Oppgavetekst

En robot basert på Lego Mindstorms, NXT-robot, skal konstrueres for å kartlegge ukjente omgivelser. Kartleggingen og navigeringen gjøres eksternt på PC med et tidligere utviklet system, CAS.

I CAS er det en annen robot, IR-robot, som er utviklet og fungerer sammen med CAS for å kartlegge ukjente områder. NXT-roboten skal ha funksjonaliteten som trengs for å utføre samme jobb som IR-roboten utfører.

NXT-roboten skal integreres i CAS. CAS er utviklet for IR-roboten, og en av utfordringene blir å få integrert NXT-roboten i dette systemet.

Det skal utvikles programvare til NXT-roboten som kjører lokalt på robotens mikrokontroller.

Kartleggingen blir utført med å måle avstanden rundt roboten med infrarøde sensorer. Roboten må bli konstruert for å kunne måle i 360°omkrets.

Forord

Dette prosjektet har vært en del av en lengre kjede av oppgaver som omhandler kartlegging med legoroboter. Det har vært interessant å være en del av dette arbeidet. Iløpet av semesteret har det vært flere forskjellige deler som har måtte passe sammen. Det har vært forskjellige fremgangsmåter for de forskjellige delene av prosjektet som har gitt en større oversikt.

Prosjektet har gitt god erfaring med programmering i MatLab, strukturering av kode og programmering av NXT-brikker. Selv om dette har vært interessant individuelt, er det ikke minst erfaringen med å få alt til å fungere opp mot hverandre som har vært lærerikt.

Rapporten er skrevet med tanke på videre arbeid og med tanke på en innføring i systemet. Vedlegg skal forhåpentligvis være dekkende sammen med rapporten til at mange relevante spørsmål blir besvart.

Det ønskes å takke

- NTNU for å ha gitt en plass å jobbe på, utstyr og PC.
- Professor Tor Onshus for å ha vært en god veileder igjennom semesteret

Innhold

Sammendrag

Det ble laget en ny robot for å kjøre i et kartleggingssystem. Dette systemet har til nå bestått av en legorobot, som er laget fra bunnen av i liknende prosjekter, og programvare på PC i MatLab kalt CAS.

Den nye legoroboten består av Lego Mindstorms komponenter og NXT-brikke. Her ble det utforsket måter å programmere NXT-brikken på og språket NXC ble valgt. Kode til NXT-brikken er utviklet og fungerer i samarbeid med MatLab.

For å behandle kommunikasjonen mellom MatLab og NXT-brikken ble det laget et grensesnitt som en egen modul i MatLab.

Videre ble roboten forsøkt integrert i MatLab, og dette resulterte i utviklingen av bedre struktur på programvare mellom databehandling og robotstyring. Det ble utviklet et grensesnitt mellom databehandlingen og robotene kalt RobotHandler.

Med bruk av RobotHandler er roboten velykket integrert i MatLab og fungerer på samme måte som tidligere robot.

Omstruktureringen av MatLab koden gjør at tidligere robot ikke fungerer slik den gjorde. Det er nødvendig å sette koden relatert til denne roboten i et eget grensesnitt før den kan kjøre i systemet. Koden er tatt vare på og lagret i et utkast til et grensesnitt.

Etter velykket integrering av robot er det utført tester for å sjekke at programvaren kjører med fysisk robot og at simuleringer fungerer.

GUI er utvidet for å håndtere RobotHandler. Der iblant er det en funksjonalitet for å plotte global data for å sjekke input fra robot og sammenlikne med generert kart.

I avslutningen er det diskusjoner rundt deler av prosjektet som kan gi tips for videre arbeid, og innsikt i systemet. Deretter er det foreslått retninger for videre arbeid.

I vedlegget er det lagt tabeller og brukermanual for at fremtidige prosjekter skal komme lett igang og ha nødvendig dokumentasjon.

Kapittel 1

Innledning

Dette prosjektet handler om å kartlegge et ukjent område ved hjelp av roboter som kjøres fra en PC. Denne PC'en har programvare som kjører i MatLab. Denne programvaren blir betegnet som CAS. CAS ble utviklet i 2004, og har blitt benyttet i studentprosjekter siden 2005 og har etter den tid blitt forbedret og bygget på. Mer om CAS blir beskrevet kapittel 2.2.

1.1 Konklusjon

Det er laget en legorobot med NXT-brikke, kalt NXT-roboten. NXT-roboten er konstruert for å kunne ta målinger i 360° omkrets og det benyttes to sensorer med 180° vinkel mellom hverandre Sensorene er identiske og blir oppgitt av leverandør til å ha maksimal rekkevidde på 20cm. Denne rekkevidden er litt kort i forhold til tidligere robot.

Programvaren til NXT-roboten ble skrevet i NXC, **N**ot e**X**actly **C**, og fungerer som det skal sammen med programvaren i MatLab. Det har tatt tid å lage denne koden fra bunnen av, og det kan anbefales en annen løsning, beskrevet i delkapittel 7.2.2, for videre arbeid.

For å håndtere NXT-roboten med egen kode ble det laget NXTInterface i MatLab, og deretter et grensesnitt mellom CAS og robotene kalt RobotHandler. Både NXTInterface og RobotHandler er gode bidrag til systemet. RobotHandler gjør det vesentlig enklere å integrere nye roboter, og sammen med NXTInterface gir det systemet en bedre og mer oversiktlig struktur.

RobotHandler bidrar til at verdier blir definert på en mer naturlig måte i forhold til hvordan systemet var med tidligere robot. Det gir også et naturlig skille mellom CAS og robotene, slik at det ene kan debugges uten det andre.

Nå er RobotHandler velykket integrert i CAS, slik at en ny robot kan legges til i RobotHandler og vil fungere i CAS om den tilfredsstiller kravene til RobotHandler.

Å legge til RobotHandler i CAS med riktig dataflyt var mer krevende enn antatt. Men resultatet ble dokumenterte inn- og ut-verdier og et fungerende system som føles bedre egnet for videre arbeid.

NXT-roboten ble testet og det viste seg at rådata ikke er så bra som forventet. Sensorene er dyre og nøyaktige, men trenger kalibrering og fungerer kun på samme materiale som kalibreringen er utført på. Rådata ble plottet og sammenliknet med generert kart, som viser at CAS fungerer som det skal. Dette kan bekreftes igjennom simuleringer.

En testbane med A4 papir og kalibrerte sensorer gav en tilfredsstillende kartgenerering, sammen med en forbedret posisjonsestimering.

Rapporten er skrevet for både fremtidig arbeid og for å beskrive systemet. Dette er gjort med å dokumentere og svare på spørsmål som har kommet opp under arbeidet med CAS og NXT-roboten. Det er lagt vekt på å dokumentere det som er nødvendig for videre arbeid.

1.2 Robotene

Det er to legoroboter som benyttes for å kartlegge områder. Den ene er bygget fra bunnen av med mikrokontroller og motorstyringskort. Den andre er basert på Lego Mindstorms NXT-brikke.

Robotene blir heretter omtalt som NXT-roboten og IR-roboten.

1.2.1 IR-roboten

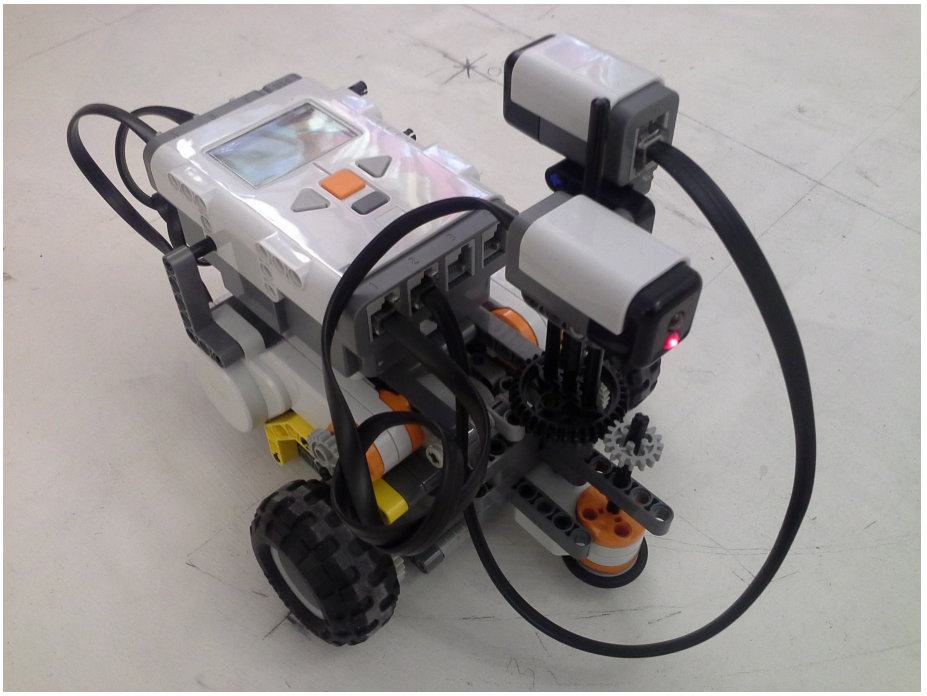
IR-roboten blir benyttet i andre, liknende prosjekter. Denne roboten har blitt laget og forandret opp igjennom flere år i forbindelse med prosjekter og master-oppgaver. IR-roboten er laget av studenter på NTNU og det er mange som har bidratt til arbeidet som kreves for at roboten skal være slik den er idag.

Fordi denne roboten ble laget og forbedret på samme tid som programvaren for SLAM ble utviklet, er denne roboten godt integrert i tidligere programvare.

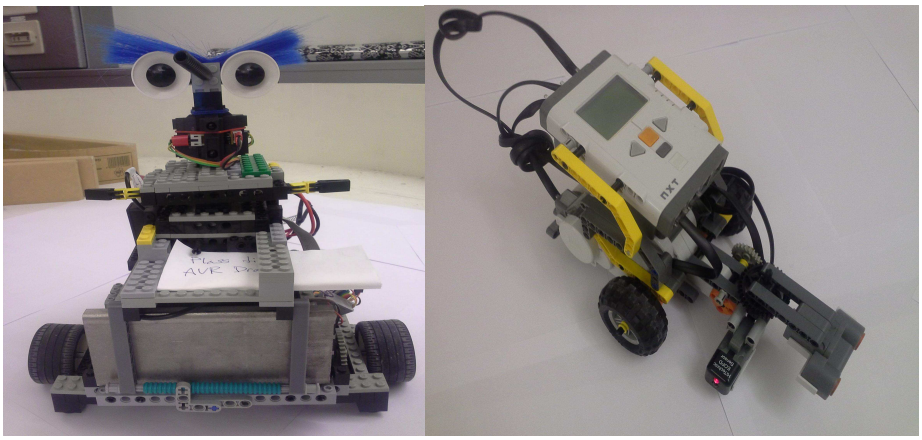
1.2.2 NXT-roboten

NXT-roboten er den roboten som benyttes i dette prosjektet og består av Lego Mindstorms deler. Figur 1.2 viser et bilde av denne roboten slik den var før ombyggingen til bruk i dette prosjektet. Og figur 1.1 viser roboten slik den brukes i prosjektet.

Roboten måler avstandene til objekter og flater i 360° omkrets. Dette blir gjort med et roterende tårn i front på roboten. Som det fremkommer av figur så er dette en bipedal robot, samme som IR-roboten. Det er tre kontaktpunkter som sammen lager en likebeint trekant.



Figur 1.1: NXT-roboten slik den brukes i dette prosjektet.



Figur 1.2: IR-roboten er til venstre og NXT-roboten, før ombygging, til høyre.

Mindstorm komponenter

Komponentene som er brukt for å lage NXT-roboten blir beskrevet kort under.

NXT-brikken Det ble først benyttet en eldre versjon av NXT-brikken, versjon 1.0. Skjermen fungerte ikke og heller ikke bluetooth. Dette gjorde det vanskelig å jobbe med koden til NXT-brikken, men det ble ledige NXT 2.0 brikker som ble benyttet senere. Dette løste en del problemer, og det anbefalles å benytte NXT 2.0 i videre arbeid.

Motorene Motorene til Lego Mindstorm er nøyaktige, kraftige og nedgiret. Det er innebygget sensor som gjør roboten nøyaktig til $\pm 1°$. Innebygget sensor er en stor fordel. På IR-roboten er sensor og motorer separat.

Sensorene Sensorene er tredjeparts utstyr fra firmaet HiTecnic. Sensorene blir kalt elektro optisk avstandssensor, EOPD. Rekkevidden er maksimalt 20 cm, og er nøyaktige i forhold til alternativer innen Lego Mindstorms.

Grunnnet høy pris og lavere rekkeveidde enn ønsket anbefales det å bytte ut disse sensorene med tilsvarende sensorer som brukes på IR-roboten i videre arbeid.

Mål på robot

Målene blir gitt inkludert ledninger. Dette kan variere med hensyn på hvordan ledningene er plassert, men målene på roboten må bli gitt inkludert ledningene da disse hindrer rotasjon der det er smalt i omgivelsene.

Lengden til robot: 26cm

Bredden til robot: 16cm

1.3 Systemet

Kartlegging av ukjente områder med legorobot har blitt jobbet med i mange prosjekter på NTNU fra 2004. En beskrivelse av tidligere arbeider er beskrevet i kapittel 1.4.

I hovedsak handler alle prosjektene, master- og prosjekt-oppgaver, om å kartlegge et ukjent område. Det har vært mest arbeid rettet mot IR-roboten. I tillegg har der vært forgreininger med forskjellige ekstrafunksjoner og arbeider med NXT-brikken.

Dette systemet følger tidligere arbeider ganske tett, uten forgreining. Det er en videreutvikling av CAS-toolbox, og utvikling av NXT-roboten.

CAS-toolbox kjører i MatLab på en PC. Det er dette som er hjernen i systemet, der robotene kun samler inn data. CAS lager kartet, navigerer og oppdaterer posisjon for roboten.

Basert på forventet feil fra roboten, gitt av feilkoeffisienten, EGC, oppdaterer CAS posisjon og sammenlikner nye måledata med gamle måledata. Denne feilen er det ikke mulig å bli kvitt, fordi robotene bruker «dead reckoning». Et eksempel på dette er å bruke tilbakelagt avstand på hjulene for å regne ut neste posisjon.

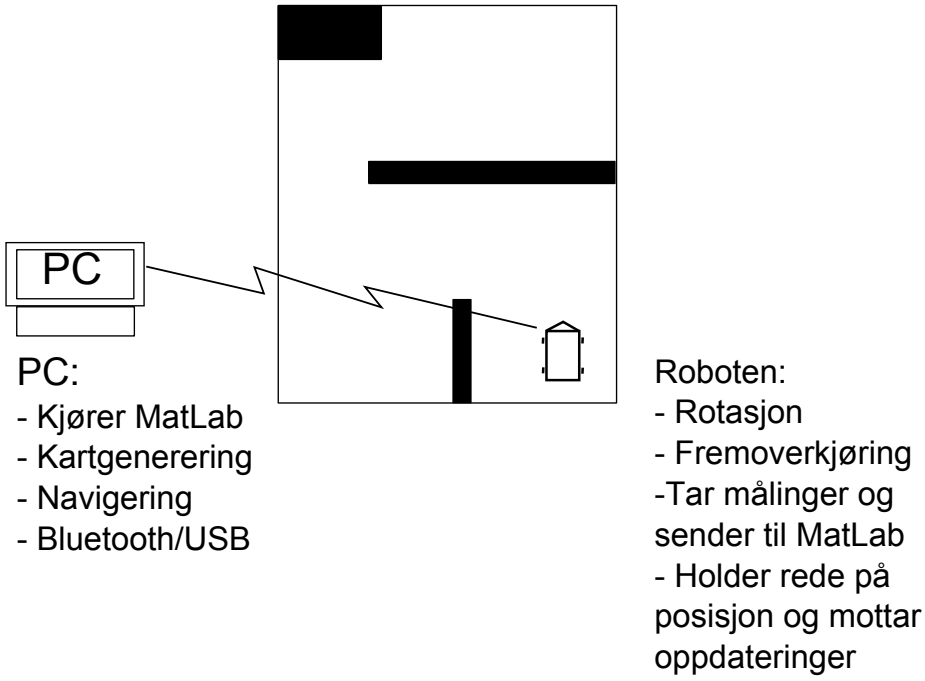
Med å bruke MatLab ved siden av roboten, er det mulig å gjøre avanserte

utregninger i tilknytning til kartleggingen i forhold til hva som er mulig med å bare bruke en mikrokontroller.

Roboten i dette prosjektet har ikke blitt brukt på denne måten før, og har derfor fått en egen kode for å takle kommunikasjon og kommandoer fra MatLab. Denne koden er skrevet i NXC, beskrevet i kapittel 4.2.

Generelt sett så er all tid- og plass-krevende operasjoner på roboten fjernet og lagt i MatLab, eller C++ programmer i tidligere oppgaver. Det som er igjen blir da funksjonalitet som å kjøre en gitt avstand, eller å svinge en gitt vinkel. Posisjonen må bli tatt vare på og må oppdateres. Alle målinger med avstandssensorer blir sendt til MatLab og blir ikke lagret lokalt.

Posisjonsestimatet vil ha en feil som akkumuleres over tid, eller rettere sagt avstand. Denne feilen er det ikke mulig å bli kvitt uten å bruke absolutt posisjonering. Dette skjer kun ved resetting av posisjon. Feilen kan derimot minimaliseres med hardware og programvare på roboten.



Figur 1.3: En oversikt over systemet slik det er satt opp i de fleste oppgavene.

1.4 Tidligere arbeider

Denne type prosjekter, kartlegging med legorobot, har blitt utført siden 2004. Det har vært en jevn progresjon siden da, men med forskjellige forgregninger. Med forgreininger menes prosjekter som har vært like i stil med litt andre mål eller løsninger. Dette er for eksempel bruk av kamera, prosjektoppgaven til Tøraasen09[15], kartlegging uten CAS, prosjektoppgaven til Homestad12[10] og oppgaven til Auestad11[17], hvor det er laget et konsept på førerløse biler.

IR-roboten har hatt egen kode ifra begynnelsen av, men NXT-brikken har kun kjørt fra MatLab med et grensesnittet utformet av Acchen universitetet i tyskland, RWTHMindstormsNXT-toolbox [4].

En liste over progresjon av tilgjengelige arbeider er oppsumert under:

Skjelten 2004 prosjekt,*Fjernnavigasjon av LEGO-robot*: Tar utgangspunktet i å teste infrarøde sensorer og ultralyd sensor. Lager et dataprogram som loggfører dataene og bygger en legorobot som med kompass. Denne roboten brukte RCX-brikke til å styre motorene, og en mikrokontroller for å håndtere komunikasjonen. Dataprogrammet ble kalt RoboRadar og fungerte bra til å teste sensorer. Programvaren ble skrevet for Linux.

Helgeland 2005 master,*Autonom legorobot*: Helgeland tok tak i det tidligere prosjektet til Skjelten, og bygget ut med tre moduler. Navigasjon, kartgenerering og kollisjonsunngåelse. Han satte på en sensor for å måle vinkelutslaget til motorene slik at han da hadde det som trengs for å estimere posisjonen til roboten. Kompass, tilbakelagt avstand og avstandssensorer. Programvaren kjørte på PC og målene med å kartlegge ble ikke oppnådd i ønsket grad. Helgeland mener at kompasset ikke er en god sensor, og som ødelegger for en del av kartleggingen.

Syvertsen 2005 prosjekt,*Ukjent tittel*: Programvaren blir overført til MatLab/simulink for å gjøre videre utvikling enklere. Dette er også et miljø som kan antas å være kjent for fremtidige studenter. Programvaren gir omtrent samme resultater som tidligere oppgaver, og det antas igjen at det er kompasset som ødelegger for kartgenereringen. En navigasjonsalgoritme blir implementert, men er såpass enkel at den bare lar seg bruke til testing av kartgenerering.

Syvertsen 2006 master, *Autonom legorobot*: Kompassensoren blir fjernet og sensorer for hvert hjul blir implementert for å kunne estimere orienteringen. CAS blir innført for å generere kart og en navigasjonsalgoritme basert på Tremaux algoritmen ble implementert. Denne navigasjonsalgoritmen blir fremdeles brukt.

Magnussen 2007 prosjekt, *Fjernstyring av LEGO-robot*: Det ble utviklet en simulator til systemet fra bunnen av, som fungerte sammen med CAS. Kartlegging og navigasjon er forbedret. Roboten kan navigere i kryss med opp til fire innganger og blindveier. Dette er begrenset på grunn av at algoritmen er bygget på Treumax algoritmen.

Magnussen 2007 master, *Fjernstyring av Legorobot* Kartgenerering ble utvidet med at det ikke lenger kun var basert på linjesegmenter, men også punkter. Det ble gjort litteraturstudie på navigasjonsalgoritme, og den implementerte navigasjonsalgoritmen ble utvidet til å basere seg på punkter og linjer. En ladestasjon ble laget.

Kallevik 2007 prosjekt, *Forbedring av hardware til Legorobot*: Det ble laget et nytt hovedkretskort og RCX-modulen ble byttet ut med et motorstyringskort. I tilligg til å utføre studier og bidra til dokumentasjon om sensorer, ble antall sensorer økt fra én til fire.

Schrimpf 2007 prosjekt, *Forbedring av sanntidsegenskapene til en legorobot*: Det ble lagt vekt på sanntid og kompatibilitet. Muligheten til å kunne måle mens roboten kjører ble implementert. Arbeidet med å gjøre systemet kompatibelt med tilleggsmoduler og eksisterende moduler er viktig. Andre prosjekter har hatt dette i bakhodet, men har allikevel utformet koden slik at den blir skreddersydd IR-roboten.

Næss 2008 prosjekt, *Fjernstyring av legorobot*: Små forbedringer på roboten som å forbedre motorstyringskortet med potensmeter og optimalisering av strømforbruk. Det ble gjort en litteraturstudie på absolutt posisjonering ved hjelp av trådløse sendere. IR-sensorene ble kalibrert inviduelt som forbedret målingene.

Bakken 2008 prosjekt, forgregning, *Bygge og programmere ny Legorobot*: NXT brikken ble brukt sammen med IR-sensorene til IR-roboten. RWTH-toolbox ble benyttet og forsøkt integrert i CAS. Resultatet ble regnet som et innledende prosjekt til videre arbeid.

Haugedal 2008 prosjekt, *Forbedring av de autonome egenskapene til en legorobot*: Ladestasjon til roboten ble forbedret og løsninger for å lade roboten på en fornuftig måte med sikringer ble laget. Det ble implementert en metode for å koble seg til ladestasjon, men ikke en «gå hjem» funksjon. Det blir påpekt dårlig programering i tidligere prosjekter med forskjellige stiler og løsninger som å lage funksjoner med store kodeblokker rett i GUI'en.

Tusvik 2009 prosjekt, *Fjernstyring av legorobot*: Posisjonsestimatet har blitt forbedret med å rette opp småfeil, og prosjektet har blitt samlet i rapporten for å gjøre videre arbeid enklere. Det ble gjort endringer i kartgenerering og forbedringer i avstandsmålene som førte til kart med høyere kvalitet, som igjen fører til bedre navigering. Det ble også påpekt rot i programvaren.

Tøraasen 2009 prosjekt, forgrening *Kartlegging ved hjelp av robot og kamerasensor*: NXT-brikken blir brukt med ultralydsensor og Lego motorene. I tillegg til ultralydsensoren så er det et webkamera som brukes til å ta stillbilder. I MatLab brukes så kant-detektering for å lete etter hjørner. Konseptet fungerer, men det er ikke pålitelig nok til at man kan anta at alle hjørner blir oppdaget. Når et hjørne oppdages blir det koblet sammen med de forrige hjørnene, og en representasjon av omgivelsene blir plottet i 2D og 3D. Denne representasjonen kan da være feil, som følge av måten kartet blir plottet på. Arbeidet er et godt grunnlag for å fortsette med bildebehandling. Men fordi det oppdages bare et hjørne, i beste fall, vil det ikke være nok til pålitelig SLAM i seg selv.

Kristiansen 2009 prosjekt, *Fjernstyring av LEGO-robot*: Det ble utviklet en metode for å returnere til ladestasjon med å bruke punkter med $0,1m$ oppløsning. Det nevnes at det er feilmeldinger som ikke har blitt ordnet opp i på grunn av tidsmangel, og at det ikke er kolisjonsungåelse på tilbakeveien.

Auestad 2011 prosjekt, forgregning, *Førerløs bil*: Prosjketet baserer seg på konseptet med førerløse biler. Roboten er bygget med NXT-brikken og følger veggen. Det er forskjellige muligheter når bilen treffer et kryss som er modellert på forhånd. Det er laget metoder for å svinge bilen i forskjellige rette kryss og det blir tatt hensyn til om det kommer biler imot. Bilen kjører en forhåndsbestemt rute. Dette er et prosjekt som står alene med hensyn på at roboten ikke kartlegger omgivelsene. Prosjektet er allikevel relevant for videre arbeid.

Hannaas 2011 masteroppgave, *Samarbeidende legoroboter*: Vektfordelingen til roboten er forandret for å eliminere noe av hjulspinn. En regulator er implementert for å forbedre nøyaktigheten til roboten. For å gjøre roboten mer autonom er batteriet simulert for å gi beskjed om å kjøre tilbake til ladestasjon når dette blir nødvendig. NXT-brikken med kamerasensor er forsøkt implementert i CAS. Forsøket på å få robotene til å samarbeide er ikke utført. NXT-roboten med kamera ser ikke ut til å ha vært mobil ut ifra programvaren gitt[1].

Homestad 2012 prosjektoppgave, forgregning, *Fjernstyring av legorobot*: NXT-brikken ble brukt til å kartlegge en labyrint ved hjelp av veggfølging og odometri. Denne løsningen var uavhengig av CAS og ble en egen måte å løse et kartgenereringsproblem på. Roboten kartlegger raskt og forholdsvis nøyaktig når hjulene holder kontakt med underlaget, men denne implementasjonen er avhengig av rette vinkler på kryss. Noe som naturligvis ikke er gitt i naturlige omgivelser. Programmeringen ble foretatt i MatLab, og konseptet kan videreutvikles om det skulle være ønskelig. Men i så fall bør noe av koden overføres til NXT-brikken, som veggfølgingen og odometrien.

Som det fremkommer av listen over, har det vært mange studenter innom dette arbeidet, og mange av rapportene påpeker at det har vært vanskelig å jobbe videre med prosjektet av den grunn. Men det kan også sees at alle prosjektene bidrar med noe for å løfte systemet fremover i riktig retning. Prosjektene som ikke er direkte koblet opp mot CAS gir også grunnlag for videre utvikling. For eksempel er det ikke gitt at dette prosjektet hadde blitt basert på NXT-brikken uten de tidligere oppgavene Tøraasen09, Auestad09 og Homestad12[15, 17, 10].

[1] Se for eksempel filen setRobotTarget.m, som er ansvarlig for å kjøre roboten til gitt sted. Her er det ikke lagt til kode for NXT kamerarobot.

Kapittel 2

SLAM

SLAM står for ≪Simultaneous Localization And Mapping≫ og vil derfor være et generelt begrep som gjelder mange forskjellige algoritmer. SLAM systemet som er implementert i MatLab består av navigasjon, estimering av posisjon, skanning av området og utvinning av linjer og punkter.

Navigeringen blir utført av en algoritme basert på Trémaux algoritmen. Posisjonen blir estimert med et EKF hvor odometri og landemerker blir benyttet. Landemerkene kan være linjesegmenter og beacons.

2.1 Navigasjonsalgoritmen

Selv om navigeringen er basert på Trémaux algoritmen er den ikke identisk. Trémaux algoritmen, og den implementerte algoritmen blir beskrevet under.

2.1.1 Trémaux algoritme

Trémaux algoritmen finner frem i alle labyrinter som har veldefinerte veier. En vei er enten ubesøkt, markert én gang, eller markert to ganger. Algoritmen fungerer med at når man går i labyrinten markerer man hvor man går. I begynnelsen velges en tilfeldig vei, om det er mer enn én, og deretter når det oppstår et kryss så velges en tilfeldig vei. Hvis krysset er markert så gå tilbake samme vei som du kom fra og fortsett. Hvis veien du kom fra har to merker, så velges den veien med færrest markeringer. Forklaringen er tatt fra wikipedia [5].

2.1.2 Implementert navigasjonsalgoritme

Algoritmen som brukes ble implementert i masteroppgaven, Syvertsen06[14]. Dette er en variant av Trémaux algoritmen og krever at roboten klassifiserer kryss for å kunne navigere riktig. Det kreves også at startposisjonen er gunstig.

Veggfølgingen foregår med at roboten sjekker mulige vinkler fra venstre, $+45°$, til høyre, $-90°$. Den første mulige vinkelen blir retningen roboten kjører for å følge venstre vegg. Hvis det er hindringer vil roboten da rotere $90°$ til høyre og hvis det er et åpent område vil roboten kjøre $45°$ til venstre. Derfor vil roboten kjøre i en sirkel om det ikke er noe vegger eller objekter å detektere, og deretter kjøre mot et ukjent område.

Men startposisjonen må altså være gunstig, og det er noen feil knyttet til dette. Hvis roboten starter i en gang mot en blindvei vil den snu og finne stien den kjørte og avslutter kartleggingen. For detaljer om implementering og detektering av kryss henvises det til rapporten rapporten til Syvertsen06, vedlagt på CD.

2.2 CAS toolbox

CAS[1] Robot Navigation Toolbox er en toolbox til MatLab under GNU lisens[2]. Det er denne toolboxen som er benyttet i tidligere arbeider med legorobotene,

[1] Center for Autonomous Systems

og som skal benyttes videre i dette prosjektet. Det er gjort modifiseringer opp igjennom forskjellige år.

CAS toolbox ble innført i 2005 i prosjektoppgaven til Syvertsen[14]. Før dette var det laget et program i C++ fra bunnen av. Syvertsen argumenterte for byttet med at det tidligere systemet ble for dårlig i lengden. Mer konkret, fra prosjektoppgaven til Syvertsen, mente han at C++ programvaren,

- hadde lavt abstraksjonsnivå.
- var vanskelig å feilsøke og isolere algoritmefeil.
- hadde dårlig dataflyt, flere tråder og globale variabler delt mellom flere filer.
- er tidkrevende å forstå og å få oversikt over. Tunglest kode.
- er plattforavhengig.

Med disse argumentene er det lett å se hvorfor han valgte å bytte til CAS toolbox i MatLab. MatLab er ofte tilgjengelig for studenter og er ikke plattformavhengig. CAS toolbox er modulbasert og har muligheten til å utvide og isolere kode.

CAS toolbox blir allsidig i forhold til hvilken robot som brukes derson koden er modulbasert. Baksiden med å bytte er at MatLab blir tregere enn det en implementasjon i C++ kan bli. CAS toolbox støtter også bare 2D, men roboten som brukes i dette prosjektet har kun mulighet til å måle avstander horisontalt slik at 3D ikke har vært aktuelt.

2.2.1 Teori

Det er ikke jobbet videre med kartlegging og navigering, og teorien bak dette blir derfor referert videre til tidligere rapporter.

Tusvik08 går kort igjennom prinsippene, og refererer videre til mer utdypende kilder, vedlagt på CD.

2.2.2 Programflyten

Hovedløkken til programmet befinner seg i filen «lineBeaconSLAM». Denne løkken kjører til programmet er ferdig med kartleggingen eller det oppstår en feil.

Det er selvsagt mye annen funksjonalitet i programvaren. Som valg av parametere og manuell kjøring. Men det viktigste for å forstå kjøringen av SLAM blir gjengitt nedenfor.

Innhenting av estimert posisjon: Posisjonen hentes fra *getRobotPose*. Posisjonene blir gitt som en radvektor med verdiene $[x, y, \theta]$.

Posisjonen blir vist i GUI: Posisjonen blir vist i GUI for fysisk robot og simulator.

Tilstandsestimering og posisjonsoppdatering: I tilstandsestimeringen så brukes *getAngularDispl* for å se hvor mange omdreininger hjulene har hatt. Dette brukes for å regne ut feilen som kan antas å være til stede. Videre brukes disse verdiene i *predict* for å oppdatere posisjonen til roboten og det globale kartet oppdateres med *robotdisplacement*. *predict* og *robotdisplacement* er en del av SLAM rammeverket(MathWorks,Arras), sammen med andre funksjoner som brukes her, som *getrobot* og *setrobot*.

Skanning: *fullScan* returnerer en tabell med måledata. Tabellen har tre kolonner hvor den første kolonnen sier hvilken sensor som er brukt for målingen og de to andre kolonnene gir x og y verdier til målepunktet i det globale kartet.

Med RobotHandler blir denne dataen uavhengig av antall sensorer som er brukt.

Linjeutvinning og beacons: *extractlines* er en del av SLAM rammeverket(Arras). Denne funksjonen bruker en algoritme for å tilpasse datapunktene linjer fra *fullScan*. *predictmeasurement*, *matchnnsfLines*, *estimatemapLines* og *intergratenewobs* er funksjoner av Arras som oppdaterer det globale kartet ut ifra robotens nye posisjon og sammenlikner linjer fra målingen med tidligere linjer. Robotens posisjon, internt i CAS, blir oppdatert.

Det samme skjer med beacons som finnes i kartet så lenge det benyttes både linjeutvinning og beacons samtidig.

Det er lagt til en funksjon fixCorners.m i tidligere prosjektoppgaver som skal forbedre hjørner, og det er lagt til kode for å slette datapunkter nær linjer.

Post-prosessering av kart: Robotens posisjon blir lagret i navData.robotPath og total avstand beregnes og lagres i navData.totalDist.

Videre blir det fjernet punkter som overlapper linjesegmenter i deleteFromMap() og det finnes hull med gapFinder3().

Visualisering av kart: Robotens posisjoner, linjesegmenter og beacons blir tegnet opp. Det er mulig å krysse av for forskjellige plots i tillegg til hovedvinduet.

linkPoints() er en funksjon av Magnusen07[11] som finner kanter basert på punkter.

Gå hjem: Hvis riktige kriterier er satt så går roboten hjem. Det kan være fordi den er ferdig eller vil oppdatere posisjon.

Navigasjon: Hvor roboten skal kjøre får den beskjed om av leftWallFollowerToBackTracker.m. leftWallFollowerToBackTracker.m benytter quasiFollowLeftWall.m for å finne rotasjon i radianer.

Batteri sjekk: MatLab spør roboten om batteristatus og setter 'flag' om det er nødvendig med lading. Dette er ikke i bruk under simulering.

Kapittel 3

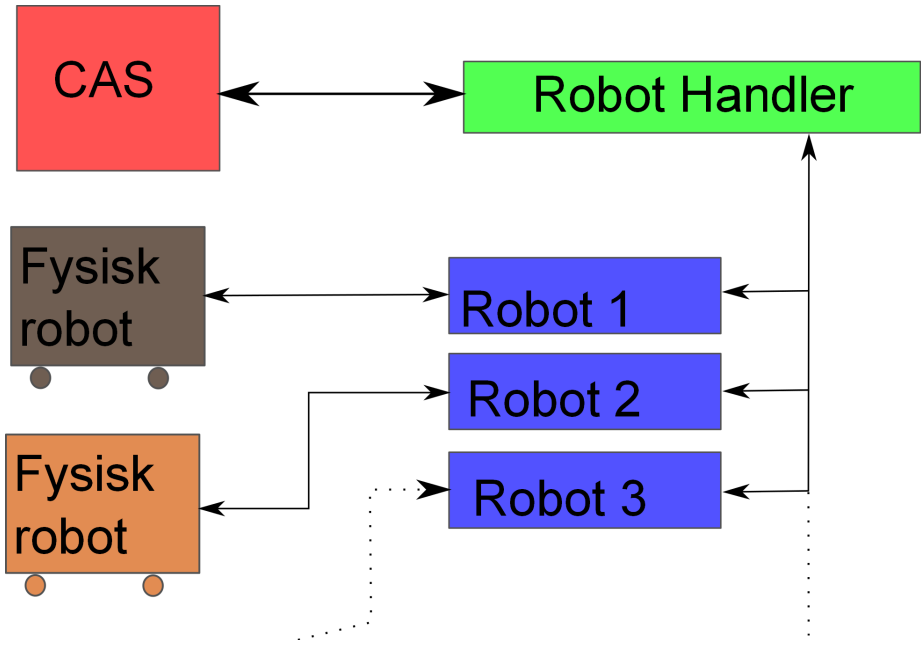
Struktur, grensesnitt og komunikasjon

CAS programkoden er relativt tunglest i forhold til det det burde ha vært. Det er fordi det har jobbet mange studenter med forskjellige mål på samme kode. Koden ved oppstart av prosjektet var rotete i forhold til kommunikasjon med den tilhørende IR-roboten, men greit satt opp ellers. Det var først tenkt å legge NXT-roboten til direkte i programkoden, men etter kort tid ble det klart at mer struktur var nødvendig. Da ble først grensesnittet til NXT-roboten utviklet, og deretter ble grensesnittet mellom CAS og robotenes grensesnitt utviklet. Dette er beskrevet i delkapittel 3.1 og 3.2.

Kommunikasjonen mellom PC og robotene varierer fra robot til robot. Det er derfor det ble bestemt å lage egne grensesnitt til hver robot[1]. Slik kan kommunikasjonen til CAS bli standardisert og videre arbeid blir vesentlig enklere.

Delkapittel 3.2 og 3.4 beskriver hvordan robotenes kommunikasjon varierer, og beskriver hvordan variabler og definisjoner ble satt.

[1] Identiske roboter trenger naturligvis ikke forskjellige grensesnitt, som er aktuelt med hensyn på NXT-roboter. Disse roboten med lik programvare kan kjøres enkeltvis fra samme grensesnitt: NXTInterface.



Figur 3.1: Struktur på systemet med et grensesnitt mot robotene. Robotene er ment som alternativer. Det er ikke mulig å opperere mer enn én robot av gangen med nåværende programvare.

Programvaren CAS, og programvaren på NXT-roboten ble beskrevet i kapittel 4 og 5. En oversikt over strukturen til systemet slik det er laget i dette prosjektet kan sees på figur 3.1.

3.1 Grensesnitt – Robot Handler

Å integrere en ny robot i systemet har vist seg problematisk i tidligere oppgaver og iløpet av dette prosjektet.

For å gjøre dette enklere, ble det tenkt ut en ny struktur på programvaren.

Denne strukturen er den som vises på figur 3.1. Her vil CAS kun ha kontakt med RobotHandler som gjør at eventuelle feil i CAS og i robotene blir separert. Hvis CAS fungerer slik det skal, vil det være enklere å legge til en ny robot.

RobotHandler, heretter også omtalt som RH, er navnet på grensesnittet mellom CAS og individuelle grensesnitt til robotene.

Slik programvaren er nå, er det ikke mulig å bruke mer enn én robot av gangen. Robotene på figur 3.1 er ment som alternativer.

3.1.1 RH – CAS

CAS med RH blir all programvare på PC hvor hvilken robot som brukes ikke er relevant. Dette er kartlegging, databehandling og navigering. Parametre i CAS forandres ettersom hvilken robot som brukes, men dette er standardiserte variabler som hentes fra RH.

Når dette prosjektet ble påbegynt var CAS kun kompatibelt med IR-roboten. Og filene som ble lagt til i CAS igjennom årene var skreddersydd IR-roboten. CAS er i utgangspunktet ment å være modulært, men det har ikke vært en prioritet i tidligere oppgaver, bortsett fra i prosjektoppgaven til Schrimpf i 2007[13].

Hvis NXT-roboten skulle bli lagt til i CAS slik IR-roboten er lagt til hadde dette vært en stor jobb, som ikke nødvendigvis hadde blitt velykket. Derfor ble det bestemt å sørge for å separere CAS fra robotene, og modulisere strukturen så mye som mulig.

Når CAS kun forholder seg til RH kan det defineres et sett med funksjoner som er helt nødvendig. Disse funksjonene ble kalt basisfunksjoner og vises i tabell B.1. Alle andre funksjoner er ekstrafunksjoner, og er ikke nødvendige for å kjøre systemet. Men noen av dem har praktisk nytte, som å teste ut skanning uten å kjøre systemet.

Fordi all databehandling og navigering blir utført i CAS, burde alt som er variabelt bli trukket ut av CAS, slik at om en ny robot skal brukes er det ikke nødvendig å gjøre endringer i programkoden. Variabler som er tilknyttet roboter

må da hentes med standardiserte funksjoner. Som for eksempel variabelen EGC, feilkoeffisienten. EGC er et mål på hvor mye feil roboten kjører per meter og vil variere fra robot til robot. Andre parametre er størrelse på robot og hjulradius.

Resultatet blir at CAS i praksis ikke vet hvilken robot som brukes, og kun inneholder «hjernen» i systemet.

Et eksempel på hvordan dette ser ut kan være:
En forespørsel fra CAS om at roboten skal kjøre fremover uavhengig av robot,
`RobotHandler('driveForward',25)`
hvor 25 er i *cm*. RobotHandler vil sjekke hvilken robot som er tilkoblet, og om NXT-roboten brukes så sendes kommandoen videre med:
`NXTInterface('driveforward',25)`
og NXTInterface vil utføre gitt kommando.

3.1.2 RH–roboter

Fordi det er standardisert kommunikasjon mellom CAS og RH, er det mellom RH og individuelle robotgrensesnitt kommunikasjonen varierer. De aktuelle robotene må være lagt til i RH. Dette gjøres med å legge inn kode tilhørende roboten i switch-caser i basisfunksjonene i RH.

En av grunnene til at NXT-roboten ikke er lagt direkte inn i RH er for å forhindre at samme problematikk som ved IR-roboten og CAS oppstod.

Et eget grensesnitt til hver av robotene gjør RH mye mer oversiktlig. Som gjør videre arbeid med å legge til nye roboter enklere. Det vil også lage et klarere skille mellom roboter og CAS som gjør at datastrømmen blir lettere standardisert.

Et problem som er nevnt i mange av tidligere rapporter, er at koden er kompleks og omfattende å sette seg inn i. Når koden blir for kompleks, uten tilstrekkelig lesbarhet og kommentarer, så vil det kreve mye tid å sette seg inn i systemet, og gi mindre tid til videre arbeid.

Med å strukturerer koden på denne måten vil RH forhåpentligvis bidra til å

gjøre dette problemet mindre.

Feilsøking var også en stor motivasjon for å lage RH. Hvis man vet at CAS fungerer, så er det bare å få til basisfunksjonene fra RH med den aktuelle roboten. Hvis NXT-roboten hadde blitt lagt direkte til i CAS hadde feilsøkingen blitt adskillig vanskeligere. Det er enkelt å teste roboter manuelt fra oppgavelinjen med å bruke RH, slik at eventuelle feil blir oppdaget før man bruker CAS.

Det er ikke noen gode grunner til å ikke innføre RH. Men det er vanskelig å få det til i et system der IR-roboten er såpass integrert. Og det er ment som en måte å videreføre arbeidet på. Hvis neste student ikke ønsker å bruke denne strukturen når han eller hun jobber med IR-roboten, vil det hemme arbeidet videre med NXT-roboten, som igjen kan være en viktig del av å få til samarbeidende roboter. Se kapittel 7.1 for diskusjon om videre arbeid.

I tillegg krever RH standardiserte verdier mellom CAS og robotene, men dette sees som en fordel der forskjellige enheter bidrar til forvirring. En oversikt over standardiserte variabler er gitt i tabell B.6.

Selv om alle robotene må kunne utføre basisfunksjoner, er det muligheter for å utføre flere funksjoner. Oversikten over mulighetene gitt fra RobotHandlers side er gitt i tabell B.5.

Grensesnittene mellom RH og robotene blir beskrevet i neste delkapittel, 3.2.

3.1.3 Basisfunksjoner

Fordi RobotHandler er et bindeledd mellom roboter og CAS, er det nødvendig å definere et sett med basisfunksjoner. Basisfunksjoner blir gitt i tabell B.1.

For en beskrivelse av disse funksjonene, se tabell. Det gis her en begrunnelse for hvorfor disse funksjonene er gitt som basisfunksjoner.

ping En enkel og vanlig funksjonalitet som kan beregnes som en nødvendighet selv om den ikke er nødvendig for å kjøre roboten.

getpos Posisjonen må kunne hentes for å kunne bli oppdatert i kartet. Dette er altså den estimerte posisjonen til roboten og siden fullscan er lokal data er denne funksjonaliteten helt nødvendig for kartgenerering.

setpos Blir brukt i begynnelsen for å sette posisjonen til startposisjon og senere ved returnering til ladestasjon for oppdatering av posisjon.

fullscan Skanning av omgivelser. Kravet er at det må gis indikasjon på omgivelsene. Flere punkter gir bedre kartgenerering, men det er ikke noe krav til antall punkter.

getrobotparams Henter parametre relatert til roboten som brukes. Det må derfor være en basisfunksjon for å standardisere CAS.

connect og disconnect Til- og frakobling av en robot. Hvilken robot som tilkobles og hvordan det skal velges blir avgjort i RH.

iserror En funksjon som må brukes for å finne feil på roboten. Siden CAS ikke skal bry seg om hvilken robot som brukes må dette være standardiserte feilmeldinger som typisk timeout og kollisjonsvarsel.

3.2 Grensesnitt – Individuelle roboter

Grensesnittene til de individuelle robotene er først og fremst for å holde RH ryddig og lettlest.

Fremtidig arbeid kan raskt legge til robotene i RH, og dermed CAS, når de har kode som fungerer i forhold til roboten som skal brukes.

Denne oppgaven handler om NXT-roboten og det ble derfor utviklet et grensesnitt til denne roboten. Men når RH ble integrert i CAS ble IR-robotens kode byttet ut med RH, og denne koden ble tatt vare på i funksjonen *IRInterface()*.

Hvis en enkel løsning som dette hadde blitt brukt i tidligere arbeid, ville det vært mye enklere å legge til NXT-roboten. Dette er ikke bare på grunn av oversikt, men også fordi det er lettere å skrive riktige kommentarer til inn og ut data.

3.2.1 NXTInterface

Grensesnittet til NXT-roboten heter NXTInterface. NXTInterface var det første steget mot strukturering under arbeidet med å integrere NXT-roboten i CAS. All kode og parametere tilknyttet NXT-roboten er samlet her.

Alle variablene tilhørende NXT-roboten blir derfor lagret i NXTInterface som såkalte ≪persistente≫[2] variabler, og nødvendige get-/set-funksjoner er implementert.

Et eksempel på bruk av NXTInterface er på denne formen:

```
[out error] = NXTInterface('rotaterobot',angle)
```

Antall argumenter avhenger av ønsket funksjon og oversikten over mulige kall finnes i tabell B.4.

Posisjonsestimering

Roboten skal ha et estimat av posisjon som kan settes eller hentes. Det var meningen at dette skulle være på roboten, og det ble implementert og brukt fra roboten.

Men etter bruk viste det seg enklere å ha posisjonsestimeringen i NXTInterface. Grunnen til dette er at posisjonsestimeringen er enkel vektorregning. Grunnen til å ha posisjonsestimering på roboten er at det brukes kontinuerlig posisjonsestimering, som ikke er implementert på NXT-roboten.

Det er implementert en slik løsning på IR-roboten, og på NXT-roboten i tidligere oppgave som prosjektoppgaven Homestad12[10].

Det ble ikke implementert på denne roboten fordi det ikke ble sett nødvendig. Roboten kjører ganske nøyaktig rett fremover med NXTMotor. Se fremdrift i delkapittel 4.1.4.

[2] Nøkkelordet **persistent** i MatLab gjør at variabler blir lagret lokalt i funksjoner.

Posisjonen blir da estimert med å bruke vinkel på robot, lengde kjørt og rotasjonsmatrise.

3.2.2 IR-Interface

Grensesnittet mellom IR-roboten og RH er ikke utarbeidet fordi det ikke var en del av oppgaveteksten. Men det er laget et utkast, IRInterface. Koden som ble tatt ut av CAS ble plassert her.

3.3 Kommunikasjon

Kommunikasjonen mellom robot og RobotHandler varierer fra robot til robot, selv om den største variasjonen bør ligge mellom robot og dens individuelle grensesnitt før kommunikasjonen til RH. Hittil er det kun IR-roboten og NXT-roboten som er benyttet i CAS.

Robotene har forskjellige måter å kommunisere på. Koden på NXT-brikken, NXT-SLAM, ble utviklet med inspirasjon fra NXTMotor. NXTMotor er åpen kildekode fra RWTH Mindstorm, en gruppe studenter på RWTH Acchen universitetet i tyskland[4]. NXT-SLAM blir beskrevet i kapittel 4.

RWTH Mindstorm gruppen har utviklet kode til MatLab som kan brukes til å styre NXT-brikken. Denne koden har blitt brukt i tidligere prosjekter som Bakken08, Tøraasen09, Auestad11 og Homestad12[6, 15, 17, 10].

IR-roboten har blitt bygget opp fra bunnen av med egenprodusert hovedkort og motorstyringskort. Det er derfor forskjellig nivå på kommunikasjonen.

Begge måtene fungerer bra, og blir beskrevet individuelt under.

3.3.1 NXT-roboten

NXT-roboten bruker funksjoner i RWTHMindstormNXT-toolbox, som igjen bruker NXT-brikkens system for kommunikasjon for å kommunisere med PC.

NXTMotor

NXTMotor er programvare skrevet i NXC og kjører på NXT brikken slik som programvaren i dette prosjektet gjør. NXTMotor er åpen kildekode som er underlagt *GNU General Public License*[7].

NXTMotor er laget på RWTHAcchen universitet[4].

Programvaren blir kompilert og overført til NXT-brikken som bitkode, også kalt maskinkode. På NXT-brikken kjører programvaren og svarer på forespørsler fra MatLab.

Grunnen til at dette er en god løsning, fremfor å ha all programvare liggende i MatLab er at da kan programvaren på NXT-brikken ta seg av oppgaver som krever lite regnekraft og hurtig reaksjoner. I tilfellet med NXTMotor så gjelder dette kun motorene.

Det er veldig bra for dette prosjektet, og antagelig mange andre prosjekter rundt om i verden, at kildekoden til NXTMotor er åpen kildekode. Dette gjør det mye enklere for andre å få en god innsikt i hvordan MatLab og NXT kan samarbeide, selv om ikke koden brukes direkte. Koden er godt skrevet og dokumentert, og har fornuftig bruk av kommentarer.

Det var tenkt i begynnelsen å bruke NXTMotor og utvide med egen kode. I praksis ble det at egen kode tok over, og motorfunksjonene ble beholdt. Uten å ta utgangspunktet i NXTMotor hadde det vært vanskelig å komme i gang med å programmere NXT-brikken.

Kommunikasjon

Kommunikasjonen mellom NXT-brikken og MatLab foregår med funksjonene beskrevet under.

MatLab:
`rep = NXT_MessageRead();`
`NXT_MessageWrite();`
Argumentene forteller om hvilke innganger og utganger kommunikasjonen foregår på, og om meldingen i NXT-brikken skal slettes eller bare leses. Inngangene og utgangene kalles for ≪mailbox≫.

NXT:
`ReceiveRemoteString();`
`SendMessage();`
Argumentene er meldingen som sendes og om meldingene skal fjernes fra ≪mailbox≫ etter at beskjeden er sendt.

Funksjonene på MatLab siden er en del av RWTHMindstormsNXT-toolbox. Det brukes noen av disse funksjonene igjennom grensesnittet til NXT-roboten. Alle funksjonene som brukes fra RWTHMindstormsNXT-toolbox er beskrevet i tabell B.8.

3.3.2 IR-roboten

IR-roboten er bygget opp fra bunnen av og har derfor en enklere protokoll. Dataoverføringer foregår byte for byte. Et inntrykk av hvordan dette foregår kan sees i eksempelkoden til IR-roboten under fullscan i vedlegg D.1. For mer informasjon henvises det til prosjekter som jobber med IR-roboten direkte. Dette er de fleste av rapportene vedlagt på CD.

3.4 Standardisering og definisjoner

Under arbeidet med systemet var det stor mangel på standardiserte variabler og definisjoner som måtte finnes med feilsøking og gjetning.

Det ønskes her å dokumentere enheter og definisjoner, slik at det skal være lettere å holde orden på hvilke verdier som passer sammen over ulike moduler. Dette har alltid vært nyttig, men er lettere å definere nå som systemet er mer modulbasert.

3.4.1 Aksesystem

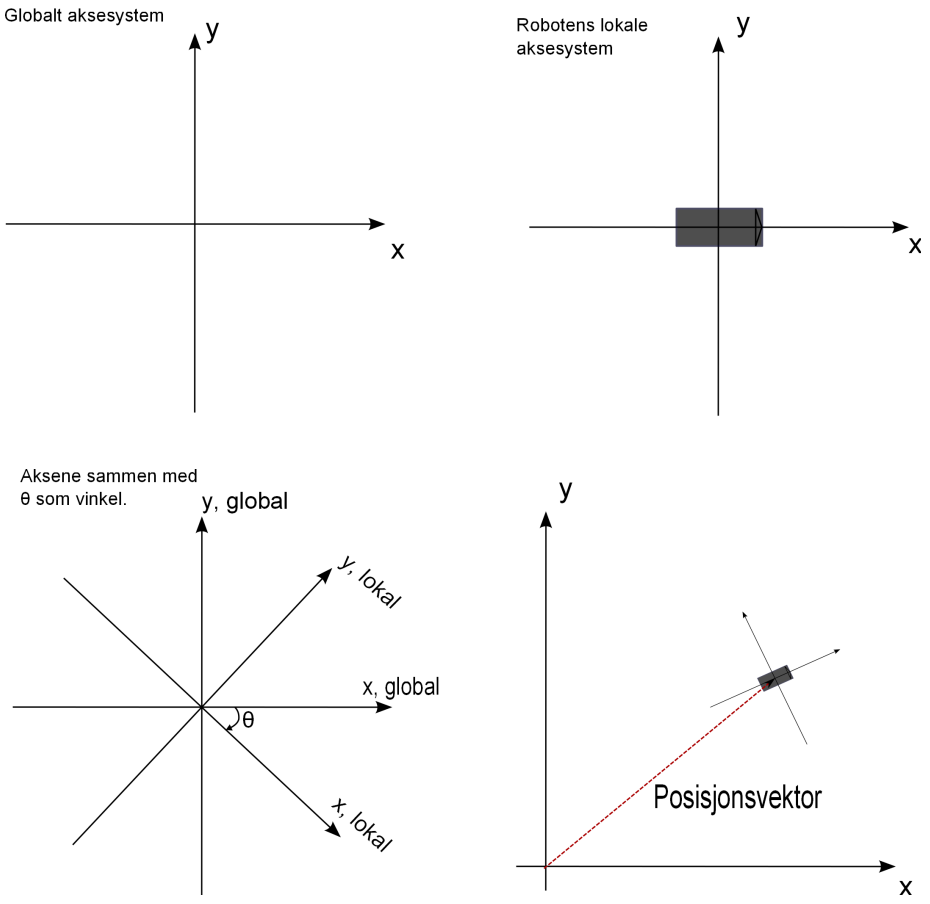
Dette kan virke enkelt, men er absolutt nødvendig å holde orden på under videre arbeid, og er derfor tatt med i rapporten.

Robotens positive x-akse ligger i front, slik at positiv y-akse er til venstre for roboten. Posisjonen til roboten er gitt av en radvektor, $[x,\ y,\ \theta]$, der θ ligger mellom global x-akse og lokal x-akse, og posisjonsvektoren er gitt fra origo til det globale koordinatsystemet til det origo i det lokale koordinatsystemet. Figur 3.2 gir en oversikt over aksene.

3.4.2 Fullscan

IR-roboten gav målingene i det globale kartet ved å bruke posisjon og orientering. Men CAS bruker lokal data, og en rotasjon og translasjon av målepunktene ble utført etter at fullscan returnerte måledata. Fordi IR-roboten gjorde dette, gjør også simuleringen dette.

Dette er ikke en fornuftig måte å gjøre dette på, og RobotHandler vil derfor gi lokal måledata direkte. Tidligere databehandling blir lagt inn i simulatordelen av *fullScan()* og koden blir også tatt vare på i IRInterface. Filen IRInterface.m inneholder all kode som er fjernet fra CAS i forhold til IR-roboten.



Figur 3.2: Koordinatsystemene slik dem er definert i CAS og RH.

Lokal data er måledata relativt til robotens koordinatsystem i både orientering og avstand.

Målingene gjøres fra robotens positive x-akse, mot robotens positive y-akse. Målingene er punkter rundt roboten, men må sorteres etter grader. Dette gjøres på IR-roboten og simuleringen med å ta måleserien og sortere etter de fire sensorene som blir brukt. Fra RH blir det nå sortert med stigende tall fra og med 1 til slutten av måleserien. Antall sensorer er uvesentlig. Sensornummer indikerer rekkefølge på målingene, og ikke hvilken sensor som brukes. Selv om det ofte gjelder sensorene også da sensor 1 oftest er før sensor 2.

3.4.3 Posisjon

Posisjonen til roboten lagres i både roboten og i CAS. Roboten lagrer posisjonen basert på bevegelse. Når en robot får beskjeden om å rotere og å kjøre fremmover, er det det samme for CAS hvordan den nye posisjonen blir estimert, eller bestemt, så lenge det lagres på andre siden av RH.

Det er tre funksjoner knyttet til posisjon i CAS, der RH brukes:

```
[out,error] = getRobotPose( );
clearPose();
[out,error] = setRobotPose( );
```

Inn- og ut-data fra disse funksjonene er i $[mm]$. I RH blir det brukt $[m]$, men posisjon blir transformert til $[mm]$ inn til CAS. Se tabell B.6.

3.4.4 Bevegelse

Bevegelsen til roboten i CAS består av rotasjon, og deretter fremoverkjøring. Funksjonen som brukes til dette er `[out,error] = setRobotTarget( )`. Input er ønsket vinkel på roboten i forhold til det globale kartet, altså vinkelen mellom x-aksene til det globale og det lokale koordinatsystemet, gitt i $[rad]$. Se figur 3.2. Og ønsket distanse roboten skal kjøre fremover, gitt i $[cm]$.

Vinkelen roboten skal rotere, θ, er gitt som argument: $\frac{\theta}{2}$. Dette er fordi IR-roboten hadde et system slik at vinkelen ble gitt som halvparten av ønsket vinkel. Dette burde selvsagt bli rettet opp i CAS, men det lyktes ikke fordi det var såpass dypt i alle utregninger. Systemet ble laget slik at vinkelen ut er halvparten. Derfor ble det enkelt og greit rettet opp i tilkoblingen mellom CAS og RH. Det er lagt til tilstrekkelig med kommentarer i programvaren CAS og RH for å dekke dette.

I *setRobotTarget()* blir det enten sendt kommando videre til simuleringen, eller til RH, ettersom hva som er tilkoblet.

Kapittel 4

Robot – programvare

Både IR-roboten og NXT-roboten har lokal kode installert. Dette er for å dele opp oppgavene slik at roboten utfører gitte kommandoer lokalt uten at programvare på PC trenger å styre dette, som for eksempel motorstryring.

4.1 NXT-brikken – programvare

NXT-SLAM er koden som kjører på NXT-brikken.

Det finnes et stort utvalg av muligheter når man skal lage kode til NXT-brikken. For å velge riktig alternativ til dette prosjektet ble det sett på forskjellige alternativer som vises under.

Valget falt på NXC, Not eXactly C, som blant annet er gratis og greit dokumentert.

4.1.1 Programmeringsmetoder

Av de mange mulighetene for å programmere NXT-brikken på, er nok det mest vanlige LEGO's eget programmeringsspråk, NXT-G. NXT-G er et visuelt programmeringsspråk som er mulig å bruke uten tidligere erfaring innen programmering. Selv om det i utgangspunktet ser veldig enkelt ut, er det mange muligheter med denne måten å programmere på, men i dette prosjektet blir det nødvendig å finne et språk med mer kompleksitet. Det er mye å velge mellom, og det finnes kommersielle og ikke-kommersielle alternativer.

En oversikt over forskjellige alternativer kan finnes på nettsiden `teamhassenplug.com`[9] [1].

I dette prosjektet er det nødvendig med et programmeringsspråk som gir mulighet for kommunikasjon både til NXT-brikken fra PC, og fra PC til NXT-brikke. Det må være mulig å utføre alle ønskelige oppgaver og programvaren skal kunne utvides. Programvaren skal kjøre på NXT-brikken og utføre handlinger etter forespørsel fra MatLab. Koden må ikke ta for mye plass og den må støtte tredjeparts sensorer fra HiTechnic[1].

MSDN Microsoft Robotics

I microsofts robotic development studio kan NXT brikken programmeres med et visuelt programmeringsspråk[12].

Men denne programvaren kjøres på PC og sender kommandoer til NXT. Det vil derfor ikke være et passende alternativ for dette prosjektet.

RobotC

RobotC er et kommersielt programmeringsspråk for undervisning og konkurranser basert på C.

[1] Dette er en oversikt over mulige måter å programmere NXT-brikken på, men det kan godt finnes flere måter.

RobotC trenger en annen type firmaware enn det andre programmeringsspråk krever for å bruke avansert funksjonalitet.

ROBOTC er en IDE med inne bygget kompilering. Her blir altså programkoden som man lager kompilert og lastet ned på NXT brikken.

RobotC støtter tråder. Det kalles ≪task≫ og er samme som en ≪void≫ funksjon, men er en egen tråd. Kalles med ≪StartTask(*taskName*)≫. Det er maksimum 20 task som kan defineres.

Ifølge nettstedet `teamhassenplug.com` er RobotC klart raskere enn andre alternativer. Men programkode kan optimaliseres og det er forskjellige løsninger på problemer slik at hastighetstestene på denne nettsiden ikke kan bli tatt som en klar fasit.

Men fordi RobotC bruker egen firmware, vil RobotC kunne kjøre raskere enn alternativene som bruker den tregere firmware versjonen.

Det negative med RobotC er jo da at det er kommersielt og koster penger. Det er en prøveversjon gratis i 30 dager, men etter det koster det omtrent 50 dollar for én lisens i ett år[3]

NXC - Not eXactly C

NXC er, som navnet antyder, likt C. Det er ikke et generelt programmeringsspråk fordi det blir begrenset til NXT-brikken. Koden blir kompilert til bit-kode som kjøres på NXT brikken. NXC ligger over programmeringsspråket NBC. NBC, Next Byte Code, er et assembly liknende språk.

For å programmere i NXC kan man benytte IDE'en ≪Bricx Command Center≫. Her kompileres koden og overføres til NXT-brikken. Koden kan kjøres og stoppes fra ≪Bricx Command Center≫, eller koden kan starte seg selv på NXT-brikken.

Samme som RobotC, så støtter NXC også tråder, ≪tasks≫.

4.1.2 Valgt programmeringsspråk

Valget bestod i hovedsak mellom RobotC og NXC, selv om andre alternativer ble vurdert som for eksempel LabView. Grunnen til at valget falt på et tekstbasert programmeringsspråk fremfor et visuelt programmeringsspråk var at det vil være lettere å holde orden på etter en viss kompleksitet og at all tidligere kode var tekstbasert.

Det kan være lettere å jobbe videre med et tekstbasert programmeringsspråk, gitt at det blir kodet på en fornuftig måte. Det er fullt mulig å jobbe videre med et grafisk programmeringsspråk også, men det er stor sannsynlighet for at fremtidige studenter har erfaring med C/C++. Og det er ikke nødvendigvis alle som har erfaring med LabView.

Mellom RobotC og NXC ble valget NXC. Dette var fordi RobotC er kommersielt og det ikke ble funnet noen gode grunner til å ikke velge NXC i dette prosjektet. Det er ingen åpenbare forskjeller mellom IDE'ene til RobotC og NXC heller. Det er mulig det er gode grunner til å velge RobotC i konkurransesammenheng fordi det finnes egen firmware og at det kan se ut til at det er raskere kjøretider med RobotC.

Videre motivasjon for å bruke NXC er at fremtidige studenter kan begynne å jobbe uten å tenke på lisens.

4.1.3 NXTSLAM

NXTSLAM er navnet på koden skrevet for NXT-brikken.

Funksjonene på NXT-brikken

Oversikten over alle funksjonene som er implementert og kjørbare fra MatLab på NXT-brikken vises i tabell B.3. Funksjonene kan kalles direkte, men det anbefales å bruke NXTInterface, se delkapittel 3.2.1, for å tilkalle funksjoner.

0 PROTO_MOVE

1 PROTO_MEASURE

2 PROTO_COMMUNICATION

3 PROTO_DEBUGG

4 PROTO_PING

5 PROTO_ACCESS

Tabell 4.1: Tabellen viser hoveddelene av koden til NXT-brikken med tilhørende tegn i hierarkiet. Noen av delene ble fjernet fra programversjonen som er levert på CD, fordi delene ikke lenger var nødvendige, eller lagt til i andre deler.

Funksjonskall – Kommunikasjonshireraki

Funksjonene på NXT-brikken er satt opp i et hierarki, inspirert av NXTMotor, se delkapittel 3.3.1. Dette gjør koden strukturert og enkel å debugge.

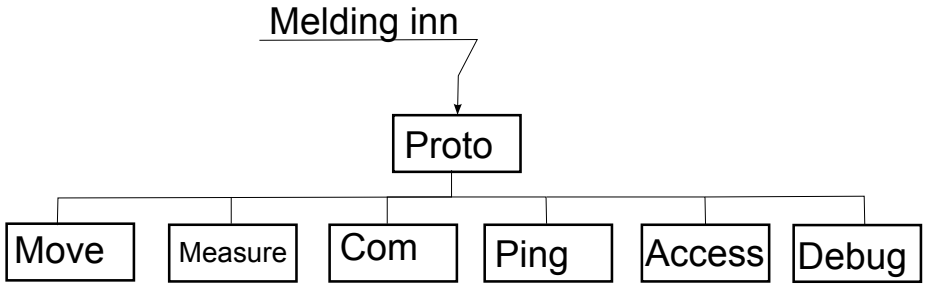
MatLab sender en streng av tegn som blir tolket av NXT-brikken. Denne meldingen blir tolket med å se på et og et av tegnene og sortere for hver tegn etter hierarkiet med funksjoner. Figur 4.1 viser grafisk hvordan dette skjer. Proto[2] er definert som det første tegnet og sorterer meldingene inn i hovedklasser. Under hver hovedklasse er det er det underkategorier. Tabell 4.1 viser hovedklassene som er definert i koden til NXT-brikken.

Dataoverføring

Det er kun når roboten utfører *fullScan()* at det er nødvendig å koordinere dataoverføring. Det blir overført data med for eksempel enkeltmålinger og kalibrering også, men dette kan gjøre med en enkelt melding.

Med *fullScan()* skal målingene fra hver av sensorene, sammen med tilhørende

[2]Proto – betyr ≪først≫



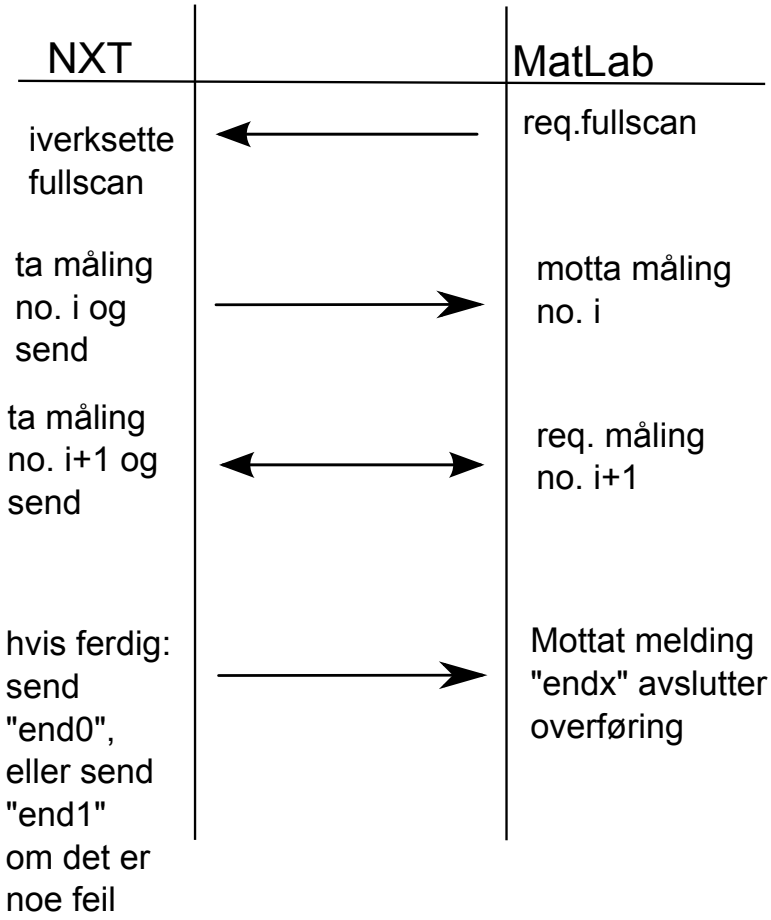
Figur 4.1: Figuren illustrerer hvordan meldingene blir sortert etter hvilken funksjon som blir forespurt.

informasjon, sendes over til MatLab for hver inkrementering av måletårnet. Dette blir koordinert med MatLab slik at hver måling blir sendt og et svar fra MatLab blir mottatt før ny måling blir sendt. Figur 4.2 viser grafisk hvordan denne kommunikasjonen foregår.

Data må sendes pakkevis fordi det er en grense på 58 bytes per sending. Dette har med kapasiteten til NXT-brikkens ≪mailbox≫ å gjøre. Hvis det sendes mer enn 58 bytes blir det feil.

Parsing av datapakker

Hver melding fra NXT-brikken inneholder måledata og nummeret på målingen. Informasjonen blir sendt med formatet:
`s1xxxxms2xxxxmfi`
Dette blir sortert på MatLab siden. Fordi målingene kan variere fra 0 til 9999, og målenummeret variere mellom 0 og 99, må koden sortere ut informasjonen som er nødvendig. Koden tar ett og ett tegn om gangen og setter flagg som indikerer hva som kommer etter. Tabell 4.2 forteller hvilke tegn som blir brukt som indikatorer.



Figur 4.2: Figuren illustrerer dataoverføringen mellom MatLab og NXT-brikken under *fullscan()*.

s – indikerer sensor

1-2 – indikerer hvilken sensor det gjelder

m – slutt på målingsdata

f – indikerer at resten av meldingen er nummeret på målingen

x – måledata

Tabell 4.2: Tabellen viser tegnene som brukes for å sortere ut informasjonen fra datapakkene fra NXT-brikken til MatLab.

4.1.4 Rotasjon og fremdrift

Rotasjon og fremdrift er kanskje de viktigste funksjonene for roboten i forhold til kartleggingen, hvis man ser bort ifra datainnsamlingen.

Det er enkle funksjoner, men det blir beskrevet kort her fordi det er grunnleggende og essensielle funksjoner.

Fremdrift

Roboten kjører fremover med kommandoen:
`NXTInterface('driveforward',distance)`, der avstanden er i cm.

På NXT-brikken blir beskjeden med tall mottatt og sortert etter funksjon, avstand og hastighet, der hastigheten er valgfri.

For kjøring fremover blir motortask «moveSync» brukt, og denne tråden[3] sørger for at motorene kjører likt. Dette fungerer bra, og gjør at posisjonsestimering forenkles til enkel vektorregning. Hvis motorene hadde kjørt ulikt hadde det vært nødvendig med kontinuerlig posisjonsestimasjon.

Avstanden må bli gitt i antall «tikk», kalt tacho. 360 tacho er en omdreining

[3] NXC tråder blir kalt «tasks»

på hjul. Antall tacho per cm blir,

$$\text{tacho} = \frac{360}{[\text{omkrets til hjul i cm}]} \frac{[\text{tannhjul1}]}{[\text{tannhjul2}]} \cdot [\text{antall cm fremover}]$$

Rotasjon

Roboten kjører til ønsket vinkel og regner selv ut avstand og retning for å kjøre rasket vei til målet.

Rotasjon blir utført med å kjøre hjulene likt, med forskjellig fortegn på hastighet.

Avstanden hjulene må kjøre regnes ut ifra diameter(radius) mellom hjulene, og antall tacho regnes ut på samme måte som for fremoverkjøring.

4.2 IR-roboten – programvare

IR-roboten er bygget opp fra bunnen av, og har egen kode skrevet i C.

Tabell B.7 inneholder en oversikt over hvilke kommandoer som sendes fra MatLab til IR-roboten, og omvendt. NXT-brikken og IR-roboten deler samme basisfunksjonalitet.

Kapittel 5

CAS RH

Dette kapittelet tar for seg CAS med tillegget RH og endringer gjort i dette prosjektet.

5.1 Programvaren i MatLab – CAS toolbox

Når NXT-roboten skulle legges til ble det klart at en større omstrukturering enn planlagt var nødvendig. Dette ble RobotHandler, som er beskrevet i delkapittel 3.1. Hovedløkken i programmet er ikke forandret med hensyn på at data blir behandlet på samme måte.

Forandringene går på hvordan data blir hentet på. CAS har ikke noe med robotene å gjøre lenger, og bryr seg kun om data inn og ut.

5.1.1 Endringene som er gjort i CAS-toolbox

Endringene som er gjort i dette prosjektet med hensyn på integreringen av NXT-roboten er gjengitt i tabell B.2. Endringene er dokumentert med filene som er endret og funksjonen, om det skulle være en funksjon.

Når NXT-roboten ble integrert var det meste av endringene i CAS der RobotHandler ble integrert, og NXT-roboten kommuniserer med RobotHandler.

De endringene som ikke er gjengitt i tabell B.2 er små og skal være klart gitt av nødvendige kommentarer. Det er også lagt til en god del kommentarer i forhold til ny og eldre kode. Det er lagt til kommentarer i toppen av filene for å beskrive funksjonene og når dem sist ble endret. Dette kan virke trivielt, men mangel på dette gir irritasjon ved videre arbeid. Hvis filene ikke har original forfatter og forfattere på endringer er det vanskelig å vite hvor dokumentasjonen kan finnes i form av rapporter, som er aktuelt i dette tilfellet.

Videre er filer med lite beskrivende navn og ubeskrivelige variabler veldig vanskelig å tolke. Derfor er det viktig med kommentarer øverst i filene som forteller hva funksjonen skal utføre og hva som er inn- og ut-data.

5.2 Programkjøring

I tidligere kapittel ble hovedløkken i systemet beskrevet. Se delkapittel 2.2.2. I dette kapittelet blir det beskrevet hva som skjer fra det trykkes ≪connect≫ i GUI, til roboten er ferdig. Det er med fokus på endringene gjort i dette prosjektet, og vil derfor ikke gjelde tidligere versjoner, i motsetning til delkapittel 2.2.2.

connect Når man trykker på **connect** knappen i GUI'et så kalles callback funksjonen *connectButttton_Callback()*. Her sjekkes det om roboten er tilkoblet allerede med SERIALLINK variabelen. Hvis ikke en robot er tilkoblet så kjøres funksjonen *connectTo()*.

connectTo: I denne funksjonen kobles enten simulatoren til, eller RobotHandler kalles med `RobotHandler('connect')`. myCon variabelen blir

brukt for å formidle om det er simulator eller RobotHandler som er tilkoblet med å være enten `'simulator'` eller `'robothandler'`. Om det er RobotHandler eller simulatoren som kobles til avgjøres av valget bruker tar i GUI.

Start Når **Start** knappen trykkes så kobler CAS seg til en robot om den ikke allerede er tilkoblet, og deretter samles det inn verdier fra GUI'et som brukes videre. Med forskjellige valg satt så kalles hovedløkken i funksjonen *lineBeaconSLAM()*.

lineBeaconSLAM: Roboten følger veggen med funksjonene *leftWallFollowerToBackTracker()* og *quasiFollowLeftWall()*. Hovedløkken i *leftWallFollowerToBackTracker()* kjører til roboten er ferdig med å kartlegge. For detaljer om hva som skjer i hovedløkken, se delkapittel 2.2.2.

Ping robot Sender ping til tilkoblet robot igjennom RobotHandler.

Full Scan Gir en test av fullscan uten at systemet trenger å kjøre. Fullscan fra robot skjer igjennom RobotHandler.

Set robot pose: Setter intern posisjonen til roboten igjennom RobotHandler.

Get robot pose Henter intern posisjon til robot igjennom RobotHandler.

Reset robot pose Resetter posisjon og orientering til roboten igjennom RobotHandler.

RobotHandler RobotHandler knappen åpner et nytt vindu som gir tre valg.

Plot data Åpner et nytt vindu der det er mulig å bla igjennom iterasjonene av kartleggingen. Global data, lokal data og posisjon blir plottet ved siden av hverandre. Nyttig for å sammenlikne inn-data med kartlegging.

Calibration Åpner et nytt vindu for å kalibrere sensorer. Foreløpig er det bare NXTInterface som er lagt til, men her bør også IRInterface legges til. Da kan begge robotene benytte kalibreringen.

Export Eksporterer data fra kartleggingen til workspace i MatLab('base' workspace).

Noen av knappene er ikke i bruk for NXT-roboten. Det er fordi det ikke var nok tid til å prioritere mindre viktige egenskaper. Dette er for eksempel manuell

kjøring. Egenskapene som trengs for å kjøre systemet er nevnt som basisfunksjoner. For å lette arbeidet med å integrere nye roboter er disse basisfunksjonene definert som nødvendige. Altså må alle roboter dekke dem før de integreres i CAS. Manuell kjøring er ikke definert som en basisfunksjon, fordi det ikke har noe med kartleggingen til CAS å gjøre.

Knappen **Full Scan** er heller ikke en basisfunksjon[1], men er allikevel implementert for NXT-roboten fordi det tar lite tid og er nyttig i feilsøking. Det er derfor nødvendige funksjoner og valgfrie funksjoner for alle robotene.

5.3 Utvidelse og endringer i GUI

Tidligere GUI var ment for IR-roboten. Det var nødvendig å gjøre endringer slik at RobotHandler fikk funksjonalitet i GUI. Knappen **RobotHandler** vil åpne et nytt vindu med tre valg, beskrevet over i delkapittel 5.2.

Beskrivelsen av GUI utvidelsen, RobotHandler, er beskrevet i vedlegget, A.4.1.

Grunnen til at det ble utvidet med dette, i egne vindu i GUI, er at det føltes nødvendig å dele opp GUI på denne måten. Når flere roboten kan tilkobles må funksjonaliteten til den aktuelle roboten ha eget GUI vindu, da ikke alle robotene har samme funksjonalitet.

Til nå har det bare vært étt vindu i GUI. Det er enklere å programmere med étt vindu, men passer ikke med hensyn på utvidelser og flere roboter. Mer om dette i diskusjon og videre arbeid, kapittel 7.

[1] her menes callback funksjonen som kaller funksjonen *fullscan()*, mens funksjoenen *fullScan()* er allikevel en basisfunksjon.

Kapittel 6

Testkjøring

Det er utført tester med roboten for å sjekke at den faktisk fungerer i CAS. Det er ikke gjort noen forbedringer i CAS med hensyn på kartlegging og navigering. Kartleggingen er derfor det samme som i tidligere prosjekter, og det henvises til disse rapportene for testresultater relatert til navigering og kartlegging. Siste rapport med dekkende avsnitt med testresultater er prosjektoppgaven til Tusvik[16].

Testene her skal vise hvordan NXT-roboten klarer å samle inn data og hvordan CAS mottar data til kartlegging. Global data, transformert fra robotens lokale data, er plottet for å se faktisk måledata i forhold til kartleggingen som er behandlet data. Det er dessverre ikke liknende plot fra IR-roboten for sammenlikning.

6.1 Test 1

Roboten er testet i første omgang på en sirkulær bane og en variert bane. Resultatet er rotete i begge tilfellene der den varierte banen er for dårlig til å kunne sies å være velykket.

Det kan sees ut ifra global data, fra NXT-roboten, at det er rådata som er for dårlig til at en nøyaktig kartlegging kan oppnås i dette tilfellet.

Sirkulær bane

Målene på banen:
diameter = $151cm$

Den sirkulære banen kan sees på figur 6.1. Når roboten hadde gjort én runde på kartleggingen, hadde den gjort omtrent 80% av runden på bordet. Det kan ikke forventes at denne roboten skal klare å ta en perfekt runde, men er allikevel litt høy feil i posisjonsestimatet.

På figur 6.2 vises brukergrensesnittet mot CAS, med kartleggingen. Som det sees av kartet så er det en feil «bak» roboten. Denne feile er lett å forklare når man ser på sensordata fra roboten. Og det kan sees fra kjøring og kartlegging med fysisk robot, og med simuleringer, at det er sensordata som bidrar til feil i kartlegging. Figur 6.3[1] viser sensordata fra roboten. Global data er punktene som finnes fra fullscan, plassert i det globale kartet. En perfekt robot ville derfor ha gitt et pent bilde av omgivelsene. Når global data fra simuleringen plottes passer punktene perfekt med linjene på kartet som genereres.

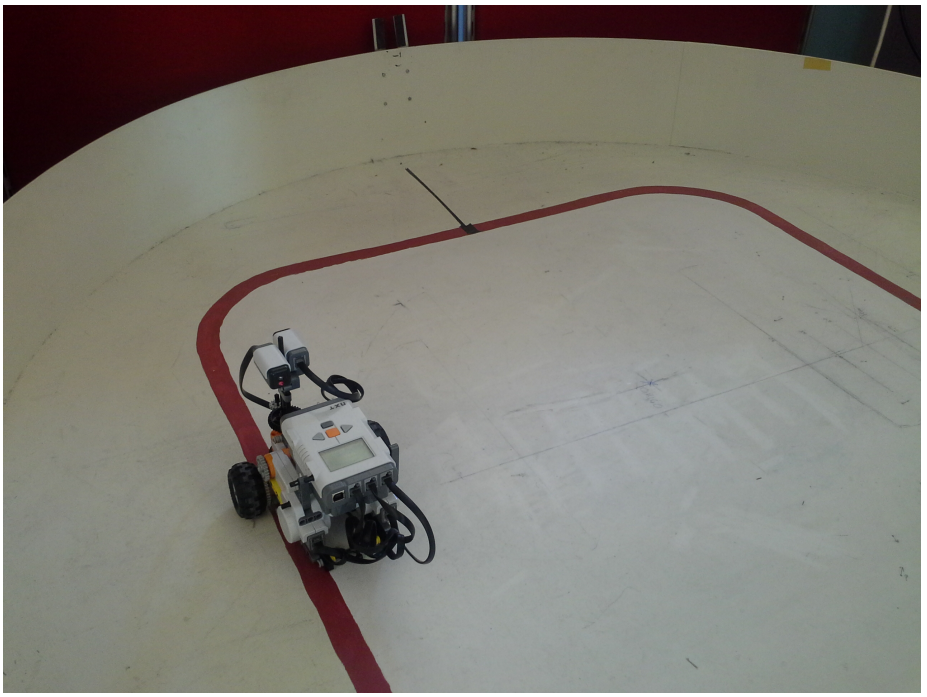
I figur 6.3 er det også lokal data og posisjon. Posisjon er plotting av robotens posisjon i det globale kartet, og lokal data er slik roboten ser en måling i forhold til egne akser.

Global data i denne testen viser at det er en feilmåling i alle tilfellene av fullscan, og som bidrar til en dårlig kartgenerering.

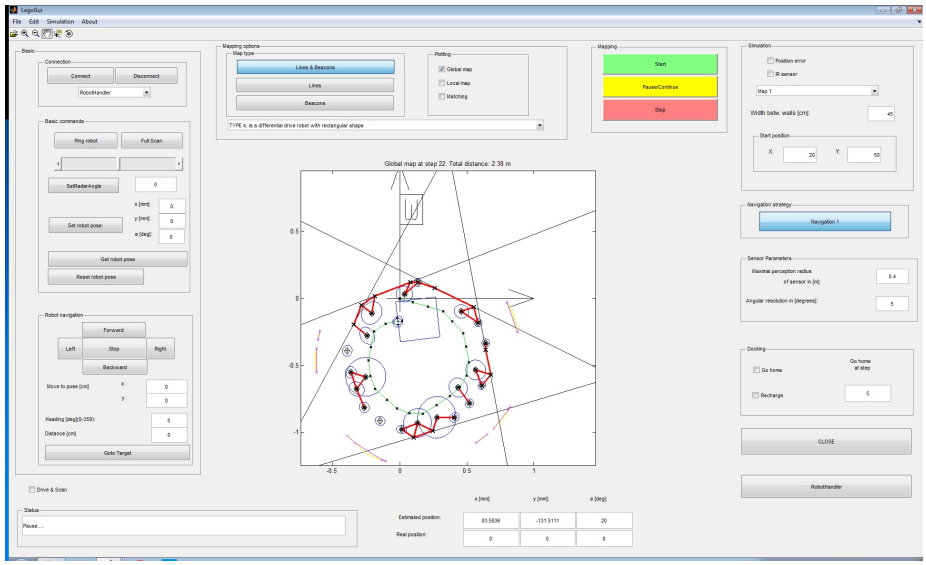
Variert bane

Den varierte banen kan sees på figur 6.4 og kartleggingen kan sees på figur 6.5. Her kommer feilen frem mer tydelig enn i testen over med sirkulær bane. Data vi-

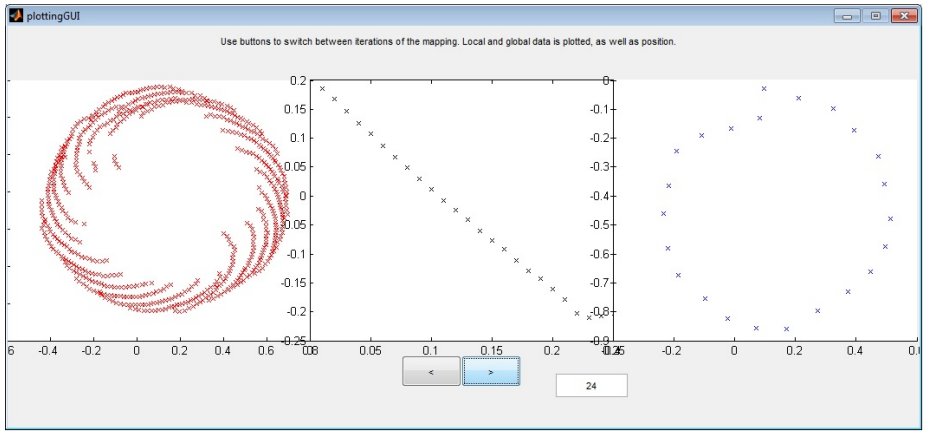
[1] Figuren er av en tidlig versjon. Endelig versjon har merkelapper på dataplottene.



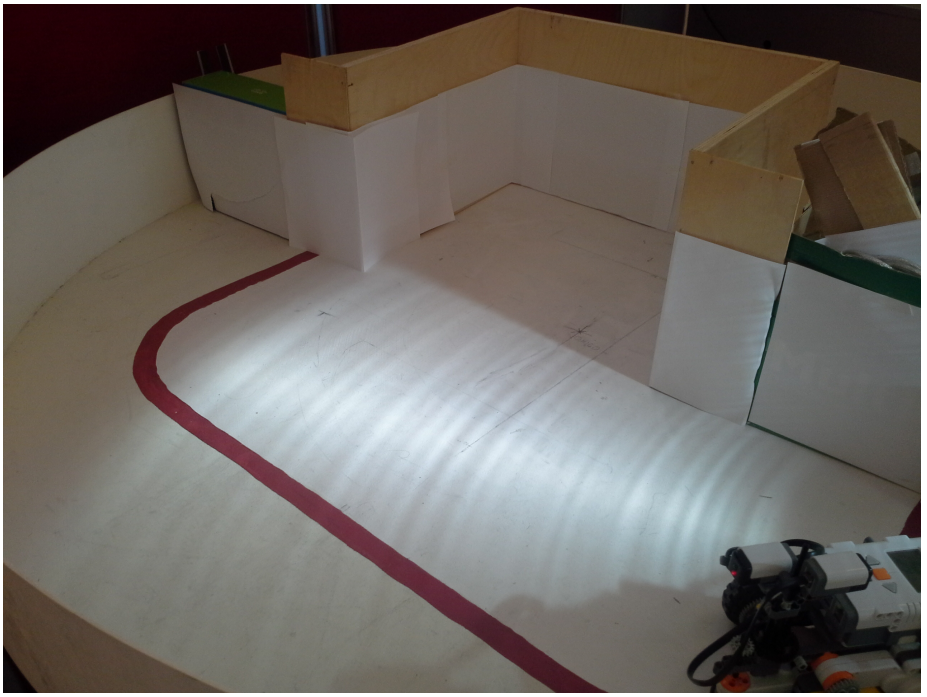
Figur 6.1: Halvdelen av den sirkulære banen med roboten som følger venstre vegg.



Figur 6.2: CAS grensesnitt etter kartlegging.



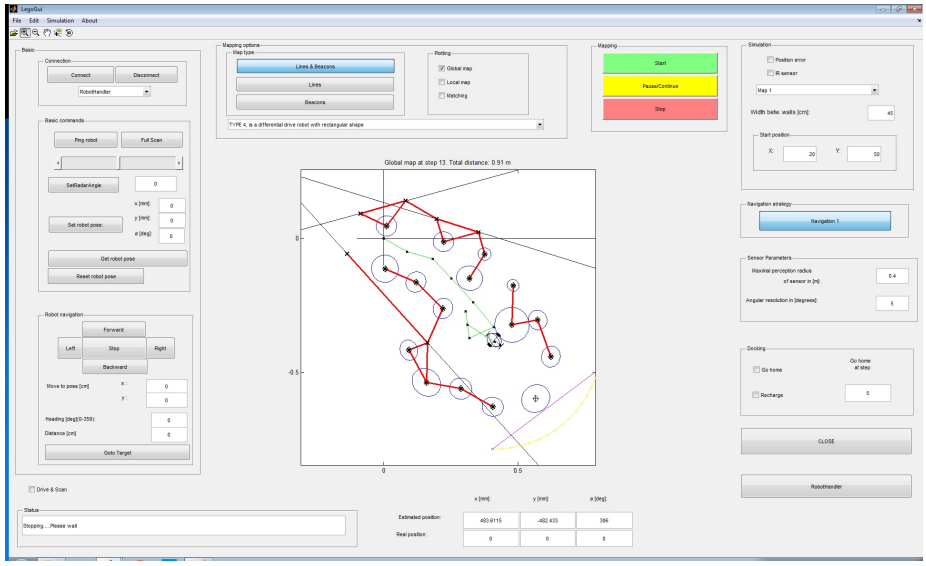
Figur 6.3: Global data til venstre, lokal data i midten og posisjon til høyre.



Figur 6.4: En tilfeldig valgt bane.

ser at sensorene returnerer feil måleverdi. Dette har med vinkler på måleobjekt, avstand, kalibrering og ikke minst materiale på måleobjektene å gjøre. I denne banen er det forskjellig materiale på overflater(A4 ark og hvitmaling) som gir forskjellige måleverdier på samme avstand. Dette i tillegg til alle andre feilkilder gjorde kartgenereringen for dårlig.

Når kartgenereringen er for dårlig kan heller ikke navigasjonen bli god nok.



Figur 6.5: Et forsøk på kartgenerering i variert bane.

6.2 Test 2

Rektangulær bane

Mål på bane:
Lengde: 91cm
Bredde: 92cm

Testresultatene fra test no. 1 ble forsøkt forbedret ved å:

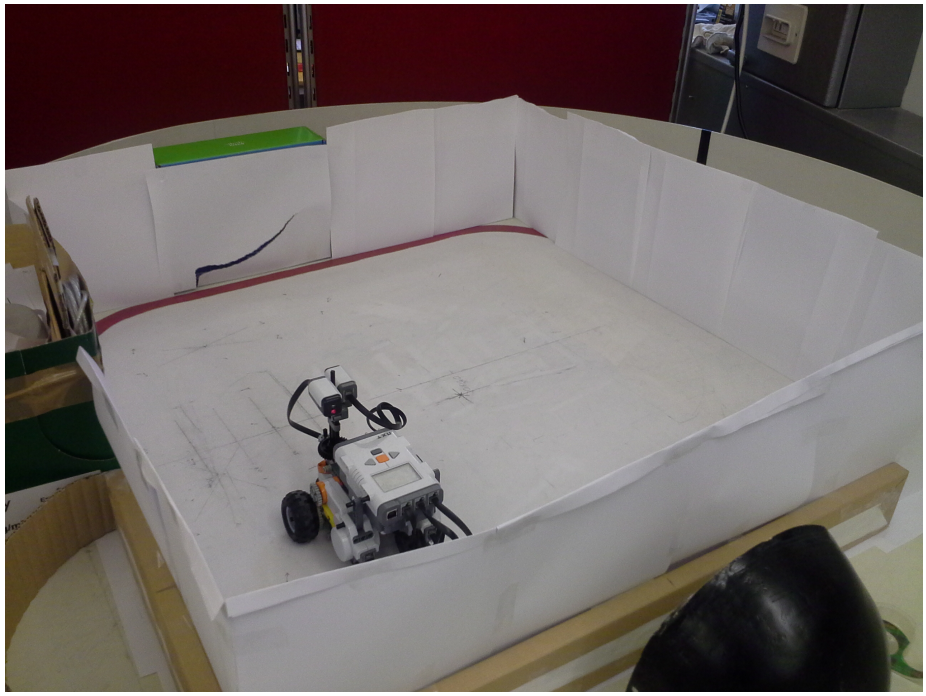
- Kalibrere sensorene på nytt og individuelt.
- Lage en testbane med samme materiale.
- Introdusere en faktor på posisjonsestimatet.

- Reduserte maks avstand på sensorene til 20cm.

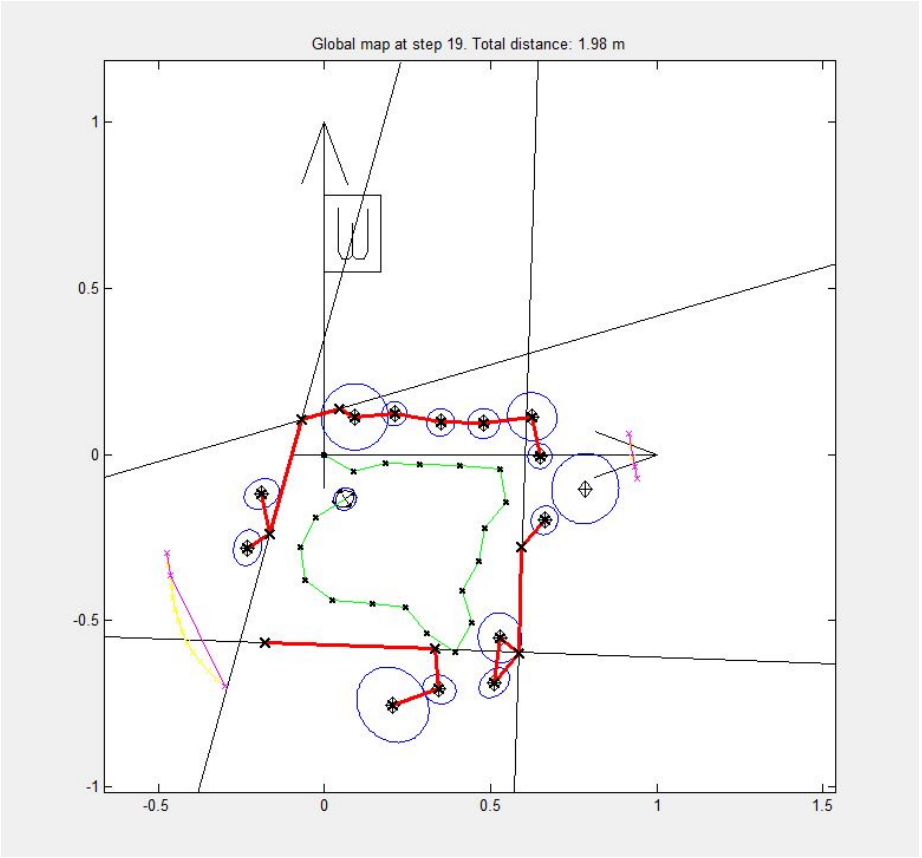
Testbanen kan sees på figur 6.6. Veggene er satt opp med vanlig A4 ark og sensorene er kalibrert mot veggene. Kartleggingen er vist i figur 6.7 og figur 6.8. Plotting av data sees i figur 6.9 og 6.10.

Figur 6.7 viser hvor roboten har kjørt en runde. Dette er før backtrackingen har satt igang. Backtrackingen kjører ned til det nedre venstre hjørnet og tetter igjen dette hullet i kartet, før CAS sier seg ferdig med kartleggingen. Dette kan sees på figur 6.8.

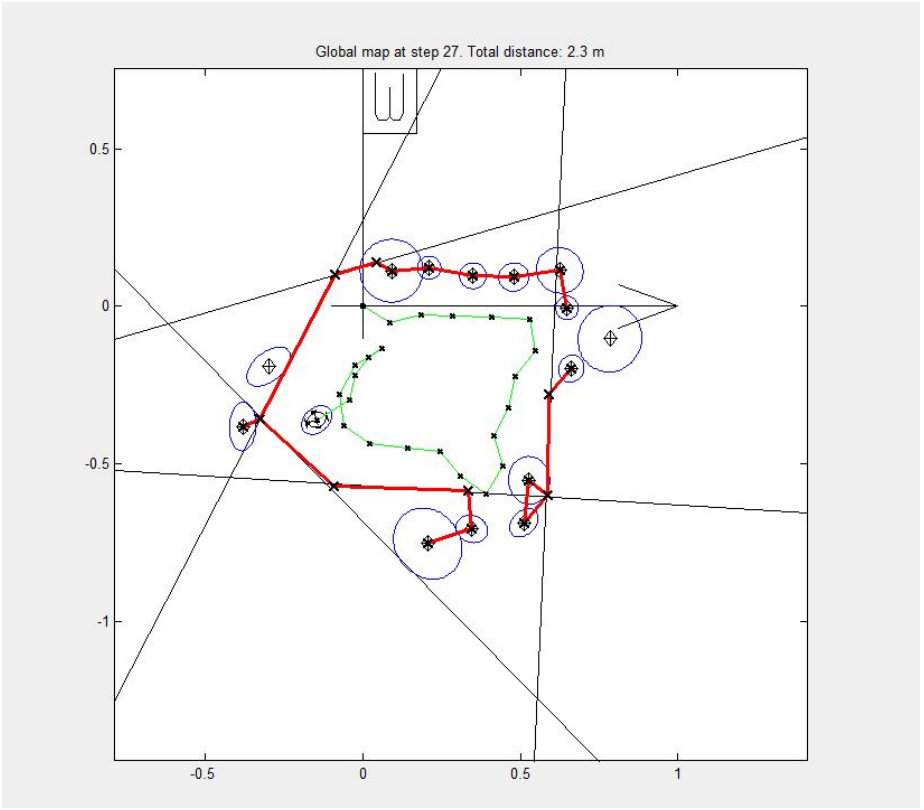
Plotting av data før backtrackingen kan sees på figur 6.9. Her sees det at det mangler måledata i det venstre nedre hjørnet og det kan sees hvordan målefeilen er til stedet. Figur 6.10 viser at datainnsamlingen dekker hele testbanen.



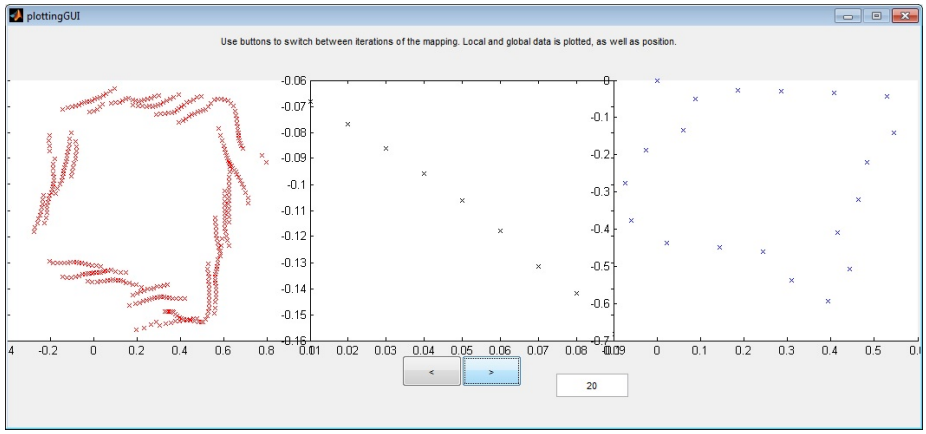
Figur 6.6: En rektangulær testbane.



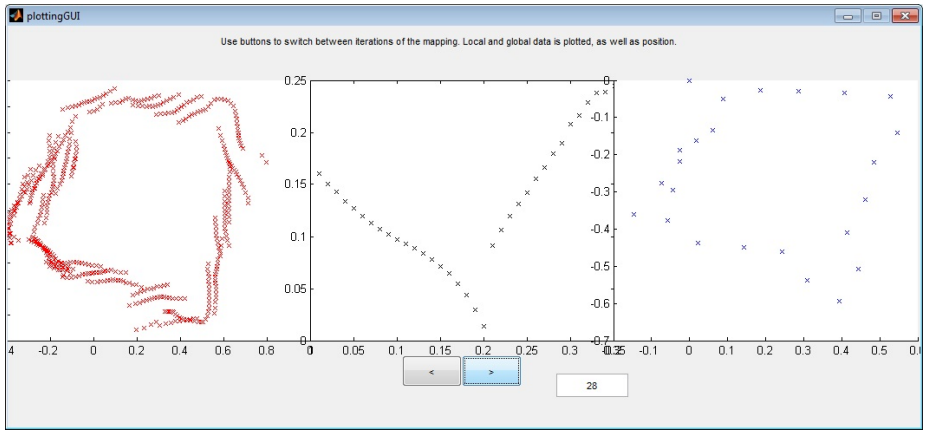
Figur 6.7: Kartleggingen av en kvadratisk testbane uten backtracking.



Figur 6.8: Kartleggingen av en kvadratisk testbane ved fullført kartlegging.



Figur 6.9: Plotting av data uten backtracking i kvadratisk bane.



Figur 6.10: Plotting av data til fullført kartlegging av kvadratisk bane.

Kapittel 7

Avslutning

Det blir her gitt forslag til videre arbeid, og diskusjon rundt deler av prosjektet. Til slutt blir det gitt en oversikt over kjente feil og mangler.

7.1 Videre arbeid

Alternativ navigasjons algoritme

Med en alternativ navigasjonsalgoritme vil det være mulig å kartlegge alle områder, og ha flere roboter samtidig. Hvis flere roboter jobber sammen må det gjøres mer arbeid med CAS, og blir beskrevet nedenfor.

Et forslag til navigasjonsalgoritme er lagt i vedlegg E. Denne algoritmen ble ikke fullstendig utarbeidet og har ikke blitt implementert/testet. Det er mulig den er egnet til videre arbeid, men det må bli tatt en vurdering på.

Flere roboter

Det har blitt forsøkt med flere roboter samtidig i oppgaven til Hannaas i 2011, men det ble ikke velykket.

Etter arbeidet i dette prosjektet er systemet, forhåpentlig vis, mer oversiktlig og modulisert. Det kan derfor være lurt å prøve videre med flere roboter. En ny algoritme, og videre arbeid med CAS i forhold til håndteringen av flere roboter vil da være nødvendig.

Videre modulisering og opprydding

For at koden skal bli lettere for fremtidige studenter å jobbe med, kunne det vært lurt å modulisere koden enda mer. Simulering, kjøring med måling m.m. bør moduliseres slik at det ikke ligger blant annen kode.

Alle funksjoner burde kunne byttes ut med forbedret kode, eller anderledes kode, uten å forandre på resten av koden. Som for eksempel navigeringen.

Robot Handler har et sett med basisfunksjoner som alle roboter skal ha. Men det bør i tillegg være ekstrafunksjoner for robotene som har dette. Det kan være manuell kjøring og måling, eller andre ting som ikke er en del av den autonome kjøringen.

Hvis navigasjonsalgoritmen brukes videre burde den bli ryddet opp med kommentarer og omprogrammeres slik at ønsket vinkel blir gitt i θ fremfor $\frac{\theta}{2}$.

Utvidelse av GUI, RH

Et eget GUI vindu bør åpnes for den enkelt robot fordi det kan være forskjellige ekstrafunksjoner. Det kan testes hvilken robot som er tilkoblet og dermed åpne tilhørende vindu. For NXT-roboten blir dette en videre utvikling av GUI, mens for IR-roboten må det lages nytt.

Dette vinduet vil da inneholde alle ekstrafunksjoner denne roboten kan håndtere. Nå er det slik at ekstrafunksjoner ligger i GUI vinduet. Dette er et problem da det ikke er gitt at nye roboter har samme funksjonalitet.

Roboten

Sensorene kan byttes ut med samme sensor som brukes på IR-roboten. Det ble gjort arbeid med dette i prosjektoppgaven til Bakken08[6], som kan videreføres.

7.2 Diskusjon

Hva som var bra, og hva som kunne blitt gjort anderledes blir gjennomgått i dette kapittelet i tillegg til tips for videre arbeid.

7.2.1 NXT-roboten

NXT-roboten ble laget for å kunne kjøre uten hjulspinn, ta 360°måling og navigere uten problemer. Hjulspinn er ikke et stort problem i dette prosjektet. Men det kan skyldes hastighet i tillegg til vektfordeling og konstruksjon.

I starten av prosjektet var tårnet for lite og motoren stod i 90° vinkel i forhold til omdreiningen til tårnet, figur 1.2. Motoren ble snudd og tårnet ble forsterket.

Men med to sensorer og to stive kabler er ikke motoren og tårnet stivt nok til at målingene blir helt nøyaktige. Etter en fullscan er tårnet forvridd noe. Det kan løses lett med å sette på pause og stille tilbake tårnet, men dette er ikke ideelt i forhold til at roboten skal være autonom. Det ble forsøkt å bruke aktiv brems på motoren for å holde igjen tårnet når kablene drar, men dette var ikke nok alene til å løse problemet. Dette kan sees på video vedlagt på CD.

Måten roboten får tatt fullscan på bør forbedres. Dette kan gjøres med å gire ned overgangen fra motor til tårn mer enn det er idag, og å bytte kablene med

mykere kabler. Det sistnevnte er lettere sagt enn gjort fordi pluggene er spesielle for Lego og må spesialbestilles.

7.2.2 NXC

NXT-roboten er programmert i NXC, og funksjonene som er implementert er testet og fungerende. Men det ble brukt for mye tid på å lage og teste denne programvaren i forhold til utbytte.

Det sees i ettertid at NXT-roboten hadde vært mer robust og funksjonell om all programvare lå på PC siden, nærmere sagt i NXTInterface.

Grunnen til å lage kode lokalt er at man får fordelt oppgavene slik at PC ikke trenger å bruke regnekraft på funksjonalitet lokalt på roboten. Det er mange grunner til å lage programvare i NXC og kjøre det på NXT-roboten. Men det som ble definert som **basisfunksjoner**, etter at NXT-roboten var ferdig utviklet, kan kjøres fra PC.

I det tilfellet ville roboten kjøres fra programvaren RWTHMindstormsNXT-toolbox [4]. RWTH er utviklet av flere studenter igjennom flere år. Det er godt testet og er en god løsning når nødvendig funksjonalitet er definert.

RWTH kan brukes for å utføre alle funksjonene som er laget på NXT-roboten, og det hadde vært mindre tidkrevende å teste og debugge kode laget i MatLab enn på NXT-brikken. Den første stunden av prosjektet ble det brukt en eldre NXT-brikke uten fungerende skjerm som gjorde kodingen i NXC mer utfordrende enn nødvendig. Hvis all kode hadde blitt laget med funksjoner fra RWTH som i prosjektene Tøraasen09, Auestad11 og Homestad13 [15, 17, 10], ville det være mer tid til å jobbe med andre deler av prosjektet.

Det må nevnes at det ikke nødvendigvis er den beste løsningen å kjøre roboten fra RWTH i fremtidig arbeid. Det vil være avhengig av hvilken funksjonalitet som er planlagt. Et eksempel er fullscan som kjører ganske tregt slik det er satt opp nå(se video). Med å bruke en kontinuerlig metode: kjøre tårnet 180° uten stopp, og måle for hver 5°, kan bluetooth tilkoblingen være for treg til at målingene blir nøyaktige. Da er det nødvendig i lage koden lokalt på

NXT-roboten.

Til videre arbeid kan det derfor være en god løsning å fortsette med å utvikle NXC-koden. Programvaren i dette prosjektet er allerede utviklet og testet så videre utvikling vil være lettere enn å begynne på nytt med RWTH. Med lokal kode vil flere muligheter være åpne, men til gjengjeld krever det mer tid og feilsøking.

7.2.3 CAS – før og nå

I tidligere prosjektoppgave, Bakken08[6], ble det forsøkt å integrere legoroboten i CAS, uten at det lyktes helt. I masteroppgaven til Hannaas11[8] ble det forsøkt å integrere en NXT-robot med kamera. Det ble utført en enkel test for å legge til punkter til kartgenerering i CAS, men det kommer ikke frem av rapport eller programkode at roboten var mobil.

Dette prosjektet ser ut til å være det første der en ny fungerende og mobil robot er velykket lagt til i CAS. Rettere sagt integrert, der lagt til impliserer at systemet er som før. IR-roboten fungerer ikke nå på grunn av modifiseringen i systemet.

Før

CAS har med IR-roboten blitt forbedret fra prosjekt til prosjekt. Men det har altså ikke vært nok fokus på flere roboter, eller forskjellige roboter.

Fokuset har vært på å få til god kartlegging med IR-roboten, som førte til problemer med å integrere nye roboter.

Systemet kan trenge en ny navigasjonsalgoritme, men ellers er det meste relatert til kartlegging og GUI bra.

Nå

Arbeidet med å integrere NXT-roboten kan sees på som to faser. Den første fasen var en «rett frem»-løsning der roboten ble forsøkt lagt til direkte i programkoden CAS.

Dette ble ikke velykket. Arbeidet var også demotiverende fordi alt som ble gjort førte til en mer rotete kode. Det ble klart at det var nødvendig med struktur, og NXTInterface ble laget. NXTInterface sørget for å fjerne kodeblokker tilhørende NXT-roboten ut av CAS, men systemet så fremdeles rotete ut. Og var lite egnet til å leveres videre til fremtidige prosjekter.

Fase to bestod av å ta et skritt tilbake og utarbeidet en plan på ny ønsket struktur i programvaren. RobotHandler ble utviklet og NXTInterface ble integrert i RobotHandler. Å få roboten til å fungere igjennom RobotHandler fra CAS ble forventet å være enkelt.

Dette viste seg å være grovt feilberegnet. Men etter å få dokumentert hvilke verdier som skal inn og ut av CAS, samt mye frem og tilbake mellom CAS og RobotHandler, ble integreringen velykket.

Problemet var at inn- og ut-data, som var tilpasset IR-roboten, ikke var intuitiv. Mangel på dokumentasjon og kommentarer gjorde dette arbeidet vanskelig med at problemer forplantet seg fra CAS til NXT-roboten. For eksempel krever IR-roboten halvparten av rotert vinkel. Det vil si at i CAS er vinkelen θ, mens vinkelen ut til roboten er $\frac{\theta}{2}$. Dette gjør at antagelser om inn og ut data fort blir feil.

Dette eksempelet på vinkel fra CAS til IR-robot ble forsøkt tilpasset i NXT-roboten, som igjen førte til feil i blant annet posisjonsestimeringen. Feil mange steder gjør at feilsøkingen blir vanskelig.

Det ble til slutt bestemt å få CAS og RobotHandler til å fungere sammen, og definere inn- og ut-data til RobotHandler. Da ble det fornuftige verdier mellom RH og robotenes individuelle grensesnitt. Kommunikasjonen fra robot til CAS er ikke perfekt med hensyn på forskjellige enheter mellom modulene. Men når RH og CAS fungerer, og kommunikasjonen mellom RH og robotene fungerer og

er intuitiv, så er det en god start. Det hjelper også godt at inn- og ut-data er dokumentert.

7.2.4 RobotHandler og NXTInterface

Som beskrevet over i delkapittel 7.2.3, ble RobotHandler og NXTInterface utviklet ut ifra et ønske om en strukturert programkode.

Resultatet ble bra, og føles nødvendig og naturlig. Når det innføres nye elementer i et system bør det være godt begrunnet da flere deler kan gi problemer i videre arbeid.

RobotHandler

Utviklingen av RobotHandler var relativt enkel da dette kun var et mellomledd mellom forskjellige moduler av MatLab kode.

Problemet oppstod da roboten skulle integreres i CAS. Det ble ikke vektlagt godt nok hvilken format data skulle ha inn og ut mellom modulene: CAS, RobotHandler, og NXTInterface.

All data fra roboten til CAS må passe sammen. Når roboten ble forsøkt integrert i CAS med å rette opp feil i alle modulene, forplantet feilen seg og feilsøkingen ble vanskelig. Noe som fungerte på en del av systemet fungerte ikke på andre deler.

Grunnen til at det ble problemer var måten IR-robotens datastrøm foregikk på.

Det fungerte først når data inn til RobotHandler ble definert og «satt i sten», og fokuset ble satt til å få RobotHandler til å samarbeide med CAS.

Slik det er nå vil en ny robot fungere i CAS dersom den gir riktige data til basisfunksjonene i RobotHandler. For å gi et annet eksempel, i tillegg til det som er beskrevet over med at IR-roboten trenger $\frac{\theta}{2}$, er det nå slik at CAS

mottar lokal data fra fullscan, slik CAS ønsker. IR-roboten gir ut global data, som krever en transformering med rotasjonsmatriser før det kan benyttes i CAS. Nå, når RobotHandler returnerer lokal data fra fullscan, må roboten gi ut lokal data. Da vil grensesnittet til IR-roboten måtte rotere global data fra roboten før den leverer data videre, slik det burde være.

RobotHandler blir regnet som et positivt bidrag til systemet. Spesielt i forhold til flere roboter, slik det ble utviklet i forhold til. Men om kun NXT-roboten ble benyttet er det allikevel god grunn til å benytte RobotHandler.

Det blir et naturlig skille mellom CAS og robotene. Roboten kan feilsøkes individuelt og enkelt. Når inn- og ut-data er definert ligger problemet i å få roboten til å fungere slik den skal, og ikke i å finne ut av hvor data skal inn og ut, og i hvilket format.

NXTInterface

Grensesnittet mot roboten ble en naturlig måte å forholde seg til NXT-roboten på. Det er bra å få vekk kode som ikke har med kartlegging å gjøre fra CAS.

Det virker også naturlig med et grensesnitt mellom IR-robot og CAS, men av ukjente grunner ble det ikke slik. Selv om det kan tenkes at det er et resultat av stadig påbygging fra flere studenter som har jobbet individuelt med koden.

For IR-roboten er måten kommunikasjonen foregår på én og én byte, mens NXT-roboten mottar serier av bytes. Det gjør at IR-roboten trenger flere kodelinjer for å håndtere kommunikasjonen[1].

Slik NXTSLAM er satt opp på sendes det en serie av tall i en melding. Denne serien av tall er vanskelig for mennesker å forholde seg til uten å sjekke opp i tabeller, og derfor var det naturlig å legge dette inn i en funksjon som behandlet kommunikasjonen til NXT-roboten.

[1] NXT-roboten benytter funksjoner fra RWTHMindstormsNXT-toolbox for å sende over beskjeder, og totalt antall kodelinjer er nok strengt tatt flere for NXT-roboten enn for IR-roboten. Her menes antall ≪synlige≫ kodelinjer i CAS.

NXTInterface har en melding som argument i tillegg til variable argumenter i forhold til funksjonen som kalles, som er mye enklere å forholde seg til. For eksempel `[out error] = NXTInterface('ping')` istedet for `'4'`.

Selv om dette gjør det lettere å kalle funksjoner manuelt er det en annen funksjon NXTInterface har som gjør det nødvendig med individuelle grensesnitt mellom RobotHandler og roboter generelt. Det er at all kode relatert til NXT-roboten er samlet og ikke trenger å legges direkte inn i RobotHandler. Hvis mer enn én robot hadde blitt lagt direkte inn i RobotHandler, hadde det blitt en veldig stor fil som hadde vært lite motiverende å jobbe med.

7.2.5 Feilhåndtering

Når IR-roboten var implementert direkte i CAS var feilhåndteringen lagt til i tilhørende program-kode/-blokk. Nå er det ikke tilstrekkelig med feilhåndtering i forhold til om det oppstår problemer med roboten.

Det er laget en funksjon for feilsjekk i basisfunksjoner. Dette er ikke nødvendigvis den beste løsningen, og må vurderes i videre arbeid.

Men poenget med denne funksjonen er at robotens grensesnitt setter en variabel om det skjer noe feil, som kollisjon eller annet. Da vil denne feilsjekken hente denne variabelen og da enten være ≪true≫, eller ≪false≫.

CAS kan bruke denne sjekken før eller etter en iterasjon for å se om det er trygt å fortsette.

En annen mulighet er å legge inn alvorlige feil i feilhåndteringen til RH. Det er to output argumenter til kallene i RH og NXTInterface, `out` og `error`. Hvis det oppstår en alvorlig feil, som kollisjon, kan det være en mekanisme som gir output: `error`, en feilmelding for alle kall.

7.2.6 Testing av robot

Testene ble utført for å sjekke at CAS fungerte som det skulle med den nye roboten. Simuleringer viser at CAS fungerer som før, og testene viser at kartleggingen fungerer med hensyn på inn-data.

De første testene viste at roboten fungerte etter prinsippene. Men kartleggingen var dårlig og det ble laget funksjonalitet for å plotte global data. Dermed kunne global data og kartgenereringen sammenliknes. Denne funksjonaliteten ligger i GUI tilknyttet RobotHandler.

Etter kalibrering av sensorene klarte roboten å kartlegge tilstrekkelig nøyaktig til at navigering var bra, og kartleggingen ble fullført. Kvaliteten til kartet var ikke så bra som forventet, men dette er på grunn av inn-data.

Grunnen til at det ble forventet bedre resultater var at sensorene er dyre og har vist seg nøyaktige i prosjektoppgaven Homestad12[10].

Når inn-data blir undersøkt kan det sees at rette overflater ikke blir gjengitt riktig. Dette er grunnet loven om refleksjon. Bølger som sendes inn mot overflater vil ha samme utgangsvinkel. Når man benytter infrarød sensor vil dette gjelde fordi IR-sensoren har egen lyskilde. Videre er det slik med loven om refleksjon at vinkelen ut er den samme som den som er inn, dersom ruheten til materialet er mindre enn bølgelengden til signalet/bølgene. Dette er grunnen til at ultralyd sensor ikke egner seg til disse prosjektene.

Det er ikke så mye å gjøre med dette problemet. Men CAS løser en del av problemet med å ta flere målinger av samme område. Ellers kan det brukes materialer med større ruhet for å øke nøyaktigheten til målingene.

Det er ikke plottet rådata fra IR-roboten i tidligere rapporter, og kan derfor ikke sammenliknes med NXT-roboten. Kartgenereringen med IR-roboten ser ut til å være bedre enn resultatene fra NXT-roboten, basert på testresultatene i rapporten Tusvik09[16].

Annet enn at IR-roboten bruker en annen type IR-sensor, skal robotene fungere på samme måte. IR-robotens sensorer har lengre rekkevidde, og er muligens

bedre, selv om de er vesentlig billigere. Det hadde derfor vært interessant å sammenlikne en plotting av global data fra IR-roboten mot NXT-roboten.

Hvis det skulle vise seg at det er sensorene som er grunnen til at IR-roboten har bedre kartgenerering kan det videreutvikles et sensortårn fra prosjektoppgaven til Bakken08[6]. I denne prosjektoppgaven blir det laget et kretskort, som håndterer samme IR-sensorer som brukes på IR-roboten, som fungerer sammen med NXT-brikken.

Dette kan uansett anbefales da disse sensorene har vist seg godt egnet, er billigere, og det har blitt gjort arbeid med kalibrering av sensorene og montering på NXT-brikker.

Andre grunner til at kartgenereringen er bedre i tidligere rapporter kan være at parametre i CAS er forandret for å tilpasse kartleggingen. Et kart med mange beacons som ikke samsvarer med virkeligheten kan forbedres med å øke kravene for å sette beacons. Det kan også være at den lille forskjellen som skjer med forvridning i tårnet til NXT-roboten er nok til at datainnsamlingen blir vesentlig dårligere. Hvis i tillegg posisjonsestimatet er dårligere enn for IR-roboten kan dette være nok til å forklare forskjellen i kartgenerering.

7.3 Feil og mangler, og tidligere versjoner

Dette kapittelet er laget for fremtidig arbeid, og kan hoppes over ellers.

Programvaren som blir gitt på CD er testet og fungerende ved levering. Men det er feil og mangler som burde nevnes for videre arbeid.

7.3.1 Mangler

Selv om prosjektet fungerer slik det skal er det mangler som ikke ble implementert av tidsmessige grunner eller forandret prioritering.

Manuell kjøring

Manuell kjøring fra GUI ser bra ut og er morsomt, men er ikke nødvendig for prosjektets kartlegging. Da dette ikke var et trivielt problem på NXT-brikken ble det implementert en alternativ løsning. Istedet for at roboten starter og stoppes, kjøres den en gitt avstand fremover for hver gang den mottar kommandoer. Når den mottar kommandoen «kjør fremover» vil den kjøre 5cm fremover og tilsvarende med rotasjon og bakoverkjøring.

Grunnen til dette er fordi det må benyttes eldre motorstyringsfunksjoner på NXT-brikken for at motorene skal kjøre uten tacho-begrensning. Dette er mulig å få til om det skulle være ønsket og det er nok informasjon i programvaren til RWTH og NXTMotor til å se hva som må gjøres for å få det til. Hvis roboten kjøres fra PC, med RWTH, er det bare å sette tachocount til 0 for å kjøre uten begrensning.

Kollisjonsvarsel

Kollisjonsvarsel er en grunnleggende egenskap til roboten. Det ble laget en tråd, «task», på NXT-brikken for å håndtere dette. Når et kollisjonsvarsel skjer settes det en variabel som kan hentes, kalt «isError».

Det er nødvendig å oppdatere firmware på NXT-brikken for å kunne stoppe tasks enkeltvis. Slik at om det oppstår kollisjonsvarsel så kan variabel hentes, og roboten stopper å kjøre fremover uten å stoppe annen funksjonalitet. Slik det er nå så må roboten stoppe alle tasks, eller bare kjøre videre inn i veggen. Hvis roboten stopper alle tasks, vil det ikke være mulig å hente status på roboten, som sier at roboten har et kollisjonsvarsel.

I tillegg så er det samme problem med kalibrering her som i all deler av prosjektet med hensyn på EOPD sensorene. I NXT-brikken brukes bare rådata, slik at en måling på aktuelt materiale må gjennomføres for å finne en passende verdi å bestemme kollisjonsvarsel på, istedet for å bruke en fast avstand i cm som hadde vært det enkleste. Det burde være en egen funksjon for å håndtere dette slik at det ikke var nødvendig å åpne kildekoden for hver gang.

Kollisjonsvarsel og funksjonalitet for dette er ikke testet. Det ble rett og slett glemt da integreringen i CAS var mer krevende enn forventet. Men programkode og funksjonalitet er altså implementert og klar for testing/bruk.

Feilhåndtering i CAS

På slutten av prosjektet ble det klart at det manglet feilhåndtering i CAS etter implementeringen av RH. Løsningen på dette ble å sette «isError» som en av basisfunksjonene.

Dette er ikke implementert i CAS. Det bør sjekkes for hver iterasjon, eller oftere og vil være et naturlig ledd i videre arbeid med CAS. Mer om dette i diskusjon, delkapittel 7.2.

7.3.2 Feil

Kjente feil tilknyttet prosjektet.

Tårn

Som nevnt er tårnet ikke nøyaktig i setting av posisjon. Dette bør fikses på for å gjøre roboten mer autonom og gi bedre input data.

Nedprioritering

Det ble nedprioritert eldre funksjoner i NXT-roboten når det ble jobbet med integreringen i CAS, og basisfunksjonene var satt. Det finnes mange, flere enn nødvendig, funksjoner, og de skal være testet og kjørbare. Men det kan ikke garanteres at alle fungerer uten feil fordi det er mange versjoner siden dem ble brukt.

Hvis det oppstår feil i disse funksjonene er det mest sannsynlig enkle grep som trengs for å få dem til å kjøre igjen.

7.3.3 Versjoner

Det legges ved tidligere versjoner til Hannaas11 og Tusvik09 på CD. Ingen av disse versjonene fungerer ved oppstart. Hovedfilen «lineBeaconSLAM» har stor bokstav, og en liten del av koden må forandres, eller fjernes, for at simuleringen skal kjøre.

Versjonene er ikke modifisert, da dette er versjoner som videreføres direkte.

Programvaren i dette prosjektet er basert på Tusviks versjon, da Hannaas versjon ikke nødvendigvis fungerte som det skulle.

Tillegg A

Brukermanual

A.1 Utstyrsliste

Utstyret som ble brukt i dette prosjektet blir gitt under.

Roboten Roboten som ble benyttet med komponenter er beskrevet i kapittel 1.2, og består kort av:

- NXT-brikke versjon 2.0
- 2 stk,- EOPD sensorer
- 3 stk,- Lego Mindstorms motorer
- divers lego og Lego Mindstorms kabler

PC Programvare som kjørte på PC:

- Windows 7
- MatLab 2012-2013
- BrixCC[1]

[1] Kan lastes ned gratis fra `http://bricxcc.sourceforge.net/`

- Lego Mindstorm driver(Phantom), vedlagt på CD.
- libusb-win32, vedlagt på CD

NXT-brikke Utsyr forbundet med NXT-brikken:

- bluetooth dongel[2]
- USB-kabel
- Programvare utviklet i dette prosjektet lastet over med BrixCC.
- Firmware: Offisiell firmware fra Lego, ver. 1.31

A.2 NXT-roboten

På NXT-roboten er det NXT-brikken og sensorene som trenger oppsett. Programvaren må overføres og er dokumentert i rapporten.

Fra dette prosjektet til det neste kan det ha vært forandringer, så det kan være lurt å sjekke på nettsidene til utviklerne av RWTHMindstormNXT, `www.mindstorms.rwth-aachen.de` først. Fremgangsmåten som ble brukt i dette prosjektet blir gjennomgått her.

A.2.1 Hardware

NXT

NXT-brikken trenger batteri, og ellers er det USB eller bluetooth tilkobling mellom PC og NXT-brikke. For å koble til NXT-brikken med bluetooth er det bare å legge til i bluetoothenheter i windows, og skrive inn passord. Passordet kan velges selv, men standard er ≪1234≫.

[2]Det ble brukt en bluetooth dongel fra firmaet ≪best connectivity≫, og denne fungerte utmerket.

EOPD - avstandssensorene

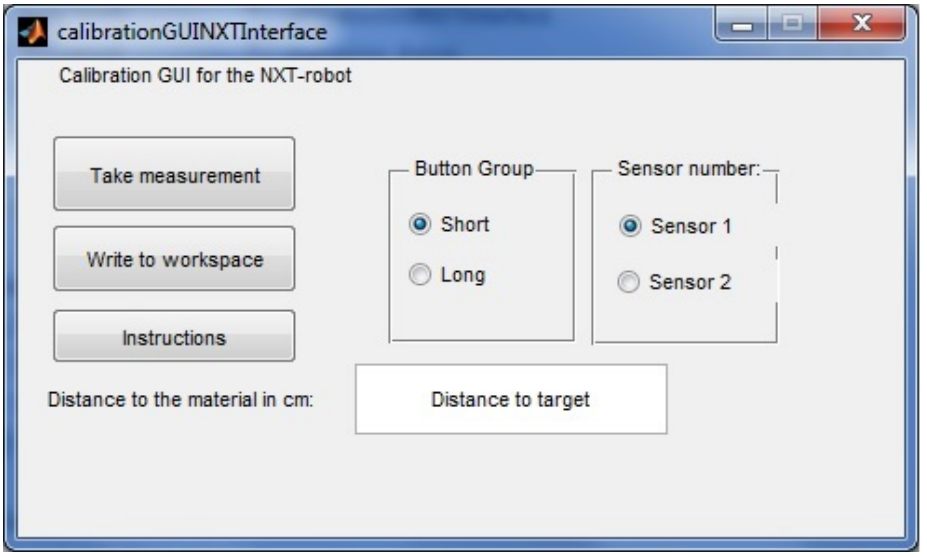
EOPD'ene trenger kalibrering før de kan brukes skikkelig. Det er et forslag til kalibreringsmatrise i programkoden for at systemet skal kunne brukes uten kalibrering. Men hvis det skal kartlegges må altså sensorene kalibreres.

Kalibreringen består av å velge sensor og avstand. Avstand er enten ≪near≫ eller ≪far≫, der den faktiske avstanden blir satt. For eksempel kan man måle 10cm med ≪near≫ og 20cm med ≪far≫.

Kalibreringen må skje på samme materiale som roboten skal måle på. Dette er fordi sensorene gir forskjellige verdier etter hvilken glans materialet har.

Det ble laget et GUI for å kalibrere sensorene ifra CAS. Trykk knappen ≪RobotHandler≫ nede til høyre i CAS og deretter knappen ≪Calibration≫, og figur A.1 vil dukke opp.

Her velges sensorene som skal kalibreres, målt avstand og avstanden som det kalibreres på. Valget ≪Write to workspace≫ sender kalibreringsmatrisen til MatLabs workspace og viser matrisen for at bruker kan sjekke om målingene er riktige.



Figur A.1: GUI som brukes for å kalibrere sensorene til NXT-roboten. Kommandoer fra oppgavelinjen kan også benyttes.

A.2.2 Software

PC'en kjører Windows 7 og det kan være nødvendig å installere libusb-win32 for at kommunikasjon mellom PC og NXT-skal fungere. Det kan være det ikke er nødvendig med bluetooth og NXT versjon 2.0, så det anbefales å prøve med kun Lego Mindstorms drivere først. libusb-win32 og Lego Mindstorms drivere er vedlagt på CD.

Bluetooth er enkelt om det fungerer. Det har vært problemer med tidligere NXT-brikke, men med NXT 2.0 var det «rett frem». Installer bluetooth dongel og legg til NXT-brikken som en bluetooth enhet. Aksepter paringen med valgt passord og når NXT-brikken er i listen over bluetooth-enheter så er den delen gjort.

Noen av problemene med bluetooth tilkoblingen kan være versjonen av RWTH-

MindstormsNXT [4], og derfor anbefales det å benytte den siste versjonen.

BrixCC

BrixCC er en IDE for NXC. I BrixCC kobles NXT-brikken til og programvare overføres. Hvis NXT-brikken er tilkoblet MatLab, CAS, så fungerer det ikke å legge til NXT-brikken i BrixCC, og motsatt.

BrixCC brukes også til å skrive og å kompilere NXC kode. Programkoden kan overføres med USB-kabel eller bluetooth tilkobling.

A.3 IR-roboten

For oppsett av IR-roboten henvises det til Tusvik09[16].

A.4 MatLab, CAS toolbox

For å bruke CAS er det letteste å kjøre filen inittoolbox.m. Denne filen kjører initialisering og åpner GUIDE vinduet til CAS. GUIDE er et hjelpeprogram i MatLab for å lage GUI programmer. I GUIDE er det bare å trykke på «kjør», eller hurtigtastene «ctrl+T».

Det meste i GUI'et er selvforklarende. For å kjøre systemet må tilkobling velges, og deretter er det bare å starte systemet. Etter at RobotHandler ble integrert er tilkoblingsmulighetene: simulering, «simulator»; og aktuelle roboter, «RobotHandler».

Hvis simuleringen velges så må kartet som simuleres velges i høyre øvre hjørnet av GUI'et. Det er ikke alle kartene som fungerer uten å forandre på parametere.

A.4.1 RobotHandler GUI

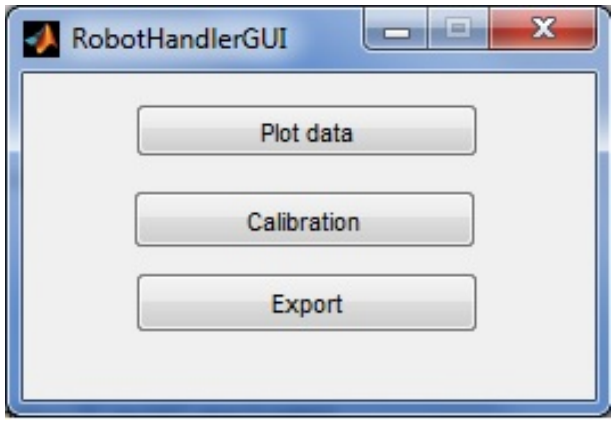
Det ble lagt til en endring i CAS GUI'et, knappen ≪RobotHandler≫. Når denne knappen trykkes kommer det opp en ny meny. Se figur A.2.

Det er tre valg, som kjører kode relatert til den roboten som er tilkoblet til MatLab.

Plot data – en funksjon som ble lagt til for å kunne se på data fra roboten i forhold til valgt iterasjon. Global data, lokal data og posisjon blir plottet fra første til siste iterasjon med pilene under plottene. Dette er nyttig for å sjekke om roboten gir den måledata som er nødvendig for CAS.

Calibration – kalibrering er nødvendig for både IR-roboten og NXT-roboten. Egen GUI eller kode relatert til robot er nødvendig.

Export – legger måledata brukt i ≪Plot data≫ i workspace til MatLab(workspace 'base').



Figur A.2: RobotHandler GUI.

Plott av data

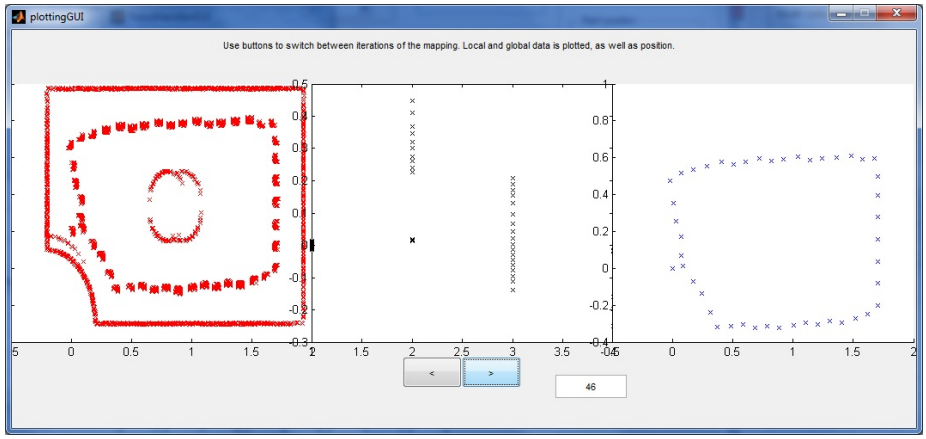
Plotting av data brukes for å sjekke data fra robot. Dette er enkelt og greit bare å øke eller minske tallet på iterasjoner i GUI'et. Se figur A.3. Figuren viser data fra en simulering, som viser hvordan global data skal forekomme, og danner et bilde av omgivelsene.

Kalibrering

Kalibreringen er individuell i forhold til roboten. Det er hittil kun laget for NXT-roboten i GUI'et tilknyttet RH. Kalibrer med to avstander, og følg instruksjonen gitt i GUI, eller som beskrevet over.

Eksportering av data

Brukes for å ha dataplott tilgjengelig i workspace.



Figur A.3: RobotHandler plotter. Global data til venstre, lokal data i midten og posisjon til høyre.

Tillegg B

Tabeller

Tabellene skal forhåpentligvis bidra til å dekke det som er av spørsmål i forhold til datastrøm og definisjoner.

RH og NXTInterface blir også dokumentert.

Nr.	Funksjonsnavn, RH	Beskrivelse
1	`RobotHandler('ping')`	Ping er strengt tatt ikke en nødvendig funksjon. Men er såpass vanlig og nyttig at den blir en del av basisfunksjonene.
2	`RobotHandler('getpos')`	Henting av posisjon er nødvendig for at CAS skal vide estimatet av robotposisjonen.
3	`RobotHandler('setpos')`	Setting av posisjon er en nødvendig for å sette posisjonen. Brukes også til resetting av posisjon.
4	`RobotHandler('fullscan')`	Fullscan er ikke nødvendigvis gitt på den tradisjonelle måten gitt av NXT-roboten og IR-roboten. Fullscan består av punkter lokalt i forhold til roboten, og kan hentes på hvilken som helst måte. Punktene må være lagt etter hverandre nummerert med sensornummer.
5	`RobotHandler('getrobotparams')`	Parametre tilknyttet til roboten. Må være en basisfunksjon fordi det varierer i forhold til roboten som brukes.
6	`RobotHandler('connect')`	–
7	`RobotHandler('disconnect')`	–
8	`RobotHandler('iserror')`	For å sjekke om det er noe galt som har skjedd med roboten. For eksempel timeout eller kollisjon. Dette er en basisfunksjon for å forenkle feilhåndteringer.
9	`RobotHandler('drivetopoint')`	Det er to argumenter. Først rotasjon og deretter fremoverkjøring.

Tabell B.1: Basisfunksjoner mellom CAS og RH. Basisfunksjonene er funksjoner som roboten må takle for å kunne kjøre i CAS.

Nr.	Fil	Funksjon	Endring
1	getRobotPose.m	getRobotPose	Filen er omgjort til å håndtere RH. Noe av tidligere kode er lagret i IR Interface.
2	fullScan.m	fullScan	Kode er byttet ut med RH. Tidligere kode er lagt til i IRInterface.
3	lineBeaconSLAM.m	–	Variabler tilknyttet roboten, som hjulradius, er lagt til i RH. Oppdateringen av verdiene er tatt ut av while-løkken, da det kun settes én gang.
4	lineBeaconSLAM.m	–	Avhengighet av å vite antall sensorer er fjernet. CAS mottar nå lokal måledata fra fullscan.
5	connectTo.m	connectTo	Gammel kode er byttet ut med RH.
6	LegoGui.m	disconnectButton-_Callback	Beholdt kode for håndtering med variabler, og byttet ut frakoblingen med RH.
7	LegoGui.m	pingButton_Callback	Byttet ut kode med RH.
8	setRobotPose.m	setRobotPose()	La til RH.
9	clearPose.m	clearPose()	La til RH.
10	setRobotTarget.m	setRobotTarget()	Koden lagt i IR-interface, og gjort om til å behandle RH.

Tabell B.2: Endringene som er gjort i CAS i forhold til integreringen av RobotHandler.

PROTO	Funksjon	Beskrivelse	Kall	Svar
MOVE	moveRobotForward	Kjører roboten en gitt avstand, *ddd*,med en gitt hastighet *pp*.	`00dddpp`	–
MOVE	turnRobot	Snur roboten til gitte grader *ggg*, med gitt hastighet *pp*.	`01gggpp`	–
MOVE	turnTower	Snur tårnet til roboten til gitte grader *ggg*, med gitt hastighet *pp*.	`02gggpp`	–
MOVE	manualDrive	Kjører roboten i gitt retning. Retningene er rett frem(w), bakover(x), snu høyre/venstre(d/a) og stopp(s).	`03(w/a/d/s/x)`	–
MES	fullScan	Måler avstanden rundt roboten i 360°.	`10`	Vektorer med måledata og tilhørende sensor.
MES	singleMesSens1	Gir en enkeltmåling med sensor nr. 1	`11`	Måledata
MES	singleMesSens2	Gir en enkeltmåling med sensor nr. 2	`12`	Måledata
MES	mesOnScreen	Begge sensorene måler, og gir ut verdien på skjermen til NXT-brikken.	`13`	–
DEBUGG	–	– fjernet fra programkode –	`2xxx`	–
COM	–	– fjernet fra programkode –	`3xxx`	–
PING	pingRobot	Et ping fra MatLab	`4`	≪true/1≫
ACCESS	resetNXT	Resetter interne verdier på NXT-brikken til $(x, y, \theta) = (0, 0, 0)$, der θ er orienteringen til roboten	`500`	–
ACCESS	resetTower	Setter intern verdien på tårnet til 0°.	`501`	–
ACCESS	resetNXTall	Setter $(x, y, \theta) = (0, 0, 0)$ og tårnet til 0°, der θ er orienteringen til roboten	`502`	–
ACCESS	getPos	Gir posisjon og orientering på roboten.	`510`	Verdiene som en streng.
ACCESS	getTowerAangle	Gir intern orientering på tårnet.	`511`	≪*verdi*≫
ACCESS	setPos	Setter den interne posisjonen til roboten. x og y kan være et ubestemt antall siffre.	`520xxyy`	–
ACCESS	setTowerAngle	Setter intern orientering på tårnet.	`521`	–
ACCESS	isBusy	Sjekker om roboten er opptatt med å kjøre.	`53`	0/1

Tabell B.3: Oversikt over funksjonene til NXT-brikken.

Nr.	Funksjonsargument	Funksjon	Tillegsargumenter
1	`'disconnect'`	Kobler fra	–
2	`'connect'`	Kobler til og setter relevante variabler.	–
3	`'driveForward'`	Kjører roboten fremover.	distance, *power*
4	`'rotateRobot'`	Roterer roboten mot gitte grader.	angle, *power*
5	`'rotateTower'`	Roter måletornet til ønsket vinkel.	angle
6	`'purgeMailbox'`	Tømme «mailbox» på NXT-brikken. Hvis gamle meldinger ligger inne, vil dette naturligvis føre til problemer.	–
7	`'singleMeasurement'`	Enkeltmåling på sensor 1 eller sensor 2.	`'sensor1'`,`'sensor2'`
8	`'fullScan'`	Returnerer måledata fra sensor 1 og sensor 2 som en $n \times 2$ matrise	–
9	`'mesOnScreen'`	Gir måledata fra sensorene på NXT-brikkens skjerm.	–
10	`'manualDrive'`	Kjører roboten et lite stykke i ønsket retning. Kan brukes for å kjøre roboten manuale til en posisjon, eller for å teste roboten. Étt tilleggsargument for å bestemme retningen.	{`'w'`,`'a'`,`'d'`,`'s'`,`'x'`} (h.h.v. fremover, høyre, venstre, stop, bakover.)
11	`'ping'`	Pinger roboten.	–
12	`'setTowerAngle'`	Setter måletårnet til ønsket orientering.	angle
13	`'getPos'`	Returnerer posisjon og orientering på roboten i kartesiske koordinater.	–
14	`'getTowerAngle'`	Returnerer vinkel på måletårnet.	–
15	`'setPos'`	Setter posisjonen til NXT-roboten lokalt. Nødvendig etter databehandling i CAS.	x, y
16	`'setOrientation'`	Setter orienteringen til roboten.	theta
17	`'resetnxt'`	Setter verdier til 0. Étt tilleggsargument som avgjør hva som skal resettes.	{`'nxtall'`, `'nxttower'`, `'nxtpos'`}
18	`'getrobotvariables'`	Returnerer variabler tilknyttet NXT-roboten. Hjulradius, aksel og feilkoffisient.	–
19	`'isbusy'`	Sjekker om roboten er opptatt	–
20	`'drivetopoint'`	Roterer og kjører roboten med gitt argumenter, vinkel og lengde i cm	angle, distance
21	`'setpath'`	Setter MatLabs «path» til å inkludere undermapper.	–

Tabell B.4: Liste med argumenter til NXTInterface. Argumentene i kursiv er valgfrie.

Nr.	Funksjonsnavn	Beskrivelse	inn-/ut-data
1	getPos	Hent posisjonen fra roboten.	radvektor med x, y og θ
2	getOrientation	Hent orienteringen til roboten	θ
3	setPos	Sett posisjonen til roboten lokalt i robotens programvare.	radvektor med x, y og θ
4	setOrientation	Sett orienteringen til roboten lokalt.	θ
5	fullscan	Gjennomfører fullscan på roboten og returnerer standardisert måledata i meter.	Lokal data. Tre kolonner med sensor nummer, x- og y-data
6	driveforward	Kjør fremover, gitt avstand.	avstand i cm
7	rotate	Roter roboten til gitt vinkel.	Vinkel i grader. Gradene gis som halvparten av ønsket rotasjon, $\frac{\theta}{2}$.
8	driveToPoint	Kjør roboten til gitt posisjon. Altså rotere, og deretter kjør rett frem.	Avstand i cm, vinkel gitt i grader og halvparten av ønsket vinkel, $\frac{\theta}{2}$.
9	getRobotType	Hent en melding som kan vise bruker hvilken robot som er tilkoblet.	En melding til bruker.
10	getRobotVariables	Returnerer variabler tilknyttet roboten. Dette er radius på venstre og høyre hjul. Lengde på aksel og feilkoffisient.	`out = [LH RH ... WB EGC]`
11	isConnectedToRobot	Forteller CAS om det er en robot tilkoblet.	true/false
12	connect	Tilkobling av robot. En liste over mulige roboter kjøres igjennom til tilkobling er velykket.	*melding/feilmelding*
13	disconnect	Frakobling av robot fra RobotHandler.	*melding/feilmelding*
14	isError	Forespørsel fra CAS om det har skjedd noe feil med roboten. Kan brukes mellom hver iterasjon og vil gi beskjed om kontakten mellom RobotHandler og robot grensesnitt er brutt, eller om det er et hinder i banen til roboten.	false/*feilmelding*
15	ping	Pinger roboten	svar i antall sekunder

Tabell B.5: Funksjonene som støttes av RH. Det er ikke nødvendigvis alle robotene som kan utføre fullt sett av funksjonalitet.

Retning	Funksjon	Data	Format	Enheter
RH→CAS	*fullScan*	Måleverdier fra én eller flere sensorer. Representerer en 360°måling.	Data blir representert som en $[n \times 3]$ matrise der første kolonne angir sensor, og andre og tredje kolonne angir hendholdsvis x- og y-koordinater. Alle verdiene er lokalt, relativ til roboten. Kolonnen som angir sensor kan være en stigende tallrekke og trenger ikke å være faktisk sensornummer. Sensornummer er for å sortere målevektor slik at målingene er ved siden av hverandre.	$[m]$
RH→CAS	*getPos*	Posisjonen til roboten i x- og y-verdi, samt orienteringen. Posisjonen blir gitt i [mm], men er lagret som [m].	En radvektor med verdiene, $[x,\ y,\ \theta]$.	[mm], rad
RH→CAS	*getRobotParams*	Bestemte verdier som varierer i forhold til hvilken robot som brukes. Verdiene er: radius for venstre og høyre hjul, LW, RW; avstanden mellom hjulene,WB; og EGC(Error Growth Cofficient), feilen roboten kjører per meter.	Radvektor med verdiene, [LW, RW, WB, EGC]	$[m]$, $\frac{1}{[m]}$
CAS → RH	*setPos*	Posisjonen til roboten i x- og y-verdi, samt orienteringen. Posisjone blir gitt i [mm], men brukes lokalt i RH, samt videre til NXTI, som [m].	En kolonnevektor med verdiene, $[x,\ y,\ \theta]^T$.	[mm], rad
CAS→RH	*driveToPoint*	Vinkel roboten skal rotere og hvor langt frem den skal kjøre.	Gitt som argumenter til funksjonen. Vinkel blir gitt som halvparten av den vinkelen roboten skal rotere. Vinkelen må derfor ganges med to før den sendes til NXTInterface.	$\frac{\text{Grader}}{2}$, [cm]

Tabell B.6: Standard verdier mellom CAS og RobotHandler.

Kommandoer fra MatLab:	Kommandoer fra IR-robot:
Robot kjører fremover	Ping svar til MatLab
Robot stopper	Ping MatLab
Robot svinger venstre	Beskjed: Robot er fremme
Robot svinger høyre	Beskjed: Header for fullskann
Sett intern posisjon til roboten	Beskjed: Fullscan ferdig
Få intern posisjon til roboten	Beskjed: Header for enkeltmåling sensor 1
Reset intern posisjon til 000000	Beskjed: Header for enkeltmåling sensor 2
Sett tårn til vinkel t	Beskjed: Header for enkeltmåling sensor 3
Returner vinkel til tårn	Beskjed: Header for enkeltmåling sensor 4
Sett orientering til robot.	Beskjed: Ingen objekter i området til sensor 1
Forflytt roboten med xx cm.	Beskjed: Ingen objekter i området til sensor 2
Flytt roboten til gitt posisjon	Beskjed: Ingen objekter i området til sensor 3
Returner statusen til roboten.	Beskjed: Ingen objekter i området til sensor 4
Ping roboten	Beskjed: Måling fra én sensor med en avstand ferdig.
Ping svar til robot	Beskjed: Sender posisjon
Full IR-skan	Beskjed: Sender tårnvinkel.
Måling sensor 1	Beskjed: Header for status
Måling sensor 2	Beskjed: Opptatt, kan ikke motta beskjed
Måling sensor 3	Beskjed: NACK
Måling sensor 4	Beskjed: ACK
Kalibrer sensorer	Beskjed: Kolisjonsvarsel < 15cm
Meneskemodus	Beskjed Kolisjonsfeil < 10cm
Maskinmodus	

Tabell B.7: Komandoer som kan sendes til og fra IR-roboten.

Nr.	RWTH funksjon	NXTI funksjon	Beskrivelse
1	COM_OpenNXT	`NXTInterface('connect')`	Åpner tilkobling til NXT-roboten og returnerer en handle. En .ini fil bør benyttes. Her brukes «bluetooth.ini» filen som inneholder data om tilkoblingen.
2	COM_CloseNXT	`NXTInterface('disconnect')`	Lukker forbindelsen som er tidligere opprettet og sletter handle.
3	COM_SetDefaultNXT	–	Brukes for å gjøre handle til roboten tilgjengelig globalt slik at den kan brukes i andre funksjoner uten å bli gitt som argument.
4	CalibrateEOPD	`NXTInterface('calibrate')`	Brukes for å kalibrere EOPD sensorene, sette tidligere kalibreringsmatrise og for å hente ut ny kalibreringsmatrise.

Tabell B.8: Oversikt over funksjonene som brukes fra RWTHMindstormsNXT-toolbox i NXTInterface.

Tillegg C

Vedlagt CD

Vedlagt på CD:

Drivere Drivere for kommunikasjon mellom NXT og PC:

- Lego Mindstorm Fantom driver
- libusb-win32

Rapporter Tidligere rapporter nevnt i ≪tidligere arbeid≫ og denne rapporten.

MatLab-programvare Koden som ble utvikler i dette prosjektet:

- CAS med integrert RobotHandler
- RobotHandler
- NXTInterface
- IRInterface

NXC programvare Koden som ble utviklet til NXT-brikken.

Tidligere programvare MatLab kode, CAS, og IR-robotens programvare.

Bilder og video En video av kjøring og bilder av roboten med testbanene.

Tidligere arbeid Det er tatt med noe av innholdet fra Tusvik09 og Hannaas11. Dette inkluderer C-kode til IR-robot og CAS.

Figurer Figurene som er lagt til i rapporten.

Tillegg D

Programvare eksempler

D.1 Programvare

Det sees ikke på som nødvendig å legge ved hele programvaren i rapporten. Alt ligger på vedlagt CD.

Programvaren her skal gi en pekepinn på hvordan programvaren er satt opp, og på forskjeller og endringer.

Eksempler fra tidligere kode i CAS med kommunikasjon mot IR-robot, nye RobotHandler, NXTInterface og noe av koden på NXT-roboten blir gitt.

All kode er på engelsk for å være konsistent med tidligere arbeid.

D.1.1 CAS

Det er mange endringer i koden for å gjøre CAS uavhengig av roboten. Endringer kan sees i tabeller og i koden fra Tusvik09 i forhold til koden i dette prosjektet.

Det blir her gitt et enkelt eksempel på parametersetting, og hvordan forskjellen i CAS er i forhold til fullscan.

Parametersetting

Et eksempel på å gjøre CAS uavhengig av roboten, og heller avhengig av RobotHandler, er måten parametre blir satt på.

```
1  if strcmp(myCon,'simulator')
2      RAD_LEFT_WHEEL=0.02149;
3      RAD_RIGHT_WHEEL=0.02149;
4      WHEEL_BASE=0.1894938;
5      EGC = 0;
6  else
7      [par, er] = RobotHandler('getrobotvariables');
8      RAD_LEFT_WHEEL = par(1);
9      RAD_RIGHT_WHEEL = par(2);
10     WHEEL_BASE = par(3);
11     EGC = par(4);
12 end
```

Fullscan

I tidligere versjoner var all kode knyttet til roboten lagt direkte i CAS. Der RobotHandler er lagt til er det typisk en del kodeblokker som er fjernet. Funksjonen fullscan.m, slik den var før:

```
1      % serial interface to full scan...
2      if myCon.bytesAvailable,
3              disp('fullScan: Bytes in buffer before sending "f" ...
                   to robot. Number of bytes:')
4              disp(myCon.bytesAvailable)
5              disp('bytes read from buffer:')
6              dummyData=fread(myCon)
7      end
8
9      %if( serialAsyncWrite(myCon,'1') ),
10     if( serialAsyncWrite(myCon,'f') ),
```

```
11          set(myHandles.statusText,'String','navigation_develop/fullScan: ...
                Unable to send serial data' );
12      else
13
14          [myError,data1]=serialSyncRead(myCon,1);
15          disp('fullScan: Received response to f: ')
16          disp(char(data1))
17          if(myError)
18              disp('error reading byte after sending f ')
19              return;
20          %elseif data1=='1', % start receiving full scan data
21          elseif data1=='f', % start receiving full scan data
22              i=0;
23              disp('got f, waiting for data')
24              data2=0;
25              while data2≠'g', % while not finished...
26                   %disp('getting sensornummer or g to leave mode:')
27                  bytes_available2=myCon.bytesAvailable;
28                   %if myCon.bytesAvailable,
29                   [myError,data2]=serialSyncRead(myCon,1);
30                   disp('Data motat fra robot i fullscan');
31                   disp(data2);
32                  %end
33                   if (myError)
34                      return
35                  elseif data2=='g'
36                      %outData
37                      %figure(2);
38                      b=outData(:,2); % x data from all of the sensors
39                      c=outData(:,3); % y data from all of the sensors
40                      %plot (b,c,'bx')
41
42
43                      disp('scan ferdig, fikk g')
44                      set(myHandles.statusText,'String','Finished ...
                            fullscan' );
45
46                      disp('antall maalepunkter mottatt:')
47                      disp(length(b))
48
49                  elseif data2=='5'   %Has to be here for no data
50                      disp('no data 1')
51                  elseif data2=='6'
52                      disp('no data 2')
53                  elseif data2=='7'
54                      disp('no data 3')
55                   elseif data2=='8'
56                      disp('no data 4')
57                  elseif (data2=='1' || data2=='2' || data2=='3' ...
```

```
                    || data2=='4'),
58                  i=1+i; % Number of row in outData[]
59                  %disp('got sensornummer (49-52) waiting for ...
                        xxyy')
60                  %disp(data2-48)
61                  %data(1)
62                  outData(i,1)=data2 - 48; % sensor number
63
64                  [myError,data3]=serialSyncRead(myCon,4); % ...
                        data3 has four rows
65                  disp('Data fra sensorer');
66                  disp(data3);
67                  if (myError),
68                      return
69                  else
70                      %converting from signed int,
71                      xpos = data3(1)*256+data3(2);
72                      if( xpos > 32767) % negative value
73                          myX= -(65536-xpos);
74                      else
75                          myX=xpos;
76                      end
77
78                      ypos = data3(3)*256+data3(4);
79                      if(ypos > 32767)% negative value
80                          myY= -(65536-ypos);
81                      else
82                          myY=ypos;
83                      end
84
85                  end
86                  outData(i,2)=myX;
87                  outData(i,3)=myY;
88              else
89                  disp('received f, but the following position ...
                        data is wrong')
90              end
91          end
92      end
93  end
94  if myCon.bytesAvailable,
95      data = fread(myCon);
96      disp('leftover data on serialconnection after fullscan: ')
97      disp(double(data))
98  end
```

Og slik koden er nå:

```
1   % Fullscan returns local data from robot in meters.
2       [outData, myError] = RobotHandler('fullscan');
```

D.1.2 RobotHandler og NXTInteraface

RobotHandler og NXTInterface er to lange filer, funksjoner, som starter med en switch case. Alle casene kjører funksjoner lenger ned i filen. Alternativet hadde vært å ha en fil for hver av casene, som hadde blitt mer rot enn struktur.

RobotHandler

Under er starten på MatLab-filen RobotHandler.m. Koden er modifisert for å passe inn i rapporten med å vise det viktige i filen. Filen på CD ser litt anderledes ut.

```
1  function [ out,error ] = RobotHandler( in, varargin )
2  %ROBOTHANDLER Handles the connection between CAS and Robots.
3  % RobotHandler contains a set of functions and  metohods to
4  % standardize the dataflow between robots and CAS. RobotHandler
5  % comunicate with robots trough individual robot interfaces.
6  % Persistent variables makes this file able to store data, such as
7  % which robot is connected.
8
9  % RobotHandler is used with CAS.
10
11 % Written by T. K. Homestad, at NTNU
12 % Date: may-13.
13
14 persistent robotConnected isError;
15
16 if(isError)
17     out = 'Robot has reportet an error.';
18     error = 'checkForError has replyed with error.';
19 else
20     switch lower(in)
21         case {'ping'}
22             [out, error] = ping();
23         case {'getpos'}
```

```
24              [out, error] = getPos();
25          case {'getorientation'}
26              [out, error] = getOrientation();
27          case {'setpos'}
28              [out, error] = setPos(varargin{:});
29          case {'setorientation'}
30              [out, error] = setOrientation(varargin);
31          case {'fullscan'}
32              [out, error] = fullScan();
33          case {'driveforward'}
34              [out, error] = driveForward(varargin);
35          case {'rotate'}
36              [out, error] = rotateToAngle(varargin{:});
37          case {'drivetopoint'}
38              [out, error] = driveToPoint(varargin{:});
39          case {'getrobottype'}
40              [out,error] = getRobotType();
41          case {'getrobotparams','getrobotpars'}
42              [out,error] = getRobotParams();
43          case {'isconnectedtorobot'}
44              [out,error] = isConnectedToRobot();
45          case {'connect'}
46              [out,error] = connect();
47          case {'disconnect'}
48              [out, error] = disconnect();
49          case {'iserror'}
50              [out, error] = checkForError();
51              isError = out;
52          otherwise
53              error = 'Invalid argument';
54
55      end
56  end
```

NXTInterface

Under er starten på NXTInterface, grensesnittet til NXT-roboten. Funksjoner kan kalles individuelt, men er ment å bli integrert i RobotHandler. Koden er modifisert for å passe inn i rapporten med at det viktige er tatt frem, og uvesentlige kommentarer og linjer er fjernet. Filen som brukes vil se litt anderledes ut.

Odometrien sees på funksjonene *driveForward()* og *rotateRobot()*, der posisjonen

og orienteringen må oppdateres.

Det er mer funksjonalitet i denne filen enn det som vises her. Der iblant behandlingen av rådata fra sensorene.

```
function [ out,error ] = NXTInterface( in, varargin )
%NXTINTERFACE Handles the comunication with the NXT-brick.
%   NXTInterface is made to be used with RobotHandler, and is an
%   interface to a NXT-brick with associated NXC code(NXT SLAM).
%   This interface will not work on the NXT-brick without the NXC
%   code, NXT SLAM, written for NXTInterface. Use stings to call
%   the desired function on the brick. Position and orientation
%   is stored as a rowvector: [x, y, theta] in [m] and in [rad].
%   Drive length of robot is given in [cm]. An error message is
%   usualy given if any errors occur; if no error, an empty string
%   is returned as error message. e.g. to drive the robot forward
%   5 cm, use the command:
%   [out, error] = NXTInterface('driveforward',5);

% Written by T. K. Homestad, spring-13, NTNU

persistent h position initializedNXTInterface isConnected;

if(~ischar(initializedNXTInterface))
    setPath();
    initializedNXTInterface = 'Initialized';
    CalibrateEOPD(SENSOR_1,'SETMATRIX',mat1);
    CalibrateEOPD(SENSOR_2,'SETMATRIX',mat2);
    position.x = 0;
    position.y = 0;
    position.t = 0;
end

if(isempty(h)&&~strcmp(in,'connect')&& ...
    ~strcmp(in,'disconnect')&&~strcmp(in,'setpath'))
    disp('Need to connect to NXT-brick')
    disp('first. Use "NXTInterface("connect")');
elseif(isempty(h)&&strcmp(in,'disconnect'))
    COM_CloseNXT all
    out = 'Disconnecting MatLab from NXT-brick.';
else
    switch lower(in)
        case {'test'}
            position.t
        case {'isbusy'}
            [out,error] = isBusy(h);
        case {'drivetopoint'}
```

```
43              [out,error] = driveToPoint(varargin{:});
44          case {'calibrateeopd'}
45              [out, error] = calibration(h,varargin{:});
46          case {'isconnected'}
47              [out, error] = isRobotConnected(isConnected);
48          case {'setpath'}
49              setPath();
50          case {'disconnect'}
51              [isConnected,error] = disconnect(isConnected);
52          case {'connect'}
53              [h,isConnected,error] = connect(isConnected);
54          case {'getrobotvariables','getvariables'}
55              [out, error] = getRobotVariables();
56          case {'driveforward'}
57              % distance given in [cm]
58              [out, error] = driveForward(h,varargin{:});
59              if(isempty(error))
60                  % convert from [cm] to [m]
61                  position = newPos(position,varargin{1}/100);
62              end
63          case {'rotaterobot'}
64              ang = varargin{1};
65              [out,error] = rotateRobot(h,position.t,ang);
66              if(isempty(error))
67                  position.t = ang*pi/180; % pos in radians.
68              end
69          case {'rotatetower','turntower'}
70              [out,error] = turnTower(varargin{:},h);
71          case {'purgemailbox','purge'}
72              [out,error] = purgeMailbox(h);
73          case {'singlemeasurement','singlemes'}
74              [out,error] = singleMes(h,EOPDMat,varargin{:});
75          case {'fullscan'}
76              [out, error] = fullscan(h,EOPDMat,position);
77          case {'mesonscreen'}
78              [out,error] = mesOnScreen(h);
79          case {'manualdrive'}
80              [out,error] = manualDrive(h,varargin{:});
81          case {'ping','pingrobot'}
82              [out,error] = pingRobot(h);
83          case {'settowerangle','settower'}
84              [out,error] = setTowerAngle(varargin{:},h);
85          case {'getpos','getrobotpos'}
86              out = [position.x, position.y, position.t];
87          case {'gettowerangle','gettower'}
88              [out,error] = getTowerAngle(h,varargin{:});
89          case {'setpos','setrobotpos'}
90              if(length(varargin)==1)
91                  pos = varargin{1};
```

```
92                  position.x = pos(1);
93                  position.y = pos(2);
94                  position.t = pos(3);
95              else
96                  error = 'Wrong arguments';
97              end
98          case {'setorientation'}
99              position.t = varargin{1};
100         case {'resetnxt','nxtreset','nxtresets'}
101             position.x = 0;
102             position.y = 0;
103             position.t = 0;
104         otherwise
105             error = 'Wrong arguments, unknown function';
106             out = 'Unknown function.';
107     end
108 end
109 end
```

D.1.3 NXTSLAM

Koden til NXT-roboten er skrevet i NXC. For å gi et inntrykk av programvaren er det valgt ut noe kode. Main() er for stor til at det kan vises, men strukturen er godt beskrevet i rapporten over.

```
void writeToScreen(string a)
{
    int i = 0;
    int lineNr = 40; // Line nr. is multitude of 8 and counting down.
    while(i<StrLen(a)){
        if(lineNr>56){
        }
        else{
            TextOut(0,lineNr,SubStr(a,i,i+16),false);
            lineNr-=8;
        }
        i+=16;
    }
    if(i<StrLen(a)&&lineNr<=56){
        TextOut(0,lineNr,SubStr(a,i,StrLen(a)),false);
    }
}
```

Figur D.1: Funksjon som kan skrive ut lengre beskjeder til skjermen på NXT-brikken.

```
void moveRobotForward(int distance,int power)
{
    float factor = (360*40/24)/17.59392; // one rotation = 17.593 cm
    // factor times distance gives tacho.

    if(power>100){
        if(power>200)
            power = -35;
        else
            power = 100-power;
    }
    int dist = distance*factor;

    motorParamsSync.power = power;
    motorParamsSync.tacholimit = dist;
    motorParamsSync.speedreg = true;
    motorParamsSync.holdbrake = false;
    motorParamsSync.smoothstart = true;
    SyncPorts = 3; // OUT_AB
    start MoveSync;

    // wait for robot
    while(taskArunning||taskBrunning){
        Wait(30);
    }


    // store new position
    robPos.x = robPos.x - Sin(robPos.theta)*distance;
    robPos.y = robPos.y + Cos(robPos.theta)*distance;
}
```

Figur D.2: Funksjonen som kalles når roboten kjører fremover. Det er ikke en egen semaphore for sync. Motor A og motor B semaphorene settes, og derfor er det trygt å benytte disse.

Tillegg E

Algoritme

Det var et ønske om å forbedre navigasjonsalgoritmen til systemet. Det ble ikke nok tid til at dette ble gjennomført, men et forslag til løsning gis her.

Algoritmen som ble jobbet med er ikke implementert eller testet. Det er heller ikke sikkert den er god nok til å brukes med hensyn på vanskelig implementasjon eller feil.

Men om den virker interessant kan den jobbes videre med.

Algoritmen er som følger:

1. Ta målinger rundt roboten og marker punkter som røde eller grønne. Røde punkter er tomme og grønne punkter indikerer vegg eller objekter.
2. Del opp antall røde punkter etter hverandre i grupper, og sett mulige veier fra sentrum av roboten til disse punktene. Nummerer mulige veier og kjør mot den første.
3. Ta ny måling. All nye punkter som ligger på kjent område sees bort ifra, og tidligere røde punkter i det nye kjente området forsvinner. Nummereringen oppdateres etter nye røde punkter. Posisjonen til roboten blir gitt som

noder fra start til nåværende posisjon. Grupperinger blir satt som mulige veier fra nærmeste node til mulig ny retning.

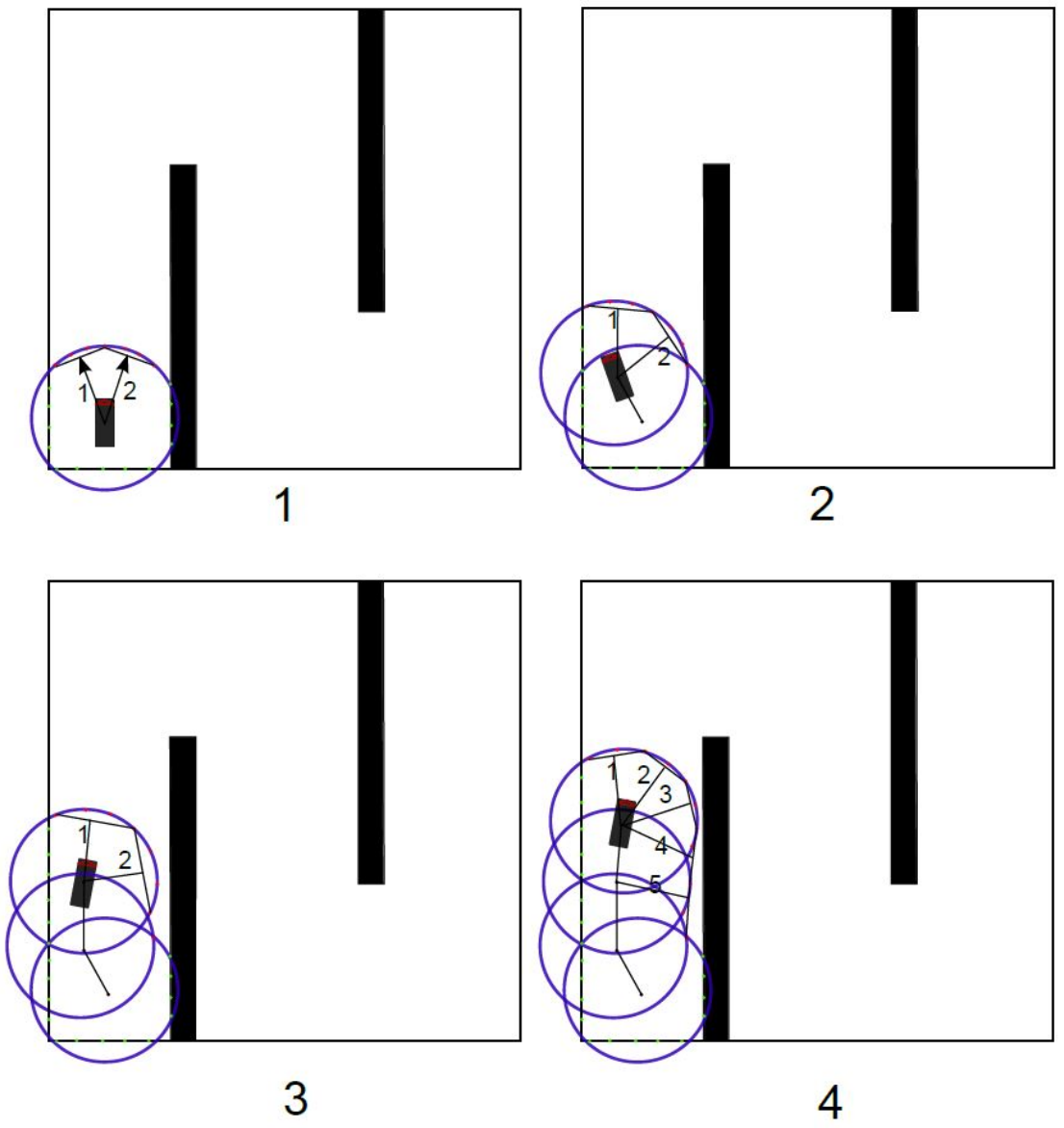
Ny nummerering sorteres slik at den nyeste mulige retningen blir satt som den første.

Dette er et dybde-først-søk. Grunnen til det er at roboten ikke bør kjøre tilbake mer enn nødvendig.

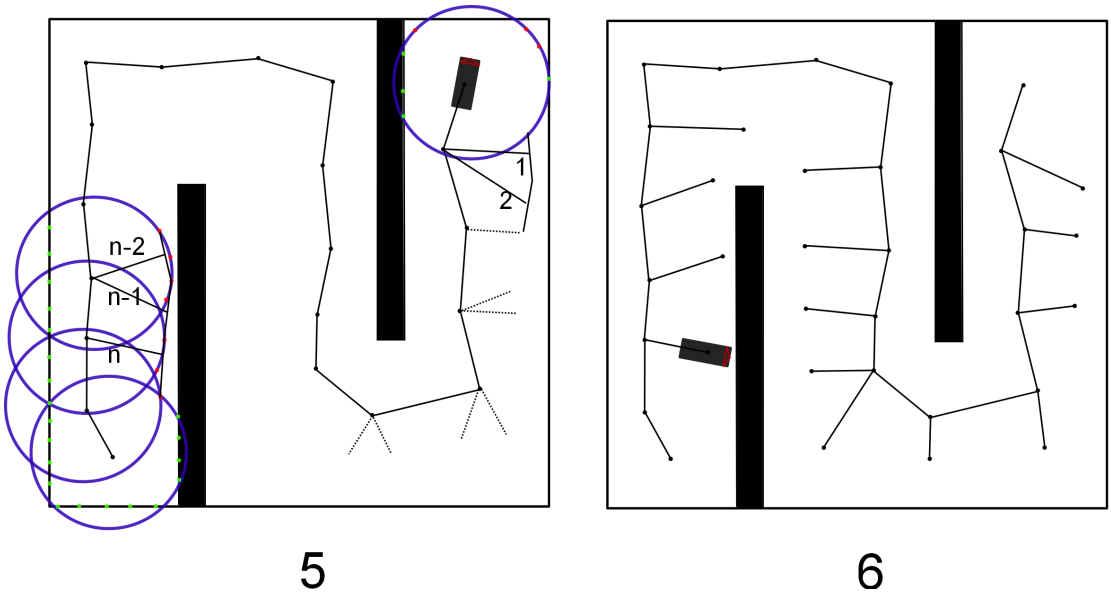
Figurene E.1 og E.2 viser algoritmen. Det kan bli vanskelig å tyde på papir, og det kan være lettere å se disse figurene fra CD.

Denne algoritmen vil ikke navigere etter kartet slik den nåværende algoritmen gjør. Istedet vil denne algoritmen gi data til CAS som CAS bruker til kartgenerering. I implementert algoritme skanner roboten omgivelsene, og deretter følger den veggen til det genererte kartet.

Fordi retninger blir nummerert er det muligheter for å kjøre flere roboter samtidig. Det kan brukes kjente algoritmer for å finne korteste vei mellom noder og med riktige definerte stier kan kollisjoner unngås.



Figur E.1: Viser de første iterasjonene fra startposisjon.



Figur E.2: Viser punktet der roboten er ≪i bunnen≫, venstre, og når roboten er ferdig, høyre.

Bibliografi

[1] HiTecnic, firma som produserer sensorer for Lego Mindstorms. http://www.hitechnic.com/.

[2] The cas robot navigation toolbox. http://www.cas.kth.se/toolbox/.

[3] Robotc. http://www.robotc.net/.

[4] RWTHMindstorms toolbox. http://www.mindstorms.rwth-aachen.de/.

[5] Traumaux algoritme. http://en.wikipedia.org/wiki/Maze_solving_algorithm.

[6] Jon Martin Harstad Bakken. Bygge og programmere ny legorobot. Technical report, NTNU, Høst 2008.

[7] GNU. General public license. http://www.gnu.org/licenses/.

[8] Sigurd Hannaas. Samarbeidende legoroboter, 2011.

[9] Hassenplug. Team hassenplug. http://www.teamhassenplug.org/NXT/NXTSoftware.html.

[10] Trond Kåre Homestad. Fjernstyring av legorobot. Technical report, NTNU, 2012.

[11] Trond Magnussen. Fjernstyring av legorobot. Technical report, NTNU, 2007.

[12] Microsoft. MSDN Microsoft Robotics. http://www.microsoft.com/robotics/.

[13] Johannes Schrimpf. Forbedring av sanntidsegenskapene til en legorobot. Technical report, NTNU, Høst 2007.

[14] Bjørn Syvertsen. ukjent tittel, 2006.

[15] Jens Kristian Tøraasen. Kartlegging ved hjelp av robot og kamerasensor. Technical report, NTNU, Desember 2009.

[16] Jannicke Selnes Tusvik. Fjernstyring av legorobot. Technical report, NTNU, 2009.

[17] Øyvind Auestad. Førerløs bil. Technical report, NTNU, Vår 2011.