NTNU
Det skapende universitet

Fjernstyring av Legorobot

Trond Magnussen

Master i teknisk kybernetikk
Oppgaven levert: Januar 2008
Hovedveileder: Tor Engebret Onshus, ITK

Norges teknisk-naturvitenskapelige universitet
Institutt for teknisk kybernetikk

Oppgavetekst

Det har blitt utviklet en Legorobot som styres halvautomatisk, der roboten beveger seg fysisk, mens resten bare er synlig på dataskjermen som overlagret informasjon. En har også en tilkoblet enhet for å kartlegge omgivelsene til roboten ved hjelp av infrarød avstadsmåling, slik at en kan kartlegge en labyrint som roboten skal bevege seg gjennom. Kartlegging av labyrinten og ruteplanlegging er også mulig. Roboten kan også overvåkes ved hjelp av kamerabasert posisjoneringssystem, slik at en får vist fram hvordan den egentlig beveger seg i labyrinten.

Oppgaven går ut på:
1. Videreutvikling av eksisterende navigasjon- og kartleggingsmetoder.
2. Utvikle en strategi som fjerner posisjoneringsfeil som oppstår under kartlegging.
3. Utrede og implementere en ladestasjon for roboten.
4. Optimalisere og forbedre kartleggingen av objekter og labyrinter.

Oppgaven gitt: 04. august 2007
Hovedveileder: Tor Engebret Onshus, ITK

Forord

Denne masteroppgaven er en videreføring av min prosjektoppgave ”Fjernstyring av Legorobot”. Rapporten dokumenter arbeidet med å videreutvikle et styringsprogram i MATLAB for en Legorobot utviklet ved Institutt for Teknisk Kybernetikk ved NTNU, Trondheim. Roboten er autonom og laget for å kartlegge ukjente omgivelser.

Arbeidet som er lagt ned har vært interessant og spennende. Jeg anser denne oppgaven som en god og relevant avslutning av studiene ved NTNU.

Personer som har vært til god hjelp i løpet av mitt arbeid med denne masteroppgaven og som fortjener en stor takk:

- Professor Tor Onshus som kunne tilby masteroppgaven og har vært en godfaglig veileder gjennom alle faser med arbeidet.
- Paul Sverre Kallevik og Johannes Schrimpf som jeg har samarbeidet og delt kontor med gjennom høsten. Kontorlivet hadde ikke vært det samme uten ”Legoteam 2007”.
- Ann Torill Sandbæk for uvurderlig moralsk støtte og tålmodighet, samt korrekturlesing av oppgaven.
- De ansatte ved ITK’s komponentlager og mekaniske verksted som alltid stiller opp med et smil. ☺

Sammendrag

Denne masteroppgaven omhandler videreutvikling av en fjernstyrt LEGO robot ved Institutt for Teknisk Kybernetikk. Roboten har tidligere gjennomgått flere utviklingssteg via prosjekter og diplom/masteroppgaver ved instituttet.

Utgangspunktet for oppgaven er en robot bygget av Lego som kan fjernstyres trådløst fra PC. Roboten hadde innebygd et posisjoneringssystem og sensor for måling av avstand til omgivelsene. På PC siden ble MATLAB med tilhørende kartleggings- og navigasjonsalgoritmer brukt for å produsere kart over omgivelsene mens roboten kjørte, såkalt *SLAM (Simultaneous Location And Mapping)*. Roboten kunne derfor under visse betingelser autonomt navigere i ukjente omgivelser. Mer konkret baserte kartbyggingen seg på å utvinne et linjebasert kart der alle objekter i robotens miljø ble redusert til linjer. Derav ble det vanskelig å produsere for eksempel runde objekter i kartet. Andre begrensinger var at posisjoneringssystemet besto kun av en *dead-reckoning* metode som gjorde at posisjonen til roboten etter hvert som den kjørte ble beheftet med usikkerhet som etter hvert ville gå mot uendelig. I tilegg var det svakheter i programmet når det gjaldt navigeringsstrategier i ukjente områder.

I denne oppgaven gjøres det programutvikling i MATLAB for å bedre navigasjons- og kartleggingsegenskapene. Systemet baserer seg på at roboten skal kartlegge og navigere i sanntid. Det er implementert algoritmer som gjør det mulig å motta sensordata og behandle disse mens roboten kjører. Dataene presenteres grafisk i et utvidet brukergrensesnitt der både linjesegmenter og punkter tegnes i samme kart. Slik kan objekter med ulik geometrisk form representeres på en bedre måte enn før. Den innebygde simulatoren er utvidet med flere relevante labyrinter og funksjonalitet.

Fra tidligere arbeid var det implementert en veggfølgingsalgoritme laget for å navigere i en linjekart. I denne oppgaven har den eksisterende veggfølgingsalgoritme blitt tilpasset til å håndtere et kombinert linje- og punktbasert kart. En ny metode for å detektere ukjente områder er laget og brukes i navigeringen. Det er foretatt en studie av en ny navigasjonsstrategi som baserer seg på et *action-selected* prinsipp der navigasjonsstrategier blir valgt på bakgrunn av prioriteter.

Det er innført en ladestasjon som roboten kan starte og returnere til. Hensikten er å gjøre roboten mer autonom ved at den kan få ladet batteripakkene uten menneskelig påvirkning. Posisjonsestimatet til roboten kan også forbedres ved at den kan returnere til et fast punkt. Et tilpasset *EKF* (*extended kalman filter*) muliggjør oppdatering av posisjon.

Tester viser at de utviklete algoritmer for kartlegging og navigering isolert fungerer tilfredsstillende. Likevel kan det ikke konkluderes med at kartleggingen har blitt mye bedre. Posisjonsestimatet er den avgjørende faktoren som begrensninger systemet.

Videre arbeid er foreslått til å være forbedring av posisjonsestimatet til roboten. Dette kan gjøres ved endringer i hardware og software, samt å bruke kamera for regulering av robotposisjon. Når posisjoneringen har blitt bedre kan det være aktuelt å se på kartlegging i tre dimensjoner. Det er i rapporten henvist til artikler som beskriver en utvidelse til 3D-*SLAM*. En annen retning å bevege seg i er å se på samhandlede roboter, der to eller flere LEGO-roboter samarbeider om kartleggigen.

Innhold

1 Innledning

Målet med denne oppgaven er å videreutvikle en LEGO-robot. Roboten er i stand til å kartlegge enkle labyrinter som oppfyller visse kriteria. Disse kriteriene går hovedsakelig på avstanden mellom veggene i labyrinten og selve formen. Årsaken til disse begrensingene ligger eksisterende kartleggings- og navigasjonsstrategi. I tillegg er posisjonering en stor begrensning. Denne oppgaven hovedsakelig handler om videreutvikling av eksisterende navigasjons- og kartleggingsmetoder, samt å gjøre robotens posisjonsestimat bedre og mer robust.

Roboten er bygget av Lego og at det brukes billige og enkle sensorer i dette prosjektet. Dette legger rammer på hva som er mulig å oppnå av resultater med denne oppgaven. Et godt styringsprogram kan ikke fullt og helt kompensere for tvetydige måledata. Resultatet skal bli en forbedring av totalsystemets evne til å utføre sin oppgave.

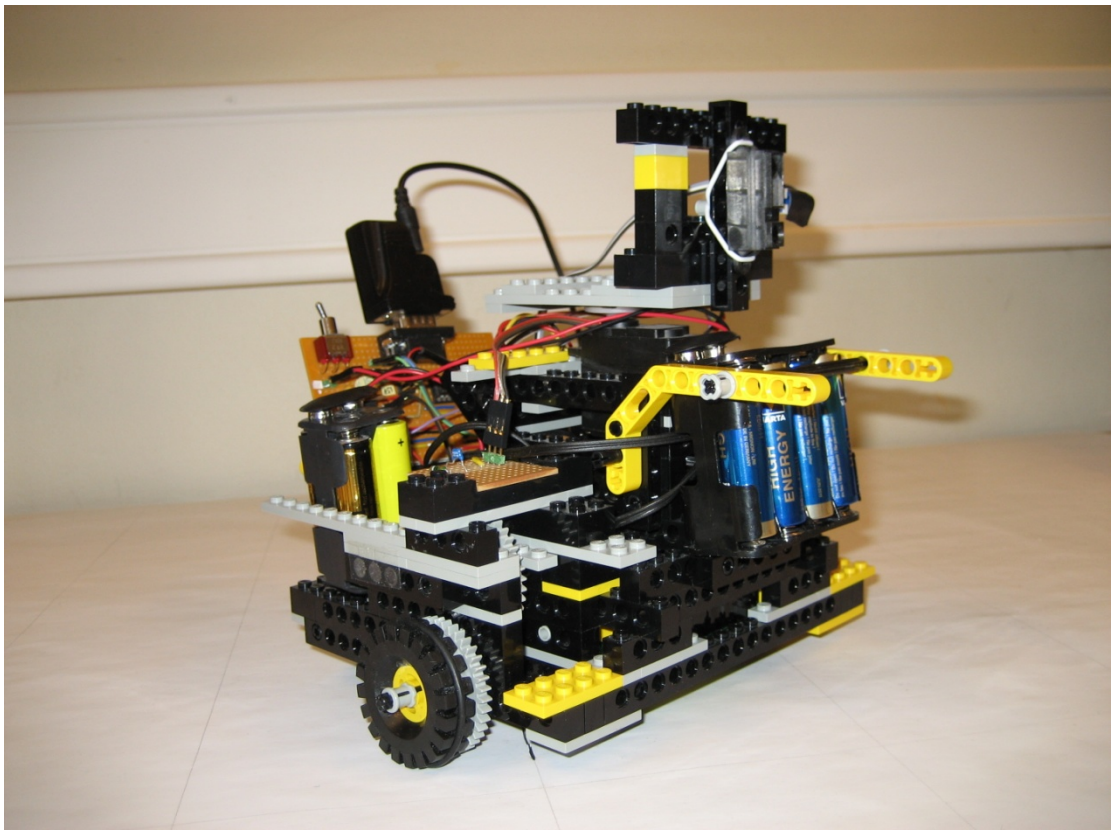
Arbeidet er gjort i samarbeid med Johannes Schrimpf og Paul Sverre Kallevik som har hatt ansvar for hardware til roboten. Det henvises til deres prosjektrapporter for utfyllende informasjon om dette [1] [2]. Det samlede resultatet skal bli en forbedring av totalsystemets evne til å utføre sin oppgave.

Rapporten begynner med å beskrive utgangspunktet for oppgaven i kapittel 2. Kapittel 3 tar for seg bakgrunnsteori for oppgaven.. I kapitel 4 handler om posisjonering. Kartlegging er beskrevet i kapitel 5. Kapitel 6 omhandler modellering og simulering. Kapitel 7 handler om navigering og ruteplanlegging. En oversikt over det totale systemet kommer i kapitel 8. Resultater og tester beskrives i kapittel 9. I kapittel 10 gjøres en diskusjon og vurdering av resultat og prosess. Konklusjon kommer i kapitel 11 Til slutt settes det opp forslag til videre arbeid i kapittel 12.

2 Utgangspunkt

Prosjektets utgangspunkt er LEGO-roboten i Figur 2.1. Roboten har gått igjennom flere utviklingssteg og kan oppsummeres i følgende punkter:

- Roboten bygges av Håkon Skjelten i forbindelse med prosjekt i det 9. semester ved NTNU [3]. Et enkelt brukergrensesnitt utvikles også.
- Systemet videreutvikles gjennom masteroppgaven til Sveinung Helgeland ved NTNU [4] til å kunne utføre samtidig kartlegging og navigasjon i en labyrint. Det lykkes ikke å kartlegge en labyrint tilfredsstillende.
- Ytterligere arbeid med kartgenerering gjøres i prosjektet til Bjørn Syvertsen i det 9. semester ved NTNU [5]. Systemet lykkes fortsatt ikke i å kartlegge en labyrint tilfredsstillende
- Bjørn Syvertsen fortsatte arbeidet med kartlegging og navigasjon i sin masteroppgave ved NTNU [6]. Roboten lykkes i å kartlegge noen ferdige definerte labyrinter.
- I prosjektoppgaven til undertegnende [7] utvikles en enkel simulator for roboten. Navigasjon og kartlegging forbedres slik roboten er i stand til å kartlegge større labyrinter.



Figur 1: LEGO-robot, august 2007.

2.1 Hardware

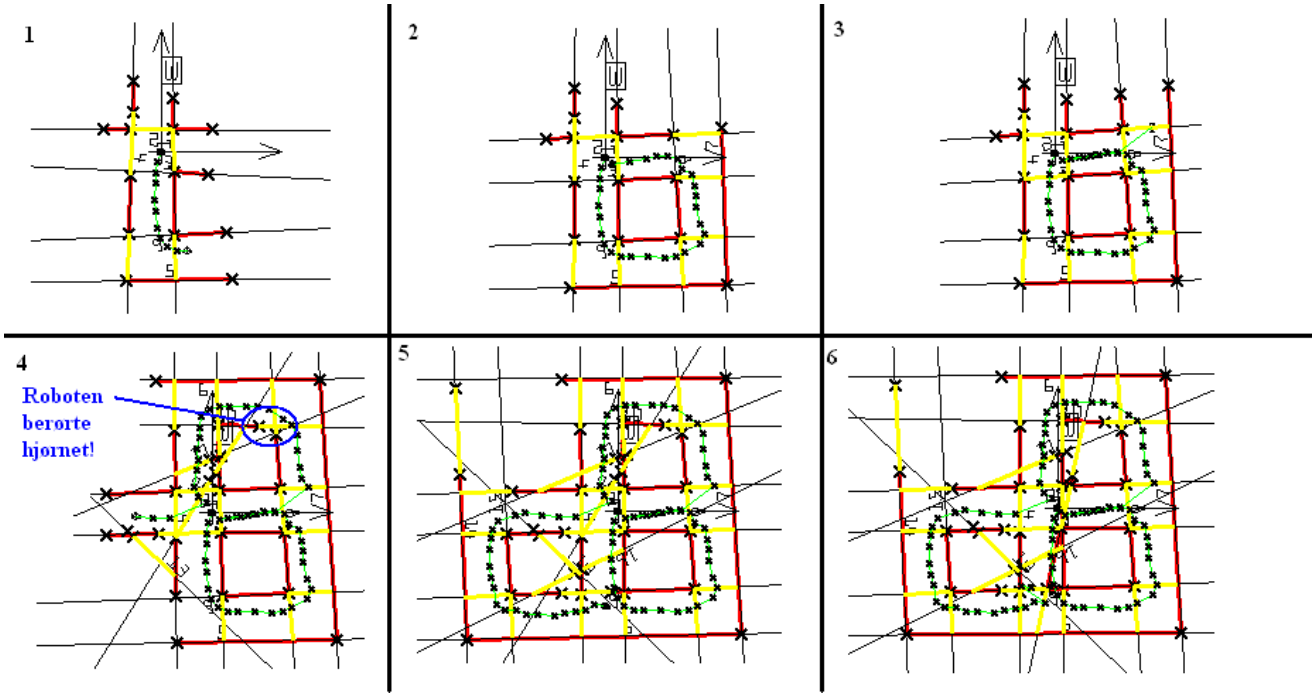
Roboten var utstyrt med et roterbart hode der det var påmontert en avstandsmåler som baserte seg på infrarødt lys. Det roterbare robothodet var festet på en servomotor som kunne dreies 180 grader. Slik kunne avstanden til labyrintvegger måles. Roboten hadde to LEGO motorer som sørget for fremdrift. Motorene ble drevet av en standard RCX styringsenhet fra LEGO. To optokoblere ble brukt for måling av tilbakelagt distanse. Et kretskort var montert bak på roboten og besto hovedsakelig av en mikrokontroller fra Atmel og et trådløst-RS232 grensesnitt. Mikrokontrolleren påmontert kretskortet lyttet etter signaler fra PC og utførte disse fortløpende. Oppgavene til mikrokontrolleren var motorstyring og innsamling av data. Tabell 3.1 viser kommandosettet som mikrokontrolleren aksepterte. RCX-enheten mottok sine styresignaler fra mikrokontrolleren.
Den infrarøde sensoren hadde en rekkevidde fra 10 cm til 80 cm og relativt god vinkeloppløsning. Servomotoren som roterte avstandssensorene var uten tilbakekobling og det var antatt en usikkerhet på omtrent 2 grader ved setting av vinkel ifølge [6].

2.2 Software

Prosjektet bygger på et eksisterende software design kalt *The CAS Robot Navigation Toolbox* [8] [9] og er en samling algoritmer for MATLAB. Det er et rammeverk for å gjøre *Simultaneous Localization and mapping* (SLAM). Rammeverket har blitt modifisert ved flere anledninger tidligere [5] [6] [7] og er i stand til å generere et kart ut fra sensordata som innsamles av en robot i sanntid. De vesentligste egenskapene som dette rammeverket har er at algoritmene, kartrepresentasjonen, robotmodellen og sensordataene er bygd inn egne uavhengige moduler. Dette gjør blant annet at endringer i kommunikasjonen med roboten, brukergrensesnittet eller i navigasjonsstrategien kan gjøres relativt uavhengig av hverandre. Det eksisterer også en enkel simulator med innebygde labyrinter som kan brukes i utviklingsfasen med nye kartbyggingsmetoder og navigasjonsstrategier. Utgangspunktet for videre utvikling i software regnes for å være godt.

2.3 Kartlegging og navigering

Robotens egenskaper for kartlegging og navigering har blitt testet i alle tidligere arbeid og resultatet har vært varierende. I de siste arbeider med roboten [6] [7] klarte den å navigere og kartlegge enkle labyrinter bygget opp av rette veggflater. Gjennom disse testene og simuleringer ble det konkludert med at begrensingene først og fremst lå på tilbakelagt lengde og dermed posisjonsestimatet til roboten. Det ble også registrert at avstandsmålinger mot skrå vegger kunne forårsake vesentlige feil i kartet. Simuleringer av lange kjørelengder viste at det var lite sannsynlig at systematiske feil i software gjorde at kartleggingen etter hvert ble dårligere og dårligere jo lengre roboten kjørte. Det var imidlertid ikke blitt gjort større tester av kartleggings- og navigasjonsalgoritmene i MATLAB når det gjaldt andre typer omgivelser enn de enkle labyrintene.



Figur 2: Kartlegging fra tidligere arbeid [7]

3 Teori og tidligere arbeid

En forstudie til oppgaven ble utført for å finne aktuelle fremgangsmåter og metodikk som kunne forbedre navigasjon og kartlegging i MATLAB. Det vil gjøres rede for tidligere arbeid gjort innenfor robotnavigasjon lokalt og globalt. Forskning innenfor robotteknikk og roboter er omfattende og det vil fokuseres på faktorer som har sammenheng med oppgaveteksten; navigasjonsstrategier og samtidig posisjonering og kartlegging, såkalt *SLAM* (*Simultaneous Location And Mapping*), i det todimensjonale plan.

3.1 Andre Legoroboter ved NTNU

Etter at Lego Mindstorm ble kjøpt inn ved Institutt for teknisk kybernetikk har det blitt bygget flere legoroboter i prosjekt/masteroppgaver. En oversikt over hva som har vært gjort tidligere ved instituttet var ønskelig da det ikke fantes noe slikt fra før. Oppgavene som har vært gitt har hatt forskjellige veiledere og kildehenvisningene i rapportene er ikke helt fullstendige.

3.1.1 Institutt for teknisk kybernetikk

Det er ved institutt for teknisk kybernetikk gitt mange oppgaver som involverer Legoroboter. Det viktigste blir presentert her:

- LEGOLAB [10], Håkon Svendsen, hovedoppgave, vår 2002. Oppgavetekst: ”Denne oppgaven tar utgangspunkt i en prosjektoppgave høsten 2001 hvor det ble utviklet en applikasjon for fjernoperasjon av en LEGOrobot. Roboten er bygget ved bruk av LEGO MINDSTORMS. Det er utviklet en sirkelrund bane. Roboten kan styres via Internett. Videre er det support for audio- og videoforbindelse. Denne hovedoppgaven skal videreutvikle applikasjonen og integrere denne i Cyberlab-nettverket; et nettverk for global Internettbasert aksess til fysiske laboratorier. Brukergruppen defineres å være ungdomsskoleelever.” Veileder: Professor Bjarne A. Foss.

- LEGOROBOT [11], Harald Bjørtomt, hovedoppgave, vår 2003. Oppgavetekst: ”En eksisterende LEGO Mindstorm-robot skal ferdigstilles slik at den kan inngå i en robotutstilling ved Vitensenteret i Trondheim. Roboten skal kunne styres over Internett via labformidleren cyberlag.org, og på robotens nettside skal et spill med robotstyring tilbys. Spillet skal gå ut på å plukke opp bordtennisballer med roboten innenfor en tidsgrense. Antall bordtennisballer registreres og lagres i en poengliste.” Veileder: Amund Skavhaug.

Robotene som er presentert her var ikke laget for å være fullt ut autonom, men skulle styres av brukere via Internet. Det var implementert en ladestasjon som robotene kunne returnere til når spenningen på batteriene ble for lav. Robotene hadde påmonterte lyssensorer slik at den kunne følge opptegnede striper i banen tilbake til ladestasjonen.

3.1.2 Andre institutt

Bruken av LEGO Mindstorm er ikke utbredt ved andre institutt, men følgende oppgave ble funnet ved Institutt for datateknikk, IDI.

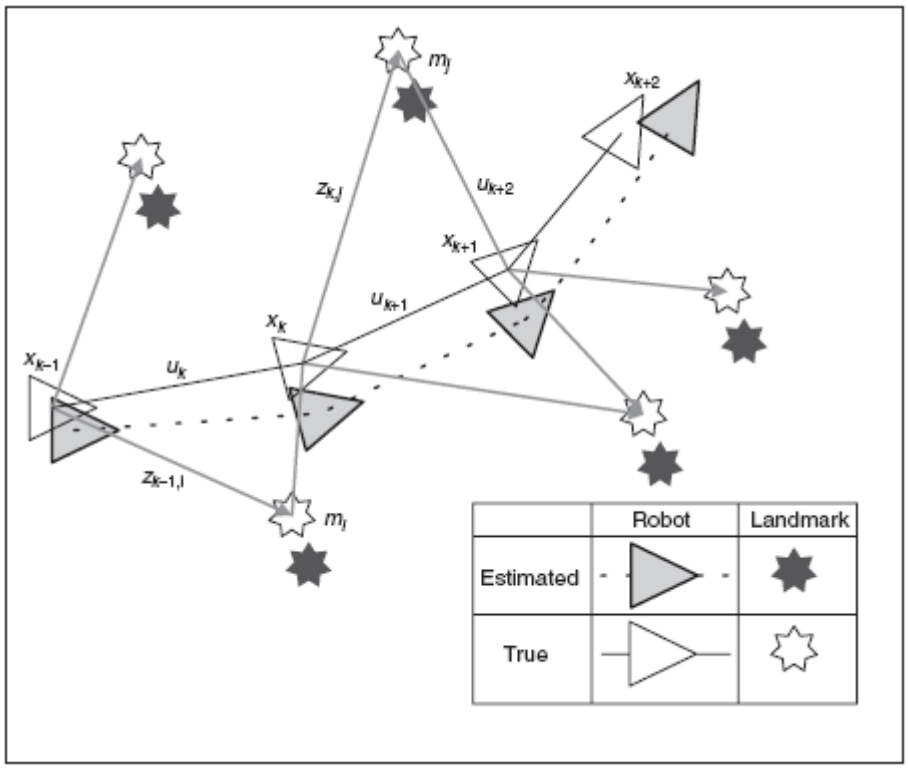
- An analysis of the development of a safety-critical system for an Urban Search, Jean Paul Franky Friquin [12], masteroppgave, vår 2006, oppgavetekst: ”This report describes the

design and implementation of a safety-critical system of a LegoMindstorms Urban Search and Rescue (USAR) robot prototype. Veileder: Professor Tore Stålhane.

Dette var en robot laget for å teste ut egenskaper ved et sikkerhetskritisk system. Roboten kunne navigere i enkel labyrint ved hjelp av et påmontert kamera.

3.2 SLAM (Simultaneous Location and Mapping)

Begrepet *SLAM* er et konsept som bygger på at en robot skal kunne samtidig kartlegge og navigere i ukjente omgivelser. På grunn av usikkerhet i målemetoder når det gjelder posisjon til roboten og i avstanden til observerte objekter er ikke *SLAM* å anse som trivielt. Derfor brukes teknikker som *EKF*(*extended kalman filter*), partikkel filtre (Monte Carlo metoder) og tilpasning av måledata fra avstandssensorer i *SLAM*. Forskning på *SLAM* har det siste tiåret vært omfattende og det har ført til mange ulike innfallsvinkler for å løse *SLAM*-problemet [13] [14].



Figur 3: Det essensielle *SLAM*-problemet [13]

Utgangspunktet for *SLAM* problemet beskrives i Figur 3-1. Et estimat av posisjonen til både robot og landemerker er nødvendig. De virkelige posisjonene er aldri kjent eller på noen måte målt direkte. Observasjoner gjøres mellom virkelig robotposisjon og landemerker.

3.2.1 Formulering av *SLAM* problemet

Formuleringen av *SLAM* problemet er i litteraturen beskrevet meget utfyllende. Følgende beskrivelse er hentet fra [13] og presenterer dagens status for *SLAM*. Det gjøres oppmerksomt på at beskrivelsen er kortfattet og noen overganger er kuttet. Likevel skal informasjonen være intuitiv og relevant for en kjapp innføring i *SLAM*.

En robot beveger seg i ukjente omgivelser og foretar relative målinger av observerte landemerker med sensorer plassert på roboten, slik som i Figur 3-1. Ved hvert tidssteg *k* er følgende størrelser definert:

- $\mathbf{x}_k$: tilstandsvektor som beskriver robotens plassering og orientering i rommet
- $\mathbf{u}_k$: pådragsvektor satt ved tidssteg *k*-1 for å bevege roboten til tilstand $\mathbf{x}_k$.
- $\mathbf{m}_i$: vektor som beskriver plassering av landemerke *i* og dens virkelige posisjon regnes som tidsinvariant
- $\mathbf{z}_{ik}$: observasjon av landemerke *i* ved tidssteg *k*.

Følgende sett defineres:

- $\mathbf{X}_{0:k} = \{\mathbf{x}_0, \mathbf{x}_1, \ldots, \mathbf{x}_k\} = \{\mathbf{x}_{0:k-1}, \mathbf{x}_k\}$: alle robotposisjoner
- $\mathbf{U}_{0:k} = \{\mathbf{u}_1, \mathbf{u}_2, \ldots, \mathbf{u}_k\} = \{\mathbf{u}_{0:k-1}, \mathbf{u}_k\}$: alle pådrag
- $\mathbf{m} = \{\mathbf{m}_1, \mathbf{m}_2, \ldots, \mathbf{m}_n\}$ alle landemerker
- $\mathbf{Z}_{0:k} = \{\mathbf{z}_1, \mathbf{z}_2, \ldots, \mathbf{z}_k\} = \{\mathbf{z}_{0:k-1}, \mathbf{z}_k\}$: alle observasjoner av landemerker

Det kreves at sannsynlighetsfordelingen

$$P(\boldsymbol{x_k}, \boldsymbol{m} | \boldsymbol{Z_{0:k}}, \boldsymbol{U_{0:k}}, \boldsymbol{x_0})$$

(1)

kan bli beregnet for alle *k* for å bygge opp en modell som kan løses rekursivt ved bruk av Bayes teorem. En observasjonsmodell kan lages for å beskrive målinger

$$P(\boldsymbol{z}_k | \boldsymbol{x}_k, \boldsymbol{m})$$

(2)

På samme defineres en modell for robotens bevegelse

$$P(\boldsymbol{x}_k | \boldsymbol{x}_{k-1}, \boldsymbol{u}_k)$$

(3)

Beregningene videre er detaljert beskrevet i [13] og blir ikke gjengitt her. Hovedtrekkene er at *SLAM* problemet kan reduseres til en todelt rekursiv(tidsoppdatering) prediktiv(måleoppdating) korreksjons form

- Tidsoppdatering:

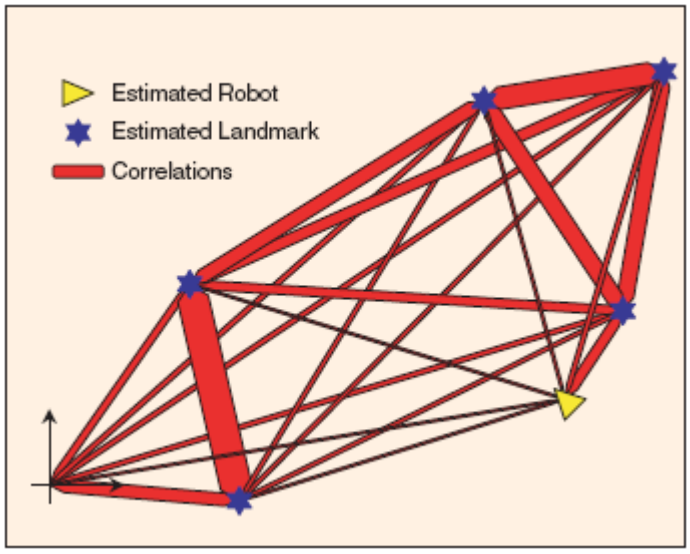
$$P(\boldsymbol{x_k}, \boldsymbol{m} | \boldsymbol{Z_{0:k-1}}, \boldsymbol{U_{0:k}}, \boldsymbol{x_0}) = \int P(\boldsymbol{x_k} | \boldsymbol{x_{k-1}}, \boldsymbol{u_k}) \times P(\boldsymbol{x_{k-1}}, \boldsymbol{m} | \boldsymbol{Z_{0:k-1}}, \boldsymbol{U_{0:k-1}}, \boldsymbol{x_0}) \boldsymbol{dx_{k-1}}$$

(4)

- Oppdatering av måling**:**

$$P(\boldsymbol{x_k}, \boldsymbol{m}|\boldsymbol{x_{0:k}}, \boldsymbol{U_{0:k}}, \boldsymbol{x_0}) = \frac{P(\boldsymbol{z_k}|\boldsymbol{x_k}, \boldsymbol{m})\, P(\boldsymbol{x_k}, \boldsymbol{m}|\boldsymbol{Z_{0:k-1}}, \boldsymbol{U_{0:k}}, \boldsymbol{x_0})}{P(\boldsymbol{z_k}, \boldsymbol{m}|\boldsymbol{Z_{0:k-1}}, \boldsymbol{U_{0:k}})} \tag{5}$$

Det utredes videre i [13] om teorien bak problemet og dette vil ikke bli gjengitt her. Utredningene om *SLAM* kan lettest oppsummeres som en analogi presentert i Figur 3-2. Landemerkene er koblet sammen med fjærer som beskriver korrelasjonen mellom disse. Når roboten beveger seg mellom landemerkene øker fjærstivheten eller korrelasjonen (de røde linjene blir tykkere). Ettersom landemerkene blir observert og estimert posisjon blir korrigert, forplanter endringene seg utover i nettverket. Figuren viser at roboten også er korrelert med landemerkene.



Figur 4: *SLAM* analogi [13]

3.2.2 Løsning av *SLAM problemet*

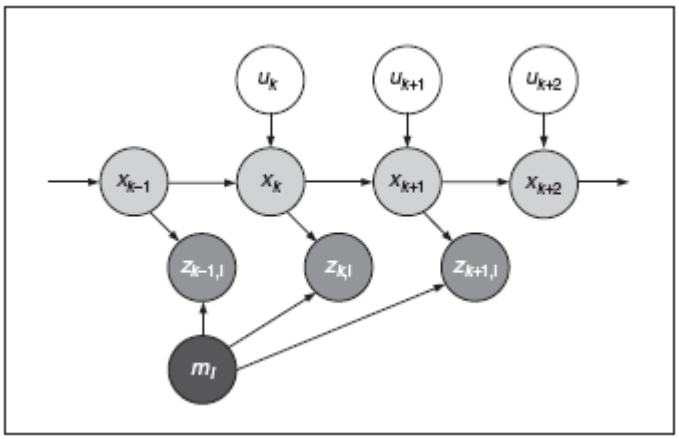
Løsning på *SLAM* problemet innebærer å finne en fornuftig representasjon for observasjonsmodellen (2) og bevegelsesmodellen (3) som gir en effektiv og konsistent beregning av priori og posteriori fordelingene i (4) og (5). Den vanligste representasjonen er en tilstandsrom modell som tillegges gausisk støy og som impliserer at man kan bruke *EKF*(*extended Kalman filter*) for å løse *SLAM* problemet. En alternativ representasjon er å beskrive robotens bevegelse (3) som et sett av en mer generell ikke-gausisk sannsynlighetsfordeling. Denne representasjonen leder til bruk av et Rao-Blackwellized partikkel filter, eller en såkalt *FastSLAM* algoritme, for å løse *SLAM* problemet. *EKF-SLAM* og *Fast-SLAM* er de to mest anvendte løsningene, men nye potensielle løsninger som informasjons-tilstand form [15] har blitt foreslått de siste årene.

3.2.3 *EKF-SLAM* og Rao-Blackwellized filter

Den matematiske beskrivelsen av *EKF-SLAM* er grundig beskrevet tidligere i [5] [6] [8] [13]. *EKF-SLAM* løsningen er derfor velkjent i fagmiljøet og den innehar mange av fordelene og ulempene som eksisterer i standard *EKF* løsninger til generelle navigasjons- og posisjoneringsproblemer (*tracking*). De viktige momenter å ta hensyn til:

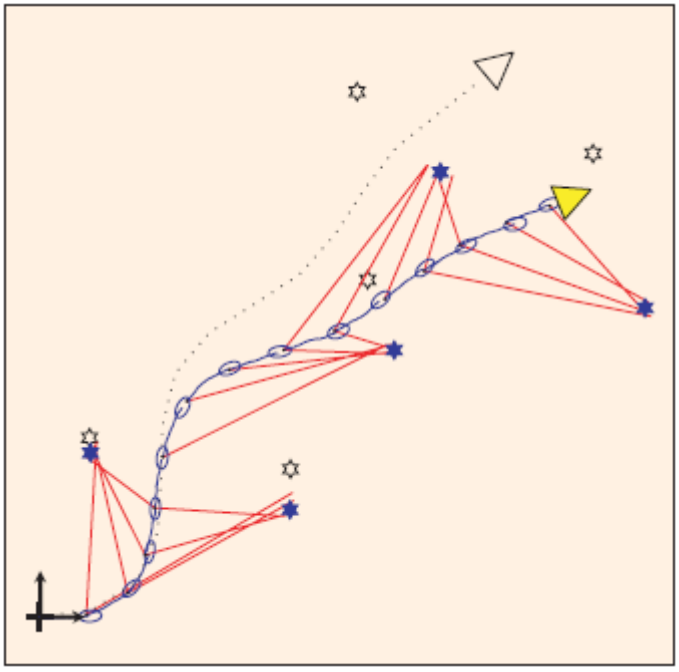
- Konvergens: Variansen til alle landemerker konvergerer mot en nedre grense gitt av initial usikkerhet i robotposisjon og observasjoner.
- Databehandling: Målingsoppdateringen i algoritmen krever at kryss-kovarians matrisen oppdateres hver gang en observasjon gjøres. Dette impliserer at kjøretiden blir $O(n^2)$, der n er antall observasjoner av landemerker.
- Data assosiasjon: Formuleringen av *EKF-SLAM* løsningen er spesielt ømfintlig ovenfor feil i data assosieringen av observerte landemerker [16]. Problemet med å lukke sløyfen, det vil si når roboten observerer kjente landemerker på nytt, kan være vanskelig. Assosieringsproblemet øker i omgivelser der landemerkene ikke består av punkter, men er mer komplekst utformet, og at de ser forskjellig ut fra ulike vinkler.
- Ulinearitet: Bruk av linearisering i bevegelsesmodellen og målemodellen kan gi store feil i kartbyggingen [17].

FastSLAM algoritmen [18] baserer seg på et Rao-Blackwellized filter. Teorien bak denne algoritmen og bruk av filteret er beskrevet nærmere i [13]. Figur 4 viser en grafisk modell av Fast-SLAM algoritmen. Ut fra den matematiske gjennomgangen kan man anta følgende: Hvis alle tidligere posisjonstilstander er kjent, vil alle tilstandene i kartet være stokastisk uavhengige fordi observasjonene av landemerkene er betinget uavhengige. Fast-SLAM algoritmen definerer hver partikkel som ulike robot trajektorie hypoteser. For å gi et intuitivt bilde av algoritmen er en realisasjon av en trajektorie gitt i Figur 5. De avbildede usikkerhetsellipsene for hvert oppdateringssteg blir brukt til å estimere robotposisjon. Ved å anta at de estimerte robotposisjonene er perfekt kan man ut fra det estimere landemerkene. Kvaliteten på kartet er derfor styrt av nøyaktigheten til trajektorien. Ved å sette sammen mange slike trajektorier kan man lage en sannsynlighetsmodell for robotposisjonen.



Figur 5: *Fast SLAM* [13]

Effektiviteten til *Fast-SLAM* algoritmen reduseres på grunn av dens statistiske svakheter [13]. Når kartet marginaliseres introduseres avhengigheter på tidligere posisjon og målinger, og når man deretter gjør oppdateringer kan mye av denne informasjonen forsvinne og nøyaktigheten på stokastiske data reduseres. Imidlertid finnes det praktiske løsninger som fungerer [19].



Figur 6: Fast-SLAM realisasjon [13]

3.3 Navigasjonsstrategier

Her vil det gjøres rede for ulike navigasjonsstrategier.

3.3.1 Tilstandsmaskiner

Tilstandsmaskiner vil si at navigasjonsalgoritmen baserer seg på å skifte mellom ulike tilstander. For eksempel er Tremaux' algoritme [20] som er implementert en tilstandsmaskin.

3.3.2 Fuzzy logic

Fuzzy logic er en reguleringsmetode som brukes i mange anvendelser. For styring av roboter er metoden ikke så ubredt, men interessen rundt temaet øker. Eksempler på fungerende implementasjoner er gitt i [21] [22].

3.3.3 Hierarkiske og reaktive paradigmer

Hierarkiske og reaktive paradigmer omtales i det skriftlige bidraget fra Eurobot 2007 utført ved

NTNU [23]. Disse paradigmene representerer to tilnærmingsmåter til kunstig intelligens og planleggingssystemer hos roboter. Hovedforskjellen mellom det hierarkiske og det reaktive paradigme er måten roboten planlegger på. Den hierarkiske framgangsmåten bygger på at roboten bruker sensorinformasjon til å planlegge sin neste handling for deretter å utføre denne handlingen (Sense – plan – act). Her kommer også kognitiv aktivitet inn med bruk av både tilbakekopling og foroverkobling. Slike kognitive aktiviteter kan være å bestemme hva en robot skal gjøre når den er klar for nye handlinger.

Det reaktive paradigme er inspirert av biologisk intelligens og baserer seg på å bruke sensorinformasjon direkte for å produsere handlinger (Sense - act). Den overordnede planleggingsdelen eksisterer ikke i det reaktive paradigme i motsetning til hierarktiske paradigme. Det foreslåes en vertikal oppdeling med flere lag liggende vertikalt. Hvert lag har sin oppførsel og tilstandsmaskin. Lagene kan plasseres parallelt eller i serie slik at utførelsen blir deretter. Prioriteter brukes for å favorisere eller undertrykke bestemte lag. Den reaktive fremgangsmåten er mye brukt der roboten ikke har oppdatert sensorinformasjon eller når det er usikkerhet rundt utfallet av en handling.

3.4 Valg av videre framgangsmåte

Analyser og teori i 3.1, 3.2 og 3.3 ble vurdert opp mot utgangspunktet for oppgaven i 2.2 og 2.3 for å avgjøre hvordan navigasjon og kartlegging skulle utvikles videre.

3.4.1 Kartlegging og navigasjon

Utgangspunktet for kartleggingen var en modifisert *EKF-SLAM* versjon [7] implementert i MATLAB. Applikasjonen har støtte for linjebaserte kart og en navigasjonsløsning som er beregnet for et linjekart. Punktbaserte kart kan også lages, men det eksisterer ikke noen form for navigeringsløsninger for slike kart. Mulighetene for å fusjonere disse to karttypene vurderes som en naturlig videreføring og utvikling i programmet.

Definering av ukjente områder gjøres på en ny måte i et kombinert linje/punktbasert kart. Tidligere ble ukjente områder definert med linjestykker som tegnes opp mellom de allerede oppdagede linjesegmenter i kartet. I Figur 2 markeres åpninger som gule linjer i kartet, mens veggene er markert som røde linjesegmenter. Dette fungerer tilfredsstillende i et linjekart, men for et punktbasert kart vil ikke denne fremgangsmåten fungere. Et annet argument er at i omgivelser med få landemerker vil den implementerte måten å lokalisere åpninger på feile totalt.

Når man etter hvert er i stand til å gjengi flere typer objekter på en adekvat måte i et kombinert linje/punktbasert kart og man kan definere de ukjente områdene for roboten under kjøring, er det muligheter for å implementere ulike navigasjonsstrategier og teste ut disse.

4 Posisjonering

4.1 Odometri

Roboten måler retning og posisjon ved bruk av *dead-reckoning.* Ulempen ved å måle posisjon på denne måten er at usikkerheten i målingene øker ubegrenset. Det er ikke implementert noen form for oppdatering av posisjon.

En metode for å korrigere posisjonsestimatet er å bruke faste punkter i omgivelsene, *beacons*, som roboten kan bruke for å oppdatere posisjonen sin. Eksempler på dette finnes for eksempel i [24]. Problemer som kan oppstå med denne metoden er at roboten kan bevege seg utenfor det synlige området for de faste punktene og posisjonen kan da ikke oppdateres. Ofte vil roboten være i kontakt med en eller flere beacons slik at en viss posisjonsoppdatering kan gjøres. I tillegg kan robotens odometri sørge for en god posisjonsoppdatering i en begrenset tid. Sammen kan odometri og navigering med faste punkter gi en god posisjonsoppdatering dersom roboten ikke utenfor synlig område for lenge av gangen.

Bruk av kamera til å oppdatere posisjonen er en annen metode som er aktuell. Kameraet kan da plasseres direkte på roboten [25] eller det kan plasseres slik at det dekker hele området roboten kan bevege seg på [26]. Er området så stort at et kamera ikke dekker alt kan flere kameraer benyttes. En bruker da kjente bildegjenkjenningsteknikker for å finne linjer og objekter. Skal roboten lokaliseres under kjøring må man også ta høyde for bevegelse i bildet. Hastighet og akselerasjon må beregnes. Bildeoppdatering er gir gode resultater og det forskes mye på metoder for å få robust og effektiv tracking av objekter.

En versjon som også kan benyttes er å få roboten til å returnere til et kjent fast punkt for å oppdatere posisjonen sin. Ved å velge startpunktet som et kjent fast punkt har man et posisjonsestimat som ikke er beheftet med usikkerhet. Videre kan man også få gjort servicefunksjoner på roboten som for eksempel lading av batterier. En slik ladestasjon som roboten kan returnere til krever ikke noe ekstra utstyr som kameraer og tilhørende programvare. Metoden er fysisk enkel og krever ikke mye ekstra implementasjon i software.

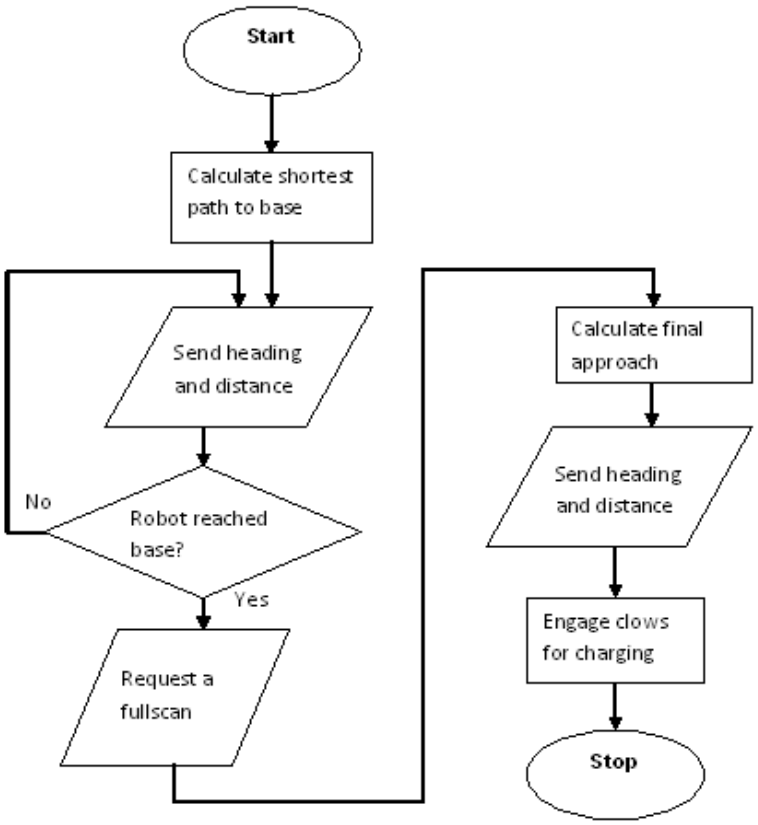
4.2 Oppdatering av posisjon

Roboten skal returnere til et kjent punkt når posisjonsestimatet blir for usikkert. Det kjente punktet settes til å være en ladestasjon som roboten også kan bruke for å lade batterier og kalibrere sensorer.

Hvilke kriterier må være oppfylt for å returnere til et fast/kjent punkt?

- Usikkerheten til posisjonsestimatet blir større enn en fastsatt grense
- Lavt batterinivå

For å komme seg tilbake til ladestasjonen er det blitt implementert en navigeringsstrategi som er integrert i funksjonen *lineBeaconSLAM*. Se Figur 7 for flytdiagram.



Figur 7: Gå tilbake til ladestasjon

4.2.1 Prediksjon av posisjon i software

Det er ikke implementert noen form for prediksjon av posisjonsestimatet i MATLAB. I rapporten til Bjørn Syvertsen [6] er det gitt et forslag på hvordan det en slik funksjonalitet kan implementeres. Det er i rammeverkets originalversjon lagt til rette for en posisjonsprediksjon, så det gjenstår derfor å tilpasse modellen og tillegge usikkerhet på målingene. Beregningene gjøres da i etterkant av en forflytning og ikke i sanntid som ideelt sett kunne vært gjort i mikrokontrolleren på roboten. Funksjonen *predict.m* tilhørende robotobjektet beregner en sannsynlighetsellipse rundt roboten som viser der roboten sannsynligvis befinner seg.

4.3 Ladestasjonen

Verkstedet ved instituttet sto for den fysiske byggingen av ladestasjonen. Det er påmontert en brakett med to tynne aluminiumsplater som roboten kan koble seg til for å lade batteriene. Figur 8 viser ladestasjonen.

Når roboten er på vei tilbake til ladestasjonen er posisjonsestimatet dårlig. Derfor kan det være en utfordring å få roboten til å kjøre inn på slik måte at gripeklørne treffer aluminiumsplatene. En

løsning var å bruke markeringer på underlaget i nærheten av ladestasjonen som brukes i H. Bjørtomt sin rapport [11]. Ulempen med det er at man vil trenge en ny sensor på roboten og at man er avhengig av at roboten klarer å detektere markeringene.

En annen metode er å bruke landemerker i kartet til å kjøre inn i ladestasjonen og nullstille posisjonsestimatet. Veggene i ladestasjonen kan brukes som referanse fordi de ble tegnet opp ved et tidligere tidspunkt da posisjonen til roboten var sikker.

Figur 8: Ladestasjon

5 Kartgenerering

5.1 Tolkning av nye objekter

Objekter og labyrinter som ble testet i tidligere arbeid var

- Rektangler og kvadrater
- Linjebaserte objekter

Andre varianter av disse objekter kunne muligens blitt gjenkjent og brukt i navigeringen. Likevel var det nødvendig å utvide kartleggingen fra et linjekart til et kombinert linje/punkt basert kart.

5.1.1 Linjebaserte kart

Opptegning av sirkulære og bueformede objekter er vanskelig å gjengi med linjer og segmenter. Det er ikke umulig, men krever da at man klarer å stille inn kartleggingsparametere for linjer på en riktig måte. Hvis man for eksempel skal detektere en sylinder (flasker, runde bokser, etc) vil dette objektet avbilde seg som en sirkel i planet. Ved å bruke linjer og segment kan man tegne opp objektet som et polygon. Det krever imidlertid at man har satt grensen for segmentlengder som utvinnes lavt nok. Settes denne for høyt forkastes rådataene som mottas og ingenting registreres i kartet. I tilfelle med en sylinder på diameter 10 cm kan denne for eksempel representeres som en polygon med seks kanter som hver har lengde 5 cm. Minimal segmentlengde må da settes 5 cm eller noe mindre for å ta høyde for målefeil som oppstår. Man klarer da i de fleste tilfeller å detektere sylinderen(se avsnitt under testing). Ulempen er da at man kan få forholdsvis mange falske segmenter i andre deler av kartet, særlig dersom posisjonsestimatet er dårlig. Jo større usikkerhet i posisjonen jo større er sjansen for å detektere falske linjestykker. Derfor er det viktig at ikke grensen for segmentlengde ikke settes for lavt. Med for lavt menes da at man bør vite noe om området roboten skal kartlegge i og robotens egne fysiske egenskaper. Består robotens miljø av mange små objekter og robotens posisjonsestimat er bra, kan det være fordelaktig å sette segmentlengden noe lavere. Er derimot posisjonen beheftet med mye usikkerhet bør segmentlengden økes. Dette krever at man har god forhåndskunnskap om området som roboten skal kartlegge og gjør hele systemet mindre autonomt. En løsning på dette er funnet ved å innføre dynamisk parametertilpasning som er implementert slik at grenseverdien for segmentlengden endres mens roboten kjører.

Et alternativ ville være å lage nye objektklasser som kan representere formen direkte, det vil si klasser for ellipser, hyperbler, parabler og så videre. Fordelen er at man får avbildet objektet mer nøyaktig. Imidlertid er ulempene ved å bruke en slik løsning ganske klare. For det første er praktisk umulig å dekke alle typer objekter med forskjellig form og uregelmessigheter. Det ville kreve mye tid å utvikle en slikt konsept. For det andre vil det kreve flere målinger per objekt jo flere objektklasser man innfører for å skille mellom de ulike objektene. Da er det like greit å bare tegne opp objektene punktvis.

5.1.2 Punktbaserte kart

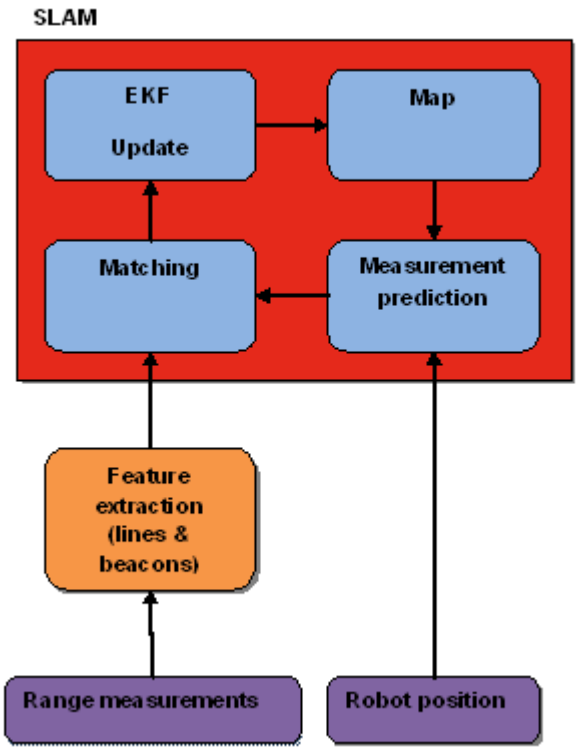
Fra tidligere arbeid er det gjort et forsøk på å lage en kartleggingsmetode som baserer seg på utvinne punkter istedenfor linjer. Dette arbeidet har blitt studert og er videreutviklet. Utgangspunktet var at man kunne velge mellom hvilke *features* (punkter eller linjer) man ville bruke for kartlegging. Det

var laget et enkelt rammeverk for bruk av såkalte *beacons* eller punkter. Rammeverket kunne ta inn måledata, behandle disse og presentere resultatet i lokale og globale plott. Ingen form for navigering var laget. Det var nødvendig å lage et fungerende rammeverk for punktbasert kartlegging og deretter fusjonere dette med det linjebaserte kartleggingssystemet. Slik kunne man beholde fordelene med at det linjebaserte kartet allerede var implementert og fungerte tilfredsstillende samtidig som det punktbaserte kartet kunne ta seg av andre objekttyper.

5.2 Integrering av punkt- og linjebasert kart

Integrasjon av linjebaserte og punktbaserte kart krevde endringer i mange av funksjonene som eksisterte fra tidligere. En ny funksjon, *LineBeaocnSLAM*, ble opprettet der hovedløkka for kartleggingen skulle ligge. Linjebasert- og punktbasert *SLAM* ble plassert etter hverandre slik at programmet først laget linjer og segmenter ut fra sensordataene, deretter ble de gjenværende sensordata brukt til punktbasert kartlegging. Mellom disse to hoveddelene i det nye rammeverket måtte det opprettes en del vedlikeholdsfunksjoner. Splitting og fusjonering av ulike *features* i det globale kartet måtte gjøres da de to kartleggingsmetodene ikke kunne håndterte samme type objekter. Forskjellen mellom punktbasert og linjebasert kartlegging lå mest på behandling av datastrukturer, kjøretid i MATLAB og opptegning av kartet. Selve behandlingen av sensordata foregår relativt likt.

Figur 8 viser strukturen på den nåværende *SLAM* algoritmen.



Figur 9: *SLAM*

6 Modellering og simulering

6.1 Bakgrunn

Ved utvikling av algoritmer for navigasjon og kartlegging vil et simuleringsverktøy være til stor nytte. Fordeler ved å bruke en simulator vil være redusert tid for utvikling av programvare og at man lettere kan rette opp feil i implementerte algoritmer. Videre er man ikke nødt til å kjøre det fysiske systemet hver gang man vil fremskaffe navigasjonsdata fra roboten. Til sammen gir dette spart tid og energi da en unngår unødig slitasje på det fysiske utstyret.

En simulering vil gjøre det mulig å overvåke fysiske prosesser bedre og man har tilgang til variabler og tilstander i systemet som ellers ville være vanskelig å måle i virkeligheten. En algoritme som er tilfredsstillende i en simulering har også stor sjanse til å fungere bra når den blir implementert på et fysisk system. Likevel er det viktig at eksperimenter som gjøres i en simulator også blir testet ut i virkeligheten. En simulator representerer kun en forenklet modell av roboten og dens omgivelser, og tar derfor ikke hensyn til prosesstøy, virkelig oppførsel til sensorer og så videre [27].

6.1.1 Posisjonering

I den eksisterende løsning for posisjonering brukes det to optokoblere, en for hvert hjul. Hver optokobler måler rotasjonen av et tannhjul som via flere drev er koblet til robotens hjul. Måling skjer altså ikke på det hjulet som er i kontakt med underlaget. Som følge av den valgte løsning for posisjoneringen kan det listes opp feilkilder som påvirker posisjonsmålingen [6]:

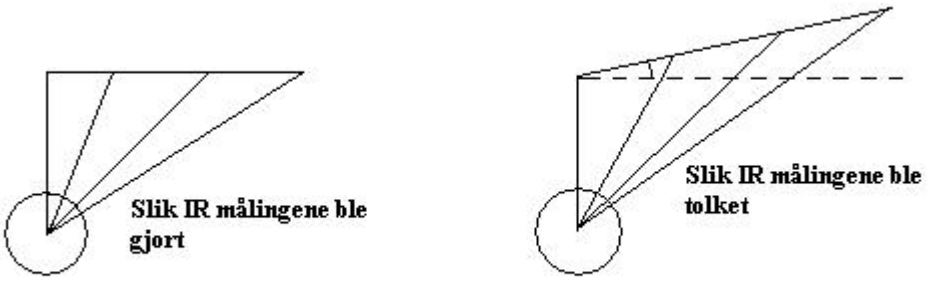
- Endelig oppløsning i vinkelsensor og endelig samplingsrate
- Glidning mellom hjul og underlag
- Slark i gir og tannhjul

På bakgrunn av disse punktene og empiriske data kan man komme frem en karakteristikk av posisjoneringssystemet. Karakteristikken kan sammenfattes slik:

- Robotens retningsestimat viser seg å være dårligere enn posisjonsestimatet (x, y).
- Maksimal distanse før posisjoneringsfeilen blir for stor synes å være rundt 10 meter i allerede kartlagte omgivelser.
- Maksimal distanse før posisjoneringsfeilen blir for stor er omtrent 6 meter i ukjente omgivelser.

6.1.2 Infrarød sensor

For å komme fram til en modell for den infrarøde sensoren brukes blant annet data fra tidligere arbeid med roboten. I [5] beskrives et fenomen der den infrarøde sensoren gjengir vegger som er parallelle med roboten som skrånende utover. Figur 5.2viser oppførselen til sensoren.

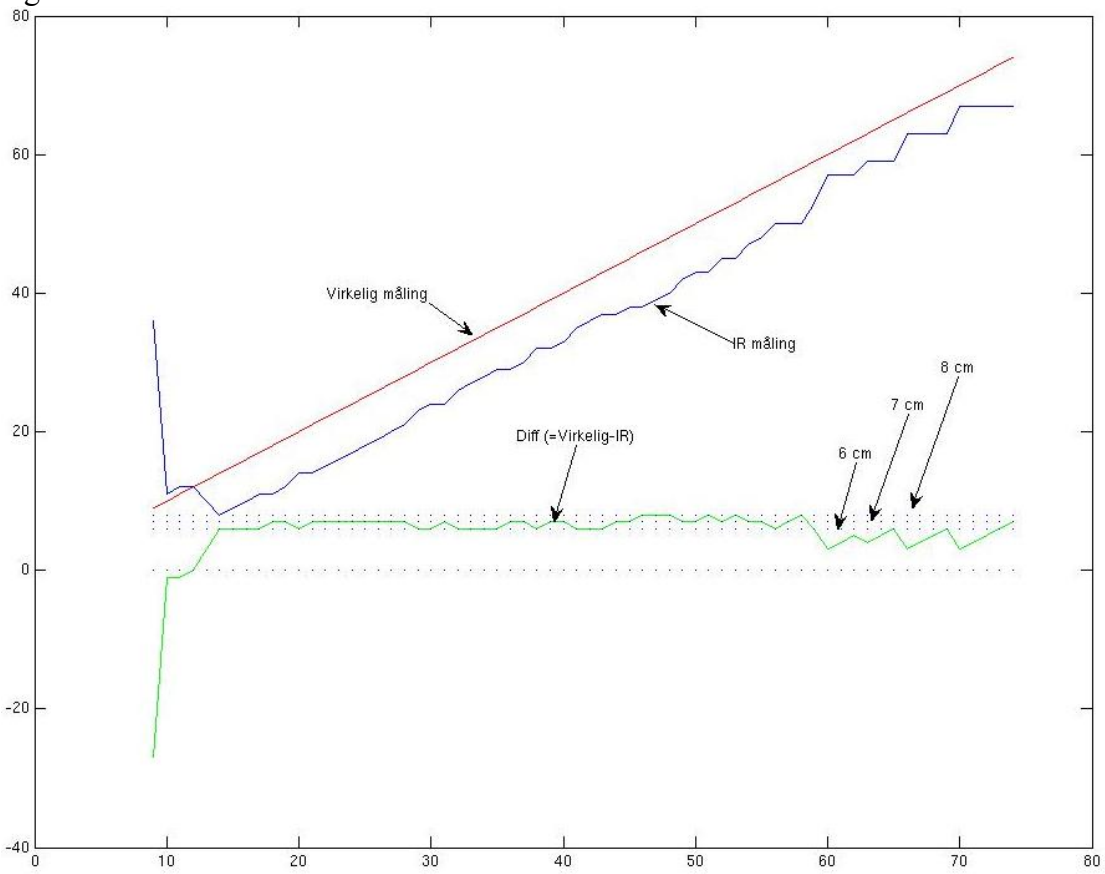


Figur 6.1 Tolkning av parallell vegg

Videre trekkes det fram tre typiske karakteristikker om IR sensoren:

- Store feil på de korteste målingene
- Minkende oppløsning på de lengste målingene
- Ett rimelig konstant avvik i området som ligger imellom ytterpunktene. Muligens en trend i retning av økende avvik opp mot punktet der kvantiseringseffektene overtar.

Karakteristikken er basert på empiriske data fra måling mot en vegg på forskjellige avstander, se Figur 5.3.



Figur 6.2 Modell for infrarød sensor [5]

I det høyre bildet i Figur 3.1 er det tegnet inn en horisontal hjelpelinje som danner en vinkel med

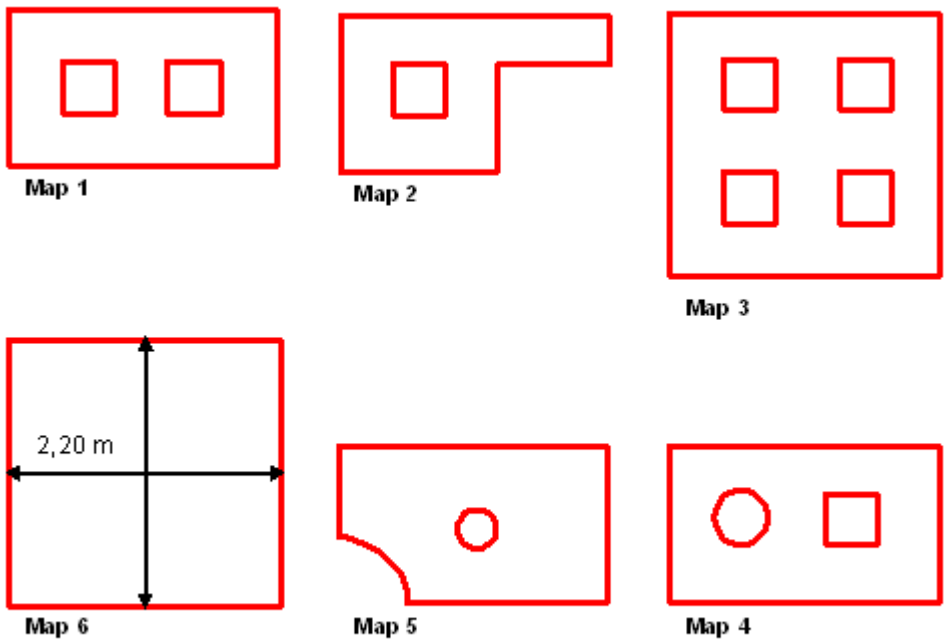
skråveggen. Denne vinkelen er også markert og kan brukes for å beskrive hvor skrått veggen blir oppfattet. Når denne vinkelen er gitt og posisjonen til roboten er kjent kan man ved hjelp av trigonometriske uttrykk danne en relasjon mellom virkelig avstand og sensorens målte avstand. Denne avstanden vil utgjøre målefeilen til sensoren ved en gitt servovinkel. Til sammen gir dette en matematisk modell for fenomenet med skråvegger. I tillegg kan man regne med at vi har normalfordelt hvitt støy på målingene fra den infrarøde sensoren.

6.2 Utgangspunkt/tidligere arbeid

I tidligere arbeid gjort av undertegnende [7] ble det utviklet en enkel simulator som kunne simulere oppførselen til roboten. Input til simulatoren var posisjonsdata, radarvinkel og simuleringsparametere bestående av bredde mellom veggene og startpunkt for simuleringen. Simulatoren returnerte sensorinformasjon i henhold til robotposisjon og radarvinkel. I tillegg var det å tillegge usikkerhet i posisjon og måling.

Det var laget tre labyrinter der en kunne simulere oppførselen til roboten. For å teste ut punktbasert kartlegging ble det laget tre nye simuleringslabyrinter. Figur 12 viser en oversikt over hvordan alle labyrintene ser ut. Figuren viser labyrintene i samme relative størrelse. *Map* 4, *map* 5 og *map* 6 er nye labyrinter. Som en ser inneholder *map* 4 og *map* 5 sirkelformede og buede objekter. *Map* 6 er laget for å teste robotens oppførsel i åpne områder med få landemerker.

Spesielt var det viktig å simulere hvordan det punktbaserte kartet ble tegnet opp med ulike parametere og målefeil. Mange simuleringer ble også kjørt med det kombinerte linje- og punktbaserte kartet. Det henvises til testing for nærmere beskrivelse av gjennomførte tester.



Figur 3: Simuleringslabyrinter

6.3 Implementering

Dette underkapittelet tar for seg hvordan endringer i simulatoren ble implementert. Dokumentering av funksjoner og filer som simulatoren er bygget opp av gjennomgåes.

Kildekoden kan finnes igjen på den vedlagte CD som følger med rapporten. Kommentarer og beskrivelser av funksjonalitet er skrevet på engelsk for å være konsistent med resten av systemet. Det gjør også systemet mer generelt og brukervennlig dersom koden skal videreutvikles i fremtiden.

Simulatoren er programmert i MATLAB og er konsistent med resten av systemet. Fordelen ved å bruke MATLAB er at programmeringsspråket har et høyt abstraksjonsnivå og er objektorientert. Dette gir et oversiktlig bilde av hva som skjer mellom modulene i programmet. Man trenger derfor ikke vite hva som spesifikt skjer inne i hver modul, bare hvordan man gjør funksjonskall mellom de forskjellige modulene. Kildekoden til simulatoren er samlet under mappen ”*simulation*”. Videre følger en oversikt over de filer og funksjoner som brukes i simulatoren.

simRobot.m

Denne filen består av flere funksjoner og skal hovedsaklig ta seg av to hovedpunkter; dynamikken til roboten og kommunikasjonen med resten av rammeverket. Funksjonen simRobot tar inn kommandoer fra det eksisterende brukergrensesnittet, behandler data og lagrer informasjonen om hva som eventuelt skal sendes tilbake senere. Når rammeverket senere kaller på respons fra roboten kjøres simRobot og returnerer verdier som var satt tidligere. Dette er konsistent med hvordan den virkelige roboten ville ha reagert på forespørsler fra det overordnede systemet. I tillegg skaper dette et fornuftig abstraksjonsnivå mellom simuleringsprogrammet og resten av rammeverket. Håndtering av posisjonsfeil skjer også i simRobot. Simulatoren bruker samme protokoll som er implementert i roboten.

simMap.m

SimMap administrerer valg av labyrinter og målefeil på den infrarøde sensoren. Funksjonen kalles enten fra simRobot som forespør verdien på infrarød sensor eller så kalles den fra LegoGUI som vil sette verdier for målefeil. I simMap settes blant annet størrelse på labyrintmodulene, startpunktet for roboten og andre variable for feilhåndtering. Grunnleggende filtreringer av sensordata gjøres også.

mazeX.m

Dette er til sammen seks filer der X={1,2,3,4,5,6}. Filene administrer de ulike labyrintene som benyttes i simulatoren. Funksjonene i filene setter sammen labyrintmoduler til hele labyrinter og består hovedsakelig av mange if – else løkker. Filene kan også betraktes som et mellomledd mellom simuleringsroboten og generering av kunstige data fra avstandssensorer.

@simrobotdd

Mappe for objektet simrobotdd. En instans av simrobotdd blir opprettet ved simulering.

Filer i mappen *lib*

I denne undermappen finner vi de enkelte modulene som kan settes sammen til hele labyrinter. Mappen består av følgende filer:

- *corner.m*
- *cornerC.m*
- *corners_Map5*
- *deadEnd.m*
- *Parallel1.m*
- *Parallel1C.m*
- *Parallel2.m*
- *Parallel2C.m*
- *Parallel3.m*
- *Tcross.m*
- *TcrossC.m*
- *X.m*

Navnet på disse filene representer de enkelte modulene en gitt labyrint er bygget opp av. De enkelte filene kalles på av mazeX.m når en forespørsel om sensordata foreligger. Filene tar inn parametere som er viktige for å beregne riktig sensorverdi, det vil si avstanden til veggen som det måles mot. Disse parametrene er posisjonsdata fra roboten, bredde mellom labyrintveggene, eventuelt retningen på åpninger i modulen og avstandsfeil. Det er i disse filene den faktiske sensorinformasjonen blir generert som brukes i simulatoren.

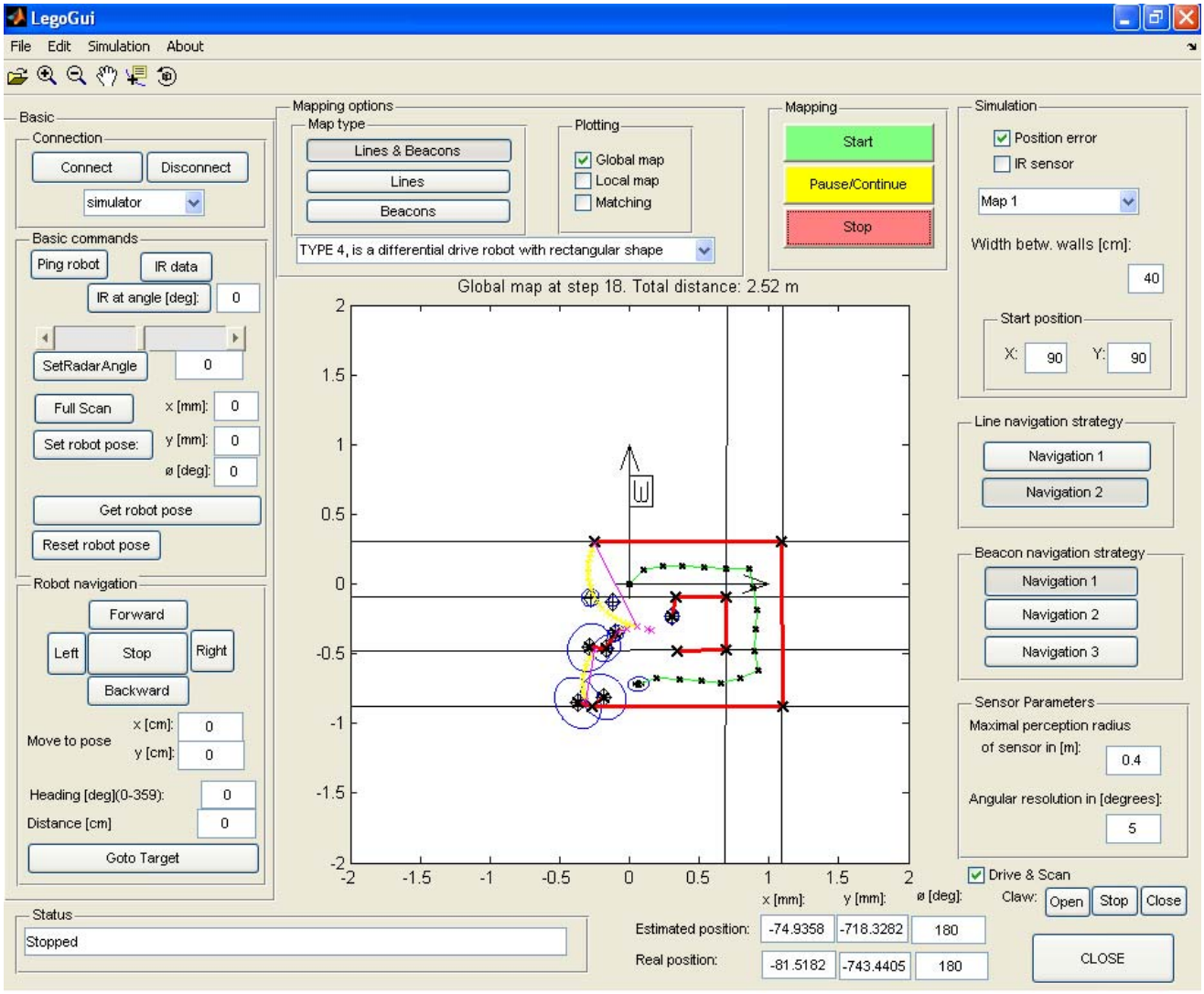
6.4 Brukergrensesnitt

Fra tidligere arbeid har det blitt laget flere brukergrensesnitt med GUIDE i MATLAB [6] [7]. Brukergrensesnittet har blitt utviklet fra å være en eneste stor startknapp som igangsatte kartleggingen til å omfatte høynivåfunksjoner og integrert plotting av kart. Figur 5.4 viser hva som er lagt til av innstillinger og valg i brukergrensesnittet.

En oppsummering av hovedpunktene for det nye brukergrensesnittet presenteres her, mens det henvises til vedlegg for en brukermanual over simulatoren og brukergrensesnittet.

Endringer som er gjort med brukergrensesnittet i denne oppgaven:

- Det globale kartet ble tidligere plottet i et eget vindu, men er nå integrert i selve GUI'en og kalles *mainPlotWindow*.
- Nye *pushbuttons* er laget for all ny funksjonalitet i systemet. Herunder; kartleggingsmetoder, nye lavnivåfunksjoner på roboten, samtidig kjøring og kartlegging.
- Det skilles mellom estimert robotposisjon som inneholder målefeil og virkelig robotposisjon som er en intern datastruktur i simulatoren. Disse data presenteres i brukergrensesnittet.
- En menylinje er lagt til øverst for å skjule innstillinger som ellers ville tatt plass i brukergrensesnittet.
- Det er lagt til en *Toolbar* for behandling avplot.



Figur 6.4: Nytt brukergrensesnitt

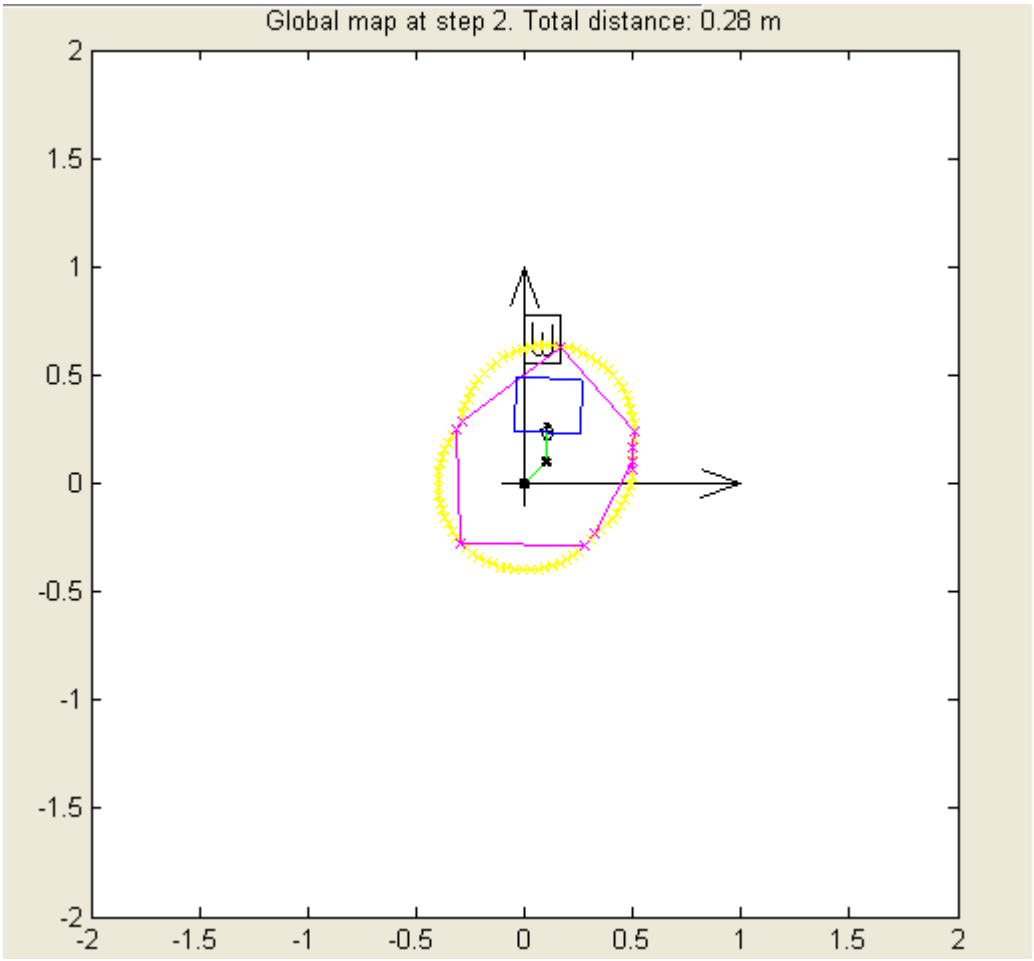
7 Navigering og ruteplanlegging

7.1 Definering av ukjente områder.

Det er implementert en ny algoritme for å detektere ukjente områder. Metoden baserer seg på egenskaper ved sensorene til roboten. En avstandssensor med en definert maksimal måleavstand som kan roteres 360 grader om hovedaksen til roboten danner en sirkel rundt roboten. Denne sirkelen kan defineres til å være den maksimale oppfattelsesradiusen til roboten ved en gitt posisjon. Roboten kan altså ikke ”se” lengre enn denne avstanden uansett retning.

Når posisjonen til roboten endrer seg, dannes en ny sirkel med origo i den nye robotposisjonen og radius lik den maksimale oppfattelseslengden. Området innenfor sirkelen er kjent område og området utenfor definerer ukjente omgivelser. Etter hvert som roboten oppdager nye vegger og hindringer i omgivelsene tegnes disse inn i det globale kartet. Da må også robotens bilde av ukjente områder oppdateres. Sirklene som defineres rundt roboten blir derfor delt opp i sirkelbuer som begrenses av vegger, hindringer og sirkler fra andre robotposisjoner.

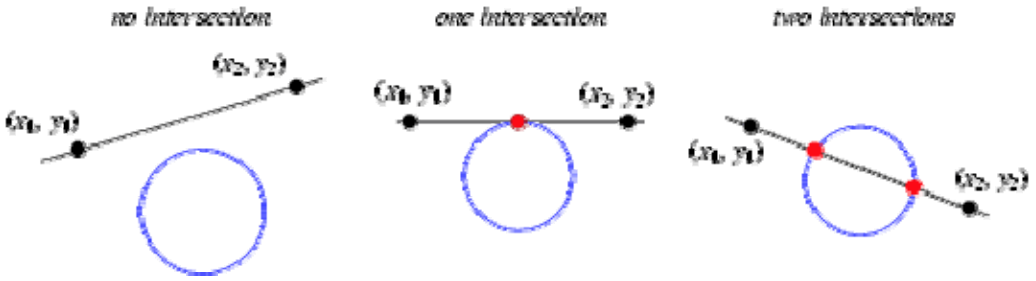
Figur 13 viser hvordan algoritmen fungerer i løpet av de to første stegene i en kartlegging. Roboten har så langt ikke funnet noen landemerker. Det gule punktene beskriver skillet mellom kjent og ukjent område for roboten. En kan se at de gule punktene er sirkler som ligger delvis over hverandre. De lilla linjesegmentene beregnes ut fra de gule punktene og brukes for å definere åpninger som roboten må utforske senere.



Figur 5: *getUnknownAreas*

7.1.1 Teori

For å detektere linjer og segment som ligger innenfor oppfattelsesradius ble det brukt kjent geometri om sirkler og linjesegment. Slik teori finnes for eksempel hos [27] og Figur 9 viser de tre situasjonene som oppstår ved krysning av sirkler og linjesegment.



Figur 6: Sirkel/segment geometri

Matematisk bakgrunn for å finne krysningspunkter som brukes i *getUnknownAreas*:

$$d_x = x_2 - x_1 \tag{6}$$

$$d_y = y_2 - y_1 \tag{7}$$

$$d_r = \sqrt{d_x^2 + d_y^2} \tag{8}$$

$$D = \begin{vmatrix} x_1 & x_2 \\ y_1 & y_2 \end{vmatrix} = x_1 y_2 - x_2 y_1 \tag{9}$$

$$x = \frac{D d_y \pm \operatorname{sgn}^*(d_y) d_x \sqrt{r^2 d_r^2 - D^2}}{d_r^2} \tag{10}$$

$$y = \frac{-D d_x \pm |d_y| \sqrt{r^2 d_r^2 - D^2}}{d_r^2}, \tag{11}$$

$$\operatorname{sgn}^*(x) \equiv \begin{cases} -1 & \text{for } x < 0 \\ 1 & \text{otherwise.} \end{cases} \tag{12}$$

Behandling av slik geometri ble ikke funnet i MATLAB versjon 2007a/b. Det ble søkt på Internett etter ferdige implementerte løsninger og det ble funnet flere aktuelle kodelinjer for MATLAB. Blant annet ble det funnet en samling av filer for behandling av geometriske figurer i det to dimensjonale plan [28]. Filsamlingen viste seg å være nyttig da det ikke finnes noe liknende ferdig implementert i MATLAB fra før. I funksjonen *getUnknownAreas* som er implementert av undertegnede er kalles funksjonen *iscircle.m* [28] som utfører beregningene i ligning (13)-(14).

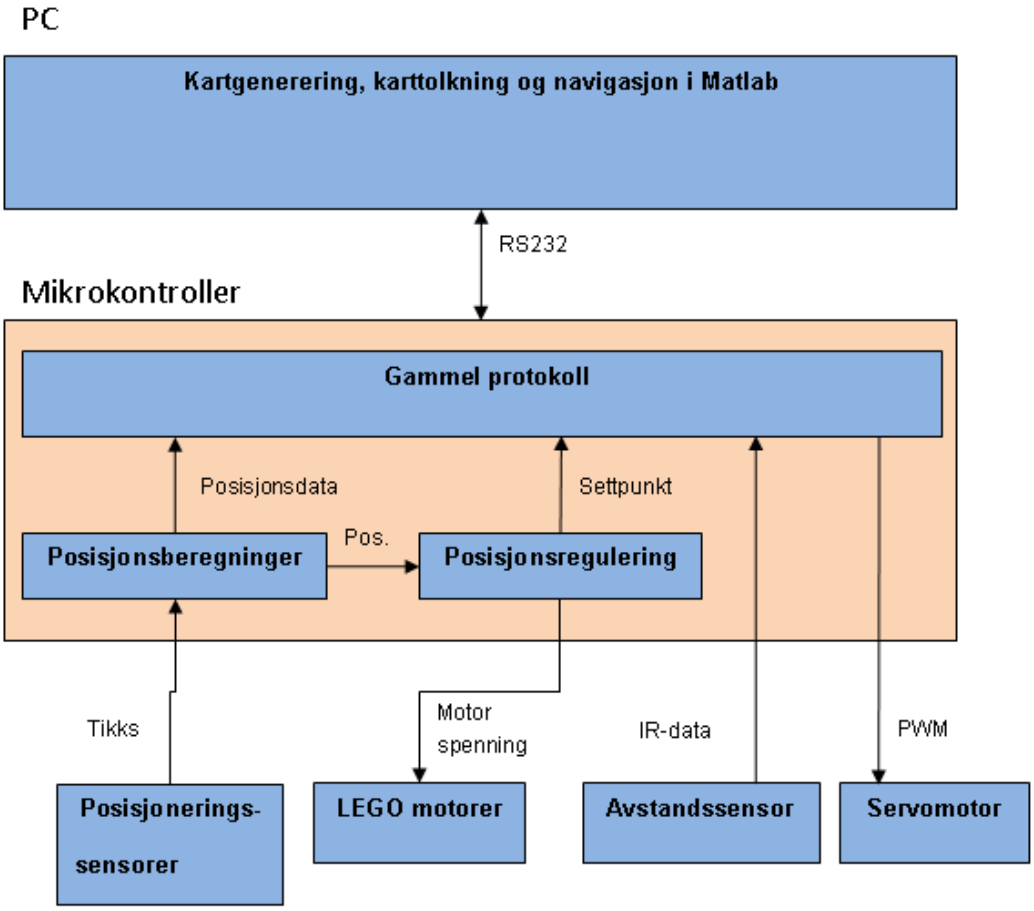
7.1.2 getUnknownAreas

Funksjonen tar inn det globale kartet og trekker ut linesegmentene. Disse blir slått sammen med segmentene fra punktkartet. Sammen med tidligere robotposisjoner blir disse segmentene brukt for å generere en matrise kalt *cT* som holder oversikt over de ukjente områdene i kartet sett fra robotens posisjonsliste. For nærmere beskrivelse se vedlegg på CD.

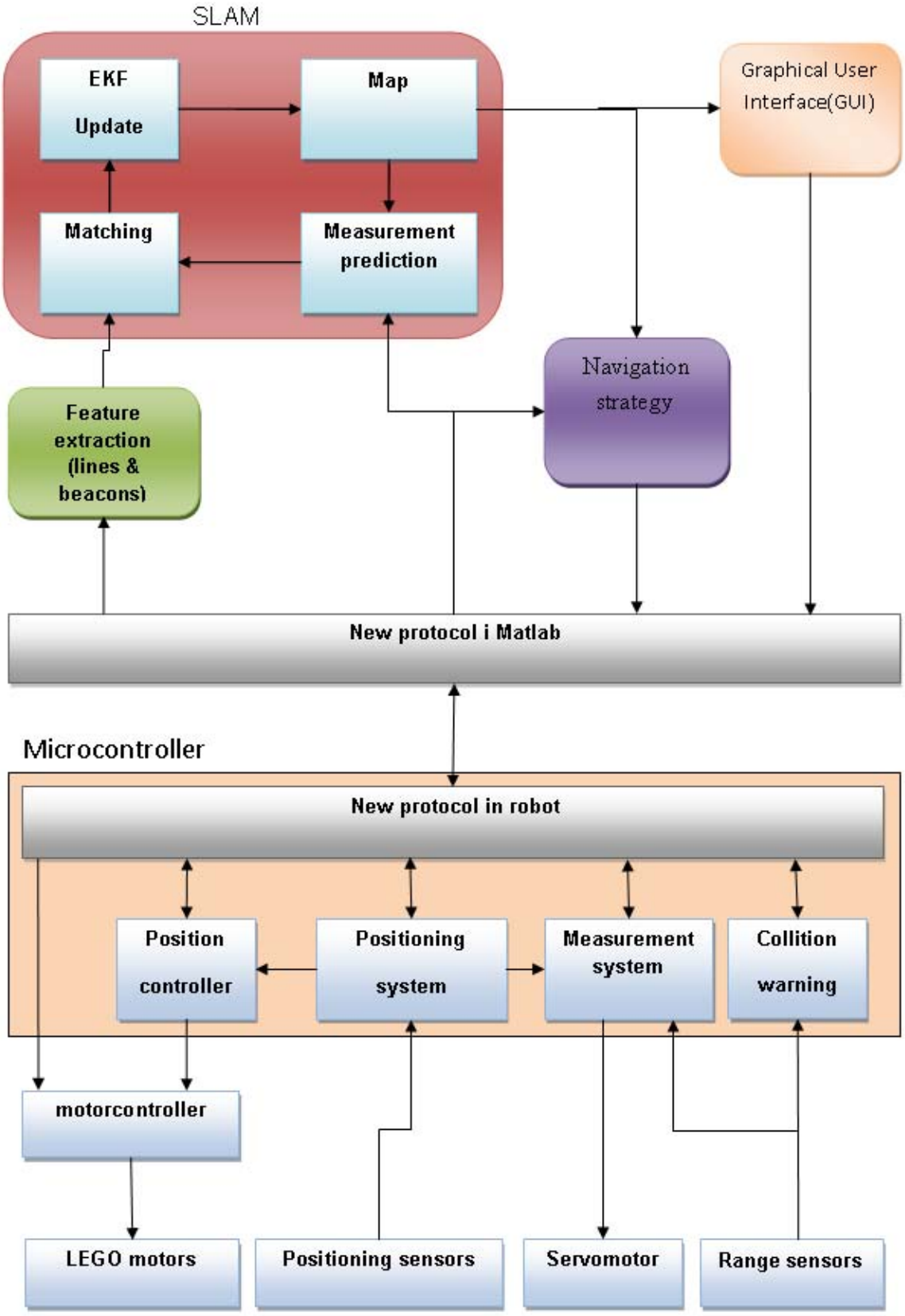
8 Oversikt over systemet

8.1 Arbeidsfordeling

Her vises det hvordan de ulike delsystemene henger sammen.



Figur 7: Oversikt over det gamle systemet



Figur 8: Oversikt over det nye systemet

8.2 Kommunikasjonsprotokoll

Robotens protokoll ble totalt omskrevet fra tidligere arbeid. Den tidligere protokollen fungerte tilfredsstillende, men det var behov for å få gjort flere oppgaver på roboten i sanntid og dette krevde mer avansert datautveksling mellom robot og MATLAB. Den nye protokollen ble laget i samarbeid med Johannes Schrimpf [1] og Paul Sverre Kallevik [2]. Det ble gjennom hyppige møter funnet fram til flere problemstillinger det var vesentlig å ta hensyn til i forbindelse med robotens nye funksjonalitet.

Nye funksjoner på roboten som kunne skape problemer:

- Automatisk måling: MATLAB mottar måleverdier fra roboten ved diskrete intervaller under kjøring.
- Automatisk 360° skann: Antall måleverdier kan variere på grunn av støy og tap av data.
- Kollisjonsvarsel: MATLAB mottar et varsel om kollisjon.

Et eksempel på en slik situasjon kan være at MATLAB forventer måledata fra avstandsensor under kjøring og et kollisjonsvarsel sendes fra robot. I beste fall blir data mottatt i MATLAB tolket som en ugyldig verdi og forkastet, i verste tilfellet blir det tolket som en gyldig måleverdi og gir falske landemerker i kartet.

8.2.1 Meldingsbasert protokoll

I den gamle protokollen ble alle data etterspurt av MATLAB eksplitt mens den nye protokollen er meldingsbasert. Data som sendes over det serielle grensesnittet starter med en header-pakke etterfulgt av spesifiserte antall datapakker i henhold til protokoll. Slik unngår man situasjoner beskrevet i forrige avsnitt der datapakker kan mistolkes.

Endringer som måtte gjøres i MATLAB når roboten kjører:

- MATLAB går inn i en tilstand for å motta data fra robot. Når for eksempel header for måledata registreres blir de neste seks bytes som mottas tolket som sensornummer, x- og y-verdier til et punkt i rommet og lagt inn i en matrise for videre behandling senere.
- Headere som mottas blir lagt i en prioritetskø og behandlet i henhold til tabell xx.
- Registreres det ugyldige headere må disse behandles særskilt. Det er da mest sannsynlig at bytes ikke har blitt sendt fra robot eller at de har gått tapt over det serielle grensesnittet. MATLAB håndterer slike situasjoner ved å midlertidig gå ut av lyttetilstanden og tømme inputbuffer på seriellporten. Dette er viktig slik at ikke det ligger ubehandlet data på seriellporten som senere kan mistolkes igjen. Videre skrives det ut et varsel i kommandolinjen i MATLAB og i brukergrensesnittet om at ugyldig header har blitt mottatt. MATLAB går så tilbake til lyttetilstand for å motta nye data fra roboten.

Oversikt over protokollene finnes i vedlegg.

8.3 Optimalisering av MATLAB kode og sanntidskrav

Etter hvert som rammeverket for kartlegging og navigering blir større og større, øker kompleksiteten og utførelsen av kode begynner å bli ta tid. Programmet skal gjøre stadig mer per tidssteg. Kjøretiden er også en medvirkende faktor når det gjelder tilfredsstillende kommunikasjon mellom robot og MATLAB. En av bakdelene med programmering i MATLAB som er nevnt tidligere er nettopp at programmet kan bli tregt ettersom mer og mer funksjonalitet legges til.

En gjennomgang av tidligere kode viste at det var behov for en trimming av koden i rammeverket. Spesielt i koden fra tidligere arbeid var det nødvendig å endre koden slik at beregningene gikk fortere. Det ble funnet flere artikler om optimalisering av kode i MATLAB, deriblant [29] som tar for seg hastighet i MATLAB.

Fra MATLAB 5.0 og utover finnes det et verktøy kalt ”profiler” som kan hjelpe til med å finne flaskehalser i koden. Verktøyet kan blant annet generere kjøretidsrapporter for MATLAB kode og ble ofte benyttet for å finne flaskehalser under programutviklingen.

9 Tester og resultater

9.1 Tester i simulator

Testene i simulatoren ble gjort for å få en effektiv utviklingsfase av nye algoritmer. Man kunne på en enkel måte verifisere om algoritmene fungerte som tiltenkt etter utallige små endringer i programmet.

Simulatortester ble gjort med og uten målefeil. Tester gjort uten målefeil ble mye brukt i starten for å sjekke at labyrintene i simulatoren og andre algoritmer var implementert riktig. Simulering med målefeil ble gjort for å prediktere hvordan roboten og kartleggingen ville oppføre seg i en virkelig labyrint.

Figurene som presenteres her viser simulering med den siste versjonen av kartleggings- og navigasjonsalgoritmene og kan derfor betraktes som resultatet av arbeidet med kildekoden. Det henvises til appendiks for figurer som viser simulering med kildekode under utvikling.

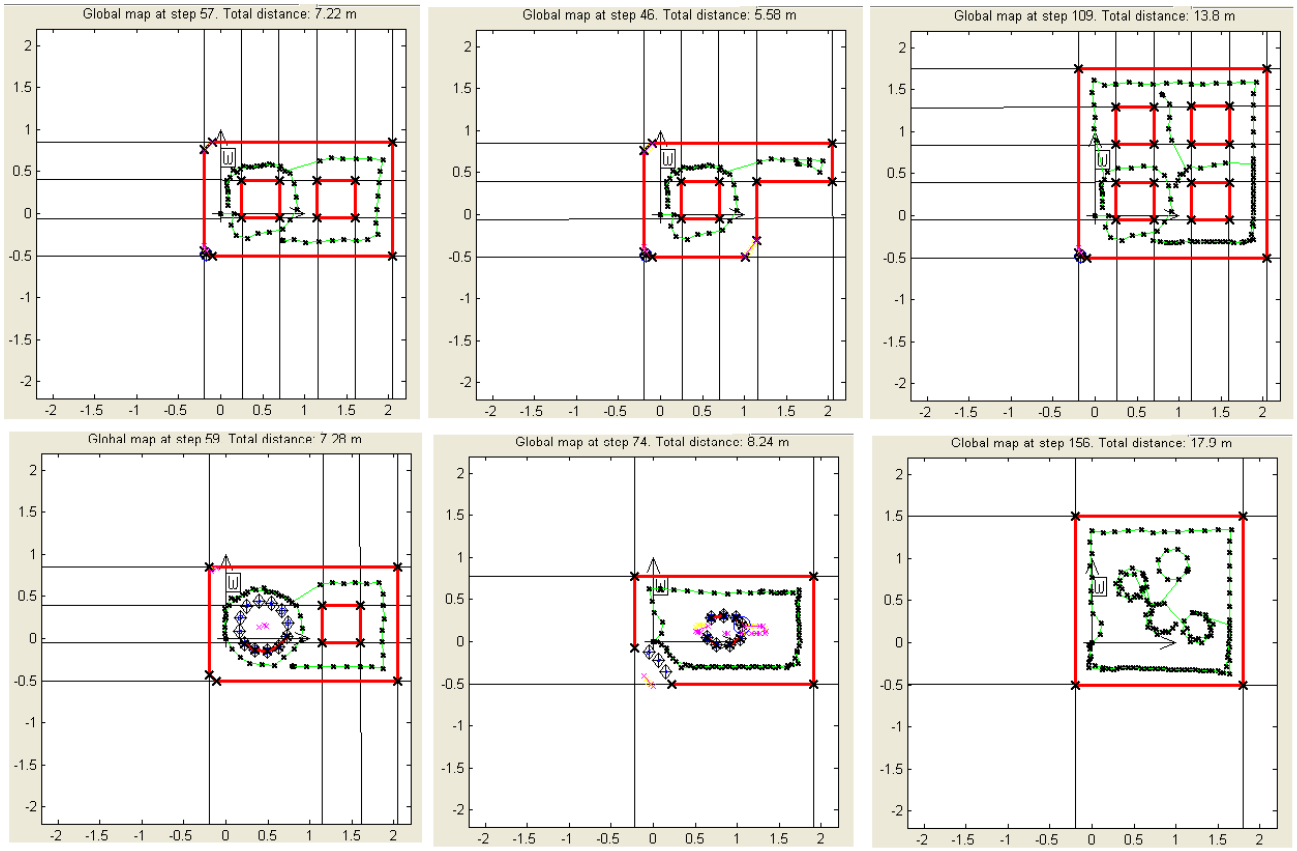
9.1.1 Kartlegging uten målefeil

En samlet simulering av alle labyrinter uten målefeil ble gjort for å se hvordan kartleggingen fungerer i et ideelt tilfelle.

Testene i Figur 17 er gjort med standardinnstillinger i simulatoren:

- Startpunkt (x, y)=(20, 50)
- Bredde mellom vegger: 45 cm
- Målefeil: Ingen.

Siden disse simuleringene er gjort uten målefeil er resultatene deterministiske og det mulig å gjenskape de med de nevnte initialverdier.

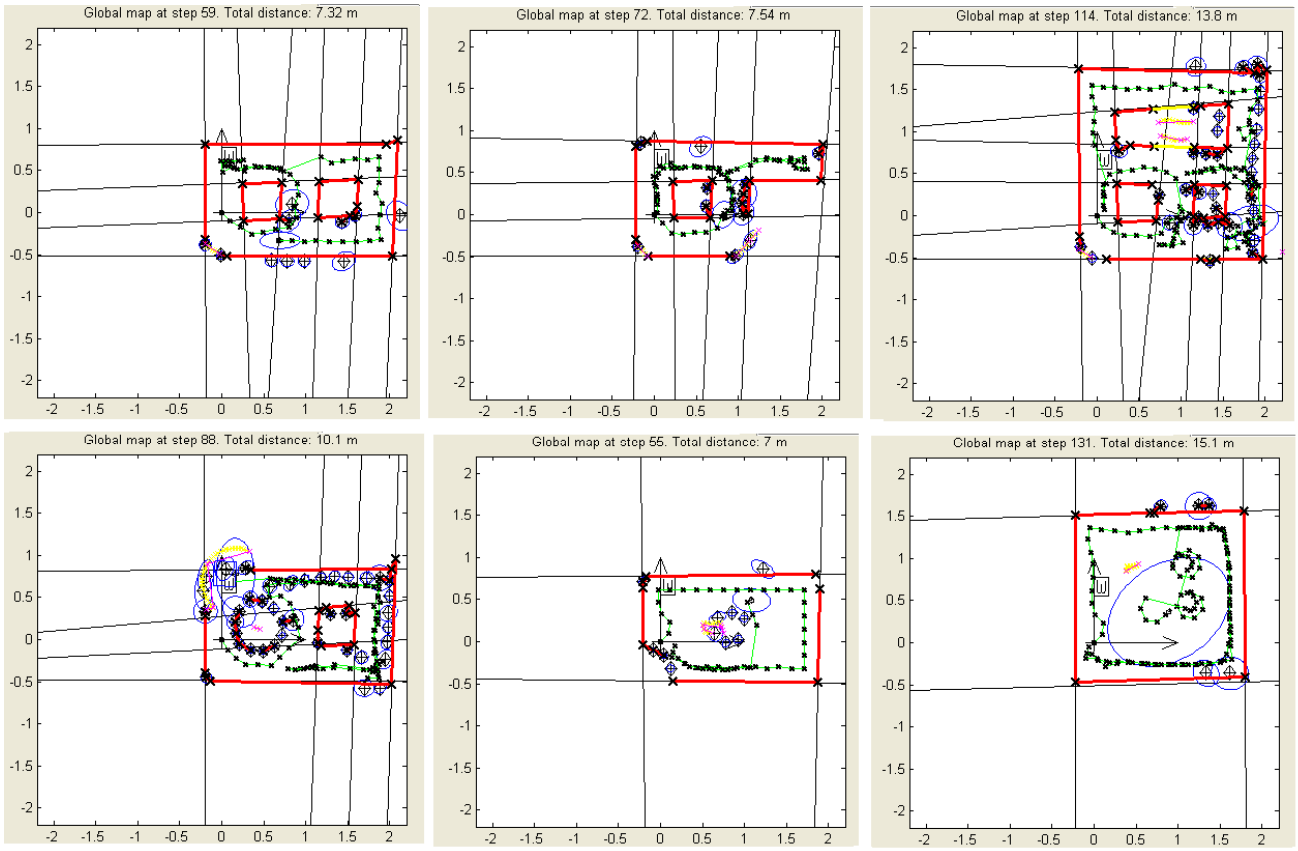


Figur 9: Simulering uten målefeil

9.1.2 Kartlegging med målefeil

Kartlegging med målefeil skal prediktere en virkelig kartlegging. Figur 18 viser simulering av kartlegging i alle seks labyrinter.

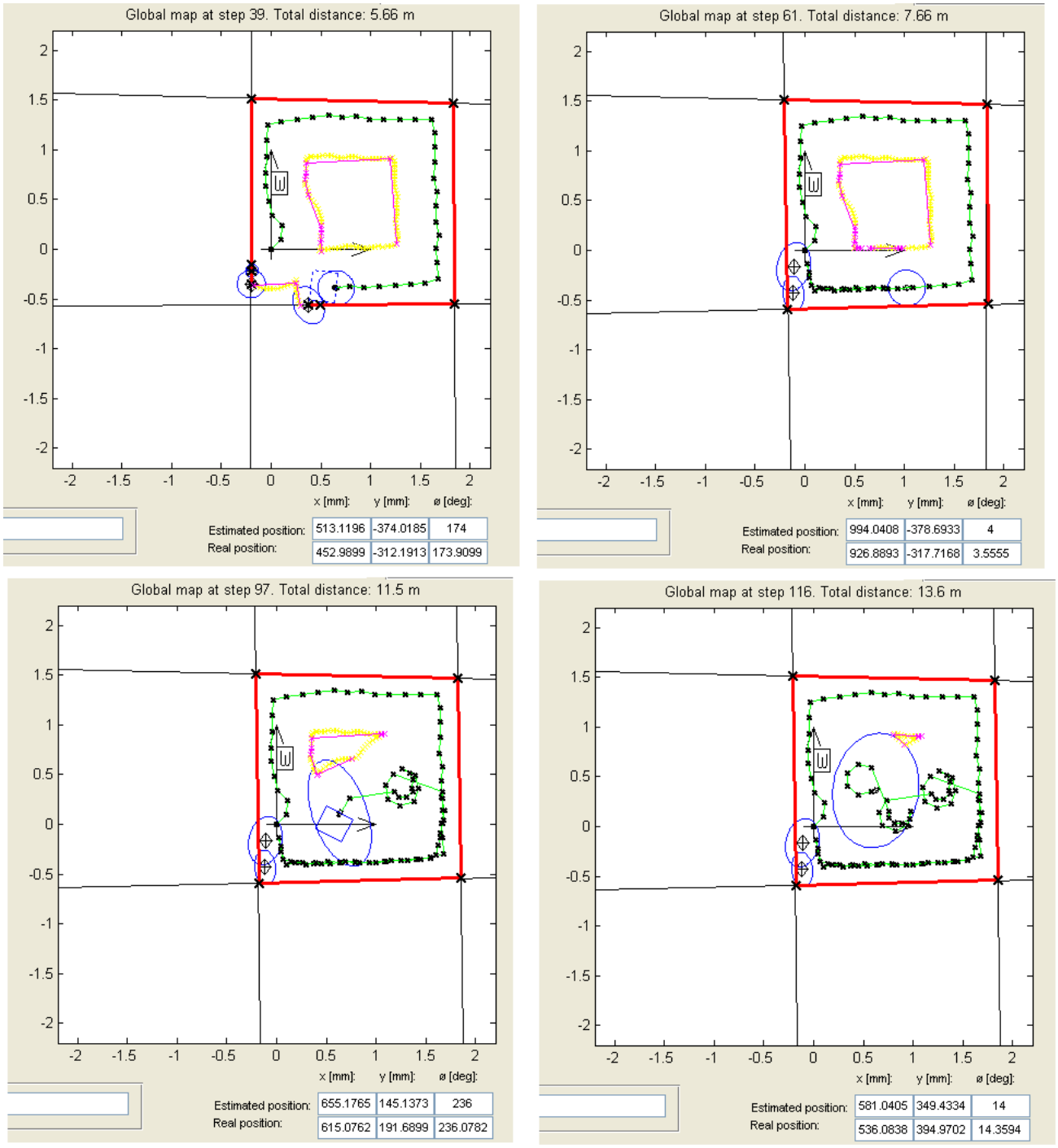
- Startpunkt (x,y)=(20,50) .
- Bredde mellom vegger: 45 cm.
- Målefeil: En stokastisk normalfordeling med forventningsverdi μ og varians σ^2 brukes i simulatoren.
    - Posisjon: x- og y-posisjon tillegges med μ=0 og σ^2=0.2 cm ved hvert diskrete tidssteg under forflytning. Retningen tillegges med μ=0 og σ^2= 0,0873 radianer (5°) ved hver retningsendring.
    - Avstandssensor: En feil ved måling av vinkelen på parallelle vegger tillegges med μ=0 og σ^2= 0,122 radianer (0.7°).



Figur 10: Simulering med målefeil

9.1.3 Posisjonering og navigering

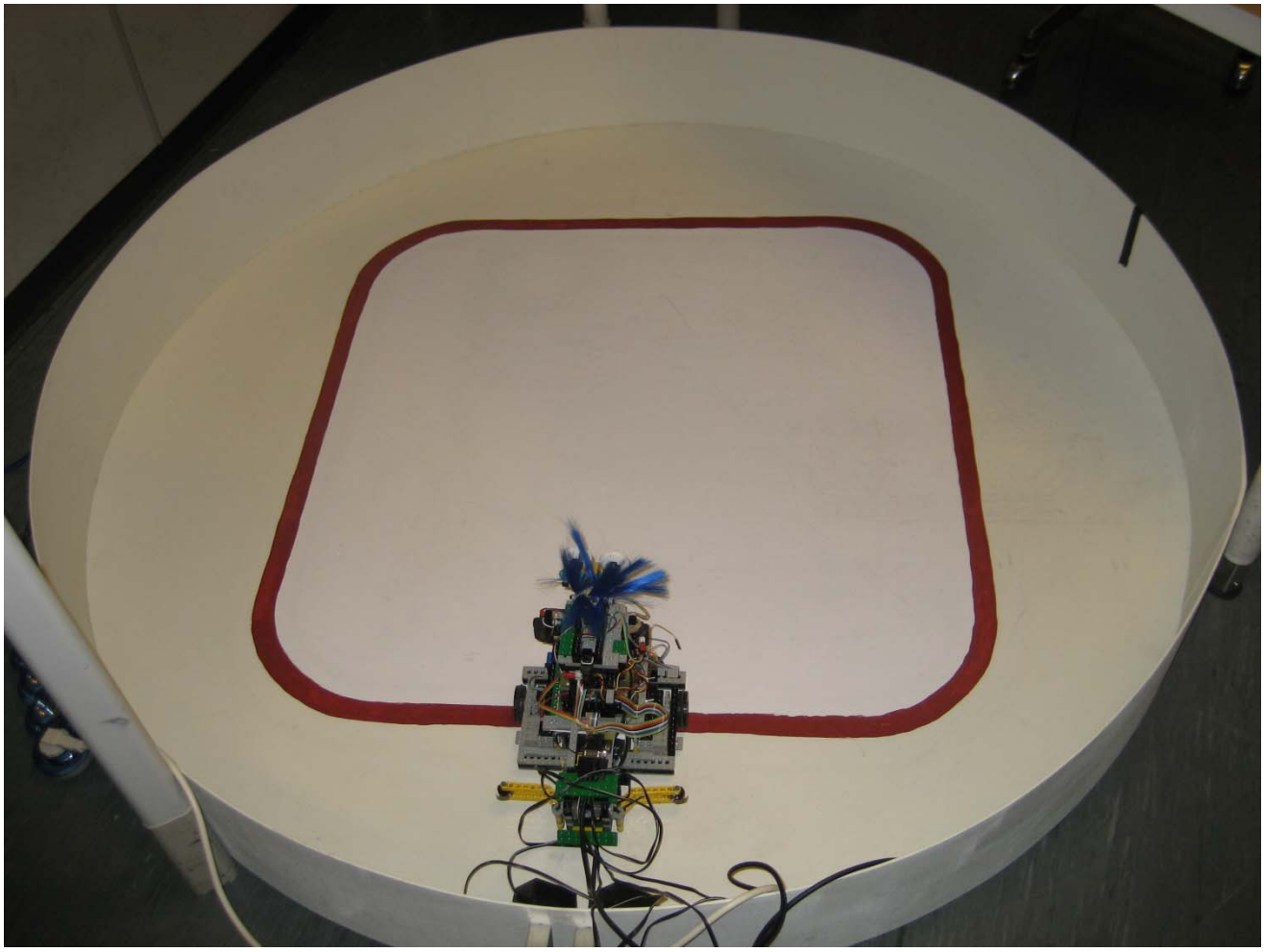
En simulering i *map* 6 ble gjort for å observere hvordan posisjonsestimatet endrer seg når algoritmene har lite sensordata fra landemerker i omgivelsene. Simuleringsparametere er de samme som i forrige avsnitt.



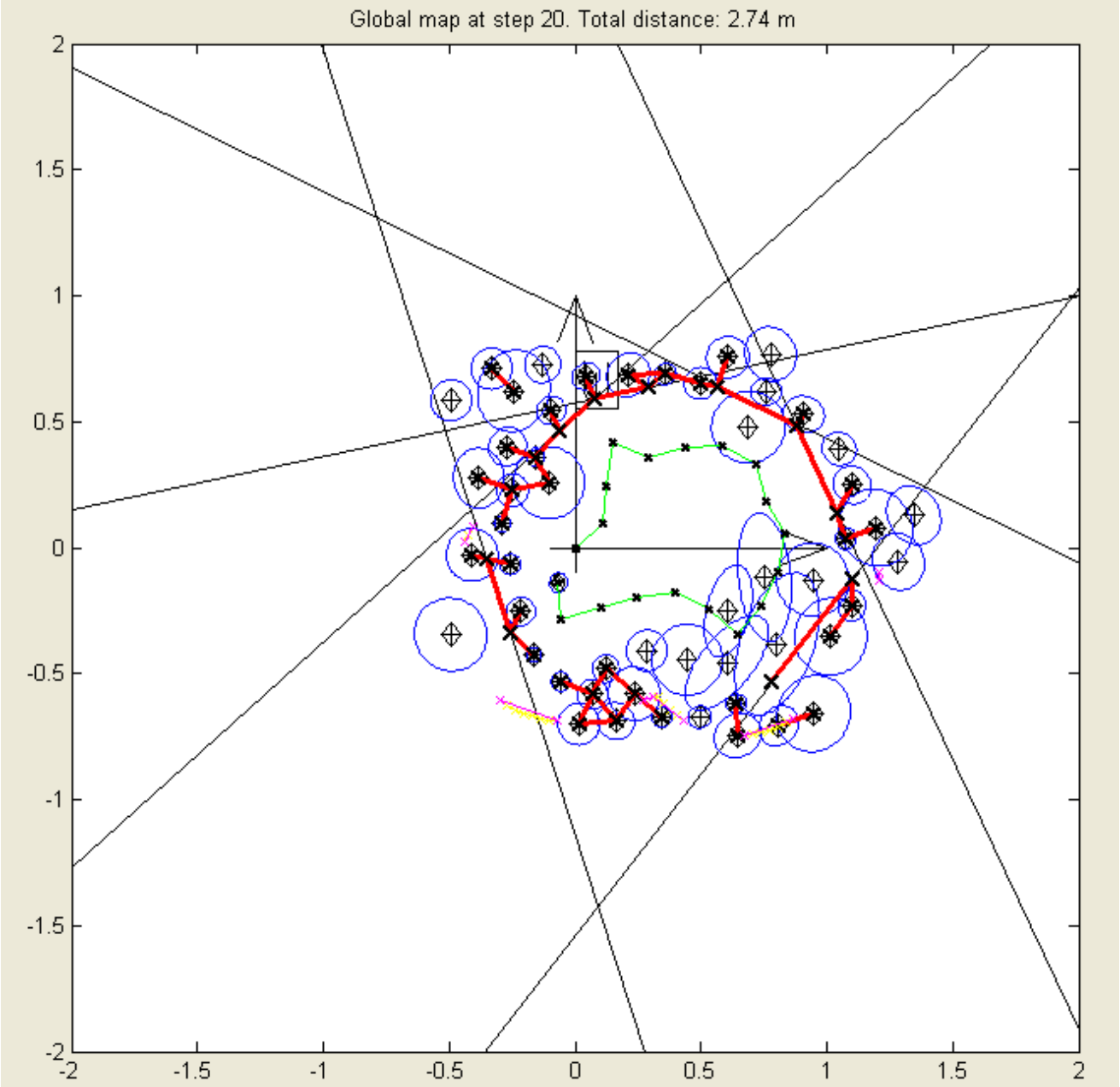
Figur 11: Simulering av posisjonering og kartlegging

9.2 Tester i en fysisk bane

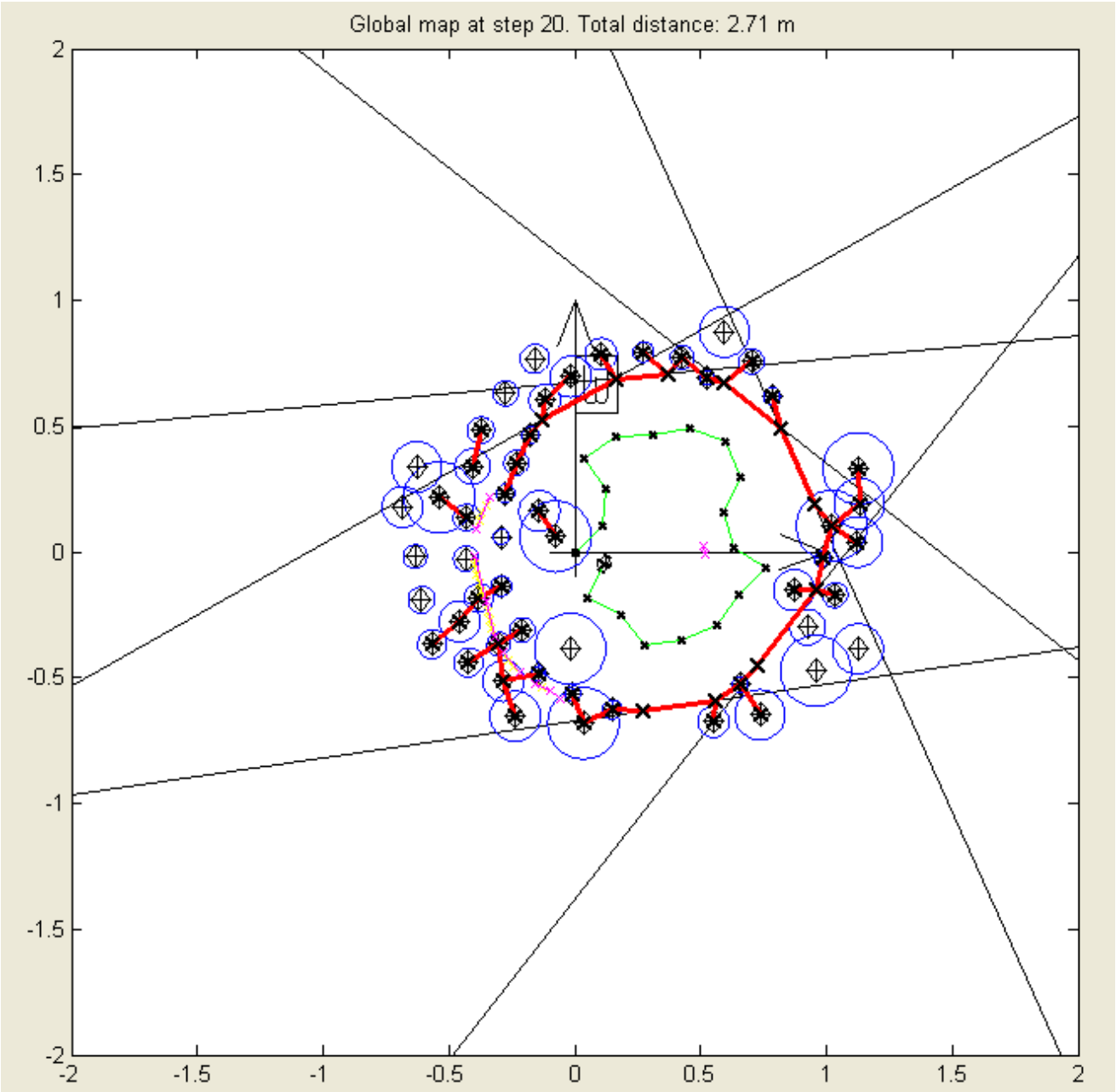
En sirkulær bane ble brukt for å teste kartlegging og navigasjon. Se Figur 21 viser hvordan testbanen så ut. Den røde linjen har ingen påvirkning på roboten i noen sammenheng.



Figur 12: Testbane



Figur 13: Test i sirkulærbane (1)



Figur 14: Test i sirkulær bane (2)

Banen kartlegges, men kartet har dårlig kvalitet. Dette kommer hovedsakelig av et dårlig posisjoneringssystem.

10 Diskusjon

10.1 Posisjonering

I denne oppgaven har det blitt gjort forbedringer i software i forbindelse posisjonsestimatet og det har blitt laget en ladestasjon som roboten kan returnere til.

Ved å tilføre den manglede tilstandsestimering av robotposisjon i software har blitt mulig å estimere posisjonen mer nøyaktig. En usikkerhetsellipse som tegnes opp i kartet gjør at man får et visuelt bilde av usikkerheten i posisjonen. Som en kan forvente seg av teorien bak *EKF-SLAM* øker usikkerheten når roboten kjører i omgivelser med få landemerker og minker igjen når algoritmene har flere observasjoner av landemerker.

Ved simulering ble det antatt en bestemt usikkerhet på målinger av posisjon. Denne usikkerheten ble satt til å være av gausisk natur, normalfordelt med forventingsverdi lik null og estimert varians ut fra tidligere empiriske resultater. Imidlertid er det funnet fra testing at disse antagelsene i noen tilfeller ikke er representativt for den virkelige verden. Spesielt er det andre og større forstyrrelser ved beregning av posisjon. Fra testene i Figur 20 og Figur 21 ser man tydelig hvordan posisjonen påvirker kartleggingen.

10.2 Kartlegging

Integrering av linje- og punktbasert kart gjør det mulig å detektere objekter med forskjellig geometri mer nøyaktig enn tidligere. Likevel finnes det flere problemer med denne implementeringen. Det blir mange parametere å holde styr på når kartlegging skal foretaes. Mange kan settes til standardinnstillinger som foretar kompromisser mellom ulike egenskaper i kartet, mens andre må tilpasses noe ettersom hvilke omgivelser som skal kartlegges. For eksempel må bredde og lengde på rektangelet som definerer kjørbart område foran roboten tilpasses noe for å få en mest mulig effektiv kartlegging.

10.3 Navigasjon

Ved kjøring i områder uten mange hindringer/landemerker fungerer den nye *getUnknownAreas*-algoritmen bedre enn den som var utviklet i forrige prosjekt. Algoritmen fra tidligere var bundet opp til linjekart og var ikke laget for å fungere på andre typer kart som for eksempel et punktbasert kart. Den nye algoritmen var ikke bundet av at kartet det navigeres i må ha spesielle egenskaper og er således mye mer robust og fleksibelt enn tidligere.

11 Konklusjon

Utgangspunktet for oppgaven var en Legorobot med sensorer for posisjonsmåling og avstandssensorer som kunne kartlegge enkle labyrinter. Kartene ble laget gjennom en *SLAM*(*Simultaneous Location And Mapping*) applikasjon i MATLAB.

Målet med denne oppgaven var hovedsakelig å forbedre robotens posisjonerings- og kartleggingsevner. For å oppnå dette skulle en ladestasjon for roboten utvikles og eksisterende kartleggingsprogramvare videreutvikles. Det var i tillegg nødvendig å studere forbedringer i navigasjon.

En ladestasjon som roboten kunne kjøre inn i og koble seg til ble laget. Ladestasjonen besto av trevegger og et utspring med metallskinner som tillot lading av robotens batterier. I programvaren ble det laget en algoritme for å kjøre tilbake til stasjonen når posisjonsestimatet ble for stort. En utvidelse i programvaren ble gjort for å estimere posisjon.

Videreutvikling av kartleggingen i dette prosjektet baserte seg på en eksisterende algoritmesamling for *SLAM* kartlegging. Dette systemet ble modifisert slik at det kunne tegne opp et kombinert linje- og punktbasert kart. Ved hjelp av simulering og testing ble resultatet bra.

Den eksisterende navigasjonsalgoritmen fungerte tilfredsstillende for labyrintene i tidligere arbeid, men det var behov for en utvidelse. En ny metode for å finne ukjente områder har blitt laget. Metoden baserer seg på egenskapene ved avstandssensorene og observasjonsradius. Denne metoden gjør at navigeringen ikke er avhengig av å finne åpninger mellom linjesegment slik det gjøres i tidligere arbeid, men nå kan finne ikke-kartlagte områder helt uavhengig av landemerker. Dette gjør at alle typer labyrinter kan kartlegges fullstendig med den gitte navigasjonsalgoritmen. I tillegg er det gjort en utredning om andre navigasjonsstrategier som kan implementeres i videre arbeid.

Den største begrensningen i kartleggingen var fra tidligere arbeid var tolkning av kartet og posisjoneringssystemet. Karttolkningen er blitt forbedret og fungerer på en bedre måte enn før, mens det fortsatt ligger begrensninger i posisjonsbestemmelsen. Det har blitt gjort endringer i programvaren som er tett opp til hva som er mulig å få til uten å måtte omforme hele posisjoneringssystemet. Derfor henvises det først og fremst i videre arbeid til å bruke eksterne sensorer som for eksempel kamera til forbedring av posisjonsestimatet.

12 Videre arbeid

Kapittelet oppsummerer hva som har blitt gjort i rapporten og gir forslag på hva som bør gjøres videre i arbeidet med roboten.

12.1 Sensorer og hardware

Roboten har følgende sensorer påmontert:

- 4 infrarøde avstandssensorer
- 2 sensorer for odometrimåling

I litteraturen om robotnavigasjon skilles det mellom to hovedtyper av sensorer for observasjonsmålinger:

- Avstandssensorer, herunder:
    - Infrarøde sensorer
    - Ultralyd baserte sensorer
- Kamerasyn

Ut fra dette er det altså tre typer sensorer det kan være aktuelt å bruke i videre arbeid; ultralydsensorer, infrarøde sensorer og kamera. Ultralyd har vært prøvd ut i tidligere arbeid med Legoroboter ved NTNU og fungerte ikke tilfredsstillende [5]. Det bør vurderes å utrede mer om bakgrunnen for det. Ultralydsensorer har tross alt fordeler på flere områder i forhold til lysbaserte avstandssensorer, blant annet rekkevidde på målinger. Det brukes også ultralyd i mange andre lignende applikasjoner med suksess.

Infrarøde sensorer brukes på roboten og fungerer bra innenfor rammene i oppgaveteksten. Sensorene har god vinkeloppløsning og avstandsmålingene er nøyaktige innenfor operasjonsområdet. Problemet med sensorene er at operasjonsområdet kan bli for snevert, for eksempel er de påmonterte sensorene bare i stand til å gjengi avstanden i området 10~80 cm. Nærmere beskrivelse av de påmonterte sensorer og hardware finnes i [1] og [2].

Kameraovervåkning av roboten ble inngangsatt i et annet prosjekt tilknyttet denne rapporten, men ble ikke fullført. Samtaler undertegnede hadde med prosjektansvarlig for kameraovervåkning av roboten virket meget lovende. Det er all grunn til å tro at innføring av kamera for å posisjonsregulering av roboten vil være et viktig arbeid i fremtiden. Bruk av kamerasyn er et aktuelt forskningsområde innenfor robotnavigasjon. Spesielt aktuelt kan det være å se på bruk kamera kombinert med gode *SLAM* algoritmer. Innføring av kamera gir også muligheter for observasjon i tre dimensjoner. I løpet av arbeidet med roboten ble det brukt tid på å funne informasjon om dette temaet og undertegnende henviser til vedlagt CD der en kan finne artikler for å komme hurtig i gang med et slikt arbeid i fremtiden.

12.2 Navigasjon

Det bør vurderes i videre arbeid å om det er nødvendig å implementere flere navigasjonsalgoritmer. Faktorer som vil påvirke et slikt valg er hvordan videreutvikling av kartlegging vil skje i framtiden.

12.3 Kartlegging

Kartleggingen isolert sett har det vært jobbet mye med i dette prosjektet. Forbedringspotensialet regnes ikke for å være veldig stort dersom det ikke gjøres store endringer i rammebetingelsene rundt nåværende kartleggingen. Imidlertid er det sannsynlig at det vil bli endringer i framtiden med tanke på innføring av for eksempel 3D-kartlegging, kameraovervåkning av roboten og multirobotkartlegging. I en slik sammenheng er det funnet et par interessante artikler er funnet om slike tema:

- Multirobot *SLAM* [30].
- Look-ahead-mapping, Prediktiv-*SLAM* [31].

13 Vedlegg

13.1 Protokoller

Funksjon	Byte 1		Byte 2	
	Hex	Char	Hex	Char
Ping robot	0x6F	o	None	None
SetRadarAngle	0x72	r	0x00-0xB4	[0-180]
SetRobotHeading	0x68	h	0x00-0xB4	[0-180]
SetDeltaDistance	0x64	d	0x00-0xFF	[0-255]
GetLeftTick	0x67	g	0x6C	l
GetRightTick	0x67	g	0x72	r
GetRobotPosition	0x67	g	0x70	p
GetIR	0x67	g	0x69	i
RobotForward	0x6D	m	0x69	f
RobotStop	0x6D	m	0x73	s
RobotLeft	0x6D	m	0x61	a
RobotRight	0x6D	m	0x62	B

Figur 15: Gammel protokoll (kommandosett)

Command to Robot	Hex	Char	Description	Data to robot	Dim.	Returns data	Dim.
ROBOT_FORWARD	0x77	w	Moves robot foward			single Sensordata (automatic)	
ROBOT_STOP	0x73	s	Stop robot (also navigation)			single Sensordata (automatic)	
ROBOT_LEFT	0x61	a	Rotates robot left				
ROBOT_RIGHT	0x64	d	Rotates robot right				
ROBOT_BACKWARD	0x78	x	Moves robot backward				
ROBOT_SET_POS	0x71	q	Set Intern position	xxyytt	1,2	ACK	
ROBOT_GET_POS	0x70	p	Get intern position			MSG_POSITION_HEADER,xxyytt	1,2
ROBOT_CLR_POS	0x63	c	Clear Position to 000000			ACK	
RADAR_SET_ANGLE	0x72	r	Set radar Angle to t	t	3	MSG_RADAR_HEADER, t	
RADAR_GET_ANGLE	0x74	t	Message: No Object in range - sensor 1				3
ROBOT_SET_HEADING	0x68	h	rotates Robot to heading 2t cegrees	t		ACK	
ROBOT_SET_DELTA_DIST	0x6A	j	Moves Robot zz cm	zz		ACK, single Sensordata (automatic)	
GO_POS	0x67	g	Moves robot direct to pos xxyy	xxyy		ACK, single Sensordata (automatic)	
STATUS	0x62	b	Returns the robot status		4	MSG_STATUS_HEADER, s	7
PING	0x6F	o	Ping Matlab->Robot		5	PING_RESPONSE	
PING_RESPONSE_MATLAB	0x6b	k	Ping response from MATLAB		1		
GET_IR_SCAN	0x66	f	Full IR Scan			MSG_IR_SCAN_HEADER, single Data MSG_IR_SCAN_HEADER_END	
GET_SENSOR_1	0x31	1	Get single Sensordata from sensor 1			MSG_IR_HEADER, xxyy	1
GET_SENSOR_2	0x32	2	Get single Sensordata from sensor 2			MSG_IR_HEADER, xxyy	1
GET_SENSOR_3	0x33	3	Get single Sensordata from sensor 3			MSG_IR_HEADER, xxyy	1
GET_SENSOR_4	0x34	4	Get single Sensordata from sensor 4			MSG_IR_HEADER, xxyy	1
CLAW_OPEN	0x79	y	Open the claw				
CLAW_CLOSE	0x75	u	Close the claw				
CLAW_STOP	0x69	i	Close the claw				
HUMAN_MODUS	0x7A	z	Switch to human/data IO modus				

Message to Matlab	Hex	Char	Description	Data to robot
PING_RESPONSE	0x6b	k	Ping response from Robot	
PING_MATLAB	0x6F	o	Ping Robot	PING_RESPONSE_MATLAB
MSG_ARRIVED	0x71	q	Message: Robot arrived	
MSG_IR_SCAN_HEADER	0x66	f	Message: Header for full IR Scan	
MSG_IR_SCAN_HEADER _END	0x67	g	Message: Finished full IR Scan	
MSG_IR_HEADER_1	0x31	1	Message: Header for single IR Data from sensor 1	
MSG_IR_HEADER_2	0x32	2	Message: Header for single IR Data from sensor 2	
MSG_IR_HEADER_3	0x33	3	Message: Header for single IR Data from sensor 3	
MSG_IR_HEADER_4	0x34	4	Message: Header for single IR Data from sensor 4	
MSG_IR_HEADER_NO_DATA_1	0x35	5	Message: No Object in range - sensor 1	
MSG_IR_HEADER_NO_DATA_2	0x36	6	Message: No Object in range - sensor 2	
MSG_IR_HEADER_NO_DATA_3	0x37	7	Message: No Object in range - sensor 3	
MSG_IR_HEADER_NO_DATA_4	0x38	8	Message: No Object in range - sensor 4	
MSG_POSITION HEADER 0x70	0x71	p	Message: Now comes a position	
MSG_RADAR_HEADER	0x72	r	Message: Radar Angle following	
MSG_STATUS_HEADER	0x73	s	Message: Header for status	
MSG_BUSY	0x62	b	Message: Can't receive command now (busy)	
NACK	0x6E	n	Message: NACK	
ACK	0x61	a	Message: ACK	
COLLISION_WARNING	0x77	w	Message Collision warning maybe < 10cm	
COLLISION_ERROR	0x65	e	Message Collision Error < 5cm (Robot stopped)	

1	x,y in mm
2	theta in rad/1000 (0-2000*pi)
3	theta in deg (0-180) (left to right)
4	theta in deg/2 (0-180)
5	z in cm
6	"1"or "2"
7	"1", "2",...,"7"

Figur 16: Dimensjoner i protokollene

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	driving into charging station (unused)

Figur 17: Tilstander i robot

14 Bibliografi

[1] **Schrimpf, Johannes.** *Improvement of the real-time-characteristics of a legorobot.* Trondheim : [prosjektoppgave], NTNU, 2007.

[2] **Kallevik, Paul Sverre.** *Improvement of hardware on Legorobot.* Trondheim : [prosjektoppgave], NTNU, 2007.

[3] **Skjelten, H.** *Fjernnavigasjon av LEGO-robot.* Trondheim : [prosjektoppgave], NTNU, 2004.

[4] **Helgeland, S.** *Automon Legorobot.* Trondheim : [diplomoppgave], NTNU, 2005.

[5] **Syvertsen, B.** *Autonom Legorobot.* Trondheim : [prosjektoppgave], NTNU, 2005.

[6] —. *Autonom Legorobot.* Trondheim : [masteroppgave], NTNU, 2006.

[7] **Magnussen, T.** *Fjernstyring av Legorobot.* Trondheim : [prosjektoppgave], NTNU, 2007.

[8] The CAS-toolbox. [Internett] http://www.cas.kth.se/toolbox/.

[9] The CAS-toolbox userguide. [Internett] http://www.cas.kth.se/toolbox/UsersGuide-0.9.pdf.

[10] **Svendsen, H.** *LEGOLAB.* Trondheim : [hovedoppgave], NTNU, 2002.

[11] **Bjørtomt, H.** *LEGOROBOT.* Trondheim : [hovedoppgave], NTNU, 2003.

[12] **Friquin, Jean Paul Franky.** *An analysis of the development of a safety-critical system for an Urban Search.* Trondheim : [masteroppgave], NTNU, 2006.

[13] **H. Durrant-Whyte, T. Bailey.** *Simultaneous Localization and Mapping part 1.* s.l. : IEEE Robotics & Automation Magazine, 2006.

[14] —. *Simultaneous Localization and Mapping part2.* s.l. : IEEE Robotics & Automation Magazine, 2006.

[15] *Simultaneous localisation and mapping with sparse extended information filters.* **S. Thrun, Y. Liu, D. Koller, A. Ng, H. Durrant-Whyte.** s.l. : Int. J. Robot Res, 2004, Vol. 23, ss. 693-716. no. 7-8.

[16] **J. Neira, J. D. Tardos.** *Data association in stochastic mapping using the joint compatibility test.* s.l. : IEEE Trans. Robot. Automat., 2001. ss. 890-897. 6.

[17] **S.J. Julier, J.K. Uhlmann.** *A counter example to the theory of simultaneous localization and map building.* s.l. : Proc. IEEE Int. Conf. Robot. Automat., 2001. ss. 4238-4243.

[18] **M. Montemerlo, S. Thrun, D. Koller, B. Wegbreit.** *Fast-SLAM: A factored solution to the simultaneous localization and mapping problem.* s.l. : AAAI Nat. Conf. Artif. Intell., 2002. ss. 593-598.

[19] —. *Fast-SLAM 2.0: An improved particle filtering algotithm for simultaneous localization and mapping that provably converges.* s.l. : Proc. Int. Joint Conf. Artif. Intell., 2003. ss. 1151-1156.

[20] Pledge og Tremaux' algoritme. [Internett] http://en.wikipedia.org/wiki/Mazes.

[21] **J. Taheri, N. Sadati.** *A Fully Modular Online Controller for Robot Navigation in Static and Dynamic Environments.* Kobe, Japan : IEEE, 2003.

[22] **S. Homayoun, H. Ayanna.** *Behavior-Based Robot Navigation on Challenging Terrain: A Fuzzy*

Logic Approach. s.l. : IEEE Transaction on robotics and automation, 2002.
[23] **Kjemphol, Gunnar.** *Eurobot 2007.* Trondheim : [masteroppgave], NTNU, 2007.
[24] **C. D. McGillem, T. S. Rappaport.** *A Beacon Navigation Method for Autonomous Vehicles.* s.l. : IEEE Transaction on vehicular technology, 1989.
[25] **S. Ahn, M. Choi, J. Choi, W.K. Chung.** *Data Association Using Visual Object Recognition for EKF-SLAM in Home Environment.* Beijing, China : IEEE/RSJ International Conference on Intelligent Robots and Systems, 2006.
[26] **P. Hong, H. Sahli, E. Colon, Y. Baudoin.** *Visual Servoing for Robot Navigation.* Brussels : Royal Military Academy, Free University of Brussels.
[27] **O. Egeland, J.T. Gravdahl.** *Modelling and Simulation for Automatic Control.* Trondheim : Tapir Forlag, 2002.
[28] **Eric W. Weisstein.** "Circle-Line Intersection". *Wolfram MathWorld.* [Internett] http://mathworld.wolfram.com/Circle-LineIntersection.html.
[29] The Mathworks - the Matlab Central. [Internett] http://www.mathworks.com/matlabcentral/fileexchange/loadCategory.do.
[30] **Getreuer, P.** *Writing Fast MATLAB Code.* June 2006.
[31] **H. Jacky Chang, C. S. George Lee, Y. Hu C., Y.-H. Lu.** *Multi-Robot SLAM with Topological/Metric Maps.* s.l. : IEEE, 2007.
[32] **H. Jacky Chang, C. S. George Lee, Y. Charlie Hu, Y.-H. Lu.** *P-SLAM: Simultaneous Localization and Mapping With Environmental-Structure Prediction.* s.l. : IEEE Transaction on robotics, 2007.
[33] *Mobile Robot Localization and Mapping with Uncertainty using Scale-Invariant Visual Landmarks.* **S. Se, D. Lowe, J. Little.** 8, s.l. : The International Jounal of Robotics Research, 2002, Vol. 21, ss. 735-758.

15 Appendiks

15.1 Vedlagt CD

På CD ligger følgende mapper:

- Bakgrunnsstoff
- Figurmappe
- LegoGUI_Høst2007
- Tidligere rapporter

15.2 Figurliste