilden er ikke fullstendig troverdig, men ettersom påstandene som dokumentasjonen underbygger uansett verifiseres senere i den grad de påvirker problemstillingen, har ikke kildens påstander innvirkning på resultatene.

Selv om tallene i Tabell 8 har stor usikkerhet knyttet til seg, er det trolig at de illustrerer et noenlunde riktig bilde av forholdet i strømbruk mellom de forskjellige komponentene. Det medfører at følgende lineære kapasitetsmodell kan kalibreres kun ved å endre koeffisientene K_Q og K_i:

$$Q(t) = K_Q \cdot Q_{max} - K_i \cdot it \qquad 3\text{-}1$$

der *Q* er øyeblikkskapasiteten, Q_{max} er batteriets maksimalkapasitet, *i* er strømbruk og *t* er tid. Strømbruk kan beskrives slik:

$$i = i_1(\sigma) + i_2(\sigma) + i_3(\sigma) \qquad 3\text{-}2$$

der i_k viser til linje *k* i Tabell 8, og σ er robotens tilstand.

Det vil være mulig å modifisere modellen til å ikke behøve jevnlig rekalibrering av K_Q ved å sette $Q_{max} = Q_{max}\big(t, Q(t)\big)$, der batteriets maksimale kapasitet modelleres som funksjon av tid og oppladningsgrad/gjenværende kapasitet (i følge (Cadex Electronics, 2011) vil den totale kapasitetsendringen til en hver tid være avhengig av oppladningsgraden[4]). Dette fordrer at roboten kjenner funksjonen $Q_{max}(t, Q)$, for eksempel fra tabelloppslag eller en god matematisk modell. I tillegg må roboten vite alderen på batteriet, altså må den ha greie på hvilken dato det er. Dette betinger at roboten må utstyres med en klokke som går også når batteriet er frakoblet og mikrokontrolleren avslått, og fordrer derfor endringer av hovedkortet. Hvis maskinvaren først skal modifiseres, vil det være en bedre løsning å la mikrokontrolleren overvåke batterispenning og strømbruk direkte, så denne løsningen er verken enkel nok å implementere sammenlignet med en enkel simulering, eller nøyaktig nok til å forsvare et større implementasjonsarbeid (det antas at direkte målinger gir større nøyaktighet enn selv en avansert simulering) . Det er mulig å gå rundt problemet ved å la Matlab-grensesnittet sende sin systemtid til roboten. Dette vil da fungere som en automatisk kalibrering av batterikapasiteten, og må i tilfelle gjøres hver gang grensesnittet kobles til roboten.

Det største problemet med tanke på simulering av batterikapasiteten er at roboten kan lades opp uten å være slått på. Dermed har ikke mikrokontrolleren oversikt over hvor lenge sist lading varte, og har dermed ingen informasjon om riktig initialverdi for gjenværende batterikapasitet $Q(t)$ når den slås på. Det finnes to løsninger på dette problemet; en robust og en lettvint. Den robuste løsningen består i å bygge om roboten slik at mikrokontrolleren styrer ladingen, og dermed har full oversikt over denne prosessen. Dette er fullt gjennomførbart, men en for omfattende jobb i denne omgang, og det er derfor foreslått i kapittel 6 Videre arbeid. Den lettvinte løsningen er å manuelt sørge for at batteriet lades fullt opp hver gang. Da kan roboten anta $Q(t)\big|_{t=0} = K_Q \cdot Q_{max}$.

For å gjøre implementasjonen enkel, men med mulighet for å kunne utvides ved behov, implementeres den enkle modellen ved å la en av mikrokontrollerens timere holde rede på hvor lang tid det har gått siden roboten ble slått på (og dermed siden den var fulladet). Med en gitt frekvens sjekker denne timeren robotens tilstand og oppdaterer kapasitetsfunksjonen med riktig strømbruk. Når kapasiteten har nådd en nedre grenseverdi må grensesnittet få beskjed om dette, slik at hele systemet er klar over at roboten skal lades før Kristiansens lade-automatikk (Kristiansen, 2009) settes i sving. Modellen implementeres ikke med dynamisk Q_{max}, men det vil være en naturlig fremtidig utvidelse av (Cadex Electronics, 2011)(Cadex Electronics, 2011)modellen, og er listet i kapittel 6 Videre arbeid. I denne omgang er det altså modellen i ligning 3-1 som implementeres.

Selve implementasjonen er selvsagt i diskret tid, så ligning 3-1 diskretiseres til følgende:

$$Q(k) = K_Q \cdot Q_{max} - K_i \cdot i(k) \cdot \tau \cdot k, \qquad k \in \{0, \mathbb{N}\}$$

3-3

Der *k* representerer tidsskritt og τ er varigheten av et tidsskritt. For å gjøre implementasjonen raskere, med en separat initialisering ved $k = 0$, deles systemet i to ligninger på denne måten:

$$Q(k) = \begin{cases} K_Q Q_{max}, & k = 0 \\ Q(k-1) - K_i i(k)\tau, & k \in \mathbb{N} \end{cases}$$

3-4

3.1.5.2. Kalibrering av K_Q

For å undersøke hva kalibreringskoeffisienten K_Q bør settes til, ble robotens kildekode modifisert til å la den konstant skanne sine omgivelser. Den skulle i teorien da holde på i ca. 29 timer, men det antas som nevnt at denne verdien er for høy, og at roboten vil gå tom for strøm før den gang. Ved å løse ligning 3-1 for $Q = 0$ med hensyn på K_Q, fremkommer følgende løsning (K_i settes i denne omgang lik 1, ettersom den vil kalibreres senere):

$$K_Q = \frac{i}{Q_{max}} t = \frac{180\ [mA]}{5200\ [mAh]} t\ [h]$$

3-5

Ligningen gir kalibreringskonstanten K_Q direkte og entydig som funksjon av tiden t roboten bruker fra fullt til tomt batteri.

Eksperimentet ble gjennomført, og roboten sto og skannet omgivelsene i totalt 17,5 timer. Da var den helt tom for strøm, så det er tydelig at enten er Q_{max} betraktelig mindre enn den (teoretisk) var i utgangspunktet, eller så er Næss' målinger i Tabell 8 svært unøyaktige. Kalibreringen resulterer uansett i $K_Q = \frac{63}{104} \approx 0{,}606$ når $Q_{max} = 5200$ mAh.

I etterkant av eksperimentet ble batteriet ladet helt opp igjen. Det ble da tydelig at ladetiden oppgitt i Tabell 7 overhodet ikke stemmer. Laderen har et indikatorlys som skifter fra rødt til grønt når batteriet er ferdig ladet, så det er mulig å ta tiden på prosessen. Virkelig ladetid fra tomt til fullt batteri viste seg å være rundt fem timer.

3.1.5.3. Kalibrering av K_i

For å finne riktig verdi for K_i ble det gjennomført et eksperiment, der mikrokontrollerens kildekode ble modifisert til å sende et signal til grensesnittet ved to forskjellige verdier av $Q(k)$. Verdiene $Q(k_1) = 13{,}5$ og $Q(k_2) = 10$ ble valgt mer eller mindre tilfeldig, men akkurat hvilke verdier som settes her har kun med eksperimentets lengde å gjøre.

Grensesnittet ble konfigurert til å logge nøyaktig tidspunkt for disse to signalene fra roboten, slik at differansen $k_2 - k_1$ kan beregnes.

Følgende konstantverdier ble også benyttet:

Tabell 9: Simulatorkalibrering - konstanter

Konstant	Verdi i eksperimentet	Forklaring
$\boldsymbol{f}$	20 [Hz]	Simuleringsfrekvens
$\boldsymbol{K_Q}$	0,003	Ukalibrert. Satt svært lav, slik at startverdien $Q(0)$ også blir lav.
$\boldsymbol{Q_{max}}$	5200 [mAh]	Teoretisk maksimalkapasitet
$\boldsymbol{i}$	100 [mA]	Gjennom hele eksperimentet kjørte roboten fremover, slik at $i(k) = i = 60 + 20 \cdot 2 = 100 \; \forall \; k$, jf. Tabell 8
$\boldsymbol{\tau}$	$\frac{1}{3600f}$ [h]	Invers av f, men skaleres til timer for å beholde benevning [mAh] i beregninger

Eksperimentet ble gjennomført, og tidspunktene for signalene fra roboten ble logget. Differansen $t_2 - t_1$ ble 2 min. 47 sek.

$$\Rightarrow k_2 - k_1 = (t_2 - t_1) \cdot f = 167 \; [\text{s}] \cdot 20 \; [\text{Hz}] = 3340$$

3-6

Ved å sette inn for $Q(k_1)$ og $Q(k_2)$ i ligning 3-3, men bytte ut konstanten K_i mot sin invers (simuleringen er kjørt uten bruk av denne konstanten, så den fungerer i praksis invertert i ligningene), samt bruke resultatet fra ligning 3-6, kan følgende ligningssystem løses med hensyn på k_1, k_2 og K_i:

$$\begin{cases} a - \frac{1}{K_i} \cdot b \cdot k_1 = 13{,}5 \\ a - \frac{1}{K_i} \cdot b \cdot k_2 = 10 \\ k_2 - k_1 = 3340 \end{cases} \qquad \text{3-7}$$

Der konstantene $a = K_Q \cdot Q_{max}$ og $b = i \cdot \tau$.

Første ledd gir:

$$\frac{1}{K_i} = \frac{a - 13{,}5}{bk_1}$$
$$K_i = \frac{bk_1}{a - 13{,}5}$$

Innsatt i andre ledd:

$$a - \left(\frac{a - 13{,}5}{bk_1}\right) bk_2 = 10$$
$$a\left(1 - \frac{k_2}{k_1}\right) + \frac{13{,}5k_2}{k_1} = 10$$

Tredje ledd satt inn for k_2 i dette resultatet gir:

$$a\left(1 - \frac{k_1 + 3340}{k_1}\right) + \frac{13{,}5(k_1 + 3340)}{k_1} = 10$$
$$a\left(1 - \frac{k_1}{k_1} - \frac{3340}{k_1}\right) + 13{,}5 + \frac{13{,}5 \cdot 3340}{k_1} = 10$$
$$-3340a + 13{,}5k_1 + 45090 = 10k_1$$
$$\boxed{k_1 = \frac{3340a - 45090}{3{,}5}} \qquad \text{3-8}$$

Dette kan videre settes tilbake i tredje ledd av ligning 3-7 og løses for k_2:

$$k_2 = 3340 + k_1$$
$$\boxed{k_2 = 3340\left(1 + \frac{a - 13{,}5}{3{,}5}\right)}$$

3-9

Ved å sette inn ligning 3-8 i uttrykket for K_i, fremkommer løsningen:

$$K_i = \frac{b\left(\frac{3340a - 45090}{3{,}5}\right)}{a - 13{,}5}$$

$$K_i = \frac{3340b(a - 13{,}5)}{3{,}5(a - 13{,}5)}$$

$$\boxed{K_i = \frac{3340}{3{,}5}b}$$

3-10

Verdiene fra Tabell 9 satt inn i uttrykkene for a og b gir:

$$a = 0{,}003 \cdot 5200 = \underline{15{,}6}$$

$$b = \frac{100}{3600 \cdot 20} = \underline{\frac{1}{720}}$$

Dette gir endelig svar på kalibreringseksperimentet, utfra ligning 3-10:

$$K_i = \frac{3340}{3{,}5 \cdot 720} = \underline{\underline{\frac{167}{126} \approx 1{,}32540}}$$

Tidsskrittene fra ligning 3-8 og 3-9 blir:

$$k_1 = 2004$$

$$k_2 = 5344$$

Verdiene uten kalibreringskonstanten($K_i = 1$), er gitt ved

$$k_{1ref} = \frac{a - 13{,}5}{b} = 1512$$

$$k_{2ref} = \frac{a - 10}{b} = 4032$$

Som gir en teoretisk tidsdifferanse på $t_{2ref} - t_{1ref} = \frac{k_{2ref} - k_{1ref}}{f} = \frac{2520}{20} = 126$ [s]. Dette er helt tydelig for lavt (jf. eksperimentdata i ligning 3-6).

K_i ble undersøkt ved å kjøre samme eksperiment på nytt, men denne gang med kalibreringskonstanten inkludert. Tidsdifferansen ble da:

$$(t_2 - t_1) = 125{,}3 \text{ [s]}$$

Som er meget nært det teoretiske tallet på 126 sekunder. Konklusjonen blir da at strømlastens forsterkningskoeffisient, K_i, er godt kalibrert.

3.1.5.4. Lading

Ettersom Matlab-grensenittet alt er utvidet med en lade-trigger og automatisk docking til ladestasjonen, er det naturlig å la denne oppførselen styres av grensesnittet. Dette medfører at protokollen mellom robot og grensesnitt må utvides,slik at roboten kan rapportere batteristatus, og få grensesnittet til å initiere lading. Følgende signaler ble implementert:

Tabell 10: Utvidelse av kommunikasjonsprotokoll

Til robot fra grensesnitt	Til grensesnitt fra robot
Sjekk batteri	*Batteri OK*
Lading fullført	*Batteri trenger lading*

Implementasjonen ble gjort slik at hovedløkken i Matlab-grensesnittet sender *sjekk batteri*-signalet til roboten med jevne mellomrom (se vedlegg C, LineBeaconSLAM.m). Robotens program undersøker da en boolsk semafor som forteller om batteriet trenger lading (denne semaforen settes høy av batterisimulatoren ved en gitt nedre grensekapasitet). Dersom batteriet ikke trenger lading, sendes signalet *batteri OK*, og grensesnittet går videre. Dersom batteriet må lades, sendes *batteri trenger lading*, etterfulgt av robotens ønskede ladetid i sekunder (se vedlegg A). På den måten fungerer grensesnittet generelt, og alle batteridata holdes i roboten. Grensesnittet setter nå sin egen lade-trigger høy, som gjør at ladestasjonsautomatikken igangsettes og roboten får ladet seg opp. Etter et tidsrom bestemt av ladetiden roboten sendte til grensesnittet, anses batteriet som fullstendig oppladet, og signalet *lading fullført* sendes til roboten, som da nullstiller batterisimulatoren. Samtidig blir roboten satt til å fortsette sin kartlegging fra de koordinatene hvor den avsluttet for å lades opp, som beskrevet av Kristiansen (Kristiansen, 2009). Tidligere ble roboten satt til å lade i fem sekunder, bare for å illustrere en ladesituasjon. Med implementasjonen av reell ladetid i stedet for den fem-sekunders-sekvensen som var implementert fra før, ble tiden roboten tilbringer i ladestasjonen betraktelig lengre. Grensesnittet ble derfor endret til å fortelle hvor lang tid som gjenstår til ladingen er fullført, da dette ikke var implementert tidligere. Dette kan være nyttig for å vite hvorfor roboten ikke gjør noe aktivt, slik at en lading enkelt kan skilles fra en feil.

Selve oppladingssekvensen går som følger:

1. Roboten går til det stedet den ble satt i gang (sannsynligvis i nærheten av ladestasjonen)
2. Roboten forsøker å identifisere ladestasjonen i omgivelsene
3. Når ladestasjonen er identifisert svinger roboten, slik at baksiden er mer eller mindre parallell med ladestasjonens bakvegg, der det er påmontert strømførende skinner.

4. Roboten rygger, til den er i kontakt med ladestasjonen, men fortsetter å ha pådrag bakover, slik at hjulene spinner, og roboten dermed retter seg inn etter bakveggen av seg selv, og blir stående helt parallelt med denne.
5. Roboten lader
6. Roboten fortsetter arbeidet den avsluttet for å lade

Etter endringene foretatt i kapittel 3.1.2, der drivhjulenes bakkekontakt ble kraftig forbedret, er det innlysende at punkt 4 i listen ikke lenger nødvendigvis fungerer etter hensikten. Før disse endringene hadde roboten hjul som lett kunne spinne mot underlaget, og tyndepunktet lå langt bak hjulakslingen. Da ladeprosedyren ble igangsatt på den ombygde roboten ble det klart at i stedet for å rette seg inn etter ladestasjonen, begynte den å klatre oppover veggen med baksiden, før den tippet fremover og veltet. For å motvirke dette, ble den påmontert støtter i forkant av drivhjulene, slik at den ikke hadde mulighet for å tippe fremover. Likevel ville hjulene bare spinne nok til at roboten fikk rettet seg inn etter bakveggen; da dette var gjort, hadde ikke motorene nok moment til å holde hjulene i gang. Roboten hadde fått beskjed fra grensesnittet om å rygge helt til hjulakslingen (robotens nullpunkt) lå parallelt med ladestasjonens bakvegg. Dette er selvsagt umulig i virkeligheten, men ettersom roboten spant, oppfattet den selv at den var i bevegelse. Når hjulene nå ikke kunne spinne, ville roboten dermed bli stående og forsøke å rygge, uten at grensesnittet kunne oppfatte at den hadde nådd ladestasjonen, og rettet seg inn etter den. Løsningen ble å endre kriteriet for å fortsette ryggingen. I utgangspunktet var det eneste kriteriet at robotens posisjon langs aksen vinkelrett på ladestasjonens bakvegg var større enn bakveggens posisjon (positiv retning ut fra veggen defineres av robotens startposisjon, som antas med fronten ut fra veggen). Det ble lagt til et tilleggskriterium, der grensesnittet sjekker robotens nåværende posisjon mot en tidligere posisjon. Dersom det ikke har vært noen endring, anses roboten for å være docket i ladestasjonen. Denne løsningen fungerte etter intensjonen, slik at den automatiske ladeprosedyren igjen fungerte etter hensikten.

3.2. Kamera-robot

Kameraroboten er ikke forbedret, selv om den måtte bygges opp på nytt, og dermed fikk et noe endret utseende (se avsnitt 2.1.2). At den nå skal kunne håndteres av Matlab-grensesnittet har for øvrig gitt en del utfordringer, nærmere beskrevet i kapittel 4.3. De største endringene gjennomgått i avsnitt 4.3.1.

3.3. Resultater oppsummert

- IR-roboten er forbedret på en rekke områder, og har som følge av det fått forbedret presisjon i sine bevegelser.
- Det har blitt implementert en batterisimulator i IR-robotens mikrokontroller, og kommunikasjonsprotokollen har derfor blitt utvidet til å håndtere informasjon om batteristatus. Dette fungerer etter intensjonen.

Se for øvrig vedlegg A for all nyprodusert C-kode.

4. Utvidelse av grensesnittet

4.1. Teoretisk grunnlag for kartgenerering

4.1.1. SLAM

Ettersom kartgenereringen kun forholder seg til data fra roboten i systemet, blir all posisjonsestimering et resultat av robotens relative posisjon. Dersom systemet kunne overvåke robotens absolutte posisjon, for eksempel ved hjelp av videokameraer, ville robotens posisjon og dermed kartgenereringen fungert mer presist. Feilene som følger av robotens unøyaktigheter i odometri øker kumulativt og kan bli et problem ved lengre turer, selv om de er minimert med forskjellige forbedringer diskutert i kapittel 3.1. En ekstern observasjon av robotens absolutte posisjon ville eliminert en slik kumulativ feilkilde, men er altså ikke tilgjengelig her. Hovedutfordringen i kartleggingen av omgivelsene blir dermed at roboten starter ved en ukjent lokasjon i et ukjent landskap, og så gradvis må produsere et kart over omgivelsene, samtidig som den til enhver tid holder rede på sin egen posisjon i dette kartet (Durrant-Whyte & Bailey, 2006). Denne problemstillingen kalles *simultaneous localization and mapping* (SLAM).

I det eksisterende systemet benyttes et rammeverk som inneholder alle de nødvendige algoritmene for SLAM, *The CAS Robot Navigation Toolbox* (Arras, 2004a). Hensikten med dette rammeverket er å redusere problemet med kumulativ feil som følge av odometriberegninger. Dette gjøres ved at et *Extended Kalman Filter (EKF)* holder rede på en rekke landemerker som roboten har observert. Filteret bruker disse landemerkene, sammen med odometriske målinger for å korrigere/forbedre robotens posisjonsestimat, og på den måten å holde nøyaktigheten best mulig. Kort oppsummert fungerer SLAM på følgende vis (fritt etter Tusvik (Tusvik, 2009)):

1. Oppdatering av nåværende posisjonsestimat ved hjelp av odometri
2. Oppdatering av posisjonsestimatet ved hjelp av landemerker som har blitt observert tidligere
3. Landemerker som ikke er tidligere observert blir lagt til i EKF

Magnussen har en oppsummert gjennomgang av rammeverket, og prinsippene bak, så for mer informasjon rundt de tekniske aspektene ved dette henvises til (Magnussen, 2008). Det teoretiske grunnlaget er publisert av Durrant-Whyte og Bailey (Durrant-Whyte & Bailey, 2006), og er en mer inngående forklaring av SLAM-problematikk og løsninger.

4.2. Systemets virkemåte

Arras' SLAM-rammeverk er grunnlaget for kartgenereringen i systemet, og er dermed en integrert del av systemets virkemåte. Dette gjør at det er hensiktsmessig å benytte rammeverket i en umodifisert utgave, og heller tilpasse annen programkode slik at systemet fungerer enn å endre rammeverket, for på den måten å la SLAM-rammeverket være upåvirket, og helt i henhold til dokumentasjonen (Arras, 2004b).

Figur 19 (se vedlegg D) viser systemets oppbygging, slik det ble strukturert av Magnussen (Magnussen, 2008), og programflyten kan beskrives slik (fritt etter (Tusvik, 2009), 4.2.7 og (Arras, 2004b)):

- Startknappen i grensesnittet trykkes, som fører til at nødvendige parametre for SLAM blir satt, og valg fra brukergrensesnittet blir lagret. Dette gjøres av funksjonen *LegoGui* (se vedlegg C, LegoGui.m).
- LegoGui kaller så på *lineBeaconSLAM*, som inneholder systemets hovedløkke (en while-løkke som kjøres til kartgenereringen er komplett, systemet stanses av brukeren, eller det oppstår en feil).

Hovedløkken går så iterativt gjennom følgende trinn:

1. Robotens posisjon blir innhentet av metoden *getRobotPose* (se vedlegg C, getRobotPose.m), som gir en radvektor med verdier for robotens posisjon og orientering, henholdsvis x, y og θ. Dette blir så lagret som robotens nye posisjon.
2. SLAM-rammeverket bruker metoden *predict* til å forutsi hvor roboten nå befinner seg i omgivelsene. Dette er implementert med en usikkerhetsparameter, *EGC* (*error growth coefficient*), som forteller systemet hvor stor usikkerhet det er i posisjonen, basert på unøyaktigheter i robotens bevegelser (se også 3.1.4 og 4.3.1).
3. Det globale kartet oppdateres med informasjon om robotens nye posisjon ved hjelp av SLAM-rammeverkets *robotdisplacement*-metode.
4. Omgivelsene skannes (forutsatt at det ikke er krysset av for *Drive & scan* i brukergrensesnittet) ved hjelp av *fullscan* (vedlegg C, fullscan.m). Denne metoden returnerer en n X 3-tabell med n målepunkter, der første kolonne sier noe om hvilken sensor som ble benyttet til dette målepunktet, andre kolonne inneholder målepunktets x-koordinat, og tredje kolonne inneholder y-koordinaten.
5. Data fra *fullscan* benyttes av metoden *extractlines* i SLAM-rammeverket, for å identifisere linjesegmenter (vegger). Disse linjene lagres i et lokalt kart som så benyttes av *predictmeasurement* for å oppdatere det globale kartet. Dette foregår ved at linjene sammenlignes med linjer i det globale kartet, og der disse sammenfaller innen en viss feilmargin, slik at de sannsynligvis tilhører samme

linje/vegg, blir de slått sammen. Linjer i det lokale kartet som ikke sammenfaller med vegger i det globale, blir lagt til som nye vegger.

6. Posisjonsestimatet for roboten blir oppdatert basert på linjesegmentene i det globale kartet. Dette gjøres av et *Extended Kalman Filter* (EKF) via SLAM-rammeverkets metoder

7. De datapunktene som ikke inngår i linjesegmenter kan bli registrert som beacons i det globale kartet, forutsatt at datapunktet er mellom 10 og 40 cm fra roboten og at det er minst tre målinger i nærheten av hverandre. Dette foregår på samme måte som for linjesegmenter, ved at et lokalt kart sammenlignes med det globale, og punktene slås sammen, eller blir registrert som nye beacons.

8. Robotens posisjonsestimat blir nok en gang oppdatert av EKF, denne gang basert på eventuelle nye målinger av eksisterende beacons.

9. Robotens posisjonsestimat lagres i *navData.robotPath*, slik at grensesnittet har kontroll på den stien roboten har fulgt.

10. Det oppdaterte globale kartet blir tegnet opp visuelt i brukergrensesnittet, der både linjesegmenter, beacons, robotens posisjon og dens bevegelsessti hittil er tegnet inn.

11. Roboten får beskjed om hvor den skal bevege seg, regnet ut av metoden *leftWallFollowerToBackTracker*. Beskjeden den mottar er slik at den først får beskjed om å snu seg i riktig retning før den får beskjed om å kjøre rett frem. Dersom roboten ender opp i et område som allerede er kartlagt, vil den gå i backtracking-modus, som vil si at den følger sin egen sti bakover til den kommer til et område som ikke er kartlagt. Stien hentes ut fra *navData.robotPath*. Når roboten har gått tilbake dit den skal, blir den bedt om å skanne det ukjente området, før den fortsetter med sin normale navigasjon.

12. Robotens batteri undersøkes, og dersom det trengs lading, blir roboten sendt tilbake til ladestasjonen. Når ladingen så er fullført, kjører roboten tilbake dit den var da den fikk beskjed om å returnere til ladestasjonen.

Når et lukket område er ferdig kartlagt vil roboten returnere til ladestasjonen og anse kartleggingen som ferdig.

4.3. Endringer

Ettersom systemet i utgangspunktet er basert på det generelle SLAM-rammeverket, som beskrevet av Arras (Arras, 2004a), burde det la seg gjøre å inkludere en ny robot i det samme systemet. Det har for øvrig vist seg at systemet gjennom flere prosjekter har blitt mer og mer formsydd til IR-roboten, noe som gjør denne utvidelsen betraktelig mer komplisert. Det som

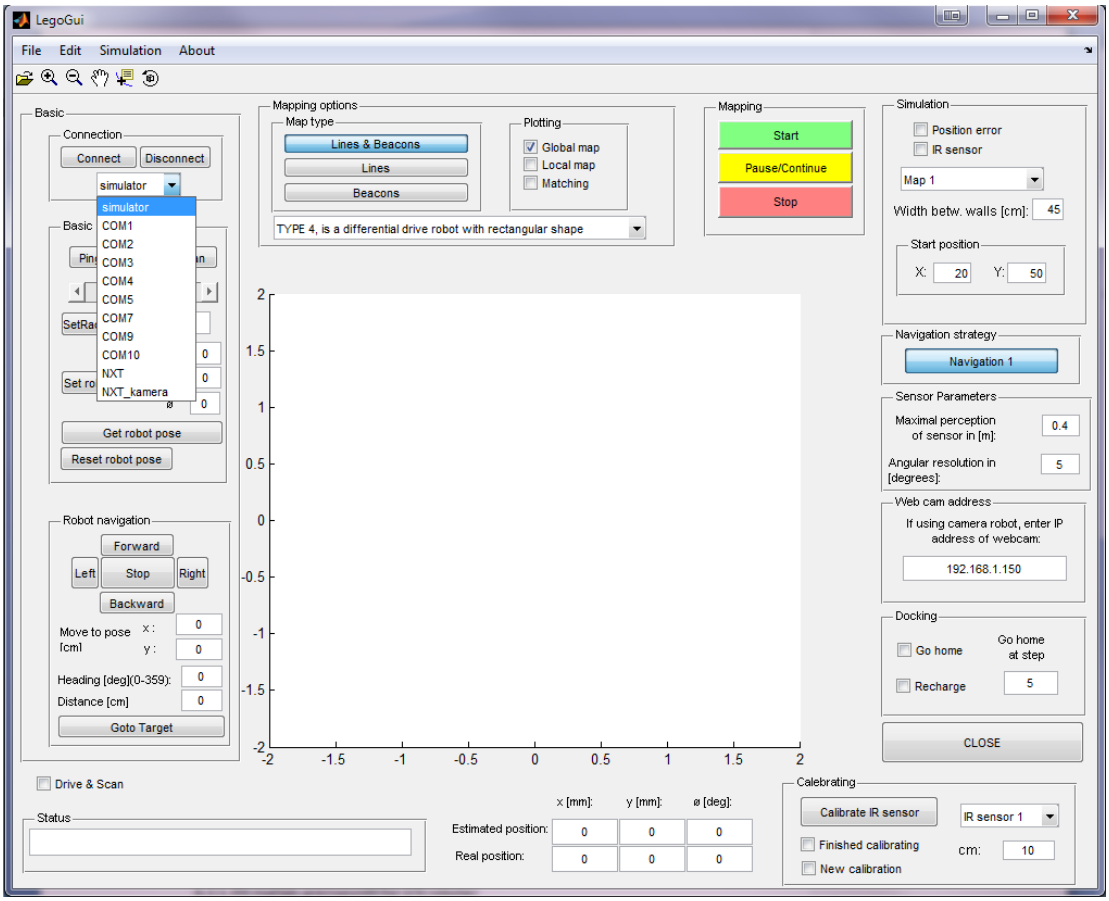
i utgangspunktet burde være en relativt enkel implementering av en ny robot, har vist seg å bli en øvelse i debugging, der svært mange metoder og filer må ”regeneraliseres” for å fungere med både IR-roboten og kamera-roboten. Denne generaliseringsprosessen har som mål å utvide systemet til å kunne kontrollere flere ulike roboter av ulik art, slik at systemet igjen vil bli brukbart med kamera-roboten, som også er idéen bak Arras’ SLAM-rammeverk (Arras, 2004b).

4.3.1. Oppdatering av grensesnitt

Ettersom grensesnittet er bygget opp med IR-roboten for øye, må koden endres mange steder. Som beskrevet i avsnitt 3.1.4, regner grensesnittet på hvor mye roboten har beveget seg feil, akkumulert over tid. Denne beregningen tar utgangspunkt i en konstant feilkoeffisient (*error growth coefficient, EGC*), som forteller hvor stor feil som oppstår per meter tilbakelagt strekning. For å finne ut hva denne koeffisienten burde settes til, ble roboten bedt om å kjøre rett frem én meter. Resultatet var et voldsomt drag mot venstre, slik roboten endte ca. 20 cm til venstre for der den burde endt opp. Eksperimentet ble gjentatt flere ganger, uten store avvik fra dette tallet, så EGC ble satt til 0,200 [1/m] (se vedlegg C, LineBeaconSLAM). Dette er et svært stort tall, og betyr at måleusikkerheten er stor, men så lenge systemet er klar over dette, via EGC-koeffisienten, vil det tas hensyn til i kartleggingsprosessen. Det er for øvrig klart at dette er en uholdbar feil, som kunne rettes opp på tilsvarende vis som det ble gjort med IR-roboten i avsnitt 3.1.3.2, men det er ikke en del av denne tekstens omfang, og er derfor nevnt i kapittel 6 Videre arbeid.

Ettersom EGC alt implementerer usikkerhet i kartgenereringen, er ikke dette noe som har blitt videre utvidet jf. oppgaveteksten. Kalmanfilteret tar hensyn til denne usikkerheten, og forbedrer dermed det globale kartet med dette i ”bakhodet”. Derfor vil det være relativt greit å benytte denne usikkerhetsinformasjonen på et senere tidspunkt.

Visuelt er grensesnittet kun modifisert i liten grad. Listen med mulige tilkoblinger er utvidet med ”NXT_kamera” som sikter til kamera-roboten. I tillegg har et felt med IP-adresse til kameraet også blitt inkludert (se Figur 14).



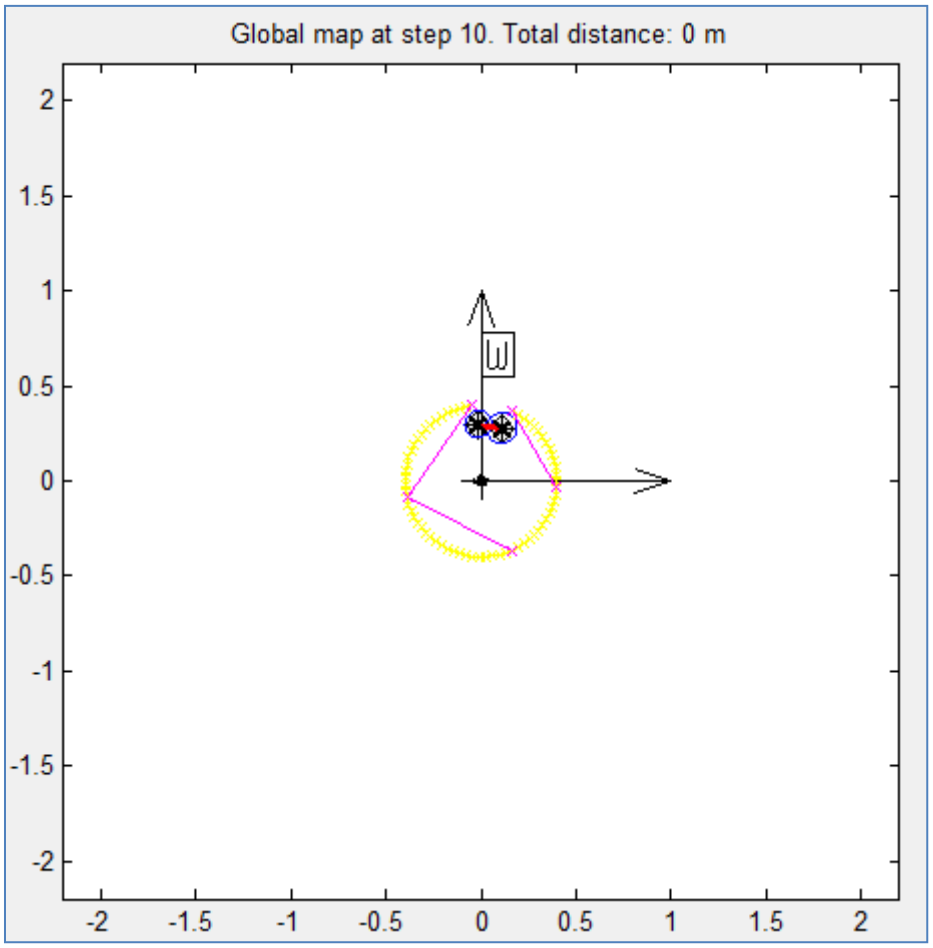
Figur 14: Grafisk brukergrensesnitt

4.3.2. Skanning av omgivelsene

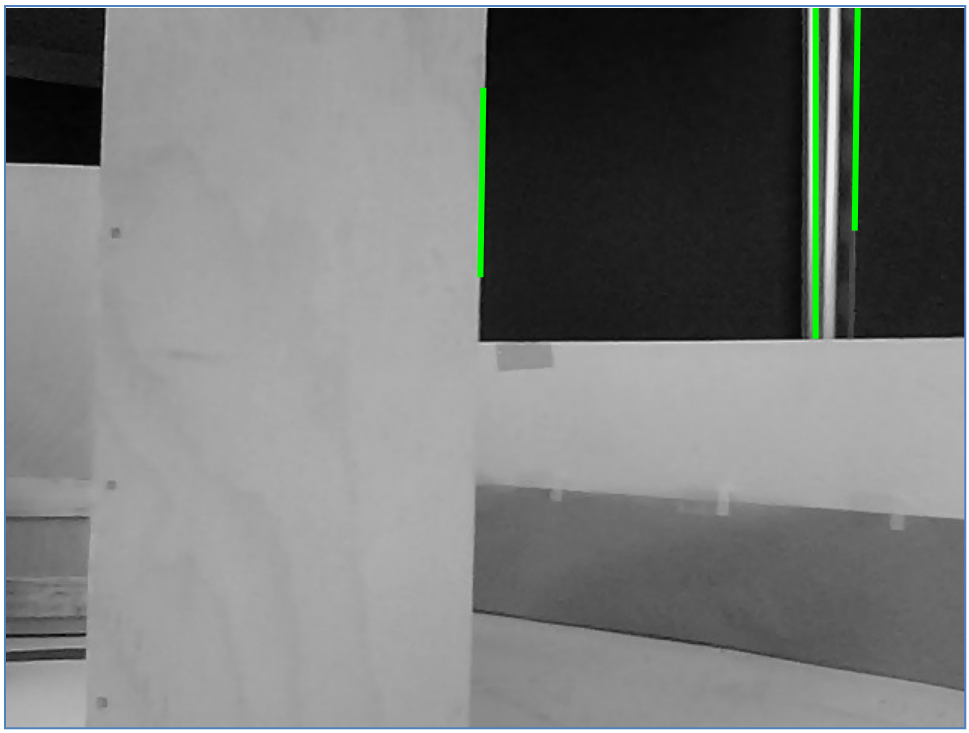
fullscan er en metode laget for å skanne omgivelsene (se vedlegg C, fullscan.m). Den er utvidet til å fungere med kamera-roboten, ved å benytte eksisterende metodikk (Tøraasen, 2009), og tilpasse måledata til det formatet som benyttes. Dette betyr at Kalman-filteret kan håndtere data fra begge robotene, også samtidig dersom det skulle bli nødvendig. Kamera-robotens metoder registrerer kun hjørner, og kun ett hjørne for hver skanning, i motsetning til IR-roboten som registrerer ett målepunkt for hver 5. grad, altså 72 punkter på en skanning. Dette betyr at *fullscan* kun returnerer ett punkt dersom kameraroboten benyttes, og dermed kan ikke disse skanningene brukes til å generere linjesegmenter. SLAM-rammeverket klarer for øvrig å håndtere denne begrensningen selv, og bruker målingene til beacons i stedet, slik at disse målingene kan gjøres nytte av likevel. Kalmanfilteret sørger for å optimalisere det globale kartet basert på alle målinger det mottar, og med denne utvidelsen vil målinger fra kamera-roboten kunne inkluderes. Videre benyttes tidligere målinger i kartgenereringen, slik at etter flere iterasjoner vil kartet likevel danne linjer mellom målepunktene (se Figur 15).

På grunn av en defekt ultrasonisk avstandssensor, og metoder som ikke detekterte dette på fornuftig vis, ble det brukt mye tid og energi på å feilsøke og implementere unødvendige

endringer i koden. Da feilen ble oppdaget var det dessverre for sent å få erstattet sensoren. For å vise at den virker, er det gjennomført et forsøk der distansen som egentlig skal måles av denne sensoren i stedet er hardkodet inn. Resultatet er at alle målepunkter naturligvis blir oppfattet like langt fra roboten, men Figur 15 og Figur 16 viser at systemet fungerer. Her er det oppfattet tre ulike målepunkter, men to av dem er slått sammen, da de ligger svært tett. I tillegg kommer det frem at kartgenereringen tegnet inn en linje (vegg) mellom de to observerte beacons.



Figur 15: Etter 10 iterasjoner er det dannet en linje mellom målepunktene (se også Figur 16)



Figur 16: Bilde av området som ble skannet (se også Figur 15), med detekterte hjørner i grønt

4.3.3. Andre endringer

Batterisjekken som ble implementert for IR-roboten (se avsnitt 3.1.5, samt vedlegg C, LineBeaconSLAM.m) fungerer nå også mot kamera-roboten. Løsningen er implementert ved hjelp av en ny funksjon kalt *nxt_get_battery_state* (se vedlegg C, nxt_get_battery_state.m), som undersøker om batteriet trenger lading ved å sammenligne batterispenningen mot en gitt nedre grense. Kommunikasjons-*toolboxen* (RWTH Aachen University, 2011) har allerede implementert en sanntidssjekk av batterispenningen, så det er ikke nødvendig å konstruere en simulator, slik det ble gjort for IR-roboten i avsnitt 3.1.5.

I tillegg til nevnte endringer, er det foretatt en rekke modifikasjoner i resten av koden også, men det vil ikke bli gjennomgått i detalj her, da det kun dreier seg om mindre tilpasninger. Det er kun de Matlab-filene med viktige endringer som er vedlagt (vedlegg C). De mindre endringene i koden (vedlagt i sin helhet på CD) kan identifiseres ved at de dukker opp etter følgende test:

```
if strcmp(myCon, 'NXT_kamera')
```

Denne testen undersøker simpelthen om grensesnittets forbindelse faktisk er til kameraroboten.

4.4. Resultater oppsummert

Inkluderingen av kamera-roboten i IR-robotens grensesnitt fungerer, men det tar lengre tid å få et globalt kart på plass sett i forhold til IR-roboten, da metodene for skanning av omgivelsene med kameraet kun returnerer hjørner, ikke målepunkter langs veggene.

Det er alt implementert en usikkerhetsparameter til bruk i kartgenereringen, så dette er ikke gjort på nytt, selv om det står spesifisert i oppgaveteksten.

5. Konklusjon

Utgangspunktet for denne oppgaven var to roboter, hvor den ene er basert på skanning av omgivelsene basert på IR, mens den andre bruker et kamera og bildebehandlingsalgoritmer for å oppfatte sine omgivelser. Dette er to svært ulike prinsipper, med ulike fordeler og ulemper knyttet til hver av dem.

Målet med denne oppgaven har vært todelt. Den første delen har dreid seg om å forbedre IR-roboten på en rekke områder, for å øke presisjonen i kartleggingen av omgivelsene, og bedre dens autonome egenskaper. Det har blitt implementert mekaniske og programmeringstekniske løsninger på en rekke problemer, og målet om økt presisjon har blitt nådd ved å regulere hastigheten på drivhjulene og ombygge roboten. Roboten fremstår som et klart bedre kartleggingsredskap nå enn ved prosjektets begynnelse. For å forbedre de autonome egenskapene har den fått implementert en batterisimulator. Dette gjør at den vil returnere til ladestasjonen ved behov, slik at operatøren ikke trenger å involveres.

Andre del av oppgaven har hatt som mål å benytte målinger fra kamera-roboten i kartgenereringssystemet, som i utgangspunktet er laget for IR-roboten. Dette målet er bare delvis nådd. Brukergrensesnittet har ikke blitt utvidet til å kunne håndtere flere roboter samtidig, og på den måten bruke flere forskjellige måleprinsipper parallelt, som var intensjonen. Det er for øvrig lagt til rette for at dette kan implementeres i fremtiden, ved at kameraroboten nå fungerer i det samme grensesnittet som IR-roboten, og dens målinger kan benyttes av kartgenereringssystemet. Selve kartgenereringen var i utgangspunktet så generell og godt konstruert at delpunktene i oppgaveteksten om usikkerhet i målingene og kombinasjon av måledata fra de to robotene ikke var nødvendig å implementere, da det alt fungerte tilfredsstillende. I stedet ble databehandlingsmetodene endret for å tilpasse målingene fra kameraroboten til systemet. Dette fungerte etter intensjonen.

Videre er det lagt ned mye arbeid i å identifisere områder med forbedringspotensiale, særlig i IR-roboten. Det har vist seg at en konstruksjon basert på lego har visse begrensninger, så en fullstendig ombygging om omstrukturering av denne roboten vil være et mulig tema for nye prosjekter.

6. Videre arbeid

Kapittelet er tredelt, der arbeidspunktene er kategorisert etter hvilken robot som er i fokus, eller om det gjelder systemet som helhet.

6.1. IR-robot

- Fullstendig ombygging av roboten. Gjennom arbeidet med roboten har det dukket opp mange små problemer som kunne vært unngått dersom roboten var konstruert av andre materialer enn lego. Fordelen med lego er at konstruksjonen raskt kan endres, men ulempene ligger i manglende rigiditet og presisjon. I tillegg opptar lego stort volum hvis det skal kunne bære særlig vekt. Det er ikke sikkert dette er en nødvendig endring, men muligheten bør kartlegges.

- Motorene er også av lego-typen, og tydelig konstruert for lavere laster enn de nå blir utsatt for. En omstrukturering av roboten bør også innebære en ombygging av drivlinjen, inkludert motorstyringskort og motorer.

- Batterisimulatoren kan byttes ut mot sanntidsovervåkning av batteriet. Dette fordrer nytt hovedkort, og sannsynligvis ny ladekrets. Fordelen er at mikrokontrolleren da ikke trenger å anta noe i forhold til batteriets status, særlig hvis ladingen også er styrt av mikrokontrolleren.

- Dersom kretskortene endres, kan det være hensiktsmessig å implementere en spenningsovervåkning over kontaktene mot ladestasjonen. Da kan mikrokontrolleren forsikre seg om at batteriet faktisk lades, som kan være nyttig for eksempel hvis laderens støpsel er dratt ut.

- Dersom batterisimulatoren beholdes, kan den forbedres ved å la maksimalkapasiteten være en dynamisk funksjon. Dette er diskutert i kapittel 3.1.5.

- For å utelukke hjulspinn som feilkilde i odometrimålinger kan det konstrueres egne slepehjul, eller sensorer av annet slag, frikoblet fra drivakslingene.

- Flere punkter finnes oppsummert i (Tusvik, 2009), kapittel 10.

6.2. Kamera-robot

- Forbedret design av kamera-roboten, med bedre oppsett av kameraet, slik at konstruksjonen blir mer stabil er ønskelig. Kameraet er relativt stort, og et mer rigid sensortårn er nok hensiktsmessig.

- Gjøre roboten trådløs ved å:

    - Legge inn ladestasjonsmulighet også i kameraroboten. Enten må det bygges en egen ladestasjon, eller så må roboten tilpasses den eksisterende ladestasjonen. Dette medfører en ombygging av roboten med ladepunkter, men burde være nokså enkelt å implementere.

    - Implementere batteridrift av webkamera. Kameraet drives av 5 V DC og trekker opptil 1 A. Batteri bør på plass for å gjøre kamera-roboten helt trådløs. Det bør også konstrueres en ladekrets for dette batteriet, helst med inngangsspenning på 12 V AC, da dette er ladespenningen NXT-enheten benytter. Dette punktet er naturlig å ta med dersom roboten får ladestasjonsmuligheten nevnt ovenfor.

    - Bytte ut gamle deler. NXT-enheten bør byttes, da den bærer preg av å være utslitt (skjermen virker bare unntaksvis, og bluetooth-forbindelsen er mer eller mindre ikke-eksisterende, slik at USB-kabel må benyttes). Samtidig kan det være nyttig å teste alle tilgjengelige sensorer, og skaffe nye der det trengs. Det hadde også vært hensiktsmessig med ekstra sensorer for redundans.

- Roboten benytter differensialdrift, altså en motor på hvert av de to drivhjulene. Dette medfører nødvendigvis ulik hastighet på grunn av ulikheter i motorer og hjuloppheng. En feilkorreksjon av samme type som beskrevet for den andre roboten i avsnitt 3.1.3.2 kan være en mulig løsning.

6.3. Systemet som helhet

- Utvide systemet til å håndtere flere roboter samtidig. En begynnelse kan være å ta utgangspunkt i denne oppgavebesvarelsen, og få det overordnede systemet til å virke med IR-roboten og kamera-roboten parallelt. Det er flere mulige deloppgaver her, blant annet:
    - Implementere en fornuftig navigasjonsalgoritme for n roboter. Algoritmen som benyttes per i dag er ikke hensiktsmessig med mer enn én robot samtidig. En mulig innfallsvinkel er å ta utgangspunkt i det globale kartet, og se hvor det er foretatt få eller ingen målinger, for så å sende en robot dit. Kalmanfilteret vil automatisk øke presisjonen i kartet dersom det kommer flere målinger av samme område.
    - Sikre at robotene ikke oppfatter hverandre som deler av omgivelsene
    - Endre brukergrensesnittet, slik at det er mulig å koble til n roboter samtidig, og overvåke dem parallelt. En enkel løsning vil være å åpne flere instanser av GUI'et samtidig, og la dem kjøre paralellt, men da blir det fort fullt på skjermen, ettersom en enkelt brukerflate er nokså stor.

7. Referanser

Arras, K. O. (2004a). The CAS Robot Navigation Toolbox Lastet ned 12. mai, 2011, fra http://www.cas.kth.se/toolbox/

Arras, K. O. (2004b) The CAS Robot Navigation Toolbox - User Guide and Reference. Lastet ned fra http://www.cas.kth.se/toolbox/UsersGuide-0.9.pdf

Atmel. (2011a). ATmega32 Datasheet rev. 2503Q-AVR-02/11.

Atmel. (2011b). AVR Studio 5 Lastet ned 1. april, 2011, fra http://www.atmel.com/microsite/avr_studio_5/default.asp

Cadex Electronics. (2011). Battery University - How to Store Batteries Lastet ned 9. mai, 2011, fra http://batteryuniversity.com/learn/article/how_to_store_batteries

Durrant-Whyte, H., & Bailey, T. (2006). Simultaneous localization and mapping: part I. *Robotics & Automation Magazine, IEEE, 13*(2), 99-110.

Free2Move. (2011). Wireless UART Configuration Software Lastet ned 1. april, 2011, fra http://www.free2move.se/?page_id=905

Kallevik, P. S. (2007). *Forbedring av hardware til Legorobot.* Prosjektoppgave, Norges teknisk-naturvitenskapelige universitet, Trondheim.

Kristiansen, M. A. (2009). *Fjernstyring av LEGO-robot - Ny funksjonalitet for returnering til ladestasjon.* Prosjektoppgave, Norges teknisk-naturvitenskapelige universitet, Trondheim.

LEGO. (2011). LEGO Mindstorms USB Driver Lastet ned 20. juni, 2011, fra http://mindstorms.lego.com/en-us/support/files/Driver.aspx

Magnussen, T. (2008). *Fjernstyring av Legorobot.* Masteroppgave, Norges teknisk-naturvitenskapelige universitet, Trondheim.

Mathworks. (2011). SimPowerSystems - Battery Lastet ned 9. mai, 2011, fra http://www.mathworks.com/help/toolbox/physmod/powersys/ref/battery.html

Næss, E. (2008). *Fjernstyring av legorobot.* Prosjektoppgave, Norges teknisk-naturvitenskapelige universitet, Trondheim.

RWTH Aachen University. (2011). Mindstorms NXT Toolbox for MATLAB Lastet ned 20. juni, 2011, fra http://www.mindstorms.rwth-aachen.de/

Tusvik, J. S. (2009). *Fjernstyring av legorobot.* Prosjektoppgave, Norges teknisk-naturvitenskapelige universitet, Trondheim.

Tøraasen, J. K. (2009). *Kartlegging ved hjelp av robot og kamerasensor.* Prosjektoppgave, Norges teknisk-naturvitenskapelige universitet, Trondheim.

8. Vedlegg

Vedleggsliste

A. Ny funksjonalitet i IR-robotens C-kode

Alle filer med endret eller ny funksjonalitet er vedlagt i sin helhet for kontekstens og lesbarhetens del, uavhengig av endringenes omfang.

battery.h

Batterisimulatorens header-fil

```
/*
 * battery.h
 *
 * Created: 10.05.2011 16:51:32
 *  Author: Sigurd Hannaas
 */


#ifndef __battery_h__
#define __battery_h__


#include <inttypes.h>
#include <math.h>
#include <stdio.h>

extern uint8_t ui8_needsToCharge;
extern uint16_t ui16_chargingTime;

void battery_init(uint16_t ui16_startCapacity);

void updateBatteryCapacity();


#endif /* __battery_h__ */
```

battery.c

Batterisimulatorens c-kode

```
/*
 * battery.c - simple battery capacity simulator/monitor
 *
 * Created: 10.05.2011 16:51:03
 *  Author: Sigurd Hannaas
 */

#include <avr/io.h>
#include <avr/interrupt.h>
#include <compat/deprecated.h>
#include <inttypes.h>
#include <math.h>
#include <stdio.h>

#include "main.h"
#include "timer.h"
#include "battery.h"
#include "uart.h"

//capacity variables:
float f_batteryCapacity;
float f_alarmCapacity;

//Calibration coefficients
float f_batteryMaxCal;
float f_batteryLoadGain;

//current drawn from battery [mA]
float f_currentFlow;

//Simulation step period [h]
float f_batterySimPeriod;

//Flag to set robot's need to charge
uint8_t ui8_needsToCharge;

//Preferred charging time [seconds]
uint16_t ui16_chargingTime;

//counter
uint8_t ui8_alarmCounter = 0;
uint8_t ui8_debugCounter = 0;
uint8_t ui8_debugCounterTwo = 0;



void battery_init(uint16_t ui16_startCapacity)
{
	f_alarmCapacity = 10.0;				// minimum remaining capacity before
                                        charging [mAh]
	f_batteryMaxCal = 0.606;			//calibration coefficient for battery
                                        life
	f_batteryLoadGain = (float)167/126;	//calibration coefficient for
                                        consumption
```

```
	f_batteryCapacity = f_batteryMaxCal * ui16_startCapacity; // mAh

	f_currentFlow = 60.0;                  // mA (idle)

	f_batterySimPeriod = 1/(3600*(float)ui8_batterySimFreq);  //(in hours to
                                                              preserve units)

	ui8_needsToCharge = 0;

	ui16_chargingTime = 5*60*60;           //charging time in seconds (5 hours)

}

void updateBatteryCapacity()
{
	switch (ui8_status)
	{
		case 1:      //idle
			f_currentFlow = 60.0;     //mA
			break;
		case 5:      //scanning
			f_currentFlow = 180.0;    //idle=60, ir=4*30 mA
			break;
		case 2:      //turning
		case 3:      //turning
		case 4:      //forward
		case 7: //backward
			f_currentFlow = 100.0;    //idle=60, motors=2*20 mA
			break;
		case 6:      //low-level command. Means driving.
			f_currentFlow = 100.0;    //idle=60, motors=2*20 mA
			break;
		case 8:      //calibration of IR-sensors (one at a time)
			f_currentFlow = 90.0;     //idle=60, ir=1*30 mA
			break;
		default:     //should never occur
			f_currentFlow = 220.0;    //assumes max usage, all bets are off
                                      if this is ever invoked.
			break;
	}

	f_batteryCapacity -= f_batteryLoadGain*f_currentFlow*f_batterySimPeriod;
                                                                      //mA*h
	if (f_batteryCapacity < f_alarmCapacity)
	{
		//ALARM!!! recharge!
		ui8_needsToCharge = 1;
	}
}
```

encoder.h

Encoderens header-fil. Konstantene er endret for å reflektere ny konfigurasjon. Ingen endringer i *encoder.c*, derfor er ikke den vedlagt.

```
/**
 *  encoder.h - Interface to angle sensor.
 *
 * @author Bjrøn Syvertsen
 * @author Changed by Johannes Schrimpf
 * @author Changed by Jannicke Selnes Tusvik
 * @author Changed by Sigurd Hannaas
 * @file encoder.h
 * @brief Interface to angle sensor.
 */
#ifndef __encoder_h__
#define __encoder_h__

#include <inttypes.h>

//#define WHEELRADIUS_MM 22.34 versjon 3
//#define WHEELRADIUS_MM 21.0*1.004629 versjon 2
//#define NUM_TICS_ROT 200 /* Number of tics per rotation of wheel.*/
//#define RADS_PER_TIC 0.0314159265

//#define WHEELBASE_MM 189.4938  // found by testing. Not measured
#define WHEELBASE_MM 210        // Measured
#define TO_PI 2*M_PI
//#define X_FACTOR 0.675        //Circumference(135mm)/Tics pr rotation(200)
                                SHOULD WORK. How far the robot travles per tick
#define X_FACTOR 0.78           //Found by calculation. Circumference(156 mm)/Tics
                                per rotation (200)
#define X_FACTOR_ROTATE 0.7587; //Found by testing
#define X_FACTOR_DRIVE 0.8315;  //Found by testing

extern int16_t i16_rightWheelTicks ;
extern int16_t i16_leftWheelTicks ;
/* Distance measured by angle sensor given as number of tics registered.*/

int encoder_init();

#endif  /* __encoder_h__ */
```

lego.h

Header-fil for lego.c, utvidet med konstanter for bremsing.

```
/**
 *  lego.h - LEGO driver interface.
 *
 * @author Håkon Skjelten <skjelten@pvv.org>
 * @author Changed by Bjørn Syvertsen
 * @author Changed by Johannes Schrimpf
 * @author Changed by Sigurd Hannaas
 * @file lego.h
 * @brief LEGO driver interface.
 */


#ifndef __lego_h__
#define __lego_h__


#include <inttypes.h>



#define LEGO_LEFT                    0
#define LEGO_RIGHT                   1
#define LEGO_FORWARD                 2
#define LEGO_BACKWARD                3
#define LEGO_STOP                    4
#define LEGO_PWM_BREAK_RIGHT         5
#define LEGO_PWM_BREAK_LEFT          6

#define WHEEL_DIRECTION_STOP         0
#define WHEEL_DIRECTION_FORWARD      1
#define WHEEL_DIRECTION_BACKWARD     2

#define SET_PC0 PORTC |= _BV(PC0)
#define SET_PC1 PORTC |= _BV(PC1)
#define SET_PC2 PORTC |= _BV(PC2)
#define SET_PC3 PORTC |= _BV(PC3)
#define SET_PC4 PORTC |= _BV(PC4)
#define SET_PC5 PORTC |= _BV(PC5)
#define SET_PC6 PORTC |= _BV(PC6)
#define SET_PC7 PORTC |= _BV(PC7)

#define CLR_PC0 PORTC &= ~_BV(PC0)
#define CLR_PC1 PORTC &= ~_BV(PC1)
#define CLR_PC2 PORTC &= ~_BV(PC2)
#define CLR_PC3 PORTC &= ~_BV(PC3)
#define CLR_PC4 PORTC &= ~_BV(PC4)
#define CLR_PC5 PORTC &= ~_BV(PC5)
#define CLR_PC6 PORTC &= ~_BV(PC6)
#define CLR_PC7 PORTC &= ~_BV(PC7)

extern int8_t i8_rightWheelDirection;
extern int8_t i8_leftWheelDirection;
```

```
/**
 * Initialize LEGO driver.
 */
void lego_init(void);

/**
 * Set driving direction command to LEGO robot.
 */
void lego_drive(int direction);


#endif /* __lego_h__ */
```

lego.c

Utvidet med funksjonalitet for å bremse hjulene individuelt, uten å påvirke oppfattet kjøreretning.

```
/**
 *  lego.c - LEGO driver functions.
 *
 * @author Håkon Skjelten <skjelten@pvv.org>
 * @author Changed by Johannes Schrimpf
 * @author Changed by Jannicke Selnes Tusvik
 * @author Changed by Sigurd Hannaas
 * @file lego.c
 * @brief LEGO driver functions.
 */

#include <avr/pgmspace.h>

#include "lego.h"
#include "main.h"
#include "messages.h"


int8_t i8_rightWheelDirection = WHEEL_DIRECTION_STOP;
int8_t i8_leftWheelDirection  = WHEEL_DIRECTION_STOP;


/**
 * Initialize LEGO driver.
 */

void lego_init(void)
{
      /* Set direction to output */
      DDRC |= _BV(PC7) | _BV(PC6) | _BV(PC5)| _BV(PC4)| _BV(PC3)| _BV(PC2) |
_BV(PC1) | _BV(PC0);

      /* Default to 0V */
      PORTC &= ~_BV(PA7) & ~_BV(PA6) & ~_BV(PA5) & ~_BV(PA4)& ~_BV(PA3) &
~_BV(PA2) & ~_BV(PA1) & ~_BV(PA0) ;
}


/**
 * Set driving direction command to LEGO robot.
 */
void lego_drive(int direction)
{
      switch(direction)
      {
      case LEGO_RIGHT:

                  i8_rightWheelDirection = WHEEL_DIRECTION_BACKWARD;
                  i8_leftWheelDirection  = WHEEL_DIRECTION_FORWARD;


                  SET_PC0;
                  CLR_PC1;
                  SET_PC6;
```

```
                CLR_PC7;


            break;

        case LEGO_LEFT:

                i8_rightWheelDirection = WHEEL_DIRECTION_FORWARD;
                i8_leftWheelDirection  = WHEEL_DIRECTION_BACKWARD;

            CLR_PC0;
                SET_PC1;
                CLR_PC6;
                SET_PC7;

            break;

        case LEGO_FORWARD:

                i8_rightWheelDirection = WHEEL_DIRECTION_FORWARD;
                i8_leftWheelDirection  = WHEEL_DIRECTION_FORWARD;

                SET_PC0;
                CLR_PC1;
                CLR_PC6;
                SET_PC7;
                break;

        case LEGO_BACKWARD:

                i8_rightWheelDirection = WHEEL_DIRECTION_BACKWARD;
                i8_leftWheelDirection  = WHEEL_DIRECTION_BACKWARD;

                CLR_PC0;
                SET_PC1;
                SET_PC6;
                CLR_PC7;

                break;

            case LEGO_STOP:

                i8_rightWheelDirection = WHEEL_DIRECTION_STOP;
                i8_leftWheelDirection  = WHEEL_DIRECTION_STOP;

                SET_PC0;
                SET_PC1;
                SET_PC6;
                SET_PC7;

                break;

            case LEGO_PWM_BREAK_RIGHT:

//                NO CHANGE TO DIRECTION VARIABLES, AS GENERAL DIRECTION IS
                PRESERVED.
//                i8_rightWheelDirection = WHEEL_DIRECTION_STOP;
//                i8_leftWheelDirection  = WHEEL_DIRECTION_FORWARD;
//
```

```
                    //NO CHANGE TO LEFT WHEEL
//                  SET_PC0;
//                  CLR_PC1;
                    SET_PC6;
                    SET_PC7;

                    break;

             case LEGO_PWM_BREAK_LEFT:

//                  NO CHANGE TO DIRECTION VARIABLES, AS GENERAL DIRECTION IS
                    PRESERVED.
//                  i8_rightWheelDirection = WHEEL_DIRECTION_STOP;
//                  i8_leftWheelDirection  = WHEEL_DIRECTION_FORWARD;
//
                    SET_PC0;
                    SET_PC1;

                    //NO CHANGE TO RIGHT WHEEL
//                  SET_PC6;
//                  SET_PC7;

                    break;

             default:
                    break;
    }
}
```

main.c

C-kodens hovedløkke. Mange små endringer som følge av ny/endret funksjonalitet implementert andre steder.

```
/**
 *  main.c - main() implementation.
 *
 * @author Håkon Skjelten <skjelten@pvv.org>
 * @author Changed by Sveinung Helgeland for version 2.0 of RoboRadar
 * @author Radicaly changed by Bjørn Syvertsen for version 3.0 of RoboRadar
 * @author Changed by Johannes Schrimpf
 * @author Changed by Jannicke Selnes Tusvik
 * @author Changed by Sigurd Hannaas 2011
 * @file main.c
 * @brief main() implementation.
 */

#include <avr/interrupt.h>
#include <avr/pgmspace.h>
#include <avr/io.h>
#include <util/delay.h>
#include <math.h>
#include <stdio.h>

#include "gp2d12.h"
#include "main.h"
#include "lego.h"
#include "servo.h"
#include "timer.h"
#include "uart.h"
#include "encoder.h"
#include "position.h"
#include "navigation.h"
#include "messages.h"
#include "battery.h"


/**
 * State Variable
 * 0: init
 * 1: idle
 * 2: robot is rotating
 * 3: robot is rotating and will drive afterwards
 * 4: driving to position
 * 5: IR-Scan
 * 6: low level command
 * 7: backing up to charging station
 * 8: performing a calibration of the IR sensors
 */
uint8_t ui8_status = 0; // state variable
uint8_t ui8_n      = 0; // counter for loops
uint8_t ui8_h      = 0; // human modus aktivated=1

/**
 * Initializes the system
 */
void init()
{
```

```
	// Initialize UART (RS-232) driver. Using PORT D (0, 1)
	uart_init(57600);
	// Initialize internal AD converter
	gp2d12_init();
	// Initialize servo (PWM). Using PORT B (0)
	servo_init(0);
	// Initialize timer
	timer_init();
	// Initialize LEGO system
	lego_init();
	// Initialize encoders
	encoder_init();
	//Initialize battery simulator (battery capacity in mAh as input)
	battery_init(5200);
	// Enable global interrupt flag
	sei();
	// Reset Position
	resetRobotPos();
}


/**
 * Main loop
 */
int main(void)
{
	init();

	ui8_status=1;

	if(ui8_h==1)
	{
		send_msg(1);
	}
	//Main-loop
	while(1)
	{
		updateRobotPos();
		switch(ui8_status)
		{

		case 5:
			if(ui8_h==1)
			{
				send_msg(2);
			}
			else
			{
				uart_send(V2_MSG_IR_SCAN_HEADER);
			}

			if (i8_servoAngle !=0)
			{
				servo_set_angle(0);
				wait_ms(1000);
			}

			gp2d_12_on();
			for (ui8_n=0;ui8_n<18;ui8_n++)
```

```
                {
                        sendIR(0);
                        sendIR(1);
                        sendIR(2);
                        sendIR(3);
                        servo_set_angle_p5();
                        wait_ms(200);
                }
                gp2d_12_off();

                servo_set_angle(0);
                wait_ms(1000);
                ui8_status=1;
                if(ui8_h==1)
                {
                        send_msg(3);
                }
                else
                {
                        uart_send(V2_MSG_IR_SCAN_HEADER_END);
                }
                break;
        case 1:
                robotControl(LEGO_STOP);
                break;
        case 2:
        case 3:
                robotRotate();
                break;
        case 4:
                //uart_send(V2_DEBUG_MARKER1); //DEBUGGING
                robotDrive();
                break;
        case 6:
                if(i8_rightWheelDirection==WHEEL_DIRECTION_FORWARD &&
i8_leftWheelDirection==WHEEL_DIRECTION_FORWARD)
                {
                        robotControl(LEGO_FORWARD);
                        break;
                }
                else if (i8_rightWheelDirection==WHEEL_DIRECTION_FORWARD &&
i8_leftWheelDirection==WHEEL_DIRECTION_BACKWARD)
                {
                        robotControl(LEGO_LEFT);
                        break;
                }
                else if (i8_rightWheelDirection==WHEEL_DIRECTION_BACKWARD &&
i8_leftWheelDirection==WHEEL_DIRECTION_FORWARD)
                {
                        robotControl(LEGO_RIGHT);
                        break;
                }
                else if (i8_rightWheelDirection==WHEEL_DIRECTION_BACKWARD &&
i8_leftWheelDirection==WHEEL_DIRECTION_BACKWARD)
                {
                        robotControl(LEGO_BACKWARD);
                        break;
                }
                else
```

```
                        {
                                robotControl(LEGO_STOP);
                                ui8_status=1;
                                if(ui8_h==1)
                                {
                                        send_msg(8);
                                }
                        }
                        break;
                case 7:
                        robotControl(LEGO_BACKWARD);    //backing up to charging
                                                        station
                        break;
                case 8:
                        gp2d_12_on();
                        sendAnalogValue();
                        gp2d_12_off();
                        ui8_status=1;
                        break;
                default:
                        break;
                }//Select
        } //while

        return 0;

} //main


/**
 * Waits ui16_ms milliseconds.
 */
void wait_ms(uint16_t ui16_ms)
{
        uint16_t ui16_m=0;

        for (ui16_m = 0 ; ui16_m< ui16_ms ; ui16_m++)
        {
                _delay_ms(1);
        }
}
```

navigation.h

Header-fil for navigation.c. Ny funksjon for korreksjon av bevegelser.

```
/**
 *  navigation.h - navigation routines
 *
 * @author Håkon Skjelten <skjelten@pvv.org>
 * @author Changed by Sveinung Helgeland for version 2.0 of RoboRadar
 * @author Radicaly changed by Bjørn Syvertsen for version 3.0 of RoboRadar
 * @author Changed and moved to own file by Johannes Schrimpf
 * @author Changed by Sigurd Hannaas
 * @file navigation.h
 * @brief navigation routines
 */


#ifndef __navigation_h__
#define __navigation_h__


#include <inttypes.h>

//extern int8_t i8_targetHeadingSat;
//extern int8_t i8_targetDistanceSat;
extern uint8_t ui8_targetHeading;
extern int8_t i8_targetDistance;


int8_t i8_status_distance;
float f_startPointX;
float f_startPointY;

void robotRotate();
void robotDrive();
void robotControl(int direction);


#endif /* __navigation_h__ */
```

navigation.c

Styrer robotens bevegelser. Ny funksjon for korrigering av bevegelsene, i tillegg til enkelte endringer for å dra nytte av denne.

```
/**
 *  navigation.c - navigation routines
 *
 * @author Håkon Skjelten <skjelten@pvv.org>
 * @author Changed by Sveinung Helgeland for version 2.0 of RoboRadar
 * @author Radicaly changed by Bjørn Syvertsen for version 3.0 of RoboRadar
 * @author Changed and moved to own file by Johannes Schrimpf
 * @author Changed by Sigurd Hannaas
 * @file navigation.c
 * @brief navigation routines
 */

#include <avr/interrupt.h>
#include <avr/pgmspace.h>
#include <math.h>
#include <stdio.h>


#include "navigation.h"
#include "position.h"
#include "lego.h"
#include "main.h"
#include "uart.h"
#include "messages.h"
#include "encoder.h"

// extern variables
uint8_t ui8_targetHeading     = 0;
int8_t i8_targetDistance      = 0;

// intern variables
int8_t i8_status_distance = 1;
float f_startPointX=0;
float f_startPointY=0;
int16_t i16_pwm_maxError = 1;          //maximum acceptable number of ticks
                                       different between right and left wheel
uint8_t ui8_messageLimiter = 0;        //limiter for debug messages
uint8_t ui8_lastDirection = 255;       //Status change control
uint8_t ui8_lastStatus = 0;            //Status change control
int16_t i16_startRightWheelTicks = 0;  //temporary ticks counter (signed int)
int16_t i16_startLeftWheelTicks = 0;   //temporary ticks counter (signed int)
int16_t i16_tempRightWheelTicks = 0;   //temporary ticks counter (signed int)
int16_t i16_tempLeftWheelTicks = 0;    //temporary ticks counter (signed int)


/**
 * rotates the robot to the desired heading
 */
void robotRotate()
{
      float f_deadZone     = (float) ((2.0*M_PI)/180.0); // 2 degree's
      float f_target       = (float) ui8_targetHeading;
      float f_targetInRads = M_PI*2.0*f_target/180.0;
```

```
	if( f_targetInRads-f_robotTheta > M_PI )
	{
		robotControl(LEGO_RIGHT);
	}
	else if( f_targetInRads-f_robotTheta < -M_PI )
	{
		robotControl(LEGO_LEFT);
	}
	else if( f_targetInRads+f_deadZone < f_robotTheta )
	{
		robotControl(LEGO_RIGHT);
	}
	else if( f_targetInRads-f_deadZone > f_robotTheta )
	{
		robotControl(LEGO_LEFT);
	}
	else
	{
		if (ui8_status==3)
		{
			ui8_status=4;
		}
		else
		{
			ui8_status=1;
		}
		lego_drive(LEGO_STOP);
	}
}


/**
 * moves the robot a desired distance
 */
void robotDrive()
{

	if(i8_status_distance==1 )
	{
		if(i8_targetDistance !=0 )
		{
			f_startPointX=f_robotX;
			f_startPointY=f_robotY;
			i8_status_distance=2;
			robotControl(LEGO_FORWARD);
		}
		else
		{
			ui8_status=1;
		}
	}
	else if(i8_status_distance==2)
	{
		float diffXSquareInCm = ((f_startPointX-f_robotX)*(f_startPointX-
f_robotX))/100;
		float diffYSquareInCm = ((f_startPointY-f_robotY)*(f_startPointY-
f_robotY))/100;
```

```
            if( (i8_targetDistance*i8_targetDistance) <=
(diffXSquareInCm+diffYSquareInCm) )
            {
                  ui8_status=1;
                  i8_status_distance=1;
                  i8_targetDistance=0;
                  lego_drive(LEGO_STOP);
                  if(ui8_h==1)
                  {
                        send_msg(4);
                  }
                  else
                  {
                        uart_send(V2_MSG_ARRIVED);

                  }
            }
            else
            {
                  robotControl(LEGO_FORWARD);
            }
      }
}
/**
 * Controls the robot's movements by comparing ticks
 */
void robotControl(int direction)
{
      if (direction != ui8_lastDirection || ui8_status != ui8_lastStatus)
      //resetting controller tick counters
      {
            i16_startLeftWheelTicks = i16_leftWheelTicks;
            i16_startRightWheelTicks = i16_rightWheelTicks;
            ui8_lastDirection = direction;
            ui8_lastStatus = ui8_status;
      }

      i16_tempLeftWheelTicks = i16_leftWheelTicks - i16_startLeftWheelTicks;
      i16_tempRightWheelTicks = i16_rightWheelTicks - i16_startRightWheelTicks;

      lego_drive(direction);          //ensuring proper movement, as the brakes
                                      do not affect the other wheel.

      switch (direction)
      {
            case LEGO_FORWARD:

                  if (i16_tempRightWheelTicks > i16_tempLeftWheelTicks +
i16_pwm_maxError)
                  {
                        //uart_send(V2_DEBUG_MARKER2);
                        lego_drive(LEGO_PWM_BREAK_RIGHT);
                  }
                  else if (i16_leftWheelTicks > i16_rightWheelTicks +
i16_pwm_maxError)
                  {
                        //uart_send(V2_DEBUG_MARKER3);
                        lego_drive(LEGO_PWM_BREAK_LEFT);
                  }
```

```
                break;


        case LEGO_BACKWARD:

                if (i16_tempRightWheelTicks < i16_tempLeftWheelTicks -
i16_pwm_maxError)
                {
                        lego_drive(LEGO_PWM_BREAK_RIGHT);
                }
                else if (i16_tempLeftWheelTicks < i16_tempRightWheelTicks -
i16_pwm_maxError)
                {
                        lego_drive(LEGO_PWM_BREAK_LEFT);
                }

                break;


        case LEGO_RIGHT:

                if (-i16_tempRightWheelTicks > i16_tempLeftWheelTicks +
i16_pwm_maxError)
                {
                        lego_drive(LEGO_PWM_BREAK_RIGHT);
                }
                else if (i16_leftWheelTicks > -i16_rightWheelTicks +
i16_pwm_maxError)
                {
                        lego_drive(LEGO_PWM_BREAK_LEFT);
                }

                break;


        case LEGO_LEFT:

                if (i16_tempRightWheelTicks > -i16_tempLeftWheelTicks +
i16_pwm_maxError)
                {
                        lego_drive(LEGO_PWM_BREAK_RIGHT);
                }
                else if (-i16_leftWheelTicks > i16_rightWheelTicks +
i16_pwm_maxError)
                {
                        lego_drive(LEGO_PWM_BREAK_LEFT);
                }

                break;


        default:

                break;

    } //switch case
}
```

timer.c

Kode som har med mikrokontrollerens timere å gjøre. Ny funksjonalitet i forbindelse med kontroll av bevegelser og batterisimulering. Det er ingen endringer i den tilhørende *timer.h*-filer, derfor er den utelatt.

```
/**
 *  timer.c - timer functions.
 *
 * @author Håkon Skjelten <skjelten@pvv.org>
 * @author Changed by Johannes Schrimpf
 * @author Changed by Jannicke Selnes Tusvik
 * @author Changed by Sigurd Hannaas
 * @file timer.c
 * @brief Timer functions.
 */

#include <avr/interrupt.h>
#include <avr/io.h>

#include "timer.h"
#include "servo.h"
#include "main.h"
#include "position.h"
#include "uart.h"
#include "lego.h"
#include "gp2d12.h"
#include "messages.h"
#include "navigation.h"
#include "battery.h"

uint16_t ui16_ms_count = 0;            //not really millisecs. incremented approx.
                                       61 times per sec.
uint16_t ui16_simLimiter = 0;          //counter to limit
uint16_t ui16_TEMPms_count = 0;
uint16_t ui16_sek = 0;
uint16_t ui16_min = 0;

uint8_t ui8_batterySimFreq = 20;       //battery simulation update frequency [Hz]

/**
 * Initialize timer.
 * Sets timer interrupts to approx. 17314,45 HZ
 * CORRECTION: 16 MHz crystal / 1024 = 15625 Hz --> Overflow frequency (8 bit
timer): 15625 / 256 = 61,03515625 Hz.
 */
void timer_init(void)
{
      TIFR  &= ~(1 << TOV1);    //??? Bit used by 16 bit TIMER1, not TIMER0.
      TIFR  |= _BV(TOIE0);      //??? TOIE0 is a bit in TIMSK register, not
                                TIFR...

      // prescaler = 1024
      TCCR0 |= _BV(CS02) | _BV(CS00);

       // start timer
      TIMSK |= _BV(TOIE0);
      TCNT0 = 0;
```

```
}

/**
 * Delays for approx. ms number and milliseconds.
 */
void ms_sleep(uint16_t ui16_ms)
{
	// Restart HW counter
	TCNT0 = 0;

	// Restart software limiter
	ui16_ms_count = 0;
	while (ui16_ms_count < ui16_ms);
}

/**
 * Increment ms_count
 * Collision system. When backing up, collision system is off.
 */
SIGNAL(SIG_OVERFLOW0)
{
	ui16_ms_count++;	//(is later reset if > 60)

	ui16_simLimiter++;

	if(ui16_simLimiter > ((61 / ui8_batterySimFreq) - 1)) // batterySimFreq = 20
	{
		ui16_simLimiter = 0;

		//update battery simulation
		updateBatteryCapacity();

		//Ensuring robot movements
		switch (ui8_status)
		{
			case 1:
				robotControl(LEGO_STOP);
				break;
			case 2:
			case 3:
				robotRotate();
				break;
			case 4:
				robotDrive();
				break;
			case 6:
				if(i8_rightWheelDirection==WHEEL_DIRECTION_FORWARD &&
i8_leftWheelDirection==WHEEL_DIRECTION_FORWARD)
				{
					robotControl(LEGO_FORWARD);
					break;
				}
				else if (i8_rightWheelDirection==WHEEL_DIRECTION_FORWARD
&& i8_leftWheelDirection==WHEEL_DIRECTION_BACKWARD)
				{
					robotControl(LEGO_LEFT);
					break;
				}
```

```
                        else if
(i8_rightWheelDirection==WHEEL_DIRECTION_BACKWARD &&
i8_leftWheelDirection==WHEEL_DIRECTION_FORWARD)
                        {
                              robotControl(LEGO_RIGHT);
                              break;
                        }
                        else if
(i8_rightWheelDirection==WHEEL_DIRECTION_BACKWARD &&
i8_leftWheelDirection==WHEEL_DIRECTION_BACKWARD)
                        {
                              robotControl(LEGO_BACKWARD);
                              break;
                        }
                        else
                        {
                              robotControl(LEGO_STOP);
                              ui8_status=1;
                              if(ui8_h==1)
                              {
                                    send_msg(8);
                              }
                        }
                        break;
                  case 7:
                        robotControl(LEGO_BACKWARD);
                        break;
                  default:
                        break;
            }
      }
      //end simLimit interrupt

      if (ui16_ms_count>60)
      {
            if ((ui8_status == 4 || ui8_status == 6) && (ui8_status != 7))
            {
                  //uart_send(V2_DEBUG_MARKER1);         //DEBUG: send signal
                  every 1 sec? TEST SHOWS 1.197 secs. (61 / 61,03515625 Hz =
                  0,999 theoretically) --> CPUFREQUENCY= 19140 KHz??
                  //collision

                  if ( i8_servoAngle > 2)
                  {
                        servo_set_angle(0);
                        wait_ms(500);
                  }
                  if(i8_rightWheelDirection==WHEEL_DIRECTION_FORWARD)
                  {
                        uint8_t ui8_ir=sendIR(0);
                        sendIR(1);
                        sendIR(3);
                        if (ui8_ir < 24) // Stops when the robots front is about
                                          12 cm from an obstacle
                        {
                              lego_drive(LEGO_STOP);
                              ui8_status=1;
                              if(ui8_h==1)
                              {
```

```
                                        send_msg(6);
                                }
                                else
                                {
                                        uart_send(V2_COLLISION_ERROR);
                                }
                        }
                        else if(ui8_ir < 27) // collision warning when the robot
                                                is 15 cm from an obstacle
                        {
                                if(ui8_h==1)
                                {
                                        send_msg(5);
                                }
                                else
                                {
                                uart_send(V2_COLLISION_WARNING);
                                }
                        }
                }
                else if(i8_rightWheelDirection==WHEEL_DIRECTION_BACKWARD)
                {
                        uint8_t ui8_ir=sendIR(2);
                        sendIR(1);
                        sendIR(3);
                        if (ui8_ir < 17.5) // Stops when the robots back is 10
                                                cm from an obstacle
                        {
                                lego_drive(LEGO_STOP);
                                ui8_status=1;
                                if(ui8_h==1)
                                {
                                        send_msg(6);
                                }
                                else
                                {
                                        uart_send(V2_COLLISION_ERROR);
                                }
                        }
                        else if(ui8_ir < 22.5) // Collision warning when the
                                                robots back is 15 cm from an obstacle
                        {
                                if(ui8_h==1)
                                {
                                        send_msg(5);
                                }
                                else
                                {
                                uart_send(V2_COLLISION_WARNING);
                                }
                        }
                }
            }// if status
            ui16_ms_count=0;
            ui16_sek++;
        }// if count
} // SIGNAL
```

uart.h

Header-fil for uart.c. Utvidet kommunikasjonsprotokoll i forbindelse med batterisjekk.

```
/**
 *  uart.h - UART interface.
 *
 * @author Håkon Skjelten <skjelten@pvv.org>
 * @author Changed by Sveinung Helgeland for version 2.0 of RoboRadar
 * @author Changed by Bjørn Syvertsen for version 3.0 of RoboRadar
 * @author Changed by Johannes Schrimpf
 * @author Changed by Jannicke Selnes Tusvik
 * @author Changed by Sigurd Hannaas
 * @file uart.h
 * @brief UART interface.
 */


#ifndef __uart_h__
#define __uart_h__




/**
 * Initialize the internal UART driver.
 */
int uart_init(uint16_t baudRate);

/**
 * Sends one byte data on the UART.
 */
inline void uart_send(int8_t i8_data);
inline void uart_sendh(uint8_t ui8_data);
inline void uart_send16(uint16_t ui16_data);
inline void uart_send16h(int16_t i16_data);


uint8_t uartVer;

inline int uartVer1(uint8_t ui8_data);
inline int uartVer2(uint8_t ui8_data);



/**
 * Commands to Robot
 */
#define V2_HUMAN_MODUS                0x7A   //char -> z              dec: 122
#define V2_MACHINE_MODUS              0x6D   //char -> m              dec: 109

#define V2_ROBOT_FORWARD              0x77   //char -> w              dec: 119
#define V2_ROBOT_STOP                 0x73   //char -> s              dec: 115
#define V2_ROBOT_LEFT                 0x61   //char -> a              dec: 97
#define V2_ROBOT_RIGHT                0x64   //char -> d              dec: 100
#define V2_ROBOT_BACKWARD             0x78   //char -> x              dec: 120

#define V2_ROBOT_SET_POS              0x71   //char -> q <-(xxyytt)   dec: 113
#define V2_ROBOT_GET_POS              0x70   //char -> p ->(xxyyyy)   dec: 112
```

```
#define V2_ROBOT_CLR_POS              0x63   //char -> c              dec: 99

#define V2_RADAR_SET_ANGLE            0x72   //char -> r <-(t)        dec: 114
#define V2_RADAR_GET_ANGLE            0x74   //char -> t              dec: 116


#define V2_ROBOT_SET_HEADING          0x68   //char -> h <-(t)        dec: 104
#define V2_ROBOT_SET_DELTA_DIST       0x6A   //char -> j <-(zz)       dec: 106
#define V2_ROBOT_GO_POS               0x67   //char -> g <-(xxyy)     dec: 103

#define V2_STATUS                     0x62   //char -> b              dec: 98
#define V2_PING                       0x6F   //char -> o              dec: 111
#define V2_PING_RESPONSE_MATLAB       0x6b   //char -> k              dec: 107


#define V2_GET_IR_SCAN                0x66   //char -> f              dec: 102
#define V2_GET_SENSOR_1               0x31   //char -> 1              dec: 49
#define V2_GET_SENSOR_2               0x32   //char -> 2              dec: 50
#define V2_GET_SENSOR_3               0x33   //char -> 3              dec: 51
#define V2_GET_SENSOR_4               0x34   //char -> 4              dec: 52
//#define V2_GET_SENSOR_5             0x35   //char -> 5
//#define V2_GET_SENSOR_6             0x36   //char -> 6
//#define V2_GET_SENSOR_7             0x37   //char -> 7
//#define V2_GET_SENSOR_8             0x38   //char -> 8

#define V2_CALIBRATE_SENSORS          0x69   //char -> i              dec: 105

#define V2_SET_VER                    0x76   //char -> v              dec: 118
#define V2_SET_C                      0x35   //char -> 5              dec: 53

#define V2_CHECK_BATTERY              0x3F   //char -> ?              dec: 63
#define V2_RECHARGE_COMPLETE          0x3E   //char -> >              dec: 62

#define V2_GET_LEFTWHEEL              0x01   //char -> -              dec: 1
#define V2_GET_RIGHTWHEEL             0x02   //char -> -              dec: 2


/**
 * Messages to Matlab  (decimal values are displayed in the Matlab console if
Matlab interprets the values as wrong/not expected)
 */
#define V2_PING_RESPONSE              0x6B   //char -> k              dec: 107
#define V2_PING_MATLAB                0x6F   //char -> o              dec: 111

#define V2_MSG_ARRIVED                0x71   //char -> q              dec: 113
#define V2_MSG_IR_SCAN_HEADER         0x66   //char -> f
      dec: 102
#define V2_MSG_IR_SCAN_HEADER_END     0x67   //char -> g              dec: 103
#define V2_MSG_IR_HEADER_1            0x31   //char -> 1              dec: 49
#define V2_MSG_IR_HEADER_2            0x32   //char -> 2              dec: 50
#define V2_MSG_IR_HEADER_3            0x33   //char -> 3              dec: 51
#define V2_MSG_IR_HEADER_4            0x34   //char -> 4              dec: 52
#define V2_MSG_IR_HEADER_NO_DATA_1    0x35   //char -> 5              dec: 53
#define V2_MSG_IR_HEADER_NO_DATA_2    0x36   //char -> 6              dec: 54
#define V2_MSG_IR_HEADER_NO_DATA_3    0x37   //char -> 7              dec: 55
#define V2_MSG_IR_HEADER_NO_DATA_4    0x38   //char -> 8              dec: 56
#define V2_MSG_IR_CALIBRATION_END     0x21   //char -> !              dec: 33
#define V2_MSG_POSITION_HEADER        0x70   //char -> p              dec: 112
#define V2_MSG_RADAR_HEADER           0x72   //char -> r              dec: 114
```

```
#define V2_MSG_STATUS_HEADER          0x73   //char -> s              dec: 115

#define V2_NACK                       0x6E   //char -> n              dec: 110
#define V2_ACK                        0x61   //char -> a              dec: 97
#define V2_MSG_BUSY                   0x62   //char -> b              dec: 98

#define V2_COLLISION_WARNING          0x77   //char -> w              dec: 119
#define V2_COLLISION_ERROR            0x65   //char -> e              dec: 101

#define V2_BATTERY_LOW                0x3C   //char -> <              dec: 60
#define V2_BATTERY_OK                 0x3D   //char -> =              dec: 61

#define V2_DEBUG_MARKER1              0x23   //char -> #              dec: 35
#define V2_DEBUG_MARKER2              0x24   //char -> $              dec: 36
#define V2_DEBUG_MARKER3              0x25   //char -> %              dec: 37

#endif /* __uart_h__ */
```

uart.c

Utvidet kommunikasjonsprotokoll i forbindelse med batterisjekk, samt noen endringer ifm. bevegelseskontroll.

```
/**
 *  uart.c - UART functions.
 *
 * @author Håkon Skjelten <skjelten@pvv.org>
 * @author Changed by Sveinung Helgeland for version 2.0 of RoboRadar
 * @author Changed by Bjørn Syvertsen for version 3.0 of Roboadar
 * @author Changed by Johannes Schrimpf
 * @author Changed by Jannicke Selnes Tusvik
 * @author Changed by Sigurd Hannaas
 * @file uart.c
 * @brief UART functions.
 */

#include <avr/io.h>
#include <avr/interrupt.h>
#include <stdio.h>
#include <math.h>

#include "lego.h"
#include "main.h"
#include "uart.h"
#include "servo.h"
#include "lego.h"
#include "gp2d12.h"
#include "encoder.h"
#include "position.h"
#include "navigation.h"
#include "messages.h"
#include "battery.h"

uint8_t ui8_robot_cmd     = 0;
uint8_t ui8_wait_for_data = 0;
uint8_t ui8_data1         = 0;
uint8_t ui8_data2         = 0;
uint8_t ui8_data3         = 0;
uint8_t ui8_data4         = 0;
uint8_t ui8_data5         = 0;
uint8_t ui8_data6         = 0;

uint8_t  ui8_uartVer   = 2;
uint8_t  ui8_temp  = 0;
uint16_t ui16_tmpx = 0;
uint16_t ui16_tmpy = 0;
uint16_t ui16_tmpt = 0;

/**
 * Initialize the internal UART driver.
 */
int uart_init(uint16_t baudRate)
{
      switch(baudRate)
    {
      case 9600:
            UBRRL = 207;
```

```
                break;

        case 19200:
                UBRRL = 103;
                break;

        case 38400:
                UBRRL = 51;
                break;

                case 57600:
                UBRRL = 34;
                break;

                default:
                UBRRL = 207;
        break;
    }

        UCSRA = _BV(U2X);
        UCSRB = _BV(RXCIE) | _BV(RXEN) | _BV(TXEN);

        DDRD &= ~(_BV(DDD0));
        DDRD |= _BV(DDD1);

        //fdevopen(uart_send, NULL, 0);  // Initialize UART to STDOUT

        return 0;
}


/**
 * Sends one byte data on the UART.
 */
inline void uart_send(int8_t i8_data)
{
        while(!(UCSRA & (1 <<  UDRE))) ;
        UDR = i8_data;
}


/**
 * Sends an uint16_t value on the UART.
 */
inline void uart_send16(uint16_t ui16_data)
{
        uart_send((uint8_t) (ui16_data >> 8));
        uart_send((uint8_t) (ui16_data & 0xff));
}


/**
 * Sends an uint8_t in a human readable format
 */
inline void uart_sendh(uint8_t ui8_data)
{
        uint8_t ui8_tmp=0;
        if (ui8_data>=100)
        {
```

```
		ui8_tmp=(ui8_data/100);
		uart_send(ui8_tmp+48);
		ui8_data-=ui8_tmp*100;
	}
	else
	{
		uart_send(48);
	}
	if (ui8_data>=10)
	{
		ui8_tmp=(ui8_data/10);
		uart_send(ui8_tmp+48);
		ui8_data-=ui8_tmp*10;
	}
	else
	{
		uart_send(48);
	}

	uart_send(ui8_data+48);
}

/**
 * Sends an uint16_t in a human readable format
 */
inline void uart_send16h(int16_t i16_data)
{

	uint16_t ui16_tmp=0;
	if (i16_data>=0)
	{
		uart_send(' ');
	}
	else
	{
		uart_send('-');
		i16_data=-i16_data;
	}

	if (i16_data>=10000)
	{
		ui16_tmp=(i16_data/10000);
		uart_send(ui16_tmp+48);
		i16_data-=ui16_tmp*10000;
	}
	else
	{
		uart_send(48);
	}
	if (i16_data>=1000)
	{
		ui16_tmp=(i16_data/1000);
		uart_send(ui16_tmp+48);
		i16_data-=ui16_tmp*1000;
	}
	else
	{
		uart_send(48);
	}
```

```
	if (i16_data>=100)
	{
		ui16_tmp=(i16_data/100);
		uart_send(ui16_tmp+48);
		i16_data-=ui16_tmp*100;
	}
	else
	{
		uart_send(48);
	}
	if (i16_data>=10)
	{
		ui16_tmp=(i16_data/10);
		uart_send(ui16_tmp+48);
		i16_data-=ui16_tmp*10;
	}
	else
	{
		uart_send(48);
	}

	uart_send(i16_data+48);

}


/**
 * Signal handler for incoming data
 */
SIGNAL(SIG_UART_RECV)
{
	uint8_t ui8_data = UDR;

	uartVer2(ui8_data);

} // SIGNAL(SIG_UART_RECV)


/**
 * The
 */
inline int uartVer2(uint8_t ui8_data)
{
		switch (ui8_wait_for_data)
		{
		case 0:
			switch(ui8_data)
			{
				case V2_HUMAN_MODUS:
					ui8_h = 1;
					send_msg(1);
					break;

				case V2_MACHINE_MODUS:
					ui8_h = 0;
					break;
```

```
// -------------------------
case V2_ROBOT_FORWARD:
       robotControl(LEGO_FORWARD);
       ui8_status=6;
       if(ui8_h==1)
       {
              send_msg(7);
       }
       break;

case V2_ROBOT_STOP:
       robotControl(LEGO_STOP);
       ui8_status=1;
       if(ui8_h==1)
       {
              send_msg(8);
       }
       break;

case V2_ROBOT_LEFT:
       robotControl(LEGO_LEFT);
       ui8_status=6;
       if(ui8_h==1)
       {
              send_msg(9);
       }
       break;

case V2_ROBOT_RIGHT:
       robotControl(LEGO_RIGHT);
       ui8_status=6;
       if(ui8_h==1)
       {
              send_msg(10);
       }
       break;

case V2_ROBOT_BACKWARD:
       robotControl(LEGO_BACKWARD);
       ui8_status=7;
       if(ui8_h==1)
       {
              send_msg(11);
       }
       break;

// -------------------------
case V2_ROBOT_SET_POS:
       ui8_robot_cmd = ui8_data;
       ui8_wait_for_data =6;
       break;

case V2_ROBOT_CLR_POS:
       i8_reset = 1;
       if(ui8_h==1)
       {
              send_msg(12);
       }
       break;
```

```
                        case V2_ROBOT_GET_POS:


                                if(ui8_h==1)
                                {
                                        uart_send(0x0A);
                                        uart_send(0x0D);
                                        uart_send('P');
                                        uart_send('o');
                                        uart_send('s');
                                        uart_send(':');
                                        uart_send(' ');
                                        uart_send('x');
                                        uart_send(':');
                                        uart_send16h((uint16_t)f_robotX);
                                        uart_send(' ');
                                        uart_send('y');
                                        uart_send(':');
                                        uart_send16h((uint16_t)f_robotY);
                                        uart_send(' ');
                                        uart_send('T');
                                        uart_send('h');
                                        uart_send('e');
                                        uart_send('t');
                                        uart_send('a');
                                        uart_send(':');

        uart_send16h((uint16_t)(f_robotTheta*180/M_PI));
                                        uart_send(0x0A);
                                        uart_send(0x0D);
                                }
                                else
                                {
                                        uart_send(V2_MSG_POSITION_HEADER);
                                        uart_send16((uint16_t)f_robotX);
                                        uart_send16((uint16_t)f_robotY);
                                        uart_send16((uint16_t)(f_robotTheta*1000));
    // 1/1000 rad resolution.
                                }


                                break;

                        // --------------------------
                        case V2_RADAR_SET_ANGLE:
                                ui8_robot_cmd = ui8_data;
                                ui8_wait_for_data =1;
                                break;

                        case V2_RADAR_GET_ANGLE:
                                if(ui8_h==1)
                                {
                                        uart_send(0x0A);
                                        uart_send(0x0D);
                                        uart_send('R');
                                        uart_send('a');
                                        uart_send('d');
                                        uart_send('a');
```

```
                uart_send('r');
                uart_send('a');
                uart_send('n');
                uart_send('g');
                uart_send('l');
                uart_send('e');
                uart_send(':');
                uart_send(' ');
                uart_sendh((uint8_t) (90-i8_servoAngle));
                uart_send(0x0A);
                uart_send(0x0D);
        }
        else
        {
                uart_send((uint8_t) (90-i8_servoAngle));
        }
        ui8_wait_for_data =1;
        break;

// --------------------------
case V2_STATUS:
        if(ui8_h==1)
        {
                uart_send(0x0A);
                uart_send(0x0D);
                uart_send('R');
                uart_send('o');
                uart_send('b');
                uart_send('o');
                uart_send('t');
                uart_send('s');
                uart_send('t');
                uart_send('a');
                uart_send('t');
                uart_send('u');
                uart_send('s');
                uart_send(':');
                uart_send(' ');
                uart_sendh(ui8_status);

                uart_send(0x0A);
                uart_send(0x0D);
        }
        else
        {
                uart_send(V2_MSG_STATUS_HEADER);
                uart_send(ui8_status+48);

        }
        break;

case V2_PING:
        uart_send(V2_PING_RESPONSE);
        uart_send('k');
        break;

// --------------------------
case V2_ROBOT_SET_HEADING:
        ui8_robot_cmd     = ui8_data;
```

```
		ui8_wait_for_data = 1;
		break;

	case V2_ROBOT_SET_DELTA_DIST:
		ui8_robot_cmd     = ui8_data;
		ui8_wait_for_data = 1;
		break;

	case V2_ROBOT_GO_POS:
		ui8_robot_cmd     = ui8_data;
		ui8_wait_for_data = 4;
		break;

	// -------------------------
	case V2_GET_IR_SCAN:
		ui8_status=5;
		break;

	case V2_GET_SENSOR_1:
		if (ui8_status !=5)
		{
			sendIR(0);
			if(ui8_h==1)
			{
				uart_send(0x0A);
				uart_send(0x0D);
			}
		}
		break;

	case V2_GET_SENSOR_2:
		if (ui8_status !=5)
		{
			sendIR(1);
			if(ui8_h==1)
			{
				uart_send(0x0A);
				uart_send(0x0D);
			}
		}
		break;

	case V2_GET_SENSOR_3:
		if (ui8_status !=5)
		{
			sendIR(2);
			if(ui8_h==1)
			{
				uart_send(0x0A);
				uart_send(0x0D);
			}
		}
		break;

	case V2_GET_SENSOR_4:
		if (ui8_status !=5)
		{
			sendIR(3);
			if(ui8_h==1)
```

```
                {
                        uart_send(0x0A);
                        uart_send(0x0D);
                }
        }
        break;

case V2_CALIBRATE_SENSORS:
        ui8_robot_cmd = ui8_data;
        ui8_wait_for_data =1;
        break;

// -----------

case V2_RECHARGE_COMPLETE:
        battery_init(5200);
        break;

case V2_CHECK_BATTERY:
        if (ui8_needsToCharge)
        {
                uart_send(V2_BATTERY_LOW);
                uart_send16(ui16_chargingTime);
        }
        else
        {
                uart_send(V2_BATTERY_OK);
        }
        break;


// --------------------------
case V2_GET_RIGHTWHEEL: //For testing and debug only
        // Total measured distance 8 MSB //
        uart_send((uint8_t) (i16_rightWheelTicks >> 8));
        // Total measured distance 8 LSB //
        uart_send((uint8_t) (i16_rightWheelTicks & 0xff));
        break;

case V2_GET_LEFTWHEEL: //For testing and debug only
        uart_send((uint8_t) (i16_leftWheelTicks >> 8));
        // Total measured distance 8 LSB //
        uart_send((uint8_t) (i16_leftWheelTicks & 0xff));
        break;

case V2_SET_VER:
        ui8_robot_cmd     = ui8_data;
        ui8_wait_for_data = 1;
        break;

case V2_SET_C: //Sets data to Port C (for testing)
        ui8_robot_cmd     = ui8_data;
        ui8_wait_for_data = 1;
        break;

// Didn't get a right command
default:
        uart_send(V2_NACK);
        break;
```

```
                        } // switch(data)
                        break;
                case 1:
                        ui8_data1=ui8_data;
                        switch(ui8_robot_cmd)
                        {
                                case V2_SET_C:
                                        PORTC = ui8_data;
                                        uart_send(V2_ACK);
                                        break;

                                case V2_ROBOT_SET_POS:
                                        setRobotPos(
ui8_data5|ui8_data6<<8,ui8_data3|ui8_data4<<8,ui8_data1|ui8_data2<<8);
                                        uart_send(V2_ACK);
                                        break;

                                case V2_RADAR_SET_ANGLE:
                                        servo_set_angle(90-ui8_data);
                                        wait_ms(1000);
                                        uart_send(V2_ACK);
                                        break;

                                case V2_ROBOT_SET_HEADING:
                                        ui8_status=2;
                                        ui8_targetHeading=ui8_data;
                                        uart_send(V2_ACK);
                                        break;

                                case V2_ROBOT_SET_DELTA_DIST:
                                        if (ui8_status==2 || ui8_status==3)
                                        {
                                                ui8_status = 3;
                                        }
                                        else
                                        {
                                                ui8_status = 4;
                                        }
                                        i8_targetDistance = ui8_data;
                                        uart_send(V2_ACK);
                                        break;

                                case V2_ROBOT_GO_POS:

        goRobotPos(ui8_data3|ui8_data4<<8,ui8_data1|ui8_data2<<8);
                                        uart_send(V2_ACK);
                                        break;


                                case V2_SET_VER:
                                        uartVer = ui8_data;
                                        uart_send(V2_ACK);
                                        break;

                                case V2_CALIBRATE_SENSORS:
                                        ui8_status = 8;
                                        ui8_sensornr = ui8_data;
                                        break;
```

```
                        default:
                              uart_send(V2_NACK);
                              break;

                  } // switch(robot_cmd)
                  ui8_robot_cmd     = 0;
                  ui8_wait_for_data = 0;
                  break; //case 1

            case 2:
                  ui8_data2 = ui8_data;
                  ui8_wait_for_data--;
                  break;

            case 3:
                  ui8_data3 = ui8_data;
                  ui8_wait_for_data--;
                  break;

            case 4:
                  ui8_data4 = ui8_data;
                  ui8_wait_for_data--;
                  break;

            case 5:
                  ui8_data5 = ui8_data;
                  ui8_wait_for_data--;
                  break;

            case 6:
                  ui8_data6 = ui8_data;
                  ui8_wait_for_data--;
                  break;

            case 7:
                  ui8_data4 = ui8_data;
                  ui8_wait_for_data--;
                  break;

            case 8:
                  ui8_data5 = ui8_data;
                  ui8_wait_for_data--;
                  break;

            case 9:
                  ui8_data6 = ui8_data;
                  ui8_wait_for_data--;
                  break;

            case 10:
                  ui8_data6 = ui8_data;
                  ui8_wait_for_data--;
                  break;

            } // switch (wait_for_data)
      return 0;
} //int uartVer2(char data)
```

B. Utdrag av tidligere produsert C-kode

C-kode henvist fra teksten, som ikke er med i vedlegg A, ettersom koden er uendret.

position.c

Beregner robotens posisjon

```
/**
 *  position.c - position routines
 *
 * @author Håkon Skjelten <skjelten@pvv.org>
 * @author Changed by Sveinung Helgeland for version 2.0 of RoboRadar
 * @author Radicaly changed by Bjørn Syvertsen for version 3.0 of RoboRadar
 * @author Changed and moved to own file by Johannes Schrimpf
 * @author Changed by Jannicke Selnes Tusvik
 * @file position.c
 * @brief position routines
 */

#include <avr/io.h>
#include <math.h>
#include <stdio.h>

#include "position.h"
#include "navigation.h"
#include "encoder.h"
#include "main.h"
#include "messages.h"


//extern globals

float               f_robotX            = 0;   // robot X pos in mm
float               f_robotY            = 0;   // robot Y pos in mm
float               f_robotTheta        = 0;   // robot heading in rad
int16_t             i8_servoAngle       = 0;   // Servo Angle in deg (90 is on left
                    side and -90 on right side)
int8_t              i8_reset     = 0;   // reset=1 => position will be resetet in
                    next updatePos()


// intern
int16_t i16_leftWheelOld    = 0;
int16_t i16_rightWheelOld   = 0;
int16_t i16_leftWheelNew    = 0;
int16_t i16_rightWheelNew   = 0;


/**
 * reset the robotposition to (0,0,0)
 */
void resetRobotPos()
{

      f_robotX            = 0;
      f_robotY            = 0;
      f_robotTheta        = 0;
```

```
	i16_leftWheelOld    = 0;
	i16_rightWheelOld   = 0;
	i16_leftWheelNew    = 0;
	i16_rightWheelNew   = 0;

	i16_rightWheelTicks = 0;
	i16_leftWheelTicks  = 0;
	i8_reset            = 0;
}

/**
 * Calculates the position of the robot
 */
void calcRobotPos(int16_t i16_oldTicksLeft,int16_t i16_newTicksLeft,int16_t
i16_oldTicksRight,int16_t i16_newTicksRight)
{


	float f_dthetaR = 0;
	float f_dxR     = 0;
	float f_sl      = 0;
	float f_sr      = 0;

	if(ui8_status == 2 || ui8_status == 3)
	{
		f_sl         = (i16_newTicksLeft  - i16_oldTicksLeft ) *
X_FACTOR_ROTATE;
		f_sr        = (i16_newTicksRight - i16_oldTicksRight) *
X_FACTOR_ROTATE;
	}
	else if(ui8_status == 4)
	{
		f_sl         = (i16_newTicksLeft  - i16_oldTicksLeft ) *
X_FACTOR_DRIVE;
		f_sr        = (i16_newTicksRight - i16_oldTicksRight) *
X_FACTOR_DRIVE;
	}
	else
	{
		f_sl         = (i16_newTicksLeft  - i16_oldTicksLeft ) * X_FACTOR;
//How far the left wheel has travelled in [mm]. See encoder.h for X_FACTOR
		f_sr        = (i16_newTicksRight - i16_oldTicksRight) * X_FACTOR;
// How far the right wheel has travelled in [mm]
	}


	f_dxR           = (f_sr+f_sl)/2.0 ; // How far the midpoint of the robot has
                                          travelled
	f_dthetaR       = (f_sr - f_sl) / WHEELBASE_MM ; // Find dthetaR by using
                                        the formula for arclength, (buelengde).

	// These formulas can be found in Syvertsen Master 2006 report, page 15.
	f_robotX        = f_robotX + ( f_dxR * cos(f_robotTheta + f_dthetaR/2.0) );
    f_robotY        = f_robotY + ( f_dxR * sin(f_robotTheta + f_dthetaR/2.0) );
    f_robotTheta    = f_robotTheta + f_dthetaR;


    if( f_robotTheta >= TO_PI )
		f_robotTheta = f_robotTheta - TO_PI;
```

```
    else if( f_robotTheta < 0 )
             f_robotTheta = f_robotTheta + TO_PI;
}


/**
 * Set a new robot position
 */
void setRobotPos(int16_t x, int16_t y, int16_t theta){
      f_robotX     = (float) x;
      f_robotY     = (float) y;
      f_robotTheta = ((float) theta)/1000;

}

/**
 * Updates the position of the robot
 */
inline void updateRobotPos(){

      //sample
      i16_leftWheelNew  = i16_leftWheelTicks;
      i16_rightWheelNew = i16_rightWheelTicks;

      //calculate
      calcRobotPos(i16_leftWheelOld,i16_leftWheelNew,i16_rightWheelOld,i16_rightWh
eelNew);

      //update
      i16_leftWheelOld  = i16_leftWheelNew;
      i16_rightWheelOld = i16_rightWheelNew;

      if( i8_reset )
      {
            resetRobotPos();
      }

}


/**
 *  calculates the x-position of a meassured object.
 */
inline int16_t objectPosX(int16_t i16_sensorT, uint8_t ui8_sensorDelta)
{

      ui8_sensorDelta+=2;
      //i16_servoAngle negative because -90 is on left side and +90 on right side
      float f_theta   = ((float)(i16_sensorT+i8_servoAngle))/180*M_PI +
f_robotTheta;
      float f_objectX = f_robotX - (32*cos(f_robotTheta)) + (
((float)ui8_sensorDelta*10) * cos(f_theta));

      return((int16_t)f_objectX);

}

/**
 *  calculates the y-position of a meassured object.
```

```
 */
inline int16_t objectPosY(int16_t i16_sensorT, uint8_t ui8_sensorDelta)
{
	ui8_sensorDelta+=2;
	//i16_servoAngle negative because -90 is on left side and +90 on right side
	float f_theta   = ((float)(i16_sensorT+i8_servoAngle))*M_PI/180 +
f_robotTheta;
	float f_objectY = f_robotY - (32*sin(f_robotTheta)) + (
((float)ui8_sensorDelta*10) * sin(f_theta));

	return((int16_t)f_objectY);

}



/**
 * Calculates a heading and a distance to move to a position
 */
void goRobotPos(int16_t i16_goX, int16_t i16_goY){
	float f_goX           = (float) i16_goX;
	float f_goY           = (float) i16_goY;
	float diffXSquareInCm = ((f_goX-f_robotX)*(f_goX-f_robotX))/100;
	float diffYSquareInCm = ((f_goY-f_robotY)*(f_goY-f_robotY))/100;
	int16_t i16_angle       = (int16_t) (atan2(f_goY-f_robotY,f_goX-
f_robotX)*180/M_PI);
	if (i16_angle<0)
	{
		i16_angle+=360;
	}
	ui8_status=3;
	ui8_targetHeading      = i16_angle/2;
	i8_targetDistance     = (int8_t)sqrt(diffXSquareInCm+diffYSquareInCm);

}
```

C. Matlab-kode

Nye filer og filer med viktige endringer er vedlagt her. De er tatt med i sin helhet av kontekst- og lesbarhetshensyn. Dette fører til veldig mange sider vedlegg, ettersom det er snakk om mye kode, men uten omkringliggende kode ville det i mange tilfeller vært umulig å forstå funksjon og virkemåte.

LineBeaconSLAM.m

Grensesnittets hovedløkke. Dette er et stort dokument, hvor noen endringer er spredt rundt, men de største endringene finnes til slutt.

```
%% SLAM which produces a global map based on lines

function lineBeaconSLAM(myCon,myHandles,myLineParams,myBeaconParams)
global ROBOTPOSE;
global REALPOSE;
global RUNNING;



REALPOSE=[0 0 0];
% "robothouse" parameters
%WIDTH=0.515; % [meter]
%DEPTH=0.408; % [meter]


myLineParams.robot.class = 'robotdd';

dispmode = myLineParams.displayGlobal;
display = myLineParams.displayLocal;
displaying= myLineParams.displayMatch;

goHome = myLineParams.goHome;
timeLeftToGoHome = myHandles.goHometext;

% ----- Slam ----- %
% optional axis vector for global map figure. Useful with infinite lines
myLineParams.axisvec = [-2.2 2.2 -2.2 2.2];

G = initmap(0,'global map',myLineParams);
myLineParams.robot.xsensor= [0 0 0];
IRdata.params.xs=myLineParams.robot.xsensor;
IRdata.params.stdrho = myLineParams.sensor.stdrho;


istep = 0;

navData.robotPath=[];
navData.state=1;
navData.backTrackPoint=[];
navData.STARTbackTrackPoint=[];
navData.backTrackPointAtGap=[];
navData.allBackTrackPoints=0;
navData.backTrackPointAtGap=[];
navData.lastHeading=0;
```

```
scndata=[];
fullscan=1;
aL_total=0;
aR_total=0;
oldPose=[0 0 0];
clc;


%% main while loop for mapping

while(RUNNING)


    drawnow;%keep the GUI active

    set(G,'time',istep);
    Gin = G;

    [myNewpose,error]=getRobotPose(myCon,myHandles);
    if(error)
        disp('error fetching robot pose');
        continue;
    end;


    % display current position in Gui
    set(myHandles.xPosText,'String',num2str(myNewpose(1)) );
    set(myHandles.yPosText,'String',num2str(myNewpose(2)) );
    set(myHandles.tPosText,'String',num2str(myNewpose(3)*180/pi) );
    if strcmp(myCon,'simulator'),
        set(myHandles.edit_realposeX,'String',num2str(REALPOSE(1)*10 ));
        set(myHandles.edit_realposeY,'String',num2str(REALPOSE(2)*10 ));
%
set(myHandles.edit_realposeTheta,'String',num2str(myNewpose(3)*180/pi ));
        set(myHandles.edit_realposeTheta,'String',num2str(REALPOSE(3) ));
    end

    pause(0.01);
    if RUNNING==2,
        while ~(RUNNING==1 || RUNNING==0)
            pause(0.1);
        end
        if ~RUNNING,
            break;
        end
    end

    myNewpose=[myNewpose(1)/1000 myNewpose(2)/1000 myNewpose(3)];

    utover= get(Gin,'x');
    utover{1,1}=set( utover{1,1},'x', myNewpose' );
    Gin=set(Gin,'x',utover);

    % --------------------
    % --- State prediction
    % --------------------

    % Changed to allow for different setups
    if strcmp(myCon, 'NXT_kamera')
        RAD_LEFT_WHEEL=0.028;       % As printed on tyre
```

```
        RAD_RIGHT_WHEEL=0.028;      % As printed on tyre
        WHEEL_BASE=0.120;           % Measured
        EGC=0.200;                  % error growth coefficient on left and
right wheel in [1/m].
        [aL,aR]=getAngularDispl(myNewpose,oldPose,
RAD_LEFT_WHEEL,RAD_RIGHT_WHEEL,WHEEL_BASE);
        aL_total=aL_total+aL/500; % Angular displacements of left wheel in
[rad]
        aR_total=aR_total+aR/500; % Angular displacements of right wheel in
[rad]
    else
        RAD_LEFT_WHEEL=0.0248;  % Changed from 0.02149 because the wheels
are replaced
        RAD_RIGHT_WHEEL=0.0248; % Changed from 0.02149 because the wheels
are replaced
        WHEEL_BASE=0.210;    % Changed from 0.1894938 because the wheels
are replaced
        EGC=0.004;           % error growth coefficient on left and right
wheel in [1/m]. Decreased from 0,03 because error correction is now
implemented on microcontroller level.
        [aL,aR]=getAngularDispl(myNewpose,oldPose,
RAD_LEFT_WHEEL,RAD_RIGHT_WHEEL,WHEEL_BASE);
        aL_total=aL_total+aL/500; % Angular displacements of left wheel in
[rad]
        aR_total=aR_total+aR/500; % Angular displacements of right wheel in
[rad]
    end

    if strcmp(myCon,'simulator') && ~myHandles.poseError, % simulation
without position error: set errors to zero
        EGC=0;
        aL_total=0;
        aR_total=0;
    end

%    ENC.PARAMS.KL: error growth coefficient of left wheel with unit in
[1/m].
%    ENC.PARAMS.KR: error growth coefficient of right wheel with unit in
[1/m].
%    ENC.STEPS(i).DATA1: angular displacements of left wheel in [rad] and
%    monotonically increasing.
%    ENC.STEPS(i).DATA2: angular displacements of right wheel in [rad] and
%    monotonically increasing.

    enc.params.kr=EGC;
    enc.params.kl=enc.params.kr;
    enc.steps(istep+1).data1=aL_total;
    enc.steps(istep+1).data2=aR_total;


        % Get the robot object
        r = getrobot(Gin);

        % Given the robot at its initial pose and the encoder data in
sensors(i)
        % the predict method performs the state prediction of the robot. It
re-
        % turns the new pose, its covariance matrix, the Jacobian of the
process
        % model derived with respect to the robot and a 3xn matrix PATH of
all
```

```
        % poses between xr and xrout
        [r,Frout,path] = predict(r,enc);

        % Predict the map given the predicted robot and the Jacobian FROUT
        Gin = setrobot(Gin,r);

        % Predict the map given the predicted robot and the Jacobian FROUT
        Gin = robotdisplacement(Gin,get(r,'x'),get(r,'C'),Frout);



    IRdata.steps.data=[];
% -----performing a full scan-------- %
    if fullscan,
        %set robot pose in map



        [error,data]=fullScan(myCon,myHandles);

        %Because cam robot often finds no data points, this condition is
        %necessary:
        if(~isempty(data)),
            data(:,2:3)=data(:,2:3)/1000; % converting from mm to meter

            %figure(66);clf;hold on;axis equal;
            %plot(data(:,2), data(:,3),'yx');

            % rotate the global x,y points received from robot to local x,y
points
            % in order to use them in extractlines
            rotmat=[cos(myNewpose(3)) sin(myNewpose(3)); -sin(myNewpose(3))
cos(myNewpose(3)) ];

            data(:,2)=data(:,2)-myNewpose(1);
            data(:,3)=data(:,3)-myNewpose(2);

            data(:,2:3)=(rotmat*(data(:,2:3))')';

            %data(find(-0.1<data(:,3) & data(:,3)<0.1),:)=[];

            IRdata.steps.data=[];

            if strcmp(myCon,'NXT_kamera')
                NS=1;   %Just one "sensor"
            else
                NS=4;   %Four sensors
            end
            for sens=1:1:NS,
                tmpData=data(data(:,1)==sens,:);

                IRdata.steps.data=vertcat(IRdata.steps.data,tmpData);
            end
            %plot(IRdata.steps.data(:,2), IRdata.steps.data(:,3),'mx');
        else
            IRdata.steps.data=[];
        end
```

```
% ------- processing data from driving------- %
    elseif size(scndata,1)>0, % there are data collected during driving
which need to be considered
        myNewpose(1:2)=oldPose(1:2);


        utover= get(Gin,'x');
        utover{1,1}=set( utover{1,1},'x', myNewpose' );
        Gin=set(Gin,'x',utover);


        % rotate the global x,y points received from robot to local x,y
points
        % in order to use them in extractlines
        rotmat=[cos(myNewpose(3)) sin(myNewpose(3));-sin(myNewpose(3))
cos(myNewpose(3)) ];

        scndata(:,2)=scndata(:,2)-myNewpose(1);
        scndata(:,3)=scndata(:,3)-myNewpose(2);
        scndata(:,2:3)=(rotmat*(scndata(:,2:3))')';

        scndata(find(-0.1<scndata(:,3) & scndata(:,3)<0.1),:)=[];
        for sens=2:2:5,

            tmpData=scndata(scndata(:,1)==sens,:);
            % artifical scandata..
            [artifx artify]=pol2cart((-1:0.2:1)*pi/8 ,0.02);
            artif=zeros(length(artifx),3);
            artif(:,2:3)=horzcat(artifx',artify');
            tmpData=vertcat(tmpData,artif);

            IRdata.steps.data=vertcat(IRdata.steps.data,tmpData);
        end



    end %fullscan


    IRdata.steps.data1=[];
    IRdata.steps.data2=[];
    IRdata.steps.data3=[];
    if size(IRdata.steps.data,1)>0, % If there are sensordata
        IRdata.steps.data1=IRdata.steps.data(:,1);
        IRdata.steps.data2=IRdata.steps.data(:,2);
        IRdata.steps.data3=IRdata.steps.data(:,3);



%% Manipulate raw sensor data in order to make a global map
%% Line extraction

    % Exclude point features here
    [Gin,E]=include(Gin,'alpha,r line feature');

    % Extract features from the raw measurements in data structure given
the
    % extraction algorithm parameters in myPARAMS. The local map L is
returned
    % With the flag DISPLAY = 1, the extraction result is plotted in
```

```
    % a new figure window.

    if fullscan,
        [L,rawSegs,lines] = extractlines( IRdata
,myLineParams,myNewpose,display);
    else
        myLineParams.cyc=0;
        [L,rawSegs,lines] = extractlines2( IRdata
,myLineParams,myNewpose,display);
        myLineParams.cyc=1;
    end



    % --------------------------
    % --- Measurement Prediction
    % --------------------------

    % Predict the map G at the current robot pose, its uncertainty and the
    % robot-to-sensor displacement in PARAMS.
    [Gpred,Hpred] = predictmeasurements(Gin);

    % ------------
    % --- Matching
    % ------------


    % Apply the nearest neighbor standard filter to match the observations
in
    % the local map L to the predicted measurements GPRED and accept the
    % pairing on the level alpha in PARAMS.
    [nu,R,H,associations,matches_ID_G_L] =
matchnnsfLines(Gpred,Hpred,L,myLineParams,displaying);

    % --------------
    % --- Estimation
    % --------------

    % Update the map with an extended Kalman filter based on the stacked
    % innovation vector nu and the stacked observation covariance matrix R.
    %Bjørn S
    G = estimatemapLines(Gin,nu,R,H,matches_ID_G_L,myNewpose,rawSegs);
    %Bjørn S

    % ------------------------------
    % --- Integrate new observations
    % ------------------------------
    % Add the non-associated observations marked by a -1 as the global
matching
    % feature index in ASSOCIATIONS to the map and return the augmented map
    G = integratenewobs(G,L,associations);



% ---------post processing of map after line-extraction----------- %

%FIXCORNERS. Nearby segments that make up a corner gets linked.
    %The segments must be at least minLength long and have endpoints
    %cloaser than maxDist to be liked.
    maxDist= 0.1; minLength=0.1;
    G=fixCorners(G, maxDist, minLength);
```

```
% delete pointfeatures close to segments
    Tmp=get(E,'x');
    Ctmp=get(E,'c');
    if ~isempty(Tmp),
        xyp=[];
        Xg=get(G,'x');
        %Cg=get(G,'c');
        for p=1:length(Tmp)
            xyp(p,1:2)=get(Tmp(Kallevik),'x');
            %Cp(p+1:2*p,)=Tmp{p,'c'};
        end
        segs=[];
        % find segments
        for i=2:length(Xg)
            ss=get(Xg{1,i},'ss');
            tmp=[];
            for j=1:4:size(ss,2),
                tmp=vertcat(tmp,ss(j:j+3));
            end

            for t=1:size(tmp,1),
                if tmp(t,1)> tmp(t,3), % sort with respect to x
                    tmp(t,1:4)=[tmp(t,3) tmp(t,4) tmp(t,1) tmp(t,2)];
                end
            end
            segs=vertcat(tmp,segs);
        end
        seg=size(segs,1);
        while seg>0,
            if ~isempty(xyp),
            dist = distancePointEdge(xyp, segs(seg,:));
            indInte=dist<0.03;
            Tmp(indInte)=[];
            % update covarianse matrix
            Ctmp(indInte,:)=[];
            Ctmp(:,indInte)=[];

            xyp(indInte,:)=[];
            end
            seg=seg-1;
        end
    end


%% preparation for beacon extraction


    %find raw data not integrated in global map
    xR=IRdata.steps.data(:,2);
    yR=IRdata.steps.data(:,3);

    rawData=[xR yR];
    if size(rawSegs,2)>0,
        segs=rawSegs(13:16,:)';

        for seg=1:size(segs,1),
            dist = distancePointEdge(rawData, segs(seg,:));
```

```
            indInte=dist<0.03;
            rawData(indInte,:)=[];
        end

    end


    % -----include pointfeatures to map----- %
    Xg=get(G,'x');
    Cg=get(G,'c');
    Xnew=horzcat(Xg,Tmp);

    totalL=size(Cg,2)+size(Ctmp,2);
    Cnew=Cg;
    if size(Ctmp,2)>0,
        Cnew(size(Cg,2)+1:totalL,size(Cg,2)+1:totalL)=Ctmp;
        % add zeros to covarianse matrix
        for e=size(Cg,2)+1:totalL,
            Cnew(e,1).C=eye(2,3)*zeros(3,3);
            Cnew(1,e).C=zeros(3,3)*eye(3,2);
            for h=2:size(Cg,2),
                Cnew(h,e).C=zeros(2,2);
                Cnew(e,h).C=zeros(2,2);
            end
        end
    end
    % put all features back to map G
    G=set(G,'x',Xnew,'c',Cnew);



%% Beacon extraction..
    if fullscan, % extract points from a fullscan only

    % check for beacon extraction
    % constant beacon covariance matrix
    myBeaconParams.Cb = 0.01*eye(2);
    % minimal nof raw points on reflector in order to be valid
    myBeaconParams.minnpoints = 3;

    % produce a perception map P containing only pointfeatures in a radius
equal to maxperceptionRaduis.
    % Gin contains pointfeatures outside perception radius and all other
    % features not considered during pointextraction
    [P,Gin]=GtoPconv9(G,myHandles);


    % use only rawData which haven't contributed to line extraction
    for k=1:size(rawData,1),
        Pin=P;


        x=rawData(k,1);
        y=rawData(k,2);
        IRdata.steps.data1=3;
        IRdata.steps.data2=[x;x;x];
        IRdata.steps.data3=[y;y;y];


        % beacon extraction functions
```

```
        L = extractbeacons2( IRdata ,myBeaconParams,0,myNewpose);
        [Ppred,Hpred] = predictmeasurements(Pin);
        [nu,R,H,associations] =
matchnnsf(Ppred,Hpred,L,myBeaconParams,displaying);
        P = estimatemap(Pin,nu,R,H);
        P = integratenewobs(P,L,associations);

    end

    % merge updated perception map P and map Gin containing the other
    % features

    G=PtoGconv9(P,Gin);


    end % fullscan



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%
%% Map post-processing


    %record robot path
    navData.robotPath(istep+1,1:1:3) = myNewpose;

    poseM(1:istep,1:2)=navData.robotPath(1:istep,1:2);
    poseN(1:istep,1:2)=navData.robotPath(2:istep+1,1:2);
    totalDist = sum(sqrt((poseN(:,1)-poseM(:,1)).^2+(poseN(:,2)-
poseM(:,2)).^2));
    navData.totalDist=totalDist;
    %disp('total distance travelled:');
    %disp(totalDist);
    %%%%%%%%%%%%%%%%%%

% ---------post processing of map after point-extraction----------- %
    % deleting points overlapping linesegments
    G=deletePointsFromMap(G);
% ------------------------------------------------------------------- %



    %GAPFINDER. finds gaps/openings on single lines in the map
    % and other types of gaps...

    minGap=0.25; maxGap=0.7;

    initGaps=[];
    gaps = gapFinder3(G,minGap,maxGap,navData.robotPath,initGaps);


    end %if there are sensordata
%% Visualize mapping step
    % ------------------
    % --- Visualize mapping step
    % ------------------

    cla(myHandles.mainPlotWindow);
    if dispmode >= 1,
```

```
        axes(myHandles.mainPlotWindow);
        hold(myHandles.mainPlotWindow,'on');

        set(gca,'Box','On'); axis equal;
        title(['Global map at step ',int2str(istep),...
            '. Total distance: ',num2str(totalDist,3),' m']);
        draw(G,0,1,0,'k');  % draw lines(infinite length) and robot


        if isfield(myLineParams,'axisvec');
            axis(myLineParams.axisvec);
        end;

        % draw segments
        globalMap=get(G,'x');
        % only consider arline features
        f=length(globalMap);
        while f>1,
            if ~strcmp(get(globalMap(Free2Move),'Type'),'alpha,r line
feature'),
                globalMap(f)=[];
            end
            f=f-1;
        end
        endpoints=[];
        for i=2:length(globalMap)
            startnstopp=get(globalMap{1,i},'ss');
            for j=1:4:size(startnstopp,2)
                plot([startnstopp(j) startnstopp(j+2)],[startnstopp(j+1)
startnstopp(j+3)],'LineWidth',2.7,'Color',[1 0 0]); % Red segments
                plot(
startnstopp(j),startnstopp(j+1),'kx','LineWidth',2,'MarkerSize',9); %
Endpoint on red segment, marked with x
                plot(
startnstopp(j+2),startnstopp(j+3),'kx','LineWidth',2,'MarkerSize',9); %
Endpoint on red segment, marked with x
                endpoints=vertcat(endpoints,[startnstopp(j)
startnstopp(j+1);startnstopp(j+2) startnstopp(j+3)]);
            end;
        end;

        %draw gaps
        if numel(gaps) > 1,
            for i=1:1:size(gaps,1)
                for j=1:4:size(gaps,2)
                    plot([gaps(i,j) gaps(i,j+2)],[gaps(i,j+1)
gaps(i,j+3)],'y','LineWidth',2.7);
                end;
            end;
        end;

        % ---draw/concatenate points to edges---- %
        globalMap1=get(G,'x');
        % only consider arline features
        f=length(globalMap1);
        points=[];
        while f>1,
            if ~strcmp(get(globalMap1(Free2Move),'Type'),'point feature'),
                globalMap1(f)=[];
            end
            f=f-1;
```

```
        end
        for i=2:length(globalMap1)
            points(i-1,1:2)=get(globalMap1{1,i},'x');
        end
        points=vertcat(endpoints,points);

        edges=linkPoints(points,0.15,1);

        % unknown area detection function
        newDetect=1;
        if newDetect && fullscan

myHandles.checkTable=getUnknownAreas(G,myHandles,navData.robotPath,edges);
        end

        % draw robot path
        for i=1:1:size( navData.robotPath,1 )-1
            plot([navData.robotPath(i,1)
navData.robotPath(i+1,1)],[navData.robotPath(i,2)
navData.robotPath(i+1,2)],'g','LineWidth',1);
            plot(
navData.robotPath(i,1),navData.robotPath(i,2),'kx','LineWidth',2,'MarkerSiz
e',5);
        end;

    end;

%% Go Home

        if goHome == 1;
            timeLeftToGoHome = timeLeftToGoHome -1;
            set(myHandles.goHomeText,'String',timeLeftToGoHome);
        end;
        %Set robot to go home after a chosen value or when it is finished
        %mapping
        if timeLeftToGoHome == 0 || navData.state == 9 ||
get(myHandles.Recharge_checkbox,'Value') == 1;
            set(myHandles.statusText,'String','Calculating path home.');

            if exist('globalMap1') == 0;
                sp = computeShortestPath(globalMap,navData.robotPath);
            else
                sp =
computeShortestPathPoints(globalMap,globalMap1,navData.robotPath);
            end

            % Creating x and y vector
            x = [];
            y = [];
            for i=1:2:size(sp,2)
                x = [x sp(i)];
                y = [y sp(i+1)];
            end
            hold(myHandles.mainPlotWindow,'on');
            plot(x,y)
            hold(myHandles.mainPlotWindow,'off');

            sp = flipdim(sp,2);
            t=1;

            [poses,myError]=getRobotPose(myCon,myHandles);
```

```
            orgPos = [poses(1)/1000 poses(2)/1000]; %[m]
            oldPos = orgPos;
            set(myHandles.statusText,'String','Path found, going home!');

            for i=1:2:size(sp,2)
            % use cartesian coordinates to go to target
            x=sp(t+1)*1000; % x in [mm]
            y=sp(t)*1000; % y in [mm]

            error=goToPose2(myCon,x,y,myHandles);

            [poses,myError]=getRobotPose(myCon,myHandles); %[mm]
            newPos = [poses(1)/1000 poses(2)/1000]; %[m]


            % draw robot path
            hold(myHandles.mainPlotWindow,'on');
            plot([oldPos(1) newPos(1)],[oldPos(2)
newPos(2)],'g','LineWidth',1);
            plot( newPos(1),newPos(2),'kx','LineWidth',2,'MarkerSize',5);
            hold(myHandles.mainPlotWindow,'off');
            oldPos = newPos;
            t = t+2;

            end
            %Not active during simulation
            if ~myLineParams.simulation;
                [error]=dock(myCon,myHandles,myLineParams,navData);
                if(exist('chargeTime','var')==0)
                    chargeTime = 5;     %charge time in secs, if no charge
time received from robot
                end
                while (chargeTime > 0)  %charging
                    set(myHandles.statusText,'String',sprintf('Robot has
docked and is recharging. %d s left.', chargeTime));
                    pause(1);
                    chargeTime = chargeTime -1;
                end
                set(myHandles.statusText,'String','Finished recharging.');
                if( serialAsyncWrite(myCon,'>') )
                    set(myHandles.statusText,'String','Recharge complete,
NO MESSAGE TO ROBOT' ); %robot doesn't know..
                else
                    set(myHandles.statusText,'String','Recharge complete'
);
                end
            end;
            set(myHandles.Recharge_checkbox,'Value',0);


            %% Go back
            %If the robot is not finished mapping the reason it was going
            %back was to update the position.
            if navData.state ~= 9;
                navData.state = 93;
                %go back to start posistion
                [poses,myError]=getRobotPose(myCon,myHandles); %[mm]
                newPos = [poses(1) poses(2)]; %[m]
                x = newPos(1) + 150; %messured number from the wall to
starting position
```

```
                y = newPos(2);
                error=goToPose2(myCon,x,y,myHandles);
                error=setRobotPose(myCon,myHandles, [0 0 0]);
                sp = flipdim(sp,2);
                sp(1:2) = [];
                sp = [sp orgPos(1) orgPos(2)];
                sp = sp*1000;
                t=1;
                set(myHandles.statusText,'String','Returning to previous
position..');
                for i=1:2:size(sp,2)
                % use cartesian coordinates to go to target
                x=sp(t); % x in [mm]
                y=sp(t+1); % y in [mm]

                error=goToPose2(myCon,x,y,myHandles);
                [poses,myError]=getRobotPose(myCon,myHandles); %[mm]
                newPos = [poses(1)/1000 poses(2)/1000]; %[m]


                % draw robot path
                hold(myHandles.mainPlotWindow,'on');
                plot([oldPos(1) newPos(1)],[oldPos(2)
newPos(2)],'g','LineWidth',1);
                plot(
newPos(1),newPos(2),'kx','LineWidth',2,'MarkerSize',5);
                hold(myHandles.mainPlotWindow,'off');
                oldPos = newPos;
                t = t+2;
                end
            %Setting the robot state back to mapping
            set(myHandles.statusText,'String','Arrived, continuing
mapping');
            navData.state = 1;
            end
        end
%% Navigation

 % --- Navigation step --- %

    % produce points from checkTable in 2D plane
            cT=myHandles.checkTable;
            szcT=size(cT,1);
            rad=myHandles.maxPerceptRadius;
            res=myHandles.angularRes; %
            NOP=360/res;
            THETA=linspace(0, 2*pi, NOP);
            unknown=[];
            for pos=1:szcT,
                tmpAngles=THETA(find(cT(pos,2:end)==0))';
                [tmpX
tmpY]=pol2cart(tmpAngles,ones(length(tmpAngles),1)*rad);
                tmpX=tmpX +
repmat(navData.robotPath(cT(pos,1),1),length(tmpAngles),1);
                tmpY=tmpY +
repmat(navData.robotPath(cT(pos,1),2),length(tmpAngles),1);
                unknown=vertcat(unknown,[tmpX tmpY]);
            end

            [unknownGaps,edges]=linkPoints2(unknown,0.06,1);
```

```
            gaps=vertcat(gaps,unknownGaps);


    if fullscan,
        if( myLineParams.navigation == 1 )
            lastState=navData.state;


[navError,myNavData,myHandles,scndata]=leftWallFollowerToBackTracker(G,myCo
n,navData,gaps,myHandles);
            navData=myNavData;
            oldPose=myNewpose;
            if(navError)
                disp('Navigation error');
            end
        end
        if navData.state==1 && lastState==1 && myHandles.driveScan==1, %
driveScan is a checkbox in the GUI
            fullscan=0;
        end

        istep = istep+1;
    else
        drawRectangle(G,myNavData.b,myNavData.l);
        fullscan=1;
    end


    if ~myLineParams.simulation && ~strcmp(myCon,'NXT_kamera'),
        pause(0.1); %this line is not active during simulation

        % read data on COM port

        if myCon.bytesAvailable,
            data = fread(myCon);
            disp('leftover data on serialconnection between timesteps.
Timestep: ')
            disp(istep)
            disp('data: ')
            disp(data)
        end
    end
%% BATTERY RECHARGE CHECK

    % --- Set recharge checkbox if robot needs charging --- %

    %Not if simulator is active
    if ~myLineParams.simulation,
        if strcmp(myCon,'NXT_kamera') %NXT camera robot
            %get battery data
            [recharge chargeTime] = nxt_get_battery_state();
            if(recharge)
                disp('battery capacity LOW, setting recharge flag')
                set(myHandles.statusText,'String','BATTERY LOW! Recharge
initiated.');
                % Setting battery recharge flag
                set(myHandles.Recharge_checkbox,'Value',1);
            else
                % Battery capacity OK
                disp('battery capacity OK');
            end
```

```
        else
            %Asking for battery state
            if( serialAsyncWrite(myCon,'?') )
                set(myHandles.statusText,'String','Recharge check: Unable
to send serial data' );
            else
                [myError,data]=serialSyncRead(myCon,1);
                if(myError)
                    disp('slam/LineBeaconSLAM: Recharge check: no data
received..')
                elseif data=='='
                    % Battery capacity OK
                    disp('battery capacity OK')
                elseif data=='<'
                    disp('battery capacity LOW, setting recharge flag')
                    set(myHandles.statusText,'String','BATTERY LOW!
Recharge initiated.');
                    % Setting battery recharge flag
                    set(myHandles.Recharge_checkbox,'Value',1);
                    %Receiving robot charging time [seconds]
                    [myError,timeVector]=serialSyncRead(myCon,2);
                    if(myError)
                        disp('slam/LineBeaconSLAM: Error receiving battery
charging time from robot')
                    else
                        %converting 2 x 8 bit unsigned to 16 bit unsigned
                        chargeTime = timeVector(1)*256+timeVector(2);
                    end
                end
            end
        end
    end

end % main while loop
```

nxt_get_battery_state.m

Undersøker om kamera-robotens batteri trenger lading

```
function [recharge chargetime] = nxt_get_battery_state()
%checks whether the battery needs charging

    %Set recharge limits
    chargeVoltage = 8000; % lower battery limit in mV
    chargetime = 3600;      % secs

    mvolts = NXT_GetBatteryLevel; %millivolts
    disp('Battery: ');
    disp(mvolts);

    if (mvolts < chargeVoltage)
        recharge = 1;
    else
        recharge = 0;
    end

end
```

LegoGui.m

Denne filen styrer alt som har med brukerflaten (GUI) å gjøre. Den er svært lang, da alle knapper, og oppførselen deres, må defineres.

```
function varargout = LegoGui(varargin)
% LEGOGUI M-file for LegoGui.fig
%      LEGOGUI, by itself, creates a new LEGOGUI or raises the existing
%      singleton*.
%
%      H = LEGOGUI returns the handle to a new LEGOGUI or the handle to
%      the existing singleton*.
%
%      LEGOGUI('CALLBACK',hObject,eventData,handles,...) calls the local
%      function named CALLBACK in LEGOGUI.M with the given input arguments.
%
%      LEGOGUI('Property','Value',...) creates a new LEGOGUI or raises the
%      existing singleton*.  Starting from the left, property value pairs
are
%      applied to the GUI before LegoGui_OpeningFunction gets called.  An
%      unrecognized property name or invalid value makes property
application
%      stop.  All inputs are passed to LegoGui_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Copyright 2002-2003 The MathWorks, Inc.
% Edit the above text to modify the response to help LegoGui

% Last Modified by GUIDE v2.5 03-Jul-2011 22:30:33

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @LegoGui_OpeningFcn, ...
                   'gui_OutputFcn',  @LegoGui_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before LegoGui is made visible.
function LegoGui_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to LegoGui (see VARARGIN)
```

```
%a=get(LegoGui)
% Choose default command line output for LegoGui
handles.output = hObject;
handles.lineParam.changed='false';
handles.beaconParam.changed='false';

%handles.simParams.changed='false';

handles.checkTable=[];

%need to define robot-object and vertex-list for NXT camera robot

handles.kamerarobot=[];
handles.vertex_list_left = [];
handles.vertex_list_right = [];

% Update handles structure
guidata(hObject, handles);
% UIWAIT makes LegoGui wait for user response (see UIRESUME)
% uiwait(handles.figure1);


%%%%% defining globals %%%%%%
global SERIALLINK;
global RUNNING;
global table;

SERIALLINK = 0;
RUNNING = 0;
table = [];
%mappingChoise = 1;

%add all folders under/below the current to path
path(path,genpath(pwd));


% --- Outputs from this function are returned to the command line.
function varargout = LegoGui_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;

%% BASIC - define callbacks


% --- Executes on selection change in COMValg.
function COMValg_Callback(hObject, eventdata, handles)
% hObject    handle to COMValg (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: contents = get(hObject,'String') returns COMValg contents as cell
array
%        contents{get(hObject,'Value')} returns selected item from COMValg
```

```
% --- Executes during object creation, after setting all properties.
function COMValg_CreateFcn(hObject, eventdata, handles)
% hObject    handle to COMValg (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: popupmenu controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes on button press in connectButton.
function connectButton_Callback(hObject, eventdata, handles)
% hObject    handle to connectButton (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global SERIALLINK;

if(SERIALLINK==0)
    SERIALLINK = connectTo(handles);
elseif strcmp(SERIALLINK,'simulator'),
    set(handles.statusText,'String',strcat('Already connected to
simulator.' ));
else
    set(handles.statusText,'String',strcat(SERIALLINK.port,' is already:',
SERIALLINK.status,'. If you want to connect to another port, please close
the current connection first' ));
end



% --- Executes on button press in disconnectButton.
function disconnectButton_Callback(hObject, eventdata, handles)
% hObject    handle to disconnectButton (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global SERIALLINK;

if( SERIALLINK == 0 )
    set(handles.statusText,'String','Already disconnected');
elseif strcmp(SERIALLINK,'simulator'),
    SERIALLINK=0;
    set(handles.statusText,'String','Disconnected from simulator');
elseif strcmp(SERIALLINK,'NXT')
    SERIALLINK=0;
    set(handles.statusText,'String','Disconnected from NXT');
elseif strcmp(SERIALLINK,'NXT_kamera')
    SERIALLINK=0;
    set(handles.statusText,'String','Disconnected from NXT_kamera robot');
else
    stopasync(SERIALLINK); % Stop asynchronous read and write
    fclose(SERIALLINK); % Close file
    set(handles.statusText,'String',strcat(SERIALLINK.port,' is:',
SERIALLINK.status ));
    delete(SERIALLINK);
    SERIALLINK = 0;
end;
```

```
% --- Executes on button press in pingButton.
function pingButton_Callback(hObject, eventdata, handles)
% hObject    handle to pingButton (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global SERIALLINK;
if( SERIALLINK == 0 )
    SERIALLINK = connectTo(handles);
end;

set(handles.statusText,'String','Ping!........' );
pause(0.1); % Pauses for 0.1 second

if( serialAsyncWrite(SERIALLINK,'o') )   % p = ping kommand
    set(handles.statusText,'String','Unable to send serial data' );
else
    [error,data] = serialSyncRead(SERIALLINK,1); % 1 byte response expected
    if( error )
        set(handles.statusText,'String','No response from robot. Timeout.'
);
    else

        if( data == 'k') % k = ping response
            set(handles.statusText,'String','Robot responding' );
        else
            disp(data);
            set(handles.statusText,'String','Robot responding. But wrong
response' );
        end;
    end;
end;


% --- Executes when user attempts to close_pushbutton figure1.
function figure1_CloseRequestFcn(hObject, eventdata, handles)
% hObject    handle to figure1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global SERIALLINK;

if(SERIALLINK==0)
    clear SERIALLINK;
else
    try
        stopasync(SERIALLINK);
        fclose(SERIALLINK);
        delete SERIALLINK;
        clear SERIALLINK;
    catch

    end;
end;

delete(hObject);

% --- Executes on slider movement.
function angleSlider_Callback(hObject, eventdata, handles)
% hObject    handle to AngleSlider (see GCBO)
```

```
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'Value') returns position of slider
%        get(hObject,'Min') and get(hObject,'Max') to determine range of
slider

sliderVal = floor( get(hObject,'Value') );
set(handles.angleText,'String', sliderVal ); % angleText is the textbox
next to the SetRadarAngle push button

% --- Executes during object creation, after setting all properties.
function angleSlider_CreateFcn(hObject, eventdata, handles)
% hObject    handle to AngleSlider (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: slider controls usually have a light gray background.
if isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end


% --- Executes on button press in radarButton.
function radarButton_Callback(hObject, eventdata, handles)
% hObject    handle to radarButton (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global SERIALLINK;

if( SERIALLINK == 0 )
    SERIALLINK = connectTo(handles);
end;

tmpAngle = floor( str2double( get(handles.angleText,'String') ) );
rAngle = char( (90+tmpAngle) );
if( serialAsyncWrite( SERIALLINK,strcat('r',rAngle) ) )   % rx = set radar,
x=angle
     set(handles.statusText,'String','Unable to send serial data' );
else
    [error,data]=serialSyncRead( SERIALLINK,1);
    if error || data~='a'
        set(handles.statusText,'String','Radar angle not set' );
    else
        set(handles.statusText,'String','Radar set' );
    end
end;


% --- Executes during object creation, after setting all properties.
function angleText_CreateFcn(hObject, eventdata, handles)
% hObject    handle to angleText (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
```

```
end

% --- Executes on button press in resetRobtPoseButton.
function resetRobtPoseButton_Callback(hObject, eventdata, handles)
% hObject    handle to resetRobtPoseButton (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global SERIALLINK;

if( SERIALLINK == 0 )
    SERIALLINK = connectTo(handles);
end;

clearPose(SERIALLINK,handles);

roboposeButton_Callback(handles.roboposeButton, eventdata, handles);


% --- Executes on button press in startButton.
function startButton_Callback(hObject, eventdata, handles)
% hObject    handle to startButton (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global RUNNING;
global SERIALLINK;
global ROBOTPOSE;


if(RUNNING==1)
    set(handles.statusText,'String','Allready started');
    return;
end
if RUNNING==2, % user have pushed pausebutton, resume navigation..
    RUNNING=1;
    set(handles.statusText,'String','Started');
    return
end

if( SERIALLINK == 0 )
    SERIALLINK = connectTo(handles);

    set(handles.statusText,'String','Started');
end;
RUNNING=1;

if strcmp(SERIALLINK,'simulator'),
    % setting default simulation properties
    ROBOTPOSE=[0 0 0];

    handles.simParams.maze=SimMapMenu_Callback(handles.SimMapMenu,
eventdata, handles);
    handles.poseError=checkbox_posErr_Callback(handles.checkbox_posErr,
eventdata, handles);
    handles.irError=checkbox_irErr_Callback(handles.checkbox_posErr,
eventdata, handles);

    set(handles.statusText,'String','Simulation running..');
end

% sensor properties
```

```
handles.maxPerceptRadius=str2double(get(handles.maxPerceptRadius,'String'))
;
handles.angularRes=str2double(get(handles.angularRes,'String'));

handles.goHometext = str2double(get(handles.goHomeText,'String'));

% scan while driving
handles.driveScan=get(handles.checkbox_driveScan,'Value');
%if( get(handles.checkbox_driveScan,'Value')==1  )
%    handles.driveScan=1;
%end

% Start mapping from 0,0,0
resetRobtPoseButton_Callback(handles.resetRobtPoseButton, eventdata,
handles);

% testing...
if strcmp(SERIALLINK,'simulator'),
    setRobotPose(SERIALLINK,handles,[0 0 0]);
end

%make figure to plot image in
if strcmp(SERIALLINK,'NXT_kamera'),
    handles.kamerafigur=figure();
end


%% Beacon navigation
if( get(handles.togglebutton_beaconNav,'Value')==1  )
    if strcmp(handles.beaconParam.changed,'false'), % parameters not
manipulated, set them to standard values
        beaconParam.beaconthreshdist = 0.02;    % Jump distance for
segmentation in [m]
        beaconParam.alpha = 0.75;   % Sannsynligheten for at to punkter er
ett punkt
    elseif strcmp(handles.beaconParam.changed,'true'),
        beaconParam=handles.beaconParam;
    end
    %robot parameters
    %beaconParamParam.robot.x =
[str2double(get(handles.edit20,'String'));str2double(get(handles.edit19,'St
ring'));str2double(get(handles.edit21,'String'))];
    %beaconParam.robot.xsensor =
[str2double(get(handles.edit24,'String'));str2double(get(handles.edit22,'St
ring'));str2double(get(handles.edit23,'String'))];
    beaconParam.robot.formtype = get(handles.robotType,'Value')-1;

    %plotting parameters
    beaconParam.displayGlobal = get(handles.checkbox_globalMap,'Value');
    beaconParam.displayLocal = get(handles.checkbox_localMap,'Value');
    beaconParam.displayMatch = get(handles.checkbox_matching,'Value');

    if strcmp(SERIALLINK,'simulator'),
        beaconParam.simulation = 1;
        %simParams.init=1;

handles.simParams.startSimX=str2double(get(handles.startSimX,'String'));

handles.simParams.startSimY=str2double(get(handles.startSimY,'String'));
```

```
handles.simParams.widthBetweenWalls=str2double(get(handles.widthOfWalls,'St
ring'));
        %handles.simParams.irError=-2;
        handles.simParams.maxPerceptRadius=handles.maxPerceptRadius;
        handles.simParams.angularRes=handles.angularRes;

    else
        beaconParam.simulation = 0;
        handles.simParams=[];
    end

    beaconSLAM(SERIALLINK,handles,beaconParam);

%% lineSLAM
elseif( get(handles.togglebutton_lineNav,'Value')==1 ) % linenavigation
    if strcmp(handles.lineParam.changed,'false'), % parameters not
manipulated, set them to standard values
        lineParam.windowsize = 9;
        lineParam.threshfidel= 0.05;
        lineParam.fusealpha= 0.95;
        lineParam.minlength= 0.15;
        lineParam.compensa= 0.017453;
        lineParam.compensr= 0.01;
        lineParam.cyclic= 1;
        lineParam.alpha= 0.85;
        lineParam.sensor.stdrho = 0.02;
    elseif strcmp(handles.lineParam.changed,'true'),
        lineParam=handles.lineParam;
    end
    %lineParam.robot.x =
[str2double(get(handles.edit20,'String'));str2double(get(handles.edit19,'St
ring'));str2double(get(handles.edit21,'String'))];
    %lineParam.robot.xsensor =
[str2double(get(handles.edit24,'String'));str2double(get(handles.edit22,'St
ring'));str2double(get(handles.edit23,'String'))];
    lineParam.robot.formtype = get(handles.robotType,'Value')-1;

    lineParam.displayGlobal = get(handles.checkbox_globalMap,'Value');
    lineParam.displayLocal = get(handles.checkbox_localMap,'Value');
    lineParam.displayMatch = get(handles.checkbox_matching,'Value');

    if strcmp(SERIALLINK,'simulator'),
        % simulation properties
        lineParam.simulation = 1;

handles.simParams.startSimX=str2double(get(handles.startSimX,'String'));

handles.simParams.startSimY=str2double(get(handles.startSimY,'String'));

handles.simParams.widthBetweenWalls=str2double(get(handles.widthOfWalls,'St
ring'));
%        handles.simParams.irError=-2;
        handles.simParams.maxPerceptRadius=handles.maxPerceptRadius;
        handles.simParams.angularRes=handles.angularRes;
    else
        lineParam.simulation = 0;
        handles.simParams=[];
    end

    %navigation type
```

```
    if( get(handles.lineNavButton1,'Value')==1)
        lineParam.navigation=1;
    end
    lineSLAM(SERIALLINK,handles,lineParam);


%% lineBeaconSLAM
elseif( get(handles.togglebutton_lineBeaconNav,'Value')==1 ) % line/beacon
navigation
    if strcmp(handles.lineParam.changed,'false'), % parameters not
manipulated, set them to standard values
        lineParam.windowsize = 9; % Size of sliding window. N første
punktene i datasettet.11
        lineParam.threshfidel= 0.05; % Hvert punkt må være næremere linja
enn denne verdien.
        lineParam.fusealpha= 0.95;  % Sammensmeltning av linjer gjøres når
de med denne sikkerheten er én linje
        lineParam.minlength= 0.15; % minimum length a segment must have to
be accepted
        lineParam.compensa= 0.017453;  % pi/180
        lineParam.compensr= 0.01;
        lineParam.cyclic= 1;        % 0: non-cyclic or 1: cyclic
        lineParam.alpha= 0.85;  % Sannsynligheten for at to linjer er den
samme
        lineParam.sensor.stdrho = 0.02; % Beskriver en konstant radiell
usikkerhet i avstandsmålingene
    elseif strcmp(handles.lineParam.changed,'true'),
        lineParam=handles.lineParam;
    end
    %lineParam.robot.x =
[str2double(get(handles.edit20,'String'));str2double(get(handles.edit19,'St
ring'));str2double(get(handles.edit21,'String'))];
    %lineParam.robot.xsensor =
[str2double(get(handles.edit24,'String'));str2double(get(handles.edit22,'St
ring'));str2double(get(handles.edit23,'String'))];
    lineParam.robot.formtype = get(handles.robotType,'Value')-1;

    lineParam.displayGlobal = get(handles.checkbox_globalMap,'Value');
    lineParam.displayLocal = get(handles.checkbox_localMap,'Value');
    lineParam.displayMatch = get(handles.checkbox_matching,'Value');

    lineParam.goHome = get(handles.checkbox_goHome,'Value');

    if strcmp(SERIALLINK,'simulator'),
        % simulation properties
        lineParam.simulation = 1;

handles.simParams.startSimX=str2double(get(handles.startSimX,'String'));

handles.simParams.startSimY=str2double(get(handles.startSimY,'String'));

handles.simParams.widthBetweenWalls=str2double(get(handles.widthOfWalls,'St
ring'));
%        handles.simParams.irError=-2;
        handles.simParams.maxPerceptRadius=handles.maxPerceptRadius;
        handles.simParams.angularRes=handles.angularRes;
    else
        lineParam.simulation = 0;
        handles.simParams=[];
    end
```

```
    %navigation type
    if( get(handles.lineNavButton1,'Value')==1)
        lineParam.navigation=1;
    end


%%%-- integration of lineSLAM and beaconSLAM --%%%

    if strcmp(handles.beaconParam.changed,'false'), % parameters not
manipulated, set them to standard values
        beaconParam.beaconthreshdist = 0.02;
        beaconParam.alpha = 0.75;
    elseif strcmp(handles.beaconParam.changed,'true'),
        beaconParam=handles.beaconParam;
    end
    %robot parameters
    %beaconParamParam.robot.x =
[str2double(get(handles.edit20,'String'));str2double(get(handles.edit19,'St
ring'));str2double(get(handles.edit21,'String'))];
    %beaconParam.robot.xsensor =
[str2double(get(handles.edit24,'String'));str2double(get(handles.edit22,'St
ring'));str2double(get(handles.edit23,'String'))];
    beaconParam.robot.formtype = get(handles.robotType,'Value')-1;

    %plotting parameters
    beaconParam.displayGlobal = get(handles.checkbox_globalMap,'Value');
    beaconParam.displayLocal = get(handles.checkbox_localMap,'Value');
    beaconParam.displayMatch = get(handles.checkbox_matching,'Value');

    if ~strcmp(SERIALLINK,'simulator'),
        beaconParam.simulation = 0;
        handles.simParams=[];
    end


    lineBeaconSLAM(SERIALLINK,handles,lineParam,beaconParam);

else
    disp('no navigation method chosen...LegoGui.m')

end;


set(handles.statusText,'String','Stopped');

% --- Executes on button press in stopButton.
function stopButton_Callback(hObject, eventdata, handles)
% hObject    handle to stopButton (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global RUNNING;
global SERIALLINK;

if ((RUNNING == 1) || (RUNNING ==2)) && ~strcmp(SERIALLINK,'simulator'),
    RUNNING = 0;
    set(handles.statusText,'String','Stopping.....Please wait');

elseif ((RUNNING == 1) || (RUNNING ==2)) && strcmp(SERIALLINK,'simulator'),
    RUNNING = 0;
    set(handles.statusText,'String','Simulation stopped by user');
else
```

```
    set(handles.statusText,'String','Not started...');
end;



% --- Executes on button press in roboposeButton.
function roboposeButton_Callback(hObject, eventdata, handles)
% hObject    handle to roboposeButton (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global SERIALLINK;

if( SERIALLINK == 0 )
    SERIALLINK = connectTo(handles);
end;

[poses,error]=getRobotPose( SERIALLINK,handles);
if( error )
    set(handles.statusText,'String','Error when prosessing getRobotPose' );
else
    set(handles.xPosText,'String',num2str(poses(1)) );
    set(handles.yPosText,'String',num2str(poses(2)) );
    set(handles.tPosText,'String',num2str(poses(3)*180/pi) );
end;


function xPosText_Callback(hObject, eventdata, handles)
% hObject    handle to xPosText (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of xPosText as text
%        str2double(get(hObject,'String')) returns contents of xPosText as
a double


% --- Executes during object creation, after setting all properties.
function xPosText_CreateFcn(hObject, eventdata, handles)
% hObject    handle to xPosText (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


function tPosText_Callback(hObject, eventdata, handles)
% hObject    handle to tPosText (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of tPosText as text
%        str2double(get(hObject,'String')) returns contents of tPosText as
a double
```

```
% --- Executes during object creation, after setting all properties.
function tPosText_CreateFcn(hObject, eventdata, handles)
% hObject    handle to tPosText (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
%if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
%    set(hObject,'BackgroundColor','white');
%end
% Hint: slider controls usually have a light gray background.



function yPosText_Callback(hObject, eventdata, handles)
% hObject    handle to yPosText (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of yPosText as text
%        str2double(get(hObject,'String')) returns contents of yPosText as
a double

% --- Executes during object creation, after setting all properties.
function yPosText_CreateFcn(hObject, eventdata, handles)
% hObject    handle to yPosText (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
%if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
%    set(hObject,'BackgroundColor','white');
%end



% --- Executes on button press in robotForwardButton.
function robotForwardButton_Callback(hObject, eventdata, handles)
% hObject    handle to robotForwardButton (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global SERIALLINK;

if( SERIALLINK == 0 )
    SERIALLINK = connectTo(handles);
end;

if( serialAsyncWrite(SERIALLINK,'w') )
    set(handles.statusText,'String','Unable to send serial data' );


end;


% --- Executes on button press in robotStopButton.
function robotStopButton_Callback(hObject, eventdata, handles)
% hObject    handle to robotStopButton (see GCBO)
```

```
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global SERIALLINK;

if( SERIALLINK == 0 )
    SERIALLINK = connectTo(handles);
end;

if( serialAsyncWrite(SERIALLINK,'s') )
    set(handles.statusText,'String','Unable to send serial data' );
else
    if strcmp(SERIALLINK,'NXT')
        [myError,data]=serialSyncRead2(SERIALLINK);
    elseif SERIALLINK.bytesAvailable,
        [myError,data]=serialSyncRead2(SERIALLINK) % fetch all data
received from robot
    end
end;


% --- Executes on button press in robotLeftButton.
function robotLeftButton_Callback(hObject, eventdata, handles)
% hObject    handle to robotLeftButton (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global SERIALLINK;

if( SERIALLINK == 0 )
    SERIALLINK = connectTo(handles);
end;

if( serialAsyncWrite(SERIALLINK,'a') )
    set(handles.statusText,'String','Unable to send serial data' );
end;


% --- Executes on button press in robotRightButton.
function robotRightButton_Callback(hObject, eventdata, handles)
% hObject    handle to robotRightButton (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global SERIALLINK;

if( SERIALLINK == 0 )
    SERIALLINK = connectTo(handles);
end;

if( serialAsyncWrite(SERIALLINK,'d') )
    set(handles.statusText,'String','Unable to send serial data' );
end;


% --- Executes on button press in pushbutton_robotBackward.
function pushbutton_robotBackward_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton_robotBackward (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global SERIALLINK;

if( SERIALLINK == 0 )
    SERIALLINK = connectTo(handles);
```

```
end;

if( serialAsyncWrite(SERIALLINK,'x') )
    set(handles.statusText,'String','Unable to send serial data' );
end;



function statusText_Callback(hObject, eventdata, handles)
% hObject    handle to statusText (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of statusText as text
%        str2double(get(hObject,'String')) returns contents of statusText
as a double


% --- Executes during object creation, after setting all properties.
function statusText_CreateFcn(hObject, eventdata, handles)
% hObject    handle to statusText (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% --------------------------------------------------------------------
function menuFile_Callback(hObject, eventdata, handles)
% hObject    handle to menuFile (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)


% --- Executes on selection change in robotType.
function robotType_Callback(hObject, eventdata, handles)
% hObject    handle to robotType (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: contents = get(hObject,'String') returns robotType contents as
cell array
%        contents{get(hObject,'Value')} returns selected item from
robotType


% --- Executes during object creation, after setting all properties.
function robotType_CreateFcn(hObject, eventdata, handles)
% hObject    handle to robotType (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: popupmenu controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
```

```
    set(hObject,'BackgroundColor','white');
end


function linesWindowsize_Callback(hObject, eventdata, handles)
% hObject    handle to linesWindowsize (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of linesWindowsize as text
%        str2double(get(hObject,'String')) returns contents of
linesWindowsize as a double


% --- Executes during object creation, after setting all properties.
function linesWindowsize_CreateFcn(hObject, eventdata, handles)
% hObject    handle to linesWindowsize (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


function linesCompactness_Callback(hObject, eventdata, handles)
% hObject    handle to linesCompactness (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of linesCompactness as text
%        str2double(get(hObject,'String')) returns contents of
linesCompactness as a double


% --- Executes during object creation, after setting all properties.
function linesCompactness_CreateFcn(hObject, eventdata, handles)
% hObject    handle to linesCompactness (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


function linesFusion_Callback(hObject, eventdata, handles)
% hObject    handle to linesFusion (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of linesFusion as text
%        str2double(get(hObject,'String')) returns contents of linesFusion
as a double
```

```
% --- Executes during object creation, after setting all properties.
function linesFusion_CreateFcn(hObject, eventdata, handles)
% hObject    handle to linesFusion (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


function linesMinLenght_Callback(hObject, eventdata, handles)
% hObject    handle to linesMinLenght (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of linesMinLenght as text
%        str2double(get(hObject,'String')) returns contents of
linesMinLenght as a double


% --- Executes during object creation, after setting all properties.
function linesMinLenght_CreateFcn(hObject, eventdata, handles)
% hObject    handle to linesMinLenght (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


function linesHeura_Callback(hObject, eventdata, handles)
% hObject    handle to linesHeura (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of linesHeura as text
%        str2double(get(hObject,'String')) returns contents of linesHeura
as a double


% --- Executes during object creation, after setting all properties.
function linesHeura_CreateFcn(hObject, eventdata, handles)
% hObject    handle to linesHeura (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
```

```
function linesHeurr_Callback(hObject, eventdata, handles)
% hObject    handle to linesHeurr (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of linesHeurr as text
%        str2double(get(hObject,'String')) returns contents of linesHeurr
as a double


% --- Executes during object creation, after setting all properties.
function linesHeurr_CreateFcn(hObject, eventdata, handles)
% hObject    handle to linesHeurr (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function linesCyclic_Callback(hObject, eventdata, handles)
% hObject    handle to linesCyclic (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of linesCyclic as text
%        str2double(get(hObject,'String')) returns contents of linesCyclic
as a double


% --- Executes during object creation, after setting all properties.
function linesCyclic_CreateFcn(hObject, eventdata, handles)
% hObject    handle to linesCyclic (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function linesAlpha_Callback(hObject, eventdata, handles)
% hObject    handle to linesAlpha (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of linesAlpha as text
%        str2double(get(hObject,'String')) returns contents of linesAlpha
as a double
```

```
% --- Executes during object creation, after setting all properties.
function linesAlpha_CreateFcn(hObject, eventdata, handles)
% hObject    handle to linesAlpha (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% --- Executes on button press in checkbox_globalMap.
function checkbox_globalMap_Callback(hObject, eventdata, handles)
% hObject    handle to checkbox_globalMap (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of checkbox_globalMap


% --- Executes on button press in checkbox2.
function checkbox_localMap_Callback(hObject, eventdata, handles)
% hObject    handle to checkbox2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of checkbox2


% --- Executes on button press in checkbox_matching.
function checkbox_matching_Callback(hObject, eventdata, handles)
% hObject    handle to checkbox_matching (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of checkbox_matching

% --- Executes on button press in togglebutton_lineNav.
function togglebutton_lineNav_Callback(hObject, eventdata, handles)
% hObject    handle to togglebutton_lineNav (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of togglebutton_lineNav

% --- Executes on button press in togglebutton_beaconNav.
function togglebutton_beaconNav_Callback(hObject, eventdata, handles)
% hObject    handle to togglebutton_beaconNav (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of togglebutton_beaconNav

% --- Executes on button press in togglebutton_lineNav.
function togglebutton_lineBeaconNav_Callback(hObject, eventdata, handles)
% hObject    handle to togglebutton_lineNav (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
```

```
% Hint: get(hObject,'Value') returns toggle state of togglebutton_lineNav


function edit_heading_Callback(hObject, eventdata, handles)
% hObject    handle to edit_heading (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit_heading as text
%        str2double(get(hObject,'String')) returns contents of edit_heading
as a double


% --- Executes during object creation, after setting all properties.
function edit_heading_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit_heading (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


function edit_distance_Callback(hObject, eventdata, handles)
% hObject    handle to edit_distance (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit_distance as text
%        str2double(get(hObject,'String')) returns contents of
edit_distance as a double

% --- Executes during object creation, after setting all properties.
function edit_distance_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit_distance (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% --- Executes on button press in pushbutton_goto_target.
function pushbutton_goto_target_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton_goto_target (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global SERIALLINK;

if( SERIALLINK == 0 )
    SERIALLINK = connectTo(handles);
end;
```

```
goToX=str2double( get(handles.edit_goToX,'String') );
goToY=str2double( get(handles.edit_goToY,'String') );

heading=str2double( get(handles.edit_heading,'String') );
dist=str2double( get(handles.edit_distance,'String') );


if goToX==0 && goToY==0,
    % use heading + distance to go to target destination
    tmpHead = floor( heading/2 ); % Floor rounds the element to nearest
integer towards minus infinity
    rAngle = char( tmpHead );

    tmpDist = floor( dist );
    rDist = char( tmpDist );
    set(handles.edit_distance,'String',0);%% clear the box
%    set(handles.slider_target_distance,'Value',0);%% clear the slider

    error=setRobotTarget2(SERIALLINK,rAngle,rDist,handles);

    if( error )
        set(handles.statusText,'String','Unable to send serial data. Target
not sat' );
    else
        set(handles.statusText,'String',strcat('Target heading sat to: ',
num2str(tmpHead*2),'. Distance to:',num2str(tmpDist) ) );
    end

else
    % use cartesian coordinates to go to target
    x=goToX*10; % x in [mm]
    y=goToY*10; % y in [mm]

    error=goToPose2(SERIALLINK,x,y,handles);
    if error,
        set(handles.statusText,'String','Unable to send serial data. Target
not sat' );
    else
        set(handles.statusText,'String',strcat('Target position sat:
X=',num2str(goToX),' cm, Y=',num2str(goToY),' cm.') );
    end
end


% --- Executes on slider movement.
function slider_target_heading_Callback(hObject, eventdata, handles)
% hObject    handle to slider_target_heading (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'Value') returns position of slider
%        get(hObject,'Min') and get(hObject,'Max') to determine range of
slider
sliderVal = floor( get(hObject,'Value') );
set(handles.edit_heading,'String', sliderVal );


% --- Executes during object creation, after setting all properties.
function slider_target_heading_CreateFcn(hObject, eventdata, handles)
% hObject    handle to slider_target_heading (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
```

```
% handles    empty - handles not created until after all CreateFcns called

% Hint: slider controls usually have a light gray background.
if isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end


% --- Executes on slider movement.
function slider_target_distance_Callback(hObject, eventdata, handles)
% hObject    handle to slider_target_distance (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'Value') returns position of slider
%        get(hObject,'Min') and get(hObject,'Max') to determine range of
slider
sliderVal = floor( get(hObject,'Value') );
set(handles.edit_distance,'String', sliderVal );


% --- Executes during object creation, after setting all properties.
function slider_target_distance_CreateFcn(hObject, eventdata, handles)
% hObject    handle to slider_target_distance (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: slider controls usually have a light gray background.
if isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor',[.9 .9 .9]);

end

% --- Executes on button press in lineNavButton1.
function lineNavButton1_Callback(hObject, eventdata, handles)
% hObject    handle to lineNavButton1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of lineNavButton1




% --- Executes on button press in checkbox_posErr.
function poseError=checkbox_posErr_Callback(hObject, eventdata, handles)
% hObject    handle to checkbox_posErr (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of checkbox_posErr
global REALPOSE;

if (get(hObject,'Value')),
    poseError=1;
    REALPOSE(1)=0;
    REALPOSE(2)=0;
    disp('position error ON')
else
```

```
    poseError=0;
    disp('position error OFF')

end


% --- Executes on button press in checkbox_irErr.
function irError=checkbox_irErr_Callback(hObject, eventdata, handles)
% hObject    handle to checkbox_irErr (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of checkbox_irErr
if (get(hObject,'Value')),
    irError=1;
    disp('IR sensor error ON');
else
    irError=0;
    disp('IR sensor error OFF');

end

% --- Executes on selection change in SimMapMenu.
function maze=SimMapMenu_Callback(hObject, eventdata, handles)
% hObject    handle to SimMapMenu (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: contents = get(hObject,'String') returns SimMapMenu contents as
cell array
%        contents{get(hObject,'Value')} returns selected item from
SimMapMenu


contents = get(hObject,'String');
chosenMap=contents{get(hObject,'Value')};

handles.simParams.changed='true';
switch chosenMap
    case 'Map 1'
        handles.simParams.maze=1;
    case 'Map 2'
        handles.simParams.maze=2;
    case 'Map 3'
        handles.simParams.maze=3;
    case 'Map 4'
        handles.simParams.maze=4;
    case 'Map 5'
        handles.simParams.maze=5;
    case 'Map 6'
        handles.simParams.maze=9;
    case '(Empty square)'
        handles.simParams.maze=6;

end
maze=handles.simParams.maze;


% --- Executes during object creation, after setting all properties.
function SimMapMenu_CreateFcn(hObject, eventdata, handles)
% hObject    handle to SimMapMenu (see GCBO)
```

```
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: popupmenu controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% --- Executes on button press in pauseButton.
function pauseButton_Callback(hObject, eventdata, handles)
% hObject    handle to stopButton (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global RUNNING;

if(RUNNING == 1)
    RUNNING = 2;
    set(handles.statusText,'String','Pause.....');
elseif (RUNNING ==2),
    RUNNING = 1;
    set(handles.statusText,'String','Started');

else
    set(handles.statusText,'String','Not started ...');
end;

function widthOfWalls_Callback(hObject, eventdata, handles)
% hObject    handle to linesWindowsize (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of linesWindowsize as text
%        str2double(get(hObject,'String')) returns contents of
linesWindowsize as a double


% --- Executes during object creation, after setting all properties.
function widthOfWalls_CreateFcn(hObject, eventdata, handles)
% hObject    handle to linesWindowsize (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


function startSimX_Callback(hObject, eventdata, handles)
% hObject    handle to linesWindowsize (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of linesWindowsize as text
%        str2double(get(hObject,'String')) returns contents of
linesWindowsize as a double
```

```
% --- Executes during object creation, after setting all properties.
function startSimX_CreateFcn(hObject, eventdata, handles)
% hObject    handle to linesWindowsize (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


function startSimY_Callback(hObject, eventdata, handles)
% hObject    handle to linesWindowsize (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of linesWindowsize as text
%        str2double(get(hObject,'String')) returns contents of
linesWindowsize as a double


% --- Executes during object creation, after setting all properties.
function startSimY_CreateFcn(hObject, eventdata, handles)
% hObject    handle to linesWindowsize (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% --------------------------------------------------------------------
function set_breakpoint_Callback(hObject, eventdata, handles)
% hObject    handle to set_breakpoint (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)


% --------------------------------------------------------------------
function parameters_Callback(hObject, eventdata, handles)
% hObject    handle to parameters (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)


%pos_size = get(handles.figure1,'Position');

if strcmp(handles.lineParam.changed,'false') &&
strcmp(handles.beaconParam.changed,'false'),
    OutputParam2 = params2('Title','Navigation parameters');
    if length(OutputParam2)>1,
        handles.lineParam=OutputParam2{1};
        handles.beaconParam=OutputParam2(Free2Move);
```

```
    end
else
    allParam=handles.lineParam;
    allParam.beaconThresh=handles.beaconParam.beaconthreshdist;
    allParam.beaconAlpha=handles.beaconParam.alpha;
    OutputParam2 = params2('Title','Navigation
parameters','current_data',allParam);
    if length(OutputParam2)>1,
        handles.lineParam=OutputParam2{1};
        handles.beaconParam=OutputParam2(Free2Move);
    end
end

% Update handles structure
guidata(hObject, handles);


% --- Executes on button press in close_pushbutton.
function close_pushbutton_Callback(hObject, eventdata, handles)
% hObject    handle to close_pushbutton (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get the current position of the GUI from the handles structure
% to pass to the modal dialog.
pos_size = get(handles.figure1,'Position');
% Call modaldlg with the argument 'Position'.
user_response = Close_Gui('Title','Confirm Close');
switch user_response
case {'No'}
    % take no action
case 'Yes'
    % Prepare to close GUI application window
    %
    delete(handles.figure1)
end


% --------------------------------------------------------------------
function close_Callback(hObject, eventdata, handles)
% hObject    handle to close (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
close_pushbutton_Callback(hObject, eventdata, handles);


function lineNavButton1_ButtonDownFcn(hObject, eventdata, handles)
% hObject    handle to lineNavButton1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)


function maxPerceptRadius_Callback(hObject, eventdata, handles)
% hObject    handle to maxPerceptRadius (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of maxPerceptRadius as text
%        str2double(get(hObject,'String')) returns contents of
%        maxPerceptRadius as a double
```

```
% --- Executes during object creation, after setting all properties.
function maxPerceptRadius_CreateFcn(hObject, eventdata, handles)
% hObject    handle to maxPerceptRadius (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


function angularRes_Callback(hObject, eventdata, handles)
% hObject    handle to angularRes (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of angularRes as text
%        str2double(get(hObject,'String')) returns contents of angularRes
as a double


% --- Executes during object creation, after setting all properties.
function angularRes_CreateFcn(hObject, eventdata, handles)
% hObject    handle to angularRes (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end




% --------------------------------------------------------------------
function edit_Callback(hObject, eventdata, handles)
% hObject    handle to edit (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)



% --- Executes on button press in checkbox_driveScan.
function checkbox_driveScan_Callback(hObject, eventdata, handles)
% hObject    handle to checkbox_driveScan (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of checkbox_driveScan
```

```
% --- Executes on button press in pushbutton_fullScan.
function pushbutton_fullScan_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton_fullScan (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global SERIALLINK;

if( SERIALLINK == 0 )
    SERIALLINK = connectTo(handles);
end;

[error,data]=fullScan(SERIALLINK,handles);

figure(66);clf;hold on;axis equal;
plot(0,0,'ko')
%plot(data(:,2), data(:,3),'bx');


IRdata=[];
NS=4;
for sens=1:1:NS,

     tmpData=data(data(:,1)==sens,:); % Missing data from one sensor!
     IRdata=vertcat(IRdata,tmpData); % IRData = the data sorted by the
sensors, first sensor 1, then sensor 2 and at last sensor 3 with its data
end
%for i=1:size(IRdata,1),
    plot(IRdata(:,2), IRdata(:,3),'rx');

%end
```

```
% --- Executes on button press in pushbutton_setRobotPose.
function pushbutton_setRobotPose_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton_setRobotPose (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global SERIALLINK;
global ROBOTPOSE;

if( SERIALLINK == 0 )
    SERIALLINK = connectTo(handles);
end;


setX = ( str2double( get(handles.edit_setPoseX,'String') ) );
%setXtoRob = ( setX ); % x in mm
setY = ( str2double( get(handles.edit_setPoseY,'String') ) );
%setYtoRob = ( setY ); % y in mm
setTheta = ( str2double( get(handles.edit_setPoseTheta,'String') ) );
%setThetatoRob = ( setTheta*pi/180 * 1/1); % theta in rad/1000 (0-2000*pi)

if strcmp(SERIALLINK,'simulator'),
    ROBOTPOSE=[setX/10 setY/10 setTheta]; % to simulator: x,y in [cm],
theta in [deg]
    % also display result in Gui
```

```
        roboposeButton_Callback(handles.roboposeButton, eventdata, handles);

    else
        pose=[setX setY setTheta ];
        error=setRobotPose(SERIALLINK,handles, pose);

        if( error )   %
            set(handles.statusText,'String','Unable to send serial data' );
        end

    end



    function edit_setPoseX_Callback(hObject, eventdata, handles)
    % hObject    handle to edit_setPoseX (see GCBO)
    % eventdata  reserved - to be defined in a future version of MATLAB
    % handles    structure with handles and user data (see GUIDATA)

    % Hints: get(hObject,'String') returns contents of edit_setPoseX as text
    %        str2double(get(hObject,'String')) returns contents of
    edit_setPoseX as a double


    % --- Executes during object creation, after setting all properties.
    function edit_setPoseX_CreateFcn(hObject, eventdata, handles)
    % hObject    handle to edit_setPoseX (see GCBO)
    % eventdata  reserved - to be defined in a future version of MATLAB
    % handles    empty - handles not created until after all CreateFcns called

    % Hint: edit controls usually have a white background on Windows.
    %       See ISPC and COMPUTER.
    if ispc && isequal(get(hObject,'BackgroundColor'),
    get(0,'defaultUicontrolBackgroundColor'))
        set(hObject,'BackgroundColor','white');
    end



    function edit_setPoseY_Callback(hObject, eventdata, handles)
    % hObject    handle to edit_setPoseY (see GCBO)
    % eventdata  reserved - to be defined in a future version of MATLAB
    % handles    structure with handles and user data (see GUIDATA)

    % Hints: get(hObject,'String') returns contents of edit_setPoseY as text
    %        str2double(get(hObject,'String')) returns contents of
    edit_setPoseY as a double


    % --- Executes during object creation, after setting all properties.
    function edit_setPoseY_CreateFcn(hObject, eventdata, handles)
    % hObject    handle to edit_setPoseY (see GCBO)
    % eventdata  reserved - to be defined in a future version of MATLAB
    % handles    empty - handles not created until after all CreateFcns called

    % Hint: edit controls usually have a white background on Windows.
    %       See ISPC and COMPUTER.
    if ispc && isequal(get(hObject,'BackgroundColor'),
    get(0,'defaultUicontrolBackgroundColor'))
        set(hObject,'BackgroundColor','white');
    end
```

```
function edit_setPoseTheta_Callback(hObject, eventdata, handles)
% hObject    handle to edit_setPoseTheta (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit_setPoseTheta as
text
%        str2double(get(hObject,'String')) returns contents of
edit_setPoseTheta as a double


% --- Executes during object creation, after setting all properties.
function edit_setPoseTheta_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit_setPoseTheta (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end




function edit_goToX_Callback(hObject, eventdata, handles)
% hObject    handle to edit_goToX (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit_goToX as text
%        str2double(get(hObject,'String')) returns contents of edit_goToX
as a double


% --- Executes during object creation, after setting all properties.
function edit_goToX_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit_goToX (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function edit_goToY_Callback(hObject, eventdata, handles)
% hObject    handle to edit_goToY (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
```

```
% Hints: get(hObject,'String') returns contents of edit_goToY as text
%        str2double(get(hObject,'String')) returns contents of edit_goToY
as a double


% --- Executes during object creation, after setting all properties.
function edit_goToY_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit_goToY (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function edit42_Callback(hObject, eventdata, handles)
% hObject    handle to tPosText (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of tPosText as text
%        str2double(get(hObject,'String')) returns contents of tPosText as
a double


% --- Executes during object creation, after setting all properties.
function edit42_CreateFcn(hObject, eventdata, handles)
% hObject    handle to tPosText (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end




% --- Executes on button press in pushbutton_clawOpen.
function pushbutton_clawOpen_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton_clawOpen (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

global SERIALLINK;

if( SERIALLINK == 0 )
    SERIALLINK = connectTo(handles);
end;

if( serialAsyncWrite(SERIALLINK,'y') )
    set(handles.statusText,'String','Unable to send serial data' );
end;
```

```
% --- Executes on button press in pushbutton_clawStop.
function pushbutton_clawStop_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton_clawStop (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global SERIALLINK;

if( SERIALLINK == 0 )
    SERIALLINK = connectTo(handles);
end;

if( serialAsyncWrite(SERIALLINK,'i') )
    set(handles.statusText,'String','Unable to send serial data' );
end;



function edit_realposeX_Callback(hObject, eventdata, handles)
% hObject    handle to edit_realposeX (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit_realposeX as text
%        str2double(get(hObject,'String')) returns contents of
edit_realposeX as a double


% --- Executes during object creation, after setting all properties.
function edit_realposeX_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit_realposeX (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function edit_realposeY_Callback(hObject, eventdata, handles)
% hObject    handle to edit_realposeY (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit_realposeY as text
%        str2double(get(hObject,'String')) returns contents of
edit_realposeY as a double


% --- Executes during object creation, after setting all properties.
function edit_realposeY_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit_realposeY (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called
```

```
% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function edit_realposeTheta_Callback(hObject, eventdata, handles)
% hObject    handle to edit_realposeTheta (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit_realposeTheta as
text
%        str2double(get(hObject,'String')) returns contents of
edit_realposeTheta as a double


% --- Executes during object creation, after setting all properties.
function edit_realposeTheta_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit_realposeTheta (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end




% --- Executes on button press in calibrate_sensors.
function calibrate_sensors_Callback(hObject, eventdata, handles)
% hObject    handle to calibrate_sensors (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

global SERIALLINK;
global table;

if( SERIALLINK == 0 )
    SERIALLINK = connectTo(handles);
end;

% If the calibration is finished, draw a curve for each sensor that shows
% the analog voltage vs distance in cm
if (get(handles.finishedCalibrating,'Value') == 1)
    % Remove analog values that are 0
     for j=1:1:length(table)
         if(table(j,3) == 0)
             table(j,:) = [];
         end
     end
    table1 = [];
    table2 = [];
```

```
 table3 = [];
 table4 = [];
 % Make a table for each sensor that contains the analog voltage and the
 % distance in cm
 for i=1:1:length(table)
     if table(i,1) == 0 % Data from sensor 1
         table1 = [table1; table(i,2:1:3)];
     elseif table(i,1) == 1 % Data from sensor 2
         table2 = [table2; table(i, 2:1:3)];
     elseif table(i,1) == 2 % Data from sensor 3
         table3 = [table3; table(i, 2:1:3)];
     elseif table(i,1) == 3 % Data from sensor 4
         table4 = [table4; table(i, 2:1:3)];
     end
 end
x = 0:1:255;
% For sensor 1, plot curve
if length(table1) ~= 0
  figure();
  % Fit curve to table
  [P1,S1] = polyfit(table1(:,2),table1(:,1),7);
  yfit1 = polyval(P1,x);
  % Plot the curve
  plot(x,yfit1); % x-axis = voltage and y-axis = distance in cm
  title('Sensor 1');
  xlabel('Voltage');
  ylabel('cm');
end

% For sensor 2, plot curve
if length(table2) ~= 0
  figure();
  % Fit curve to table
  [P2,S2] = polyfit(table2(:,2),table2(:,1),7);
  yfit2 = polyval(P2,x);
  % Plot the curve
  plot(x,yfit2); % x-axis = voltage and y-axis = distance in cm
  title('Sensor 2');
  xlabel('Voltage');
  ylabel('cm');
end

% For sensor 3, plot curve
if length(table3) ~= 0
  figure();
  % Fit curve to table
  [P3,S3] = polyfit(table3(:,2),table3(:,1),7);
  yfit3 = polyval(P3,x);
  % Plot the curve
  plot(x,yfit3); % x-axis = voltage and y-axis = distance in cm
  title('Sensor 3');
  xlabel('Voltage');
  ylabel('cm');
end

% For sensor 4, plot curve
if length(table4) ~= 0
  figure();
  % Fit curve to table
  [P4,S4] = polyfit(table4(:,2),table4(:,1),7);
  yfit4 = polyval(P4,x);
```

```
        % Plot the curve
        plot(x,yfit4); % x-axis = voltage and y-axis = distance in cm
        title('Sensor 4');
        xlabel('Voltage');
        ylabel('cm');
      end
      return
end

% If a new calibrationis to be made, clear the table
if (get(handles.emptyCalibrationTable,'Value') == 1)
    table = [];
end

contents = get(handles.sensornr,'String');
sensornr = contents{get(handles.sensornr,'Value')};

if strcmp(sensornr,'IR sensor 1')
    sensor = 0;
elseif strcmp(sensornr,'IR sensor 2')
    sensor = 1;
elseif strcmp(sensornr,'IR sensor 3')
    sensor = 2;
elseif strcmp(sensornr,'IR sensor 4')
    sensor = 3;
end

% Get an average analog voltage value at a given distance for a given
% sensor
[error,analogData] = calibrateIRsensors(SERIALLINK,handles,sensor);

if error,
    set(handles.statusText,'String','Calibrating: Unable to send serial
data.' );
else
    cm = str2double(get(handles.edit_cmToCalibrate,'String'));

    table = [table;sensor cm analogData];

    disp(table);

end;


% --- Executes on selection change in sensornr.
function sensornr_Callback(hObject, eventdata, handles)
% hObject    handle to sensornr (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: contents = get(hObject,'String') returns sensornr contents as cell
array
%        contents{get(hObject,'Value')} returns selected item from sensornr


% --- Executes during object creation, after setting all properties.
function sensornr_CreateFcn(hObject, eventdata, handles)
% hObject    handle to sensornr (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called
```

```
% Hint: popupmenu controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function edit_cmToCalibrate_Callback(hObject, eventdata, handles)
% hObject    handle to edit_cmToCalibrate (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit_cmToCalibrate as
text
%        str2double(get(hObject,'String')) returns contents of
edit_cmToCalibrate as a double


% --- Executes during object creation, after setting all properties.
function edit_cmToCalibrate_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit_cmToCalibrate (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes on button press in emptyCalibrationTable.
function emptyCalibrationTable_Callback(hObject, eventdata, handles)
% hObject    handle to emptyCalibrationTable (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of emptyCalibrationTable



% --- Executes on button press in finishedCalibrating.
function finishedCalibrating_Callback(hObject, eventdata, handles)
% hObject    handle to finishedCalibrating (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of finishedCalibrating



function angleText_Callback(hObject, eventdata, handles)
% hObject    handle to angleText (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of angleText as text
```

```
%        str2double(get(hObject,'String')) returns contents of angleText as
a double


% --- Executes on button press in checkbox_goHome.
function checkbox_goHome_Callback(hObject, eventdata, handles)
% hObject    handle to checkbox_goHome (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of checkbox_goHome


% --- Executes on button press in Recharge_checkbox.
function Recharge_checkbox_Callback(hObject, eventdata, handles)
% hObject    handle to Recharge_checkbox (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of Recharge_checkbox



function goHomeText_Callback(hObject, eventdata, handles)
% hObject    handle to goHomeText (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of goHomeText as text
%        str2double(get(hObject,'String')) returns contents of goHomeText
as a double


% --- Executes during object creation, after setting all properties.
function goHomeText_CreateFcn(hObject, eventdata, handles)
% hObject    handle to goHomeText (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --------------------------------------------------------------------
function Untitled_11_Callback(hObject, eventdata, handles)
% hObject    handle to Untitled_11 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)


% --- If Enable == 'on', executes on mouse press in 5 pixel border.
% --- Otherwise, executes on mouse press in 5 pixel border or over COMValg.
function COMValg_ButtonDownFcn(hObject, eventdata, handles)
% hObject    handle to COMValg (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
```

```
function camIPText_Callback(hObject, eventdata, handles)
% hObject    handle to camIPText (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of camIPText as text
%        str2double(get(hObject,'String')) returns contents of camIPText as
a double


% --- Executes during object creation, after setting all properties.
function camIPText_CreateFcn(hObject, eventdata, handles)
% hObject    handle to camIPText (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
```

getRobotPose.m

Innhenter robotens posisjon

```
function [poses,myError]=getRobotPose(myCon,myHandles)

myError = 0;
poses=0;

inte=1;

if strcmp(myCon,'simulator'),
    poses=simRobot9('gp',myHandles); % [1 x 3]
    myError=0;

elseif strcmp(myCon, 'NXT_kamera'),
    myXY=mapping_get_robot_pos(myHandles.kamerarobot);
    poses(1)=myXY(1)*10;  %convert from cm to mm
    poses(2)=myXY(2)*10;  %convert from cm to mm
    poses(3)=mapping_get_robot_theta(myHandles.kamerarobot);  %positive
radians 0-2pi
    myError=0;

else
    if( serialAsyncWrite(myCon,'p') )
        set(myHandles.statusText,'String','getRobotPose: Unable to send
serial data' );
        return;
    else
        poseRead=0;
        while ~poseRead,
        [myError,data]=serialSyncRead(myCon,1);
        if(myError)
            disp('getRobotPose: no data received..')
            return;
        elseif data=='p'
            % fetch position data
            disp('pose header received')

            [myError,pose]=serialSyncRead(myCon,6);
            if(myError)
                disp('navigation_develop/getRobotPose: Error receiving
postion from robot')
                return;
            else
                %converting from signed int,
                xposi = pose(1)*256+pose(2);
                if( xposi > 32767) % negative value
                    myX= -(65536-xposi);
                else
                    myX=xposi;
                end

                yposi = pose(3)*256+pose(4);
                if(yposi > 32767)% negative value
                    myY= -(65536-yposi);
                else
                    myY=yposi;
                end

                % theta always positive 0-2*pi
```

```
                myTheta = (pose(5)*256+pose(6))/1000;

            end
            poseRead=1;

            poses(1)=myX;
            poses(2)=myY;
            poses(3)=myTheta;
        elseif data=='1' || data=='2' || data=='3' || data=='4' ||
data=='5' || data=='6' || data=='7' || data=='8',
            disp('navigation_develop/getRobotPose: IR data header
received')

        elseif data=='#'
            c=clock;
            str=sprintf('%d:\t%d:\t%d\t\t ##### DEBUG MARKER 1
#####',c(4),c(5),c(6));
            disp(str);
            %disp(sprintf('%d', c(6)));
        elseif data=='$'
            c=clock;
            inte = inte+1;
            str=sprintf('%d - %d:\t%d:\t%d\t\t $$$$$ DEBUG MARKER 2 $$$$$--
$',inte,c(4),c(5),c(6));
            disp(str);

        elseif data=='%'
            c=clock;
            str=sprintf('%d:\t%d:\t%d\t\t %%%%%%%%%% DEBUG MARKER 3
%%%%%%%%%%----%%',c(4),c(5),c(6));
            disp(str);

        % DEBUGGING - RECEIVE TOTAL TICKS
%         elseif data==char(1)
%             [myError,ticks]=serialSyncRead(myCon,2);
%             if(myError)
%                 disp('navigation_develop/getRobotPose: Error receiving
left wheel ticks from robot')
%                 return;
%             else
%                 ticks_temp = ticks(1)*256+ticks(2);
%                 if( ticks_temp > 32767) % negative value
%                     ticks_left= -(65536-xposi);
%                 else
%                     ticks_left=ticks_temp;
%                 end
%                 disp('left wheel ticks: ')
%                 ticks_left
%             end
%
        elseif data=='w', % collision warning maybe < 10 cm
            disp('collision warning received')
            set(myHandles.statusText,'String',num2str('collision warning
received') );
        elseif data=='e', % collision error < 5 cm (Robot stopped!)
            disp('collision error received, robot stopped')
            myError=1;
            break

        else
            disp('navigation_develop/getRobotPose: Wrong header received ')
```

```
            disp(data)
            pause(0.1);
        end
        end
    end
end
```

fullscan.m

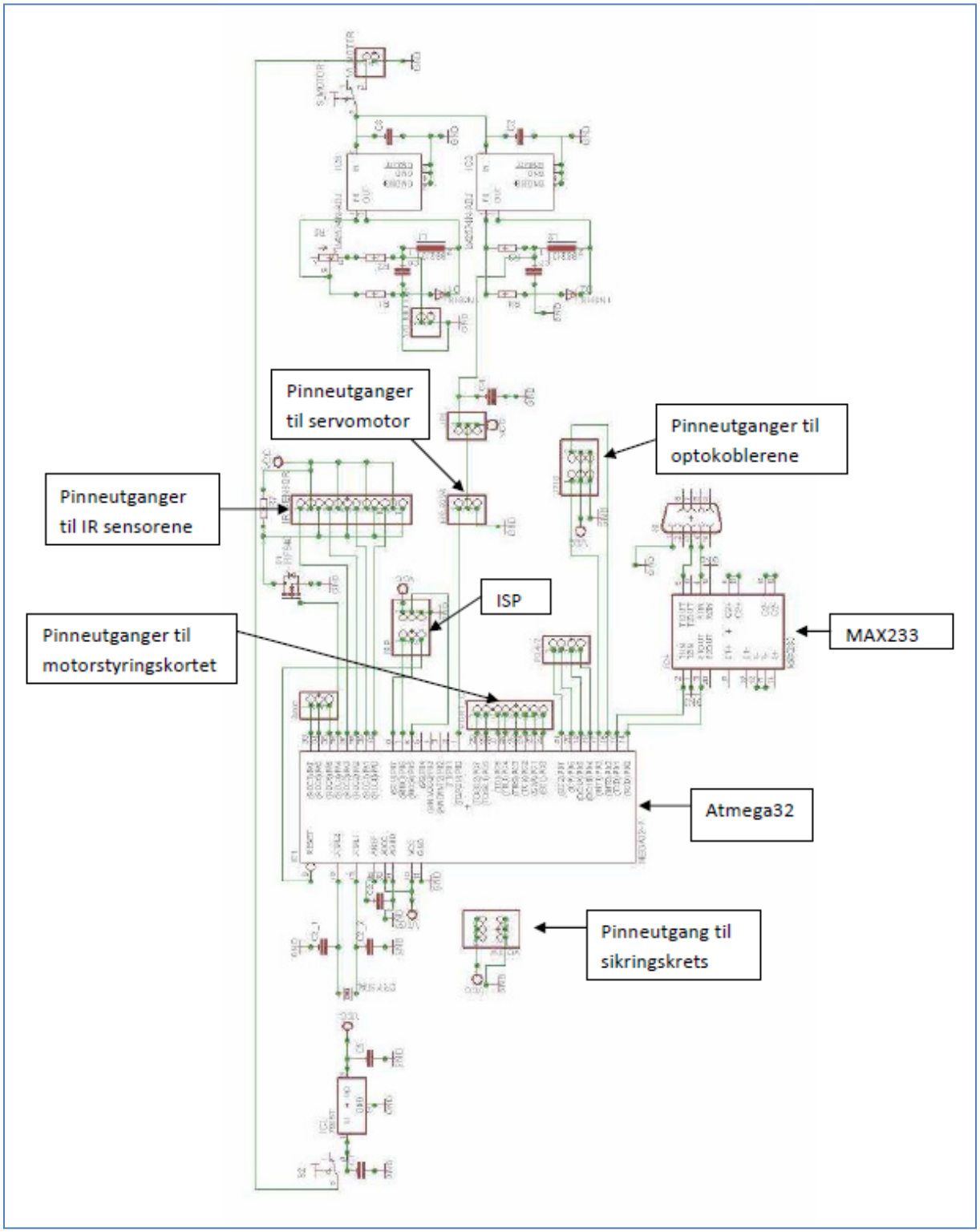
innhenter målinger fra det tilgjengelige sensorsystemet.

```
%FULLSCAN
% perform a full scan 360 degrees around the robot
% return an array with dim [n x 3], where n is the number of observations,
% first col contains sensorheader, second and third col are global position
% of sensordata given in (x,y)

function [myError, outData] = fullScan(myCon,myHandles)

myError = 1;
outData = [];



if strcmp(myCon,'simulator'),
    outData=simRobot9('f',myHandles);
    outData(:,2:3)=outData(:,2:3)*1000;
    myError=0;

elseif strcmp(myCon,'NXT_kamera'),
    camPath=strcat('http://', get(myHandles.camIPText, 'String'),
'/img/snapshot.cgi');
    tempstr=strcat('Address to snapshot from web cam: ', camPath);
    disp(tempstr);

    [myHandles.vertex_list_right myHandles.vertex_list_left] =
mapping_find_and_plot_vertex(camPath, myHandles.kamerarobot,
myHandles.vertex_list_right, myHandles.vertex_list_left, myHandles);


    %Convert from cm to mm
    verteces = 10* [myHandles.vertex_list_right;
myHandles.vertex_list_left];

    if isempty(verteces),
        %no data points
        disp('no verteces detected by camera')
        return;
    else
        myError=0;
        outData(:, 2) = verteces(:,1);
        outData(:, 3) = verteces(:,2);
        outData(:, 1) = 1;        %sensorheader (just one sensor)
    end

    %resetting vertex lists
    myHandles.vertex_list_right=[];
    myHandles.vertex_list_left=[];

else
    % serial interface to full scan...
    if myCon.bytesAvailable,
            disp('fullScan: Bytes in buffer before sending "f" to robot.
Number of bytes:')
            disp(myCon.bytesAvailable)
            disp('bytes read from buffer:')
            dummyData=fread(myCon)
    end
```

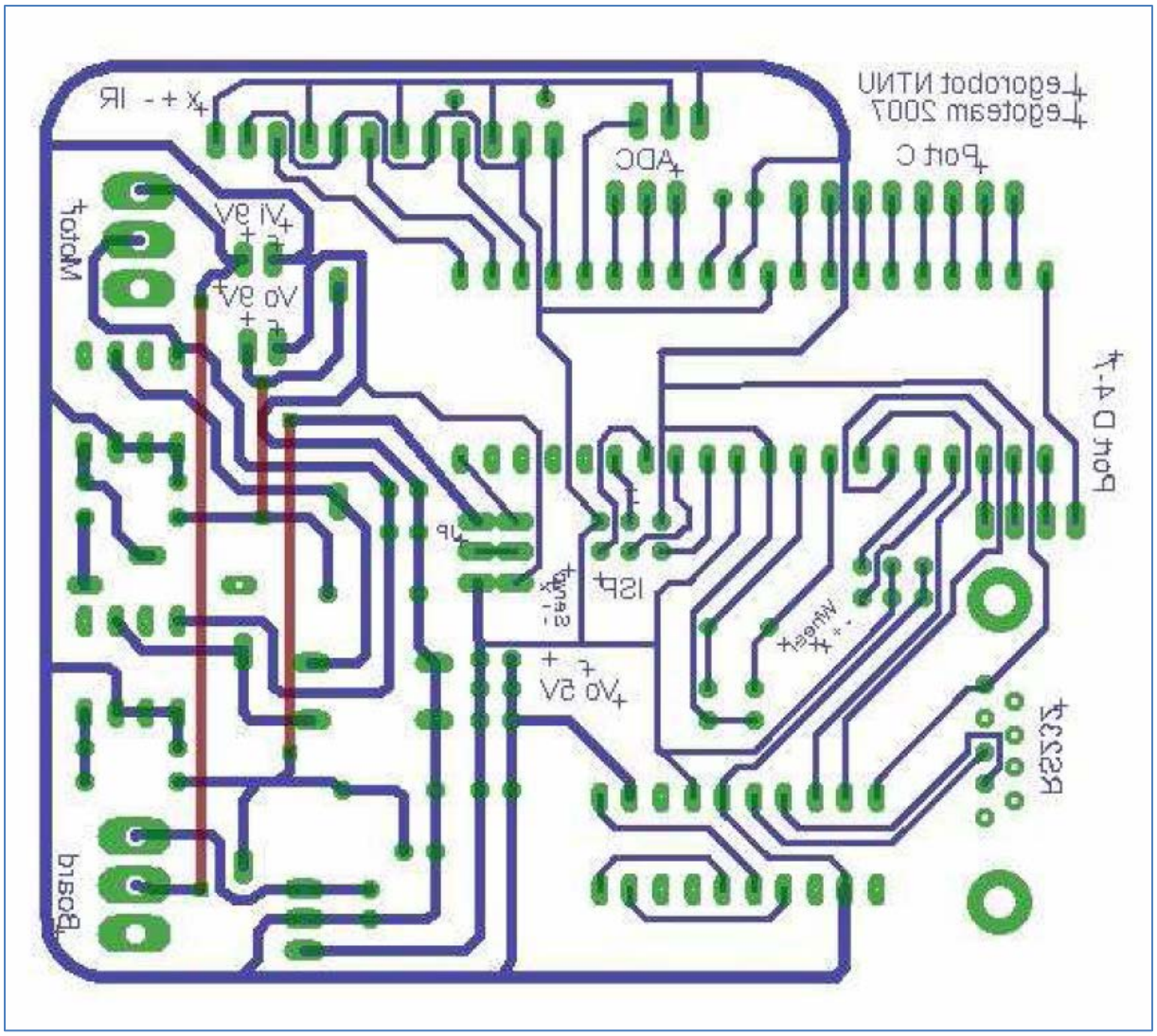
```
    %if( serialAsyncWrite(myCon,'1') ),
    if( serialAsyncWrite(myCon,'f') ),
        set(myHandles.statusText,'String','navigation_develop/fullScan:
Unable to send serial data' );
    else

        [myError,data1]=serialSyncRead(myCon,1);
        disp('fullScan: Received response to f: ')
        disp(char(data1))
        if(myError)
            disp('error reading byte after sending f ')
            return;
        %elseif data1=='1', % start receiving full scan data
        elseif data1=='f', % start receiving full scan data
            i=0;
            disp('got f, waiting for data')
            data2=0;
            while data2~='g', % while not finished...
                 %disp('getting sensornummer or g to leave mode:')
                bytes_available2=myCon.bytesAvailable;
                 %if myCon.bytesAvailable,
                 [myError,data2]=serialSyncRead(myCon,1);
                 disp('Data motat fra robot i fullscan');
                 disp(data2);
                %end
                 if (myError)
                    return
                elseif data2=='g'
                    %outData
                    %figure(2);
                    b=outData(:,2); % x data from all of the sensors
                    c=outData(:,3); % y data from all of the sensors
                    %plot (b,c,'bx')


                    disp('scan ferdig, fikk g')
                    set(myHandles.statusText,'String','Finished fullscan'
);

                    disp('antall målepunkter mottatt:')
                    disp(length(b))

                elseif data2=='5'   %Has to be here for no data
                    disp('no data 1')
                elseif data2=='6'
                    disp('no data 2')
                elseif data2=='7'
                    disp('no data 3')
                 elseif data2=='8'
                    disp('no data 4')
                elseif (data2=='1' || data2=='2' || data2=='3' ||
data2=='4'),
                    i=1+i; % Number of row in outData[]
                    %disp('got sensornummer (49-52) waiting for xxyy')
                    %disp(data2-48)
                    %data(1)
                    outData(i,1)=data2 - 48; % sensor number

                    [myError,data3]=serialSyncRead(myCon,4); % data3 has
four rows
```

```
                    disp('Data fra sensorer');
                    disp(data3);
                    if (myError),
                        return
                    else
                        %converting from signed int,
                        xpos = data3(1)*256+data3(2);
                        if( xpos > 32767) % negative value
                            myX= -(65536-xpos);
                        else
                            myX=xpos;
                        end

                        ypos = data3(3)*256+data3(4);
                        if(ypos > 32767)% negative value
                            myY= -(65536-ypos);
                        else
                            myY=ypos;
                        end

                    end
                    outData(i,2)=myX;
                    outData(i,3)=myY;
                else
                    disp('received f, but the following position data is
wrong')
                end
            end
        end
    end
    if myCon.bytesAvailable,
        data = fread(myCon);
        disp('leftover data on serialconnection after fullscan: ')
        disp(double(data))
    end

end
```

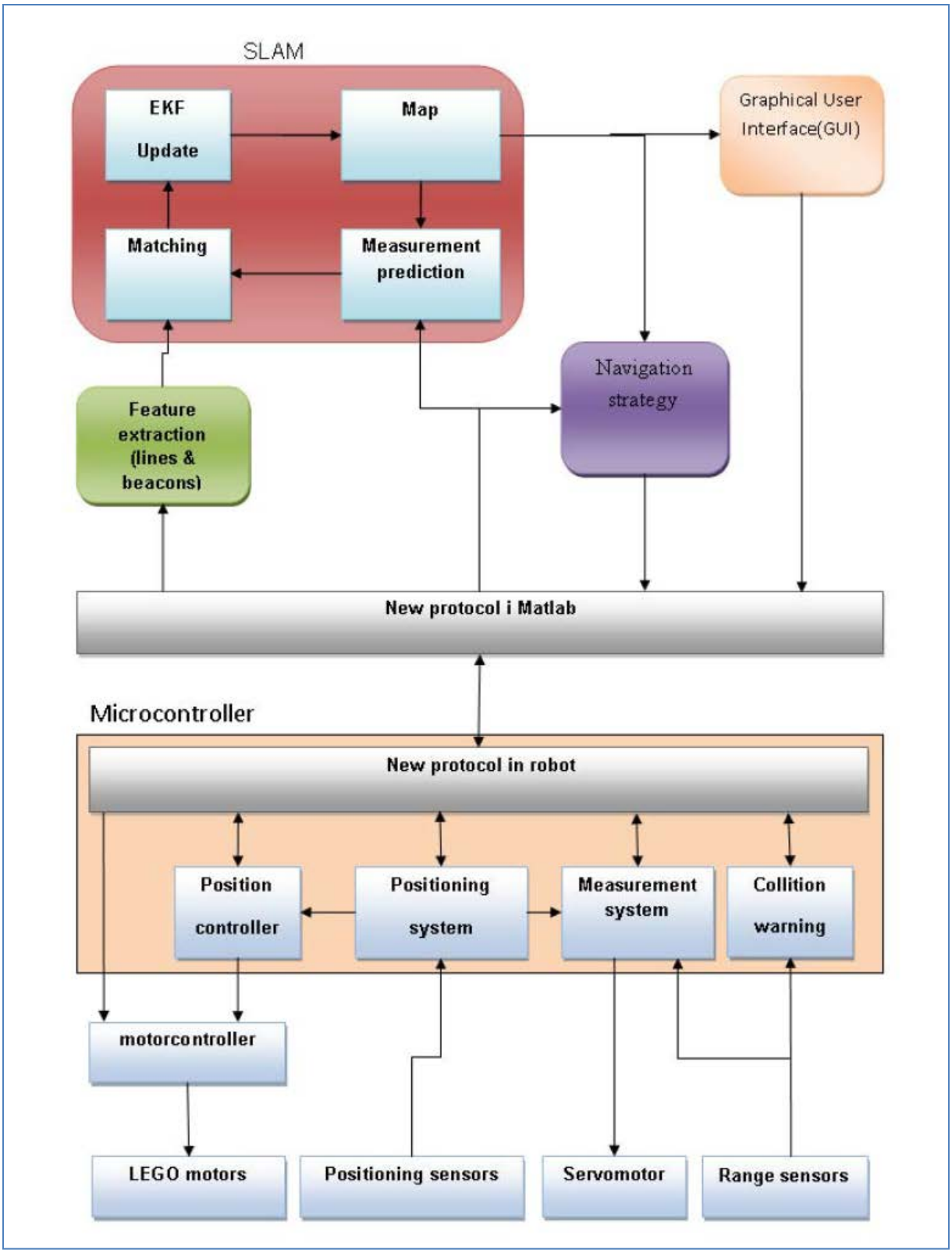
D. Figurer



Figur 17: Kretsskjema for hovedkortet (bilde fra (Tusvik, 2009))



Figur 18: Utlegg for hovedkortet (bilde fra (Tusvik, 2009))



Figur 19: Systemets oppbygging (figur fra (Magnussen, 2008))

E. Veiledning av brukergrensesnittet

Sitert i sin helhet fra (Tusvik, 2009), vedlegg A.

Vedlegg

A. Veiledning av brukergrensesnittet

Ettersom det kan være litt vanskelig å sette seg inn i alle funksjonene i brukergrensesnittet blir det i dette kapittelet gjennomgått hvordan disse fungerer og kan brukes. Figur 11 viser brukergrensesnittet i sin helhet.
Brukergrensesnittet er delt inn i forskjellige seksjoner som er beskrevet nedenfor:

Basic

På venstre side er det en seksjon "Basic" som er delt inn i underseksjonene:

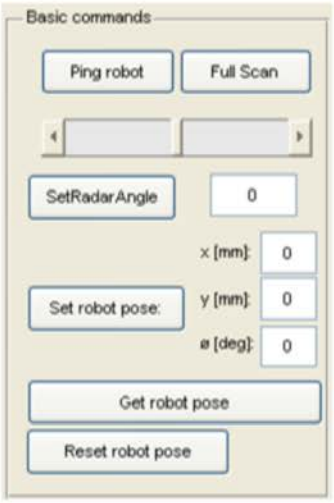
- Connection
- Basic commands
- Robot navigation

I "Connection" kan man koble til eller fra simulatoren eller COM-porten som blir benyttet for å sende og motta data fra roboten.

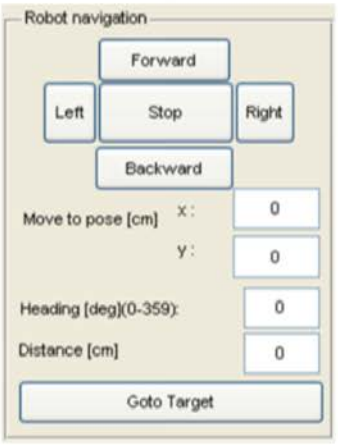
I "Basic commands" kan man pinge roboten for å se om det er kontakt mellom robot og pc. Dersom det er kontakt vil meldingen "Robot responding" bli skrevet i statusfeltet. Ved å trykke "Full Scan" utføres en fullstendig skanning av omgivelsene og en figur som viser et plott fra denne vil dukke opp. Vinkelen på IR tårnet kan settes enten ved å bruke slideren eller skrive inne en vinkel, gitt i grader,

58

inn i tekstboksen. Ved hjelp av "Set robot pose" kan man sette posisjonen internt i roboten. Roboten vil i dette tilfellet ikke flytte på seg, men skrive verdiene som angitt i tekstfeltene for x, y og θ, til minnet internt på roboten. "Get robot pose" leser verdiene x, y og θ fra minnet på roboten og skriver disse ut feltet "Estimated position". "Reset robot pose" nullstiller x, y og θ-verdiene på roboten, samt nullstiller antall tics.



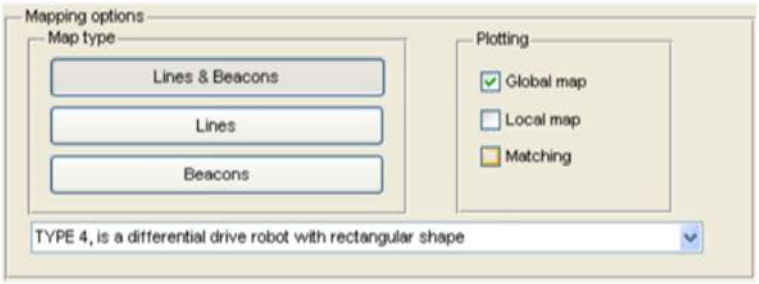
I "Robot navigation" kan man styre roboten ved å kjøre framover, bakover, snu mot høyre, snu mot venstre eller stoppe. Man kan også sette hvilken x, y posisjon man vil at roboten skal kjøre til, hvilken vinkel den skal snu seg med eller hvor langt den skal kjøre rett fram.



Mapping options

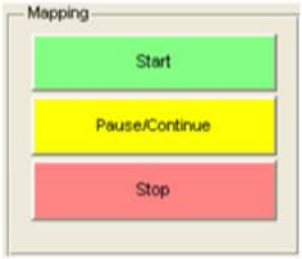
Her kan man velge om man vil kartlegge ved hjelp av linjer eller beacons, eller begge deler. Det kan også velges hvilke kart som skal vises. Det globale kartet viser alle linjer og beacons som har blitt observert av roboten, samt stien roboten har gått. Det lokale kartet viser linjer som har blitt observert etter en "Full Scan". Det vil si at det opprettes et nytt lokalt kart hver gang det blir utført en fullstendig skanning under kartlegging. Matching sammenligner observasjoner fra det lokale kartet mot predikterte observasjoner i det predikterte globale kartet ved hjelp av NNSF, Nearest Neighbor Strategy Filter. Et kart vil vise sammenligningen av beacons, mens et annet kart vil sammenligningen av linjer.

I nedtrekksmenyen kan man velge hvilken form roboten skal ha i kartet.



Mapping

Ved å trykke på startknappen begynner roboten kartleggingen av omgivelsene, mens stopp- og pauseknappen henholdsvis stopper og setter kartleggigen på pause.

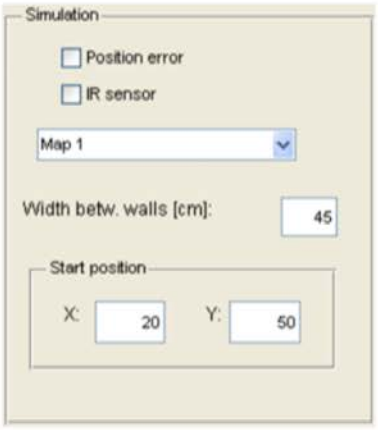


Simulation

Dersom simulatoren er tilkoblet, kan man i denne seksjonen velge hvilken labyrint roboten skal kartlegge i nedtrekksmenyen. Det finnes seks forskjellige labyrinter det kan simuleres i, se avsnitt

60

"Simulator" for å få en oversikt over hvordan de forskjellige labyrintene ser ut. Videre kan man velge om posisjonsfeil og IR-sensor-feil skal være med under kartleggingen, bredden det skal være mellom veggene i labyrinten og hvor roboten skal starte kartleggingen.

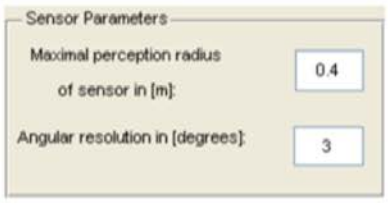


Navigation strategy

For dette prosjektet finnes det bare en navigasjonsstrategi, som vil være på hele tiden. Dersom det blir utviklet nye navigasjonsalgoritmer kan de implementeres og aktiveres i denne seksjonen.

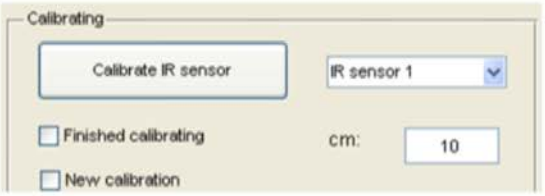
Sensor Parameters

Her bestemmes det hvilken avstand det skal være på radiusen i en sirkel som dannes rundt roboten. Området innenfor denne sirkelen defineres som kjent område for roboten. "Angular resolution in [degrees]" bestemmer hvor mange punkter sirkelen skal bestå av. Dersom denne verdien settes til 3 grader, vil sirkelen bestå av 120 punkter som har 3 grader i mellom seg.



Calibrating

Ved å trykke på "Calibrating IR sensor" kan man kalibrere IR sensoren som er oppført i nedtrekksmenyen. Man kan også markere om en ny kalibrering skal utføres eller om den pågående kalibreringen er ferdig. I tekstfeltet kan man angi hvor langt unna IR sensoren er fra gjenstanden det måles mot. For å lese om hvordan kalibreringen av en IR sensor skal gjennomføres henvises det til avsnittet "Kalibrering av IR sensorer".

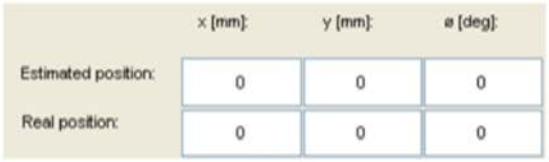


Close

Lukker brukergrensesnittet.

Posisjon

Under kartlegging og simulering vises den estimerte posisjonen til roboten i "Estimated position", mens "Real position" kun viser verdier under simulering ettersom roboten sin virkelige posisjon under kartlegging er ukjent.



Status

I statusfeltet kommer det meldinger fra Matlab etter hvert som programmet kjører.

62

Drive and Scan

"Drive and Scan" medfører at roboten også sender posisjonsoppdateringer til Matlab mens den kjører. Vanligvis sendes posisjonsoppdateringer kunne når roboten står i ro. SJEKK OPP!

Main Plot Window

I det store vinduet midt i brukergrensesnittet, tegnes det globale kartet inn. Det er også her simuleringen visualiseres.

Docking

Dersom "Recharge" eller "Go Home" krysses av, vil roboten returnere til lade stasjonen under kartleggingen. Ved "Go Home" vil roboten returnere etter angitt antall steg i tekstboksen, mens ved "Recharge" vil den returnere ved neste steg.