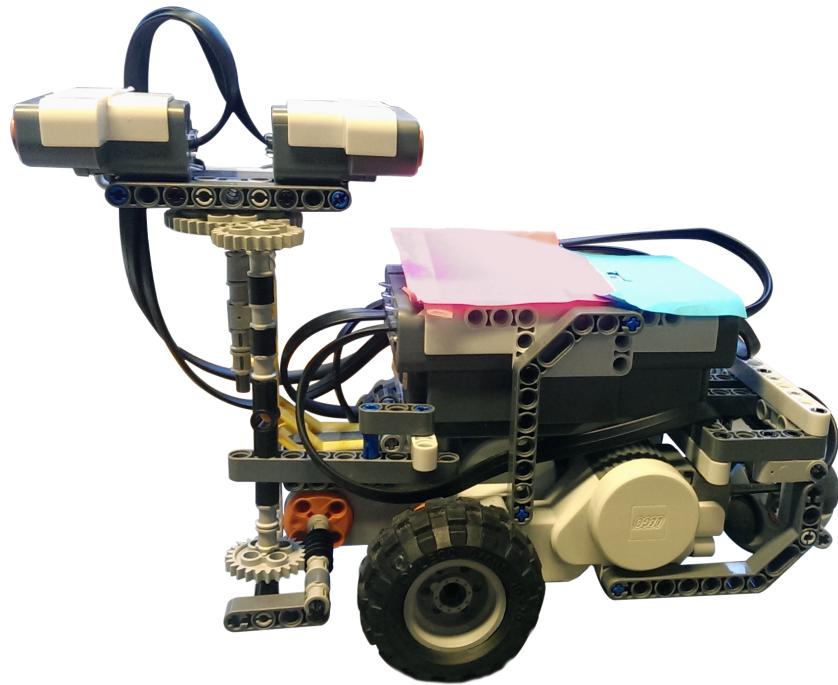

MAPPING MED LEGO-ROBOT

SW505E13



Indlejrede systemer

Efterårsssemesteret 2013

Institut for Datalogi

Selma Lagerløfs Vej 300
9220 Aalborg Ø
Phone (+45) 9940 9940
Fax (+45) 9940 9798
<http://cs.aau.dk>

Titel:

Mapping med LEGO-robot

Abstract:

Tema:

Indlejrede systemer

Projektperiode:

01-09-2013 -
20-12-2013

Projektgruppe:

sw505f13

Deltagere:

Anders R. Nielsen
Bruno Thalmann
Mikael E. Christensen
Mikkel S. Larsen
Stefan M. G. Micheelsen
Stefan M. Thilemann

Vejleder:

Nicolaj Søndberg-Jeppesen

Printings: 2

Pages: 95

Appendices: 11

Total pages: 107

Source code: <https://github.com/deaddog/sw505-code/tree/v1.0>
vejledning i appendiks F. II

Purpose: The purpose of this project was to map an unknown area using a robot, which location is given by an external device and thus, always known.

Method: To accomplish this, a robot was built using Lego Mindstorms, consisting of a rotating ultrasonic sensor construction built on top of robot. The purpose of the robot was to navigate and collect sensor measurements and provide these to a PC, which would construct a map based on the supplied data. This would result in a continuously updated occupancy grid, by using a simple- or Gaussian sensor model). Additionally, as a simplification of the problem, a PC would supply the robot with its location and orientation (pose), based on color-tracking a live stream from a Kinect, whenever needed.

Results and conclusion: The evaluation of the solution showed that the robot was able to map a room with obstacles, and navigate using the generated map.

Rapportens indhold er frit tilgængeligt, men offentliggørelse (med kildeangivelse) må kun ske efter aftale med forfatterne.

Forord

Denne rapport er udarbejdet af Software-Ingeniør studerende på 5. semester, på Aalborg Universitet, efterårssemestret 2013. Det forventes at læseren har en baggrund indenfor IT/software grundet det tekniske indhold.

Hvert kapitel starter med en introduktion, hvor indhold og udførelse kort belyses. Yderligere vil der, hvor det er relevant, blive givet en opsummering af kapitlet.

Referencer og citationer noteres med nummernotation, fx [1], hvilket refererer til den første kilde i litteraturlisten. Når der refereres til 'vi', menes der projektgruppens medlemmer.

Vi vil gerne sige tak til vores undervisere, samt vores vejleder Nicolaj Søndberg-Jeppesen for et godt samarbejde undervejs i projektforløbet.

Indhold

Introduktion	1
Baggrund	2
Problem	4
I Valg af løsningsmetode	6
1 Platform	7
1.1 Lego Mindstorms	7
1.2 NXT API	10
1.3 Beskrivelse af API'er	11
2 Sensorer og Motorer	16
2.1 Afstandssensorer	16
2.2 Motor	20
2.3 HiTechnic NXT Kompas Sensor	21
3 Lokalisering	25
3.1 Microsoft Kinect	25
3.2 Kinect for Windows SDK	28
3.3 Colortracking	30
4 Mapping	31
4.1 Metode til kortlægning	31
5 Opsummering	32

II Teori	33
6 Notation og sandsynlighedsteori	34
6.1 Notation for probabilistisk robotstyring	34
6.2 Sandsynlighedsteori	35
7 Mapping	43
7.1 Overblik	43
7.2 Vertikal og horisontal sensor model	43
7.3 Occupancy Grid algoritmen	46
8 Lokalisering	49
8.1 Farveforskelse	49
8.2 Forbedringer	52
8.3 Justering	54
8.4 Opdateringshastighed	56
8.5 Omregning fra punkt i billede til reelt punkt	57
9 Ruteplanlægning	59
9.1 Overordnet beskrivelse	59
9.2 Beregning af næste målings-celle	60
9.3 Ingen mulig rute	61
III Design	62
10 Robottens design	63
10.1 Kroppen	63
10.2 Fremdrift	63
10.3 Sensorer	64
10.4 Gearing	64
11 Forsøgsopstilling	67
11.1 Formål	67
11.2 Opsætning	67
12 System Arkitektur	71
12.1 NXT Software	72
12.2 PC Software	73
12.3 Kommunikationsprotokol	77
12.4 Flow- og Kodeeksempler	80

IV Konklusion	86
13 Evaluering	87
13.1 Observationer af robotten	87
13.2 Formål	88
13.3 Vurderingsmål	88
13.4 Test	89
14 Konklusion	93
15 Perspektivering	94
15.1 Implementering	94
15.2 Robotkonstruktion	95
15.3 Anvendelse	95
V Bilag	96
A Resultater fra test af ultrasonisk sensor	97
B Test af infrarød sensor	99
C Test af motor sensor	100
D Test af kompas sensor	101
E Sekvensdiagrammer	103
F Kildekode	105

Introduktion

Robotter bliver i større og større omfang benyttet til at automatisere komplicerede opgaver i alle brancher.^[2] Således kan man bruge robotter til lagerstyring, til at udforske et ufremkommeligt område eller fx også til at hjælpe ældre i deres hverdag.

For at robotten kan navigere og arbejde så optimalt som muligt, kræver det at den har et kort over det område den skal arbejde i. En lagerrobot kræver et kort over lageret, så den ikke kører ind i vægge når den fx fragter objekter rundt. En støvsugerrobot skal kende de rum den skal støvsuge, så den kan navigere rundt om fx stoleben, og stadig få støvsuget hele rummet.

For at lave dette kort kan robotten udstyres med sensorer og ud fra de opsamlede data konstruere et kort.

Dette problem fører til den initierende problemstilling:

Er det muligt at kortlægge et ukendt areal ved hjælp af en robot?

Denne rapport vil undersøge, hvordan dette problem kan løses samt beskrive den relevante teori. Baseret på denne teori vil der blive udviklet et system, der kortlægger et rum ved at bruge sensormålinger opsamlet af en robot, der kører rundt i rummet.

Baggrund

Når en robot skal kortlægge et område, er der overordnet set to problemer. Det første er, at robotten ikke ved hvor den befinder sig, dvs. den har ikke nogen lokation. Det andet problem er, at den ikke har et kort over steder, hvor den har været. Begge problemer set som ét problem er kendt som SLAM, hvilket er beskrevet i det følgende afsnit.

SLAM

SLAM (Simultaneous Localization and Mapping) opstår når man skal kortlægge et område uden at kende robottens lokation. De to problemer er afhængige af hinanden eftersom robotten skal kende sin lokation for at lave et kort, og for at kende dens lokation skal den bruge et kort. Uden brug af eksternt udstyr, der kan fortælle robotten hvor den befinder sig, er den nødsaget til at bruge dens sensorer til at finde ud af, hvad dens omgivelser består af. Den bruger samtidig dens motorer til at bevæge sig, hvilket kan bruges til at give den relative lokation i forhold til tidligere lokationer. Der er dog et problem; usikkerheden på sensorer og motorer gør, at robotten ikke nødvendigvis befinner sig, hvor den faktisk tror den er.

Da ovenstående er meget komplekst [3, s. 514], vil vi koncentrere os om at konstruere et kort, hvor det antages, at lokationen er givet på forhånd. Denne problemstilling er beskrevet yderligere i det følgende afsnit.

Mapping med kendt lokation

Når vi kender lokationen bliver ovenstående problem mindre komplekst. I tilfælde, hvor robotten kender sin lokation, skal den blot kortlægge det ukendte område. Kortlægningen fungerer ved at robotten bruger dens sensorer til at finde ud af, hvordan omgivelserne ser ud, og finder så sin umiddelbare lokation vha. fx GPS.

Denne fremgangsmåde fjerner mange af de usikkerheder, der opstår, når

der fokuseres på SLAM problemet, og simplificerer således processen for at finde robottens lokation.

Problem

Da baggrunden for projektet er fastlagt, defineres nu en problemformulering, der kort og præcis beskriver hvilket problem vi ønsker at løse. Først afgrænses problemet, for på den måde at simplificere det, og dermed gøre det realistisk at løse inden for den givne tidsrammen. Derefter præsenteres problemformuleringen, som projektet vil arbejde ud fra. Til sidst vil en målsætning blive sat, så det er muligt at evaluere på resultatet til sidst i projektet.

Afgrænsning

Pga. den stramme tidsramme foretages en række afgrænsninger. For at undgå at skulle tage højde for en masse specialtilfælde antages det, at robotten altid tager sensormålinger når den står vinkelret på den bane, den kører på. Desuden skal alle objekter i verdenen dermed også være vinkelrette. En anden afgrænsning er, at robottens verden er inde i et afgrænset område, hvilket gør lokaliseringen af robotten mere simpel. Desuden afgrænses robottens verden til at være plan og indendørs. Dette gør kravene til konstruktionen af robotten simple, så fokus kan holdes på at lave en algoritme til mapping.

Afgrænsningerne er således:

- Robottens verden er 90 grader
- Verdenen er et afgrænset område
- Verdenen er plan og befinner sig indendøre

Problemformulering

Vi vil arbejde ud fra følgende problemstilling på baggrund af ovenstående afgrænsning:

Hvordan kan der konstrueres software til en robot, hvis formål er at kortlægge en ukendt verden, forudsat at den til enhver tid kender sin position?

Målsætning

For at kunne evaluere på det færdige produkt, er det nødvendigt at have en konkret målsætning, der kan sammenlignes på.

I forhold til dette projekt, er der forskellige muligheder. Man kan forsøge at kortlægge et rum på kortest mulig tid, eller man kan forsøge at kortlægge rummet så præcist som muligt.

- Hastigheden af kortlægningen vil afhænge af robotkonstruktionen, sensorernes hastighed og algoritmernes kompleksitet.
- Præcisionen af kortlægningen vil afhænge af sensorernes præcision og algoritmernes præcision.

Det er i projektgruppen besluttet at prioritere præcisionen af kortlægningen højst, hvilket udtrykkes i denne målsætning.

Målet med dette projekt er følgende:

- At bygge en robot, der kan konstruere et præcist kort.

I projektgruppen vurderer vi et kort til at være præcist, såfremt at det er muligt for en robot, alene ud fra informationen i kortet, at navigere sig fra det ene hjørne af området til det diagonalt modsatte hjørne uden at støde ind i det forskellige objekter der er placeret i området eller områdets ydre afgrænsning.

Del I

Valg af løsningsmetode

I denne del beskrives de valgte dele som tilsammen udgør den endelige løsningsmetode. Først beskrives den valgte platform, og hvordan det er valgt at programmere denne. Dernæst undersøges og testes de sensorer der er stillet til rådighed gennem universitetet. Afslutningsvis beskrives hvordan robotten positioneres i verden vha. en Microsoft Kinect samt hvordan occupancy grid kan benyttes til kortlægning.

Kapitel 1

Platform

I dette kapitel vil den valgte platform, Lego Mindstorms, kort blive beskrevet, sammen med en begrundelse for dette valg. Yderligere argumenteres der for valg af API til NXTen.

1.1 Lego Mindstorms

Lego Mindstorms er et byggesæt, hvor det er muligt at bygge programmerbare robotter i Lego-klodser.

Til at bygge disse robotter, er der i Lego Mindstorms nogle sensorer og aktuatorer. Sensorerne gør det muligt for robotten at modtage input fra sine omgivelser. Ved brug af aktuatorerne kan robotten agere i verden og evt. reagere på inputs fra sine sensorer.

Ud over de originale Lego dele, er der også tredjeparts producenter, som har et udbud af andre typer sensorer og aktuatorer, der kan samarbejde med Lego Mindstorms.

1.1.1 NXT

Denne sektion er baseret på [4], hvilket omhandler NXT 2.0, som er den version der bruges i dette projekt. NXT Intelligent Brick (oftest kaldt blot 'NXT' eller 'brick') er hjernen i Lego Mindstorms robotten. Det er den der står for at modtage og behandle input fra sensorer, samt styre de monterede aktuatorer. Et billede af NXT 2.0 kan ses på figur 1.1.

1.1.1.1 Porte

NXTen har 3 motorporte (kaldet A, B og C) og 4 sensorporte (kaldet 1, 2, 3 og 4).



Figur 1.1: NXT Intelligent Brick.

1.1.1.2 Tilslutningsmuligheder

Der kan kommunikeres med NXTen ved at tilslutte den til en anden enhed med USB-kabel eller Bluetooth.

1.1.1.3 Feedback

Til output har NXTen en 100 x 64 pixel LCD display, samt en 8 kHz højttaler.

1.1.1.4 Styring

NXTen kan styres på to måder: Man kan sende kommandoer og modtage beskeder (for eksempel sensor aflæsninger) på en ekstern enhed (oftest en computer eller en anden NXT). Alternativt kan programmer sendes (via Bluetooth eller USB) til NXTen, hvorfra de kan køres direkte, uafhængig af eksterne enheder.

1.1.1.5 Tekniske specifikationer

De tekniske specifikationer for NXT 2.0 kan ses i tabel 1.1 [4].

Microcontroller	32-bit ARM7 microcontroller 8-bit AVR controller
RAM	256 Kb FLASH, 64 Kb RAM 4 Kb FLASH, 512 b RAM
Kommunikation	Bluetooth (Bluetooth Class II V2.0 compliant) USB full speed port (12 Mbit/s)
Input porte	4
Output porte	3
Display	100 x 64 pixel LCD
Højttaler	8 kHz lyd. 8-bit lydkanal og 2-16 kHz sample rate
Strømkilde	6 AA batteries

Tabel 1.1: Lego Mindstorms NXT tekniske specifikationer.

1.1.2 Valg af Lego Mindstorms

Der er mange gode grunde til at vælge Lego Mindstorms. Her er givet fire overordnede punkter, der efterfølgende vil blive gennemgået:

- Tilgængelighed
- Nemt at gå til
- Stort udvalg af sensorer
- Mange muligheder ift. styring

1.1.2.1 Tilgængelighed

Grundet at Lego (inkl. Lego Mindstorms) er ment til almindelige brugere, er det masseproduceret og derved kan købes forholdsvis billigt og i helt almindelige butikker (legetøjsforretninger og ofte også supermarkeder).

1.1.2.2 Nemt at gå til

Det faktum at man bygger sin robot i Lego klodser, med tilføjelse af Lego Mindstorms sensorer/aktautorer gør det nemt at lave en konstruktion og derefter modificere den, så den udfører sin opgave tilfredsstillende.

Denne høje alsidighed gør, at Lego er perfekt til en prototype-orienteret fremgangsmåde for bygning af robotter.

1.1.2.3 Stort udvalg af sensorer

På grund af det store udvalg af sensorer, kan man bygge robotter der kan løse et væld af opgaver. Ved et projekt med høj usikkerhed, er det derved også nemt og billigt at udskifte en sensor, hvis den ikke virker tilfredsstillende.

1.1.2.4 Mange muligheder ift. styring

Fleksibiliteten i forhold til styring er ment både som den overordnet måde hvorpå robotten styres, men især også den måde hvorpå NXTen styres. NXTen kan udstyres med brugerdefineret firmware, der kan tilføje ekstra funktionalitet i forhold til Legos egen.

I afsnittet herefter vil der blive argumenteret for valg af API, hvor API her skal forstås som måden hvorpå robotten skal styres.

1.2 NXT API

Den ideelle løsning ville være at vælge et system, hvor robotten styrer alt; navigation og indsamling af sensormålinger, samt selve kortlægningsdelen. Dog er der begrænset med plads til data på NXTen, samtidig med at den er begrænset af dens regnekraft (se evt. tabel 1.1). På grund af robottens hardwaremæssige begrænsninger og at problemformuleringen siger, at robotten til enhver tid skal kende sin position, er det derfor nødvendigt at 'out-source' visse opgaver til en PC. Dette ses blandt andet ved, at der benyttes en Microsoft Kinect til at lokalisere robotten, hvilket gør en PC påkrævet for at bearbejde billeddata fra Kinecten, for netop at fortælle robotten dens faktiske position.

Grundet disse begrænsninger og givet problemet der skal løses, er det muligt at lave et to-delt system. Den ene del bestående af selve robotten, som skal navigere sig rundt i verden og bruge sine sensorer til at opfatte verden omkring sig. Den anden del, som består af selve kortlægningen, kræver mere plads til data, samt større beregningskraft.

Overordnet valg

For at overholde ovenstående, skal der laves et to-delt stykke software, hvor den ene del kører på robotten og den anden del på en stærkere platform, i vores tilfælde en PC.

Dernæst vælges hvordan det software, der skal køre på henholdsvis robotten og på PC'en skal laves. Vigtigt er, at det skal være muligt for de to

enheder at kommunikere, da der skal kunne sendes kommandoer til robotten fra PCen, samt at PCen skal kunne modtage sensormålinger fra robotten.

NXC

Til det software, der skal køre på robotten, har vi valgt NXC, da det er simpelt og dækker alle behov. NXC-programmer skrives i et C-lignende sprog, som kan sendes til NXTen og køres direkte der på. Der er også stor mulighed for kommunikation mellem NXC-programmer og eksterne programmer. Dette gør det muligt at skrive et program til NXTen, hvorefter der kan udføres forskellige handlinger, afhængig af hvad PCen beder den om. Der vil blive gået mere i dybden med NXC i afsnit [1.3.2](#).

MindSqualls

Til det software, der skal køres på PC, altså det der skal sørge for selve opdateringen og beregningen af kort, har vi valgt MindSqualls. MindSqualls er et .NET bibliotek skrevet i C#, hvori det er muligt at kommunikere med en NXT, både ved at sende direkte kommandoer eller via abstraktion over NXTens aktuatorer/sensorer opfattet som objekter. Valget faldt på MindSqualls da dette var simpelt og dækkede alle behov ift. kommunikation med NXT. Desuden har gruppen stor erfaring med C# og Visual Studio, hvilket også ses som en stor fordel. MindSqualls vil blive uddybet i afsnit [1.3.1](#).

1.3 Beskrivelse af API'er

I dette afsnit vil de API'er, der er valgt i afsnit [1.2](#), blive beskrevet.

1.3.1 MindSqualls

MindSqualls er et .NET bibliotek til Legos NXT robotter. MindSqualls tilslader kommunikation med NXT enheder via. Bluetooth og USB og fungerer ved at sende og modtage beskeder over disse to teknologier. Biblioteket er skrevet i C# og kan dermed anvendes af alle de .NET kompatible sprog.

1.3.1.1 Kommunikation

Kommunikationen fra Mindsqualls til NXT enheden sker igennem klassen `NxtCommunicationProtocol`, som findes i en `McNxtBrick`. En `McNxtBrick` er en abstraktion over hele NXT brick enheden. Denne kan således også

styre motorer og sensorer direkte, men da dette projekt håndterer al kørsel og sensormåling på robotten, er kun kommunikationsprotokollen nødvendig. `NxtCommunicationProtocol` indeholder metoder til at sende og modtage beskeder. Metoderne til dette er:

- `MessageWrite(NxtMailbox inBox, string messageData)`
- `MessageReadToBytes(NxtMailbox2 remoteInboxNo, NxtMailbox localInboxNo, bool remove)`

`NxtMailBox` er en enumeration type, der indeholder de forskellige mailboxes, der er tilgængelige på NXT enheden.

Brugen af `MessageWrite` kan ses i kodeeksempel 1.1 hvor en streng sendes til outboxen `PC_OUT`. `CommunicationBrick` er her en `McNxtBrick` og `CommLink` er en property, der returnerer den `NxtCommunicationProtocol`, der er tilknyttet `CommunicationBrick`

```
1 public void SendMessage(string s)
2 {
3     CommunicationBrick.CommLink.MessageWrite(PC_OUTBOX
4         , s);
}
```

Kodeeksempel 1.1: Brug af `MessageWrite`.

Brugen af `MessageReadToBytes` kan ses i kodeeksempel 1.2. Her kigger `CommLink` i mailboxen `PC_INBOX`, og hvis der er en besked returneres denne som et byte array. Hvis der ikke er en besked i inboxen smides en `NxtCommunicationProtocolException` exception.

```
1 byte[] msg = CommunicationBrick.CommLink.
2     MessageReadToBytes(PC_INBOX, NxtMailbox.Box0,
3     true);
```

Kodeeksempel 1.2: Brug af `MessageReadToBytes`.

1.3.2 NXC

Til at programmere robotten (NXTen) har vi valgt at benytte NXC; et programmeringssprog designet til NXT robotten. Brugen af NXC til at styre robotten vil blive beskrevet her.^[5] Desuden bruges Enhanced Firmware, for at få adgang til trigonometriske funktioner, som er nødvendige for projektet.^[6]

1.3.2.1 NXC sproget

NXC har en syntax der minder meget om C syntax. Ud over de traditionelle programmeringskonstruktioner indeholder NXC en masse standardfunktioner til at interagere med robottens sensorer og motorer.

1.3.2.2 Sensorer og motorer

NXC sproget indeholder funktioner til at styre sensorer og motorer tilsluttet NXTen. Et eksempel på aktivering af en sensor og en motor kan ses i kodeeksempel 1.3. `SetSensor` bruges først til at fortælle robotten at indgang 1 er en touch sensor. Sensorens værdi aflæses i linje 5 fra en variabel tilknyttet porten. Moterne i port A og C instrueres til at køre med 75 % hastighed i linje 4 med `OnFwd`. Der findes tilsvarende en `OnRev` til at køre baglæns. Motoren kører indtil der trykkes på touch sensoren, hvorefter `Off` kaldes for at standse motorerne.

```
1 task main()
2 {
3     SetSensor(IN_1, SENSOR_TOUCH);
4     OnFwd(OUT_AC, 75);
5     until (SENSOR_1 == 1);
6     Off(OUT_AC);
7 }
```

Kodeeksempel 1.3: Brug af motorer og sensorer.

1.3.2.3 Kommunikation med PC

Da der skal bruges en ekstern enhed (en PC) til visse dele af systemet, er det nødvendigt at kunne kommunikere med en PC fra NXTen. Til kommunikation via Bluetooth findes der på NXT et antal mailboxe. Disse anvendes til at sende og modtage beskeder. Et eksempel på at modtage en besked kan ses i kodeeksempel 1.4. `ReceiveRemoteString` bruges til at kigge om der er en ny besked i INBOX. Denne bliver fjernet fra inboxen, hvis det andet parameter er `true`, og beskeden bliver læst over i strengen `in`.

```
1 #define INBOX 5
2 #define OUTBOX 6
3
4 while(true){
5     ReceiveRemoteString(INBOX, true, in)
```

```

6   if(StrLen(in) > 0)
7   {
8       tmp = SubStr(in,0,1);
9       packet = StrToNum(tmp);
10      PlayToneEx(100* packet, 200, 3, false);
11  }
12  in = "";
13 }
14 }
```

Kodeeksempel 1.4: Et eksempel på at modtage beskeder over Bluetooth.

I kodeeksempel 1.5 er den tilhørende kode i MindSqualls, der afsender en besked. Først gemmes beskeden i en streng og derefter afsendes beskeden via `NxtCommunicationProtocol` klassens `MessageWrite` funktion.

```

1 private const NxtMailbox2 PC_INBOX = NxtMailbox2.
2     Box6;
3 private const NxtMailbox PC_OUTBOX = NxtMailbox.Box5
4 ;
5 public static void NxtTone(NxtCommunicationProtocol
6     commLink, int tone)
7 {
8     string messageData = string.Format("{0}", (byte)
9         tone);
10    commLink.MessageWrite(PC_OUTBOX, messageData);
11 }
```

Kodeeksempel 1.5: MindSqualls kode der afsender en besked.

Afsendelse af beskeder virker tilsvarende ved brug af kommandoen `SendMessage`. Et eksempel på at afsende en besked kan ses i kodeeksempel 1.6. `SendMessage` benyttes til at sende beskeden `msg` til `OUTBOX`.

```

1 while(true){
2     string msg = NumToStr(SENSOR_1);
3     SendMessage(OUTBOX, msg);
4 }
```

Kodeeksempel 1.6: Eksempel på afsending af besked.

I kodeeksempel 1.7 ses den tilhørende MindSqualls kode, der læser en besked sendt fra NXT'en.

```
1 public static string NxtSensorReading(
2     NxtCommunicationProtocol commLink)
3 {
4     return commLink.MessageRead(PC_INBOX, NxtMailbox.
5         Box0, true);
6 }
```

Kodeeksempel 1.7: MindSqualls kode der læser en besked sendt fra NXT'en.

Kapitel 2

Sensorer og Motorer

Følgende afsnit undersøger de tilgængelige sensorer og motorer, som er interessante for projektet. Da der er stor forskel på sensorerne, der er til rådighed, er der foretaget forskellige forsøg, hvis formål er at klarlægge præcisionen af de forskellige sensorer.



(a) Ultrasonisk sensor



(b) Infrarød sensor



(c) Servomotor



(d) Kompas

Figur 2.1: Komponenter der betragtes til konstruktion af robotten.

2.1 Afstandssensorer

Dette afsnit beskriver testen af de to afstandssensorer (figurer 2.1a og 2.1b), som kan være interessante at montere på robotten. For at afgøre hvilken der passer bedst til at løse problemet, er de blevet testet.

Formålet med at teste disse sensorer er, at robotten skal have en måde at afgøre på, hvor langt der er til objekter omkring den. Derfor vil disse tests undersøge følgende:

- Hvor nøjagtigt kan sensorerne bestemme afstanden til et objekt?
- Hvilket interval kan der måles i (hvad er den minimale og maksimale måleafstand)?

2.1.1 Ultrasonisk Sensor

Den ultrasoniske sensor, som kan ses på figur 2.1a, kan måle afstand til objekter.

Det gøres ved at sende en lydbølge, hvorefter der beregnes hvor lang tid det tager for denne at ramme objektet, for derefter at blive reflekteret tilbage igen. Den maksimale afstand der kan måles er 255 cm, med en præcision på ± 3 centimeter. De bedste aflæsninger fås ved måling af store flade objekter med hård overflade, i modsætning til mindre objekter med rund og/eller blød overflade.^[4]

2.1.1.1 Test

Der er gennemført en mindre test af sensoren, for at finde ud af hvor nøjagtig den er.

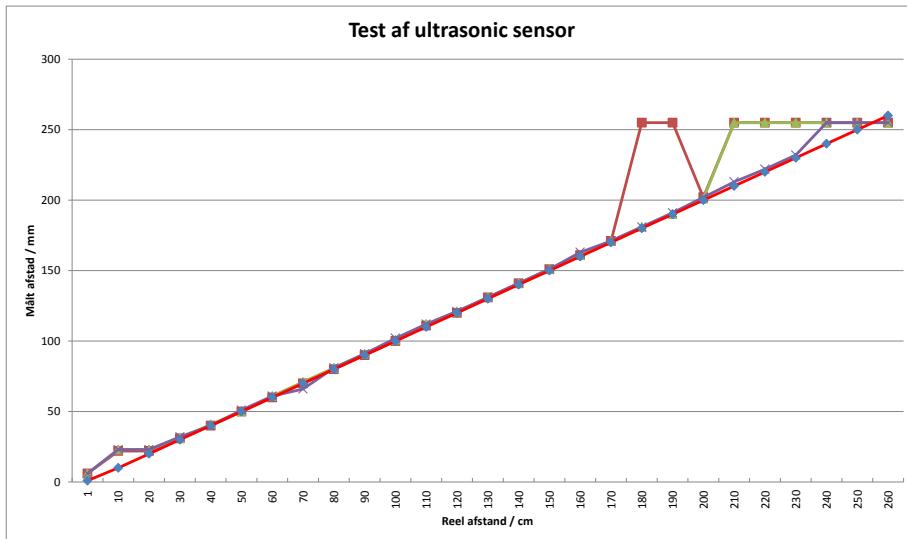
Testen blev udført ved at lave en simpel Lego konstruktion, kun bestående af NXT og den ultrasoniske sensor. Konstruktionen blev placeret i en bestemt afstand fra en væg, hvor sensoreren var placeret vandret, pegende direkte på væggen. Herefter blev sensoren aflæst, samtidig med at afstanden mellem væg og sensor blev målt med lineal. Der blev udført den samme test 3 gange, hver gang for en bestemt række afstande mellem 0 og 255 cm.

2.1.1.2 Resultater

Tabel med resultater fra forsøget kan ses i appendiks A.

En graf-repræsentation af resultaterne kan ses i figur 2.2. Den røde linje indikerer det ideelle resultat. mens den brune, grønne og lilla linje viser resultaterne fra de tre test. Det ses på grafen, at de tre test alle giver forkerte resultater, når sensoren er mindre en 20 cm fra objektet. Desuden kan det ses, at der i test 1 (brun) kun kunne måles op til 170 cm, før der opstod usikkerhed, hvor de andre tests nåede 200 cm (test 2 – grøn) og 230 cm (test 3 – lilla) før der opstod større usikkerhed. Generelt overholder sensoren dens specifikationer, idet der er en afvigelse på 3 cm på målingerne, men forsøget

viser, at der kun kan måles mellem 20 og 170 cm, uden der opstår større usikkerheder.



Figur 2.2: Graf over forsøgsresultaterne fra testen af den ultrasoniske sensor.

2.1.2 Infrarød Sensor

Den infrarøde afstandssensor fra mindsensors.com, som kan ses på figur 2.1b, er en afstandssensor med høj præcision, der kan måle afstande mellem 10 og 80 cm. Sensoren virker som den ultrasoniske sensor, men sender et infrarødt lys i stedet for en ultrasonisk impuls.

2.1.2.1 Test

For at afprøve sensorens præcision og rækkevidde, er der foretaget en test af sensoren.

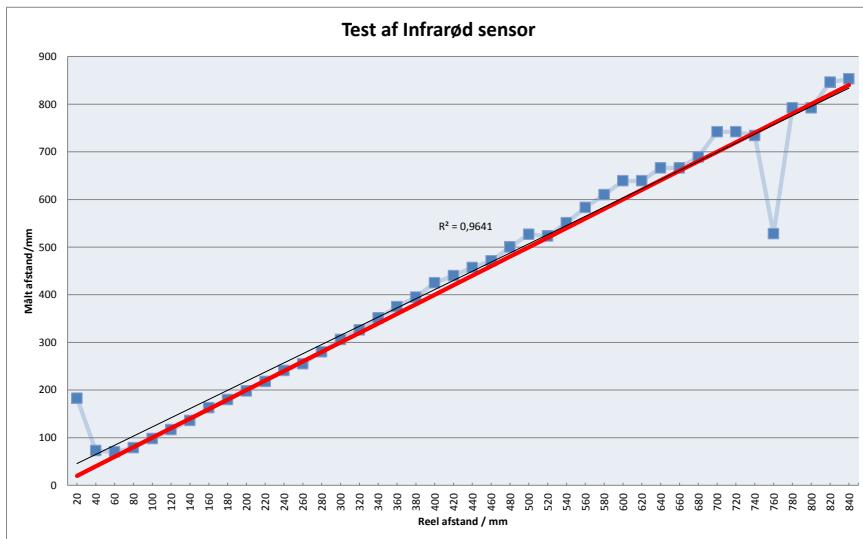
Opstillingen brugt til udførelse af testen bestod af tre A4 ark spændt ud på gulvet ind til en væg. Papiret havde markeringer for hver 2 cm fra væggen.

Udførelsen af testen gik ud på at placere en NXT med påsat infrarød sensor ved hver indikator, for derefter at aflæse sensorens måling. På denne måde blev afmålingen fra sensoren ved en given afstand holdt op imod den egentlige afstand til væggen, som afmålt på papiret.

2.1.2.2 Resultater

Resultaterne fra testen er præsenteret i appendiks B. Disse resultater er vist på figur 2.3. Den røde graf er den reelle afstand, som aflæst på papiret/med lineal, mens de blå firkanter er målepunkter. Den sorte graf er en lineær regression over måleresultaterne.

Ud fra grafen ses det, at der er stor usikkerhed i starten og slutningen, hvilket næsten stemmer overens med den lovede rækkevidde på 10 til 80 cm. De bedste resultater findes dog imellem 8cm og 56 cm. Mellem 8cm og 56cm er der maximalt 27 mm afvigelse fra det forventede, (forventet 500, målt 527).



Figur 2.3: Graf over forsøgsresultaterne fra testen af den infrarøde sensor.

2.1.3 Sammenligning af afstandssensorer

Der er foretaget test af to forskellige afstandssensorer, den ultrasoniske sensor og den infrarøde sensor. Den ultrasoniske sensor var præcis med en afvigelse på ± 3 cm mellem 20 cm og 170 cm. Den infrarøde sensor var præcis med afvigelse på ± 3 cm, mellem 10 cm og 56 cm. Den infrarøde sensor angiver sine målinger med højere præcision, men da fejlmarginen er 3 cm, anses den ikke som værende mere præcis end den ultrasoniske sensor.

Med en fejlmargin på 3cm, i de respektive intervaller, anses begge afstandssensorer for at være præcise nok til anvendelse i projektet. I forhold til rækkevidde er den ultrasoniske sensor den mest *præcise* af de to på større afstand, mens den infrarøde sensor er den mest *præcise* på kort afstand.

Valget mellem de to sensorer er derfor afhængig af hvilket krav robotten har til rækkevidde.

2.2 Motor

Motoren, der er testet her, er fremstillet af Lego og kan ses i figur 2.1c. Den består af en rotationssensor, som mäter omdrejningerne ved grader, med en nøjagtighed på én grad i følge Lego. Desuden gør denne sensor det også muligt at styre hvor meget kraft motoren skal køre med. Køres der med flere motorer har NXT'en indbygget software, der gør det muligt at synkronisere disse, hvilket er hensigtsmæssig hvis fx. den ene motor skulle være stærkere eller svagere end den anden.^[7]

2.2.1 Formål

Formålet med denne test er at teste motorens nøjagtighed. Hvis motoren ikke har høj nøjagtighed, skal der tages højde for det, når den monteres på robotten.

2.2.2 Test

For at bestemme motorens nøjagtighed i praksis, blev en test opstillet hvor to motorer roteres med et bestemt antal grader. Motorernes egentlige rotation aflæses med vinkelmåler og holdes op mod den ønskede rotation, samt den rotation motorerne angiver at de har roteret. Sidstnævnte fås ved at aflæse motorernes TachoCount egenskab.

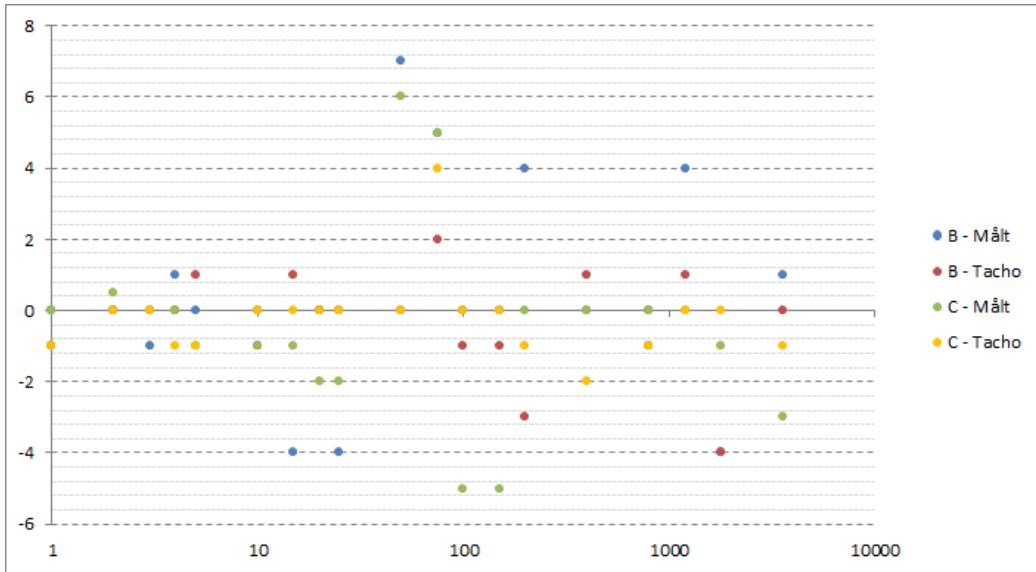
2.2.2.1 Resultat

Resultaterne fra forsøget kan ses i appendiks C. Af resultaterne kan vi bestemme de største afvigelser til +7 og -4 grader. Tacho-værdien har en afvigelse på maksimalt +1 og -4 grad. Dette stemmer ikke overens med de antagede ± 1 grads afvigelse.

+7 og +6 grader målt optræder kun én gang. Ellers er den maksimale målte afvigelse ± 4 grader, men selvom vi antager, at det er pga. unøjagtige målinger ved forsøgsudførelsen, er det stadig ikke ± 1 grad.

Af figur 2.4 ses det at målingerne ligger længst udenfor det optimale (sorte stippled linje), hvilket er en god indikator for, at det er bedre at benytte tacho-værdien.

Desuden kan vi se på samme figur, at motor b er meget mere unøjagtig end motor c. Ideelt ville de to motorer give det samme resultat.



Figur 2.4: Afgigelser i test-resultaterne.

Dvs. den målte værdi har en afvigelse på ± 4 grader, mens tacho-værdien har en afvigelse på +1 og -4 grader, og at synkronisering af de to motorer ikke giver den ønskede effekt.

Afgigelserne i motorernes præcision er ofte ikke store, og evt. fejl vil delvist blive afhjulpet af det system der konstrueres til lokalisering af robotten. Det vurderes derfor at præcisionen af motorerne er god nok til anvendelse i projektet.

2.3 HiTechnic NXT Kompas Sensor

Kompasset fra HiTechnic er et digitalt kompas, der måler jordens magnetiske felt. Kompasset kan returnere en værdi, der repræsenterer kompassets nuværende orientering. Ifølge HiTechnic er værdierne returneret fra kompasset præcise ned til 1 grad. Kalibrering skulle desuden ikke være nødvendig. Dog er det nødvendigt, for at minimere forstyrrelser, at kompasset holdes 10-15 cm væk fra forstyrrende elementer, herunder Lego NXT og motorer.^[8]

I afsnit 2.3.3 undersøges hvorvidt målinger fra kompasset kan leve op til disse oplysninger.

2.3.1 Formål

Formålet med denne test er at undersøge kompassets nøjagtighed. Hvis kompasset har høj nøjagtighed, kan det monteres på robotten og afgøre hvilken retning denne vender.

2.3.2 Kompas Sensor i MindSqualls

Til at styre denne sensor i MindSqualls, skal der bruges en af de separate klasser til HiTechnic sensorerne, som findes i namespacet `NXT.MindSqualls.HiTechnic`. Til kompas sensoren anvendes klassen `HiTechnicCompassSensor`.

2.3.2.1 Aflæsning af værdier

Ved aflæsning af egenskaben `Heading` fra sensorens klasse, gives et heltal mellem 0 og 359, der angiver kompassets orientering. Det har været nødvendigt at ændre implementationen af `Heading`, da den oprindelige implementation ikke aflæste kompassets orientering ned til én grad. Sensoren repræsenterer sin orientering på to forskellige måder. Begge anvender to bytes. Den ene metode beskriver afstanden vha. et 16-bit heltal. Den anden (som var anvendt i implementationen af `Heading`) beskriver afstanden ved at lade den ene byte angive vinklen i intervaller af 2 grader. Altså kunne de værdier der aflæses fra denne byte være 0-179, hvilket efterfølgende ganges med 2 for at få den egentlige værdi. Hertil lægges værdien af den anden byte, der kan have værdien 0 eller 1. I den oprindelige implementation af `Heading`, er der ikke taget højde for denne ekstra værdi. Dette er inddraget i test 3 i det følgende afsnit.

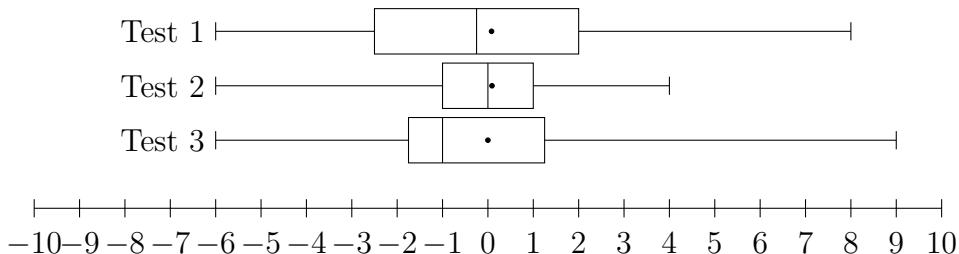
2.3.3 Præcisionstest

For at teste kompas sensoren, er målinger taget fra denne og holdt sammen med målinger foretaget med vinkelmåler. Sammenligningen er sket ved at tage to målinger med kompasset og sammenligne differencen med den værdi målt med vinkelmåler. I forbindelse med test af sensoren, blev der udført i alt tre tests:

1. Kompas monteret på robot med hjul
2. Kompas monteret på fast konstruktion
3. Gentagelse af test 2, med øget præcision

Den øgede præcision der beskrives her, dækker over en opdatering af MindSqualls klassen `HiTechnicCompassSensor`, der blev foretaget efter de første to tests. Resultaterne fra de tre tests kan ses i tabellerne [D.1](#), [D.2](#) og [D.3](#) på side [101](#).

Af resultaterne er der lavet et boksplot (figur [2.5](#)). Plottet illustrerer den afvigelse, der var fra de værdier, der blev aflæst af kompasset, til de værdier, der blev målt med vinkelmåler. Prikkerne på plottet repræsenterer den gennemsnitlige afvigelse.



Figur 2.5: Boksplot for test af kompas.

2.3.3.1 Test 1: Monteret på robot

Første forsøg blev udført med kompasset monteret ovenpå ultralyds-sensoren, for at holde en minimum-afstand på 15 cm fra brick og motorer. Denne konstruktion var dog meget ustabil, da kompasset skulle være forholdsvis højt oppe ift. base-konstruktionen.

2.3.3.2 Test 2: Monteret på stabil konstruktion

Andet forsøg blev udført med kompasset monteret på en selvstændig og langt mere stabil konstruktion, for derved at undersøge om dette kunne forbedre resultaterne.

2.3.3.3 Test 3: Forøget præcision (stabil konstruktion)

Tredje forsøg var en gentagelse af det andet, efter implementationen af `Heading` blev opdateret. Målingerne her skulle altså udtrykke en øget præcision i forhold til den forrige test. De første to test er taget med, da forbedringen af præcisionen højst kunne være én grad ved denne tredje test (se afsnit [2.3.2.1: Aflæsning af værdier](#)). Altså en marginal forbedring i forhold til testresultaterne.

2.3.3.4 Resultater

Resultaterne kan ses i appendiks D. Som det fremgår af figur 2.5 er den gennemsnitlige afvigelse (prikket) i alle test meget tæt på 0. Altså ved vi at kompasset laver lige mange og lige store (i gennemsnit) udsving til begge sider, hvilket gør det sværere at kalibrere for sådanne udsving. Samtidig viser plottet, at halvdelen af kompassets målinger er over 2-3° ved siden af de egentlige målinger - med den største afvigelse på 9°.

Det har i testfasen vist sig, at kompasset er konsistent i dets målinger. Dette betyder, at der ikke kan opnås yderligere præcision ved at foretage flere målinger af samme grad. Disse målinger vil ganske enkelt give samme resultat.

Det kan hermed konkluderes, at kompas sensoren ikke har høj nok præcision til at kunne anvendes i projektet.

Kapitel 3

Lokalisering

I forbindelse med kortlægningen af et område skal robotten have at vide hvor den er og hvordan den står i forhold til dens omgivelser. I dette kapitel vil der blive beskrevet den valgte metode i forhold til lokalisering, ved brug af Microsoft Kinect og colortracking.

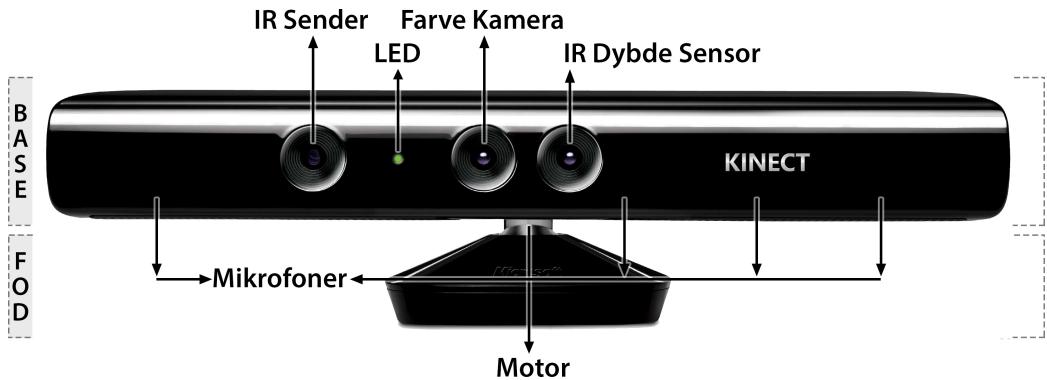
3.1 Microsoft Kinect

En Kinect er et Natural User Interface (NUI) udviklet til Microsofts spille-konsol, Xbox. Dens primære formål er at gøre det muligt at interagere med sin konsol vha. bevægelse og talte kommandoer. Dog har den også mange andre anvendelsesområder, fx.:

- Optagelse af video i real-tid
- Generere et dybde billede ved brug af kameraet og de to IR sensorer
- Vurdering af omgivelserne ved hjælp af lyd

Oprindeligt eksisterede der kun en Kinect til XBox, men i februar 2012 blev der lanceret en ny type, kaldet Kinect for Windows, der sammen med et Software Development Kit (SDK), gør det muligt at udvikle både kommercielle og ikke-kommercielle applikationer til Windows platformen.

I dette projekt skal Kinecten benyttes til at bestemme robottens placering i ukendte omgivelser. Følgende afsnit giver en kort introduktion til Kinectens egenskaber.[\[9\]](#)



Figur 3.1: Microsoft Kinect for Windows.

3.1.1 Opbygning af Kinect

I denne rapport er det versionen Kinect for Windows der fokuseres på, da den giver bedst kompatibilitet med PC, samt det faktum at den er tilgængelig gennem universitetet.

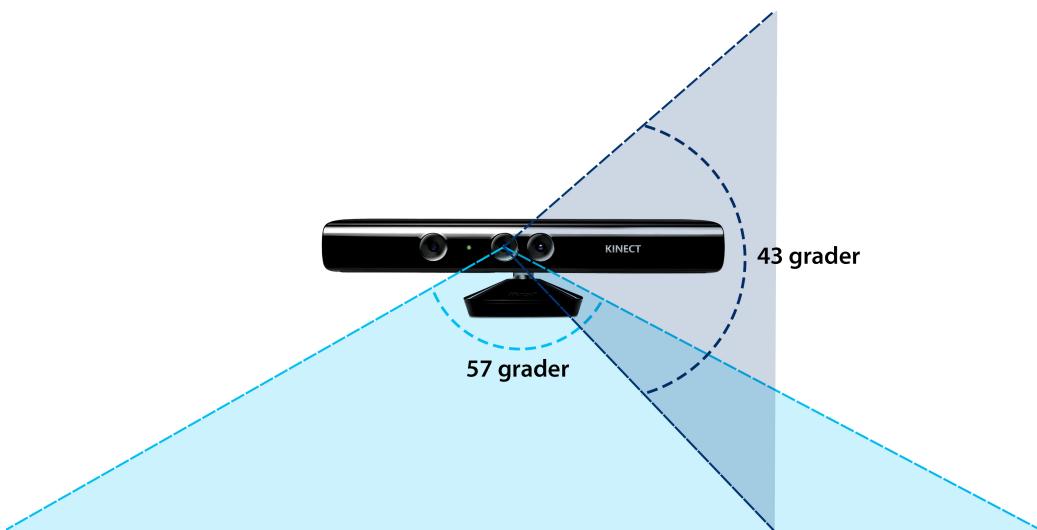
Microsoft Kinect for Windows består af en nogle hardware-komponenter, som kan tilgås via det medfølgende SDK¹. Microsoft Kinect for Windows består af følgende komponenter:

- Farvekamera
- Infrarød (IR) sender
- IR dybde sensor
- Mikrofoner
- Motor til justering af vinklen
- LED
- Accelerometer

Microsoft Kinect for Windows kan ses på figur 3.1, sammen med dens synlige komponenter, som nævnes ovenfor.

For at løse lokaliseringsproblemet benyttes Kinectens farvekamera, der ses således bort fra alle dens andre komponenter. Derfor bliver farvekameraets specifikationer på Kinecten i følgende afsnit beskrevet detaljeret.

¹Der findes også andre tredjeparts SDK til Microsoft Kinect, men som ikke er relevante ift. projektet, hvorfor de ikke nævnes.



Figur 3.2: Horisontale- og vertikale betragtningsvinkler for farvekameraet.

3.1.2 Farvekamera

For at gøre Kinecten i stand til at se, er den udstyret med et farvekamera, som er placeret ca. i midten af enheden (se evt. figur 3.1). Videoen bliver sendt som en RGB videostrøm, med mulighed for en opløsning på 1280x960 med en opdateringshastighed på 12 billede i sekundet, eller en lavere opløsning på 640x480 og 30 billede i sekundet.^[10] Kinecten har en horisontal betragtningsvinkel på 57° og en vertikal betragtningsvinkel på 43° . Et billede af Kinectens betragtningsvinkler kan ses på figur 3.2.

3.1.3 Valg af Kinect

Microsoft Kinect giver generelt mange muligheder pga. alle dens komponenter, samt brugervenligheden overfor udviklere. For at opsummere, så ser vi Kinectens overordnede fordele som:

- Nem montering
- Nem tilslutning til PC
- Mange sensorer
- Gode udviklingsværktøjer
- Tilgængelig gennem universitetet

Ovenstående fordele giver mange muligheder i forhold til hvordan Kinecten kan benyttes, men da der stræbes efter en simpel og pålidelig metode, vil dette smitte af på den endelige løsning.

3.2 Kinect for Windows SDK

Til Kinecten findes der et officielt SDK², som åbner op for al funktionalitet i Kinecten med support fra Microsoft.

API'en er bygget op omkring en solid forståelse for menneskers bevægelser og karaktertræk, således API'en kan fungere som et interface, der kan genkende personers bevægelser, følge ansigter, genkende gestus og tale.

3.2.1 Kinect API

De følgende afsnit vil fokusere på, hvordan SDK benyttes for at tilgå Kinectens mest basale funktionalitet, med fokus på hvordan billeddata hentes fra dets farvekamera.

3.2.1.1 Initialisering

Før Kinecten kan anvendes og der er adgang til dens sensorer, skal Kinecten initialiseres. Et eksempel på initialisering af en Kinect kan se i kodeeksempel 3.1. Form1_Load metoden i afsnit 3.2.1.1 køres ved applikationens opstart og sørger for at starte sensoren, hvis den findes. Fra afsnit 3.2.1.1 til afsnit 3.2.1.1 tændes for de billedstreams, der er nødvendige. I eksemplet tændes for RGB-kameraet, dybdekameraet, samt skeletdetektoren. Hvis der ikke er en Kinect tilsluttet, meldes der fejl (afsnit 3.2.1.1).

```
1 KinectSensor sensor;  
2  
3 private void Window_Loaded(object sender,  
4     RoutedEventArgs e)  
5 {  
6     if (KinectSensor.KinectSensors.Count > 0)  
7     {  
8         this.sensor = KinectSensor.KinectSensors[0];  
9         StartSensors();  
10    }  
11    this.sensor.ColorStream.Enable();
```

² Version 1.8, udgivet 17 september, 2013 [11]

```

12         this.sensor.DepthStream.Enable();
13         this.sensor.SkeletonStream.Enable();
14         this.sensor.ColorFrameReady +=
15             sensor_ColorFrameReady;
16     }
17
18     else
19     {
20         MessageBox.Show("No Kinect connected");
21         this.Close();
22     }
23
24 }
25
26 private void StartSensors()
27 {
28     if (this.sensor != null && !this.sensor.
29         IsRunning)
30     {
31         this.sensor.Start();
32     }

```

Kodeeksempel 3.1: Initialisering af en Kinect sensor.

3.2.1.2 Visning af Kinectens billeddata

For at vise de billeder, som Kinecten optager, benyttes `sensor_ColorFrameReady` eventet. Dette event affyres hver gang der modtages et nyt billede fra Kinecten. Et eksempel på visning af Kinectens data ses i kodeeksempel 3.2, hvor der i afsnit 3.2.1.2 gemmes en reference til den frame, der netop er blevet optaget af Kinecten. I afsnit 3.2.1.2 kopieres data over i et bytearray for at arbejde på dataen. I afsnit 3.2.1.2 sættes billedets source til et billede konstrueret ud fra dataen fra Kinecten. Billedet `imageFrame` viser nu hvad Kinecten ser.

```

1 byte[] pixelData = null;
2 void sensor_ColorFrameReady(object sender,
3                             ColorImageFrameReadyEventArgs e)
4 {
5     using (ColorImageFrame imageFrame =

```

```

6         e.OpenColorImageFrame())
7     {
8         if (imageFrame == null)
9             return;
10
11         this.pixelData =
12             new byte[imageFrame.PixelDataLength
13                     ];
14
15         imageFrame.CopyPixelDataTo(this.
16             pixelData);
17
18         int stride = imageFrame.Width *
19             imageFrame.BytesPerPixel;
20
21         this.KinectImage.Source =
22             BitmapSource.Create(
23                 imageFrame.Width, imageFrame.Height,
24                 96, 96, PixelFormats.Bgr32, null,
25                 pixelData, stride);
26     }
27 }
```

Kodeeksempel 3.2: Visning af billeddata fra Kinectens RGB-kamera.

3.3 Colortracking

Til at lokalisere robotten er det valgt at benytte Kinectens kamera og en farvegenkendelsesalgoritme. På den måde er det muligt at finde robotten i billedet uden at være afhængig af en bestemt form eller konstruktion. Derimod skal der bare være mulighed for at sætte nogle farvede mærkater på robotten. Ideen er at sætte to forskelligt farvede mærkater på kroppen af robotten og følge dem hver for sig. Dette vil da give en mulighed for at beregne midtpunktet af disse to mærkater samt robottens vinkel, hvilket svarer til robottens positur.

Kapitel 4

Mapping

Kapitel 3 beskriver hvordan robotten lokalisers i verden; dette kapitel omhandler hvordan robotten bygger et kort af verden. Der vil kort blive fortalt om hvordan dette problem kan løses vha. occupancy grid.

4.1 Metode til kortlægning

Behovet for at fremskaffe et kort over en robots omgivelser vil i den ene eller anden forstand altid være påkrævet før robotten er i stand til at interagere med det miljø, den er placeret i. Det kan fx. være et stort udendørs areal man ønsker at bygge et kort over. Til at kortlægge har vi valgt en algoritme kaldet *occupancy grid*.

4.1.1 Occupancy grid

Occupancy grid er en familie af algoritmer, som gør det muligt at generere konsistente kort ud fra målinger med usikkerhed og støj (upræcise sensor målinger). Et occupancy grid opdeler kortet i celler og tildeler en binær tilfældig variabel til hver celle. Værdien af disse variabler repræsenterer sandsynligheden for at den pågældende celle er farbar eller ej. Ved at måle rundt omkring robotten gentagne gange opdateres sandsynlighederne og resulterer til sidst i et kort over området.

Kapitel 5

Opsummering

Da problemet består i at kortlægge en ukendt verden, vil der blive konstrueret en robot, der har til formål at navigerer i den pågældende verden via et indlejret system. Dette vil blive implementeret på en NXT. Programmet, som kører på robotten, skal skrives i NXC.

Da det er computerdelen der skal stå for de tunge beregninger, vil robotten kommunikere med en computeren via MindSqualls. PC'en skal være tilkoblet en Microsoft Kinect og skal via color tracking med Kinectens farvekamera gøre det muligt at søge efter farver på robotten, og derved være i stand til at sende en position til robotten, når den forespørger det.

Til at bygge kortet, i takt med at robotten navigerer rundt, benyttes occupancy grid, der gør det muligt at differentiere mellem ledige og optagne celler. Opdelingen af verdenen i celler, skal gøre det muligt for robotten enten at styrke eller mindske sin tro på, om et område på kortet er fremkommeligt eller ej.

Del II

Teori

Denne del indeholder den teori, der ligger til grund for projektet. Der beskrives generel notation brugt, efterfulgt af mapping teori. Her beskrives de to sensormodeller og algoritmen der opdaterer occupancy grid. Derefter beskrives teorien bag lokalisering, som indeholder, hvordan man konstruerer en vektor mellem to farver i et billede, og følger de to farver. Til sidst er der teori omkring ruteplanlægningen, hvor måden robotten sendes rundt på i verden beskrives.

Kapitel 6

Notation og sandsynlighedsteori

I dette kapitel vil der blive introduceret den notation og sandsynlighedsteori der er nødvendig for at kunne beskrive teorien bag *occupancy grid* algoritmen.

6.1 Notation for probabilistisk robotstyring

Da det er ønskværdigt at kunne give en præcis beskrivelse af robottens positur, dvs. dens position og retning, vil dette afsnit kort introducere den nødvendige notation, som foreslået af Sebastian Thrun [1, s. 16-21].

- **Tilstand** betegner den tilstand miljøet er i; altså, robottens positur, omkringliggende objekter som vægge, bygninger osv. Tilstand kan være *dynamisk* (tilstanden kan ændre sig – fx position for en person) og *statisk* (tilstanden ændrer sig ikke – fx position for en bygning). Tilstand beskrives af variablen x , som også indeholder information omkring robotten selv, fx dens *positur*, *hastighed* og dens *sensorer*.
- **Tilstand i tiden t** betegnes af variablen x_t og beskriver den seneste *kendte* tilstand. Den forrige seneste måling angives med x_{t-1} og målingen efter den seneste som x_{t+1} .
- **Målingsdata** indeholder information om robottens omgivelser til et bestemt tidspunkt. z_t er således målingsdata til tiden t . Notationen

$$z_{t_1:t_k} = z_{t_1}, z_{t_2}, z_{t_3}, \dots, z_{t_k}$$

betegner alle målinger fra tiden t_1 til tiden t_k

- **Kontrol data** indeholder information om ændring af robottens tilstand. Kontrol data kan for eksempel være robottens hastighed, eller en aflæsning af en motors odometer, der fortæller hvor mange omdrehninger hjulet har foretaget. u_t betegner ændringen af robottens tilstand i intervallet fra $t-1$ til t . Ingen betegner notation

$$u_{t_1:t_k} = u_{t_1}, u_{t_2}, u_{t_3}, \dots, u_{t_k}$$

mængden af kontrol data fra t_1 til t_k .

6.2 Sandsynlighedsteori

Som beskrevet i kapitel 2, kan robottens viden omkring verdenen den bevæger sig i, ikke være komplet pga. måleusikkerheder. På trods af dette, er den nødt til at kunne reagere på den netop tilgængelige viden, hvorfor det er nødvendigt med metoder, der gør det muligt at resonere sig frem til et 'bedste' valg. Dette afsnit introducerer den nødvendige sandsynlighedsteori, for at kunne beskrive, hvad der gør sig gældende i en given verden, baseret på robottens observationer af netop denne.

Sandsynlighedsteori omhandler hvordan viden har indflydelse på vores opfattelse af en given verden (*belief*). I en proposition α måles vores opfattelse som en værdi mellem 0 og 1, hvor 0 betegner α som værende definitivt falsk, og 1 definitivt sand. Har α en værdi mellem 0 og 1 betegnes den således ikke til at være sand 'til en vis grad', men blot at vi ved, at den er hverken sand eller falsk. Indholdet i dette afsnit er baseret på afsnit fra [12] og [1].

6.2.1 Semantik for sandsynlighed

I dette afsnit angives den semantik, der i resten af rapporten vil blive benyttet til at beskrive sandsynligheder samt de områder indenfor sandsynlighedsteori der er relevante ift. sandsynlighedsbaseret robotteknik.

Et sandsynlighedsmål μ er en funktion fra en mængde verdener til mængden af positive reelle tal. Hvis det gælder at $\Omega_1 \cap \Omega_2 = \{\}$, hvor Ω_1 og Ω_2 er mængder af verdener, har vi at:

1. $\mu(\Omega_1 \cup \Omega_2) = \mu(\Omega_1) + \mu(\Omega_2)$
2. Hvis Ω er mængden af alle verdener vil $\mu(\Omega) = 1$

Sandsynlighedsmålet kan udvides til at dække sandsynligheden for enkelte verdener således at:

$$\mu(\omega) = \mu(\{\omega\}) \quad (6.1)$$

Vi kan nu bruge målet til at beskrive sandsynligheden for en proposition således.

$$P(\alpha) = \mu(\{\omega \mid \omega \models \alpha\}) \quad (6.2)$$

Her betyder notationen $\omega \models \alpha$ at propositionen α er sand i verdenen ω .

Notationen kan yderligere udvides til at dække *stokastiske variabler*. En sandsynlighedsfordeling $P(X)$ over variablen X , er en funktion fra domænet for X til mængden af positive reelle tal. Således at givet $x \in \text{dom}(X)$, vil $P(x)$ være sandsynligheden for at $X = x$.

Denne notation kan også benyttes til at beskrive sandsynligheder af flere variabler. Fx er $P(X, Y)$ en sandsynlighedsfordeling over variablerne X og Y , således at $P(X = x, Y = y)$, hvor $x \in \text{dom}(X)$ og $y \in \text{dom}(Y)$, har værdien $P(X = x \wedge Y = y)$, hvor $X = x \wedge Y = y$ er en proposition og P er funktionen over propositioner.

[12, s. 221-222]

6.2.2 Betinget sandsynlighed

Hvis vi har observeret noget nyt om verden, hvilket vi kalder *evidens* og betegner e , kan vi udelukke de verdener, hvor e ikke er sand, hvilket betyder, at vi kan opdatere vores sandsynlighed for hypotesen h ved at betinge dens sandsynlighed derpå. Den betingede sandsynlighed for hypotesen h givet e skrives $P(h \mid e)$.

6.2.2.1 Definition af betinget sandsynlighed

Med evidens e kan vi udelukke alle de verdener, hvor e er falsk. Dette betyder, at e introducerer et nyt mål, kaldet μ_e , hvor alle de verdener hvori e er falsk har målet 0.

1. Hvis S er en mængde af mulige verdener, hvori e er sand, kan vi definere $\mu_e(S) = c \times \mu(S)$ for en konstant c .
2. Hvis S er en mængde af verdener, der alle har e som værende falsk, defineres $\mu_e(S) = 0$.

Da vi vil have at μ_e skal være et sandsynlighedsmål, må det gælde at $\mu_e(\Omega) = 1$. Derfor gælder det at $1 = \mu_e(\Omega) = \mu_e(\{w : w \models e\}) + \mu_e(\{w : w \not\models e\}) = c \times \mu(\{w : w \models e\}) + 0 = c \times P(e)$ hvilket betyder at $c = \frac{1}{P(e)}$.

Den betingede sandsynlighed for h givet evidens e er målet μ_e på de verdener hvor h er sand, dvs.

$$\begin{aligned}
 P(h | e) &= \mu_e(\{\omega \mid \omega \models h\}) \\
 &= \mu_e(\{\omega \mid \omega \models h \wedge e\}) + \mu_e(\{\omega \mid \omega \models h \wedge \neg e\}) \\
 &= \frac{\mu(\{\omega \mid \omega \models h \wedge e\})}{P(e)} + 0 \\
 &= \frac{P(h \wedge e)}{P(e)}, \quad \text{hvor } P(e) > 0.
 \end{aligned} \tag{6.3}$$

Denne definition gør det således muligt at dekomponere en konjunktion til et produkt af betingede sandsynligheder.

[12, s. 226-227]

6.2.2.2 Kædereglen

Betingede sandsynligheder kan bruges til at dekomponere konjunktioner. For vilkårlige propositioner $\alpha_1 \dots \alpha_n$.

$$\begin{aligned}
 P(\alpha_1 \wedge \alpha_2 \wedge \dots \wedge \alpha_n) &= P(\alpha_1) \times \\
 &\quad P(\alpha_2 \mid \alpha_1) \times \\
 &\quad P(\alpha_3 \mid \alpha_1 \wedge \alpha_2) \times \\
 &\quad \vdots \\
 &\quad P(\alpha_n \mid \alpha_1 \wedge \dots \wedge \alpha_{n-1}) \times \\
 &= \prod_{i=1}^n P(\alpha_i \mid \alpha_1 \wedge \dots \wedge \alpha_{i-1}),
 \end{aligned} \tag{6.4}$$

hvor højre-siden antages til at være nul, hvis et vilkårligt produkt er nul.

[12, s. 227]

6.2.2.3 Bayes Regel

Bayes regel specificerer hvordan 'troen' på en proposition kan opdateres ud fra ny evidens. Den nye tro på hypotesen h baseret på baggrundsviden k og den nye evidens e er givet ved $P(h \mid k \wedge e)$.

Bayes regel er en direkte konsekvens af definitionen for betingede sandsynligheder og kædereglen således at.

$$\begin{aligned} P(h \wedge e \mid k) &= P(h \mid e \wedge k) \times P(e \mid k) \\ &= P(e \mid h \wedge k) \times P(h \mid k) \end{aligned} \quad (6.5)$$

$$P(h \mid e \wedge k) = \frac{P(e \mid h \wedge k) \times P(h \mid k)}{P(e \mid k)} \quad (6.6)$$

Hvilket ofte skrives med baggrundsviden implicit således.

$$P(h \mid e) = \frac{P(e \mid h) \times P(h)}{P(e)} \quad (6.7)$$

$P(h)$ og $P(e \mid h)$ kaldes henholdsvis hypotesens prior og likelihood. Bayes regel viser, at hypotesens posterior sandsynlighed er proportionel til dens likelihood multipliceret med dens prior. Hvis man skal sammenligne sandsynligheder for forskellige hypoteser, vil det være tilstrækkeligt at beregne tællerne i bayes regel. Skal man bruge den posterior sandsynlighed for hypotesen h , kan nævneren beregnes ved at summere over mængden H af hypoteser.

$$\begin{aligned} P(e \mid k) &= \sum_{h \in H} P(e \wedge h \mid k) \\ &= \sum_{h \in H} P(e \mid h \wedge k) \times P(h \mid k) \end{aligned} \quad (6.8)$$

[12, s. 229]

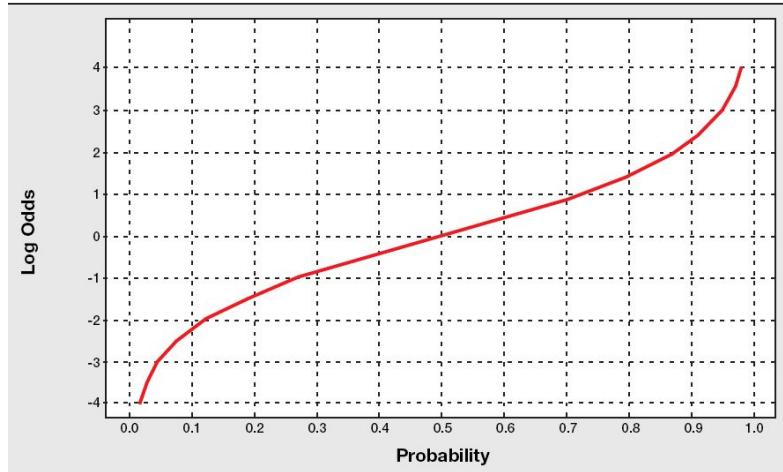
6.2.3 Binære Bayes filtre med statisk tilstand

I tilfælde hvor tilstanden er statisk, altså hvor verdenen ikke ændrer sig over tid, kan der anvendes binære Bayes filtre. Her kan en *belief* på en given tilstand x beskrives som:

$$bel_t(x) = p(x \mid z_{1:t}, u_{1:t}) = p(x \mid z_{1:t}) \quad (6.9)$$

Hvor tilstanden x er binær, dvs. enten sand (x) eller falsk ($\neg x$). Derved har vi at $bel_t(\neg x) = 1 - bel_t(x)$. Bemærk desuden at x altid er den samme, og der ikke findes en x for ethvert tidspunkt t , da verden er statisk.

[1, s. 94]



Figur 6.1: Log odds mapning.

6.2.3.1 Log Odds

Log odds er en metode der kan benyttes for at undgå, at komponenterne i Bayes Regel enten bliver definitivt sande eller falske. I så fald vil det ikke være muligt at beregne ny posterior, da de enkelte komponenter ikke vil have nogen effekt.

Der kan derfor indføres en funktion, *log odds ratio*, som mapper sandsynlighedsværdierne fra $[0; 1]$ til $[-\infty; \infty]$. Oddset for tilstand x er defineret som forholdet mellem sandsynlighederne for x og $\neg x$:

$$\frac{p(x)}{p(\neg x)} = \frac{p(x)}{1 - p(x)} \quad (6.10)$$

[1, s. 94]

Log oddset er logaritmen til forholdet mellem de to sandsynligheder: figur 6.1

$$l(x) = \log \frac{p(x)}{1 - p(x)} \quad (6.11)$$

Effekten ved at beregne forholdet mellem x og $\neg x$ er den afbildning der fåes; værdierne tilordner sig efter samme skala som kendes fra decibel skalaen til måling af lydintensitet – enhver forøgelse af 10dB giver en tifold forøgelse af den faktiske intensitet. Sagt på en anden måde, så svarer en sandsynlighed på 50% til 0dB, hvor symmetrien imellem dem afspejles ved negative værdier, fx er 99% ca. 20dB, hvor 1% svarer ca. til -20dB.

En anden umiddelbar fordel er, at forskellene bliver forstærket hvor de betyder mest. Oversætter vi værdierne 50.00% og 50.01% får vi henholdsvis 0dB og 0.0017dB, hvorimod 99.98% og 99.99% bliver til 37dB og 40dB.

[13, s. 2]

Ønsker vi at genskabe vores *belief* udfra log odds (l_x), kan vi benytte følgende ligning:

$$bel_t(x) = \frac{1}{1 + exp\{l_x\}} \quad (6.12)$$

[1, s. 95]

6.2.3.2 Udledning af opdateringsalgoritmen

Ved at benytte os af bayes regel, kan vi beskrive sandsynligheden for x udfra sensormålinger z .

$$\begin{aligned} P(x | z_{1:t}) &= \frac{P(z_t | x, z_{1:t-1})P(x | z_{1:t-1})}{P(z_t | z_{1:t-1})} \\ &= \frac{P(z_t | x)P(x | z_{1:t-1})}{P(z_t | z_{1:t-1})} \end{aligned} \quad (6.13)$$

Vi kan nu anvende Bayes regel på modellen for sensor målinger $P(z_t | x)$.

$$P(z_t | x) = \frac{P(x | z_t)P(z_t)}{P(x)} \quad (6.14)$$

Hvilket vi indsætter og får.

$$P(x | z_{1:t}) = \frac{P(x | z_t)P(z_t)P(x | z_{1:t-1})}{P(x)P(z_t | z_{1:t-1})} \quad (6.15)$$

[1, s. 95]

Vi har selvfølgelig samme type udtryk for den modsatte tilstand $\neg x$.

$$P(\neg x | z_{1:t}) = \frac{P(\neg x | z_t)P(z_t)P(\neg x | z_{1:t-1})}{P(\neg x)P(z_t | z_{1:t-1})} \quad (6.16)$$

Ved at dividere de to, kan vi eliminere nogle af de sandsynligheder som er svære at beregne.

$$\begin{aligned} \frac{P(x | z_{1:t-1})}{P(\neg x | z_{1:t-1})} &= \frac{P(x | z_t)}{P(\neg x | z_t)} \frac{P(x | z_{1:t-1})}{P(\neg x | z_{1:t-1})} \frac{P(\neg x)}{P(x)} \\ &= \frac{P(x | z_t)}{1 - P(x | z_t)} \frac{P(x | z_{1:t-1})}{1 - P(x | z_{1:t-1})} \frac{1 - P(x)}{P(x)} \end{aligned} \quad (6.17)$$

Vi beskriver log odds forholdet af vores *belief* $bel_t(x)$ som $l_t(x)$. Log oddset for vores *belief* til tiden t er givet ved at tage logaritmen til ligningen.

$$\begin{aligned} l_t(x) &= \log\left(\frac{P(x | z_t)}{1 - P(x | z_t)}\right) + \log\left(\frac{P(x | z_{1:t-1})}{1 - P(x | z_{1:t-1})}\right) + \log\left(\frac{1 - P(x)}{P(x)}\right) \\ &= \log\left(\frac{P(x | z_t)}{1 - P(x | z_t)}\right) - \log\left(\frac{P(x)}{1 - P(x)}\right) + l_{t-1} \end{aligned} \quad (6.18)$$

Her er $P(x)$ den *prior* sandsynlighed for tilstanden x , og er med i alle opdateringer af sandsynligheden for x . Den *prior* sandsynlighed er også med til at definere den oprindelige *belief* før der tages højde for *evidens* i form af sensormålinger.

$$l_0 = \log\left(\frac{P(x)}{1 - P(x)}\right) \quad (6.19)$$

[1, s. 96]

Dette giver os den endelige algoritme for det binære bayes filter.

6.2.3.3 Opdatering af celle med Bayesiansk filter

Opdaterings-algoritmen tager *log odds* for en *posterior belief* l_{t-1} for en celle, samt en ny sensor-måling z_t , hvorfra der returneres en *log odds* for en ny *belief*, l_t , for cellen. Algoritmen, skrevet i pseudo-kode, kan ses i algoritme 6.1.

```
BinaryBayesFilter( $l_{t-1}, z_t$ )
     $l_t = l_{t-1} + \log \frac{p(x|z_t)}{1-p(x|z_t)} - \log \frac{p(x)}{1-p(x)}$ 
    return  $l_t$ 
```

Algoritme 6.1: Binært Baysiansk filter algoritme på log odds form, brugt til at estimere ny posterior ud fra sensormåling.

Algoritmen tager prior (l_{t-1}), som lægges til sandsynligheden beregnet vha. log odds af sensormodellen, minus log odds for den negerede forskel af sandsynligheden for, om en celle er optaget eller ej.

[[1](#), s. 94]

Kapitel 7

Mapping

Til at kortlægge rummet robotten befinner sig i, har vi valgt at bruge *occupancy grid*. Teorien bag denne algoritme vil blive beskrevet i dette afsnit.

7.1 Overblik

Den overordnede tanke bag et occupancy grid er, at lave en ensartet inddeling af sit kort, hvor hver enkelt celle er repræsenteret af en binær *stokastisk variabel*, der fortæller om den pågældende celle er 'optaget' eller ej, hvor optaget betegnes som sandsynligheden $\mathcal{P}(\text{occupied}) = 1$. Til at begynde med initialiseres hver enkelt celle med værdien $\mathcal{P}(\text{occupied}) = 0,5$ som en indikation på, at den aktuelle tilstand endnu ikke er kendt.

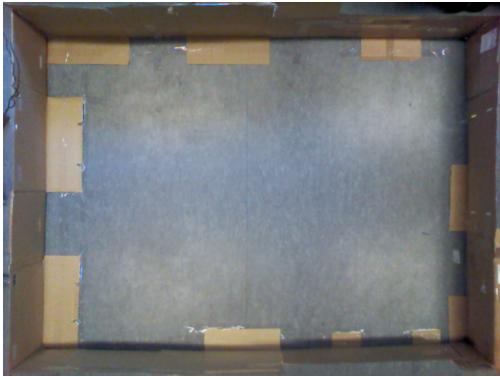
En 'ledig' celle har således værdien $\mathcal{P}(\text{occupied}) = 0$. En simpel illustration af et occupancy grid map for det kørselsmiljø, der er opstillet for vores robot, kan ses på figur 7.1.

7.2 Vertikal og horisontal sensor model

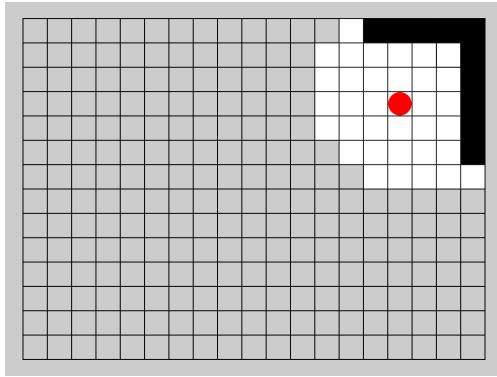
Hvis vi antager, at alle objekter i opstillingen er placeret vinkelret til områdets x og y akser, kan vi konstruere en forholdsvis simpel basal sensormodel. Først beregner vi afstanden fra robotten til den pågældende celle i vores occupancy grid.

$$r = |x_{\text{cell}} - x_{\text{robot}} + y_{\text{cell}} - y_{\text{robot}}| \quad (7.1)$$

Sensormodellen tildeler de celler, som ligger tæt på den målte afstand fra robotten z_t en højere værdi, kaldet l_{occ} , end den prior *belief* l_0 fra lighning (6.19). De celler som ligger imellem robotten og sensormålingen, tildeles en lavere værdi end l_0 , kaldet l_{free} .



(a) Aktuelt Kørselsmiljø.



(b) Eksempel på Occupancy Grid.

Figur 7.1: Illustration af et occupancy grid baseret på projekts kørselsmiljø for robotten. Sorte celler i figur 7.1b indikerer at $\mathcal{P}(\text{occupied}) = 1$, hvilket betegner væggene i kørselsmiljøet (figur 7.1a). Hvide celler indikerer at $\mathcal{P}(\text{occupied}) = 0$ og grå celler angiver ikke-udforsket område. Den røde cirkel indikerer robottens position.

for at komme frem til hvad tæt på z_t er, indfører vi en konstant α som repræsentere den gennemsnitslige tykkelse af objekter i området.

Sandsynligheden for at cellen er *occupied*, kaldet l_r , tildeles således.

$$l_r = \begin{cases} l_0 & \text{hvis } r > \min(z_{\max}, z_t + \frac{\alpha}{2}) \\ l_{\text{occ}} & \text{hvis } z_t - \frac{\alpha}{2} \leq r \leq z_t + \frac{\alpha}{2} \\ l_{\text{free}} & \text{ellers} \end{cases} \quad (7.2)$$

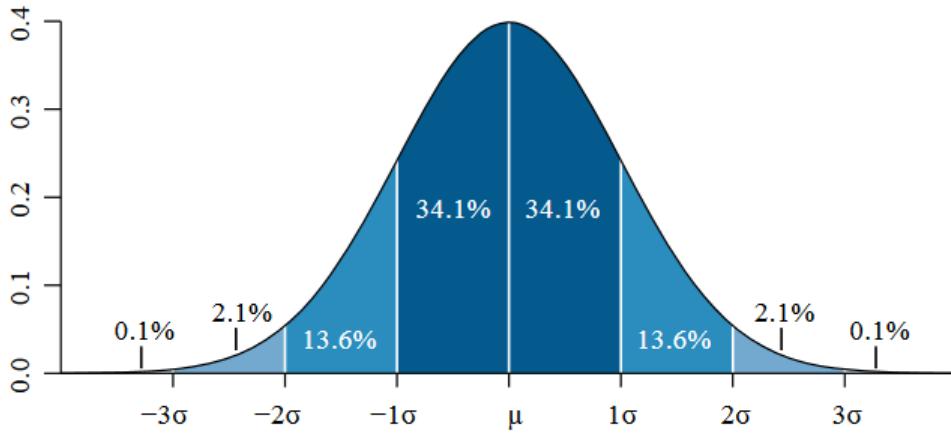
Hvor z_{\max} er sensorens maksimale måleafstand.

Modellen vil se således ud beskrevet i pseudokode:

```

inverse_sensor_model( $m_i, x_t, z_t$ )
Let  $x_i, y_i$  be the center-of-mass of  $m_i$ 
 $r = |x_i - x + y_i - y|$ 
if  $r > \min(z_{\max}, z_t + \frac{\alpha}{2})$  then
| return  $l_0$ 
else if  $z_t - \frac{\alpha}{2} \leq r \leq z_t + \frac{\alpha}{2}$  then
| return  $l_{\text{occ}}$ 
else
| return  $l_{\text{free}}$ 
```

Algoritme 7.1: Invers sensor model algoritme.



Figur 7.2: Normal fordeling $\mathcal{N}(\mu, \sigma^2)$.

7.2.1 Gaussisk sensor model

I modellen, beskrevet i det forgående afsnit, antog vi, at en celle med afstanden $\frac{\alpha}{2}$ fra robottens måling på z_t havde samme sandsynlighed som en celle med afstanden 0 fra z_t .

Hvis vi antager, at de celler, som ligger tættere på robottens måling, har en større sandsynlighed for at være *occupied*, end dem som ligger tæt på $z_t \pm \frac{\alpha}{2}$, kan vi konstruere en sensor model med en glidende overgang fra værdien l_{occ} for cellen i z_t til værdierne l_{free} og l_0 for cellerne på position $z_t \pm \frac{\alpha}{2}$.

Da summen af uafhængige fejlmålinger, ifølge *central limit theorem* vil tilnærme sig den gaussiske normalfordeling, vil det være en god approksimation for robottens måleusikkerhed.[12, p. 223]

For at finde en passende normalfordeling skal vi vælge en passende middelværdi og en passende standard afvigelse. Vi ved at centrum, dvs. middelværdien, for fordelingen skal være målingen z_t .

Udfra figur 7.2 kan vi se, at det vil være passende hvis $\frac{\alpha}{2}$ svarer til tre standard afvigelser. Dvs.:

$$3\sigma = \frac{\alpha}{2} \implies \sigma = \frac{\alpha}{6} \quad (7.3)$$

Vi kan anvende den passende normalfordeling $\mathcal{N}(z_t, (\frac{\alpha}{6})^2)$ der ses her.

$$\mathcal{N}\left(z_t, \left(\frac{\alpha}{6}\right)^2\right) = \frac{1}{\sqrt{2\pi(\frac{\alpha}{6})^2}} e^{-\frac{(x-z_t)^2}{2(\frac{\alpha}{6})^2}} \quad (7.4)$$

Vi kan beregne en sandsynlighed, for en celle i afstand r , udfra normalfordelingen ved at tage integralet af fordelingen på følgende måde.

$$P(r) = \lim_{\rho \rightarrow 0} \int_{r-\rho}^{r+\rho} \frac{1}{\sqrt{2\pi(\frac{\alpha}{6})^2}} e^{-\frac{(x-z_t)^2}{2(\frac{\alpha}{6})^2}} dx \quad (7.5)$$

Da vi vil have at sandsynlighederne mellem $z_t - \frac{\alpha}{2}$ og z_t går i en glidende overgang fra P_{free} til P_{occ} , mens vi vil have en overgang fra P_{occ} til P_0 for afstande imellem z_t og $z_t + \frac{\alpha}{2}$. Har vi valgt at lave en lineær mapning, af de sandsynligheder vi får fra normalfordelingen til de to intervaller, ved hjælp af linjens ligning. Her er P_{occ}, P_{free} og P_0 sandsynlighederne for henholdsvis l_{occ}, l_{free} og l_0 .

$$P_\kappa(r) = \begin{cases} \frac{P_{occ}-P_0}{P(z_t)-P(z_t+\frac{\alpha}{2})}(r-P(z_t)) + P_{occ} & \text{hvis } z_t < r \\ \frac{P_{free}-P_{occ}}{P(z_t-\frac{\alpha}{2})-P(z_t)}(r-P(z_t)) + P_{occ} & \text{hvis } z_t \geq r \end{cases} \quad (7.6)$$

Den nye tildeling af sandsynlighed til cellen vil nu se således ud.

$$l_r = \begin{cases} l_0 & \text{hvis } r > \min(z_{max}, z_t + \frac{\alpha}{2}) \\ \log\left(\frac{P_\kappa(r)}{1-P_\kappa(r)}\right) & \text{hvis } z_t < r \leq z_t + \frac{\alpha}{2} \\ \log\left(\frac{P_\kappa(r)}{1-P_\kappa(r)}\right) & \text{hvis } z_t - \frac{\alpha}{2} \leq r \leq z_t \\ l_{free} & \text{ellers} \end{cases} \quad (7.7)$$

Hvor l_0 er defineret i ligning (6.19) og z_{max} er sensorens maksimale måleafstand.

7.3 Occupancy Grid algoritmen

Dette afsnit vil gennemgå algoritmen, der skal opdatere et occupancy grid. Algoritmen er baseret på occupancy algoritmen fra [1, p. 286]. Formålet med *occupancy grid* algoritmen er, at beregne et nyt kort ud fra et kort og en samling data:

$$p(m | z_{1:t}, x_{1:t}) \quad (7.8)$$

hvor m er et kort, $z_{1:t}$ er mængden af målinger op til tiden t og $x_{1:t}$ er robottens rute i form af en følge af *poses*.

Kortet er inddelt i et finit antal celler. Hver celle kan findes ved m_i hvor i er cellens indeks. Hele kortet kan derfor betegnes

$$m = \sum_i m_i \quad (7.9)$$

Hver celle er tilknyttet en binær værdi der betegner hvorvidt cellen er *occupied* eller *free*. Værdien 1 betegner *occupied*, mens 0 betegner *free*.

For at begrænse mængden af celler, der skal beregnes ved en opdatering af et occupancy grid, nedbrydes problemet til en mængde delproblemer af formen

$$p(m_i | z_{1:t}, x_{1:t}) \quad (7.10)$$

Da der nu er et problem af denne type for hver celle, kan man nøjes med kun at opdatere de celler, der er ny information om. Det er således kun de celler, som er indenfor robottens synsfelt, dvs. de celler som ligger i samme række eller kolonne som robotten i vores occupancy grid, der skal opdateres.

Som i det originale binære filter (se evt. afsnit 6.2.3) benyttes log odds repræsentationen til at udtrykke om en celle er *occupied*. $l_{t,i}$ repræsenterer sandsynligheden til tiden t for at cellen m_i er *occupied* på log odds form og l_r er værdien fra vores sensor model.

$l_{t,i}$ opdateres på følgende måde.

$$l_{t,i} = \begin{cases} l_{t-1,i} + l_r - l_0 & \text{hvis } x_i = x_r \vee y_i = y_r \\ l_{t-1,i} & \text{ellers} \end{cases} \quad (7.11)$$

Hvor x_i, y_i er grid koordinatet for cellen m_i mens x_r, y_r er grid koordinatet for robotten. Hvilket giver.

$$l_{t,i} = \log \frac{p(m_i | z_{1:t}, x_{1:t})}{1 - p(m_i | z_{1:t}, x_{1:t})} \quad (7.12)$$

Sandsynligheden genfindes ud fra log odds repræsentationen ved

$$p(m_i | z_{1:t}, x_{1:t}) = 1 - \frac{1}{1 + e^{l_{t,i}}} \quad (7.13)$$

Algoritme 7.2 viser hvordan cellerne i vores occupancy grid opdateres ud fra en sensormåling. Til opdateringen bruges **inverse_sensor_model**, algoritme 7.1. Værdien l_0 er den a priori værdi for, om cellen er optaget repræsenteret som log odds ud fra ligning (6.19).

$$l_0 = \log \frac{p(m_i = 1)}{p(m_i = 0)} = \log \frac{p(m_i)}{1 - p(m_i)} \quad (7.14)$$

```

OccupancyGridMapping( $\{l_{t-1,i}\}$ ,  $x_t$ ,  $z_t$ ):
forall the cells  $m_i$  do
    | if  $x_i = x_r$  or  $y_i = y_r$  then
    |   |  $l_{t,i} = l_{t-1,i} + \text{inverse\_sensor\_model } (m_i, x_t, z_t) - l_0$ 
    | else
    |   |  $l_{t,i} = l_{t-1,i}$ 
return  $\{l_{t,i}\}$ 

```

Algoritme 7.2: Occupancy grid opdateringsalgoritmen.

Kapitel 8

Lokalisering

Til lokalisering af robotten anvendes en Kinect (se evt. afsnit 3.1.3). Denne er udstyret med et farvekamera (se evt. afsnit 3.1.2), der vil blive anvendt til at bestemme robottens placering. Ved at styre robotten med to mærkninger i klare farver, vil det på billeder fra Kinecten være muligt at lokalisere robotten.

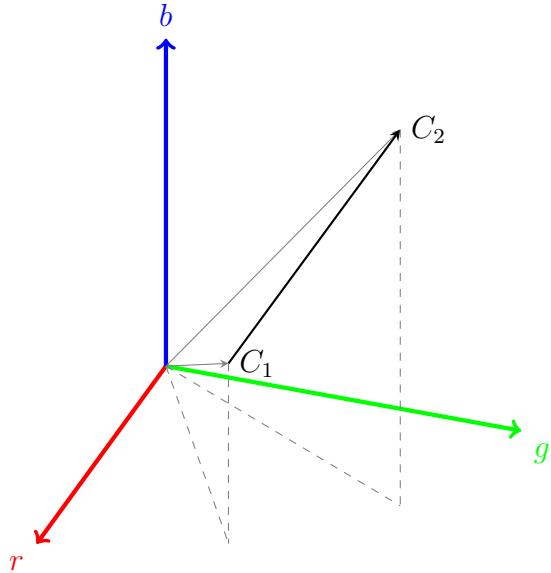
I det følgende beskrives den anvendte metode til lokalisering af robotten. Først beskrives den overordnede metode, hvorefter forbedringer af metoden præsenteres.

8.1 Farveforskel

Lokalisering af robotten foretages som udgangspunkt ud fra ét billede. Det vil sige, at systemet skal være i stand til at lokalisere robotten ud fra ét billede alene. Som beskrevet ovenfor udstyres robotten med to mærkninger i klare farver, som skal spores. Det er altså målet at finde de dele af et enkelt billede, der matcher bestemte farver.



Figur 8.1: Et billede af robotten hvori en farve skal spores.



Figur 8.2: To farver repræsenteret i et tredimensionelt rum.

I figur 8.1 vises et billede af robotten og en papkasse. Vi ønsker her at spore den blå og pink seddel, der er sat fast på robotten. I de følgende billeder tages der udgangspunkt i sporing af den pinke farve.

8.1.1 Match af farver

For at vurdere om to farver matcher hinanden, må det først gøres klart, at det ikke kan forventes at matche en farve eksakt. Til dette spiller faktorer som lys og skygge for stor en rolle. I stedet tages der udgangspunkt i en søgt farve, og ud fra denne ledes der efter alle *nære* farver. Til dette formål betragtes den farve-repræsentation der arbejdes med. Farver repræsenteres ved tre værdier; rød, grøn og blå (også kaldet RGB), der angiver alle de mulige farver for en pixel i et billede. Ved at betragte de tre farver som hver sin dimension i et tredimensionelt rum, kan farver repræsenteres som vektorer. Hermed bliver det muligt at tale om forskellen på to farver som *afstanden mellem to farver*.

På figur 8.2 ses et eksempel på afstanden mellem to farver. Her er farverne C_1 og C_2 illustreret med vektorer. Forskellen på de to farver udtrykkes ved længden af differencen mellem de to. Altså har vi, for to farver C_a og C_b , følgende afstand:

$$dist_{C_a C_b} = |C_a - C_b| \quad (8.1)$$

Hver af de tre RGB farver repræsenteres ved 1 byte og afstanden mellem to farver kan derfor højst være $\sqrt{3 \cdot 255^2} \approx 442$ (forskellen på hvid og sort).

8.1.2 Maksimal afstand

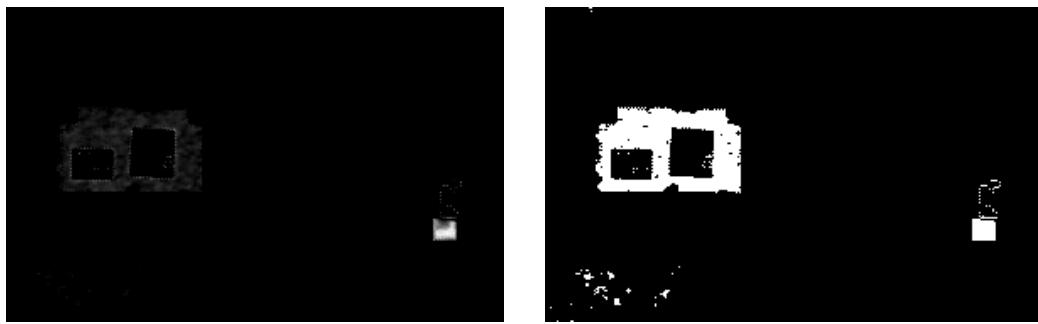
Det er dog kun interessant at kigge på de farver der ligger i en vis afstand af hinanden. Der indføres derfor en konstant ρ , der definerer et maksimum for hvor langt to farver må være fra hinanden, før de ikke længere er interessante. I afsnit 8.3 beskrives bestemmelsen af denne konstant. Med ρ i mente kan der udregnes en vægt, der bestemmer hvor interessant en farve er i forhold til en anden farve. Lad C_a og C_b være to farver, da har vi:

$$w_{C_a C_b} = \begin{cases} 0 & \text{hvis } dist_{C_a C_b} > \rho \\ 1 - \frac{dist_{C_a C_b}}{\rho} & \text{hvis } dist_{C_a C_b} \leq \rho \end{cases} \quad (8.2)$$

Her udtrykker $w_{C_a C_b}$ vægten af interesse for C_b i forhold til C_a . Det følger desuden af ovenstående, at $w_{C_a C_b} \in [0; 1]$ for ethvert par af farver.

Ved at undersøge alle pixels i et billede fra kinecten ud fra en bestemt farve, kan vægten af alle pixels bestemmes. Vægten af en pixel beskriver hvor interessant den er. Således er pixels med vægt 0 ikke interessante. De resterende pixels betegnes herefter som *interessante pixels*.

Figur 8.3a viser et eksempel på vægtning af pixels efter sporing af pink i figur 8.1. Da nogle interessante pixels har meget lav vægt er de svære at se på billedet. På figur 8.3b vises derfor alle de interessante pixels efter sporing af pink.



Figur 8.3: Billedet fra figur 8.1 hvori farven pink er blevet sporet.

Ved at finde den mindste firkant, der spænder over alle interessante pixels og udvælge dens centrum, kan farvens (og dermed robottens) *position* nu bestemmes.

8.2 Forbedringer

For at teste implementeringen af ovenstående, blev det forsøgt at lokalise re forskellige farver for varierende ρ -værdier. Af dette fremgik det tydeligt, at metoden var effektiv til genkendelse af farver. Der blev dog ved testen introduceret tre problemstillinger:

1. Dele af billedet indeholdt *farve-støj*. Ved farvestøj forstås pixels der ligger i tæt farve-afstand til den søgte farve, men langt fra det efter-søgte objekt. Dette fremgår tydeligt af figur 8.3b hvor den mindste udspændende firkant ville næsten hele billedet.
2. Farveændringer som følge af lys/skygge. I takt med at robotten bevæger sig vil lys falde forskelligt på de farvede overflader. Dette betyder at farven opfanget af kameraet ikke længere matcher den søgte farve godt. Ofte kan robotten ikke spores hvis belysningen ændres for meget.
3. Funktionens opdateringshastighed var ikke tilfredsstillende (0-3 billeder i sekundet). Omend intet krav var stillet til opdateringshastigheden, er det dog nødvendigt til enhver tid at kunne beskrive robottens lokation. Ved lave opdateringshastigheder bliver ændringerne i robottens lokation for høje (robotten flytter sig op til 30 cm mellem to billeder).

I det følgende beskrives de metoder, der er anvendt til at løse ovenstående problemstillinger. I afsnit 8.3 beskrives resultatet af indførelsen af disse metoder.

8.2.1 Filtrering af støj

Den beskrevne *farve-støj*, der opstår i visse billeder, fjernes ved at anvende en variation af et 3x3 median-filter[14]. I filteret betragtes alle værdier som binære (0 eller ikke 0). Midten af boksen erstattes af 0 hvis kun få naboer ikke har værdien 0. Variablen σ indføres her til at beskrive 'få naboer'. Lad $w_{x,y}$ beskrive vægten af den pixel i et billede, der har koordinat-sæt x, y og $N_{x,y}$ beskrive de *højst* 8 nabo-vægte til dette koordinat-sæt, da kan filteret beskrives således:

$$w'_{x,y} = \begin{cases} 0 & \text{hvis } |V_{x,y}| < \sigma \\ w_{x,y} & \text{hvis } |V_{x,y}| \geq \sigma \end{cases} \quad (8.3)$$

$$\text{hvor } V_{x,y} = \{w \in N_{x,y} \mid w > 0\}$$

Ved at anvende ovenstående filter på alle koordinat-sæt fjernes noget af den uønskede farve-støj. Filteret påføres gentagne gange, indtil ingen vægte opdateres. Herefter anses støjen som fjernet.



Figur 8.4: Billedet fra figur 8.3b efter filtrering.

Figur 8.4 vises resultatet af at anvende det beskrevne filter. Bemærk af billedet kun viser interessante pixels, og ikke deres vægte.

Det ses også på billedet, at papkassen stadig spores som værende pink. Dette problem løses indirekte, som følge af de forskellige optimeringer der anvendes. Se afsnit 8.3.1 for yderligere detaljer.

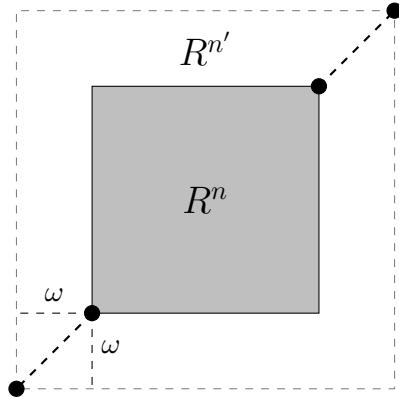
8.2.2 Farve-afstand

Som løsning på problemet med lys og skygge foretages en løbende opdatering af den farve der søges efter. Dette gøres ved, for hver opdatering, at finde den højeste vægt (efter filtrering) og dermed den farve, der er tættest på den søgte farve. Denne farve anvendes efterfølgende som den søgte farve. På denne måde opdateres den søgte farve løbende og tilpasser sig dermed de forskellige lys/skygge forhold.

8.2.3 Afgrænset område

En simpel løsning på at optimere den tid det tager at afsøge billedet, er ved at reducere problemets størrelse, altså mængden af pixels, der søges i. Som et led i lokaliseringen bestemmes den mindste firkant, der spændes over alle interessante pixels. Ved næste opdatering tages der udgangspunkt i, at robotten ikke har bevæget sig meget. Problemet kan derfor reduceres til kun at løse samme problem, men for et mindre billede. Omend robotten ikke bevæger sig meget, er det naturligvis nødvendigt at opfange den lille ændring, der måtte være. Derfor skal ovennævnte firkant udvides til at spænde over robottens opdaterede position.

På denne måde reduceres problemet, og det kan derfor løses hurtigere. Bemærk at der i denne løsningsmetode tages udgangspunkt i, at robotten



Figur 8.5: Søgning efter robotten begrænset til R^n 's ω nærmeste pixels.

bevæger sig korte afstande mellem opdateringer. Da den ønskede effekt, ved at indføre denne metode, er hurtigere opdateringer af robottens lokation, vil afstanden robotten kan bevæge sig mellem opdateringer også reduceres. Der indføres her endnu en variabel ω , der skal beskrive i hvilken grad problemet kan reduceres. Reducerer problemet ikke tilstrækkeligt, vil robotten stadig bevæge sig for store afstande i mellem opdateringer. Reducerer problemet derimod for meget, vil det reducerede problem ikke være af tilstrækkelig størrelse til at opfange ændringerne i robottens position. Herunder følger en beskrivelse af indførelsen af ω :

Lad R^n være den firkant der spænder over alle interessante vægte i løsningen af et problem (beskrevet i afsnit 8.1.2). Denne defineres ved to diagonalt modsatte punkter:

$$R^n = \left\{ \begin{pmatrix} x_1 \\ y_1 \end{pmatrix}, \begin{pmatrix} x_2 \\ y_2 \end{pmatrix} \right\} \mid x_1 \leq x_2, y_1 \leq y_2$$

Ud fra denne definition kan det reducerede problems størrelse $R^{n'}$ nu defineres:

$$R^{n'} = \left\{ \begin{pmatrix} R_{x_1}^n - \omega \\ R_{y_1}^n - \omega \end{pmatrix}, \begin{pmatrix} R_{x_2}^n + \omega \\ R_{y_2}^n + \omega \end{pmatrix} \right\} \quad (8.4)$$

Herved er det reducerede problems størrelse, beskrevet med udgangspunkt i det forrige problems løsning, som illustreret på figur 8.5.

8.3 Justering

Af de forrige afsnit fås tre variable, der søges justeret således, at de løser de beskrevne problemer tilfredsstillende. Herunder listes de tre variable, samt en beskrivelse af deres betydning og endelig justering.

Bemærk at værdierne der tildeles variablerne *ikke* er uafhængige. ændringer i ω har således betydning for opdateringshastigheden, hvilken igen har betydning for ρ . Ligeledes har ρ betydning for hvor hvilke pixels der er interessante samt i hvor høj grad der findes støj i de udregnede vægte, hvilket har betydning for σ etc.

Værdierne er derfor justeret i takt med, at de forskellige løsninger er implementeret, hvorfor de endelige værdier kan ses herunder.

Farve afstand (ρ) Som nævnt i afsnit 8.1 er den maksimale afstand mellem to farver 442 (forskellen på sort og hvid). Den minimale afstand mellem to farver er 0, hvilket er tilfældet for to ens farver. Altså skal ρ bestemmes mellem disse to. For den maksimale afstand betragtes alle pixels som interessante, mens det for den minimale kun vil være pixels med den eksakte farve, der søges efter. Da interessante pixels skal have en farve, der er *tæt på* den søgte, vil ρ have en værdi tættere på 0. Det er ved forsøg bestemt at:

$$\rho = 50$$

Hvilket angiver, at farver med afstand større end 50 *ikke* er interessante at betragte i løsningen.

Filter naboer (σ) I afsnit 8.2 blev det beskrevet at σ højst kan være 8, da en pixel har 8 naboer. Vælges $\sigma = 8$ fremgår det af ligning (8.3) at et sådant filter vil fjerne alt information i billedet. Vælges derimod $\sigma = 0$ (som er minimum værdi for σ) vil filteret ingen effekt have. Det er ved forsøg bestemt at:

$$\sigma = 4$$

Som fjerner alle vægte med færre end 4 interessante nabo-pixels.

Problem reduktion (ω) Den kraftigste reduktion af problemets størrelse, uden reduktion af R^n , vil være $\omega = 0$ (se ligning (8.4)) der angiver, at der kun skal søges i den forrige løsnings område, og altså at robotten ikke har flyttet sig. Dette betyder reelt, at det må gælde at $\omega > 0$. Der er imidlertid ingen øvre grænse for ω , omend det følger naturligt at jo kraftigere en reduktion der foretages, des hurtigere kan en opdatering ske. Det er ved forsøg bestemt at:

$$\omega = 5$$

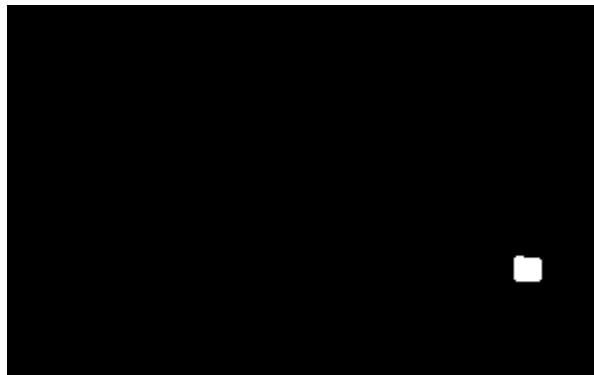
Altså vil firkanten, der indkapslede det forrige resultat udvides med 5 pixels i alle fire retninger (figur 8.5). Igennem forsøg fremgik det tydeligt at reduktionen i størrelse har en kraftig effekt på opdateringshastigheden. Derved kunne ω sættes lavt og en hurtig opdateringshastighed

kunne opnås. I afsnit 8.4 beskrives i hvilken grad farvesøgningen forbedredes som følge af $\omega = 5$.

8.3.1 Løbende optimering

Efter filtrering af støj i et billede, vil der stadig være områder som vi ikke ønsker at spore der stadig bliver sporet. På billedet i figur 8.4 ses det at sporingen stadig *genkender* papkassen som værende pink. Af figur 8.3a ses det dog, at den del af billedet har meget lav vægt.

Ved løbende at beskære billedet og ændre den sporedé farve til den med højest vægt i forrige sporing, vil støj i kanten af billedet løbende blive reduceret. Det betyder at efter få opdateringer vil papkassen ikke længere blive sporet. I figur 8.6 ses resultatet af sporing efter et antal opdateringer.



Figur 8.6: Den del af billedet der spores efter *en række* opdateringer.

8.4 Opdateringshastighed

Farverbilleder fra Kinecten kan indsamles på to forskellige måder:

- I opløsningen 640x480 ved 30 billeder i sekundet
- I opløsningen 1280x960 ved 15 billeder i sekundet

I tabellen herunder præsenteres den opdateringshastighed, der blev opnået i farvesøgning, både med og uden reduktion af problemets størrelse. Målingerne herunder er kun *omtrentlige*.

	Opdateringer pr. sekund	Bevægelse pr. opdatering
640x480 Ingen reduktion	2 – 3	5 – 10cm
640x480 Med reduktion	29 – 30	< 1cm
1280x960 Ingen reduktion	< 1	20 – 30cm
1280x960 Med reduktion	12 – 13	< 1cm

Tabel 8.1: Opdateringshastighed af farvesøgning med og uden reduktion af problemets størrelse.

8.5 Omregning fra punkt i billede til reelt punkt

Det billede, der fås fra kameraet, består af en mængde pixels. Robottens centrum, og derved lokation, vil være i præcis ét pixel. For at robotten skal kunne bruge denne til at navigere i den virkelige verden, er det nødvendigt at omregne pixel(-koordinat) til reelle koordinat.

8.5.1 Ensvinklede trekantter

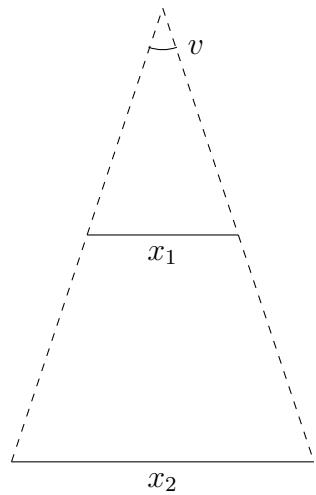
Den overordnede tanke er, at bruge reglen om ensvinklede trekanter, da det passer på situationen, som kan ses på figur 8.7. Her er v den synsvinkel som kameraet har i den givne dimension; for Kinecten er det 57° i bredden og 43° i højden. v er blot med for at vise, at der er tale om ensvinklede trekanter. x_1 er størrelsen af det billede der dannes af Kinecten. x_2 svarer til x_1 , blot den reelle størrelse af det som Kinectens billede svarer til i virkeligheden.

Sætning: Hvis alle vinkler i to trekanter er parvis lige store, er siderne parvis proportionale.

Som det kan ses på figur 8.8 er der tale om netop sådan en trekant i vores situation. Der tegnes en højde i trekanten, således at midten findes, og denne tildeles koordinatet $(0, 0)$. Derved går en dimension fra $-\frac{1}{2}x$ til $+\frac{1}{2}x$.

8.5.2 Beregning

For at finde proportionalitetsfaktoren deles en vilkårlig side af den store trekant med den tilsvarende side fra den lille trekant. Her vil proportionalitets-

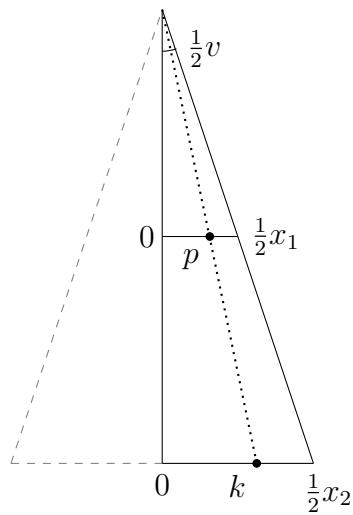


Figur 8.7: Situationen med de kendte variable.

faktoren blive $f = \frac{\frac{1}{2}x_2}{\frac{1}{2}x_1}$.

For at omregne et pixel-koordinat $p = (p_1, p_2)$ om til et reelt koordinat, skal proportionalitetsfaktoren blot påregnes. Derved får vi vores reelle koordinat $k = f \cdot p$. For vores sitation er der 2 dimensioner; længde og bredde:

$$(k_1, k_2) = (f \cdot p_1, f \cdot p_2)$$



Figur 8.8: Omregning af pixel koordinat til reelt koordinat.

Kapitel 9

Ruteplanlægning

For at automatisere kortlægningen af et rum, er det nødvendigt at automatisere ruteplanlægningen. Dette kapitel handler om den udledte metode til at planlægge en rute for robotten, så der bliver valgt den bedst mulige rute ud fra bestemte kriterier.

9.1 Overordnet beskrivelse

Den overordnede tanke er, at efter robotten har taget en scanning, skal den ud fra de celler som har høj sandsynlighed for at være ledige, finde den næste lokation, hvor der skal foretages en scanning. Ruten skal beregnes ud fra et occupancy grid, og destinationer vil derfor blive refereret til som celler. I det følgende vil der blive introduceret nogle begreber, der skal bruges til beskrivelsen af ruteplanlægningen.

9.1.1 Destinationscelle

En destinationscelle er en celle med høj sandsynlighed for at være ledig. Desuden skal alle cellens nabo-cellér ligeledes være ledige. Hvor mange celler der regnes for nabo-cellér afhænger af robottens størrelse. Denne tilgang har til formål at sikre, at der er plads til robotten på den planlagte rute. Yderligere skal der være en rute hen til destinationscellen, hvor der er nok ledige celler i hele ruten, således at robotten kan være indenfor de celler.

9.1.2 Synlig celle

De synlige celler for en destinationscelle, er de celler, som vil blive opdateret ved en scanning fra destinationscellen. Da vi opererer i en vinkelret verden,

vil synsfeltet for en given destinationscelle være de celler som ligger vandret til højre og venstre for cellen, samt de celler, som ligger lodret over og under cellen. Desuden skal også medregnes den maksimale rækkevidde for en scanning, da der ikke vil kunne opnås information for de celler, der er udenfor rækkevidden.

9.1.3 Kriterier for en god destination

Dette afhænger af den *information gain*, som robotten potentielt vil få ved at scanne i destinationen. Hvor høj *information gain* der kan opnås, bestemmes ud fra de eksisterende sandsynligheder, der findes i de celler, der i forvejen er i synsfeltet. Kort sagt er der bedst *information gain* på en celle, hvor der er flest ikke-hidtil-scannede celler i synsfeltet.

9.2 Beregning af næste målings-celle

Til bestemmelse af den næste celle, hvorfra der skal foretages en måling, gives de følgende definitioner: Lad C betegne mængden af alle de celler der udgør et occupancy grid. For alle celler $c \in C$ betegnes sandsynligheden for at cellen er optaget da som $P(c)$. Vi lader desuden $x \in C$ betegne den celle robotten starter i.

Lad funktionen $dest(d)$ udtrykke om cellen d overholder betingelserne beskrevet i afsnit 9.1.1. Da kan de mulige destinationer D bestemmes således:

$$D = \{d \in C \mid dest(d)\} \quad (9.1)$$

Endelig bestemmes også de celler der er i en celles synsfelt. Lad funktionen $visible(d, c)$ udtrykke om en celle c er i cellen d 's synsfelt (se afsnit 9.1.2). De celler der ligger i d 's synsfelt kan da bestemmes således:

$$p(d) = \{e \in C \mid visible(d, e)\} \quad (9.2)$$

9.2.1 Beregning af *information gain*

Første skridt er, at beregne værdi for alle de mulige destinationsceller:

$$v(d) = \sum_{c \in p(d)} 0.5 - |0.5 - P(c)| \quad (9.3)$$

Her er høj værdi lig med høj *information gain*. Dette skyldes, at alle celler har sandsynlighedsverdi 0.5 til at starte med, og når de opdateres kommer de enten tættere på 0 eller 1 afhængigt af, om de er ledige eller optagede.

Derved er der mere værdi i at scanne fra en celle med høj *information gain*, da dette betyder, at der er flere celler, hvor der kun er lidt information om i dens synsfelt.

Dette munder ud i en mængde, bestående af en værdi for samtlige d :

$$V = \{v(d) \mid d \in D\} \quad (9.4)$$

9.2.2 Udvælgelse af celler

Ud fra V vælges de elementer der har det højeste *information gain*.

$$Q = \{d \in D \mid v(d) \in \max V\} \quad (9.5)$$

Hvis Q kun indeholder ét element, er dette destinationscellen. Hvis der i Q er mere end én destinationscelle med samme værdi, beregnes afstanden til robotten for samtlige af de celler:

$$A = \{\text{dist}(x, q) \mid q \in Q\} \quad (9.6)$$

Destinationscellen vælges da som den celle med den korteste afstand.

9.3 Ingen mulig rute

Som det fremgår af afsnit 9.1.1 skal celler, samt deres nabo-celler, være ledige for at kunne indgå i en rute. Inden mapping-processen startes vil ingen celler kunne overholde denne egenskab. Derfor introduceres en særlig ruteplanlægning der *beregner* ruter indtil metoden beskrevet i afsnit 9.1 kan beregne en rute.

Denne særlige ruteplanlægning angiver først x (se afsnit 9.2) som destinations celle. Herefter scannes x 's nabo-celler en af gangen. Ved denne proces vil der bliver angivet en række nabo-celler som ledige hvori der kan planlægges en rute.

Skulle der ikke efter denne process kunne planlægges en rute, er robotten startet i et lukket rum hvorfra der ikke eksisterer udveje. Ovenstående sker med udgangspunkt i at robotten startes i en celle hvor der er plads til at foretage denne initierende ruteplanlægning.

Del III

Design

I denne del beskrives designet af de enkelte elementer, der indgår i projektet. Først beskrives robotten der er brugt til at udføre kortlægning, dernæst den forsøgsopstilling den er blevet brugt i, og til sidst arkitekturen som softwaren på både NXTen og PCen er opbygget omkring.

Kapitel 10

Robottens design

Dette afsnit fokuserer på designet af den robot der har til opgave at navigere rundt i et område for kortlægge det.

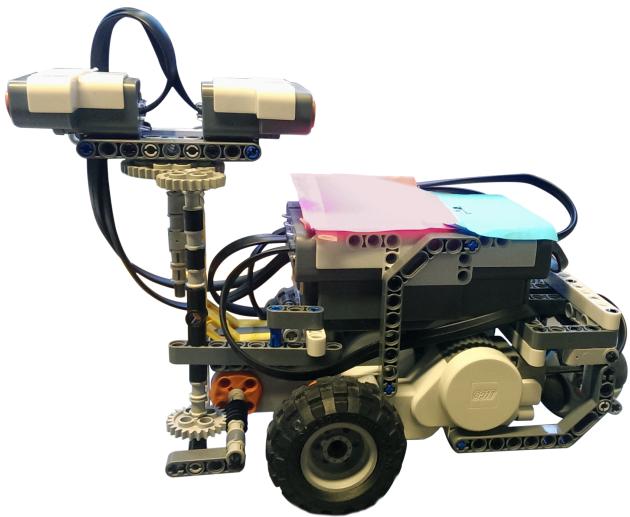
Først vil det blive beskrevet hvordan selve robotten er bygget op med LEGO, og derefter hvordan motorer og sensorer er sat på robotten.

10.1 Kroppen

Kroppen er den centrale komponent, hvor motorene, som styrer henholdsvis hjulene og rotation af sensorerne, er bygget på. Samtidig fungere den som det, der 'holder robotten sammen'. Desuden er NXTen også en central del af denne konstruktion. Placeringen af denne er primært i forhold til funktionelle behov, da der på fronten er knapper til at tænde/slukke og vælge indstillinger med. Dens placering gør det også nemt at få adgang til dens porte for tilslutning af motor, sensor, opladning samt tilslutning af PC for opdatering af software med mere.

10.2 Fremdrift

Robotten er konstrueret med et *aktivt* hjulsæt, der både giver fremdrift og styring. Denne funktionalitet opnås ved, at hvert hjul (på hver side af robotten) har sin egen motor, således der kan angives både positiv og negativ fremdrift uafhængigt af hvert hjul. Dette design gør det muligt at rottere robotten omkring sin egen akse for maksimal mobilitet – selv på et begrænset område. Foruden det forreste hjulsæt, er der bagerst på robotten monteret et 'baghjul', hvis eneste funktion er at balancere/stabilisere robotten. Valget af et hjul til at udføre en sådan funktion er forholdsvis begrænset i Lego,



Figur 10.1: Endelige design af vores robot.

hvorfor valget faldt på et 'slæbehjul' med en masse ruller, der gør det muligt for hjulet at rotere i alle retninger.

10.3 Sensorer

Den egentlige funktion af robotten er at tage afstandsmålinger til objekter indenfor sensorernes rækkevidde. Placeringen af sensorerne er derfor ikke kritisk, da de højst vil give en forskydning af konstant faktor relativ til robottens midte fra deres placering i fronten. Vigtigere er, at der er 360° udsyn når de roterer, og at robotten ikke er i vejen for målingerne, hvilket er løst ved at montere sensorerne højere end resten af robotten.

I testen af afstandssensorer (se afsnit 2.1.3) viste det sig at de to sensor typer, infrarød og ultrasonisk, havde sammenlignelig præcision, og valget mellem disse to afhænger da af behovet for rækkevidde. Den infrarøde sensor kan måle præcist tæt på, mens den ultrasoniske kan måle præcist langt væk. Til robotten blev det valgt at montere to ultrasoniske sensorer, da det ikke er lige så brugbart at vide tæt robotten er på en mur, som det er at se at der er en mur langt væk når der kortlægges.

10.4 Gearing

I afsnit 2.2 fandt vi ud af, at præcisionen på motorerne ikke var høj. Der var en afvigelse på op til 4° , hvilket gjorde det besværligt at indstille sensoren

præcist inden der foretages en måling. Dette faktum gav anledning til at udforske mulighederne for at geare motoren der styrer sensorhovedet ned for at mindske usikkerheden og øge præcisionen.

10.4.1 Simpel Teori

Gearing kan foregå på to måder; geare op eller geare ned. Det tandhjul, der er knyttet direkte til motoren kalder vi fører-tandhjulet og det tandhjul, der er knyttet til fører-tandhjulet kalder vi for følger-tandhjulet (se evt. figur 10.2).

10.4.1.1 Nedgearing

Nedgearing foregår ved at et mindre tandhjul driver et større tandhjul. Gear rationen (størrelsen af gearing) er styret af antallet af tænder på tandhjulene. For eksempel vil et 24-tands fører-tandhjul drive et 40-tands følger-tandhjul med ratioen $1 : 1.667$, hvilket betyder, at der for at give en enkelt følger-omdrejning kræves 1.667 fører-omdrejninger. Det betyder, at motoren der driver fører-tandhjulet skal rotere $\frac{40}{24} = 1\frac{2}{3}$ omgange for at rotere følger-tandhjulet én omgang og at følger-tandhjulet roterer $\frac{1}{1.667} = 0.6$ omgange pr. omdrejning af fører-tandhjulet.

Til dette projekt bliver der kun gearet ned (jævnfør robottens design på figur 10.1).



Figur 10.2: Eksempel på (ned-)gearing.

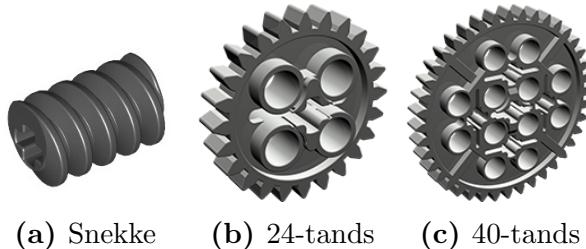
10.4.2 Gearing på ultrasonisk sensor

Dette afsnit fokuserer på den gearing, der er monteret på de ultrasoniske sensorer, som kan ses i figur 10.1*. Denne bruges til at bestemme afstanden til et objekt i en bestemt retning, hvorfor det vil være en fordel at geare motoren ned for at opnå større præcision af denne, når retningen af sensoren

*Der findes en komplet model, lavet i LEGO Digital Designer; se appendiks F.

skal bestemmes. Rotationen vil naturligvis foregå langsommere end uden gearing, men da tid ikke er en faktor på nuværende tidspunkt, er det ikke af nogen betydning for at løse problemet.

Gearingen for den ultrasoniske sensor består af i alt 3 forskellige tandhjul (4 i alt), som alle kan ses på figur 10.3.



Figur 10.3: De anvendte tandhjul til sensor rotation.

Den første kombination består af en snekke[15] (se figur 10.3a) som fører-tandhjul og 24-tands (se figur 10.3b) som følger-tandhjul. Snekken kræver en hel rotation for at flytte én tand på følger-tandhjul, hvilket giver en gear ratio på 1 : 24, som betyder, at der kræves 24 hele motor-rotationer for at rotere 24-tands (følger) tandhjulet én omgang.

Den anden kombination består af et 24-tands (se figur 10.3b) som fører-tandhjul og et 40-tands (se figur 10.3c) som følger-tandhjul, hvilket giver en ratio på 1 : 1.667.

Den samlede gear ratio for sensormotoren bliver derfor 1 : 40, som beskriver, at der for hver sensor omdrejning kræves 40 motor omdrejninger; hvilket er det samme som:

$$\frac{24}{1} \cdot \frac{40}{24} = 40$$

Kapitel 11

Forsøgsopstilling

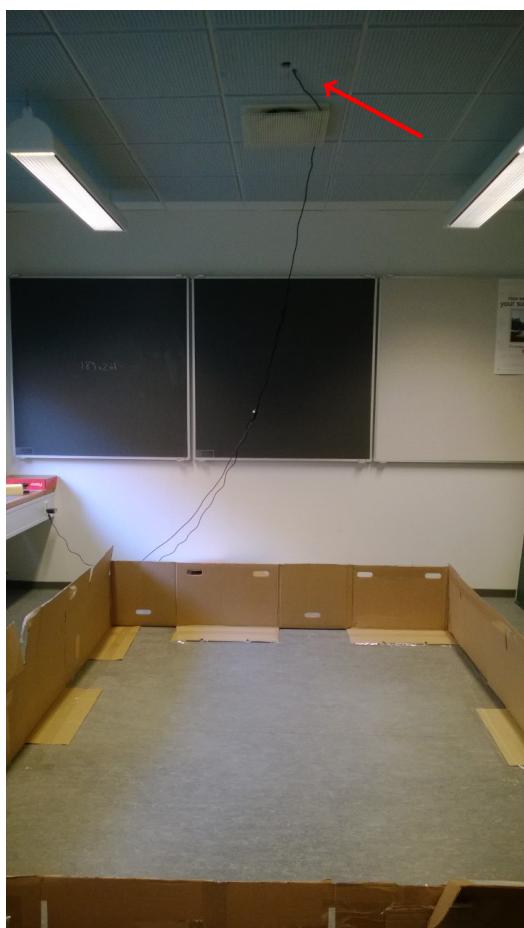
I dette afsnit præsenteres det testmiljø, der sat op til at teste robottens færdigheder.

11.1 Formål

Formålet med et testmiljø er at bedre kunne kontrollere omgivelserne. Dette giver mindre unøjagtigheder, ift. hvis det skulle testes i et nyt miljø hver gang, og gør det dermed muligt at sammenligne resultater.

11.2 Opsætning

Testmiljøet er opsat vha. papkasser og tape, som man kan se på figur 11.2. Størrelsen er $189\text{ cm} \times 261\text{ cm}$. Denne størrelse er valgt, da dette præcis passer ift. Kinectens billede, som man kan se på figur 11.3. På figur 11.1 kan man se hvordan Kinecten er monteret under en loftplade i forhold til gulvet.



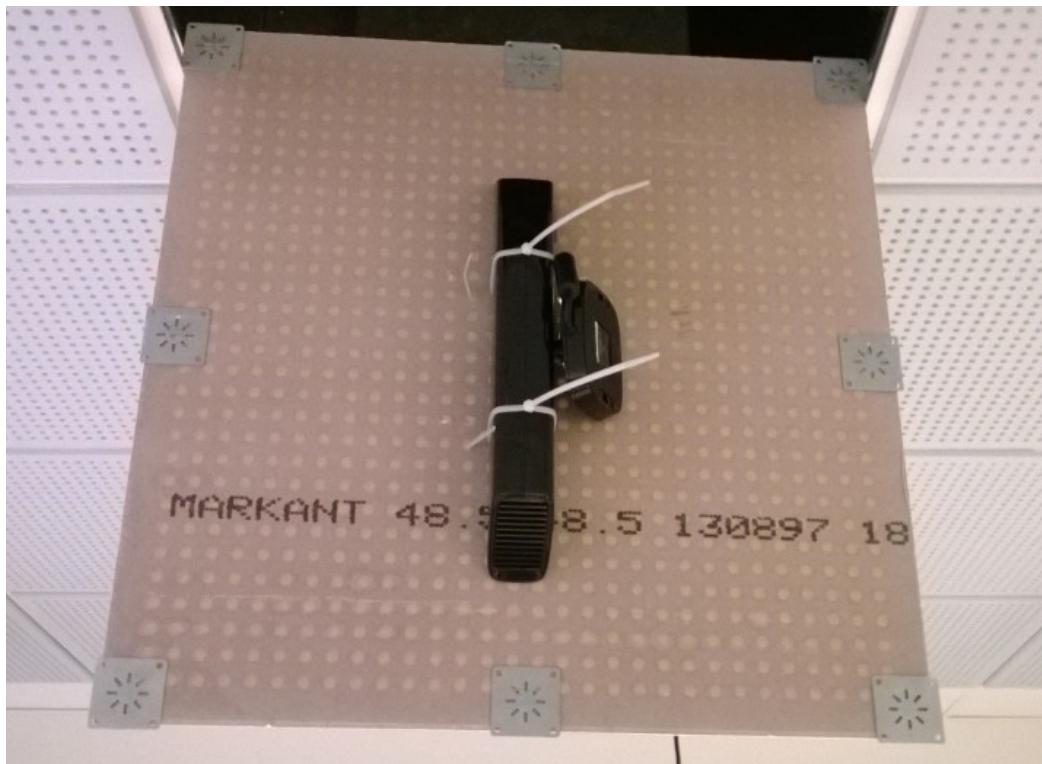
Figur 11.1: Testmiljøet set forfra. Bemærk Kinecten monteret i en loftsplade (rød pil) med hul til RGB-kameraet.



Figur 11.2: Testmiljøet sett i perspektiv.



Figur 11.3: Testmiljøet sett fra Kinecten.

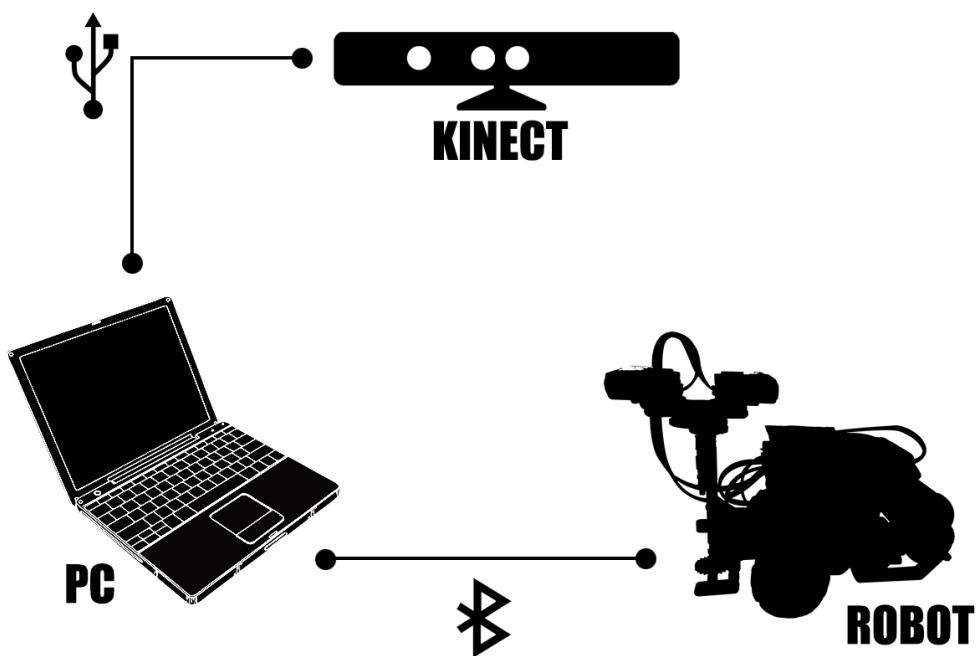


Figur 11.4: Kinecten monteret på en loftsplade.

Kapitel 12

System Arkitektur

Dette kapitel vil beskrive systemets arkitektur. Systemet er delt i to dele, én på computeren og en på NXT'en.



Figur 12.1: Systemets opbygning.

Den overordnede opbygning af systemet kan ses på figur 12.1. Selve kortlægningen sker på computeren hvor occupancy grid algoritmen køres. Computeren får sine oplysninger fra robotten gennem en bluetooth forbindelse.

Computeren bruger det grid, der er konstrueret til at fortælle robotten hvor den skal køre hen. Computeren hjælper robotten med at fortælle den hvor den er, ved at fortolke et billede fra Kinecten og sende denne oplysning til robotten, hver gang den beder om det. Når robotten er kommet frem til et punkt foretager den en sensormåling og sender denne tilbage til computeren.

I de følgende afsnit vil arkitekturen på henholdsvis NXTen og computeren blive beskrevet.

12.1 NXT Software

Koden, der kører på robotten, er bygget op omkring et modul, der hedder MotorControl. [16] MotorControl er et program, der kører på NXTen og sørger for at motorerne kører præcis den afstand, de bliver bedt om. Uden MotorControl er motorerne meget upræcise, hvilket gør det svært at navigere robotten til et bestemt sted.

Da der kun kan køre ét program ad gangen på NXTen, indeholder vores program en modifieret version af MotorControl, hvor alle navigationsfunktioner udnytter MotorControl.

Koden på robotten er opdelt i filer efter hvilket ansvar de har. I det følgende vil hver fil blive beskrevet.

12.1.1 MotorControl

For at MotorControl kan fungere, er der inkluderet følgende filer, der ikke er ændrede i forhold til kilden [16].

- **SpeedFromPosLookup.nxc**
- **MotorFunctions.nxc**
- **ControllerCore.nxc**
- **Controller.nxc**

Filen **MCTasks.nxc** indeholder de tasks, der er i den originale MotorControl, men med ændringer, som gør dem brugbare i vores program. Den indeholder desuden en **Run** metode, som sætter MotorControl igang. Denne Run funktion var i det originale MotorControl en del af et switch statement i **Main** funktionen, og er derfor trukket ud, så det er en funktion, der kan kaldes.

12.1.2 NXC kodens

Programmet samles i filen **MainTask.nxc**, som indeholder den løkke, der sættes i gang og efterfølgende køres kontinuerligt ved program-start. I denne løkke venter robotten på kommandoer fra computeren og reagerer derefter på dem.

Alt hvad robotten skal gøre håndteres i en af disse tre filer:

Navigation.nxc styrer robottens generelle navigation, fx `RunForward`, der beder robotten køre fremad, `TurnAngle`, der drejer robotten og `GoToPoint`, der beder robotten køre hen til et bestemt punkt.

Sensor.nxc bliver brugt hver gang der skal foretages en sensormåling.

Communication.nxc bliver brugt til at kommunikere med computeren med funktionerne `IAmThere`, når robotten har nået sin destination og funktionen `WhereAmI`, når robotten forespørger sin position.

12.2 PC Software

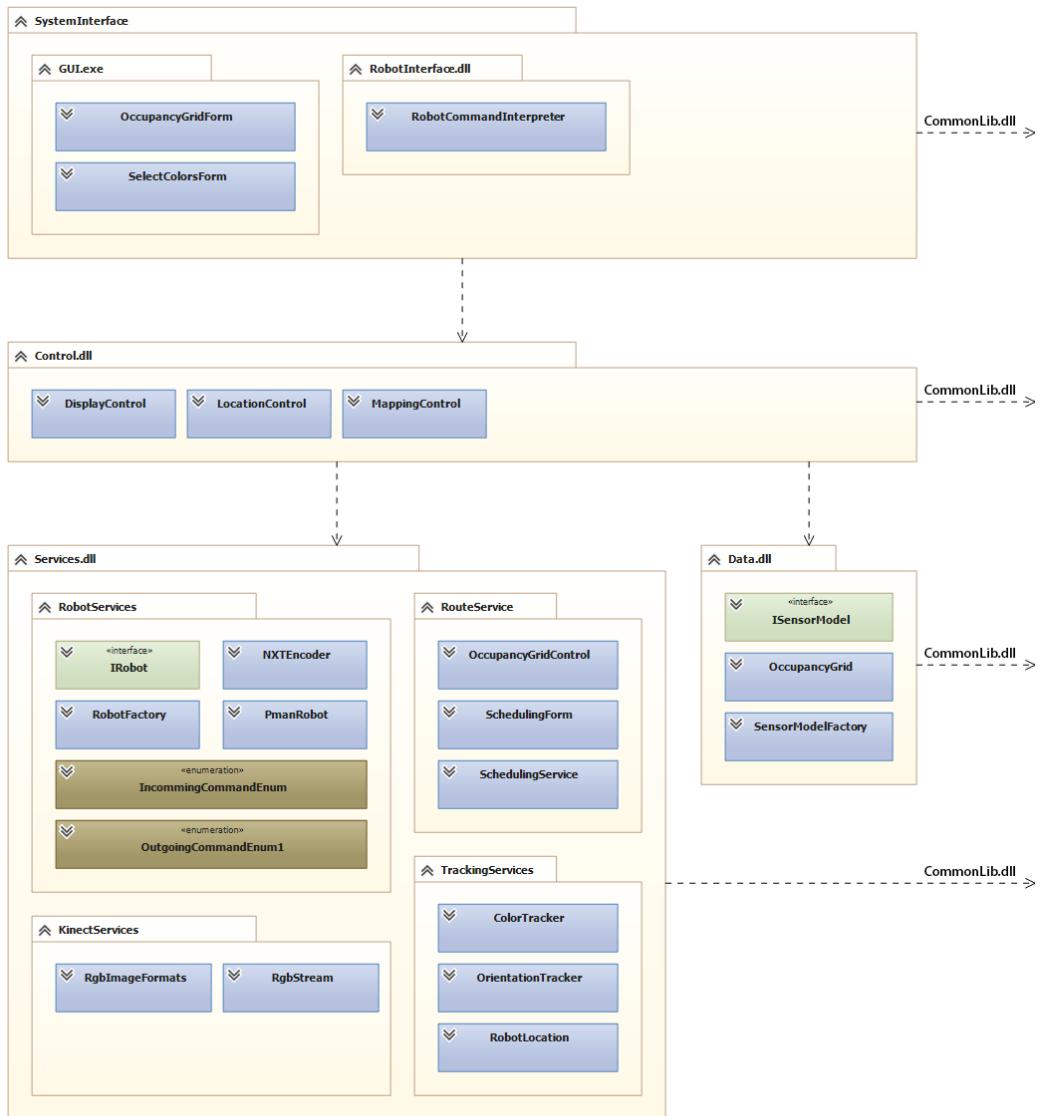
I de efterfølgende afsnit vil arkitekturen på PC (som ses på figurer [12.2](#) og [12.3](#)) blive beskrevet. Beskrivelsen vil fokusere på hvad de enkelte lag (namespaces) har til formål, samt hvordan lagene interagerer med hinanden. Som det også ses på figurene, er det kun de ydre associationer, der er angivet for at simplificere beskrivelsen af systemet. Som det første beskrives **CommonLib** i afsnit [12.2.1](#), da systemets lag alle holder referencer hertil. Efterfølgende vil lagene blive beskrevet i en rækkefølge, der følger kaldene internt i systemet.

12.2.1 CommonLib

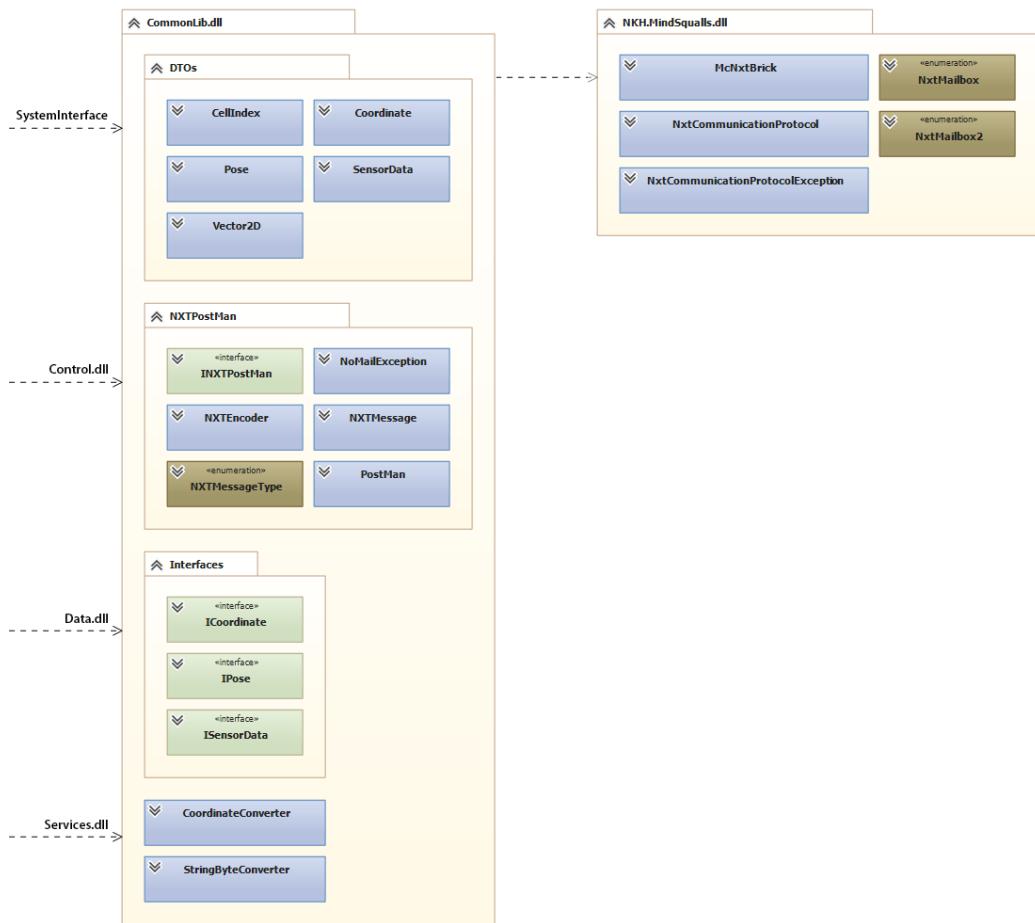
Benyttes af alle lag i arkitekturen. Dets formål er at samle de komponenter som i systemet kan benyttes af alle lag. Derfor holder det også en reference til namespacet **NKH.MindSqualls** for at kunne tilgå den del af MindSqualls (afsnit [1.3.1](#)) som indeholder kommunikationen til NXTen.

CommonLib.DTOs Dette namespace indeholder alle *Data Transferring Objects* som har til opgave at sende data mellem de forskellige lag.

CommonLib.NXTPostMan Som navnet på namespacet antyder, så er formålet med **CommonLib.NXTPostMan** at sende beskeder frem og tilbage mellem PC og robotten; hvilket gør det ansvarlig for al kommunikation imellem de to, som beskrives detaljeret i afsnit [12.3](#).



Figur 12.2: Lagene i systemarkitekturen med deres ydre associationer. Associationerne til højre fører alle til **CommonLib.dll**, som ses på figur 12.3.



Figur 12.3: Viser namespacet **CommonLib** (uden interne associationer), som er en del af den lagdelte systemarkitektur. De fleste lag i arkitekturen har associationer til netop **CommonLib**, der indeholder de elementer, som benyttes af alle lag. De refererende namespaces kan ses på figur 12.2.

CommonLib.Interfaces Definerer en række interfaces, som implementeres af de respektive DTO. Dette minimerer koblingen af koden og gør dem samtidig mere modulær, da komponenter nemt kan udskiftes.

12.2.2 NKH.MindSqualls

Dette er koden fra MindSqualls frameworkt (beskrevet i afsnit [1.3.1](#)). Det er dog ikke alle komponenter i frameworkt der benyttes; det benyttes primært til at etablere kommunikation mellem NXTen og PC samt exceptions, hvilket implementeres gennem **PostMan** klassen.

12.2.3 SystemInterface

Dette er 'indgangen' til systemet og er derfor også den komponent, som er ansvarlig for alle ydre påvirkninger; det være sig fx brugerinput eller andre systemer, som ønsker at tilgå systemet.

SystemInterface.GUI Dette namespace indeholder de grafiske komponenter for systemet, og er derfor programmet, der starter systemet. Det er ansvarlig for at modtage og reagere på brugerens input.

SystemInterface.RobotInterface Essensen i dette namespace er en tråd, der konstant tjekker efter nye beskeder på NXTen. Den benytter så henholdsvis **LocationControl** og **MappingControl** til at reagere på beskeder den finder, og anmoder dermed **Services** namespacet om at udføre den pågældende handling.

12.2.4 Control

Denne komponent i arkitekturen er ansvarlig for kontrol af robotten samt opdatering af data. Kald hertil stammer fra **SystemInterface**, der anmoder om en action, som dette namespace så reagerer på ved at benytte **Services** namespacet.

Control.DisplayControl Er en klasse som henter en videostrøm fra farvekameraet på en Microsoft Kinect.

Control.LocationControl Denne klasse opdaterer lokationen af robotten. Den indeholder således også information om robottens omgivelser og robotten selv.

Control.MappingControl Al kontrol af robotten på kortet (occupancy grid) foregår gennem denne klasse. Den indeholder således metoder til at sende robotten til et nyt koordinat, opdatere kortet og hente nye sensormålinger.

12.2.5 Services

Dette er det nederste lag i arkitekturen og benyttes af **Control** namespacet når en handling ønskes udført.

Services.RobotServices Dette namespace benyttes, når robotten skal kontrolleres. Det er således herigennem, der sendes beskeder til robotten, for at få den til at bevæge sig til en ny position og tage sensormålinger.

Services.RouteService For at få robotten til at køre en rute på kortet benyttes **RouteService** namespacet, der består af en grafisk komponent, der gør det muligt at klikke en rute ind på et occupancy grid samt en **SchedulingService**, der returnerer en kø af de punkter, som ruten består af.

Services.KinectServices Benyttes af **Control.DisplayControl** for at vise en videostrøm fra farvekameraet i Kinecten. Konverterer den returnerede videostrøm til en strøm af bitmaps, som kan vises grafisk i **SystemInterface**.

Services.TrackingServices Denne service er ansvarlig for at holde styr på robottens lokation samt positur. Robotten kan således gennem **Control** benytte dette namespace, når den ønsker en opdatering af sin positur.

12.2.6 Data

Dette assembly indeholder alle data objekter for occupancy grid samt sensormodellen. Alt data opdateres fra **Control** når robotten har taget en ny måling.

12.3 Kommunikationsprotokol

Dette afsnit beskriver kommunikationen mellem NXT og PC. Afsnittet beskriver de beskeder, der bliver sendt frem og tilbage samt hvordan de indkodes inden de sendes.

12.3.1 Besked

Beskeder vil i dette afsnit beskrives ved brug af vinklede parenteser til at indikere dele af beskeden. En besked er opbygget på følgende måde:

$$< \text{BeskedType} >< \text{Indhold} > \quad (12.1)$$

Beskedens type indkodes som strengrepræsentationen af typenummeret. Dette tal er enten en eller to bytes. Beskeder på én byte sendes fra robotten og modtages af computeren, hvor beskeder på to bytes sendes fra computeren og modtages af robotten. Beskedindholdet består af ingen eller flere bytes. I det næste afsnit vil de forskellige beskeder, der benyttes, blive beskrevet.

12.3.2 Besked typer

I tabel 12.1 ses en oversigt over alle beskedtyper. Første kolonne fortæller hvilken talkode, der benyttes til at indikere beskedens type. Anden og tredje kolonne fortæller henholdsvis om beskeden går fra computerens eller NXTens inbox eller outbox. De to sidste beskriver hvad beskeden indeholder og gør. Scenarierne, som beskederne fra tabellen indgår i samt en uddybning af indkodningen af beskedindholdet, står beskrevet nedenunder.

Beskeds type	Computer	NXT	Beskrivelse	Indhold
0	IN	OUT	Robotten forespørger om dens lokation	-
1	IN	OUT	Robotten er nået frem til dens lokation	-
2	IN	OUT	Robotten sender sensor data	$s1_x, s2_x, s1_y, s2_y$
50	OUT	IN	Computeren fortæller roboten at den skal bevæge sig til en position	x, y
51	OUT	IN	Computeren beder om at få sensor data fra robotten	-
52	OUT	IN	Computeren sender robottens position til robotten	x, y, a

Tabel 12.1: Oversigt over de forskellige beskeder.

12.3.2.1 Nuværende lokation

Denne situation forekommer, når robotten skal bede om sin lokation fra computeren. Robotten sender en besked på formen:

$<"0">$

Derefter returnerer computeren en besked på formen:

$<"52">< x >< y >< a >$

Denne besked indeholder lokationen og dens vinkel hvor x og y repræsenterer lokationen og a vinkelen. Koordinaterne fortolkes i et koordinatsystem med millimeter på akserne. Værdierne indkodes som koordinativærdierne repræsenteret som en streng på 7 tegn. Der tilføjes nuller til venstre for tallet hvis tallet er 7 tegn. Vinklen fortolkes som vinklen mellem x-aksen og en linje fra $(0,0)$ til robotten. Ligesom koordinaterne, indkodes vinklen som strengrepræsentationen af vinklen på 7 tegn.

12.3.2.2 Naviger til en ny lokation

Denne situation opstår, når computeren sender en besked der fortæller robotten, hvor den skal køre hen.

Computerens besked er på formen:

`<"50">< x >< y >`

hvor x og y er koordinaterne til det punkt robotten skal køre hen til. Koordinaterne fortolkes i et koordinatsystem med millimeter på akserne. Værdierne indkodes som koordinatværdierne repræsenteret som en streng på 7 tegn. Der tilføjes nuller til venstre for tallet hvis tallet er 7 tegn.

12.3.2.3 Nået frem til lokation

Denne situation forekommer, når robotten er nået frem til den ønskede lokation. Robotten sender da en besked på formen:

`<"1">`

12.3.2.4 Aflæs Sensordata

Når computeren beder om sensor data fra robotten, sender den en besked på formen:

`<"51">`

Robotten foretager derefter sensormålinger og returnerer dem med en besked på formen:

`<"2">< s1x >< s2x >< s1y >< s2y >`

Sensordata består af en måling fra hver sensor, før sensorerne roterer 90 grader og efter. $s1_x$ og $s2_x$ er derfor to målinger, hvor sensorerne står i startpositionen, mens $s1_y$ og $s2_y$ er to måling efter sensorerne er drejet 90 grader. Sensormålingerne sendes hver som én byte.

12.4 Flow- og Kodeeksempel

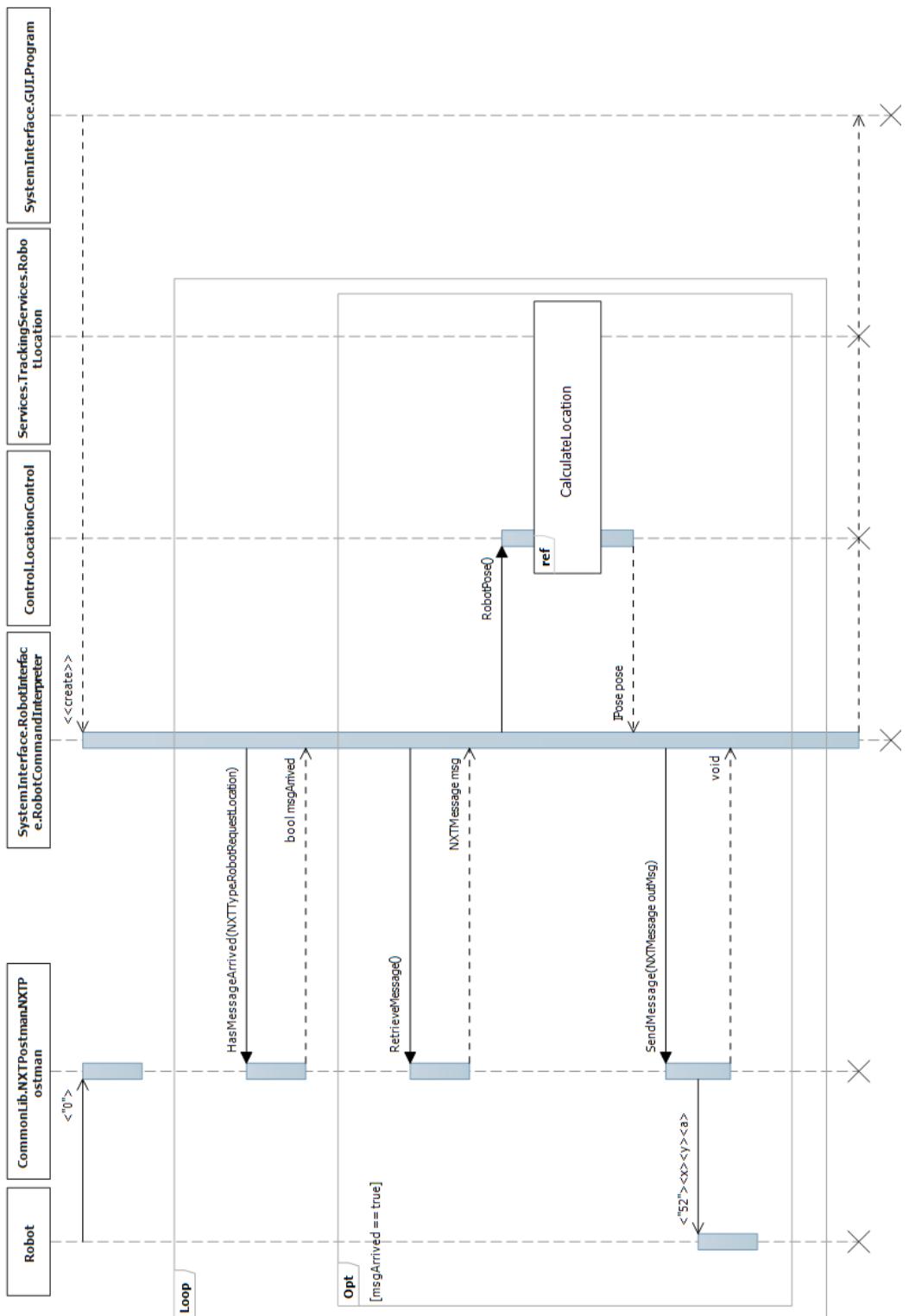
For at vise hvordan system arkitekturen (se evt. figur 12.2) benyttes, kommer der her et udvalgt eksempel, hvor det kan ses hvordan data traverserer gennem lagene i arkitekturen, ikke mindst kommunikationen mellem robot og PC.

12.4.1 Eksempel: Robotten anmoder om sin positur

Når robotten har brug for at kende dens positur, hvis den for eksempel er i gang med at navigere til et punkt, sker der kald igennem samtlige lag. Den overordnede procedure er som følgende:

1. Robotten sender anmodning om sin positur
2. `RobotInterface` afventer besked fra robotten
 - (a) Hvis der er svar, udføres noget handling
 - (b) Ellers fortsættes der med at lytte efter beskeder
3. `RobotInterface` aflæser positur i `LocationControl`
4. `LocationControl` aflæser positur i `TrackingServices`
5. `RobotInterface` bruger `Postman` til at sende positur
6. Robotten modtager svar indeholdende den ny positur

Det overordnede flow kan ses i tilstand-sekvens-diagrammet i figur 12.4. Bemærk dog at dette er en meget abstrakt beskrivelse/illustration af flowet. Det egentlige flow vil blive beskrevet nu, i højere detalje og ved brug af konkrete kodeeksempler.



Figur 12.4: Sekvens-diagram for robottens anmodning om positur.

12.4.1.1 Robotten sender anmodning om sin positur

Det første der sker er, at robotten lægger beskeden OUTGOING_WHERE_AM_I i OUTBOX:

```
1 string msg = NumToStr(OUTGOING_WHERE_AM_I);
2 SendMessage(OUTBOX, msg);
```

Kodeeksempel 12.1: Robotten sender anmodning om positur.

Herefter går robotten i gang med kontinuerligt at checke INBOX indtil beskeden med typen INCOMING_GET_POS er at finde:

```
1 while(true) {
2     ReceiveRemoteString(INBOX, true, in);
3     if (StrLen(in) > 0) {
4         cmd = SubStr(in, 0, 2);
5         cmdType = StrToNum(cmd);
6         if (cmdType == INCOMING_GET_POS) {...}
7     }
8 }
```

Kodeeksempel 12.2: Robotten venter på svar.

12.4.1.2 RobotInterface afventer besked fra robotten

Håndtering af robottens beskeder/anmodninger sker i RobotInterface. Her kører en separat tråd med funktionen `listener()`:

```
1 private void listener()
2 {
3     while (RUNNING)
4     {
5         if (checkForMessages())
6         {
7             NXTMessage msg = postman.RetrieveMessage();
8
9             switch (msg.MessageType)
10            {
11                case (NXTMessageType.RobotRequestsLocation):
12                    RobotRequestLocation();
13                    break;
14                //other cases
15            }
16        }
17        Thread.Sleep(THREAD_SLEEP_INTERVAL_IN_MILLISECONDS);
18    }
19 }
```

Kodeeksempel 12.3: listener() i RobotInterface.

Tråden med `listener()` sættes i gang når PC programmet startes.

I afsnit [12.4.1.2](#) i kodeeksempel [12.3](#) kontrolleres det, om der findes besked i `PC_INBOX` af de indgående besked-typer:

```
1 private bool checkForMessages()
2 {
3     return postman.HasMessageArrived(NXTMessageType.
4         RobotRequestsLocation)
5         || postman.HasMessageArrived(NXTMessageType.
6             RobotHasArrivedAtDestination);
7 }
```

Kodeeksempel 12.4: checkForMessages() i RobotInterface.

Hvis der findes besked, hentes denne via `RetrieveMessage()` i `NXTPostman` (se afsnit [12.4.1.2](#) i kodeeksempel [12.3](#)), som bruger Bluetooth-forbindelsen i `CommLink` til at checke efter nye beskeder på i robottens `OUTBOX`:

```
1 public NXTMessage RetrieveMessage()
2 {
3     try
4     {
5         byte[] msg = CommunicationBrick.CommLink.
6             MessageReadToBytes(PC_INBOX, NxtMailbox.Box0, true
7             );
8         return new NXTMessage(msg);
9     }
10    catch {...}
11 }
```

Kodeeksempel 12.5: RetrieveMessage i NXTPostman.

Når beskeden `NXTMessageType.RobotRequestsLocation` findes i `PC_INBOX` (se afsnit [12.4.1.2](#) i kodeeksempel [12.3](#)) udføres den pågældende metode `RobotRequestLocation()`:

```
1 private void RobotRequestLocation()
2 {
3     IPose pose = locCon.RobotPose;
4     string encodedMsg = NXTEncoder.Encode(pose);
5     NXTMessage outMsg = new NXTMessage(NXTMessageType.
6         SendPostion, encodedMsg);
7     postman.SendMessage(outMsg);
8 }
```

Kodeeksempel 12.6: RobotRequestsLocation() i RobotInterface.

12.4.1.3 RobotInterface aflæser positur i LocationControl

RobotInterface får den senest opdateret positur ved at aflæse en property på LocationControl. Dette kan ses i afsnit [12.4.1.2](#) i kodeeksempel [12.6](#).

12.4.1.4 LocationControl aflæser positur i TrackingServices

LocationControl får den senest opdateret positur ved at aflæse en property i TrackingServices. TrackingServices opdaterer sin nuværende viden om robottens positur løbende, ved kontinuerligt at beregne en ny ud fra Kinect'ens billede.

12.4.1.5 RobotInterface bruger Postman til at sende positur

RobotInterface har nu det seneste data om robottens positur. Den bruger så NXTPostman (se afsnit [12.4.1.2](#) i kodeeksempel [12.6](#)) til at lægge beskeden (først encodes beskeden jf. afsnit [12.3.2.1](#)) af type NXTMessageType. SendPosition i PC_OUTBOX, som er robottens INBOX:

```
1 public void SendMessage(NXTMessage msg)
2 {
3     string toSendMessage = String.Format("{0}{1}", (byte)msg.
4         MessageType, msg.EncodedMsg);
5     CommunicationBrick.CommLink.MessageWrite(PC_OUTBOX,
6         toSendMessage);
7 }
```

Kodeeksempel 12.7: SendMessage i NXTPostman.

12.4.1.6 Robotten modtager svar indeholdende den ny positur

Robotten, som ventede på besked ved kontinuerligt at checke sin INBOX, har nu fået en besked af typen INCOMING_GET_POS og kan derved opdatere sin interne repræsentation af dens nuværende positur, som er kroppen af if-sætningen i afsnit [12.4.1.1](#) i kodeeksempel [12.1](#) (ikke vist).

Del IV

Konklusion

Denne del er den afsluttende del af rapporten. Der vil først blive givet resultater for de udførte tests, samt evaluering af disse ud fra målsætningen. Slutteligt vil der blive konkluderet og perspektiveret over resultaterne, med forslag til fremtidige forbedringer.

Kapitel 13

Evaluering

Dette kapitel præsenterer en evaluering af projektets endelige løsning. Først evalueres der på hvad robotten rent faktisk gør, hvorefter de to valgte sensormodeller (afsnit 7.2) for at finde ud af hvilken af de to der fungerer bedst.

13.1 Observationer af robotten

Efter gentagne kørsler med de to sensormodeller, har vi gjort vigtige observationer af hvor godt systemet fungerer.

Herunder beskrives de observationer, som har indvirkning på kvaliteten af det genererede kort.

Roterer og kører ikke præcist Et generelt problem, som kan observeres for alle kørsler, er hvordan robotten til tider placerer sig forkert i forhold til, hvor den bør placere sig inden en måling. Dette giver dog ikke målinger det forkerte sted, da alle opdateringer af kortet sker efter robottens faktiske placering. Det betyder dog at det ikke altid er muligt, at foretage målinger hvor det ønskes.

Holder sig ikke altid indenfor den planlagte rute Når robotten følger den planlagte rute, skal den ideelt holde sig indenfor områder den ved der er ledige. Dog betyder flere faktorer (for eksempel cellestørrelsen og at robotten ikke kører lige), at robotten ikke altid kan holde sig indenfor ruten.

Tager ikke højde for placering af sin bagende (og forende) Dette gælder især når den drejer, og når der foretages sensormålinger. Når robotten drejer eller roterer, er der stor chance for at enten bagenden eller fronten af robotten rammer udenfor de kendte celler omkring den. Robotten bakker

tilbage for at tage højde for, at sensorerne ikke er placeret i midten ift. lokalisering, og kan også resultere i, at den kører udenfor de områder den kender til.

Sletter dele af kortet grundet forkerte sensormålinger Det er gen- tagne gange blevet observeret at dele af kortet, der ser fornuftigt ud, bliver ødelagt at gentagne fejlmålinger. Dette kan betyde at robotten ikke længere kan nå frem til de områder, som er påvirket af fejlmålingerne.

Uendelig løkke Fejlmålinger kan også føre til en *uendelig løkke*, hvilket henviser til at gentagne fejlmålinger kan 'fjerne' allerede kendte forhindringer fra kortet. Sker det, og robotten ønsker at opdatere det *nyopdagede* område (hvilket den ikke kan, da det er en fejl), så vil robotten blive ved med at forsøge at komme til det ufremkommelige område, ved at køre ind i forhindringen. Da den aldrig kommer til den nye destination, men blot rammer forhindringen, vil det fortsætte med dette sålænge der er strøm på batteriet.

Kører ind i forhindringer At robotten kan risikere at ramme forhindringer, hænger sammen med alle ovenstående beskrivelser. Dette sker således ved fejlmålinger, rotation og navigation. For at forhindre dette sker, vil det være nødvendigt enten altid at tage afstandsmålinger i kørselsretningen, eller vha. en kontakt der aktiveres ved kontakt med et objekt.

På trods af disse svagheder, opnås der stadig resultater som stemmer overens med virkeligheden; altså hvordan testmiljøet ser ud. Både banens kanter og forhindringer findes af robotten så den kan navigere rundt i området.

13.2 Formål

Formålet med denne test er at se hvilken sensormodel, der kommer frem til det mest præcise kort. Dette er for at se om målsætningen beskrevet på side 5 er opfyldt.

13.3 Vurderingsmål

Da alle tests vil returnere et helt eller delvist udfyldt occupancy grid, vil sammenligningen gå ud fra dette. Alle grids, som er resultat af en test, vil

blive sammenlignet med et optimalt occupancy grid (lavet ud fra fysisk observation). Ethvert grid (både test og optimalt) vil bestå af tre typer celler:

Optaget er en celle, hvor der er en forhindring.

Fri er en celle der er fri.

Ukendt er en celle, hvor om der ikke er opnået viden. For det optimale grid, vil det være celler inde i en større forhindring. For et test grid, kan det være det samme som for det optimale, dog kan der også forekomme ukendte celler, som skulle have været enten frie eller optagede.

13.3.1 Sammenligning

For enhver celle i et test grid, vil denne blive sammenlignet med den tilsvarende celle i det optimale grid. Der vil være i alt 9 scenarier, da enhver celle har 3 tilstande. I tabel 13.1 kan resultatet af denne sammenligning ses.

Test \ Optimalt	Optaget	Fri	Ukendt
Optaget	+1	-1	0
Fri	-1	+1	0
Ukendt	-1	-1	0

Tabel 13.1: Resultater for sammenligning af test grid celle og optimalt grid celle.

13.3.2 Resultat

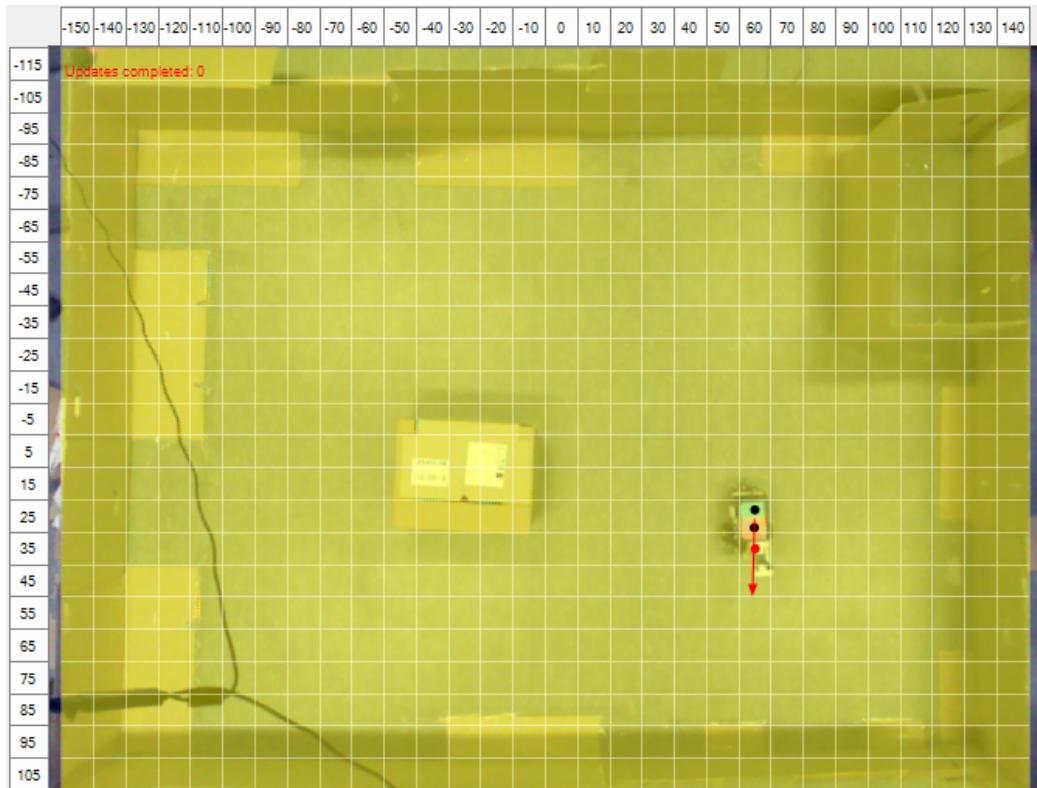
Resultatet for et enkelt test grid vil være en akkumulering af sammenligningsværdierne for cellerne. Derved kommer der et enkelt tal for enhver test, hvorfor disse efterfølgende let kan sammenlignes.

13.4 Test

Der bliver foretaget tre tests med hver sensormodel (afsnit 7.2). I alle test benyttes ruteplanlægning beskrevet i kapitel 9. Robotten kører hen til et punkt og scanner to gange - dette foretages 75 gange. Alle data bliver logget, så det er muligt at genskabe et kort udfra det.

13.4.1 Opstilling

Testmiljøet er beskrevet i kapitel 11. Selve opstillingen til testen kan ses på figur 13.1, her kan man desuden se robottens startposition, som er den samme i alle tests.



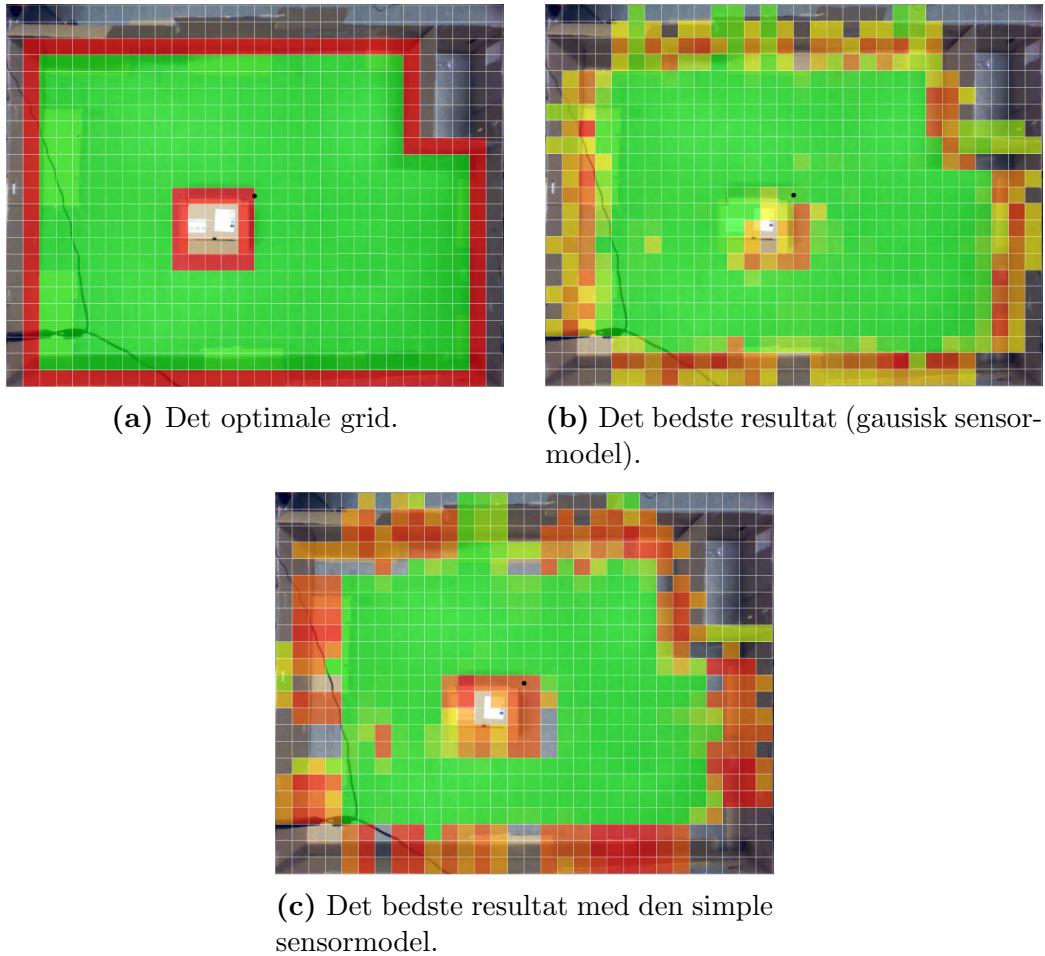
Figur 13.1: Forsøgsopstillingen inden hver test sættes i gang.

13.4.2 Resultater

Optimalt resultat

For at kunne lave en sammenligning mellem resultater, skal hvert enkelt resultat først sammenlignes med det optimale, hvorefter afvigelsen kan måles og sammenlignes.

Det optimale grid består af i alt 555 markede celler; 110 optagede celler og 445 frie celler. Det optimale grid kan ses i figur 13.2a.



Figur 13.2: Det optimale grid kontra de bedste resultater.

Bedste resultat

Det bedste resultat var for den 3. test med Gaussian Sensor Model. Resultatet var 337 korrekte celler (ud af de 555 mulige), som det kan ses i figur 13.2b. Det bedste resultat med den simple sensormodel var den 1. test. Dette kan ses på figur 13.2c. Resultatet for denne var 193 korrekte celler.

Øvrige resultater

De øvrige test-resultater kan ses i tabel 13.2. Her kan det ses at den Gaussiske sensor model giver klart det bedste resultat, og at resultaterne er nogenlunde konsistente hver især for de to modeller.

Test #	Resultat
Simpel, Test 1	193
Simpel, Test 2	187
Simpel, Test 3	193
Gaussisk, Test 1	297
Gaussisk, Test 2	325
Gaussisk, Test 3	337

Tabel 13.2: Oversigt over test-resultaterne.

Konklusion

Ud fra resultaterne kan det ses at den gaussiske model præsterer bedre end den simple sensormodel. Resultaterne for den simple model ligger omkring 190 mens resultaterne for den gaussiske liger omkring 300.

Ud fra kortene kan det dog ses at den simple finder kassen bedre end den gaussiske. Som det ses på figur 13.2b er kassens omrids kun markeret i nederste højre hjørne, mens den simple model på figur 13.2c har et større område der er markeret som optaget. I praksis vil det betyde at selvom den gaussiske giver et mere præcist resultat er der en risiko for at den vil lave et kort hvor der er frie celler hvor der burde være optaget, hvilket kan få robotten til at køre ind i en væg.

Kapitel 14

Konklusion

Vi har i vores målsætning (se side [5](#)) beskrevet, at vi vil få robotten til at konstruere et præcist kort.

I vores bedste resultat (se figur [13.2b](#)), er det frie område af kortet tydeligt defineret. Det ses desuden, at omridset af de forskellige objekter og ydervægge, er optegnet tydeligt. Det vil derfor være ligetil, at få en robot til at køre fra det ene hjørne til det andet ved udelukkende, at benytte sig af den frie del af området. Det er derfor vores vurdering, at løsningen opfylder vores målsætning tilfredsstillende.

Kapitel 15

Perspektivering

Projektets resultater kan som altid forbedres. Nogle forbedringer er bevidst ikke implementeret på grund af tidsbegrensning. Andre forbedringer er gruppen kommet frem til i løbet af projektet.

I dette afsnit præsenteres forslag til mulige forbedringer af systemet, samt en vurdering af, hvor anvendeligt det endelige produkt er.

15.1 Implementering

I dette projekt, er der blevet implementeret to forskellige sensormodeller. Udfra de test vi har udført (se tabel 13.2), kan vi se at den gaussiske sensormodel er bedre end den simple.

En mulig forbedring af den gaussiske model vil være at lave fordelingen i 2 dimensioner, således at alle de omkringliggende celler bliver opdateret. Dette giver mening, da der er en forøget sandsynlighed for, at en celle er optaget, hvis de omkringliggende celler er optaget.

Vi overvejede også at benytte en sensormodel som var baseret på vores målinger fra afsnit 2.1.1. Hvilket ville betyde at fordelingen reflektere sensorens faktiske måde at opføre sig på.

I de udførte test, er der benyttet en cellestørrelse på $10\text{cm} \times 10\text{cm}$. Denne cellestørrelse er blevet valgt med henblik på at gøre cellerne så små som muligt, uden at tiden det vil tage at kortlægge området, vil blive alt for stor. Man kan således forsøge at variere størrelsen af celler for at finde ud af, hvilken indvirkning det har på resultaterne.

Allerede i problemformuleringen blev robottens verden afgrænsset til kun at være 90 grader. Denne afgrænsning gjorde nogle aspekter af problemerne lettere. Blandt andet simplificerede det den måde sensormålinger skal behandles, når occupancy grid'et skal opdateres. En forbedring vil være at

ophæve denne begrænsning, så der kan tages målinger i 360 grader.

Ved test af systemet blev det bemærket, at der blev brugt en del tid på at robotten flyttede sig fra et punkt til et andet. I dette tidsrum blev der ikke foretaget nogle målinger og var derfor 'spildt' tid. Dette kan forbedres ved at robotten løbende tager målinger, så tiden brugt mellem to punkter bliver mere effektivt udnyttet.

15.2 Robotkonstruktion

Robotten er blevet konstrueret med to sensorer der kan dreje. Således er det muligt at tage en sensormåling forud og bagud, og derefter dreje sensortårnet til at tage sensormålinger til begge sider. Dette har vist sig at være unødvendigt, da tiden det tager at dreje tårnet gør det upraktisk. Det viste sig også at komplificere registreringen af sensormålinger, da sensortårnet ikke kunne placeres lige over det punkt, der blev brugt til at lokalisere robotten. En ændring af dette kunne være at fastmontere sensorerne og nøjes med at dreje robotten. Et andet alternativ vil være at montere 4 sensorer i stedet for to, så det slet ikke vil være nødvendigt at dreje hverken et sensortårn eller robotten for at tage målingerne.

Ved konstruktionen af systemet blev det antaget, at ruteplanlægningen skulle have ansvaret for, at robotten ikke skulle køre ind i noget. Dette gøres ud fra det occupancy grid, der indtil videre er konstrueret. Det viste sig dog til nogle af de indledende prøvekørsler, at dette ikke altid var tilstrækkeligt, og at robotten kunne finde på at køre ind i en væg eller en forhindring. Det vil derfor være nyttigt at implementere en mekanisme, der identificerer, at robotten er på vej til at køre ind i en forhindring, og i stedet stoppe robotten inden det sker.

15.3 Anvendelse

Systemet er afhængigt af, at der sidder et kamera i loftet og lokaliserer robotten. Dette begrænser anvendeligheden en del, da det i mange tilfælde ikke vil være muligt at fastmontere et kamera i loftet. Systemet kan således ikke benyttes til at kortlægge et rum der er ufremkommeligt, såsom i en krigszone eller på Mars. Det vil derimod være muligt at bruge den til at kortlægge bygninger, som robotter efterfølgende skal navigere rundt i. For at robotten skal være mere anvendelig, vil det være nødvendigt med en anden ekstern kilde til lokalisering, for eksempel GPS eller Wifi.

Del V

Bilag

I denne del findes rapportens bilag. Først findes testresultaterne for sensorerne, og derefter kommer en delmængde af sekvensdiagrammet for flowet i systemet.

Til sidst er en forklaring af hvordan det github repository hvor projektkoden er gemt i er opbygget.

Bilag A

Resultater fra test af ultrasonisk sensor

Her følger tabellen, der angiver testresultaterne fra testen, som blev lavet i forhold til den ultrasoniske sensor. Tabellen angiver først den med lineal målte afstand mellem sensor og væg. Derefter kommer de tre afmålinger, der blev udført ved hver afstand.

Målt	Aflæstning 1	Aflæstning 2	Aflæstning 3
1	6	6	6
10	22	23	23
20	22	23	23
30	31	31	32
40	40	41	40
50	50	50	51
60	60	61	61
70	70	71	66
80	80	81	81
90	90	91	91
100	100	101	102
110	111	112	112
120	120	121	121
130	131	131	131
140	141	141	141
150	151	151	151
160	161	161	163
170	171	171	171
180	255	181	181
190	255	190	191
200	202	201	202
210	255	255	213
220	255	255	222
230	255	255	232
240	255	255	255
250	255	255	255
260	255	255	255

Tabel A.1: Forsøgsresultater fra testen af den ultrasoniske sensor.

Bilag B

Test af infrarød sensor

Først angives den målte afstand mellem sensor og væg, og derefter den afmålte værdi fra sensoren.

Reel / mm	Målt / mm	Reel / mm	Målt / mm
20	183	440	457
40	73	460	471
60	70	480	500
80	79	500	527
100	98	520	523
120	117	540	551
140	136	560	583
160	163	580	610
180	180	600	639
200	198	620	639
220	218	640	666
240	241	660	666
260	255	680	689
280	280	700	742
300	306	720	742
320	326	740	734
340	351	760	528
360	375	780	792
380	395	800	792
400	425	820	846
420	440	840	853

Tabel B.1: Forsøgsresultater fra testen af den infrarøde sensor.

Bilag C

Test af motor sensor

I tabellen angives den værdi, som motorerne er sat til at dreje, hvor meget de egentlig er drejet samt hvor meget motorerne selv angiver, at de er drejet.

grader	motor grader må- lt*	motor tacho slut**	motor grader må- lt*	motor tacho slut**	optimal
1	1	0	1	0	1
2	2	2	2.5	2	2
3	2	3	3	3	3
4	5	4	4	3	4
5	5	6	4	4	5
10	10	9	9	10	10
15	11	16	14	15	15
20	20	20	18	20	20
25	21	25	23	25	25
50	57	50	56	50	50
75	80	77	80	79	75
100	100	99	95	100	100
150	150	149	145	150	150
200	204	197	200	199	200
400	400	401	400	398	400
800	799	800	800	799	800
1200	1204	1201	1200	1200	1200
1800	1796	1796	1799	1800	1800
3600	3601	3600	3597	3599	3600

Figur C.1: Forsøgsresultater for motor nøjagtighedstesten.

Bilag D

Test af kompas sensor

Her følger de tre tabeller, der angiver testresultaterne fra de tre test, som blev foretaget af kompas sensoren. I de tre tabeller, ses først hvilke kompasafmålinger der sammenlignes, sammen med den målte vinkel.

Aflæst	Målt	Aflæst	Målt	Aflæst	Målt
0-2	2°	44-60	12,5°	172-194	23°
2-4	1,5°	60-72	13,5°	194-224	30,5°
4-6	1,5°	72-84	11°	224-256	29,5°
6-8	1,5°	84-100	16°	256-282	27,5°
8-8	2°	100-108	9°	282-354	78°
8-10	1,5°	108-110	0°	354-356	2°
10-16	6°	110-116	7°	356-358	2,5°
16-26	11,5°	116-144	24,5°	358-358	1,5°
26-38	10,5°	144-158	11°	358-0	2°
38-44	8,5°	158-172	10°		

Tabel D.1: Målinger foretaget med kompas på selvstændig og stabil konstruktion.

Aflæst	Målt	Aflæst	Målt	Aflæst	Målt
0-0	4,5°	110-120	12°	296-314	20,5°
0-2	0,5°	120-146	18°	314-338	21°
2-14	13°	146-160	15,5°	338-338	6°
14-32	22°	160-188	20,5°	338-340	0°
32-42	11°	188-210	21°	340-358	17°
42-68	29°	210-234	22°	358-360	2,5°
68-74	5°	234-262	30,5°		
74-94	19°	262-288	26°		
94-110	13°	288-296	10,5°		

Tabel D.2: Målinger foretaget med kompas monteret på stang over ultralyds-sensor.

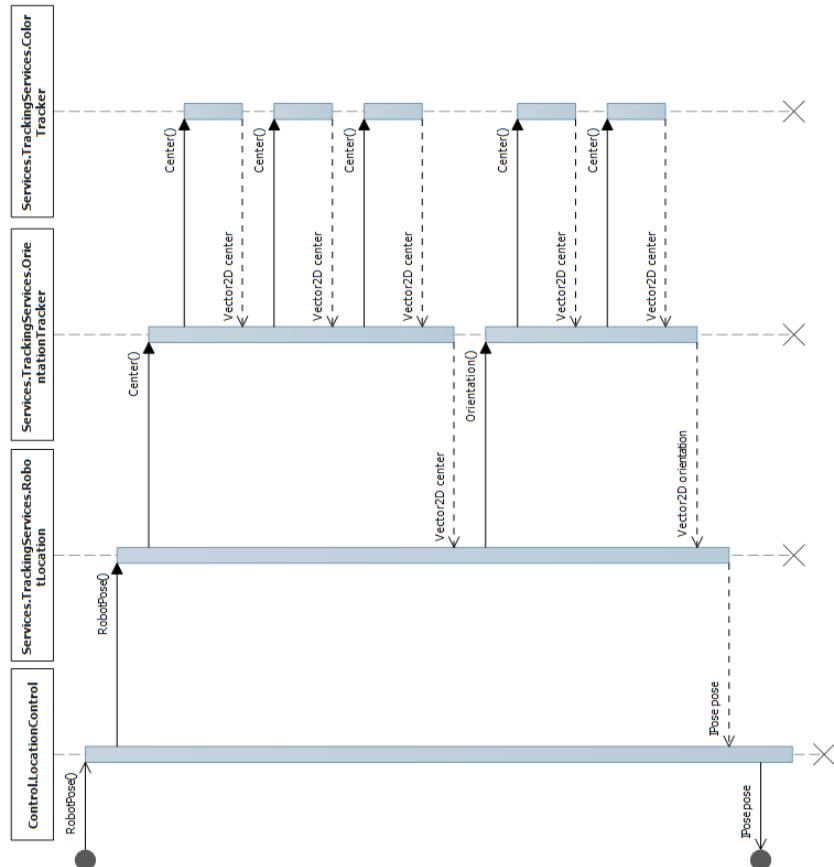
Aflæst	Målt	Aflæst	Målt	Aflæst	Målt
2-22	21°	101-139	29°	260-276	22°
22-27	6°	139-143	5°	276-305	30,5°
27-41	17°	143-161	19°	305-335	26°
41-43	2,5°	161-175	13°	335-346	10,5°
43-49	4,5°	175-190	12°	346-2	20,5°
49-50	0,5°	190-206	18°		
50-62	13°	206-221	15,5°		
62-93	22°	221-245	20,5°		
93-101	11°	245-260	21°		

Tabel D.3: Målinger foretaget med kompas på selvstændig og stabil konstruktion - efter ændring i MindSqualls.

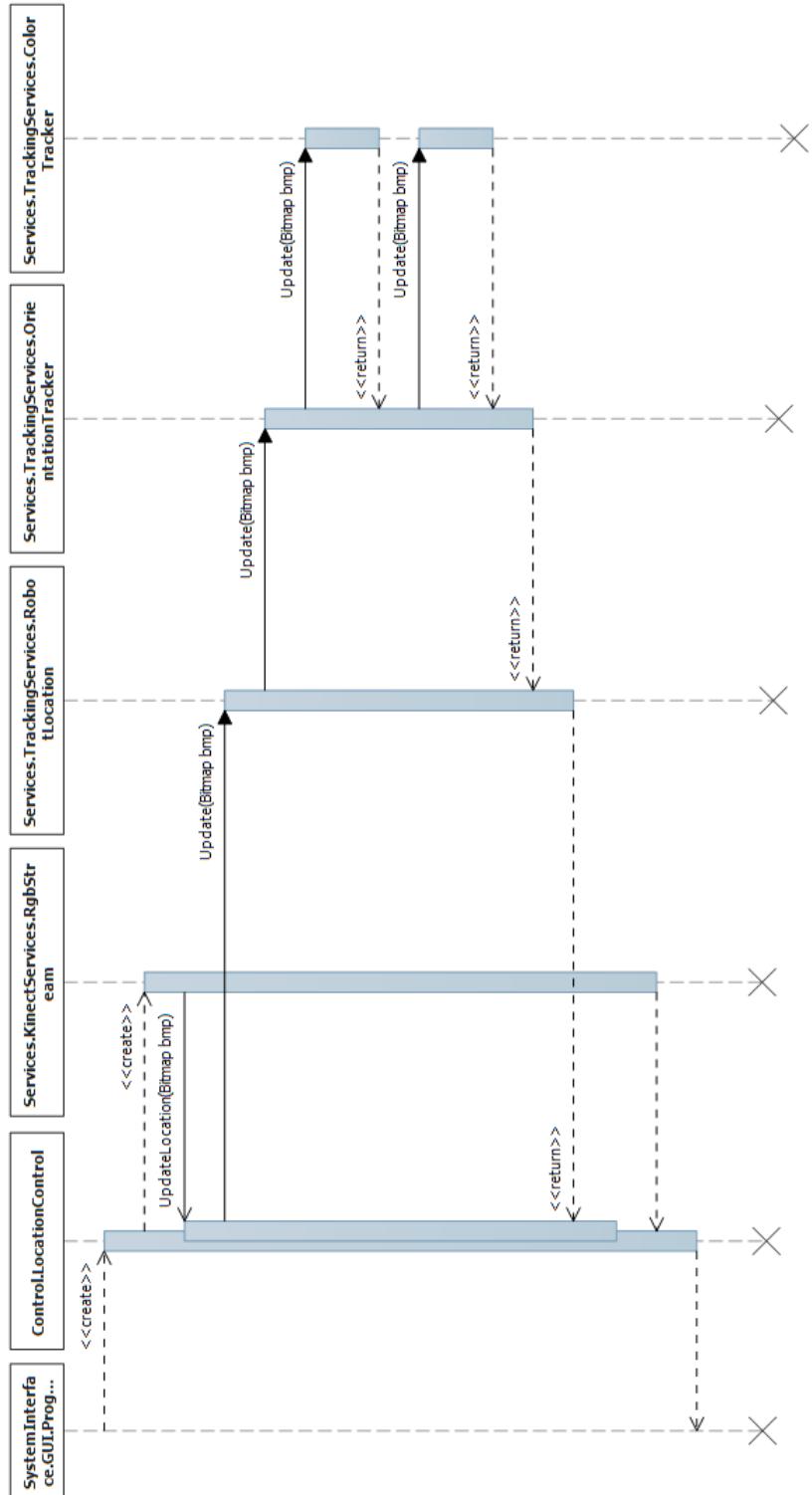
Bilag E

Sekvensdiagrammer

Nedenstående diagrammer (figurer E.1 og E.2) bliver refereret til i figur 12.4.
Hvert diagram repræsenterer således en delmængde af flowet i figur 12.4.



Figur E.1: Diagrammet viser udregningen af robottens lokation.



Figur E.2: Diagrammet viser opdatering af robottens lokation.

Bilag F

Kildekode

Herunder er der en liste over, hvad projektets GitHub repository indeholder. Projektets GitHub Repository kan findes på følgende adresse: <https://github.com/deaddog/sw505-code/tree/v1.0>.

- Software, der bliver kørt på PC ligger under mappen .\root, med undtagelse af mappen .\root\NXT.
For at starte programmet gør følgende: Åben **Master.sln** og kør **GUI**.
- Software, der ligger på NXT'en, findes i mappen .\root\NXT.
- Mindsqualls biblioteket ligger i mappen .\Mindsqualls.
- I mappen **Entropy**, er der projekter som tester de forskellige sensorer.
- Den viewer, der er brugt til at visualisere grids fra loggen, kan findes i mappen .\GridViewer.
- Filen .\lego_robot.lxf er LEGO-robot-designet, lavet i LEGO Digital Designer.

Litteratur

- [1] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic Robotics (Intelligent Robotics and Autonomous Agents)*. The MIT Press, 2005.
- [2] Ingeniøren. Robotter på vej ind i alle brancher. <http://ing.dk/artikel/robotter-pa-vej-ind-i-alle-brancher-90623>, 2008. [Online; Accessed 13-12-2013].
- [3] Sebastian Thrun. Particle filters in robotics. In *Proceedings of the Eighteenth conference on Uncertainty in artificial intelligence*, pages 511–518. Morgan Kaufmann Publishers Inc., 2002.
- [4] Lego. The nxt. <http://mindstorms.lego.com/en-us/whatisnxt/default.aspx>. [Online; Accessed 25-09-2013].
- [5] Daniele Benedettelli. Programming lego nxt robots using nxc. http://bricxcc.sourceforge.net/nbc/nxcdoc/NXC_tutorial.pdf. [Online; Accessed 04-11-2013].
- [6] BricxCC. Enhanced firmware. <http://bricxcc.sourceforge.net/firmware.html>. [Online; Accessed 11-12-2013].
- [7] Thomas Other Claudia Frischknecht. Lego mindstorms nxt - next generation. http://www.tik.ee.ethz.ch/tik/education/lectures/PPS/mindstorms/sa_nxt/download/sa-2006.18.pdf, 2006. [Online; Accessed 18-09-2013].
- [8] Hitechnic nxt compass sensor. <http://www.hitechnic.com/cgi-bin/commerce.cgi?preadd=action&key=NMC1034>. [Online; Accessed 15-10-2013].
- [9] Abhijit Jana. *Kinect for Windows SDK Programming Guide*. Packt Publishing, first edition edition, 2012.

- [10] Microsoft. Kinect for windows features. <http://www.microsoft.com/en-us/kinectforwindows/discover/features.aspx>. [Online; Accessed 20-09-2013].
- [11] Microsoft. Kinect for windows sdk 1.8 available with expanded capabilities. <http://blogs.windows.com/windows/b/extremewindows/archive/2013/09/17/kinect-for-windows-sdk-1-8-available-with-expanded-capabilities.aspx>. [Online; Accessed 23-09-2013].
- [12] David L. Poole and Alan K. Mackworth. *Artificial Intelligence: foundations of computational agents*. Cambridge University Press, 1st edition edition, 2010.
- [13] Jacob Sanders Brian Lee. Log odds. <http://dl.dropboxusercontent.com/u/34547557/log-probability.pdf>. [Online; Accessed 04-12-2013].
- [14] Median filter. http://en.wikipedia.org/wiki/Median_filter, 2013. [Online; Accessed 26-11-2013].
- [15] Den Store Danske. Snekkegear. http://www.denstoredanske.dk/It,_teknik_og_naturvidenskab/Teknologi/Makintegning,_maskinbygning_og_maskindele/snekkegear. [Online; Accessed 07-11-2013].
- [16] "http://www.mindstorms.rwth-aachen.de/. Motorcontrol source. <http://www.mindstorms.rwth-aachen.de/trac/browser/trunk/tools/MotorControl>, 2010. [Online; Accessed 18-12-2013].