Aarhus Universitet
Datalogisk Institut
Åbogade 34
8000 Århus C

Speciale

Peter Buch Jakobsen
19970450
21. juni 2004

Ad hoc netværk anvendt på autonome mobile adfærdsbaserede robotter

Forord

Dette speciale omhandler implementation af en ad hoc netværksprotokol på fysiske mobile robotter. Dette har givet visse besværligheder i forbindelse med programmering og afprøvning. I forhold til programmering af systemer, der skal køre som applikationer på for eksempel en PC, er muligheden for kontrollerede kørsler i forbindelse med afprøvning meget begrænsede. Afprøvning af programmer til LEGOs RCX består primært i at downloade et program og observere RCX'ens adfærd. Dette kan suppleres med informationer som kan skrives på RCX'ens LCD-display og feedback fra lamper tilsluttet RCX'ens outputporte, men selv med disse informationer er det ikke muligt at opnå et dækkende billede af, hvad der foregår inden i robotterne.

Disse begrænsede muligheder for at overskue programudførslen under afprøvning stiller store krav til programmeringsstilen. Det bliver meget vigtigt at undgå fejl under programmeringen samt at strukturere afprøvningen i små dele. For at understøtte disse krav til programmeringen er det nødvendigt med en høj grad af modularisering i selve programmerne således, at hvert enkelt modul kan downloades og afprøves separat fra alle andre moduler. Denne modularisering er understøttet af den *behavior*-baserede tilgang, som søges anvendt i dette speciale. Selv med en meget høj grad af modularisering, så har afprøvningsfasen taget betydlig længere tid, end man ville forvente i forbindelse med mere traditionelle applikationer.

Ud over den begrænsede mulighed for feedback under afprøvningen af programmerne, er der yderligere et sted, hvor programmeringen af robotter adskilder sig markant fra programmeringen af traditionelle applikationer. I forbindelse med programmeringen af robotter spiller robotternes og omgivelsernes fysiske karakter en stor rolle. Når en robot ikke producerer det forventede resultat, så ligger forklaringen langt fra altid i kontrolprogrammet. Ofte ligger forklaringen i at robottens fysiske komponenter ikke reagerer som forventet, eller at der er fysiske fænomener i omgivelserne, som ikke kan kontrolleres. Et Eksempel på dette er RCX'ens infrarøde kommunikation, der er meget følsom over for sollys. Hvis der er sollys i omgivelserne, så reduceres kommunikationens rækkevidde markant. Et andet eksempel er sensorernes og akutatorernes fysiske ombygning spiller en rolle, for hvordan kontrolprogrammet skal konstrueres. Ting som, hvor tit en sensor kan aflæses, hvor præcist en motor kan styres og hvad en sensor kan opfange, spiller i meget høj grad ind på robottens samlede adfærd, og skal overvejes nøje i forbindelse med udviklingen af et kontrolprogram.

Programmeringen af fysiske enheder er på disse områder markant anderledes en programmeringen af traditionelle applikationer, hvor fysiske fænomener ingen indflydelse har, og hvor udviklingen af sofistikerede værktøjer

til afprøvning gør afprøvningen simplere.

Ideen til hvad dette speciale skulle indholde er ikke kommet på en gang. Undervejs har der været flere foreskellige emner oppe at vende. Jeg havde fra kurset *Adaptive Robots* en stor interesse for programmeringen af indlejrede systemer og fysiske agenter, men det var først da jeg fulgte kurset *Advance Data Network Protocols*, at jeg opdagede mulighederne for at opnå en synergi ved at kombinere disse to interessante forskningsfelter. Der eksisterer allerede en kombination mellem disse områder for eksempel inden for multirobotsystemer - hvor robotterne ofte benytter forskellige kommunikationsprotokoller til at kommunikere - eller inden for forskningen i *sensor networks* - hvor *ad hoc* netværk ofte bliver brugt. I disse to eksempler bliver forskellige teknikker fra forskningen i netværksprotokoller inddraget i forskningen i robotter. Jeg syntes, at mange af de konkrete systemer, jeg fandt i forskellige artikler, i for høj grad blot udnyttede eksisterende kommunikationsteknologi og ikke tog højde for de specielle krav, som et netværk af autonome enheder stiller til en kommunikationsprotokol. Det blev på denne måde ideen med dette speciale at undersøge mulighederne for at udnytte at de kommunikerende enheder er fysiske, mobile og autonome.

Selvom jeg har skrevet dette speciale alene, så andre personer bidraget som sparringspartnere i forbindelse med udarbejdelsen af specialet. Ud over min vejleder Ole Caprani har dette primært drejet sig om Sune Kristensen, som jeg delt kontor med under forløbet. Sune har skrevet et speciale om *robotspil i det fysiske rum* og ligesom mig haft Ole Caprani som vejleder. Selvom Sunes og min indgangsvinkel til at benytte robotter er forskellig, har vi alligevel formået at opnå en fælles fordel ved at diskutere forskellige problemstillinger i de områder, hvor vores specialer har beskæftiget sig med de samme emner.

Indhold

1 Indledning

I 1986 revolutionerede Rodney Brooks robotverdenen med sin artikel *A Robust Layered Control System For A Mobile Robot* [Bro86]. Før dette var forskningen i robotter meget præget af forskning i kunstig intelligens, hvor begreber som *reasoning, planning og modelling* var i højsædet. [Bro86] gik helt væk fra disse begreber og afskrev dem som ubrugelige, hvis man ønsker at fremstille robotter, der kan agere i dynamiske fysiske omgivelser.

Rodney Brooks startede en bølge, som resulterede i større fokus på begreber som *situatedness, emergence og embodiment.* Robotter skulle ikke længere ræsonnere over eksterne omgivelser for derefter at lægge en plan. Nu skulle robotter i stedet anses for at være en del af omverdenen og skulle designes til at fungere i omgivelserne ved en tæt knytning mellem sensorinput og output til aktuatorer. Den udvikling, som Rodney Brooks satte igang, resulterede i begrebet *behavior based robots.* Dette begreb er gennem årene blevet benyttet til at beskrive en lang række vidt forskellige robotter, men det primære i alle tilfælde er, at *behavior based robots* består af en række uafhængige *behavior producing modules*, som ikke deler nogen central verdensmodel eller tilstand.

I de senere år er der - i takt med at trådløs kommunikation er blevet lettere tilgængelig - vokset et stort forskningsfelt frem omkring kommunikation og koordinering mellem robotter. denne forskning kan groft inddeles i to hovedfelter:

Koordinering mellem robotter: Inden for dette felt antages det at robotterne har adgang til en eller anden form for kommunikation. Dette felt er præget af problemer som *task decomposition* og *negotiation.*

Kommunikation mellem robotter: Dette felt beskæftiger sig med at stille de grundlæggende trådløse teknologier til rådighed, som gør robotter i stand til at kommunikere.

I hovedparten af forskningen i koordinering mellem robotter antages det at robotterne har adgang til et kommunikationsmedium, hvor alle kan kommunikere med alle. Eksempler på dette er [GM02], [Par98] og [WM01], som alle antager, at robotterne er forbundet med et trådløst LAN netværk. Denne antagelse er realistisk i et laboratorium, men ikke hvis robotterne skal være fordelt over større områder og alligevel skal kunne koordinere deres handlinger. Det er heller ikke nødvendigvis en fordel, at alle robotter kan modtage alle beskeder. Hvis de informationer, der kommunikeres om, kun har betydning lokalt, så vil det blot øge kompleksiteten af systemet, hvis alle kommunikerer med alle. Der findes mange eksempler på systemer, hvor det der kommunikeres om, kun har lokal interesse. For eksempel har informationer om glatføre

i LIWAS systemet kun relevans for de biler, der bevæger sig ind i området [HEK+04].

I de tilfælde, hvor et traditionelt trådløst LAN ikke er tilstrækkeligt er det oplagt at anvende teknikker som ad hoc netværk til at gøre det muligt at kommunikere ud over den enkelte robots kommunikationsrækkevidde. Eksempelvis anvender [Win00] og [GVSM02b] ad hoc netværk til kommunikation i *sensor networks*. Det er således muligt inden for robotforskningen at anvende de teknikker, som er stillet til rådighed af forskningen i ad hoc netværk. Spørgsmålet er om de erfaringer, som robotforskningen har givet indenfor autonome mobile enheder, der skal agere i dynamiske fysiske omgivelser, kan anvendes inden for forskningen i kommunikation.

Dette speciale vil klarlægge om fokus på de centrale begreber fra forskningen i *behavior based robots* kan være gavnlig i forbindelse med kommunikation mellem trådsløse mobile enheder. Der vil blive arbejdet ud fra følgende spørgsmål:

- Kan skalerbarheden af kommunikationsprotokoller i multirobotsystemer forbedres, ved at udnytte at visse informationer kun har lokal interesse?

- Kan antallet af forbindelser i et netværk af mobile enheder øges ved hjælp af teknikker fra robotforskningen?

- Kan stabiliteten af forbindelserne i et netværk af mobile enheder øges ved hjælp af teknikker fra robotforskningen?

- Kan hastigheden, hvormed data spredes til de relevante enheder i netværket øges ved hjælp af teknikker fra robotforskningen?

For at klarlægge ovenstående spørgsmål vil der dels blive fokuseret på eksisterende viden og teori, og dels vil der, hvor det er nødvendigt, blive foretaget nye eksperimenter.

Den del af specialet med fokus på eksisterende viden og teori vil søge at opnå en forståelse af de begreber, som præger forskningen indenfor henholdsvis enkelt- og multirobotsystemer samt netværk mellem trådløse mobile enheder. Ud fra dette vil det blive forsøgt at udlede nogle punkter, hvor de to forskningsfelter kan opnå en fordel ved at anvende teknikker og begreber fra hinanden.

I specialets eksperimentelle del vil der blive udført eksperimenter, som forsøger at afklare hvilke fordele, der kan opnås ved at anvende teknikker fra robotforskningen i forbindelse med trådløse mobile ad hoc netværk.

Et sted, hvor forskningen i trådløse netværk og robotforskningen har forskellige traditioner, er i forbindelse med de midler, der anvendes til at validere forskellige teknikker. Forskningen i for eksempel ad hoc netværk er i stor grad baseret på modeller og simulationer, mens robotforskningen i langt højere grad er baseret på implementation på fysiske enheder. Denne forskel giver et valg i dette speciale. Det er både muligt at anvende simulationer og implementation på fysiske robotter til at validere de teknikker, som ønskes undersøgt. Valget er faldet på det sidste. Den eksperimentelle del vil altså basere sig på implementation på konkrete fysiske robotter. Til dette er valgt LEGOs RCX, appendix A. LEGOs RCX har mulighed for at kommunikere ved hjælp af en infrarød transceiver. Denne transceiver har en kort kommunikationsradius, hvilket gør at eksperimenterne kan udføres inden for et lille område. Desuden er LEGOs RCX fleksibel, da der kan tilføjes og fjernes sensorer og aktuatorer efter behov.

Specialet vil starte med en gennemgang af forskningen inden for mobile robotter. Der vil blive lagt vægt på ideerne bag *behavior based robots* og de ændringer, det har medført indenfor robotforskningen. Efter dette kommer et afsnit, der omhandler kommunikation, hvor fokus vil være på trådløs kommunikation og ad hoc netværk. Det sidste teoretiske afsnit beskæftiger sig med multirobotsystemer, hvor der vil blive lagt speciel vægt på kommunikationens rolle i multirobotsystemer.

Der fremkommer i den teoretiske del af specialet en række forskellige konkrete ideer. Nogle af disse vil blive afprøvet i den eksperimentelle del af specialet. Der vil blive præsenteret en implementation af et ad hoc netværk på LEGOs RCX. Dette ad hoc netværk, vil være basis for eksperimenter, hvor begreber fra robotforskningen forsøges brugt, for at opnå fordele i forbindelse med kommunikationen. I afsnittet om eksperimenter er der lagt vægt på anvendelsen af *situated* kommunikation.

An agent is a physical or virtual entity

(a). which is capable of acting in an environment,

(b). which can communicate directly with other agents,

(c). which is driven by a set of tendencies (in the form of individual objectives or of a satisfaction/survival function which it tries to optimise),

(d). which possesses resources of its own,

(e). which is capable of perceiving its environment (but to a limited extent),

(f). which has only partial representation of this environment (and perhaps none at all),

(g). which possesses skills and can offer services,

(h). which may be able to reproduce itself,

(i). whose behaviour tends towards satisfying its objectives, taking account of the resources and skills available to it and depending on its perception, its representation and the communication it receives.

Figur 2.1: Definition af en agent fra [Fer99]

2 Enkeltrobotsystemer

2.1 Robotter og agenter

To centrale begreber i dette speciale er robot og agent. Der er ikke fuldstændig enighed om, hvad der definerer en agent og en robot. Dette afsnit skal forsøge at give en definition på de to begreber.

I [Fer99] defineres en agent ud fra ni punkter, som angivet i figur 2.1. Denne definition består af en liste over krav, en enhed skal opfylde for at opnå prædikatet agent. Figur 2.1 indeholder flere punkter, som ikke indgår i alle definitioner på begrebet agent. [Woo02] definerer f.eks. en agent på følgende måde.

"*An agent is a computer system that is situated in some environ-*

> *ment, and that is capable of autonomous action in this environment in order to meet its design objectives.* " [Woo02]

Definitionen i [Woo02] indeholder fire vigtige elementer: *computer system*, *situated in some environment*, *autonomous action* og *design objectives*. Af disse kan de tre sidste genfindes i definitionen i [Fer99]. Således er *autonomous action* beskrevet i figur 2.1 punkt a, *situated in some environment* er beskrevet i punkt c, e og f og *design objectives* er beskrevet i punkt c og i. Derimod kræver [Fer99] ikke, at systemet er et computer system.

En definition på en agent kan nemt blive for restriktiv, og dermed fejlagtigt udelukke systemer fra at opnå prædikatet agent.

En webcrawler, der indhenter informationer til søgetjenester på internettet, opfylder [Woo02]s definition på en agent. Den er oplagt et computer system, den befinder sig i nogle omgivelser, internettet, den er i stand til at agere i disse omgivelser i form af at hente sider over internettet og følge links, den opfatter sine omgivelser, ved at gennemsøge sider for bestemte ord og kategorisere dem derefter, den er autonom, når den først er startet, tager den selv stilling til hvilke link, den følger, den besidder mål, det primære mål er at kategorisere internetsider på nettet. Webcrawleren opfylder derimod ikke [Fer99]s definition på en agent, idet den ikke kommunikerer ikke direkte med andre agenter. Den kommunikerer med en database, hvor dens kategoriseringer bliver gemt, men en database opfylder ikke [Fer99]s definition, da en database f.eks. ikke kan sanse sine omgivelser.

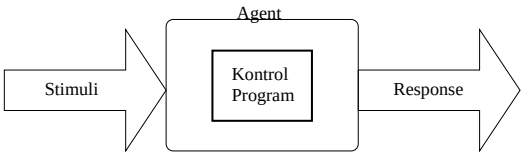
Jeg mener at definitionen af en agent i [Woo02] er både tilpas restriktiv og tilpas generel. Derfor vil jeg anvende følgende definition af en agent:

En agent: er et computersystem, der befinder sig i nogle omgivelser og er i stand til at agere selvstændigt i disse omgivelser for at opnå sine mål.

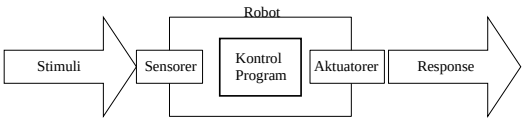
En agent vil altså bestå af en eller anden form for kontrolprogram, der tager stimuli fra omgivelserne som input og giver output i form af manipulation af omgivelserne. Kontrolprogrammet kan være vilkårligt komplekst eller vilkårligt simpelt. Det eneste krav er, at kontrolprogrammet giver agenten mulighed for at agere selvstændigt i omgivelserne.

Der er i litteraturen også en divergerende opfattelse af, hvad en robot er. Ordet robot af afledt af det tjekkiske *Robotnik*, der betyder slave eller tvunget arbejde [BÅN89]. Ordet robot blev første gang brugt af tjekken Karel Čapek i skuespillet "Rossum's Universals Robots" produceret i Prag i 1921. Ordet betød i Čapeks skuespil et mekanisk menneske, der blindt udførte ensformigt arbejde.

Den moderne betydning af ordet robot er dog noget anderledes, f.eks. definerer *Robotics Industry Association* en robot således:



Figur 2.2: Skematisk tegning af en Agent



Figur 2.3: Skematisk tegning af en robot

> "*a robot is a re-programmable, multifunctional, manipulator designed to move material, parts, tools, or specialized devices through variable programmed motions for the performance of a variety of tasks* " [Ark98]

Denne beskrivelse er særdeles præcis, men også meget restriktiv. En lang række robotter har forskellige opgaver, der ikke blot består i, at flytte ting fra ét sted til et andet. Ovenstående vil f.eks. udelukke NASAs Mars Rover [NAS03], figur 2.4. Det utal af legetøjsrobotter, som er kommet på markedet inden for de seneste år, f.eks. LEGO's RCX og Sony AIBO [Son] falder heller ikke ind under denne beskrivelse.

Robotter er med tiden blevet designet til at løse en lang række forskelligartede opgaver. Det er altså ikke løsningen af en bestemt opgave, der definerer en robot.

En definition skal være tilpas bred og ikke begrænse robotter til kun at varetage bestemte opgaver. Følgende definition er valgt:

En robot: er en agent, der opfatter sine fysiske omgivelser gennem sensorer og påvirker sine fysiske omgivelser gennem aktuatorer.

En robot er altså en agent, der er i stand til at sanse sine fysiske omgivelser gennem sensorer og manipulere de fysiske omgivelser gennem aktuatorer. Mens en agent kan være ren software, så kræver definitionen, at en robot har fysisk interaktion med omgivelserne, figur 2.3.

Denne definition af robot tager ikke udgangspunkt i de opgaver, som robotten kan udføre. Begrebet robot hæfter sig i stedet til en autonom fysisk

Figur 2.4: Mars Rover [NAS03]

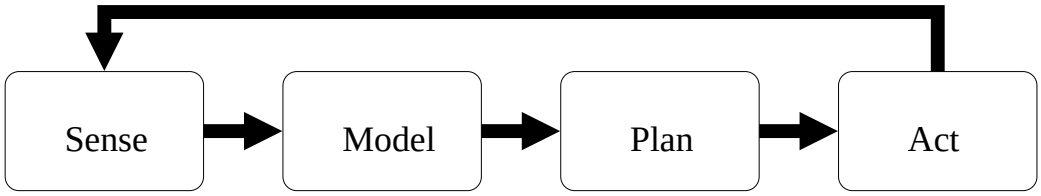
enhed, der gennem sensorer og aktuatorer interagerer med sine fysiske omgivelser for at opnå sine mål.

NASA's marsrover [NAS03], figur 2.4, og LEGO's RCX, appendix A, overholder denne definition og er dermed robotter. Derimod vil f.eks. webcrawler, også kaldet en søgerobot, ikke leve op til definitionen af en robot. En webcrawler interagerer med sine omgivelser, men omgivelserne er virtuelle og ikke fysiske. Det samme gælder en chatbot - en computerstyret deltager i online chatrooms. En chatbot interagerer med sine omgivelser og handler autonomt, men omgivelserne er igen virtuelle og ikke fysiske. Fjernstyrede enheder, som for eksempel LEGO Spybot [Tec02], er heller ikke robotter. De opfylder ikke definitionen på en agent, og dermed heller ikke definitionen på en robot, da de ikke handler selvstændigt, men bliver styret udefra. Hvis en LEGO spybot ikke bliver styret med en fjernbetjening, men derimod bliver styret af et internt kontrolprogram, opfylder den derimod definitionen på en robot, da den i dette tilfælde er i stand til at handle autonomt.

En robot er således ikke defineret ud fra den ydre fysik eller interne teknik. Derimod er det kombinationen af en agent og en fysisk virkelighed, der definerer en robot.

2.2 Artificial Intelligence

Artificial Intelligence, AI, som selvstændigt forskningsområde er opstået tidligt i datalogiens historie. Målet med AI var fra starten at udvikle maskiner,



Figur 2.5: Sense-Model-Plan-Act cyklus i traditionel AI

der kan løse opgaver ligeså godt som mennesker eller endda bedre [Nil98].

Mennesker benytter deres intelligens til at løse komplekse opgaver, og skulle maskinerne kunne løse sådanne opgaver, skulle de også udvikles til at blive intelligente. En præcis definition på begrebet intelligens kræver en længere filosofisk diskussion, der ligger uden for rammerene af dette speciale. Derfor opererer dette speciale ikke med en præcis definition af begrebet intelligens, men i stedet anvendes følgende vage definition:

> "*Therefor I prefer to stay with a more informal notion of intelligence being the sort of stuff that humans do, pretty much all the time.*" [Bro91a]

Fra starten var håbet at opnå intelligens, der var sammenlignelig med menneskelig intelligens. Dette gjorde, at tilgangen til problemet blev præget af den måde, man antog, at mennesker løser komplekse problemer. Følgende citat af Marvin Minsky fra 1955 giver et billede af hvorledes, man forstillede sig, at intelligente maskiner skulle opnås:

> "*[Intelligent machines] would tend to build up within itself an abstract model of the environment in which it is placed. If it were given a problem it could first explore solutions within the internal abstract model of the environment and then attempt external experiments.*" [Ark98]

Denne form for AI vil altså opbygge en model af omgivelserne, benytte denne model til at lægge en plan, og så føre denne plan ud i livet ved en række af handlinger. AI på denne form vil operere i en Sense-Model-Plan-Act cyklus som vist i figur 2.5. Det blev denne filosofi, der blev fremherskende inden for AI.

Der er to udfordringer ved Sense-Model-Plan-Act tilgangen til AI. For det første skal der opbygges en abstract symbolsk - men præcis - model af omverdenen, og for det andet skal der udvikles en plan på baggrund af denne model.

Opbygningen af modellen er i sig selv delt op i to delproblemer. For det første skal der vælges en passende symbolsk repræsentation af omgivelserne, og for det andet skal denne model vedligeholdes ved hjælp af efterbehandling af input fra sensorer. Ingen af disse to delproblemer er trivielle. At udtænke en symbolsk abstrakt model er besværligt, så snart det modellerede ikke er beskrevet med præcise regler. Derfor er f.eks. skak blevet et yndet testproblem inden for AI, da der i skak er præcist definerede tilstande og handlemuligheder. At vedligeholde en model er også besværligt. Ofte kræves der komplekse beregninger til at efterbehandle sensordata for at opnå den ønskede abstraktion. F.eks. er udtrækning af information fra et todimensionelt billede en videnskab i sig selv [Nil98] og næsten umuligt, når lysforhold og objekter ikke er designet specielt til formålet. Selv en simpel verdensmodel, kan altså være svær at vedligeholde, hvis der kun kan anvendes data fra sensorer.

Når modellen er opbygget, skal der lægges en plan. En plan er defineret som *en følge af handlinger, der fører til den ønskede tilstand* [Nil98]. Denne plan udvikles ved hjælp af den symbolske model.

Denne planlægning blev traditionelt udført som en søgning i tilstandsrummet for det specifikke problem med den antagelse, at den symbolske model er præcis, og at handlinger har præcist definerede effekter.

En fuldstændig afsøgning af alle mulige planer er sjældent mulig, da antallet af følger af handlinger stiger eksponentielt med antallet af mulige handlinger. F.eks vil en skakcomputer, der kan generere 3 søgeknuder pr. nanosekund, være 10^{22} år, svarende til cirka 7×10^{15} gange universets levetid, om at evaluere alle mulige opstillinger i skak [Nil98]. Dette illustrerer, at selv med en præcis og simpel verdensmodel kan planlægningen have eksponentiel kompleksitet.

Problemet med at søgegraferne bliver for store ved selv simple veldefinerede problemer, løses ved hjælp af problemspecifik viden. Denne viden anvendes til at lave heuristikker til at begrænse søgningen. Dette har været så succesfuldt, at man på trods af den store kompleksitet ved søgning har udviklet maskiner, der er i stand til at spille skak bedre end mennesker [Bro91a]. Disse maskiner har dog været specielt bygget til netop skak, og kan derfor kun løse dette specifikke problem. En sådan problemspecifik viden kan næppe benyttes til udvikling af en mere generel intelligens.

Det tradionelle syn på AI er også blevet benyttet i forbindelse med robotter. Robotten Shakey [Nil98] er et godt eksempel. Den benytter en traditionel AI planner, men kombineret med *reflekser* og *intermediate actions* i en trelags arkitektur. Shakey opererede i et specielt designet miljø og er blevet kritiseret for - på trods af de forsimplede omgivelser - at mangle evnen til at reagere på ændringer i omgivelserne [Bro91a].

Systemer, der anvender denne klassiske sense-model-plan-act tilgang til

AI kaldes i litteraturen for bl.a. *knowledge based systems*, *Top-Down systems*, *Klassisk/traditionel AI* og *Good Old Fashioned Artificial Intelligence.*

2.3 En ny ide

I midten af firserne voksede nye ideer frem. Den tradionelle tilgang til AI havde ikke været i stand til at producere robuste resultater i dynamiske omgivelser. Folkene bag de nye ideer - med Rodney Brooks i spidsen - argumenterede for, at man i stedet for den traditionelle *top-down* tilgang skulle anvende en *bottom-up* tilgang til problemet.

Der blev argumenteret for, at den traditionelle sense-model-plan-act model var utilstrækkelig og endda direkte ubrugelig, når det drejede sig om at opnå intelligente systemer, der befandt sig i virkelige dynamiske omgivelser i modsætning til specielt fabrikerede omgivelser.

Hele ideen med at opbygge en symbolsk model - og udfra denne lægge en plan - var i følge folkene bag de nye ideer forkert. Præcise modeller kunne simpelthen ikke opnås ud fra sensordata [Bro91b]. Det fundamentale problem med verdensmodeller illustreres af følgende citat.

> "*One can make a model of the real world, and that model may have states, but the real world as such does not.*" [Pfe96]

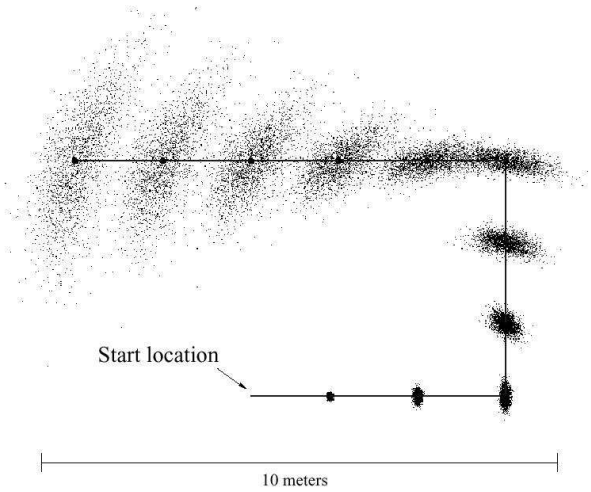
Teorien om, at det ikke var muligt at opnå komplette præcise modeller af verdenen, blev til en af de vigtigste læresætninger i den nye tilgang til AI, nemlig *situatedness*:

> "*The robots are situated in the world - they do not deal with abstract descriptions, but with the "here" and "now" of the environment that directly influences the behavior of the system.*" [Bro91b]

Brooks argumenterede for, at selve grundlaget for intelligens ligger i situatedness. Således definerer han en *physical grounding hypothesis*:

> "*[The physical grounding hypothesis] states that to build a system that is intelligent it is necessary to have its representation grounded in the physical world.*" [Bro90]

Symbolsk repræsentation af omgivelserne er altså ikke vejen til intelligente systemer. Tværtimod mener Brooks, at "the world is its own best model" [Bro90]. Dette citat er klassisk, og indeholder i sig selv den bevægelse væk fra komplekse verdensmodeler, som Brooks er fortaler for.



Figur 2.6: Usikkerhed i forbindelse med en robots bevægelse [FBKT02]

Der var altså meget kritik af selve sense-model delen af den tradionelle tilgang til AI. Men hvis man kunne opbygge en model, er det så den bedste løsning at benytte denne model til at lægge en plan?

Hvis omgivelserne var statiske og forudsigelige, er svaret muligvis ja, men hvad hvis omgivelserne er dynamiske? I følge Brooks vil svaret i så fald være nej. Dette skyldes, at dynamiske omgivelser kan ændre sig, mens en agent er i gang med at planlægge. Planen kan således være uaktuel, når den er færdig og skal føres ud i livet. En anden grund til, at planlægning kan være spildt er, at ligesom der er støj på sensorer, så er der også støj på aktuatorer. En motor opfører sig aldrig 100% som forventet, der vil altid være en afvigelse. Selvom afvigelserne enkeltvis er små, kan de akkumulere til store udsving over tid. Et godt eksempel på afvigelser fra det forventede, som akkumulerer over tid, kan ses i figur 2.6. Linien på figuren viser robottens bane, mens punkterne viser robottens viden om sin placering. Punkterne stammer fra algoritmen *Monte Carlo Localization* [FBKT02], hvor en robot genererer en endelig mængde af punkter og en sandsynlig til hvert af disse punkter. Disse punkter repræsenterer robottens *belief* om, hvor den befinder sig. Punkterne genereres ud fra de forudgående punkter ved hjælp af en probalistisk funktion, der angiver, hvordan robottens handlinger påvirker dens placering. Det ses tydeligt, at med den anvendte model for usikkerhederne i robottens handlinger, akkumulerer disse hurtigt til meget store usikkerheder.

Handlinger har altså ikke præcist definerede effekter, og små afvigelser

fra det forudsagte kan akkumulere over den følge af handlinger, som en plan består af. Dermed producerer en plan ikke nødvendigvis det forventede resultat. Selv med en præcis og komplet model af omgivelserne er det altså ikke nødvendigvis produktivt at bruge tid på at udvikle en kompliceret plan for at opnå et givet mål.

At robotter er en del af en dynamisk omverden, som kan påvirkes af robottens handlinger, er indeholdt i et andet af de vigtige nye begreber, nemlig *embodiment*:

> "*Embodiment: The robots have bodies and experience the world directly -Their actions are part of a dynamic with the world, and the actions have immediate feedback on the robots' own sensations.* " [Bro91b]

Et andet vigtigt argument var, at det ikke er muligt at anskue intelligens som et isoleret fænomen. Intelligent adfærd opstår som et samspil mellem agenten og dens omgivelser - i litteraturen kaldes dette for *emergent behavior*. *Emergent behavior* er rigt illustreret i litteraturen med robotter, der bygget ud fra simple principper kan udvise en forbløffende kompleks adfærd. *Braitenberg vehicles* [HMR91] er et klassisk eksempel på dette, men også *The Robot Sheepdog* [VFC99] og *The Didabots are cleaning up* [Pfe96] giver fremragende illustrationer af, hvor vigtigt det er også at anskue samspillet med omgivelserne, når man designer systemer.

Det giver altså ikke mening at designe robotter isoleret fra deres omgivelser. Robotter er en del af omgivelserne, og deres adfærd opstår ud af et samspil med omgivelserne. Det giver heller ikke mening at beskrive en robots adfærd uden for sine omgivelser. Man kan beskrive den interne teknik og arkitektur, men adfærd opstår kun som et samspil med omgivelserne.

> "*Behavior is by definition happening in the interaction of an agent with its environment, it cannot be reduced to internal mechanism.* " [Pfe96]

Dette giver i alt tre nye kriterier, som man bør holde sig for øje, når man designer robotter: *Situatedness, Embodiment og Emergence*. Disse begreber går stik imod Sense-Model-Plan-Act strategien og udgør således et markant skift i strategien, som robotter udvikles efter.

2.4 Subsumption-arkitekturen

Rodney Brooks var den første, der præsenterede de nye ideer. Han præsenterede ikke bare abstrakte ideer, men gik mere konkret til værks. I [Bro86]

beskrev han således en konkret måde, hvorpå man kan implementere ideerne i fysiske robotter. Arkitekturen beskrevet i [Bro86] består af en række selvstændige moduler, som hver består af en *Augmented Finite State Machine, AFSM*. Hver af disse AFSM består af et endeligt antal tilstande, input og output linier samt variable. Input kan komme fra sensorer eller fra andre AFSM, mens output kan gå til aktuatorer eller andre AFSM. De enkelte AFSM indkapsler sin egen tilstand, og det er således ikke muligt at tilgå tilstanden fra andre AFSM.

Inputlinierne har en buffer, således at det seneste element altid kan aflæses. I hver tilstand kan en AFSM beregne værdier ud fra variabler og input, skrive output og skifte tilstand alt sammen ud fra prædefinerede regler. [Bro86] opdeler arkitekturen i grupper tilstandene ud fra hvilke regler, der kan anvendes når en AFSM er i en given tilstand.

Output: En værdi til én af output linierne beregnes ud fra input og variabler. Herefter skifter den pågældende AFSM til en prædefineret tilstand.

Side effect: En variabel tilskrives en værdi beregnet ud fra variabler og input. Herefter skifter den pågældende AFSM til en prædefineret tilstand.

Conditional dispatch: Afhængigt af en betingelse beregnet ud fra variabler og input skifter den pågældende AFSM tilstand til en af to prædefinerede tilstande.

Event dispatch: Her er der række af betingelse/tilstand par, betingelserne kontrolleres indtil en betingelse bliver sand. Herefter skiftes til den tilsvarende tilstand

Alle disse moduler opererer uafhængigt af hinanden. Andre moduler kan kun påvirkes gennem input og output. Ud over de simple *ledninger* mellem output fra en AFSM til input til en anden AFSM eksisterer der to yderligere teknikker til at kombinere AFSM. For det første kan en *ledning* slutte ved et output fra en anden AFSM. Hvis et signal kommer langs denne *ledning*, sendes der ikke noget signal fra dette output. Dette output bliver altså forhindret, denne proces kaldes *inhibition*. For det andet kan en *ledning* slutte ved et input til en anden AFSM. Hvis dette er tilfældet, vil et signal på den pågældende *ledning* blive brugt som input, og det oprindelige input bliver altså undertrykt, denne proces kaldes *suppression*. Disse to teknikker løser konflikter, der opstår, når flere AFSM sender output til for eksempel aktuatorer.

Disse to teknikker benytter Brooks til at opbygge en lagdelt struktur af flere AFSM'er. Hvert lag, *competence layer*, tilføjer nye kompetencer, hvilket gør det muligt at teste robotter undervejs, hver gang et nyt lag tilføjes.

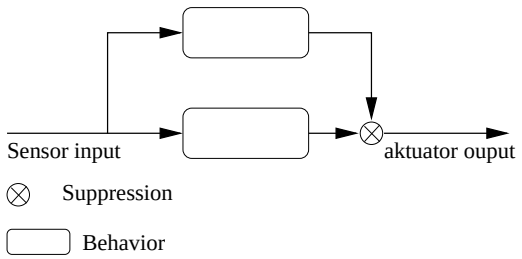
Denne modularisering gør udviklingen nemmere og giver mere robuste programmer. Den lagdelte struktur, hvor hvert lag tilføjer nye kompetencer, kaldes *Subsumption arkitektur*.

Subsumption-arkitekturen angav en bottom-up måde at implementere agenter, men den havde det problem, at den skalerede dårligt. Derfor blev teknikken udviklet yderligere ved indførelsen af begrebet *behavior*. En *behavior* er en gruppe af selvstændige processer, der tilsammen giver agenten en specifik kompetence [Bro90]. *Behaviors* svarer altså til *compentence layers* i *subsumption* arkitekturen.

Der kan være forbindelse mellem processerne internt i en *behavior* ved hjælp af *suppression* og *inhibition*, ligesom der kan være forbindelser på samme måde mellem de forskellige *behaviors*. En *behavior* fungerer som en indkapslende enhed, dvs de enkelte processer i en *behavior* kan dele tilstand, men tilstand deles ikke mellem forskellige *behaviors* [Bro90]. Fordelen ved denne indkapsling er, at hvert modul kan testes og udvikles separat.

Da hver *behavior* kan have intern tilstand, men tilstand deles ikke *behaviors* imellem, har en robot opbygget efter denne tilgang ikke nogen central model af verden. Den besidder heller ikke nogen central evne til at planlægge. Hver enkelt *behavior* benytter sensordata på sin egen måde, og hver enkelt *behavior* giver sit eget output til aktuatorerne. Hver *behavior* opererer i sin egen løkke, som genererer aktuatoroutput ud fra sensorinput. Dette giver bedre overensstemmelse med begrebet *situatedness*, da der ikke opbygges nogen central model af omgivelserne, men i stedet benyttes input direkte fra sensorerne til de enkelte *behaviors*. Desuden giver den decentrale struktur bedre overensstemmelse med begrebet *embodiment*, da kontrollen i hver *behavior* foregår som en løkke mellem input fra sensorer til output fra aktuatorer, der påvirker input fra sensorer, som igen giver nye output til aktuatorer. Det tredje begreb, *emergence*, kommer til udtryk ved, at robotternes adfærd opstår ved samspillet mellem de enkelte *behaviors* og robottens omgivelser.

Behaviors som indkapslende enhed er senere blevet kaldt *Competence Modules* [Mae89] *Behavior Producing Modules* [WM01]. Denne præcisering er foretaget for at undgå forvirring mellem en robots ydre adfærd (behavior) og en robots indre teknik bestående af indkapslende moduler (behaviors). Jeg vil i resten af dette speciale benytte begrebet *Behavior Producing Module* til at beskrive et indkapslet modul i en robots indre, der er ansvarlig for en bestemt del af robottens ydre adfærd. Dog vil begrebet *behavior baseret system* blive anvendt til at beskrive en agent, der er opbygget af *behavior producing modules*.



Figur 2.7: Suppression/Inhibition netværk

2.5 Valg af behavior

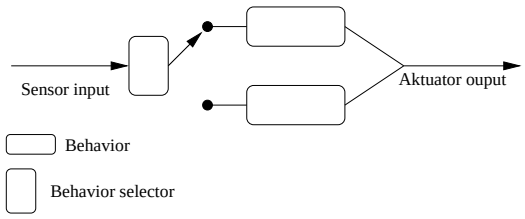
Et *behavior* baseret system består af en række *behavior producing modules*, der alle kan give output på samme tid. Det er derfor nødvendigt med en måde at sortere disse output, således at de resulterende output til aktuatorerne virker til at opfylde robottens ønskede ydre adfærd. Der er flere måder at opnå denne koordinering. Dette afsnit vil beskrive nogle af dem.

2.5.1 Suppression/Inhibition netværk

Brooks [Bro86] foreslår, at koordineringen mellem *behaviors* kan foregå ved hjælp af *inhibition* og *suppression*. Således at *behaviors* befinder sig i et netværk, hvor *suppression* og *inhibition* mellem modulerne gør, at præcis en *behavior* har kontrol over en bestemt aktuator på et givet tidspunkt, figur 2.7.

Den simpleste form for *suppression/inhibition* netværk er et præcedens hierarki, hvor de enkelte *behaviors* er organiseret i et hieraki, hvor den *behavior* øverst i hierarkiet og som ønsker kontrol over en aktuator, vinder kontrollen. Men også langt mere komplicerede hierarkier kan opnås ved hjælp at *inhibition* og *suppression*.

Denne struktur er fuldstændigt distribueret, da de forskellige *behaviors* kommunikerer ved at sende *suppress/inhibit* beskeder til hinanden. Det gør i princippet ikke nogen forskel, om de forskellige *behaviors* kører på den samme processor ved hjælp af et timesharing system, eller om *behaviors* er distribueret over flere processorer med mulighed for at kommunikere. Hvis man distribuerer *behaviors* over flere processorer, skal man dog tage højde for de karakteristika, som kommunikationen besidder. Hvor kommunikation mellem processer på samme processor som regel er hurtig og sikker, vil kommunikation mellem forskellige processorer muligvis involvere et eller flere usikkerhedsmomenter. F.eks. kan en robot implementeres ved hjælp af to RCX'er, appendix A, som er bygget sammen. Denne robot vil bestå af to proces-



Figur 2.8: Behavior Selection

sorer, som kan kommunikere ved hjælp af infrarød kommunikation, appendix B. Den infrarøde kommunikation er dog ustabil, så forbindelser i inhibition/suppression netværket kan være ustabile, hvilket vil have indflydelse på robottens adfærd. Man kan sige, at venstre hånd ikke ved, hvad højre hånd gør, og omvendt.

2.5.2 Behavior selection

[CFJ+02] beskriver en anden måde at organisere *behaviors* på. I stedet for, at alle *behaviors* er aktive og producerer output parallelt, tilføjes et *behavior selection modul*. Dette modul bestemmer ud fra sensorinput, og eventuel intern tilstand, hvilken *behavior*, der skal være aktiv. På denne måde er præcis én *behavior* aktiv på et givet tidspunkt. Denne form for organisation mellem *behaviors* kan kaldes *behavior selection*, figur 2.8.

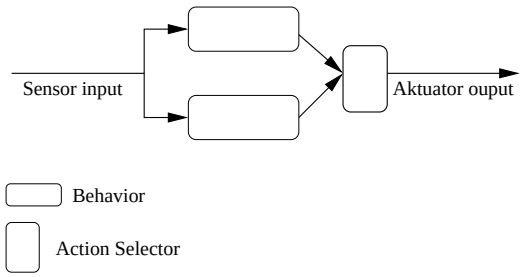
Modulet, der vælger den aktive *behavior*, kan implementeres på mange måder. I [CFJ+02] er *behavior selection* modulet baseret på motivationer [Kri00]. I dette system har hver *behavior* tilknyttet en motivation. Den *behavior* med den største udregnede motivation "vinder" kontrollen over aktuatorer. Motivationen genberegnes med passende periodiske intervaller. På denne måde opnås en reaktivitet[1] over for ændrede sensorinput. En robot baseret på *behavior* selection er afhængig af en hurtig genberegning af motivationerne for at opnå en høj reaktivitet i dynamiske omgivelser.

Behovet for hurtig genberegning af motivationerne sætter en grænse for, hvor komplekse udregningerne af motivationerne kan være.

2.5.3 Behavior koordination

I stedet for at lade et separat modul bestemme hvilken *behavior*, der er aktiv, kan man lade alle *behaviors* være aktive, og så lade et modul kontrollere

[1]Reaktivitet har flere betydninger i litteraturen, i dette tilfælde skal reaktivitet forstås som responsiveness



Figur 2.9: Behavior koordinering

aktuatorerne ud fra de forskellige *behaviors'* output. Dette kan kaldes *behavior koordination* [Ark98].

En koordination kan implementeres ved hjælp af mange forskellige strategier. Den simpleste er *Winner Takes All* [Ark98], hvor koordinationsmodulet vælger en *behavior*, der ønsker kontrol over aktuatorerne, og sender output fra denne videre til aktuatorerne. Udvælgelsen af en *behavior* i *Winner Takes All* kan implementeres som et simpelt *dominance hierarchy* [Ark98], hvor den øverste i hierarkiet, som giver output i den aktuelle situation, vælges.

Alternativ til et fast hieraki kan der implementeres en *Action-Selection* funktion [Mae90], som ud fra mål, tilstand og sensorinput udregner en værdi tilknyttet hver *behavior*. Den *behavior* med den største værdi "vinder" kontrol over aktuatorer. Dette er helt parallelt til *motivations* i [Kri00] bortset fra, at *Action-selection* funktionen også kan anvende output fra de forskellige *behaviors*, når en "vinder" skal findes.

En anden måde at implementere action selection kan være ved hjælp af *Behavioral Fusion* [Ark98]. Ved *behavioral fusion* kombineres output fra flere *behaviors* til at give et output til aktuatorerne. Som eksempel på *behavioral fusion* kan nævnes en robot med to *behaviors*, goLeft og goRight. Disse *behaviors* sender enten intet output eller et bestemt output. Det afgøres pseudotilfældigt internt i hver *behavior*, om den sender output eller ej. Robotten har implementeret et koordinationsmodul, der modtager output fra de to *behaviors* som input og ud fra dette beregner vektorsummen af disse input. Hvis kun den ene *behavior* er aktiv, vil robotten dreje henholdsvis enten mod venstre eller højre. Hvis begge *behaviors* er aktive vil resultatet være, at robotten bevæger sig lige ud. *Behavioral fusion* er ikke begrænset til vektoraddition. Man kan således forestille sig vilkårlige måder at kombinere output på, f.eks. kan output fra de forskellige *behaviors* vægtes ved hjælp af *motivationer*.

2.6 Eksempler på behavior baserede robotter

Den *behavior* baserede tilgang er grundlæggende eksperimentel. De tre kriterier *situatedness*, *embodiment* og *emergence* udtrykker, at robotter indgår i et komplekst samspil med omgivelserne. Det er således ikke muligt at evaluere den *behavior* baserede tilgang teoretisk. I stedet er det nødvendigt at implementere robotter, der kan afprøve tilgangens brugbarhed.

2.6.1 Allen

Rodney Brooks var den første, der beskrev principperne i den *behavior* baserede tilgang. I [Bro86] beskrives en fysisk implementation af en robot. Robottens kontrolprogram er implementeret ud fra *subsumption* arkitekturen. Det nederste lag implementerer *obstacle avoidance* ved hjælp af sonar. Når robotten kun kontrolleres af dette lag, vil den bevæge sig væk fra objekter og ud i åbne rum. Når robotten er tilpas langt væk fra det nærmeste objekt, vil den stoppe og blive der. Det andet lag implementerer en *wander* adfærd. Robotten er således ikke længere tilfreds med at opholde sig stille langt væk fra nærmeste objekt. Efter et stykke tid vil den begynde at vandre. Det tredje lag implementerer en *pathplan* adfærd. Denne adfærd vil, når robotten ikke bevæger sig, finde et mål, som robotten så skal bevæge sig hen imod. Når dette sker, vil *pathplan* modulet sende en heading til *obstacle avoidance* modulet, der så benytter denne heading sammen med sin egen adfærd til at beregne en ny retning.

Robotten fra [Bro86] var den første, der benyttede subsumption arkitekturen, og den blev hurtigt fulgt op af flere andre robotter, f.eks. Chengis [Bro91b], Tom & Jerry og Herbert [Bro90]. Alle demonstrerede disse robotter, hvordan robotterne kunne opnå kompleks adfærd med minimal intern repræsentation af omgivelserne.

2.6.2 Toto

[Mat92] implementerede en *behavior* baseret robot [Mat92], Toto. Toto havde fem kompetencer; *obstacle avoidance*, *boundary following*, *landmark detection*, *map construction* og *path finding*.
Landmark detection blev opnået ved at have fire *behaviors*, der hver især genkender en bestemt form for *landmark*, f.eks. *leftwall* eller *corridoor*. Hver af disse *landmark detectors* har *activation level*, og når denne når en bestemt grænseværdi, vil den pågældende *landmark detector* sende besked til *the map behaviors*. En sådan *map behavior* repræsenterer et bestemt *landmark*, og de er forbundet topologisk. Når en *landmark detector* sender output til kortet, vil den *map behavior*, der passer bedst på input, blive aktiv. Hvis der ikke

er nogen *map behavior*, der passer, oprettes en ny. På denne måde opretholder robotten hele tiden en viden om sin egen placering i omgivelserne, og efterhånden som den bevæger, sig vil den opnå større viden om omgivelserne.

Toto var interessant, fordi den demonstrerede, at opgaver, som traditionelt blev løst med en planlægning ved hjælp af en verdensmodel, kunne løses inden for den *behavior* baserede tilgang.

> "*Toto's general capabilities appear typical of a deliberative system, but were implemented with* behavior-based *approach* " [Mat97]

Toto var banebrydende fordi den præsterede at opbygge et topologisk kort inden for rammen af den *behavior* baserede tilgang. Dette var overraskende for mange af kritikkerne af den *behavior* baserede tilgang, der havde kritiseret den for ikke at kunne understøtte andet end simpel reaktiv adfærd. Toto tilbageviste dette ved med en meget elegant arkitektur at implementere et *behavior* baseret system, der indeholdt mange af de kompetencer, der ellers krævede en central verdensmodel.

2.7 Evaluering af de nye ideer

De nye kriterier - *situatedness*, *embodiment* og *emergence* - giver en helt ny måde at opbygge agenter på. Med traditionel AI håbede man på, at løsninger i dynamiske og naturlige omgivelser ville kunne opnås, når først problemerne var løst i mere statiske og specialdesignede omgivelser. Derimod håber tilhængere af den *behavior* baserede tilgang, at når simple problemer først er løst i dynamiske omgivelser, så vil mere kompleks intelligens kunne udvikles af dette. Det langsigtede mål for begge tilgange er i længden at opnå intelligens, der kan operere i naturlige og dynamiske omgivelser. Denne forskel i tilgang er illustreret i figur 2.10.

Et af de store kritikpunkter mod den nye *behavior* baserede tilgang er spørgsmålet om skalerbarhed. Skalerer den *behavior* baserede tilgang til mere komplekse problemer? Dette spørgsmål er umuligt at besvare, før man eventuelt har et bevis i form af et implementeret system, der rent faktisk opfylder dette. Man kan dog vende kritikken. Således kan spørgsmålet om skalerbarhed også stilles til den traditionelle AI. Håbet for den traditionelle tilgang var, at når problemer var løst i statiske forudsigelige omgivelser, så ville de skalere til dynamiske uforudsigelige omgivelser. Det er ikke blevet bekræftet, at systemer, der anvender den tradionelle tilgang, skalerer på denne måde. Tværtimod tyder mange resultater på, at dette ikke er tilfældet.

Spørgsmålet om skalerbarhed er et fuldstændigt legitimt spørgsmål at stille til den *behavior* baserede tilgang. Det er på ingen måde oplagt at højere

intelligens vil udspringe (*emerge*) fra samspillet mellem individuelle *behaviors*.

Rodney Brooks har brugt en del tid på at argumentere for, at den *behavior* baserede tilgang skalerer. Brooks finder sine argumenter dels i evolutionen [Bro90] og dels i studiet af menneskelig intelligens [BBI+98].

Argumentet med udgangspunkt i evolutionen går på at udvikling af mennesker på jorden har taget udgangspunkt i simple enkeltcellede organismer for ca. 3,5 milliarder år siden. Det tog herefter ca. en milliard år at udvikle planter med fotosyntese. Det tog yderligere halvanden milliard år at udvikle fisk, som dukkede op for ca 550 millioner år siden. Krybdyr kom for 370 millioner år siden og pattedyr for 250 millioner år siden. Mennesket opstod først for 2,5 millioner år siden, landbrug blev opfundet for 19000 år siden, skriftsprog for 5000 år siden og ekspertviden er udviklet over de sidste få hundrede år [Bro90].

Hele denne argumentation slutter Brooks af med argumentet for, at den *behavior* baserede tilgang er vejen mod kunstig intelligens.

> "*This suggest that problem solving* behavior*, language, expert knowledge and application, and reason, are all rather simple once the essence of being and reacting are available.* " [Bro90]

Om man finder, dette argument overbevisende, er i høj grad et spørgsmål om tro. Derfor vil debatten om skalerbarhed ikke blive diskuteret yderligere i dette speciale. I fokuseres på følgende betragtning:

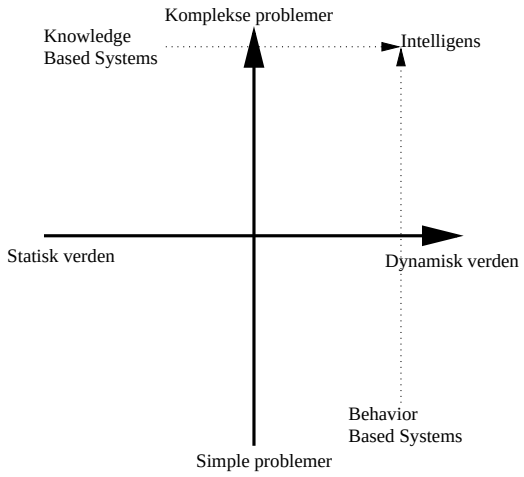
> "*it is unfair to claim that an elephant has no intelligence worth studying just because it does not play chess.* " [Bro90]

Det er altså ikke nødvendigvis altafgørende, om den *behavior* baserede tilgang skalerer hele vejen til menneskelig intelligens. Under alle omstændigheder har den *behavior* baserede tilgang resulteret i systemer, der har udvist stor succes i dynamiske fysiske omgivelser, og den egenskab gør tilgangen velegnet til implementation af robotter.

Når robotternes adfærd skal undersøges, er der en række evalueringskriterier foruden skalerbarhed, som er vigtige.

Et af de vigtigste evalueringskriterier er robusthed. Hvordan ændres robottens adfærd, hvis f.eks. en sensor giver forkerte resultater?

Generelt vil den *behavior* baserede arkitektur være mere robust end en arkitektur med en central verdensmodel. Grunden til dette er, at hver deladfærd tilføjes som et modul, der selv er forbundet direkte til sensorer og aktuatorer. Hvis en sensor holder op med at virke, vil kun de *behavior producing modules*, der er afhængige af pågældende sensor, få problemer. Da



Figur 2.10: Indplacering af traditionel og behavior baseret AI

moduler er designet uafhængigt af hinanden og derfor fungerer separat, vil de andre moduler fungere som om intet var hændt ved et sammenbrud i enkelte moduler.

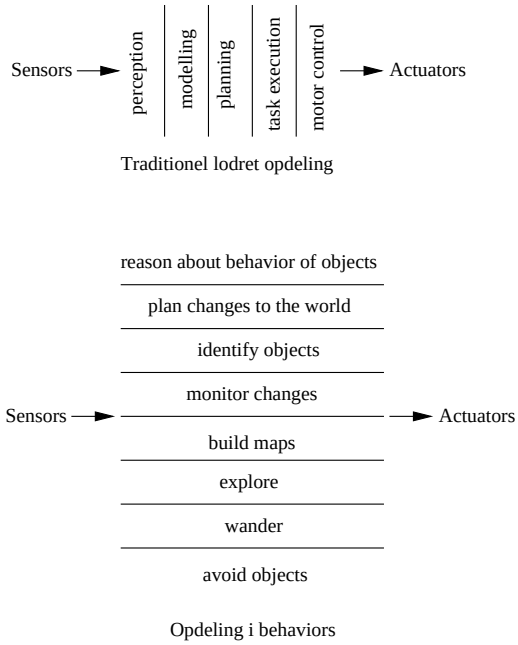
Hvis robotten derimod var opbygget ud fra den tradionelle tilgang, vil alle robotter dele en verdensmodel, som ikke længere var korrekt, og sammenbrudet er potentielt meget større.

Som alt andet inden for dette område er det vigtigt at validere robustheden af *behavior* baserede systemer. Dette er der lagt stor vægt på i litteraturen. F.eks. er Pattie Maes kommet til følgende resultat.

> "*Emergent complexity is often more robust, flexible and fault-tolerant than programmed, top-down organized complexity.*" [Mae94]

Robusthed er en egenskab, der er svær at bevise. Den skal efterprøves i tests på fysiske robotter.

Et andet vigtigt kriterium for evaluering af autonome robotter er evnen til at reagere på ændringer i omverdenen. Her er den *behavior* baserede tilgang også den tradionelle overlegen. Dette skyldes den tætte kobling mellem sensorer og aktuatorer i de enkelte *behavior producing modules* - der bliver ikke brugt tid på at opbygge en central verdensmodel. I traditionelle AI systemer skal verdensmodellen først opdateres, før ændringer i omgivelserne giver sig udslag i handling fra robotten. I den *behavior* baserede tilgang er denne kobling langt mere direkte. Figur 2.11 illustrerer forskellen mellem de traditionelle systemers lodrette opdeling og de *behavior* baserede systemer vandrette opdeling. Den vandrette opdeling giver langt færre beregninger



Figur 2.11: Modularitet i traditionel og behavior baseret AI [Bro86]

mellem sensorer og aktuatorer, dermed vil agenten reagere langt hurtigere på ændringer i omgivelserne.

2.8 Delkonklusion

I dette afsnit et to tilgange til at programmere intellignte robotter blevet præsenteret. Disse to tilgange er blevet kaldt henholdsvis den traditionelle tilgang og den *behavior* baserede tilgang. Disse to giver yderpunkterne i en interessant debat om måden at opbygge intelligente systemer. De er på ingen måde dækkende for hele forskningen i AI og robotter. Teknikker som neurale netværk [Nil98] eller genetiske algoritmer [Mic92] er også interessant, og kunne også have været diskuteret i forbindelse med kunstig intelligens.

Den traditionelle tilgang består i at der opbygges en verdensmodel, hvorefter der ud fra denne lægges en plan. Verdensmodellen består ofte af en mængde af logiske udtryk, som beskriver omgivelserne. Planlægningen består derefter i at ved hjælp af logiske regler, at beregne mulige ændringer i verdensmodellen.

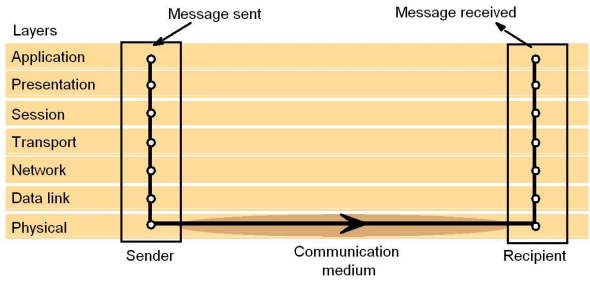
Den *behavior* baserede tilgang kritiserede den traditionelle tilgang for at mangle evnen til at reagere på ændringer i omgivelserne. Dette skyldtes i følge folkene bag den *behavior* baserede tilgang, at hele ideen med at opbygge en

verdensmodel var forkert. Problemet med at opbygge præcise verdensmodeller har man inden for den traditionelle AI forsøgt løst ved at opbygge probabilistiske verdensmodeller, hvor robotterne ikke har en præcis viden, om sine omgivelser, men i stedet nogle forskellige muligheder for, hvordan verdenen ser ud og en sandsynlighed knyttet til disse [Thr02].

Dette løser nogle af problemerne, da en sådan verdensmodel tager højde for en mangelfuld viden om omgivelserne. Men det helt fundamentale problem, nemlig at omgivelserne i modsætning til en model ikke har tilstande, kan ikke løses på denne måde.

Et andet problem ved den traditionelle tilgang er at planer er uaktuelle, når de er blevet færdige. Dette er forsøgt løst ved at implementere hurtige algoritmer til at modificere planer, [LK02b] og [LK02a] er eksempler på dette. Men uanset hvor hurtige disse *replanning* algoritmer bliver, vil de stadig konstant være et skridt efter omgivelserne, da de benytter en verdensmodel til at lægge deres planer.

I dette speciale er den *behavior* baserede tilgang blevet valgt til implementation i de eksperimentelle afsnit.



Figur 3.1: OSI modellen [CDK01]

3 Kommunikation

International Organization for Standardization (ISO) har vedtaget en standard for protokolstakke. Denne standard går under betegnelsen *the Open Systems Interconnection model* OSI-modellen [CDK01].

OSI modellen består af syv lag; *Physical, Data Link, Network, Transport, Session, Presentation* og *Application* [Sta00], figur 3.1. De forskellige lag deler ansvar imellem sig, og hvert lag implementerer funktionalitet, som kan benyttes af laget ovenpå. Dette speciale vil primært beskæftige mig med lagene *Data link* og *Network*, ligesom overvejelser på *physical* laget lejlighedsvis blive inddraget.

Physical: Det fysiske lag håndterer transmission af ustrukturede bitstrømme over et fysisk medium [Sta00].

Data Link: Datalinklaget implementerer stabil kommunikation over en direkte fysisk forbindelse mellem enheder [Sta00].

Network: Netværkslaget håndterer kommunikation mellem enheder, der ikke er direkte forbundet. Dette involverer blandt andet routing [CDK01].

OSI modellen tjener primært som fælles reference, når protokoller diskuteres. Der er mange protokoller, der afviger fra OSI-modellen. TCP/IP [Sta00] er den mest udbredte af disse. Til trods for, at OSI-modellen sjældent bliver overholdt i protokoller, tjener den alligevel et stort formål som reference, og mange beskrivelser af protokoller benytter OSI modellen.

3.1 Trådløs kommunikation

Inden for alle områder af datalogi bliver udviklingen i stor grad dikteret af teknologiske fremskridt. Nye teknologier giver mulighed for at udforske nye

	Infrared		Spread Spectrum		Radio
	Difused Infrared	**Directed Beam Infrared**	**Frequency Hopping**	**Direct Sequence**	**Narrow-band Microwave**
Data rate (Mbps)	1 to 4	1 to 10	1 to 3	2 to 20	10 to 20
Mobility	Stationary / mobile	Stationary with LOS	Mobile	Stationary / mobile	
Range (ft)	50 to 200	80	100 to 300	100 to 800	40 to 130
Detectability	Negligible		Little		Some
Wavelength / frequency	λ: 800 to 900 nm		902 to 928 Hz 2.4 to 2.4835 GHz 5.725 to 5.85 GHz		902 to 928 MHz 5.2 to 5.775 GHz 18.825 to 19.205 GHz
Modulation technique	ASK		FSK	QPSK	FSK / QPSK
Access Method	CSMA	Token ring, CSMA	CSMA		Reservation, ALOHA, CSMA
License Required	No		No		Yes unless ISM

Tabel 3.1: Sammenligning mellem forskellige trådløse teknologier [Sta00]

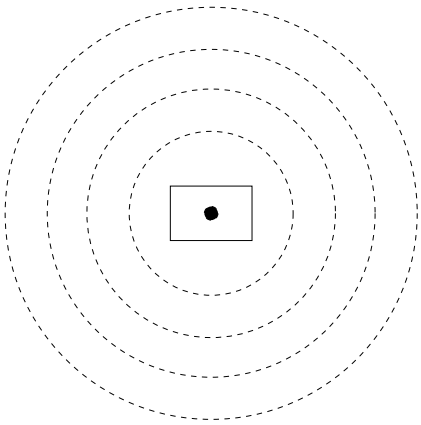
områder, som ikke tidligere er blevet undersøgt.

Sådan forholder det sig også med multirobotsystemer. Software agenter har i mange år kunnet kommunikere over netværk, men kommunikation mellem robotter har været mindre udforsket. De seneste år er forskningen i multirobotsystemer vokset eksplosivt. Dette skyldes i høj grad, at kommunikationssystemer til robotter er blevet billigere og bedre.

Der er forskellige teknologier, der understøtter trådløs kommunikation, tabel 3.1. Dem som vil blive undersøgt nærmere i dette speciale er *infrared* og *radio*

3.1.1 Radiokommunikation

Radiokommmunikation mellem robotter og andre mobile enheder er den mest udbredte form for kommunikation. Den store popularitet skyldes, at radiokommunikation er hurtig, stabil, langtrækkende og har simple udbredningskarakteristika. Radiokommunikation udbreder sig i alle retninger fra en



Figur 3.2: Udbredelse af radiokommunikation

antenne, figur 3.2. Desuden kan radiokommunikation gennemtrænge forhindringer i omgivelserne. Således kan skyggeeffekter til en hvis grad undgås. Radiokommunikation tillader altså agenter at kommunikere med andre agenter - blot de er inden for en bestemt afstand af hinanden.

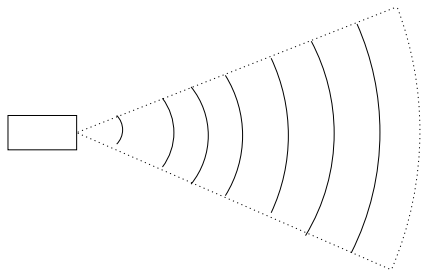
3.1.2 Infrarød kommunikation

En anden udbredt type af kommunikation er infrarød kommunikation. Infrarødt lys udbreder sig som fremadrettede kegler fra afsenderen. I forhold til radiokommunikation har infrarød kommunikation en lavere bitrate, figur 3.1. Desuden er infrarød kommunikation i højere grad udsat for skyggeeffekter fra omgivelserne. Hvor radiokommunikation til en hvis grad kan gennemtrænge forhindringer i omgivelserne, har infrarød kommunikation brug for, at afsender og modtager er inden for synsvidde. Dog kan reflektion til en hvis grad kompensere for dette.

Fordelene ved infrarød kommunikation er, at infrarøde transceivere er billige og ikke kræver ret meget strøm. Desuden har den retningsbestemte udbredelse af signalet også fordele i visse scenarier. Afsenderen har således en bedre mulighed for at rette signalet specielt til bestemte modtagere. Af disse grunde bliver infrarøde transceivere benyttet i blandt andet fjernbetjeninger.

3.1.3 Problemer med trådløs kommunikation

Der opstår en række problemer på både fysisk-, datalink- og netværksniveau, når kommunikation er trådløs. Disse problemer eksisterer ikke ved traditionel kommunikation. Fælles for problemerne er, at de på et eller andet niveau har



Figur 3.3: Udbredelse af infrarød kommunikation

betydning for specifikke valg i forbindelse med design af en kommunikationsprotokol.

Det første problem er strømforbruget. Ved ikke-mobile enheder har strømforbruget ikke nogen betydning for designet af en protokol, men da trådløse enheder som regel er batteridrevene er strøm en mangelvare. Det er derfor vigtigt at overveje, hvordan kommunikationen benyttes. Det er altså for trådløs kommunikation et specifikt designmål at minimere brugen af kommunikationen så meget som muligt. Strømforbruget kan minimeres på to måder, for det første kan sendestyrken formindskes, hvilket resulterer i kortere rækkevidde. For det andet kan mængden af kommunikation minimeres. Uanset hvilken af de to løsninger der vælges, vil hensyn til strømforbruget ofte spille ind på både gensendelsesstrategi for tabte pakker på datalink niveau og routingstrategier på netværksniveau.

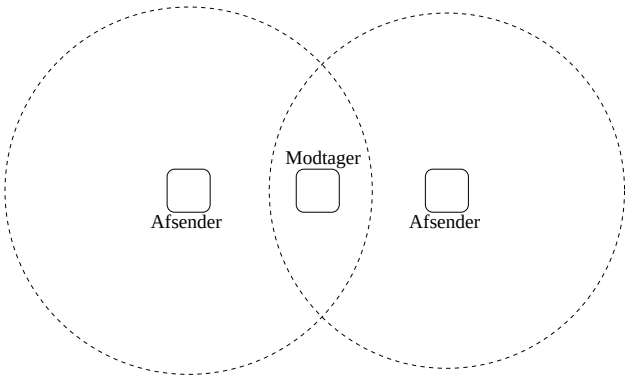
Det andet problem eksisterer også i traditionelle kommunikationsprotokoller, men bliver endnu mere udtalt for trådløs kommunikation. For trådløs kommunikation er der ofte et stort pakketab. Dette kan skyldes, at enhederne er mobile, støj i omgivelserne eller kollision mellem pakker. Det er således vigtigt for stabiliteten, at den trådløse protokol har en fornuftig håndtering af tabte pakker. Pakketab er ikke et enestående problem for trådløs kommunikation, og det er forholdsvist simpelt at implementere en fornuftig gensendelsesstrategi ved hjælp af acknowledgements.

Det tredje problem er, at der ofte er mulighed for unidirectionelle links, det vil sige at kommunikation er mulig fra A til B, men ikke fra B til A. Dette problem opstår ofte i forbindelse med retningsbestemt kommunikation, som ved infrarød kommunikation, men problemet kan også opstå i forbindelse med radiokommunikation for eksempel, hvis transceiverne i netværket ikke sender med samme styrke. Eksistensen af unidirectionelle links giver nogle udfordringer for designet af kommunikationsprotokollen, som ikke er til stede traditionelt. Der er tre måder, hvorpå man kan håndtere problemet med uni-

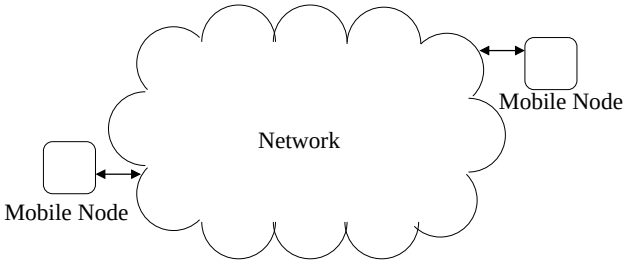
directionelle links. For det første kan man se bort fra unidirectionelle links ved at lade være med at benytte dem. Dette kan være en acceptabel løsning, hvis der er veje i netværket uden om de unidirectionelle links. For det andet kan man vælge de mobile enheder således, at disse links ikke opstår. Det kan man f.eks. gøre ved at anvende samme radiotransceivere i hele netværket. For det tredje kan man implementere protokollen til at understøtte unidirectionelle links. Dette kan for eksempel gøres på datalink niveau ved ikke at kræve acknowledgements.

Det fjerde problem opstår ved at trådløse netværk ofte er præget af en stor mobilitet. Således at forbindelser løbende opstår og brydes. Der skal udvikles en teknik, som gør det muligt for enheder at opdage, når nye forbindelser opstår. Dette problem kaldes *neighbour discovery*. IP antager at enheder er forbundet til et bestemt subnet, hvis enheden flyttes fra dette subnet til et andet, så kan den samme IP-adresse ikke anvendes til at kontakte denne enhed [CDK01]. Der er mange måder at implementere *neighbour discovery*. En måde at løse det består i at implementere en *discovery service*, der er forbundet til internettet, samt de andre enheder i nærheden [CDK01]. Denne *discovery service* skal have en opdateret liste over hvilke services, der er tilgængelige. Denne teknik egner sig udemærket til situationer, hvor en bruger anvender en mobil enhed, og ønsker at tilgå funktioner på andre mobile enheder i nærheden [CDK01]. Problemet med at anvende *discovery service* er at den antager asymmetriske roller. I et netværk, hvor ingen enheder kan påtage sig rollen som *discovery service*, er det nødvendigt at benytte en anden algoritme. Et eksempel på en anden algoritme er at lade alle enheder udsende periodiske signaler, som indeholder de information, der ellers skulle håndteres af en *discovery service*. Dette er den måde, som anvendes i [RK03], hvor periodiske *beacon* pakker udsende. En enhed, der ønsker at foretage *neighbour discovery* kan så lytte efter disse *beacon* pakker, for der igennem at opnå viden om andre enheder i nærheden.

Det sidste problem er det problem der opstår, hvis to enheder ikke kan kommunikere med hinanden, men begge forsøger at sende en besked til den samme modtager, figur 3.4. Dette problem kaldes *Hidden Terminal* [Per98]. Problemet med *hidden terminals* skyldes kommunikationens trådløse natur og kan henføres til problemet med at lave *colision detection* for trådløse medier [Kar90]. En afsender kan ikke se, om et afsendt signal er kollideret med andre signaler, da sådanne kollisioner er lokale. Det er således svært at løse *hidden terminal* problemet, men problemet kan minimeres ved flere forskellige teknikker til at ordne afsendelsen af beskeder, så der bliver færre kollisioner. For eksempel kan der benyttes en reservation af mediet således, at en afsender først spørger modtageren, om denne er klar til at modtage en besked. Dette gøres ved at sende en *Request To Send*, RTS, besked til modtageren.



Figur 3.4: Hidden terminal problemet

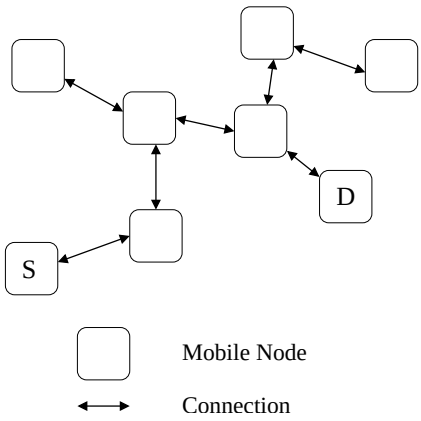


Figur 3.5: Mobile IP netværks topologi

Modtageren svarer så igen med en *Clear To Send*, CTS, besked [Kar90]. Denne form for reservation af mediet løser ikke problemet, da RTS og CTS beskeder stadig kan kollidere, men det minimerer problemet. Hvis RTS- og CTS-beskeder er mindre end beskeder med data, er sandsynligheden for, at de kolliderer mindre, og hvis de kolliderer, er de hurtigere at gensende. En anden måde hvorpå man kan minimere problemet er, at designe de fysiske kommunikationssignaler således, at kollisioner minimeres. Dette kan gøres ved at anvende kommunikation med kort rækkevidde eller retningsbestemte signaler. Denne løsning vil til gengæld gøre antallet af forbindelser mellem enheder i netværket mindre.

3.2 Ad Hoc netværk

I takt med, at kommunikationen er blevet trådløs og enhederne er blevet mobile, er de traditionelle routingsstrategier ikke længere tilstrækkelige til at håndtere kommunikationen mellem enheder, der ikke kan kommunikere direkte med hinanden [Sys]. Dette blev i første omgang løst ved at designe



Figur 3.6: Ad Hoc netværks topologi

mobile protokoller [Joh95]. Fælles for disse protokoller er, at de antager, at de mobile enheder har adgang til at stort internetværk, som kan benyttes, når de mobile enheder skal kommunikere med hinanden. Det er således kun i periferien af netværket, at enhederne er mobile, figur 3.5.

Dette er dog ikke altid tilstrækkeligt. Det er muligt, at en mobil enhed ikke kan kommunikere direkte med et fast netværk, men til gengæld kan kommunikere med andre mobile enheder og derigennem med det faste netværk.

Dette har givet anledning til en ny form for netværk, *Ad Hoc netværk*. Ad Hoc netværk adskiller sig fra de traditionelle netværk ved, at alle enhederne kan fungere som router, figur 3.6. I Ad Hoc netværk er der ikke den traditionelle opdeling, hvor der er en fast netværkstopologi, hvis formål er at understøtte kommunikation mellem enheder, som er tilkoblet i periferien af dette netværk.

At Ad Hoc netværk er fuldt decentraliserede og uafhængige af en central netværkstopologi gør Ad Hoc netværk oplagte at benytte i forbindelse med multirobotsystemer.

Ad Hoc netværk har som regel en hurtigt skiftende topologi, da de fleste enheder er mobile. Derfor er det nødvendigt at opfinde nye routingsprotokoller. En vigtig distinktion mellem forskellige Ad Hoc routingsprotokoller er, om de baserer sig på periodisk udsendelse af routing information [PB94], eller om de er *on-demand* [JMB01]. Forskellen mellem disse to er, om de enkelte enheder forsøger at vedligeholde routing-tabeller, eller om de i stedet søger efter en route, når der er behov for at kommunikere. Fordelen ved den første er, at en route allerede er kendt, således at der ikke opstår noget overhead i forbindelse med at oprette en route. Fordelen ved den anden er, at netværket ikke belastes unødigt ved at udsende route-data. En performance sammelig-

ning mellem forskellige routing protokoller til Ad Hoc netværk kan findes i [BMJ+98]. Graden af mobilitet har stor betydning for om der skal benyttes en *on-demand* eller en mere en mere traditionel routingsprotokol. Således vil det overhead, der opstår ved oprettelse af en forbindelse når der anvendes en *on-demand* routingprotokol, bedst kunne forsvares, hvis det er svært at holde tabeller med routinginformation opdateret, på grund af en stor mobilitet.

Formålet med routing protokoller i Ad Hoc netværk er traditionelt at implementere end-to-end kommunikation mellem to mobile enheder gennem et netværk af andre sidestillede mobile enheder. Det antages at der er en forbindelse gennem én eller flere intermediate knuder mellem afsender og modtager. Denne antagelse holder ikke altid stik, man kan nemt forestille sig situationer, hvor et mobilt ad hoc netværk bliver fragmenteret, således at der ikke er forbindelse mellem afsender og modtager. Hvis der ikke er forbindelse mellem afsender og modtager, vil routingsprotokoller baseret på normal end-to-end kommunikation fejle.

Der eksisterer situationer, hvor det er ønskeligt at have mulighed for at kommunikere mellem enheder, hvor der på afsendelsestidspunktet ikke er en forbindelse. Man kan f.eks. forestille sig, at netværket er fragmenteret i to mindre netværk, net1 og net2. En robot i net1, A, ønsker at sende en besked til en robot i net2, M, hvilket ikke er muligt som netværket ser ud. Hvis nu en robot, I, i net1 har modtaget beskeden fra A og bevæger sig fra net1 til net2, uden at de to net på noget tidspunkt bliver forbundet, vil I have mulighed for at kommunikere med M, og dermed kan I fungere som envejs bro mellem net1 og net2 i dette tilfælde. Det er altså muligt for A at sende en besked til M, selvom A og M er placeret forskellige steder i et fragmenteret net.

Fragmentering kan nemt opstå i et net bestående af mobile enheder, og da enhederne er mobile, er det heller ikke urealistisk, at de enkelte enheder bevæger sig fra et subnet til et andet. For at maksimere muligheden for kommunikation er det derfor fordelagtigt i stærkt fragmenterede netværk, hvis informationsudvekslingen kan ske mellem to enheder uden, at der på noget tidspunkt er en forbindelse mellem disse.

Der er blevet udviklet forskellige protokoller, der understøtter kommunikation i fragmenterede ad hoc netværk. Fælles for disse er, at der ikke oprettes en route mellem afsender og modtager, da dette ikke er muligt. I stedet benyttes en metode til at sprede data i netværket. Den simpleste måde at sprede data i netværket er *flooding*. [Win00] beskriver Ad Hoc routingsprotokol baseret på flooding. Ideen bag flooding er, at en enhed, der ønsker at sende data, sender disse data til alle sine naboer. Hver nabo gemmer disse beskeder i en liste og sender dem igen til sine naboer. Hvis en enhed bevæger sig inden for kommunikationsradius af en ny nabo, sender den sin liste af cachede beskeder til denne. Da beskeder bliver spredt mest muligt, vil beskeden

Figur 3.7: Ad Hoc netværk med RCX [GVSM02b]

i teorien nå frem til modtageren, hvis dette overhovedet er muligt.

Problemet med denne flooding protokol er, at den kræver at beskeder bliver gemt lokalt på alle enheder. Hvis der er store mængder trafik på netværket, vil der være stor sandsynlighed for, at de enkelte enheder ikke kan følge med, og dermed bliver bufferen til beskeder overfyldt med pakketab til følge. Dette problem kan minimeres ved, at man nøje overvejer bufferstørrelsen og skalerer denne efter den forventede trafikmængde [Win00]. Selv ved buffer overflows er sandsynligheden for, at datapakker tabes helt ikke særlig stor. Dette skyldes, at pakkerne som oftest er i bufferen på flere forskellige enheder af gangen. Dette hjælper på problemet med bufferoverflows, og pakkerne kan oftest bortsmides efter et simpelt first-in first-out princip uden, at dette påvirker protokollens stabilitet i alvorlig negativ grad.

Et andet problem med flooding er den store mængde trafik, som den vil generere. Denne trafik kan reduceres ved at lade hver enhed selv beslutte hvilke pakker, den sender videre. Dette giver en *gossip* protokol, som beskrevet i [GVSM02b]. Valget om kun at sende nogle af beskederne videre kan også være nyttigt i Ad Hoc netværk med en lav tæthed og en høj mobilitet. I et sådant netværk vil to enheder sjældent være naboer i lang tid af gangen. Dermed er der ikke tid til at sende samtlige pakker, men kun nogle få. Derfor er det vigtigt at have fornuftige kriterier for hvilke pakker, der skal sendes næste gang.

Man kan forestille sig mange kriterier til at vælge den pakke, der skal sendes næste gang. Den oprindelige afsender kan påsætte en prioritet til en besked, således at andre enheder kan benytte dette til at vurdere pakkens vigtighed. Pakkerne kan også tilføjes en hop-count, evt. med identifikation af de enheder, der har videresendt pakken. Endelig kan en enhed benytte rent

lokal information som for eksempel, hvornår beskeden blev modtaget, hvor mange gange enheden har videresendt pakken og så videre, ligesom en enhed kan benytte en grad af tilfældighed.

De ovenfor nævnte routingsprotokoller er baseret på flooding. Dette er nødvendigt, da de ikke benytter nogen form for viden om netværkets udformning. Flooding har det problem, at der genereres store mængder trafik på netværket, hvilket bruger meget strøm. I scenarier, hvor strøm er en mangelvare, er flooding altså ikke en optimal strategi.

I det tilfælde, at flooding ikke kan benyttes til at route beskeder, er det nødvendigt at benytte viden om netværkets struktur til at opnå en bedre routing. Systemer kan i større eller mindre grad udnytte viden om netværkets topologi. Det er stort set umuligt at opretholde en avanceret viden om netværkets topologi, men mindre kan også gøre det. Man kunne for eksempel tænke sig en kombination af *dynamic source routing* [JMB01] og flooding/gossip. Denne kombination kunne bestå i at afsøge afsenderens subnet for router til modtageren ved hjælp af dynamic source routing. Hvis en sådan route ikke eksisterer, benyttes flooding. De enheder, der modtager en pakke som følge af flooding vil forsøge at videresende denne ud fra den samme strategi, hvis den opdager en ny nabo.

Ud over viden om netværkets struktur kan der anvendes *aktiv routing* [O'H02]. Aktiv routing defineres som routingsstrategier, hvor de mobile enheders bevægelse ændres for at opnå en bedre routing. Et simpelt eksempel på aktiv routing er, at hvis A ønsker at sende en besked til B, så kan en forbindelse opnås ved, at A bevæger sig inden for kommunikationsradius af B. Som modsætning til aktiv routing defineres *passiv routing*, hvor de mobile enheders bevægelse ikke ændres. De ovenstående eksempler på routing protokoller er således passive.

[O'H02] nævner to eksempler på aktiv routing. Den første tilgang til aktiv routing kræver en stor viden om netværkets topologi. Strategien går ud på, at når en pakke skal routes fra A til B, prøver de mobile enheder i forening at konstruere en route fra A til B, hvis en route ikke allerede eksisterer. Denne tilgang har det problem, at mobile enheder, der ikke har nogen part i kommunikationen som udgangspunkt, kan blive tvunget til at ændre retning, for at kommunikationen mellem A og B kan fuldføres. Desuden kræves der en så stor viden om netværkets topologi, at denne strategi næppe er realistisk at overføre til autonome mobile robotter. Den anden tilgang til aktiv routing nævnt i [O'H02] er *message ferry*. Denne strategi involverer en dedikeret mobil enhed, der fungerer som *message ferry*. Denne *message ferry* bevæger sig langs en velkendt route, og mobile enheder, der ønsker at kommunikere, kan så bevæge sig inden for radius af denne *message ferry*. Denne strategi har - udover det oplagte single point of failure - det problem, at "færgen" kun

kan modtage beskeder, når afsendere bevæger sig inden for dens kommunikationsradius. Den kan ikke nødvendigvis aflevere beskeder til modtager, hvis denne ikke befinder sig inden for "færgens" kommunikationsradius på noget punkt på "færgens" route. Denne tilgang kan altså kun anvendes, hvis de mobile enheder er begrænset til at bevæge sig inden for et afgrænset område, som "færgen" så kan patruljere.

De to eksempler på aktiv routing nævnt i [O'H02] har åbenlyse mangler, men selve ideen bag aktiv routing er interessant. Alle routing protokoller i netværk bestående af mobile enheder lider under den meget dynamiske netværkstopologi, men til gengæld er fordelen netop, at enhederne er mobile, og dermed aktivt kan øge muligheden for at kunne kommunikere indbyrdes. Der er dog desværre - til forfatterens kendskab - endnu ikke udviklet nogen strategi til at anvende aktiv routing, som ikke er afhængig af urealistiske antagelser, og dermed er flooding/gossip baseret routing stadig det bedste bud på en routingstrategi i fragmenterede ad hoc netværk.

3.3 Situated kommunikation

De sensorer og aktuatorer, som robotter bruger til at kommunikere med, adskiller sig ikke fundamentalt fra andre sensorer og aktuatorer. Det, der gør kommunikationen mellem robotter speciel, er at ophavet til de fysiske fænomener er andre agenter således, at man kan tillægge de fysiske fænomeners indhold en abstrakt betydning. Et kommunikationssignal til en robot indeholder altså to dele - en *situated* del og en abstrakt del. [Stø01] definerer *situated communication* således:

> "*situated communication is communication where both the physical properties of the signal that transfers the message and the content of the message contribute to its meaning.* " [Stø01]

I modsætning til dette definerer [Stø01] abstrakt kommunikation således:

> "*Abstract communication is communication where the physical signal that transports the message is considered not to have any meaning* " [Stø01]

Abstrakt kommunikation er den mest udbredte kommunikationsform. På internettet interesserer det sjældent brugeren, hvordan beskederne kommer frem, f.eks. om de er overført trådløst eller ej.

Et eksempel, hvor *situated* kommunikation derimod er interessant, er robotter. Robotter benytter sensorer til at opnå viden om omgivelserne, og i denne sammenhæng er et kommunikationssystem en udmærket sensor. For

robotter giver et kommunikationsinterface en ganske bestemt oplysning om omgivelserne, nemlig: "Er der nogen, der vil snakke med mig?". Men der kan også være andre informationer, som robotter kan benytte sig af, i signalets fysiske egenskaber. Styrken af signalet kan give et mål for afstanden til afsenderen, og viden om signalets udbredelse kan for retningsbestemt kommunikation give afsenderens relative orientering.

Situated kommunikation er særdeles almindelig i kommunikation mellem mennesker, men kommunikation mellem computersystemer benytter sig traditionelt ikke af *situatedness*. Kommunikationsprotokoller baseret på OSI-stakken vil abstrahere væk fra signalets fysiske virkelighed på de nederste lag af protokolstakken. Signalet kan således være transporteret gennem mange midlertidige stationer og gennem mange forskellige medier og være længe undervejs, hvilket modtageren af en besked ikke som udgangspunkt kan opdage. Dermed er en eventuel viden om fælles fysisk placering - og dermed også *situatedness* - gået tabt i forsendelsen af beskeden. Denne abstraktion væk fra signalets fysiske udbredelse er ikke noget problem for f.eks. internet-kommunikation, hvor det ønskes, at kommunikationen er *location-transparent* [CDK01]. Dette er ikke tilfældet, når robotter kommunikerer. Der kan ligge en stor værdi i at vide, hvordan en besked er kommet frem. Er kommunikationsmediet for eksempel infrarødt lys, vil det faktum, at en robot modtager et signal, betyde, at afsenderen er i nærheden.

3.4 Situated kommunikation i forbindelse med Ad Hoc netværk

Hvordan kan *situated* kommunikation benyttes i forbindelse med Ad Hoc netværk? I mange kommunikationsprotokoller hører signalets fysik til i det eller de nederste lag. Dette gælder f.eks. protokoller opbygget efter OSI-stakken. Der kan dog opnås flere fordele ved at benytte information om signalets fysik højere oppe i protokolstakken.

Et eksempel, hvor signalets fysiske egenskaber anvendes højt i protokolstakken, er *preemptive routing*, [GAGPK03] og [PAI02]. Ideen i *preemptive routing* er, at mens traditionelle routing protokoller først reagere, når en forbindelse er brudt, så forsøger *preemptive routing* at opdage brud, før de sker. Dette opnås ved at overvåge signalstyrken på de forskellige forbindelser. Inden forbindelsen brydes helt, vil signalstyrken falde. Hvis signalstyrken på denne måde falder, kan den enhed, der opdager det, sende en route error pakke tilbage til afsenderen af pakkerne. Dette giver afsenderen mulighed for at finde en ny route, inden den gamle er brudt. På denne måde benyttes *situated* kommunikation på netværks niveau i protokolstakken. Alternativt til

signalstyrken kan antallet af tabte pakker benyttes som mål for, hvor stabil en forbindelse er. I modsætning til signalstyrken, som befinder sig på det fysiske niveau, så befinder pakketab for den enkelte forbindelse sig på datalink niveau. Hvor signalstyrken som regel ikke er interessant, og dermed ikke konverteres til en digital værdi, så er det ofte nemmere at tilgå pakketabet på en specifik forbindelse.

Et andet eksempel på benyttelse af *situated* kommunikation i netværk er på datalink niveau. *Situated* kommunikation kan benyttes til at opnå større stabilitet af en forbindelse direkte mellem to enheder. Et eksempel på dette kunne være i forbindelse med retningsbestemt kommunikation, hvor en modtager kunne benytte de *situated* egenskaber i et signal til at rette sin modtager ind, så kommunikationen er så stabil som muligt. Denne form for aktiv *link maintenance* har dog sine begrænsninger. Det er ikke altid ønskeligt at bevæge modtageren for at opnå bedre kommunikation. Men specielt i forbindelse med robotter er det ofte muligt at ændre robottens bevægelse således, at kommunikationsforbindelserne bliver mere stabile.

Aktiv *link maintenance* minder på mange måder om *preemptive routing*. De to adskiller sig ved at når forbindelserne brydes, skal der ved *preemptive routing* findes en ny forbindelse mellem afsender og modtager, mens det ved *link maintenance* forsøges at undgå at forbindelserne bliver brudt. I netværk med en stort antal forbindelser er det muligt at finde en alternativ forbindelse, mens det hvis antallet af forbindelse er svært. Dermed er fordelene ved *preemptive routing* størst når der er et tilstrækkeligt stort antal forbindelser til at det er muligt at finde en alternativ route gennem netværket. Hvis antallet af forbindelser i netværket er meget lavt, så vil der være større fordele ved at forsøge at bevare forbindelserne længe nok til, at der kan udveksles tilpas store mængder data.

Ligesom *situated* kommunikation kan anvendes i forbindelse med *link maintenance*, så kan det også anvendes i forbindelse med *neighbour discovery*. Hvis en enhed modtager kommunikation direkte fra en anden enhed, så betyder det at denne enhed er i nærheden. Denne information kan anvendes i en passiv *neighbour discovery* algoritme, og muligvis spare behovet for aktiv *neighbour discovery*, hvor der kommunikeres specielt med hensigt på *neighbour discovery*. Informationer om signalets fysik kan også finde anvendelse i forbindelse med *neighbour discovery*. For eksempel kan det være nyttigt at kende retningen og afstanden til en afsender. Retningen og afstanden kan findes ved hjælp af sensorer, der måler signalets styrke i forskellige retninger.

Der er altså mange forskellige steder i en trådløs kommunikationsprotokol, hvor *situated* kommunikation kan finde anvendelse.

4 Multirobotsystemer

For at det overhovedet er interessant at beskæftige sig med grupper af robotter, skal man kunne forestille sig situationer, hvor to eller flere robotter kan løse en opgave bedre end en enkelt robot.

Et eksempel på opgaver, hvor flere robotter er en fordel, er de tilfælde, hvor en enkelt robot slet ikke kan løse opgaven alene. Dette kunne være en opgave, der kræver, at der bliver trykket på to knapper samtidigt eller inden for en vis tidsmargin på fysisk adskilte lokationer [BP02]. Denne opgave vil kræve mindst to robotter, der er i stand til at koordinere deres handlinger.

Det er ikke kun i forbindelse med opgaver, som slet ikke kan løses af en enkelt robot, at en gruppe af robotter kan være fordelagtig. Der findes mange opgaver, som robotter med fordel kan samarbejde om, således at en koloni af robotter kan løse en opgave hurtigere, bedre, billigere eller mere sikkert end en enkelt robot.

En oplagt fordel ved en gruppe af robotter er robusthed. Et system med en gruppe af robotter kan opnå en stor fejltolerance, ved at besidde redundante egenskaber. Hvis en robot i et sådant system bryder sammen, kan dens opgaver overtages af en anden robot med redundante egenskaber. Robusthed er som med enkeltrobotsystemer et særdeles vigtigt designkriterium. Da robotsystemer skal kunne handle autonomt, er det nødvendigt, at fejltolerancen i systemet er stor. Man kunne forestille sig, at NASA sender en gruppe af robotter til Mars. Hvis der i denne gruppe er robotter, der bryder sammen, vil missionen alligevel kunne gennemføres og pengene ikke være spildt.

En anden fordel ved grupper af robotter er evnen til samtidigt at arbejde på flere delopgaver ved hjælp af en *divide and conquer* strategi. Hvis de problemer, robotterne skal løse, naturligt lader sig dele op i flere delproblemer, kan disse delproblemer deles ud til forskellige robotter, og den overordnede systemydelse kan dermed øges betragteligt. Man kan for eksempel forestille sig, at en gruppe af robotter skal kortlægge et område. Hvis alle robotter starter i samme startpunkt, vil det være en fordel, at de hver især vælger et underområde at kortlægge for så til sidst sammen at danne det fuldstændige kort ud fra de enkelte robotters kort[1].

De sidste fordele ved en gruppe af robotter er, at de kan drage fordel af distribueret sansning og handling. En gruppe af robotter kan ved at kommunikere dele sensordata. Dermed er den enkelte robot i besidelse af langt flere oplysninger om sine omgivelser. Dette kan give mulighed for at spare sensorer på den enkelte robot. For eksempel kunne man forestille sig to robotter

[1]Det er på ingen måde et trivielt problem at kombinere kort fra forskellige robotter, dette er blot tænkt som et tankeeksperiment

samarbejde om at flytte en coladåse. Den ene robot har en arm, der kan løfte coladåsen, samt sensorer til at håndtere gribningen af dåsen. Den anden robot har et kamera, som kan se coladåsen og løfte-robotten. Mens løfte-robotten kan koncentrere sig om at holde fast i dåsen og tage mod anvisninger, kan kamera-robotten fortælle løfte-robotten, hvordan den skal bevæge sig i forhold til skraldespanden, og hvornår dåsen kan slippes. Distribueret sansning og handling kan give anledning til simplere - og dermed billigere - robotter.

De mange fordele ved at distribuere sensorer, aktuatorer og kontrol, gør grupper af robotter til et interessant område, men der opstår flere problemer når opgaveløsningen skal distribueres. Hvordan samarbejder robotter optimalt? Er kommunikation en nødvendighed? Hvad skal der kommunikeres om? Hvordan fordeler robotterne opgaverne imellem sig?

4.1 Karakterisering af multirobotsystemer

Design af robotter er i høj grad dikteret af robottens fysiske virkelighed. Derfor er begreberne *situatedness* og *embodiment* centrale i designet af robotter. På samme måde kan man ikke se bort fra robotternes fysiske virkelighed, når man designer multirobotsystemer. Desuden vil begrebet *emergence* blive endnu mere centralt for multirobotsystemer. Dette skyldes, at robotternes samspil kan give mulighed for effekter, som går ud over, hvad de enkelte robotter kan præstere alene.

[DJM02] beskriver en række forskellige kriterier, som kan benyttes til at karakterisere robotgrupper. De kriterier, der nævnes i [DJM02], er:

Kommunikationsradius: Dette er et kontinuum, der spænder fra ingen kommunikation til i praksis uendelig kommunikationsradius.

Kommunikationstopologi: Dette er en beskrivelse af robotternes indbyrdes kommunikationshierarki. Robotterne kan kommunikere ved hjælp af broadcast eller adresseret kommunikation, men organisering i træer eller grafer, hvor robotterne kun kommunikere med naboer, kan også forekomme.

Kommunikationsbåndbredde: Dette er et kontinuum, der beskriver hvor dyr, kommunikationen er. Ved trådløs kommunikation skal flere robotter dele samme kommunikationsmedie, og i tilfælde hvor kommunikationen er dyr, er det nødvendigt at minimere mængden af kommunikation.

Antal robotter: Dette er et kontinuum spændende fra tilfældet med en enkelt robot til tilfældet med i praksis uendeligt mange robotter.

Gruppens dynamik: Dette er et kontinuum der beskriver hvor dynamisk gruppen er - det vil sige i hvor høj grad robotternes indbyrdes forhold såsom placering og naboskab ændrer sig.

Gruppens homogenitet: Dette er et kontinuum, der beskriver, hvor forskellige robotterne i gruppen er.

Den enkelte robots beregningskraft: Dette kriterium beskriver hvilken model, der kan anvendes til at beskrive robotternes beregningskraft. Der kan være tale om, at robotterne kan beskrives som f.eks. endelige automater eller ækvivalent med en Turing maskine. Langt de fleste robotter er produceret med en eller anden generel cpu og er dermed Turing maskine ækvivalent. Alligevel kan robotter programmeres til at opføre sig som f.eks. endelige automater.

Det er altså ifølge [DJM02] disse ting, der skal overvejes når robotgrupper designes. De tre af de syv punkter handler om kommunikation. Dermed ligger det implicit i denne taxonomi, at kommunikationen mellem robotterne er en vigtig faktor i karakteriseringen af robotgrupper. Af de andre punkter omhandler de tre gruppens sammensætning - antallet, homogenitet og dynamik. Det sidste punkt omhandler den enkelte robots beregningskraft. Det er ikke oplagt, at dette punkt hører til i en taxonomi for robotgrupper. Grunden til, at punktet alligevel er taget med, er, at en gruppe af robotter, der hver især har begrænset beregningskraft, kan opnå en større beregningskraft som gruppe:

> "*An unbounded number of robots $\{A_i\}$ whose processing abilities can be modelled individually as finite automata with the ability to communicate their state to their neighbors may simulate an arbitrary Turing machine*" [DJM02]

Der kan argumenteres for, at taxonomien i [DJM02] kun indeholder tre dimensioner og ikke syv, som punktopstillingen giver udtryk for. De tre dimensioner er kommunikation, gruppens sammensætning og den enkelte robots beregningskraft. En simplere taxonomi, der udnytter dette, bliver beskrevet i [SV02]:

> "*Although there are many possible ways to devide MAS [Multi Agent Systems], the survey is organized along two main dimensions: agent heterogeneity and amount of communication among agents*" [SV02]

De to taxonomier er ikke nødvendigvis modstridende. De varierer hovedsagligt i detaljeringsgrad, hvor [DJM02] har flere detaljer sammenlignet med [SV02].

4.2 Karakterisering af kommunikation

Behovet for kommunikation afhænger i høj grad af opgaven, omgivelserne og robottens evne til at opfatte omgivelserne. [Ark98] har følgende bud på hvornår det er en fordel eller en nødvendighed for robotter at kommunikere:

> "*Communication improves performance significantly in tasks involving little implicit communication (foraging and comsuming). In the grazing task, robots leave evidence of their passage, since the places they visit are modified. This fact is observable by other robots. These types of communication are referred to as implicit, since they require no deliberate act of transmission.*" [Ark98]

Det er altså en fordel at kommunikere, hvis det ikke er muligt for robotter at kommunikere gennem omgivelserne. Kommunikation, der foregår ved at robotterne i forbindelse med løsning af en opgave ændrer omgivelserne, defineres som *implicit kommunikation* [Ark98]. I modsætning hertil defineres eksplicit kommunikation som en situation, hvor robotterne specifikt benytter et kommunikationsmedie med det formål at kommunikere med hinanden. Behovet for eksplicit kommunikation er uløseligt forbundet med omgivelserne og robotternes evne til at opfatte ændringer i omgivelserne. Eksplicit kommunikation er således vigtig, når robotternes evne til at opfatte ændringer i omgivelserne er lille pga. begrænsede sensorer - som for eksempel på LEGOs RCX - eller når ændringer i omgivelserne ikke er mulige.

Det kan for eksempel tænkes, at en gruppe af robotter bliver sendt til Mars. Her påvirker robotterne omgivelserne ved at afsætte hjulspor, som de andre robotter kan se, men disse hjulspor er ikke en blivende ændring. Støvstorme vil overdække dem, det er derfor ikke muligt at påvirke omgivelserne permanent, og eksplicit kommunikation kan blive en nødvendighed. Hvis rejsen derimod gik til månen, hvor der ikke er støvstorme, kan hjulsporene detekteres i meget længere tid, og eksplicit kommunikationen kan dermed være overflødig. Således kan robotternes evne til at ændre omgivelserne have indflydelse på behovet for eksplicit kommunikation.

Der kan være flere grunde til, at robotter ønsker at kommunikere. Disse grunde giver anledning til at kommunikation kan spille forskellige roller i robotgruppens koordinering. [Ark98] nævner tre forskellige roller, som kommunikation kan spille i en robotgruppe:

Synkronisering: Hvis en opgave kræver at delopgaver bliver løst i en bestemt rækkefølge eller samtidigt er det nødvendigt for robotterne at kommunikere om dette.

Vidensudveksling: Hvis forskellige robotter er i stand til at sanse forskellige ting i omgivelserne, kan de kommunikere sammen for dermed at dele denne viden.

Forhandlinger: Hvis opgaven kan deles, skal det afgøres, hvilken robot der skal tage en bestemt opgave. Det kan være en fordel for robotter at kommunikere for at opnå en fordeling af opgaverne.

Synkronisering og forhandling er begge direkte forbundet til fordelingen af opgaver imellem robotterne, mens vidensudveksling gør robotter i stand til at dele sensordata og dermed viden om omgivelserne. Kommunikationens rolle er også vigtig at overveje, når kommunikationen skal designes. For eksempel er det muligt, at robotter, der ønsker at synkronisere deres handlinger, har et behov for tovejskommunikation for at garantere, at modparten har modtaget beskeden, mens en robot, der ønsker at dele sensordata, blot kan broadcaste disse uden nødvendigvis at have nogen viden om, at de bliver modtaget af ingen, en eller mange modtagere.

Rækkevidden af kommunikation er også en vigtig faktor. Hvis man benytter kommunikation med en meget lille rækkevidde, begrænser man robotternes mulighed for at udveksle informationer. Men hvis man derimod benytter kommunikation med en meget stor rækkevidde, vil der opstå problemer. Hvis mange robotter sender information, vil modtagerene ikke kunne følge med eller kommunikationsmediet kan blive en flaskehals for systemet. Desuden er det sjældent, at informationer fra en robot meget langt væk er vigtige for en robots arbejde på en lokal opgave. Kommunikationsrækkevidden skal designes således, at hver robot gennem kommunikation kan få præcis de nødvendige oplysninger. Rækkevidden af kommunikationen er bestemt af det kommunikationsmedie, der vælges, samt afsenderens sendestyrke.

Kommunikationensbåndbredde er også en vigtig faktor. Ligesom rækkevidden af kommunikationen er båndbredden i høj grad afhængig af det kommunikationsmedie, som anvendes.

Indholdet af eksplicit kommunikationen er også en vigtig faktor. Eksplicit kommunikation kan foregå på mange forskellige niveauer, og indholdet vil afspejle niveauet. [JZ02] definerer tre forskellige niveauer, som kommunikation kan foregå på: *Iconic*, *Indexical* og *Symbolic*:

> "*Iconic representation is by physical similarity to what it represents [...] Indexical reference represents a correlation or association between icons [...] The third level og representation is symbolic. A symbol is a relationship between icons, indices, and other symbols* " [JZ02]

Begrebet subsymbolsk kommunikation benyttes om kommunikation, der udelukkende benytter niveauerne *iconic* og *indexical*. Indholdet af kommunikationen kan altså være på flere abstraktionsniveauer.

4.3 Opgavedeling

Fordelene ved multirobotsystemer kommer bedst til udtryk, når robotterne er i stand til at koordinere deres handlinger og således påtage sig forskellige delopgaver.

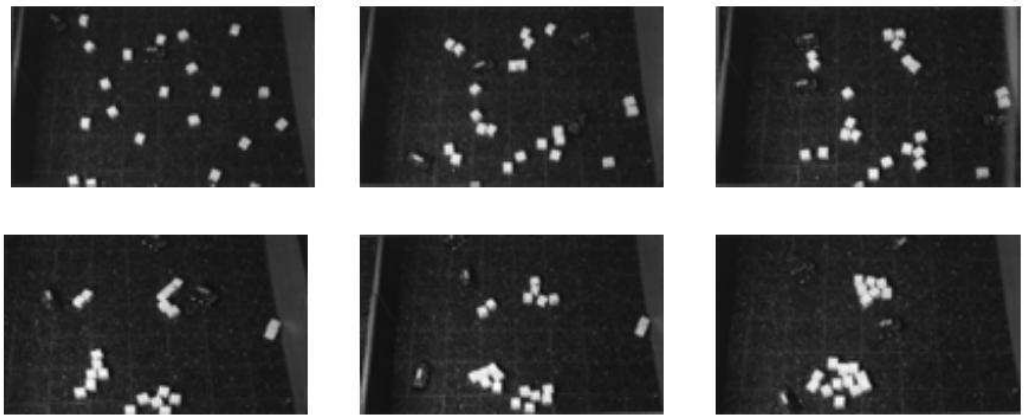
Opgavedeling mellem robotter er vigtig i forbindelse med opgaver, der kan opdeles i flere underopgaver, så robotgruppen kan anvende en *divide and conquer* strategi. Målet er at opnå den optimale fordeling af robotter på forskellige opgaver således, at flest mulige opgaver bliver løst så hurtigt som muligt.

Fordelingen af delopgaver mellem robotter kan beskrives som et *dynamisk ressourceallokeringsproblem*. Det vil sige at gruppen af robotter har nogle ressourcer og en række opgaver, der skal løses i dynamiske omgivelser. Problemet består i at fordele ressourcerne til de enkelte opgaver på en optimal måde. Dette problem er ækvivalent med det NP-komplette *conjunctive planning problem* og dermed NP-komplet [GM02]. At problemet er NP-komplet gør, at det ikke kan forventes at robotter kan løse dette problem optimalt, og samtidig bevare evnen til at reagere på ændringer i omgivelserne, med mindre problemet er begrænset i størrelse. Ligesom med enkeltrobotsystemer er det altså nødvendigt at benytte en anden tilgang til problemet, hvis man ønsker en stor udnyttelsesgrad af robotternes ressourcer til at løse de givne problemer.

Den første forudsætning for en god fordeling af ressourcerne er, at robotterne er enige om hvilke opgaver, den enkelte robot påtager sig. Med denne enighed kan det undgås, at robotter med redundante egenskaber påtager sig den samme opgave.

En god fordelin af opgaver problem kan kun løses ved at robotterne kommunikerer. Kommunikationen kan være implicit, for eksempel kan robotterne have sensorer, der direkte giver dem mulighed for at opfatte, hvilke opgaver der løses af hvilke robotter. Alternativt kan kommunikationen være eksplicit, hvor robotterne benytter sig af et fælles kommunikationsmedium til at forhandle opgavernes fordeling.

Der er forsøgt mange forskellige løsninger for at opnå enighed mellem robotter. Nogle eksempler, hvor forskellige teknikker bliver benyttet er beskrevet i afsnit 4.4.



Figur 4.1: Didabots, oprydning [Pfe96]

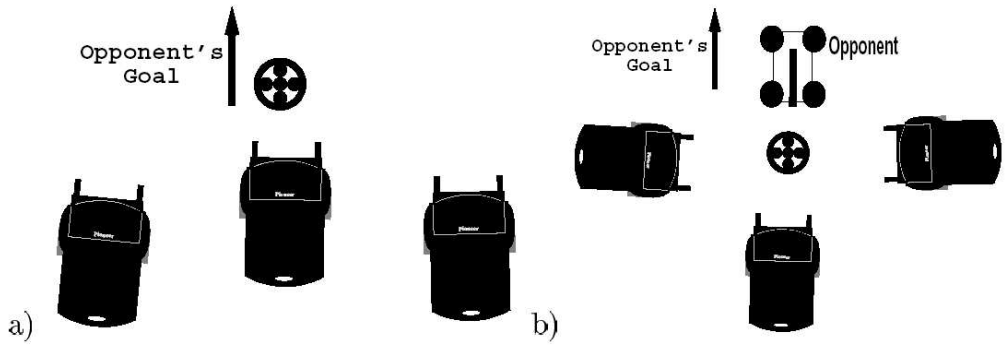
4.4 Eksempler på robotgrupper

4.4.1 Emergent group behavior

Et simpelt scenarie, hvor flere robotter løser en opgave i fællesskab, er beskrevet i [Pfe96]. [Pfe96] beskriver et eksperiment, hvor et antal didabots bevæger sig rundt i en arena fyldt med firkantede klodser, figur 4.1. Opgaven for didabots er at samle klodserne i større grupper[2]. Robotterne har ikke nogen muligheder for eksplicit kommunikation, og de enkelte robotter har ikke nogen viden om, at de ikke er alene. Alle robotter har samme adfærd, og en enkelt robot kunne løse opgaven alene - det ville blot tage længere tid. Robotterne samarbejder simpelthen ved at påvirke omgivelserne på den samme måde. Dette er et eksempel på at robotter kan benytte omgivelserne til at kommunikere. Robotterne behøver hverken at udsende signaler eller have eksplicit viden om andre robotter, de kan blot påvirke omgivelserne på en måde, som igen påvirker de andre robotters adfærd. Denne form for samarbejde kaldes *emergent*.

Et andet eksempel på samarbejde uden eksplicit kommunikation findes i [WM01]. [WM01] beskriver et hold af robotter, der spiller fodbold. De enkelte robotter har kun viden om bolden, målet og forhindringer. Der skelnes ikke mellem modstandere og holdkammerater. Robotterne kommunikerer ikke med hinanden, men opnår alligevel en kompleks adfærd som gruppe. Ved

[2] I artiklen beskrives, at robotterne er designet til at undgå objekter, og robotternes fysiske udformning bevirker, at robotternes oprydningsadfærd fremkommer som et samspil med omgivelserne - ikke som et mål for robotten. Eksemplet kan dog også benyttes til at beskrive samarbejde uden, robotterne er vidende om dette.



Figur 4.2: Et emergent robot-fodbold hold

hjælp af blandt andet tiltrækning til bolden og frastødning fra forhindringer opstår formationer, som hjælper holdet til både at forsvare, figur 4.2b, og angribe, figur 4.2a. Dette er et eksempel på at robotter, uden kommunikation eller viden om hinanden kan opnå en forbløffende høj grad af samarbejde.

Det primære problem med at benytte emergence til at opnå samarbejde mellem robotter er, at robotternes samarbejde er designet specifikt til løsningen af et bestemt problem. Emergence er altså ikke nogen generel løsning til samarbejde mellem robotter. Desuden udnytter de fleste systemer baseret på emergent samarbejde en høj grad af redundante resourcer, og da der - som regel - ikke er nogen mulighed for at forhindre, at flere robotter arbejder på samme opgave - og dermed kommer i vejen for hinanden - så er udnyttelsen af ressourcerne ikke optimal.

4.4.2 Blackboard kommunikation

En måde at lade robotter kommunikere på er at give dem adgang til et delt lager af viden. Dette lager kaldes et blackboard [Woo02]. Hver robot fungerer som *knowledge source* og skriver sin viden på det centrale *blackboard*. Dette giver flere robotter adgang til at benytte det samme centrale lager af viden og adgang til parallel indsamling af viden fra flere agenter.

Problemet med blackboard kommunikation er, at mange agenter skal kunne skrive til det samme lager af viden. Dette giver concurrency problemer. Således skal robotternes tilgang til det centrale lager af viden begrænses sådan, at kun én agent kan skrive til lageret på et givet tidspunkt. Problemet kan løses - for eksempel med semaforer - men det sætter begrænsinger for systemets parallelitet.

[ØMS01] beskriver et multirobotsystem, der anvender blackboard kommunikation - dog med visse nødvendige modifikationer primært for at tage højde for den ustabile kommunikation mellem robotter. I [ØMS01] udsender hver robot informationer om sin egen tilstand. Denne information opsamles af andre robotter, der gemmer det på sit eget blackboard. Hvis kommunikationen virker perfekt, vil hver agent have den samme information. Derfor vil systemet være ækvivalent med et klassisk blackboard system. Ideen er nu, at hvis hver robot har adgang til den samme information og udfører samme algoritme, så vil de nå frem til samme resultat. I [ØMS01] er hver robot udstyret med en mikrofon. Denne benyttes til at lytte efter alarmer, der har forskellige frekvenser. Informationen på blackboard'et består i en tabel, hvor der for hver robot er værdier for styrken af de forskellige alarmsignaler samt information om, hvorvidt robotten er i gang med at løse en opgave. Det antages, at den agent, som hører alarmen højest, er bedst egnet til at slukke for alarmen. Da alle robotter har den samme viden gennem blackboard'et, og den robot, der hører lyden stærkest og ikke er i gang med andre opgaver, påtager sig at slukke alarmen, vil en robot tage sig af hver alarm. På denne måde kan agenterne koordinere deres arbejde således, at præcist én agent er tilknyttet hver opgave.

Denne modificerede tilgang af blackboardalgoritmen lider dog under problemer med at vedligeholde det samlede informationslager. Algoritmens effektivitet hænger på, at alle robotter har adgang til den samme information. Hvis kommunikation svigter, er dette ikke tilfældet, og flere robotter kan påtage sig samme opgave. Da det fælles informationslager er implementeret lokalt på hver robot, der så opdaterer, dette når ny information ankommer fra andre robotter, vil den manglende kommunikation ikke forårsage et totalt sammenbrud. Derimod vil hver robot i dette tilfælde kun være i besiddelse af sin egen information, og dermed vil der ikke være nogen koordinering. Hver robot vil køre mod den alarm, som lyder højest.

Blackboard kommunikation er designet til, at kommunikationen spiller vidensdelingsrollen. Det er mindre oplagt at benytte *blackboard kommunikation* til synkronisering og forhandling.

4.4.3 Impatience og acquiesence

Alliance arkitekturen [Par98] benytter sig af *impatience* og *acquiesence* til at afgøre, hvilke behavior producing modules der er aktive i en given agent. Måden, hvorpå opgaveløsningen koordineres, bygger på en række af antagelser. Heraf er de to vigtigste:

- Robotter kan detektere effekten af egne handlinger med en sandsynlighed større end 0.

- Robotter kan detektere effekten af andre robotters handlinger, hvor robotten har redundante egenskaber med en sandsynlighed større en 0.

Disse to antagelser benyttes til at implementere to former for *motivational behaviors*, *impatience* og *acquiesence*. *Impatience* gør robotter i stand til at overtage opgaver fra andre robotter, hvis robotten ikke er tilfreds med fremskridtet. *Acquiesence* gør en robot i stand til at opgive en opgave, hvis den ikke opnår tilstrækkeligt fremskridt.

Denne brug af motivational behaviors muliggør et fuldt distribueret system. Som udgangspunkt kræves der ikke kommunikation for, at robotter kan fordele opgaver ved hjælp af *impatience* og *acquiesence*. Det kræver blot, at de to antagelser er opfyldt. Problemet med denne tilgang er netop, at de to antagelser er særdeles stærke, og specielt den sidste er svær at overholde i praksis. En robots mulighed for at detektere ændringer i omgivelserne er begrænset af dens sensorer. Hvis en robot skal have mulighed for at detektere sine egne fremskridt, skal den altså besidde sensorer til dette. Man kan konstruere robotter, der er i besiddelse af sensorer, så de kan detektere effekten af egne handlinger, men den anden antagelse er stadig særdeles svær at overholde. Da en robot er afhængig af sensorer, vil dens opfattelse af omgivelserne være lokal. Den vil kun kunne opfatte ændringer i omgivelserne, hvis disse sker tæt på robotten. Derfor er det svært for en robot at detektere effekten af andre robotters handlinger medmindre, disse er tæt på. Problemet med at overholde antagelserne bliver yderligere kompliceret af, at sandsynlighederne for at detektere egen eller andres manglende fremskridt helst skal være stor - jo større sandsynlighed des mere brugbart er systemet.

Hvis antagelserne er overholdt, vil motivational behaviors kunne benyttes til at implementere et fejltolerant multirobotsystem. Hvis en robot fejler, vil de andre robotter opdage dette, deres impatience vil vokse, og til sidst vil de overtage opgaven fra den fejlede robot.

[Par98] løser problemet med at overholde antagelserne ved, at robotter udsender deres eget fremskridt til andre robotter. Denne information kan nu benyttes af andre robotter sammen med sensordata til at bestemme fremskridtet. Dette gør det lettere at overholde antagelserne, men vil også gøre systemet mindre fejltolerant. Hvis en robot ikke selv kan detektere sit manglende fremskridt, vil den fortsætte med at udsende beskeder om fremskridt. De andre robotter vil benytte denne information til muligvis at antage, at opgaven bliver løst. For at undgå dette problem er det vigtigt at robotterne har sensorer, der ud over de udsendte beskeder kan detektere, at en robot har fejlet.

I *Alliance* benyttes kommunikationen til at dele viden om fremskridt i løsning af opgaverne. Denne information kan benyttes af andre robotter til

at overtage opgaver, hvor de ikke er tilfredse med fremskridtet. Kommunikationen har altså vidensdelingsrollen i *Alliance*.

4.4.4 Cross-Inhibition

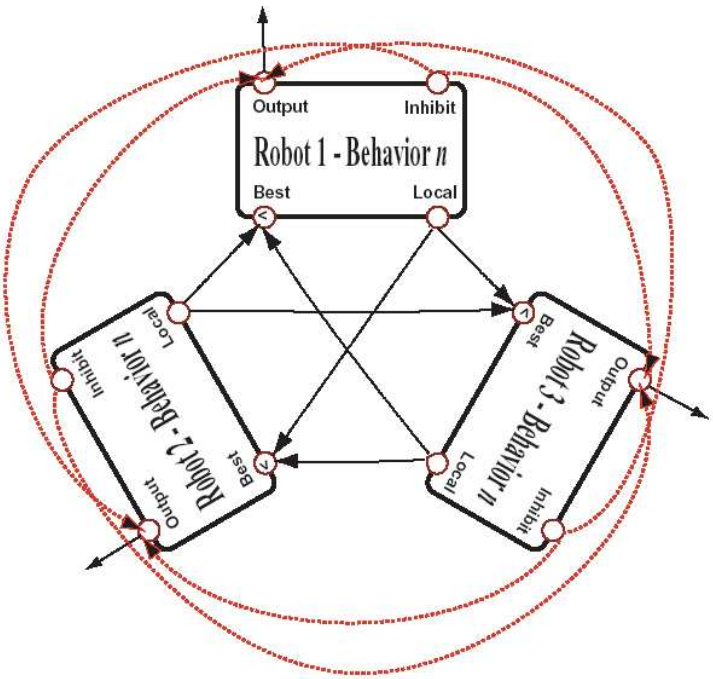
Den oprindelige *subsumption* arktitektur består af et netværk af moduler, der kommunikerer ved at sende *inhibit*- og *suppression* beskeder til hinanden. Der er ikke noget i vejen for, at en *behavior* eller *subsumption* baseret robot kan været distribueret over flere processorer eller endda over flere forskellige adskilte fysiske enheder.

En udvidelse af den *behavior* baserede tilgang er dog nødvendig for at understøtte distribuering over flere robotter. Dette skyldes, at kommunikation mellem robotter ikke er stabil, da beskeder kan tabes eller bliver ændret undervejs. For at opveje dette, er det nødvendigt at overveje, hvordan de enkelte adskilte fysiske robotter skal fungere, hvis de ikke har forbindelse med andre.

Et forsøg på at udvide den *behavior* baserede tilgang til at gælde en flok af kommunikerende robotter er *Cross-Inhibition* [WM01]. *Cross-inhibition* - som beskrevet i [WM01] og [Wer00] - benytter teknikken *Broadcast of Local Eligibility*, BLE. BLE involverer samarbejde mellem ligeværdige *behavior producing modules* mellem de enkelte robotter. Hver robot beregner lokalt en værdi for egnetheden til at løse en given opgave. Denne værdi sendes til tilsvarende *behavior producing modules* i de andre robotter. Den robot med den højst udregnede egnethed vil påtage sig opgaven ved periodisk at udsende et *inhibit* signal til de andre robotter, figur 4.3. Den beskrevne proces kaldes *cross-inhibition*.

Cross-inhibition indeholder implicit understøttelse af heterogene robotter. Således behøver robotterne i flokken ikke nødvendigvis at være ens eller indeholde samme *behavior producing modules*. Hvert *behavior producing module* koordinerer direkte med de tilsvarende *behavior producing modules* i andre robotter.

Cross-inhibition har også indbygget en høj grad af robusthed. Da *inhibit* signaler skal afsendes periodisk, vil en ny robot automatisk overtage, hvis den robot, der er i gang med opgaven, lider et kommunikationssvigt. Desuden vil andre robotter overtage en opgave, hvis de udregner en højere egnethed end den robot, der har opgaven.
Det er langt fra altid, at en given opgave kan eller bør løses af en enkelt robot. Dette udgør et problem for *cross-inhibition*, og i [WM01] foreslås problemet løst ved, at hver robot indeholder et antal ens *behavior producing modules* svarende til antallet af robotter, der ønskes tilegnet den givne opgave. Hvis der skal tre robotter til at skubbe en kasse, vil alle robotter altså skulle



Figur 4.3: Cross-inhibition [Wer00]

indeholde tre kopier af et *push behavior producing module*. Denne løsning er langt fra elegant.

De egnethedsdværdier, som udregnes i *cross-inhibition*, minder meget om de motivationsværdier, der er benyttet til *behavior selection* på enkelte robotter, afsnit 2.5.2. *Cross-inhibition* benytter således kommunikationen til at udsende egnethedsværdierne til fysiske adskilte robotter således, at en unik behavior på en bestemt robot bliver aktiv. På denne måde er *cross-inhibition* blot en distribuering af *behavior selection* over flere fysiske robotter.

Som figur 4.3 antyder, kan der hurtigt blive behov for en stor mængde beskeder, og dermed kan kommunikationens båndbredde blive en flaskehals. Dette bliver til dels afhjulpet, hvis der kan anvendes broadcast kommunikation, så hvert *inhibit*-signal kun skal afsendes én gang. Variablene i regnestykket er antallet af robotter, N, antallet af *behavior*-grupper, B, det antal genberegninger af motivationsværdier, der skal sendes pr sekund, F, og størrelsen af den enkelte pakke P. F gange i sekundet skal der udsendes $(N \times M + 1 \times M) \times P = (N + 1) \times M \times P$, da hver af de N robotter skal udsende en egnethedsværdi for hver af de M *behavior*-grupper. Derefter skal der sendes en *inhibit*-besked af vinderen. Dermed Skalerer kravet til kommu-

nikationsbåndbredden som følgende:

$$Min_{bandwith} = (N + 1) \times P \times M \times F \frac{bits}{s} \tag{4.1}$$

Dette bliver i [WM01] benyttet til at udregne, at et $2\frac{Mb}{s}$ netværk kan håndtere 50 robotter, som hver indgår i 50 *behavior*-grupper med en besked pr. sekund. På LEGOs RCX kan sendes $2400\frac{bit}{s}$, og hvis man antager, at en *inhibit*-pakke er 10 bytes inklusive header, giver dette en teoretisk kapacitet på $\frac{2400}{80}$ beskeder pr sekund = 30 beskeder pr sekund, altså en maksimalgruppe på 5 robotter med 5 behaviorgrupper og 1 genberegning af egnethed pr. sekund. For at anvende *cross-inhibition* skal man altså enten have en lille gruppe af robotter eller stor tilgængelig kommunikationsbåndbredde.

Cross-inhibition benytter kommunikationen til at afgøre hvilken robot, der skal påtage sig en given opgave. Kommunikationen spiller altså ved *cross-inhibition* forhandlingsrollen. *Cross-inhibition* kan ikke umiddelbart bruges til at dele viden mellem robotter.

4.4.5 Auktioner

En anden metode til fordeling af opgaver mellem robotter i en gruppe er beskrevet i [ZSDT02] og [GM02]. Denne tilgang er blevet beskrevet som henholdsvis *Market Economy* [ZSDT02] og *Auction* [GM02]. Der er forskelle i beskrivelserne, men lighederne gør, at de bedst beskrives sammen.

Auction er et centralt begreb i begge tilgange. En auktion er en forhandlingsmekanisme, hvor parterne er en udbyder og en eller flere bydere. Auktionen lader udbyderen finde en unik vinder blandt byderne. For at kunne byde på en auktion er det nødvendigt at kunne udregne en egnethed. Til dette skal der benyttes en metrik. Denne metrik udsendes i [GM02] sammen med udbuddet af auktionen.

Opgaver fordeles mellem robotter ved hjælp af auktioner. En auktion består af 5 trin [GM02]:

1. Task announcement. En agent udbyder en opgave til auktion.

2. Metric evaluation. Hver robot udregner sin egen egnethed til at løse den udbudte opgave.

3. Bid submission. Hver robot sender sin udregnede egnethed tilbage til udbyderen.

4. Close of auction. Efter et stykke tid lukker udbyderen for bud, og tildeler vinderen opgaven i et bestemt tidsrum.

5. Progress monitoring, contract renewal. Udbyderen overvåger fuldførelsen af opgaven. Hvis der sker tilfredsstillende fremskridt vil udbyderen sende en renewal besked til vinderen. Dette vil ske indtil opgaven er udført. Hvis der derimod ikke sker tilfredsstillende fremskridt vil udbyderen undlade at sende en renewal besked. Dermed vil vinderen af auktionen ophøre med at løse opgaven, og udbyderen annoncerer en ny auktion, hvor den oprindelige vinder er udelukket fra at deltage.

Auktioner er en løsning, der lader hver agent udregne sin egen egnethed til at løse en given opgave. Den agent med den største egnethed tager så opgaven. Tidsbegrænsningen på opgaverne giver en fejltolerance i systemet således, at hvis en agent fejler, vil den ikke få fornyet sin kontrakt, og en anden agent vil overtage. Auktioner understøtter ikke i sig selv opgaver, hvor flere robotter løser den samme opgave, for eksempel box-pushing. Dette løses ved at lade udbyderne opdele opgaverne på en passende måde og udbyde de enkelte dele til auktion.

[ZSDT02] benytter sig også af auktioner som den primære forhandlingsmekanisme mellem robotter. [ZSDT02] benytter auktioner til at koordinere en gruppe af robotter, der skal kortlægge et ukendt område. En central agent belønner robotter, der leverer ny viden om kortet. Hver agent indeholder to funktioner. Den første er en cost-funktion, C, der udregner omkostningerne ved at løse en opgave (besøge et bestemt sted på kortet) som funktion af de ressourcer, der skal bruges. Den anden er en revenue-funktion, R, der udregner værdien af at besøge et bestemt sted på kortet som funktion af ny viden om kortet. Disse funktioner bruges til at udregne profitten, P, ved at løse en opgave. Denne udregnes simpelt som P=R-C. Profitten benyttes til at udregne, det bud som en robot skal afgive, når et målpunkt udbydes til auktion. Hver robot udregner et bud, B, hvor P_s er udbyderens udregnede profit, mens P_i er byderens vurdering af opgaven. α er en konstant, som kan varieres alt efter, hvordan gevinsten ved salget skal fordeles mellem udbyder og byder:

$B_i = P_s + \alpha \times (P_i - P_s)$

Den sælgende agent udsender i forbindelse med auktionen en minimumspris P_s. Hvis ingen robot udregner et bud over denne værdi, sælges opgaven ikke. Dette benyttes i følgende algoritme:

1. Hver robot genererer en række nye målpunkter på kortet[3]. Hvis robotten kan kommunikere med den centrale agent, undersøger den, om målpunkterne er nye for hele gruppen.

[3]Der kan benyttes en række forskellige strategier til generering af målpunkterne.

2. Hver robot indsætter de genererede målpunkter ét af gangen på den placering, der forlænger turen mindst.

3. Hver robot forsøger at sælge hver enkelt opgave ved hjælp af en auktion, som beskrevet ovenfor[4].

4. Hvis en eller flere robotter byder mere end en fastsat minimumspris, får den robot, der har budt højest på opgaven.

5. Når alle auktioner er lukket, begynder hver robot sin tur til det første målpunkt.

6. Når en robot når frem til et mål, vil den generere nye målpunkter og udbyde disse til auktion.

Denne tilgang giver også en vis fejltolerance. Hvor fejltolerancen i [GM02] opnås ved at anvende specifikke timeouts, opnås fejltolerancen i [ZSDT02] ved, at opgaverne (målpunkterne) genereres dynamisk, og hvis de går tabt sammen med en robot, vil de før eller siden blive generet igen. Den anden forskel på [GM02] og [ZSDT02] er, at udbyderen i [GM02] efter auktionen stadig overvåger opgavens fuldførelse. Udbyderen videregiver altså ikke opgaven til en anden robot. Den uddelegerer blot delopgaverne til underordnede robotter, mens robotterne i [ZSDT02] afgiver hele opgaven til en ligeværdig robot.

Man kan forestille sig mange forskellige finjusteringer for auktioner - de beskrevne her er *one-shot first-price sealed-bid auctions* [Woo02]. Det kan for eksempel gøre en forskel, om budene er offentlige eller skjulte for andre bydere. Man kan derfor tro, at forskellige auktionstyper vil give forskellige resultater i et multirobotsystem. Dette er også sandt, men hvis man antager, at alle agenter byder præcis den værdi som den udbudte vare repræsenterer for dem, vil resultatet af alle de forskellige auktionstyper være identisk [Woo02]. Derfor kan man - hvis man antager "ærlige" agenter - benytte sig af den simplest mulige auktionstype - *one-shot first-price sealed-bid* - og derved reducere kompleksiteten ved at benytte auktioner.

Ligesom *cross-inhibition* minder auktioner om distribuerede motivationsværdier. Auktioner kræver $N+1$ beskeder, hvor N er antallet af bydere plus udbyderen. Da hver udbudte opgave svarer til en behavior gruppe i *cross-inhibition*, skal der i værste fald udsendes $(N+1) \times M$ beskeder hver gang opgavetildelingerne skal fornyes. Der er dog flere ting, der for auktioner kan begrænse antallet af nødvendige beskeder. For det første skal en opgave kun

[4]Hvis en anden robot allerede har besøgt det område, der bliver udbudt til auktion, vil den informere udbyderen om dette

udbydes, hvis udbyderen ikke er tilfreds med den robot, der har vundet opgaven. Hvis robotten, der vinder opgaven, løser den uden problemer, udbydes den ikke igen. For det andet kan robotter, der allerede er i gang med en opgave, ignorere andre udbudte opgaver. For det tredje kan udbyderen udsende en minimal egnethed, som en vinder skal have. Hvis en robot ikke opfylder denne, sender den ikke et bud. Endelig antager auktioner ikke at alle robotter kan kommunikere med alle. Hvis en robot ikke modtager en besked om en auktion, vil den blot undlade at byde. Da kommunikationen er lokal, vil robotter, der befinder sig langt fra den opgave, der skal løses blot undlade at byde. Hvilket er fornuftigt da disse robotter skal bevæge sig langt for at løse opgave, og dermed er deres motivation mindre. I *Cross-Inhibition* spiller alle robotterne den samme rolle i forbindelse med kommunikationen. Dette er ikke tilfældet i forbindelse med auktioner. I auktioner er det udbyderen af en opgave, der tildeler den til en byder. Dermed har udbyderen en specielt ansvar i forbindelse med uddelingen af opgaven, og udbyderen skal sørge for at opgave bliver løst.

Disse egenskaber gør, at auktioner skalerer bedre end *cross-inhibition* med hensyn til kommunikationens båndbredden, og dermed er auktioner et bedre bud, hvis gruppen af robotter er stor eller båndbredden er lille.

4.4.6 Mobile Sensor Networks

Sensor networks er netværk af sensorer, som har til opgave at indsamle informationer om omgivelserne. Et eksempel på et *Sensor network* er et netværk af fastplacerede enheder, der har til opgave at overvåge trafik, og sende oplysninger om for eksempel kødannelse tilbage til en opsamlingsstation. Disse enkelte enheders rolle er at indsamle data fra deres eget område og kommunikere disse tilbage til en opsamlingsstation. Et alternativ til stationære *sensor networks* er *mobile sensor networks*. [HMS01] definerer *mobile sensor networks* på følgende måde:

> "*A mobile sensor network is composed of a distributed collection of nodes, each of which has sensing, computation, communication and locomotion capabilities*" [HMS01]

Enhederne i et *mobile sensor network* skal have mulighed for at kommunikere trådløst. Dette kan opnås ved at alle enhederne er i direkte forbindelse med opsamlingsstationen ved hjælp af et *wireless local area network* teknologi [Win00]. Hvis det ikke er muligt for alle enhederne at være i direkte forbindelse med opsamlingsstationen er det nødvendigt at videresende data gennem en eller flere andre enheder. Da enhederne er mobile er det oplagt

at benytte teknikker som ad hoc netværk til at sende data fra de enkelte enheder til opsamlingsstationen. Kommunikationen mellem de enkelte enheder og opsamlingsstationen bliver yderligere kompliceret, hvis det ikke kan antages at der hverken direkte eller gennem andre enheder er en forbindelse til opsamlingsstationen. I dette tilfælde kaldes ad hoc netværket fragmenteret.

[Win00] har lavet nogle simulationer, hvor der afprøves teknikker til at kommunikere data tilbage til en opsamlingsstation i situationer, hvor netværket er fragmenteret. [Win00] antager at to enheder kan kommunikere, hvis afstanden mellem dem er mindre end kommunikationens rækkevidde. Denne antagelse baserer sig på at enhederne kommunikere ved hjælp af radiokommunikation, som udbreder sig *omnidirectionelt*, det vil sige lige langt i alle retninger. Som beskrevet i afsnit 3, så baserer routingprotokollen i [Win00] sig på *flooding*, således at når enheder opdager nye naboer, ved hjælp af en ikke nærmere specificeret *neighbour discovery* algoritme, så videresender de alle deres opsamlede data til denne. I det ideelle tilfælde har *flooding* den egenskab, at da data udbredes mest muligt i netværket, så vil data komme frem til destinationen, hvis dette overhovedet er muligt.

[GVSM02b] har implementeret et ad hoc netværk på LEGOs RCX. Da LEGOs RCX kommunikation har begrænset rækkevidde, kan det antages at det netværk, som er implementeret vil være stærkt fragmenteret. Da kommunikationen er retningsbestemt, er det desuden mere kompliceret at implementere en *neighbour discovery* algoritme.

Den routingprotokol, der er implementeret i [GVSM02b] minder om den, der er implementeret i [Win00]. Routingprotokollen i [GVSM02b] er baseret på at hver enhed gemmer de data, den modtager i en buffer, så de senere kan videresendes. Men hvor alle data i [Win00] videresendes til alle nye naboer, så er [GVSM02b] begrænset af at enhederne ikke kan antages at have længerevarende forbindelser. Derfor har [GVSM02b] ændret *flooding* protokollen, så enhederne i stedet for at sende alle data, prioriterer data i en afsendelsesrækkefølge. På denne måde sikres det, at hvis forbindelsen bliver brudt, så vil de vigtigste data være blevet afsendt.

Det er oplagt at *mobile sensor networks* har mange af de samme karakteristika som multirobotsystemer. De teknikker til at kommunikere, som er anvendt inden for *mobile sensor networks*, kan overføres til robotter, i de tilfælde, hvor det ikke kan antages at alle robotter kan kommunikere med alle.

4.5 Delkonklusion

Der er flere måder for robotter at fordele opgaver i mellem sig. De ovenstående eksempler giver flere forskellige måder at benytte kommunikation i

forbindelse med koordination mellem robotter. Flere afhænger af antagelser, som ikke kan overholdes i alle situationer. Eksempelvis antager *Alliance* at robotter er i stand til at overtage opgaver fra andre robotter, hvis der ikke sker tilfredsstillende fremskridt. Hvis to robotter forsøger at løse samme opgave, er det ikke sikkert at disse automatisk vil hjælpe hinanden. Man kan forestille sig mange situationer, hvor robotter direkte modarbejder hinanden, hvis de ikke er enige om, hvem der skal løse en given opgave. Hvis en robot i *Alliance* ikke er tilfreds med en anden robots fremskridt, så vil den overtage denne opgave. Hvis den robot, der oprindeligt havde opgaven, er tilfreds med sit fremskridt vil den fortsætte med at løse opgaven, og de to robotter vil muligvis ødelægge hinandens muligheder for at løse opgaven tilfredsstillende. Dette kan igen tiltrække flere robotter, der ikke er tilfreds med opgavens løsning.

Den antagelse om, at robotter kan overtage opgaver, hvis den robot, der er i gang med at udføre den fejler, ligger til grund for flere af ovenstående eksempler. Specielt dem, hvor robotterne påtager sig symmetriske roller, er det en nødvendig antagelse.

Flere af de ovenstående eksempler antager også at alle robotter er i stand til at kommunikere med alle. Dette giver potentielle problemer i forhold til kommunikationens båndbredde. Men ud over dette er antagelse slet ikke realistisk i forbindelse med robotgrupper, der ikke er samlet i et lille område. Der vil muligvis være fordele ved at anvende kommunikation med begrænset rækkevidde til at fordele opgaver i mellem sig. Dette kan gøres fordi robotter, der ikke er i nærheden, næppe er egnede til at løse en given opgave. En algoritme, der ikke antager at alle robotter deltager i forhandlingen af alle opgaver vil skalere bedre med antallet af robotter og opgaver.

Et eksempel på grupper, der skalere godt med antallet af deltagere, er de flokke af fugle eller stimer af fisk, som optræder i naturen. [Rey87] beskriver disse flokke ud fra et datalogisk perspektiv.

> "*There is no evidence that the complexity of natural flocks is bounded in any way. Flocks do not become "full" or "overloaded" as new birds join. [...] These speculations about the "computational complexity" of flocking are meant to suggest that birds can flock with any number of flockmates because they are using what would be called in formal computer science a constant time algorithm*"
> [Rey87]

[Rey87] benytter altså fænomener i naturen som et eksistensbevis for, at der eksistere algoritmer til at løse problemet *flocking*, hvor kompleksiteten af den enkelte deltagers indsats ikke er afhængig af antallet af deltagere i hele flokken.

Der er ingen tvivl om at en tilsvarende algoritme vil være ønskelig, hvis en gruppe af robotter skulle løse problemet *flocking*. Om der eksisterer algoritmer til at løse andre problemer, der skalerer lige så godt, er ikke oplagt. Men en sådan algoritme vil under alle omstændigheder benytte at de enkelte robotter kun skal koordinere med et fast antal af robotter i den umiddelbare nærhed. Derfor vil kommunikation med begrænset rækkevidde være tilstrækkelig, måske endda en fordel i dette scenarie.

Der eksisterer ikke nogen generel løsning på, hvordan en gruppe af robotter bedst kommunikerer eller koordinerer deres opgaver. Forskellige opgaver kræver forskellige løsninger, hvor den enkelte opgaves og robotgruppens karakteristika spiller ind i afvejningen af forskellige designvalg.

5 Implementation af Ad Hoc netværk på RCX

Implementationen af en kommunikationsprotokol til RCX kan tage inspiration i OSI-stakken [CDK01]. I så fald skal der implementeres et datalinklag og et netværkslag oven på det fysiske lag, som RCX'en stiller til rådighed i form af det indbyggede Serial Communication Interface, SCI, appendix B.

Et alternativ til dette er LegOS[1] Network Protocol, LNP [GVSM02a]. LNP har to lag i protokolstakken ud over det fysiske, som er bestemt af hardware. De to lag er *integrity* og *logical*. *Integrity* laget sørger for pakkernes integritet - det vil sige, at det undersøges, om pakkernes indhold er blevet ændret under transmissionen. *Logical*-laget tilføjer adressering oven på *integrity*-laget. Denne opdeling identificerer to opgaver og fordeler dem mellem to lag. Opdeling i *integrity*- og *logical*-lag svarer til den opdeling af datalinklaget i *medium access control* og *logical link control*, som IEEE 802 anvender [Sta00]. LNP implementerer udelukkende den funktionalitet, der normalt hører til på datalinkniveau. Det vil sige kommunikation mellem to direkte forbunde enheder.

Den kommunikationsprotokol, som er beskrevet i dette afsnit, følger til dels opdelingen fra LNP. Der er forskellige afvigelser fra dette, men grundlæggende opdeler protokollen sig i fire separate lag; det fysiske lag, et integritetslag, et logical link-lag samt et netværkslag. Ansvarsfordelingen mellem de fire lag er som følger:

Fysisk lag: Dette lag sørger for at sende de enkelte bytes serielt som infrarødt lys samt at konvertere modtaget lys til bytes. Dette lag er implementeret af det SCI, som er indbygget i RCX'en og er beskrevet i appendix B.

Integritetslag: Dette lag implementerer pakkeformat samt undersøger pakker for fejl opstået under transmission.

Logical link-lag: Dette lag implementerer adressering, gensendelse af tabte pakker samt oprettelse og nedlæggelse af forbindelser

Netværks lag: Dette lag implementerer en routingprotokol samt en strategi til buffering af pakker.

Det fysiske lag er implementeret i RCX'ens hardware og er beskrevet i appendix B. Det vil derfor ikke blive beskrevet nærmere i dette afsnit. De øvrige lag er implementeret i software og vil nedenfor blive beskrevet mere præcist.

[1] LegOS er ikke LEGOs, men derimod en forkortelse af Lego Operating System, som er et open source projekt uafhængigt af Lego koncernen.

5.1 Antagelser om den infrarøde kommunikation

For at designe en kommunikationsprotokol er det nødvendigt at overveje, hvilke situationer der kan opstå, og hvilke situationer der ikke kan opstå i forbindelse med kommunikationen.

Den protokol, der ønskes implementeret, er en flooding/gossip baseret protokol [GVSM02a]. Der bliver altså tale om forbindelser mellem enheder, der kan kommunikere direkte med hinanden.

De infrarøde signaler udbreder sig som en fremadrettet kegle, appendix C. Denne karakteristisk udbredelse gør, at der nemt opstår unidirectionelle links - det vil sige links, hvor kommunikation kun er mulig i den ene retning. Unidirectionelle links vil opstå, hvis enhederne ikke er orienteret således, at begge parter befinder sig inden for modpartens lyskegle.

Infrarøde signaler bliver i høj grad påvirket af lys fra omgivelserne. Dette skyldes, at naturligt lys indeholder store mængder infrarødt lys. Lyset i omgivelserne har derfor stor betydning for den infrarøde kommunikations rækkevidde og stabilitet. Desuden er det infrarøde medium delt mellem alle enhederne. Derfor kan enheder ikke afsende data samtidigt uden at ødelægge data. Disse ting gør, at der kan påregnes en relativt stor sandsynlighed for, at pakker går tabt, eller at deres indhold bliver ændret.

Implementationen af en kommunikationsprotokol på robotter giver yderligere komplikationer. Således kan forbindelser blive brudt uforudset. Dette kan ske, fordi robotterne kan fejle og derfor ophører med at deltage i kommunikationen. Det kan også ske, fordi robotterne bevæger sig væk fra hinanden, eller fordi en tredjepart bevæger sig ind mellem de to robotter. Alle kommunikationsprotokoller skal kunne komme sig oven på den slags problemer, men for kommunikation mellem mobile enheder opstår disse situationer langt oftere, da langt flere dynamiske fænomener spiller ind. Der antages følgende om den infrarøde kommunikation:

- Pakker kan tabes helt.
- Enkelte bytes i pakkerne kan tabes.
- Enkelte bits kan vendes.
- Bytes er ordnet korrekt indbyrdes.
- Der kan opstå unidirectionelle links.
- Forbindelser kan blive brudt uforudset.

Disse forskellige antagelser giver en række komplikationer, når kommunikationsprotokollen skal designes, men det giver også en række forenklinger. At

pakker kan blive tabt, skal håndteres på *logical link*-laget, hvor *flow control* skal afhjælpe dette problem for eksempel ved hjælp af acknowledgements [Sta00] . At enkelte bytes kan tabes og enkelte bit kan vendes, skal klares på integritetslaget, hvor *error detection* [Sta00] skal kunne detektere ødelagte pakker.

At bytes altid ankommer i den afsendte orden er en korrekt antagelse da kommunikationen foregår mellem to - gennem infrarød kommunikation - direkte forbundne enheder, og bytes således ikke kan overhale hinanden. Pakker kan overhale hinanden længere oppe i protokolstakken. For eksempel er det i en gossip routing protokol op til hver enkelt enhed, hvilke pakker den vil videresende, således kan pakkerne her byttes rundt alt efter videresendelses protokol.

Unidirectionelle links er en naturlig konsekvens af den retningsbestemte infrarøde kommunikation. Dette vil der blive taget højde for ved at oprette forbindelser ved hjælp af de teknikker, der bliver beskrevet i afsnit 6.

Forbindelser kan blive brudt uforudset som en konsekvens af enhedernes autonomi og mobilitet. Dette skal der tages højde for, og enhederne skal være i stand til at komme videre efter et sådant nedbrud.

5.2 Integritetslag

Integritetslagets opgaver er: At ordne strømmen af bytes, som modtages fra SCI, til pakker, at afsende pakkernes bytes i den rigtige rækkefølge samt at undgå kollisioner og opdage fejl i de data, der modtages.

5.2.1 Strategi til at undgå kollisioner

For traditionel kommunikation er der udviklet forskellige teknikker til at undgå kollisioner. En udbredt teknik er *Carrier Sense Multiple Access with Collision Detection* forkortet CSMA/CD [Sta00]. *Carrier sense* dækker over at en enhed - der ønsker at sende på kommunikationsmediet - først undersøger, om der er en transmission i gang. Hvis dette er tilfældet, venter enheden indtil kommunikationsmediet igen er ledigt. *Collision detection* dækker over at en enhed, der sender på mediet, overvåger transmissionen. Hvis der opdages en kollision, så udsendes et signal, der angiver at der er opstået en kollision. *Collision detection* kan implementeres forskelligt afhængigt af kommunikationsmediet. Således kan kollisioner på coaxialkabler detekteres ved at signalstyrken på kablet øges, mens det i ved utpkabler kan basere sig på logik i de hubs, der videresender signalet [Sta00].

Ved trådløs kommunikation mellem RCX'er er det ikke muligt at undgå kollisioner ved teknikker som CSMA/CD i traditionel forstand. For det før-

ste understøtter chippen til seriel kommunikation, SCI, ikke direkte *carrier sense*. En afsender har dog viden om igangværende transmissioner så snart første byte af headeren er modtaget, hvilket kan benyttes til at implementere en *carrier sense*, således at en enhed ikke udsender data, hvis den har modtaget en header. Et andet problem med at implementere CSMA/CD på en RCX er, at *collision detection* er besværliggjort af kommunikationens trådløse natur, hvor problemer som *hidden terminal* opstår, afsnit 3. Alternativt til CSMA/CD er der forskellige måder at undgå kollisioner, mange af dem er baseret på reservation af mediet, som f.eks. MACA [Kar90].

Der er i denne implementation kun en særdeles simpel måde at undgå kollisioner. Denne består af den simple teknik, at hver enhed sender en header til hver pakke, en enhed der opdager denne header sender ikke på mediet før hele pakken er modtaget. Headeren fungerer altså som en reservation af mediet. Problemer som; *hidden terminal*, kollisioner af headeren og kollisioner, der skyldes, at enheder som følge af mobilitet ikke modtager headeren, kan stadig opstå.

5.2.2 Strategi til at opdage kollisioner

Hvis det ikke lykkes at undgå kollisioner, er det vigtigt, at modtageren af data kan opdage, at en kollision har fundet sted, og at de data, den har modtaget, derfor er ødelagt. Dette kan gøres ved at implementere en *error detection* ved hjælp af en *errorcode* således, at en enhed, der modtager data kan kontrollere dem for konsistens [Sta00]. Er der opstået en fejl, vil denne teknik muligvis afsløre den.

I hardwaren på RCX'en er der allerede en sådan *collision detection* i form af en paritetsbit i hver byte, appendix B. Denne beskytter for fejl, hvor et ulige antal bits pr. byte bliver vendt. Det er muligt at lægge yderligere kontrol af data i integritetslaget.

Dette kan gøres ved *Cyclic Redundancy Check*, CRC [Sta00], hvilket giver gode muligheder for at detektere langt de fleste fejl. For at beskrive, hvordan CRC virker, defineres følgende:

M: Beskeden, der ønskes sendt. Længden defineres som k.

F: Errorcode, der har længden n $<$ k.

P: Et mønster på n+1 bits, som benyttes som divisor

T: Den pakke, der sendes, har længden n+k. Den er givet ved $2^n M + F$

Fejlkontrollen består i, at vælge F så $\frac{T}{P}$ ikke har nogen rest, når der regnes modulo 2. Når afsenderen skal finde F benyttes følgende: $\frac{2^n M}{P} = Q + \frac{R}{P}$,

og F angives til at være R, det vil sige resten ved divisionen. Således er $T = 2^n M + R$, dermed har $\frac{T}{P}$ ikke nogen rest da: $\frac{T}{P} = \frac{2^n M+R}{P} = Q + \frac{R}{P} + \frac{R}{P} = Q$. Regnet modulo 2 er X + X = 0, for alle sekvenser af bits, X. Dette kan indses ved at 0 + 0 = 0 samt at 1 + 1 = 0. Modulo 2 operatoren + er blot bitvis exclusive or. Dermed er der ikke nogen rest, og T er delelig med P.

Der skal altså vælges et fast divisormønster, P, som er det samme for alle enheder, der ønsker at kommunikere. Derefter skal afsenderen af en besked udregne en passende Errorcode, F, således, at den afsendte pakke er præcist delelig med P. Gøres dette kan modtageren verificere en pakkes indhold ved at dividere den modtagne pakke med divisormønstret. Hvis der opstår en rest ved divisionen, er pakken ødelagt. Opstår der ikke nogen rest antages det, at pakkens indhold er intakt.

Dette detekterer alle fejl, E, der ikke er delelig med P. Det gælder således om at vælge P således, at chancen for, at P går op i E, er minimal.

Der sendes en byte, som er afsat til implementation af *collision detection*, sidst i hver pakke, dette muliggør et divisormønster på 9 bits. Indholdet i denne byte er ikke implementeret. Det vil sige, at modtageren ikke udregner nogen *errorcode*, at modtager ikke verificerer nogen *errorcode* og at der ikke er valgt noget divisormønster i den nuværende implementation.

Ligesom modtageren af en pakke kan kontrollere dens indhold ved hjælp af en *error code* så kan afsenderen også undersøge pakken. RCX'ens kommunikation er designet således at den gennem SCI også modtager de pakker der afsendes. Hvis der kan været en kollision så vil afsenderen af en pakke kunne se det ved at sammenligne de data, den har afsendt med, de data den har modtaget.

5.2.3 Implementation af integritetslaget

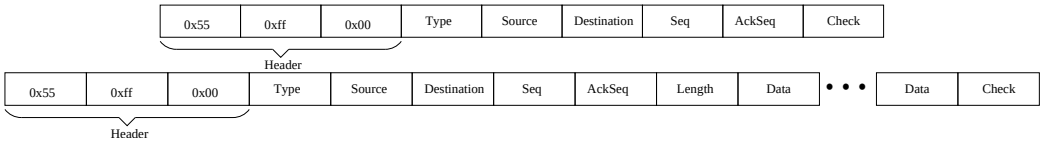
Dette lag er det nederste lag, der er implementeret i software. Det er implementeret i filen IR.h, appendix E. Integritetslaget opererer direkte oven på RCX'ens SCI, appendix B.

Kommunikationen med det underliggende lag foregår med interrupthandlere. De to vigtigste er TXI() og RXI(). Disse interrupts bliver aktiveret af SCI, når en byte er henholdsvis afsendt og modtaget. Interrupthandlernes opgave er herefter henholdsvis at skrive næste byte til SCI og læse næste byte fra SCI. Når en hel pakke er modtaget, skrives denne til en modtagebuffer.

Det pakkeformat, der er implementeret på dette niveau, er fælles for både integritetslaget og *logical link*-laget. Disse to opererer altså med samme pakkeformat. Pakkernes indhold er repræsenteret i en struct, figur 5.1. Det er de to interrupthandleres opgave at afsende pakker repræsenteret som ovenstående struct således, at de overholder det givne pakkeformat. Pakkeformattet

```
struct dl{
  //Required fields
  byte type;
  byte src;
  byte dst;
  byte seq;
  byte ackseq;
  byte check;
  //Optional fields only used in datapackets
  byte length;
  byte data[32];
};
```

Figur 5.1: Struct til repræsentation af pakker



Figur 5.2: Format for pakker på integritets- og logical link- niveau

er angivet i figur 5.2.

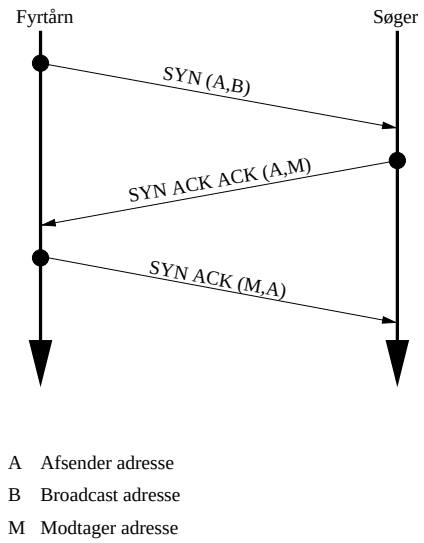
Pakker afsendes først med en header, der består af tre bytes med værdierne $0x55$, $0xff$ og $0x00$. Efter headeren kommer pakketypen. Pakketypen anvendes på integritetslaget til at skelne datapakker fra kontrolpakker og på *logical link*-laget til at skelne forskellige typer af kontrolpakker fra hinanden. De pakketyper, der anvendes, er: Heartbeat (1), Heartbeatreply (2), Heartbeatreplyreply (3), Data(4), Fin (5).

Efter pakketype følger to bytes med source og destination. Disse to angiver afsender og modtager af pakken. Hvis en robot ikke er modtager af en pakke, ignoreres denne på laveste mulige niveau. Det vil i dette tilfælde sige, at hvis interrupthandleren modtager destination byten og opdager, at den ikke er den tiltænkte modtager, vil den ignorere resten af pakken. Dette ansvar hører i LNP hjemme i *logical link*-laget [GVSM02a]. Grunden til, at det i denne protokol er implementeret på et lavere lag, er, at der derved kan undgås et unødigt overhead ved at behandle pakker, som ikke er adresseret til pågældende robot. Da RCX'ens transceiver er indrettet således, at den også modtager de pakker, den selv afsender, spares behandlingen af disse pakker på overliggende lag ved at implementere adresseringen på lavest mulige niveau.

Da der er tale om trådløs kommunikation, er alle links som udgangspunkt broadcast baserede. Den adressering, som er implementeret oven på dette, skal organiseres således, at både broadcast- og adresseret kommunikation understøttes direkte og således, at broadcastkommunikation direkte anvender den trådløse kommunikations broadcastbaserede natur. Dette er gjort ved at have en broadcastadresse, som er valgt til $0x00$. Denne broadcastadresse kan benyttes til at sende beskeder til alle enheder i nærheden. Beskeder sendt til broadcastadressen behandles af alle enheder, som om den var adresseret til dem.

Efter source og destination følger to bytes med henholdvis sekvensnummer, Seq, og sekvensnummeret på den sidst modtagne pakke, AckSeq. Disse bytes anvendes af *logical link*-laget til at implementere *flow control*. Efter sekvensnumrene følger i kontrolpakkerne et *check*-felt. Dette felt indeholder muligheden for at implementere for eksempel *Cyclic Redundancy Check* [Sta00] således, at fejl i de modtagne data kan opdages. Der er ikke i den nuværende implementation implementeret nogen *error detection*, men muligheden for at implementere dette er til stede i *check*-feltet. I datapakker kommer der først en byte, der angiver det antal databytes, som følger, og samtlige databytes inden *Check*-feltet.

Det overliggende *logical link*-lag kan kalde de to funktioner IRTransmit og IRReceive, som begge blokerer, indtil hele pakken er afsendt, eller en hel pakke er modtaget. Begge funktioner tager datapakker af den type, der er



Figur 5.3: Three way handshake

angivet i figur 5.1, som argument.

5.3 Logical link-lag

Logical link-laget skal sørge for at alle pakker når frem. Dette gøres ved at oprette forbindelser og sende acknowledgements på datapakker.

5.3.1 Oprettelse af forbindelse

Oprettelse af forbindelser foregår ved at de trådløse enheder udfører en eller anden *neighbour discovery*, som beskrevet i afsnit 3. Formålet med *neighbour discovery* er at gøre enhederne i stand til at opdage, når det bliver muligt at kommunikere med nye enheder i omgivelserne. Der er mange forskellige måder at implementere *neighbour discovery*.

En teknik til at foretage *neighbour discovery* er den *inquiry* procedure der anvendes i bluetooth [Met99]. Denne inquiry proces er designet til den situation, hvor den trådløse kommunikations rolle er at erstatte traditionel kommunikation, derfor har forskellige enheder asymmetriske roller i forbindelse med *inquiry* proceduren. [RK03] præsenterer en alternativ en algoritme for *neighbour discovery*, hvor i alle enhederne har symmetriske roller. Symmetriske roller er en fordel i forbindelse med ad hoc netværk, hvor alle enhederne indgår på lige vilkår i et dynamisk netværk. Algoritmen i [RK03] bygger på periodisk udsendelse af *beacon*-pakker. Enheder, der ønsker at udføre *neigh-*

bour discovery kan så lytte efter disse pakker, og på denne måde opnå viden om hvilke enheder, der befinder sig i nærheden. Denne algoritme minder meget om den *find-fellowship*, der er implementeret i [GVSM02b]. Problemet med denne tilgang er, at selvom en enhed modtager en *beacon*-pakke, så kan den ikke nødvendigvis sende pakker retur, dette skyldes at der i forbindelse med trådløs kommunikation kan opstå *unidirectionelle* forbindelser.

Alternativt *beacon*-pakker kan der for at sikre tovejskommunikation anvendes et *three way handshake* [Sta00], figur 5.3. Ved et *three way handshake* skal der i alt sendes tre beskeder. Først vil den ene part afsende en *SYN* besked, i modsætning til [Sta00] er beskederne. I denne implementation er *SYN* ikke rettet mod en bestemt modtager, og de kaldes derfor *Heartbeat*. Modtageren af en *SYN* besked vil besvare denne med en *SYN ACK* besked. Når afsenderen af den oprindelige *SYN* besked modtager en *SYN ACK* besked, vil den besvare denne med en *SYN ACK ACK* besked, og når denne modtages, vil begge parter have vished for, at tovejskommunikation er mulig. Desuden bliver *SYN* og *SYN ACK* benyttet til at synkronisere sekvensnumre for de efterfølgende pakker.

Three way handshake er valgt til oprettelse af forbindelser i denne implementation i modsætning til teknikkerne i for eksempel [GVSM02b]. Grunden til dette er, at det under et *three way handshake* aktivt kan forsøges at opnå forbindelser, ved at kontrollere robottens bevægelse. Desuden er *Heartbeat*-pakker mindre end datapakker, og dermed er der mindre overhead ved periodisk at udsende *Heartbeatpakker* fremfor datapakker, som [GVSM02b] gør. I afsnit 6 undersøges det om dette kan anvendes til at opnå flere og mere stabile forbindelser end alternativet med spontant opståede forbindelser som i [GVSM02b].

5.3.2 Flow control

Når først forbindelsen mellem to RCX'er er etableret, skal der anvendes en passende *flow control* [Sta00]. Den algoritme, der er valgt, er *stop and wait* [Sta00]. *Stop and wait* er særdeles simpel og består blot i at en afsender af pakker ikke afsender en pakke, før den forgående er blevet bekræftet. I tilfældet med robotter, der opretter forbindelser for at udveksle data, vil det ofte være tilfældet at begge parter har data, som de ønsker at afsende. I dette tilfælde kan *acknowledgements* sendes med som et felt i headeren på datapakkerne. Dette understøttes direkte af headeren i datapakkerne, figur 5.2, hvor feltet *AckSeq* benyttes til at sende sekvensnummeret på sidst modtagne pakke. Hvis den ene part i kommunikationen ikke har mere data at afsende, så sendes i stedet Fin beskeder, som så i dette tilfælde fungerer som acknowledgements. En god egenskab ved *stop and wait* er at kommunikationsmediet

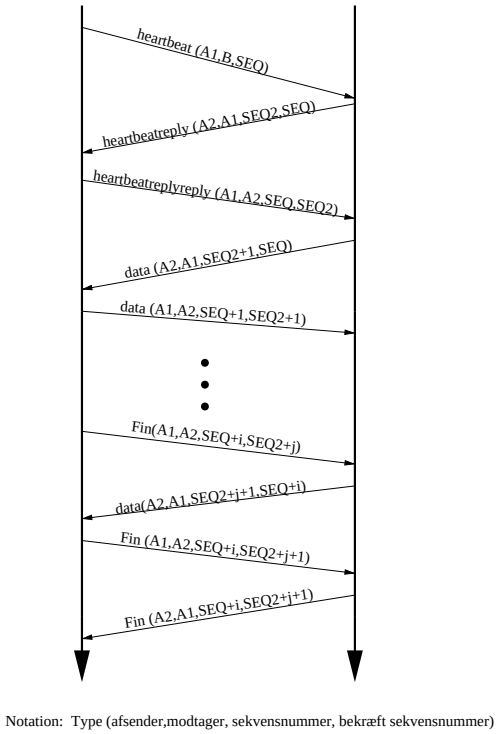
bliver fordelt mellem de to parter, da de skiftes til at afsende pakker. En afsender vil således afsende en pakke, og derefter vente indtil den har modtaget en pakke. Et eksempel på, hvordan denne kommunikation virker, kan ses på figur 5.4. Et alternativ til *stop and wait* kunne være *sliding window* [Sta00], hvor en afsender har et *window* af beskeder, som kan afsendes, inden den skal vente på et *acknowledgement*. *Sliding window* er dog ikke anvendelig i dette tilfælde. For det første vil begge enheder kunne afsende en pakke samtidigt, og dermed skabe kollisioner. For det andet udnytter *stop and wait* mediets kapacitet godt, da den tid det tager at sende en pakke er meget stor i forhold til den tid transmissionen af en enkelt bit tager, dermed er argumentet for *sliding window* algoritmens bedre udnyttelse af båndbredden væk [Sta00].

Der kan som følge af kollisioner opstå situationer, hvor pakker ikke kommer frem. Derfor er det nødvendigt med en protokol, der håndterer pakketab. I tilfælde af et pakketab er det nødvendigt at gensende pakken. Det er umuligt for en afsender at vide om det er den afsendte datapakke, der er tabt eller om datapakken er nået frem, og det i stedet er acknowledgement pakken, der er gået tabt. Der kan derfor opstå situationen, hvor begge parter i kommunikationen forsøger at gensende pakker. Dette kan føre til kollisioner, og derfor skal der implementeres en passende gensendelsesstrategi, som minimerer risikoen for kollisioner.

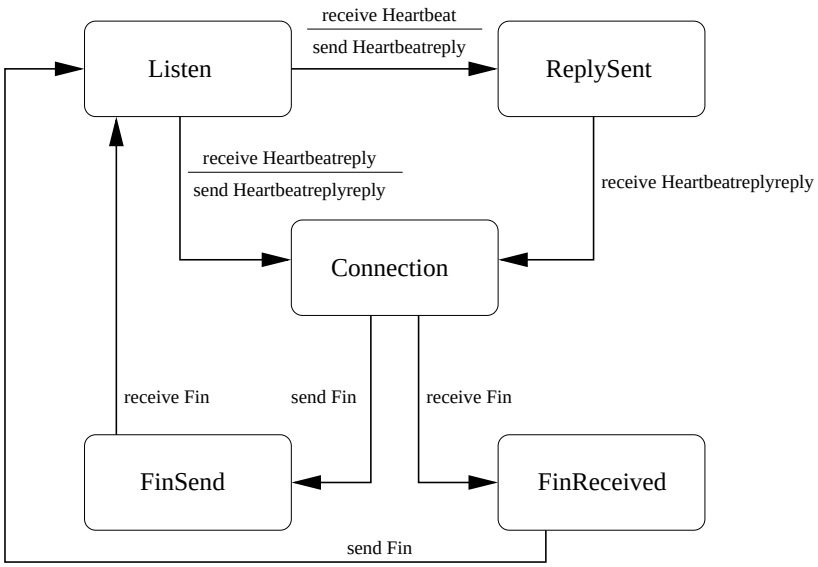
Den gensendelsesstrategi, der er valgt, er *exponential backoff* [Sta00]. Algoritmen bag *exponential backoff* består i at en enhed, der ikke modtager et *acknowledgement* på en afsendt datapakke, vil gensende pakken efter et passende *Retransmission Time Out*, RTO. Hvis der heller ikke modtages noget *acknowledgement* på den gensendte pakker forsøges der igen, men denne bliver tiden udvidet, således at RTO ændres som $RTO = q \times RTO$ for hver gensendelse. Dette løser ikke problemet, hvis to enheder forsøger gensendelse præcis samtidigt, derfor vælges et timeout pseudotilfældigt ved hver gensendelse. Dette timeout ligger mellem et passende minimum og RTO.

Ved at indføre tilfældighed og eksponentielt voksende timeouts, er det håbet at kollisioner sjældent forekommer gentagne gange efter hinanden, og at alle pakker derfor kommer igennem inden for få gensendelser.

På et tidspunkt er der ikke flere data, der skal afsendes, så derefter skal forbindelsen mellem de to enheder nedlægges. Dette gøres ved at en enhed, der ikke har flere data, i stedet for næste datapakke sender en FIN pakke. FIN pakken fungerer som acknowledgement for den sidst modtagne datapakke og den angiver desuden at enheden ikke har yderligere data, der skal kommunikeres. Den anden part i kommunikationen kan dog stadig have data, som den ønsker at afsende. derfor nedlægges forbindelsen ikke med det samme. Efter at den første enhed ikke har flere data og derfor afsender en FIN besked, så vil den fortsætte med at modtage data fra modparten og sende FIN beskeder



Figur 5.4: Stop and wait protokol



Figur 5.5: Tilstande i forbindelse med oprettelse og nedlæggelse af forbindelser

som acknowledgements på disse. Det er først når begge enheder både har afsendt og modtaget en FIN besked at forbindelsen endeligt nedlægges.

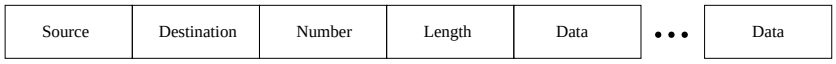
5.3.3 Implementation af logical link-laget

logical link-laget er implementeret i filen Communication.h, appendix E. Oprettelsen og nedlæggelsen af forbindelser er implementeret ved hjælp af en række tilstande. Disse tilstande er *Listen*, *Reply Sent*, *Connection*, *Fin Send* og *Fin Received*. Disse tilstande og skift mellem dem kan ses på figur 5.5.

I tilstanden *Listen* udsendes periodiske *Heartbeats*. Alle enheder forsøger på denne måde aktivt at oprette forbindelser. Hvis en enhed modtager et *Heartbeat* vil dette blive besvaret, og der skiftes til tilstanden *Reply Sent*. Hvis der i tilstanden *Listen* modtages et *Heartbeatreply* vil dette blive besvaret med et *Heartbeatreplyreply*, hvorefter der skiftes til tilstanden *Connection*.

I tilstanden *Reply Sent* afventes der, at modparten svarer med et *Heartbeatreplyreply*, hvis dette ikke sker, vil et timeout sørge for at der skiftes tilbage til tilstanden *Listen*. Hvis der derimod modtages et *Heartbeatreplyreply*, så skiftes til tilstanden *Connection*.

Da den infrarøde kommunikation mellem to RCX'er er retningsbestemt er det ikke altid muligt at besvare *Heartbeats*. Dette problem bliver undersøgt nærmere i afsnit 6.



Figur 5.6: Pakkeformat på netværksniveau

I tilstanden *Connection* kan der afsendes og modtages datapakker. Hver enhed har en buffer med data, som den sender en pakke af gangen. Når en pakke er afsendt vil afsenderen vente på at modtage et acknowledgement. Hvis dette acknowledgement ikke kommer vil den efter et tidsrum forsøge at afsende pakken igen. Efter en gensendelse ventes der igen, der vælges et timeout i et interval, hvor den øvre grænse bliver større ved hver gensendelse.

Fejltolerance er implementeret ved at tilføje timeouts i de forskellige tilstande. Hvis der inden for dette timeout ikke er modtaget data på forbindelsen, bliver den nedlagt, hvorefter der skiftes til tilstanden *Listen*. Disse timeout er implementeret ved at hver pakke højst gensendes 5 gange, hvis der ikke modtages noget acknowledgement. Efter femte forsøg opgives forbindelsen. Forbindelser kan udløbe på denne måde i tilstandene *Connection*, *Fin Send* og *Fin Received*.

Når en enhed ikke har flere pakker i sin buffer, vil den sende en Fin besked og når begge enheder har afsendt og modtaget en Fin besked bliver forbindelsen nedlagt, og enhederne skifter tilbage til tilstande *Listen*.

5.4 Netværkslaget

Netværkslaget skal implementere en flooding/gossip routing protokol. Det er på dette niveau at pakkerne gemmes i en buffer, og der skal implementeres en strategi til at håndtere denne buffer. Denne strategi bestemmer hvilke pakker, der sendes først og hvilke pakker, der smides væk, når der ikke er tilstrækkelig plads i bufferen.

På netværksniveau kan det være nødvendigt at pakker bliver videresendt gennem flere midlertidige knuder, inden den kommer frem til den endelige modtager. I en flooding/gossip protokol vil enhederne forsøge at sende pakkerne til alle sine naboer. Når enhederne er mobile, får de hele tiden nye naboer, og dermed kan to enheder være naboer et stykke tid, for derefter at bevæge sig væk fra hinanden, og efter et stykke tid igen blive naboer. Derfor kan en enhed modtage den samme pakke flere gange. Det er derfor nødvendigt at have en mulighed for, når en pakke modtages, at finde ud af om den allerede er i bufferen. Dette er implementeret ved at den oprindelige afsender af en pakke nummerere de pakker den afsender. På denne måde kan pakker genkendes som dubletter, hvis den oprindelige afsenderadresse og

deres nummer er det samme.

Det samlede pakkeformat på netværksniveau er vist i figur 5.6. Udover source og destination er der en byte med number, der angiver pakkens nummer. Desuden er der som på datalink niveau en byte til length, der angiver antallet af databytes i pakken. Det maksimale antal databytes i en pakke er bestemt af pakkestørrelsen på datalink niveau. På datalinkniveau kan der højst være 32 databytes, af disse benyttes de fire til source, destination, number og length på netværksniveau. Dermed kan der højst være 28 databytes på netværksniveau.

Det er kun de dele af netværkslaget, som har været nødvendige for at udføre forsøgene i afsnit 6, der er implementeret. Der er implementeret en buffer med plads til op til 25 pakker. Pakkerne hentes ud af denne buffer i en first-in-first-sent rækkefølge akkurat ligesom i [GVSM02b]. Hver gang der oprettes en ny forbindelse, så startes der forfra med afsendelse af pakker, således er det den samme pakke, der afsendes først hver gang en enhed opretter forbindelse. Der er implementeret en simpel kontrol af om pakker er i bufferen i forvejen. Denne kontrol består i, at hver gang en pakke modtages gennemløbe bufferen og kontrollere om source og number matcher en eksisterende pakke. Er det tilfældet ignoreres pakken.

En pakke kan komme i denne netværksbuffer på to måder. Enten kan enheden modtage en pakke, som så gemmes i bufferen til senere videresendelse. Eller også kan enheden selv opsamle data, som den ønsker at afsende, disse gemmes også i bufferen til senere afsendelse. I det første tilfælde kommer pakken nedefra i protokolstakken, det vil sige fra datalink laget. I det andet tilfælde kommer pakken oppefra i protokolstakken, det vil sige fra applikationslaget.

Den strategi, der er implementeret, til at håndtere at bufferen allerede er fuld, når der modtages en pakke, er den simplest mulige. Hvis der ikke er plads til flere pakker i bufferen, så ignoreres nye pakker. Dette vil resultere i at enhederne, efter at have kommunikeret et stykke tid, ikke længere kan håndtere flere pakker, og kommunikationen vil derfor gå i stå. Dette har ikke været noget problem i forsøgene, da enhedernes kommunikation ikke har overskredet denne grænse. Hvis det i stedet ønskes at protokollen skal anvendes til længere kørsler med udveksling af mange data, er det nødvendigt med en anden strategi. Denne strategi kan benytte mange forskellige kriterier til at smide pakker væk. Den oplagte strategi er first-in-first-out, som vil gøre at de ældste pakker hele tiden forsvinder fra bufferen.

Bufferstørrelse og strategi til at håndtere overløb er vigtig for den overordnede ydelse af netværket. Pakker skal beholdes i bufferen, indtil de er blevet afleveret til den endelige modtager, men pakker skal heller ikke gemmes længere. Da hver pakke i en flooding/gossip protokol ofte er gemt i bufferen

på mange forskellige enheder, så er der ikke nogen mulighed for den enkelte robot at vide om en pakke er leveret til den endelige modtager. Dermed er der ikke nogen generel løsning på dette problem, og buffering af pakker skal designes omhyggeligt, så den passer til netop den ønskede kommunikation.

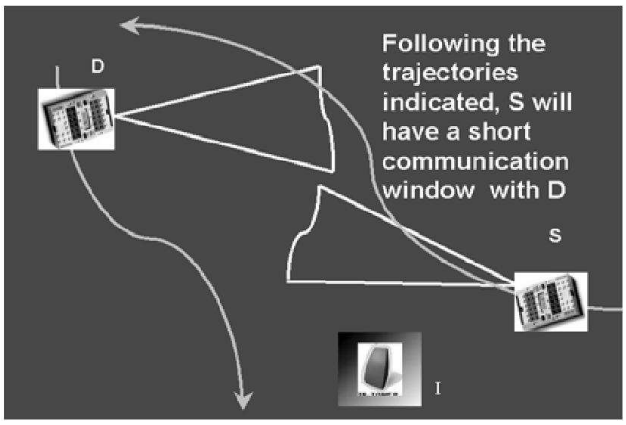
6 Anvendelse af Situated kommunikation

Situated kommunikation består af to dele: Abstract kommunikation og signalets fysik [Stø01]. Traditionelt er det den abstrakte kommunikation, der har været interessant, og abstraktioner på de laveste lag af de fleste protokolstakke - for eksempel OSI-stakken [CDK01] - abstraherer væk fra signalets fysik.

Infrarød kommunikation udbreder sig som en fremadrettet kegle, appendix C. Dermed ved en RCX, at hvis den modtager et signal fra en anden RCX, så er den inden for dennes fremadrettede lyskegle. Dermed indeholder infrarød kommunikation en *situated* egenskab - nemlig viden om afsenderens fysiske placering.

RCX'ens infrarøde signaler kan ses med RCX'ens lyssensorer. Således kan signalet på grund af sine fysiske egenskaber aflæses direkte på disse. Dette kan anvendes til at opnå yderligere information omkring signalet, f.eks. retningen.

[Stø01] demonstrerer, at det ikke kun er den abstrakte kommunikation, der kan finde anvendelse i multirobotsystemer. Således kan der opnås flere fordele ved ikke at abstrahere væk fra signalets fysiske virkelighed. [Stø01] anvender en gruppe af robotter, der bevæger sig i en korridor på 4 meter gange 35 meter. Målet for robotterne er at holde sammen i en gruppe. Robotterne er udstyret med tryksensorer, der benyttes til at implementere en simpel *obstacle avoidance* adfærd. Et andet *behavior producing module* i robotterne anvender RCX'ens infrarøde transceiver til at udsende en besked hvert halve sekund. Det samme *behavior producing module*, som udsender beskederne, registrerer hvor mange beskeder, robotten modtager pr. sekund, kaldet n. Antallet af beskeder, som en robot modtager, kan anvendes som et mål for, hvor mange robotter der er i nærheden. Denne egenskab benyttes til at udregne en værdi, $\hat{n_t} = (1-\alpha)\hat{n_{t-1}} + \alpha \dot{n_t}$, hvor $\dot{n_t} = n_t - n_{t-1}$. Et separat *behavior producing module* reagerer på ændringer i $\hat{n}$. Hvis $\hat{n}$ er positiv, betyder det, at robotten har modtaget flere beskeder, og derfor ændres adfærden ikke. Hvis $\hat{n}$ er negativ, modtager robotten færre beskeder, og derfor vendes robotten 180°. [Stø01] laver eksperimenter med to forskellige versioner af kontrolprogrammet. En version med det *behavior producing module*, der reagerer på ændringer i $\hat{n}$, og en version uden. [Stø01] sammenligner $\hat{n}$ for hver af de to kontrolprogrammer og kommer til den konklusion, at selv den simple måde at reagere på antallet af beskeder, som modtages, giver markant flere beskeder modtaget over tid. Således modtager robotterne i gennemsnit 2.90 beskeder pr. sekund, når de reagerer på ændringer i $\hat{n}$, mod 0.95 beskeder pr. sekund, når de ikke reagerer på ændringer i $\hat{n}$. Det større antal beskeder, der modtages over tid, betyder at robotterne gennemsnitligt har flere andre robotter i nærheden, når der reageres på ændringer i $\hat{n}$, end



Figur 6.1: Kommunikation mellem forbipasserende RCX'er [GVSM02b]

når der ikke reageres på ændringer i $\hat{n}$. Således kan *situated* egenskaber ved kommunikationen anvendes til at holde robotterne samlet i en gruppe.

[GVSM02b] præsenterer en implementation af Ad Hoc netværk på LEGOs RCX. En af de centrale observationer, som [GVSM02b] gør, er at mobile robotter, der bevæger sig forbi hinanden, har et begrænset tidsrum, hvor de er i stand til at kommunikere med hinanden, som illustreret på figur 6.1. Dette tidsrum approximeres i [GVSM02b] ud fra eksperimentelle observationer til omkring ét sekund, men dette er i høj grad afhængigt af robotternes hastighed, deres indbyrdes orientering og bevægelse undervejs. Implementationen i [GVSM02b] bygger på, at robotterne periodisk udsender data, og når en robot modtager sådan en datapakke, vil den besvare med sine egne data. Dette fortsætter, indtil en af robotterne ikke modtager noget svar, hvorefter de begge fortsætter med den periodiske udsendelse af datapakker. [GVSM02b] opbygger på denne måde et datalinklag, der benytter spontant opståede forbindelser mellem robotter til at kommunikere over. Hver robot har i [GVSM02b] et lager af data, som den ønsker at sende videre. Disse data kan den selv have indhentet gennem sensorer, eller den kan have modtaget dem fra andre robotter. Da forbindelserne mellem robotterne er kortvarige, kan robotter ikke nå at sende alle de data, de ønsker. Dette løser [GVSM02b] ved at implementere en *gossip*-protokol, hvor hver enkelt robot bestemmer en prioritering af pakkerne. Denne prioritering kan implementeres på mange måder. Den implementation, som [GVSM02b] har valgt, er first in - first to be promoted.

I dette afsnit vil det blive undersøgt om situated kommunikation kan anvendes til at forbedre *neighbour discovery* i Ad Hoc netværk. Således at der over tid, vil være flere forbindelser mellem knuder i netværket. Desuden

Figur 6.2: Søgeområdet

undersøges det om det ved at påvirke robotternes bevægelse er muligt at ændre varigheden af forbindelserne i forhold til [GVSM02b].

6.1 Omgivelserne

De omgivelser, som alle forsøg i dette afsnit foregår i, består af en rektangulær bane, som er omgivet af en væg på en side og omgivet af en mur bygget af LEGO på de tre andre sider. Banen er 160 cm. gange 110 cm., og der er ingen forhindringer på selve banen.

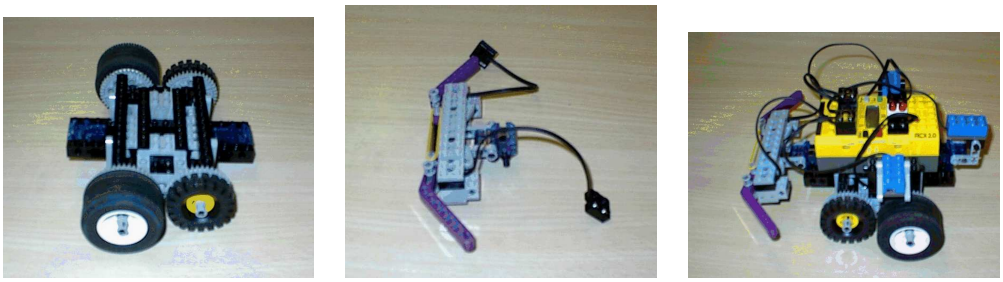
6.2 Robotternes udformning

Robotterne er bygget på en *Driving Base*, figur 6.3.A, fra [Teh97], hvorpå der er påsat hjul. På denne *Driving Base* er placeret en *Double Bumber* [Teh97], figur 6.3.B. Ud over dette er der tilføjet 3 lyssensorer - én placeret på hver af siderne og en placeret bagpå *Driving Base*. RCX'en placeres ovenpå denne driving base, og motorerne på *Driving Base* tilsluttes output A og C. På input 1 placeres den venstre trykføler fra *Double Bumper* samt den venstre lyssensor. På input 2 placeres den bageste lyssensor, og på input 3 placeres den højre trykføler samt den højre lyssensor. Input 1 og 3 håndteres som beskrevet i appendix A.3.3. Den færdige robot kan ses på figur 6.3.C.

6.3 Stand still-forsøg

I dette indledende forsøg illustreres det, at RCX'ens lyssensorer potentielt kan anvendes til at opnå viden om, hvor en afsender af infrarøde beskeder befinder sig.

Til dette benyttes to RCX'er. Den ene RCX er i rollen som fyrtårn. Fyrtårnet udsender periodisk infrarøde beskeder i form af *Heartbeats*. Den anden



A: *Driving Base* B: *Double Bumper* C: Den færdige robot

Figur 6.3: Opbygningen af robotterne

RCX er i rollen som søger. Søgerens opgave er at finde fyrtårnet. Søgeren er bygget som beskrevet i afsnit 6.2, mens fyrtårnet blot er en RCX uden tilbyggede sensorer eller aktuatorer.

I dette forsøg kan søgeren antage at den er indefor fyrtårnets infrarøde lyskegle. Det er således nok for søgeren at ændre sin orientering i forhold til fyrtårnet.

6.3.1 Robotternes kontrolprogram

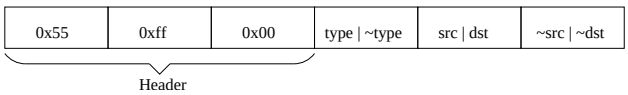
Dette forsøg anvender en forsimplet udgave af det *three way handshake*, der er beskrevet i afsnit 5. Forsøget anvender et stationært fyrtårn, der udsender periodiske *Heartbeats* (SYNs), mens søgeren lytter efter disse *Heartbeats*. Målet er, at søgeren skal finde fyrtårnet og opnå en indbyrdes placering således, at et *three way handshake* er muligt.

Søgeren kan i dette forsøg antage, at den starter inden for fyrtårnets infrarøde lyskegle, og at tovejskommunikation således er mulig, hvis søgeren blot justerer sin orientering.

Forsøget foretages både med en søgeadfærd, der benytter lyssensorer til at aflæse signalets fysik og en søgeadfærd, der kun benytter IR-transceiveren. Formålet er at undersøge, om søgeadfærden bliver mere effektiv - det vil sige om det påkrævede *three way handshake* bliver gennemført hurtigere, hvis signalets fysiske egenskaber benyttes.

6.3.2 Benyttelse af kommunikation

Da robotterne i dette forsøg har faste roller, er kommunikationsmodulet blevet forsimplet således, at de overflødige ting er skåret væk. De eneste pakkertyper i dette forsøg er *Heartbeat*, *Heartbeatreply* og *Heartbeatreplyreply*. Adresser er i dette forsøg repræsenteret som 4 bits. Adressen $0x0$ er reserve-



Figur 6.4: Pakkeformat

ret som en broadcastadresse, mens fyrtårnet og søgeren har adresserne $0x1$ og $0x2$

Pakkeformatet er desuden forsimplet. I dette forsøg bliver pakkerne afsendt med en header efterfulgt af en byte med pakketype og to bytes med source og destination, som angivet på figur 6.4. Headeren på pakkerne er konstant bestående af de tre bytes $0x55$, $0xff$ og $0x00$. Herefter følger en byte med pakkens type. Der anvendes fire bits til typen, og den afsendte byte har formatet: $(type << 4) + (\sim type)$. De fire første bits angiver altså typen og de fire sidste er typen bitvist negeret. Source og destination bliver afsendt som hver fire bits i den næste byte. Den sidste byte er to gange fire bits med henholdvis source og destination bitvist negeret.

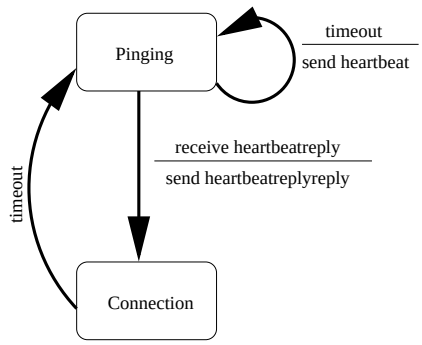
6.3.3 Den anvendte robotarkitektur i fyrtårnet

Fyrtårnet har kun én opgave - nemlig at udsende *Heartbeats* (SYN). Da fyrtårnet er stationært, er det kun den infrarøde transceiver, der benyttes som sensor/aktuator. Derfor er der kun et *behavior producing module* i fyrtårnet - nemlig et pingmodul, der udsender *Heartbeat* og besvarer *Heartbeatreplies*.

6.3.4 Pingmodulet

Pingmodulets opgave er, at instantiere et *three way handshake*. Derfor udsendes periodiske *Heartbeats*. Så længe fyrtårnet ikke har modtaget noget svar, vil den befinde sig i tilstanden *Pinging*, figur 6.5. I denne tilstand udsender den et *Heartbeat* efter hver 100 ms.

Intervallet på 100 ms. mellem afsendelse af *Heartbeats* er valgt, således at søgeren har tid til at svare, og således at pakkerne ikke kolliderer for ofte. Desuden skal intervallet have en så kort længde, at søgeren opfanger et *Heartbeat* hurtigt. Med 2400 $\frac{bits}{s}$ og 11 bits pr. byte, appendix B, tager det 27,5 ms. at afsende de 6 bytes, som beskederne i dette forsøg fylder. Dermed tager det minimalt 55 ms. fra fyrtårnet har afsendt et *Heartbeat*, til det har modtaget et svar. Da beregningerne mellem afsendelse af pakkerne er minimale, antages det, at disse ikke udgør noget væsentligt overhead i forbindelse med kommunikationen og *Heartbeats* derfor besvares hurtigt, efter de er modtaget. Derfor bør en afstand på 100 ms. mellem to *Heartbeats* være



Figur 6.5: Tilstande i pingmodulet

rigeligt til, at fyrtårnet kan antage, at hvis svaret ikke er kommet inden for 100 ms. så kommer det ikke.

Heartbeats indeholder information om fyrtårnets identitet, og de sendes til broadcastadressen. Hvis fyrtårnet modtager et *Heartbeatreply*, vil den sende et *Heartbeatreplyreply* retur, figur 5.3, og skifte til tilstanden *Connection*, figur 6.5.

Denne opførsel lader fyrtårnet være den ene part i et *three way handshake* og på denne måde oprette en tovejsforbindelse til søgeren.

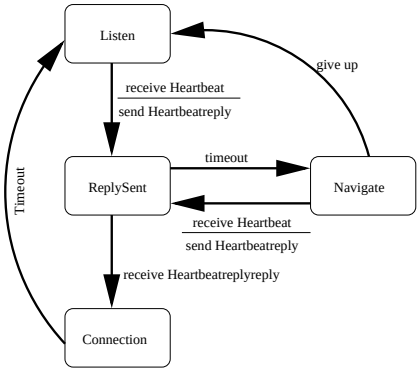
6.3.5 Den anvendte robotarkitektur i søgeren

Der er implementeret to forskellige søgere. Den ene benytter lyssensorer til at finde infrarøde signaler, den anden gør ikke. De to forskellige implementationer adskiller sig udelukkende ved denne egenskab i deres søgeadfærd. Da formålet med dette forsøg udelukkende er at illustrere, at det er muligt at navigere ved hjælp af de fysiske egenskaber i infrarød kommunikation, er der ikke tilføjet andre *behavior producing modules*.

6.3.6 Søgeadfærd

Der er implementeret to versioner af søgeadfærden - én der benytter lyssensorerne og én der ikke gør. Det *behavior producing module*, der implementerer søgeadfærden har 4 tilstande, figur 6.6:

1. *Listen*
2. *ReplySent*
3. *Navigate*



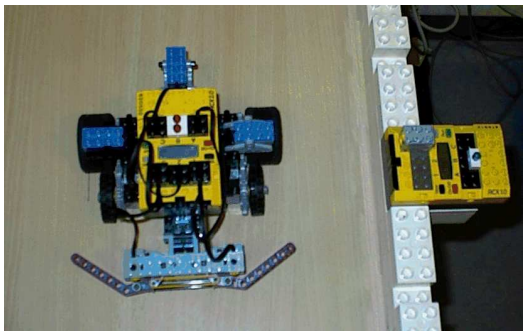
Figur 6.6: Tilstande i Seekmodulet

4. *Connection*

Så længe robotten ikke har modtaget et *Heartbeat*, vil seekmodulet være i tilstanden *Listen*. Når søgeren er i denne tilstand, vil den forsøge at ændre sin orientering ved at dreje.

Hvis robotten i tilstanden *Listen* modtager et *Heartbeat*, stopper den, besvarer *Heartbeat* og skifter til tilstanden *ReplySent*. I tilstanden *ReplySent* vil seekmodulet undersøge, om der er modtaget et *Heartbeatreplyreply*. Hvis dette er tilfældet skiftes til tilstanden *Connection*. Ellers gensendes *Heartbeatreply* i alt maksimalt 5 gange. Hvis *Heartbeatreply* ikke er besvaret efter 5 forsøg, skiftes til tilstanden *Navigate*. Antallet af gensendelse er valgt til 5 udfra empiriske erfaringer med robotterne. Der kan muligvis opnås forbedringer ved at vælge et andet antal. I tilstanden *Navigate* antages det, at søgeren er inden for kommunikationsradius af fyrtårnet, men ikke er i stand til at returnere beskeden. Dette skyldes, at søgeren ikke er orienteret korrekt i forhold til fyrtårnet. Derfor drejes robotten, og når robotten igen modtager et *Heartbeat*, stoppes rotationen, det modtagne *Heartbeat* returneres, og der skiftes tilbage til *ReplySent*. I tilstanden *Connection* antages det, at robotterne kan kommunikere med hinanden. Hvis dette ikke længere er tilfældet, vil et *timeout* sørge for, at modulet skifter tilbage til tilstanden *Listen*.

I tilstandene *Listen* og *Navigate* leder robotten efter et signal, og dermed er der potentiale i at anvende signalets *situated* egenskaber til at øge muligheden for at finde signaler. Der er lavet to forskellige implementationer af søgeadfærden. Den ene drejer til en tilfældig side i tilstandene *Navigate* og *Listen*. Den anden benytter lyssensorer placeret bagpå og på siderne af robotten til at lede efter et infrarødt signal. Hvis den registrerer lys i én af lyssensorerne, drejer den mod den side, ellers vælges en tilfældig side.



Figur 6.7: Forsøgsopstilling

6.3.7 Forsøgsopstilling

Søgeren placeres i en afstand på 10 cm lige foran fyrtårnet i vinkler med 90° interval fra 0° til 270°, figur 6.7. Disse vinkler er placeret således, at der i startpositionerne 90°, 180° og 270° er en lyssensor orienteret mod fyrtårnet. For startpositionen 0° er søgerens infrarøde transceiver orienteret mod fyrtårnet. De målte værdier for 0° kan derfor anskues som mindste værdierne for oprettelse af en forbindelse, da søgeren i denne startposition ikke skal bevæge sig for at opnå forbindelse.

Der foretages målinger, hvor lyssensorerne bruges, og hvor lyssensorerne ikke bruges.

6.3.8 Resultat

Tiden fra forsøget starter, og til der er oprettet forbindelse mellem søger og fyrtårn, kan opdeles i to dele, se formel 6.2: Tiden før søgeren modtager det første *Heartbeat*, T_{seek}, og tiden efter, hvor søgeren skiftevis forsøger at besvare *Heartbeats* og justere sin orientering, T_{adjust}.

$$T_{total} = T_{seek} + T_{adjust} \tag{6.2}$$

Søgeren måler T_{seek}, samt tiden fra den starter til der er oprettet forbindelse til fyrtårnet, T_{total}.

Resultatet af målingerne kan ses i tabel 6.1, tabel 6.2 og figur 6.8. Desuden er alle indsamlede data i appendix D. For målinger, hvor lyssensorerne ikke benyttes, er der angivet to værdier for hver vinkel. Disse to værdier er henholdsvis gennemsnitstiden for det tilfælde, hvor søgeren drejer den lange vej rundt, og det tilfælde hvor søgeren drejer den korte vej rundt.

Der er en tendens til at værdierne når lyssensorerne anvendes ligger lidt over værdierne når søgeren drejer den rigtige vej uden at benytte lyssenso-

T_{seek} med brug af lyssensorer

Vinkel	Tid / s
0	0.141
90	0.635
180	1.821
270	0.596

T_{seek} uden brug af lyssensorer aktive

Vinkel	Tid / s	Tid / s
0	0.127	–
90	2.663	0.525
180	1.650	–
270	2.768	0.464

Tabel 6.1: T_{Seek}

T_{total} med brug af lyssensorer

Vinkel	Tid / s
0	0.300
90	2.095
180	3.403
270	2.132

T_{total} uden brug af lyssensorer aktive

Vinkel	Tid / s	Tid / s
0	0.249	–
90	4.107	1.919
180	3.031	–
270	4.162	2.019

Tabel 6.2: T_{total}

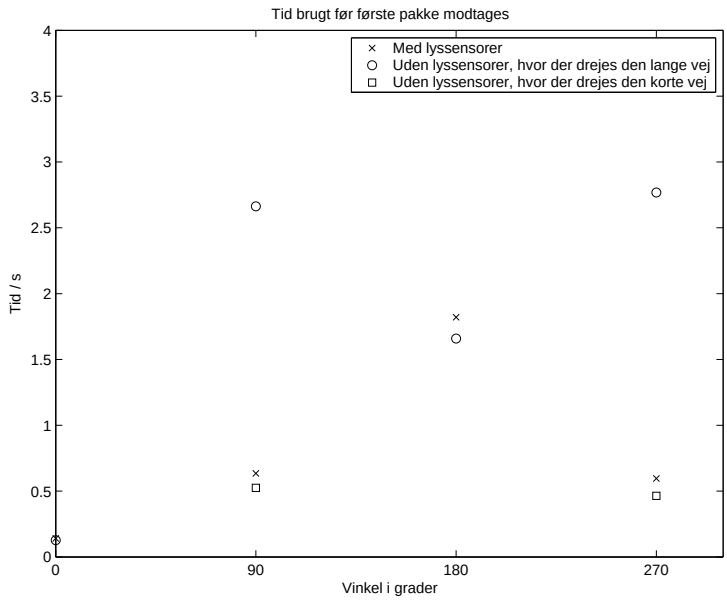
T_{adjust} med brug af lyssensorer

Vinkel	Tid
0	0.159
90	1.460
180	1.582
270	1.536

T_{adjust} uden brug af lyssensorer aktive

Vinkel	Tid
0	0.122
90	1.429
180	1.381
270	1.454

Tabel 6.3: T_{adjust} udregnet som $T_{total} - T_{seek}$



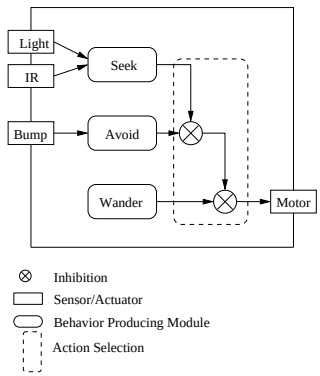
Figur 6.8: T_{seek} med og uden lyssensorer

rerne. Denne store forskel skyldes til dels enkelte målinger, som afviger meget fra de øvrige. Det kan for eksempel ses i appendix D, at der en måling ved 0°, som ligger på 0.274 ms, hvilket er næsten en fordobling i forhold til det forventede. Denne store afvigelse skyldes sandsynligvis at et *Heartbeat* er gået tabt. Da denne systematiske fejl er lille i forhold til den gevinst, som opnås ved at dreje den rigtige vej, så er det ikke undersøgt nærmere, hvordan den er opstået.

Der er en markant forskel i vinklerne 90° og 270°. Dette skyldes at søgeren ved at benytte lyssensorerne drejede den korteste vej rundt i alle kørsler, mens den uden lyssensorerne drejede til en tilfældig side, hvilket resulterede i, at den tre gange ved 90° og fire gange ved 270° drejede den korte vej rundt. Ved at udregne T_{adjust}, tabel 6.3, kan man se, at forskellen i værdierne næsten udelukkende stammer fra fordelen ved, at der skal drejes kortere, når lyssensorerne anvendes.

Det er altså muligt at benytte lyssensorerne til at opnå viden om, hvor en afsender af en infrarød besked er placeret, og det er muligt at benytte denne information til at forbedre søgningen efter afsenderen.

Det skal dog stadig undersøges, om denne information også er brugbar, når søgeren ikke kan antage, at den befinder sig i fyrtårnets lyskegle, og forbindelsen skal oprettes til en mobil anden part i stedet for et fyrtårn.



Figur 6.9: Skematisk tegning over søgerens interne arkitektur

6.4 Wander-forsøg

I foregående forsøg er det blevet illustreret, at man ved benyttelse af lyssensorer kan opnå viden om, hvor en afsender af en infrarød besked befinder sig. I dette afsnit anvendes dette i et forsøg, hvor to robotter bygget efter beskrivelsen i afsnit 6.2 begge bevæger sig, mens de forsøger at oprette en kommunikationsforbindelse. Denne kommunikationsforbindelse skal benyttes til forsendelse af en bestemt mængde pakker, hvorefter den skal nedlægges.

6.4.1 Robotternes kontrolprogram

En skematisk tegning over robotternes indvendige arkitektur kan ses af figur 6.9. Robotternes arkitektur består af tre *behavior producing modules*: Wander, Avoid og Seek. Output fra disse *behavior producing modules* kombineres af et *action selection module*. *Action selection* funktionen er et simpelt præcedenshierarki, hvor Seek har højest prioritet, Avoid næsthøjest og Wander lavest.

Wander-modulet implementerer en *random walk* funktion, som skal være aktiv, når de øvrige *behavior producing modules* ikke er aktive. Avoid implementerer *obstacle avoidance* ved hjælp af to fremadrettede trykfølere. Tilsammen skal Avoid og Wander implementere en sikker tilfældig afsøgning af forsøgsområdet.

6.4.2 Søgeadfærd

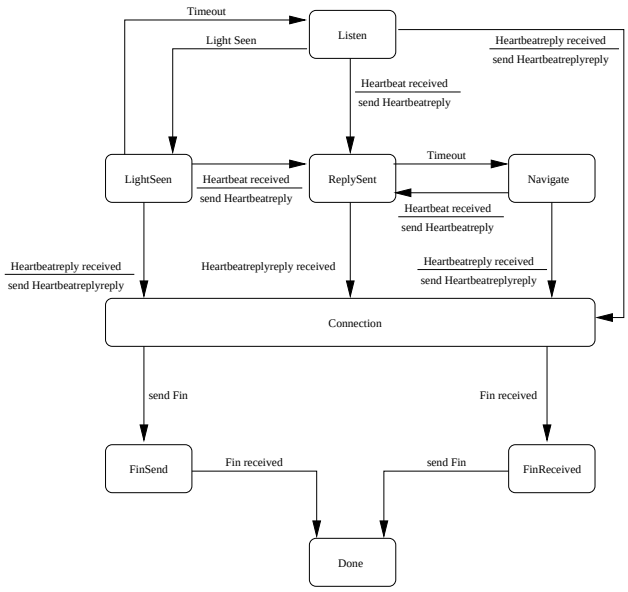
I modsætning til det første forsøg, hvor der var en klar ansvarsfordeling mellem søger og fyrtårn, er der i dette forsøg to ligestillede robotter. De skal således begge være i stand til at spille begge roller i et *three way handshake*.

Derfor er det nødvendigt med en ændret opbygning af robotternes søgeadfærd. Der er i dette forsøg følgende tilstande i søgeadfærden:

1. *Listen*
2. *LightSeen*
3. *ReplySent*
4. *Navigate*
5. *Connection*
6. *FinSend*
7. *FinReceived*
8. *Done*

I tilstanden *Listen* vil søgeadfærden ikke forsøge at opnå kontrol med robottens motorer. Når søgeadfærden er i denne tilstand, vil Wander- og Avoid-modulerne alene bestemme robottens bevægelse. I tilstande *Listen* undersøges det, om robotten har modtaget en infrarød besked. Er dette tilfældet, besvares denne, og der skiftes tilstand til *ReplySent*, hvis den modtagne besked var et *Heartbeat*, og til *Connection* hvis den var et *Heartbeatreply*, figur 6.10. Hvis robotten kan se en afsender af infrarøde beskeder ved hjælp af sine lyssensorer, skiftes til tilstanden *LightSeen*. Robotten vil desuden i tilstanden *Listen* udsende periodiske *Heartbeat* med et interval på 150 ms. Den lidt længere periode i mellem udsendelse af *Heartbeats* skyldes den risiko for kollisioner, der forekommer når to enheder udsender Heartbeats, i modsætning til forgående forsøg, hvor kun fyrtårnet udsender *Heartbeats*. Den længere periode minimerer disse kollisioner. Om perioden på 150 ms er optimal, eller der kan opnås et bedre resultat ved at ændre denne er periode er ikke undersøgt til bunds.

I tilstanden *LightSeen* vil robotten dreje til den side, hvor den så den markante lysværdi. I denne tilstand tager Seek kontrol over robottens motorer, og output fra Wander og Avoid undertrykkes. I tilstanden *LightSeen* vil robotten fortsat udsende periodiske *Heartbeats*, og den vil fortsat undersøge, om den har modtaget beskeder. Der skiftes tilstand, hvis der modtages en infrarød besked. Dette fungerer som i tilstanden *Listen*. Hvis robotten ikke i et stykke tid har set nogen markant lysværdi på nogle af sine lyssensorer, vil Seek-modulet skifte tilbage til tilstanden *Listen*.



Figur 6.10: Tilstande i søgeadfærd. Af hensyn til overskueligheden er tilstandsskift, som skyldes *timeout*s ikke tegnet ind på figuren

I tilstanden *ReplySent* vil Seek-modulet stoppe robotten og lytte efter svar på det afsendte *Heartbeatreply*. Hvis robotten modtager et *Heartbeatreplyreply*, så afsendes den første datapakke og der skiftes til tilstanden *Connection*. I tilstanden *ReplySent* sendes svaret i alt fem gange, hvis der efter fem forsøg ikke er modtaget noget *Heartbeatreplyreply*, så skiftes til tilstanden *Navigate*.

I tilstanden *Navigate* vil seekmodulet få robotten til at dreje rundt om sig selv i et forsøg på at opnå en forbindelse. Hvis robotten i tilstanden *Navigate* modtager et *Heartbeat* skiftes tilbage til tilstanden *ReplySent*. Robotten vil fortsat i tilstanden *Navigate* udsende periodiske *Heartbeats*, hvis den modtager et *Heartbeatreply* på et af disse, bliver det besvaret og der skiftes til tilstanden *Connection*.

I tilstanden *Connection* antages det, at der er oprettet forbindelse til en anden robot. Denne forbindelse benyttes til at kommunikere over som beskrevet i afsnit 5. Nedlæggelsen af en forbindelse sker ved, at en robot, når den ikke ønsker at afsende flere data, sender en Fin-besked og skifter til tilstanden *FinSend*. Modtageren af en Fin-Besked skifter til tilstanden *FinReceived*, men den kan stadig have ikke afsendte data. Derfor er det fortsat muligt for robotterne at kommunikere i tilstandene *FinSend* og *FinReceived*. Forbindelsen nedlægges først, når robotten i *FinReceived* ikke har flere data,

og derfor afsender en Fin besked, som robotten i tilstanden *FinSend* modtager. Herefter skiftes til tilstanden *Done*, da forsøget er færdigt. Hvis formålet med forsøget krævede, at robotterne ikke stoppede efter en forbindelse, men i stedet kørte videre, skulle der skiftes til tilstanden *Listen* i stedet.

6.4.3 Forsøgsopstilling

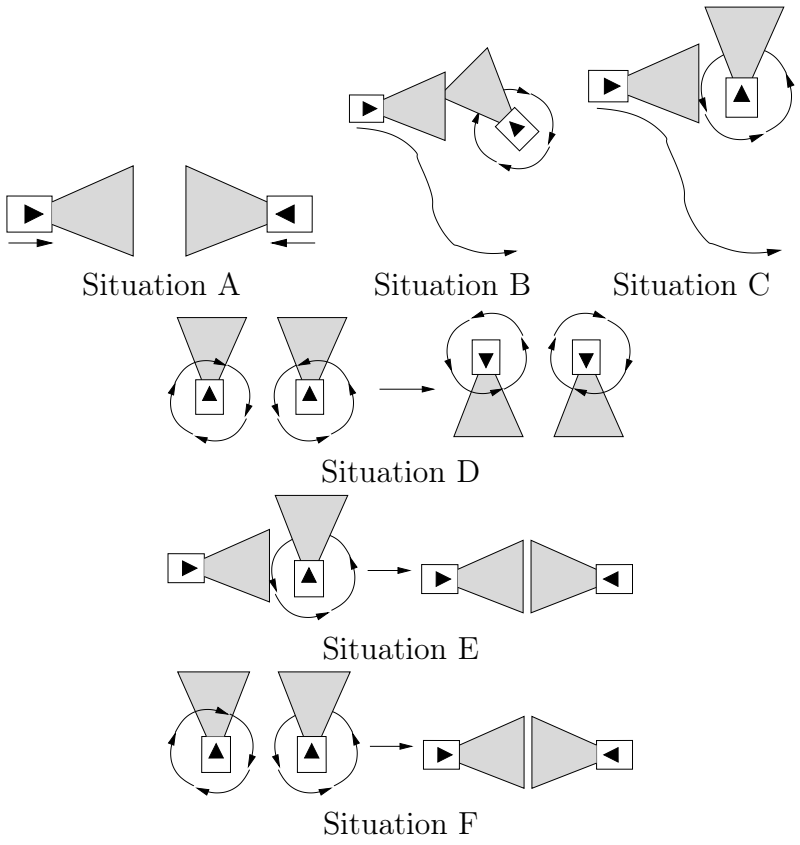
Ved empiriske undersøgelser blev det noteret, at der i forbindelse med oprettelse af kommunikationen opstår seks typiske situationer: Den første består blot i, at robotterne opnår forbindelse uden lyssensorerne har været i brug. Denne situation er i dette forsøg kaldet situation A og er skitseret på figur 6.11.

Den anden situation består i, at én af robotterne modtager en infrarød pakke, uden at der oprettes en forbindelse. Når en robot modtager en infrarød pakke, vil den først besvare denne. Hvis dette ikke medfører en oprettelse af en forbindelse, så vil robotten begynde at rotere for at orientere sig anderledes i forhold til afsenderen. Hvis afsenderen i dette tidsrum har bevæget sig væk, vil forbindelsen ikke blive oprettet, og robotten vil efter et stykke tid give op og fortsætte søgningen. Denne situation bliver i dette forsøg kaldet B og er skitseret i figur 6.11.

Den tredje situation består i, at én af robotterne ser en markant lysværdi på én af sine lyssensorer. Denne markante lysværdi kan stamme fra enten udsendelsen af en infrarød besked eller dioden på en af en anden robots lyssensorer. Der kan ikke skelnes mellem disse lysværdier, i begge tilfælde registreres det blot ved, at der er set en markant lysværdi på pågældende lyssensor. Når robotten ser en markant lysværdi, vil den dreje over mod den lyssensor, som registrerede den. Hvis afsenderen i mellemtiden har bevæget sig væk, kan der ikke oprettes en forbindelse. I stedet vil robotten efter et tidsrum opgive at opnå en forbindelse og derefter fortsætte søgningen. Denne situation bliver i dette forsøg kaldet C og er skitseret i figur 6.11.

Den fjerde situation består i, at begge robotter samtidigt opfanger en markant lysværdi, men ikke opnår en forbindelse. Dette vil resultere i, at begge robotter vil dreje for at opnå en forbindelse. Det sker dog, at robotterne i denne situation drejer forbi hinanden således, at de ikke opnår en forbindelse. Denne hændelse kan gentage sig således, at robotterne drejer frem og tilbage flere gange, inden de opgiver at opnå forbindelse. Denne situation kaldes i dette forsøg D og er skitseret på figur 6.11.

Den femte situation består i, at én robot ser en markant lysværdi på en af lyssensorerne og derefter opnår en forbindelse med den anden robot. Denne situation svarer til situation C, blot med den forskel at afsenderen ikke har bevæget sig væk, inden robotten når at dreje rundt og etablere en forbindelse.

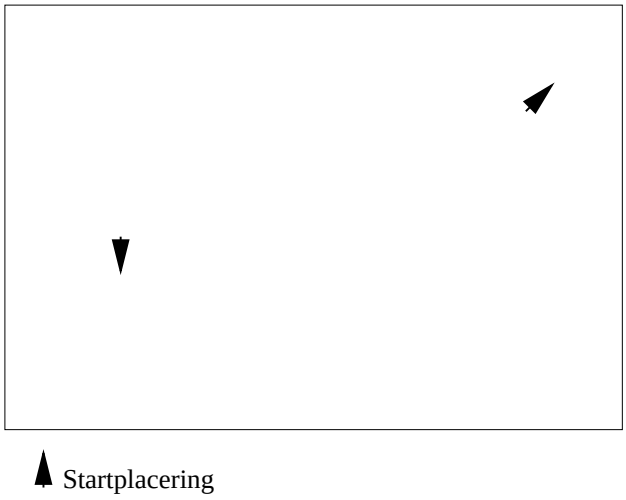


Figur 6.11: Situation A til F

Denne situation kaldes situation E og er skitseret i figur 6.11.

Den sjette situation består i, at begge robotter ser en markant lysværdi og derefter opnår at etablere en forbindelse. Denne situation svarer til situation D bortset fra, at det i dette tilfælde lykkedes de to robotter at opnå en forbindelse, inden de opgiver. Denne situation kaldes situation F og er skitseret i figur 6.11.

Der er foretaget 50 kørsler. En kørsel går fra, robotterne bliver startet, til de har oprettet en forbindelse og sendt deres datapakker over denne forbindelse. Der er to robotter i forsøget. Deres startpositioner ændres ikke i løbet af de 50 kørsler. Startpositionerne er angivet i figur 6.12. De to robotter får numrene 1 og 2. Disse numre er også deres adresser i kommunikationsprotokollen. Robot 1 har ti datapakker, som den skal afsende, mens robot 2 har fem datapakker, den skal afsende, før kørslen er slut. Det bliver noteret, hvor mange gange hver af de forskellige situationer opstår. En kørsel slutter altid med en situation af typerne A, E eller F.



Figur 6.12: Startpositioner

situation	antal
A	21
B	44
C	15
D	17
E	6
F	23

Tabel 6.4: Resultat af wander-forsøg

6.4.4 Resultat

Resultatet af dette forsøg kan ses i tabel 6.4. A, E og F er de situationer, hvor robotterne har opnået en forbindelse. Mens tilfældene B, C og D er de situationer, hvor mindst én af robotterne har haft viden om, at den anden er i nærheden, men ikke har kunnet opnå forbindelse. Af de 50 kørsler har de oprettede forbindelser i 58% af tilfældene været resultat af brug af lyssensorerne, mens forbindelse i 42% af tilfældene er opnået uden at benytte lyssensorerne. Desuden kan man se på, hvor tit robotterne med en given information om hinanden har kunnet opnå forbindelse. Hvis robotterne ikke har information om hinanden fra lyssensorerne, er succesraten $\frac{A}{A+B} = 32,3\%$. Hvis den ene robot har opfanget en lysværdi, er succesraten $\frac{E}{C+E} = 28,6\%$. Hvis begge robotterne har viden om hinanden ud fra lyssensorer, er succes-

raten $\frac{F}{D+F} = 57.5\%$.

Hvis kun den ene robot har viden om den anden - ved hjælp af lyssensorer eller den infrarøde transceiver - er succesraten altså omkring 30%. Hvis begge robotterne har viden om hinanden - ved hjælp af lyssensorer - er succesraten markant større. Der synes altså at være en klar sammenhæng mellem, hvor meget robotterne ved om hinanden, og hvor stor chance robotterne har for at udnytte denne information til at oprette en forbindelse. Dette resultat er forventeligt, da robotterne ikke bevæger sig væk fra hinanden, når de begge har opdaget den anden i nærheden.

I de 50 kørsler opstår der i alt 126 situationer, hvor de to robotter har hel eller delvis forbindelse. Af disse 126 situationer skyldes de 61 svarende til 48.4%, at robotterne anvender lyssensorerne. Muligheden for, at de to robotter opdager hinandens tilstedeværelse, bliver altså næsten fordoblet ved at anvende lyssensorerne. Dette betyder dog ikke nødvendigvis, at forbindelsen som gennemsnit bliver oprettet hurtigere, hvis man anvender lyssensorerne frem for ikke at anvende dem. Af de forskellige beskrevne situationer er D og F dem, hvor robotternes adfærd ændres mest i forhold til situationerne, hvor robotterne ikke anvender lyssensorerne. I disse situationer kan robotterne stå og dreje frem og tilbage uden at opnå kontakt, men alligevel se værdier på lyssensorerne. Robotterne kan altså i disse situationer i lang tid forsøge at oprette en forbindelse uden, at det lykkes.

I denne situation er det således et problem at begge robotter aktivt forsøger at opnå forbindelse ved at rotere. Dette gør at der opstår en situation, hvor igen og igen drejer forbi hinanden, da de ikke har nogen mulighed for at fordele roller i mellem sig, f.eks. som søger og fyrtårn. Hvis de havde en eller anden måde at bestemme hvem, der drejer, og hvem, der står stille. Så er det muligt at der vil opstå flere forbindelser.

Problemet er at robotterne ikke har nogen mulighed for at udveksle informationer før de har oprettet en forbindelse, og deres mulighed for at opnå informationer om hinanden er særdeles begrænset. Denne viden om hinanden er begrænset til den, der opnås gennem lyssensorer og den infrarøde transceiver. Hvis robotterne havde flere eller bedre sensorer, så de kunne opnå viden om modpartens orientering og bevægelse, så kunne denne information benyttes til at oprette en forbindelse.

Der eksisterer muligheder, som ikke er undersøgt i dette forsøg. Et eksempel kunne være at stoppe rotationen, når der er drejet en halv eller en kvart omgang. Således at robotten orienterer sin infrarøde transceiver i den retning, som lysværdien blev målt i. Dette kan implementeres ved en timer, der afgør, hvor lang tid, der skal drejes. Et problem med denne algoritme er at RCX'ens motorer ikke altid kører lige hurtigt. De er f.eks. meget afhængige af RCX'ens batteriniveau. Et andet muligt problem er at den opdagede

lyskilde ikke nødvendigvis er placeret direkte ud for lyssensoren. Lyskilden kan ses, hvis den befinder sig i en smal kegle ud for lyssensoren. Lyssensoren giver altså ikke nogen helt præcis retning til lyskilden. Om RCX'en overhovedet kan foretage rotationer så præcist at det udgør en forbedring er ikke undersøgt.

6.5 Delkonklusion

I det første forsøg vises det at to parter i kommunikationen kan anvende lyssensorer til aktivt at forbedre mulighederne for kommunikation. Dette kan ske ved, at når søgeren modtager det første signal fra fyrtårnet, er søgeren ikke nødvendigvis i stand til at sende et svar tilbage til dette. Dette kan skyldes, at søgerens relative orientering i forhold til fyrtårnet er forkert. Det kan løses ved, at søgeren roterer for således at opnå en orientering, hvor tovejskommunikation med fyrtårnet er mulig. Denne adfærd var succesfuld hver gang. Derfor kan det konkluderes at chancen for at opnå tovejskommunikation bliver større, hvis robotterne aktivt forsøger at opnå en bedre indbyrdes orientering, når kommunikationen startes. Dette er i modsætning til den strategi, der anvendes i [GVSM02b] og [GVSM02a], hvor robotterne ikke aktivt forsøger at forlænge tiden, hvor kommunikation mellem dem er mulig.

I det andet forsøg bliver det undersøgt om lyssensorerne også giver en fordel, når begge parter i kommunikationen er mobile. Når dette er tilfældet bliver billedet mere sløret. Der opstår situationer, hvor begge robotter påtager sig den rolle, som søgeren har i første forsøg. Dette resulterer i at robotterne begge drejer rundt om hinanden uden at en forbindelse bliver oprettet. Hvis robotterne skal kunne fordele rollerne i mellem sig, så er det nødvendigt med en eller anden måde at forhandle disse roller. Da robotterne ikke kan kommunikere eksplicit inden forbindelsen er oprettet, så er det ikke muligt at anvende de algoritmer, der er beskrevet i afsnit 4. Et alternativ til eksplicit forhandling er, at robotterne har tilstrækkelig viden om hinandens bevægelse og orientering til, at de entydigt kan bestemme fordelingen af rollerne.

Under forsøgene blev det observeret at lyssensorerne har et smalt "synsfelt". De kan således kun se lys, som falder direkte ind i lyssensoren. Dette skyldes sandsynligvis at dioderne er tilbagetrukket i forhold til sensorens forkant, appendix A, og sensorens plastikkant derfor skygger for dioden.

Forsøgene viser begge, at der er en fordel ved at benytte lyssensorer til at finde en afsender af infrarøde beskeder. Denne fordel vil utvivlsomt kunne gøres større ved at benytte lyssensorer med andre karakteristika eller måske endda ved at ændre søgeadfærden når en markant lysværdi aflæses, således at rotationen kunne kontrolleres, og der kun drejes 90° eller 180°. Dette vil

muligvis kunne løse problemet, hvor robotterne drejer forbi hinanden.

De fordele ved *situated* kommunikation, der er undersøgt i dette afsnit, bygger på at kommunikationen er retningsbestemt. De teknikker, der er anvendt, medvirker til at gøre kommunikationen mindre afhængig af at de to parter passerer gennem hinandens lyskegle. Ved kommunikation som f.eks. radiokommunikation, der ikke er retningsbestemt, er det ikke længere nødvendigt for en søger at rotere for at opnå en forbindelse. Det vil derfor være nemmere at oprette forbindelse mellem enheder, der anvender radiokommunikation. I dette tilfælde kan kommunikationens egenskaber som et *situated* fænomen dog stadig benyttes. For det første kan det være en fordel aktivt at forsøge at forlænge varigheden af forbindelser ved at ændre enhedernes bevægelse. For det andet indeholder signalets fysik også for radiokommunikation oplysninger om afsenderens afstand til modtageren, ligesom det er muligt at vide om afsenderen kommer tættere på eller bevæger sig længere væk.

7 Konklusion

I indledningen blev følgende spørgsmål opstillet i forbindelse med en undersøgelse af mulighederne for at anvende teknikker fra robotforskningen til at opnå bedre muligheder for trådløs kommunikation mellem mobile enheder.

- Kan skalerbarheden af kommunikationsprotokoller i multirobotsystemer forbedres, ved at udnytte at visse informationer kun har lokal interesse?
- Kan antallet af forbindelser i et netværk af mobile enheder øges ved hjælp af teknikker fra robotforskningen?
- Kan stabiliteten af forbindelserne i et netværk af mobile enheder øges ved hjælp af teknikker fra robotforskningen?
- Kan hastigheden, hvormed data spredes til de relevante enheder i netværket øges ved hjælp af teknikker fra robotforskningen?

De første spørgsmål adresserer skalerbarheden af kommunikationsprotokoller. Meget forskning i multirobotsystemer antager eksistensen af et kommunikationsmedie, hvor alle kan kommunikere med alle. I afsnit 4 blev der præsenteret nogle problemer i forbindelse med skalerbarheden af disse kommunikationsprotokoller. Flere af de kommunikationsprotokoller, der blev præsenteret skalerer dårligt, netop fordi de antager at alle kommunikerer med alle, og at alle derfor modtager og skal behandle alle afsendte beskeder. [Rey87] blev præsenteret som en modsætning til dette. [Rey87] benytter fænomener som flokke af fugle og stimer af fisk som argument for at der eksisterer algoritmer, der skalerer bedre. Disse algoritmer vil benytte, at hver enkelt enhed kun skal koordinere sine handlinger med et konstant antal andre enheder i nærheden. Dette vil muligvis også kunne overføres til kommunikationsprotokoller. LIWAS [HEK+04] er et eksempel hvor dette er ønsket. I LIWAS har informationer kun har lokal interesse, derfor skal spredningen af disse informationer i netværket begrænses. De præcise detaljer om hvordan denne spredning af informationer kan begrænses, og hvordan dette system skalerer med antallet af deltagende enheder, fremgår dog ikke af [HEK+04]. En sådan teknik vil uden tvivl være en fordel for skalerbarheden inden for kommunikation mellem robotter. I multirobotsystemer har informationer ofte kun lokal betydning, da en robot, der befinder sig langt fra en opgave, næppe er den rette til at løse denne. Der har dog ikke forbindelse med dette speciale været mulighed for at afprøve disse ideer i praksis.

De to næste spørgsmål handler om oprettelsen og stabiliteten af forbindelser. Disse to hænger sammen med de begreber om *neighbour discovery* og

link maintenance, der blev præsenteret i afsnit 3. Det blev konkluderet at begrebet *situated* kommunikation [Stø01], ville kunne anvendes til at forbedre netop disse to aspekter af en ad hoc netværksprotokol. Dette er blevet afprøvet i forbindelse med den konkrete implementation af et ad hoc netværk, som er præsenteret i afsnit 5.

I forbindelse med *neighbour discovery* blev det i afsnit 6 benyttet at RCX'ens lyssensorer er i stand til at detektere signaler fra RCX'ens infrarøde transceiver. Dette blev anvendt til at forbedre muligheden for at opdage en RCX, der udsender infrarøde beskeder. *Neighbour discovery* implementeret i netværk bestående af LEGOs RCX bliver besværliggjort af at kommunikationen er retningsbestemt. Det er således nødvendigt for at oprette en forbindelse at enhederne er orienteret korrekt i forhold til hinanden. Den implementerede algoritme gjorde, at enhederne drejede rundt om sig selv, når et signal blev detekteret. Dette resulterede i, at enhederne en gang i mellem drejede forbi hinanden uden at få oprettet en forbindelse. Dette problem består i, at begge robotterne påtager sig samme rolle svarende til rollen som søger i standstill forsøget. Det vil muligvis være nemmere at oprette en forbindelse, hvis de to enheder havde en mulighed for at koordinere deres roller. Da det ikke er muligt at kommunikere eksplicit under *neighbour discovery* proceduren, kan de forhandlingsprotokoller, der er beskrevet i afsnit 4 ikke anvendes. Et alternativt til eksplicit forhandling er at sørge for, at begge enheder har tilstrækkelig information om omgivelserne til entydigt at kunne fordele rollerne uden eksplicit kommunikation. For at opnå en sådan implicit forhandling er det nødvendigt, at enhederne er i stand til gennem sensorer at opnå tilstrækkelig information om omgivelserne. Da der på RCX'en kun er mulighed for at tilslutte tre sensorer foruden den infrarøde receiver, og disse sensorer endda begrænser sig til tryk- og lyssensorer, har det ikke været muligt at implementere en sådan implicit protokol. Af disse grunde er problemet, hvor enhederne drejer forbi hinanden, ikke blevet løst i dette speciale. Dette problem gjorde at eksperimenternes konklusioner ikke blev så entydige, som forventet. Det blev dog illustreret, at der er en fordel ved at anvende lyssensorerne til at opnå viden om afsenderens placering fremfor ikke at anvende lyssensorer. Dette resultat knytter sig tæt til, at kommunikationen var retningsbestemt, og at enhederne har begrænsede sensorer.

I forbindelse med *link maintenance* blev det anvendt, at enhederne står stille under kommunikationen. Dette bevirker at forbindelsen først brydes, når der ikke er flere data, der skal kommunikeres. Dette er den simpleste måde at opnå aktiv *link maintenance*, den er igen blevet bestemt af RCX'ens retningsbestemte kommunikation. Der kan være fordele ved at vælge denne aktive strategi når der opstår forbindelser, i modsætning til den spontane tilgang til kommunikation, som bliver anvendt i [GVSM02b]. Hvis spredningen

af informationer i netværket er vigtig for den samlede ydelse af systemet, og antallet af forbindelser er lavt, kan det være en fordel at vente med løsningen af andre opgaver indtil kommunikationen over en forbindelse er afsluttet. Et eksempel, hvor spredning af informationer i netværket er vigtig, er i forbindelse med visse *sensor networks*, hvor indsamlingen af data netop er selve opgaven for enhederne. Der er dog også situationer, hvor det ikke er muligt for de enkelte enheder at justere deres bevægelse for at opnå mere stabile forbindelser. I trafiksikkerhedssytemet LIWAS [HEK+04] er det ikke muligt for enhederne at ændre deres bevægelse for at optimere muligheden for kommunikation. I LIWAS er enhedernes primære opgave ikke at indsamle information, men derimod at fragte passagere fra et sted til et andet, gerne hurtigt. Dette er i direkte modstrid med ideen om aktiv *link maintenance*. Det er også muligt at forestille sig situationer, hvor det ikke er nødvendigt for enhederne at standse deres bevægelse fuldstændigt. Det kan være nok at nedsætte farten tilstrækkeligt til, at kommunikationen kan finde sted, inden enhedernes bevægelse tvinger kommunikationen til at ophøre. I forbindelse med forsøgene i afsnit 6 blev det anvendt, at enhederne stoppede deres bevægelse, når der opstod en forbindelse, og først bevægede sig igen, når kommunikationen over forbindelsen var slut. Dette resulterede i at enhederne, når de havde oprettet en forbindelse, fik alle deres data kommunikeret over denne forbindelse.

Det sidste spørgsmål handler i modsætning til de forgående to om den samlede kommunikation i hele netværket. Hvor de forgående to koncentrerer sig om fordelene set på en enkelt forbindelse mellem to robotter, så koncentrerer dette spørgsmål sig om netværkets samlede effektivitet. Dette spørgsmål er ikke blevet afklaret i dette speciale. Det vil kunne afklares ved at køre eksperimenter, som klarlægger, hvordan data kommer fra indhentningsstedet til opsamlingsstationen i et *mobile sensor network*. Dette eksperiment kunne bestå af en række mobile enheder, der indsamler informationer omkring omgivelserne og kommunikerer ved hjælp af den ad hoc protokol, der blev præsenteret i afsnit 5. Foruden de mobile dataindsamlere skulle der være en stationær opsamlingsstation, der modtager data fra de mobile enheder. I dette system kunne forskellige karakteristika omkring kommunikationen sammenlignes, når teknikkerne præsenteret i afsnit 6 anvendes, modsat når de ikke anvendes.

I forbindelse med alle resultater i dette speciale gælder det, at de er specifikke for den hardwareplatform, der er valgt til forsøgene. Da RCX'en benytter sig af retningsbestemt kommunikation, kan resultaterne ikke umiddelbart overføres til systemer, hvor kommunikationen har andre udbredningskarakteristika. At resultaterne er specifikke for en bestemt hardwareplatform, skyldes at eksperimenterne er blevet udført med fysiske robotter. Disse robotters adfærd er en kombination af deres kontrolprogram og deres fysik, dette stemmer

overens med begreber som *embodiment* og *situatedness*.

Alt i alt har dette speciale præsenteret nogle resultater, som tyder på, at brug af *situated* kommunikation under visse forudsætninger kan medvirke til at forbedre mulighederne for kommunikation i trådløse mobile ad hoc netværk.

8 English summary

In 1986 Rodney Brooks revolutionized robot research with the article *A Robust Layered Control System For A Mobile Robot* [Bro86]. This article argued that a move away from focus on reasoning, planing and modelling in robot systems to focus on situatedness, embodiment and emergence was needed in order to build robust.

Due to the falling cost and increasing availability a quite large research effort in multi robot systems and communication and coordination among robots has emerged. This research area largely is concerned with the protocols for negotiation of tasks and roles among robots. Among others [GM02], [Par98] and [WM01] represent work, where the focus is on implementing protocols for negotiating instead og the basis of communication among robots. These articles in large rely on the assumption that the robots has access to total communication - that is everybody talks directly to everybody. Less research has gone into the basis of the communication.

The main claim of this master thesis is that it is possible to take advantage of situatedness and embodiment when designing a communication protocol for a systems consisting of wireless mobile autonomous nodes. This claim is validated partly through theoretical discussion of the possibilities in situatedness and partly through a experimental evaluation of the ideas that emerged from the theoretical discussion.

The experimental method used in this thesis relies on implementation of an ad hoc network on a group of mobile robots. The robots used are based on the LEGO RCX module which gives both felxibility and access to shortrange communication. This experimental method was chosen above modelling and simulation which are widely used in the network research community. This choice was made because the focus on situatedness and embodiment required a implementation on real physical entities.

The experiments done in this master thesis have shown some interesting advantages of using situatedness. But the results are less significant than anticipated. This is due to the vast numbers of physical phenonemons that have influence when using physical robots. Sensors, actuatators and environment disturbs the clean picture which might have been achieved through simulation.

A RCX

RCX en er programmerbar enhed, der følger med LEGO MindStorms Robotics Invention System [Cap02]. RCX'en har en indbygget Hitachi H8/3292 microcontroller [Hit98], der giver RCX'en mulighed for at køre et kontrolprogram. RCX'en har mulighed for interaktion med omgivelserne gennem henholdsvis tre port til sensorer og tre porte til aktuatorer. Disse sensorer og aktuatorer kan kan skiftes, således at RCX'en kan benyttes til at bygge og eksperimentere med forskellige fysiske konfigurationer.

Dette afsnit vil beskrive de dele af RCX'en som er vigtige for dette speciale. Da RCX'ens mulighed for infrarød kommunikation anvendes en del i dette speciale, bliver denne beskrevet grundigere end de øvrige dele. Beskrivelsen af RCX'ens infrarøde kommunikation kan findes i appendix B og appendix C.

A.1 RCXens microcontroller

RCX'ens hjerte består af en Hitachi H8/3292 microcontroller. Denne microcontroller er integreret i en enkelt chip og indeholder de komponenter, der giver mulighed for at køre programmer på RCX'en.

Microcontrolleren benytter sig af 16 bits adresser, dette giver mulighed for i alt 65536 adresserbare enheder i hukommelsen. Dette adresserum er fordelt som angivet i tabel A.1. Af de i alt 64 kilobytes hukommelse er det altså omkring halvdelen, der er tilgængelig for program og data, de andre dele er reserveret til forskellige andre formål, for eksempel har registrene der anvendes til den serielle infrarøde kommukation adresser mellem 0xffc3 og 0xffdd.

Selve CPU'en på microcontrolleren har en clockfrekvens på 16 mhz. CPU'en har otte registre, hvoraf et benyttes som stackpointer, foruden disse er der også en 16 bit program counter og et 8 bits condition code register, der angiver status om for eksempel overløb. CPU'ens instruction set indeholder mulighed for 16 bits addition, 16 bits subtraktion, 8 bits multiplikation, 16 og 8 bits division, foruden logiske operationer [Cap02].

CPU'en har mulighed for at håndtere input og output fra forskellige enheder for eksempel det *serial communication interface*, der er beskrevet i appendix B. Kommunikationen med disse enheder foregår gennem forskellige interrupts og specielle device registers.

RCX'en har foruden de seks porte til sensorer og aktuatorer indbygget 4 knapper, et LCD display, en enhed, der måler batterinivaeu, en intern højtaler, en infrarød transceiver og forskellige timers [Cap02].

Adresser	Type	Indhold
0x0000 - 0x3fff	on-chip mask programmable ROM	H8/3292 interrupt vectors, RCX executive
0x8000 - 0xefff	off-chip RAM	program / data
0xf000	off-chip register	device register for RCX output ports
0xfd80 - 0xff7f	on-chip RAM	RCX interrupt vectors / program / data
0xff88 - 0xffff	on-chip register field	H8/3292 device registers

Tabel A.1: Fordeling af RCX'ens 16 bits adresserum [Cap02]

A.2 Aktuatorer

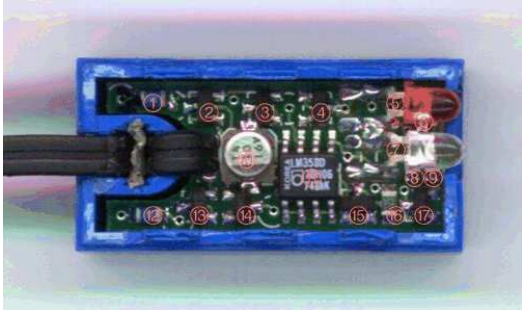
RCX'en har indbygget tre porte, hvor der kan tilsluttes eksterne aktuatorer. Disse porte har navngivet A, B og C.

De aktuatorer, der er anvendt i dette speciale, er henholdsvis motorer og lyselementer. Motorerne anvendes til at bygge RCX'en til en mobile robot, ved at forbinde dem til hjul ved hjælp af standard LEGO tandhjul. Når motorerne bliver tilkoblet en outputport, vil den have fire forskellige tilstande afhængigt af to bits, som er tilknyttet hver outputport. Disse tilstande er *float*, hvor motoren ikke er tændt, *brake*, hvor motoren heller ikke er tændt, men i modsætning til *float*, så bliver motoren bremset, de sidste to tilstande får motoren til at bevæge sig henholdsvis den ene og den anden vej. Lyselementerne anvendes til at give ekstra feedback på RCX'ens indre tilstand. Lyselementerne har tre tilstande; slukket, blinkende og konstant lys. Disse tre tilstande kan give vigtigt feedback i forbindelse med debugging af RCX'ens kontrolprogram.

A.3 Sensorer

LEGOs RCX har tre inputporte, hvortil der kan sluttes en ekstern sensor. Disse inputporte er nummeret 1, 2 og 3.

RCX'ens sensorer kan være enten aktive eller passive, de to typer behandles forskelligt [Cap02]. Til at hente værdier fra passive sensorer benyttes blot A/D konverteren, mens aktive sensorer kræver en strøm, til at tænde for eksempel en diode [Cap02]. Metoderne *Port1SetActive()* og *Port1SetPassive()* kan benyttes til at skifte mellem aktiv og passiv tilstand for hvert input.



Figur A.1: LEGOs lyssensor [Gas98]

I dette speciale vil håndtering af sensorer foregå som beskrevet i [Cap02]. Denne håndtering vil ikke blive gennemgået i detaljer, i stedet henvises til [Cap02] for en grundigere gennemgang.

A.3.1 Lyssensorer

LEGOs lyssensorer er aktive sensorer. Ved siden af den *phototransistor*, der opfanger lyset, sidder en rød LED, denne kan ses på figur A.1. Lyssensoren er mest følsom over for lys, der ligger i et interval omkring det lys, den røde LED udsender. Da sollys er meget stærkt i netop dette område [Hec98], er lyssensorerne meget følsomme over for netop sollys. Lyssensoren er også følsom i det nært synlige infrarøde område, derfor kan lyssensorerne detektere de infrarøde beskeder, som udsendes af RCX'ens infrarøde transmitter.

Værdierne for en lyssensor med den røde LED tændt ligger typisk i området mellem 300 og 800 i almindelig rumbelysning.

A.3.2 Tryksensorer

LEGOs tryksensorer er passive sensorer. Princippet er en variabel modstand, hvor modstanden ved tryk bliver mindre. I teorien er det således muligt at finde ud af hvor meget, der bliver trykket på en tryksensor. I praksis vil der dog kun blive skelnet mellem 2 tilstande - nemlig trykket og sluppet. Tilstanden trykket vil svare til en værdi på under 100, mens sluppet svarer til en værdi omkring 1000.

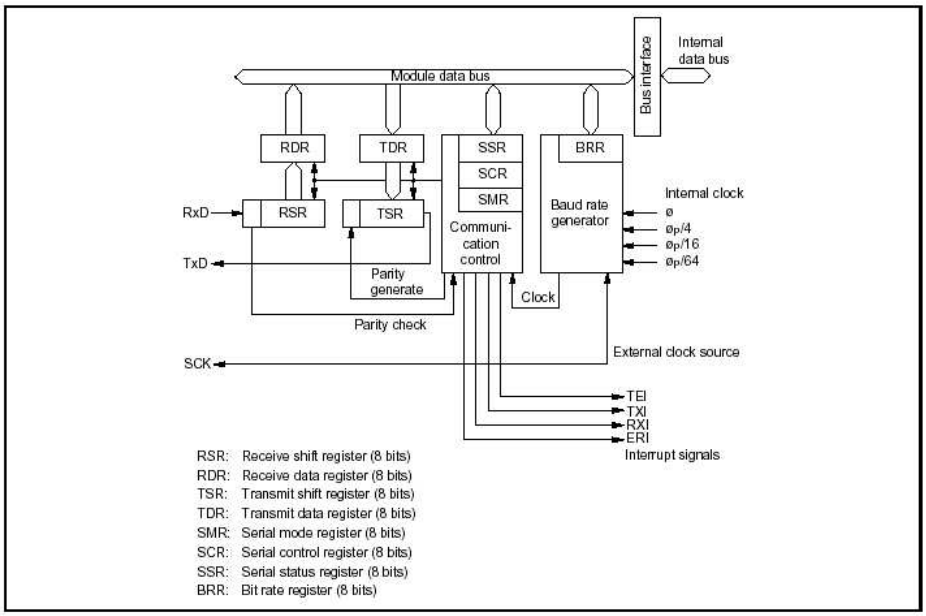
A.3.3 Multiplexing af sensorer

Da tryksensorens tilstand trykket svarer til en værdi på 100, der ligger uden for det interval, som forventes fra lyssensoren, kan en tryk- og en lyssensor

sættes på samme input. Den resulterende værdi vil være værdien for lyssensoren alene, hvis tryksensoren er sluppet, og værdien for tryksensoren alene, hvis tryksensoren er trykket.

Således vil en værdi på under 100 kunne tolkes som et tryk på tryksensoren, mens værdier over 100 kan tolkes som lysværdier. Denne form for multiplexing er brugbar, hvis reaktion på tryksensoren tager præcedens over reaktion på synligt lys. Dette er almindeligt i robotsystemer, hvor *obstacle avoidance* ofte har høj prioritet.

Det skal dog overvejes, at et tryk på tryksensorerne ikke sker øjeblikkeligt. Værdien falder således gradvist til en værdi under 100. Selvom dette fald er hurtigt, kan tryk på en tryksensor, hvis der måles på et uheldigt tidspunkt, fejltolkes som en lysværdi. At sensoren aflæses på vej nedad kan løses ved, at lysværdien nulstilles, når værdien kommer under 100. På denne måde vil en fejlagtig lysværdi forsvinde, så snart værdien er faldet ned på tryksensorniveau. Problemet med at aflæse en trykværdi på vej op kan løses ved, at der ikke aflæses værdier i et vist tidsrum efter sidste tryk på en tryksensor.



Figur B.1: SCI-Curcuit på H8/3297 [Hit98]

B Seriel Kommunikation på H8/3297

LEGOs RCX [Cap02] bruger en Hitachi H8/3297 processor [Hit98]. Denne har mulighed for seriel kommunikation ved hjælp af Serial Communication Interface, SCI. En skematisk tegning af SCI er angivet på figur B.1. SCI er forbundet til H8'ens CPU med en 8-bit databus, en intern clock source og 4 interrupts (TEI, TXI, RXI og ERI). Udover dette er der et ben til en ekstern clock (SCK) og to ben til henholdsvis serielt input (RxD) og output (TxD). På LEGOs RCX er TxD og RxD forbundet til henholdsvis en infrarød transmitter og en infrarød receiver [Vis00].

Til at udføre seriel kommunikation benyttes i alt ni 8-bits registre: Fire data- og fem kontrolregistre. Tre kontrolregistre styrer henholdsvis pakkeformat, interrupts og baudrate; et angiver status, og det sidste bruges stort set ikke. De to af dataregistrene (TDR og RDR) kan læses og skrives direkte fra CPU'en. De to andre (TSR og RSR) bruges til at serialisere og afserialisere data.

I dette afsnit beskrives kun de ting, der gælder på RCX'en. For yderligere detaljer se [Hit98].

Register	Forkortelse	adresse i
Serial Mode Register	SMR	0xffd8
Serial Control Register	SCR	0xffda
Serial Status Register	SSR	0xffdc
Bit Rate Register	BRR	0xffd9
Serial/Timer Control Register	STCR	0xffc3
Transmit Data Register	TDR	0xffdb
Transmit Shift Register	TSR	
Receive Data Register	RDR	0xffdd
Receive Shift Register	RSR	

Tabel B.1: Registre til seriel kommunikation [Hit98].

7	6	5	4	3	2	1	0
C/A	CHR	PE	O/E	STOP	MP	CKS1	CKS0

Figur B.2: Serial Mode Register [Hit98].

B.1 Registre til seriel kommunikation

I dette afsnit beskrives de registre, som anvendes til seriel kommunikation. Kontrolregistrene beskrives i detaljer, mens dataregistrene ikke ville blive beskrevet i detaljer.

B.1.1 Serial Mode Register

Figur B.2 viser indholdet af Serial Mode Register (SMR). Dette register kontrollerer pakkeformat og internal clock. Det er initialiseret med 8 nuller. Betydningen af de enkelte felter er som følger, bitværdi angivet i parentes:

C/A: Angiver byte asynkron(0)/byte synkron(1) kommunikation.

CHR: Angiver bitlængde pr tegn enten 7(1) eller 8(0) bit pr tegn.

PE: Angiver paritet enten ingen paritet(0) eller paritet(1).

O/E: Angiver lige(0) eller ulige(1) paritet.

STOP: Angiver om der skal sendes en (0) eller to (1) stopbits.

MP: Angiver multiprocessor mode.

7	6	5	4	3	2	1	0
TIE	RIE	TE	RE	MPIE	TEIE	CKE1	CKE0

Figur B.3: Serial Control Register [Hit98].

CKS1 og CKS0: Angiver internal clock source til baud rate generator. For detaljer se [Hit98].

Der anvendes asynkron kommunikation, 8 bit data, ulige paritet og en stopbit. Derfor sættes PE og O/E til 1, mens resten ikke skal initialiseres (de er sat til 0). De enkelte bits i dette register skal initialiseres, inden kommunikationen kan begynde. Herefter skal de ikke ændres.

B.1.2 Serial Control Register

Figur B.3 viser indholder af Serial Control Register. Dette register bruges til at enable/disable forskellige funktioner. Betydningen af de enkelte felter er som følger:

TIE: Angiver, om TXI (TDR-empty interrupt) er enable(1) eller disable(0).

RIE: Angiver, om RXI (Receive-end interrupt) og ERI (Recieve-error interupt) er enable(1) eller disable(0).

TE: Angiver Transmit Enable(1) / disable(0). Denne bit skal sættes til 1, før seriel kommunikation kan begynde.

RE: Angiver Recieve Enable(1) / disable(0). Denne bit skal sættes til 1, før seriel kommunikation kan begynde.

MPIE: Multiprocessor Interrupt Enable. Benyttes ikke.

TEIE: Angiver, om TEI (Transmit-end interrupt) er enable(1) eller disable(0)

CKE1: Angiver clock source til intern(0) eller ekstern(1).

CKE0: Angiver, om SCK-pin skal anvendes til clock-output(1) ved asynkron kommunikation.

Da der anvendes intern clock, skal CKE1 sættes til 0. For at starte kommunikation skal RE og TE sættes til 1. For at benytte SCI's interrupts skal TIE, TEIE og RIE også sættes til 1.

7	6	5	4	3	2	1	0
TDRE	RDRF	ORER	FER	PER	TEND	MPB	MPBT

Figur B.4: Serial Status Register [Hit98].

B.1.3 Serial Status Register

Figur B.4 viser indholdet af Serial Status Register. Dette register angiver, om sende/modtage bufferen er fuld/tømt, samt om der er opstået fejl.

TDRE: Angiver, om data i TDR kan overskrives(1). Skal sættes til 0 af CPU, når data er skrevet i TDR.

RDRF: Angiver, om der er modtaget nye data i RDR. Skal sættes til 0 af SCI når data er læst fra RDR.

ORER: Angiver, om der er opstået en overrun fejl.

FER: Angiver, om der er opstået en framing fejl.

PER: Angiver, om der er opstået en paritets fejl.

TEND: Angiver, at der ikke var nye data i TDR, da den sidste bit blev afsendt. Denne bit bliver sat til 0 af SCI, når TDRE sættes til 0.

MPB: MultiProcessor Bit bruges ikke.

MPBT: MultiProcessor Bit Transfer bruges ikke.

Dette register er meget centralt for selve afsendelsen af data. Mens SCR og SMR skal initialiseres, inden kommunikationen begynder, skal dette register aflæses og ændres under afsendelsen af data. Der er specielle regler for, hvordan bits i dette register kan ændres. De seks bits, der er i brug, sættes til 1 af SCI, når de forskellige hændelser indtræffer. Når CPU'en har aflæst en bit til 1, kan den sættes til 0 af CPU'en. TEND kan dog ikke nulstilles direkte. Den bliver nulstillet af SCI, når TDRE nulstilles.

B.1.4 Bit Rate Register

Dette register angiver sammen med CKS1 og CKS0 i SMR, hvilken baud rate data afsendes med. Dette register skal initialiseres for at styre baudrate. På RCX'en skal det initialiseres til: [11001111], hvilket giver en baudrate på 2400 bit/s [Hit98].

B.1.5 Serial/Timer Control Register

Dette register indeholder 5 reserverede bits, der ikke kan skrives til, samt felterne MPE, ICKS1 og ICKS0. Disse felter skal ikke ændres for at bruge RCX'ens serielle kommunikation. For en nærmere beskrivelse se [Hit98]

B.2 Afsendelse af data

Efter initialisering af kontrolregistrene kan den egentlige afsendelse af data begynde. CPU'en arbejder en 8-bits bus til dataregistret TDR (Transmit Data Register), men data skal afsendes serielt. Derfor er der indføjet et ekstra dataregister, TSR (Transmit Shift Register). TSR kan hverken læses eller skrives af CPU'en. SCI gør følgende:

1. Til at begynde med undersøges TDRE feltet i SSR. Hvis TDRE er 0, er der nye data i TDR, som overføres til TSR

2. Efter at have overført data til TSR, sættes TDRE til 1. Er interrupts aktiveret, afsendes et TXI interrupt.

3. Herefter sendes data serielt over TxD pinnen efter pakkeformattet, der er angivet i SMR.

4. Når pakken er afsendt, undersøges TDRE igen. Hvis TDRE er 0, er der nye data i TDR, som skal afsendes, og den næste pakke afsendes. Hvis TDRE er 1, er der ikke nye data i TDR, og TEND sættes til 1. Er interrupts aktiveret, afsendes et TEI interrupt.

CPU'ens opgave i forbindelse med afsendelse af data er således at overvåge TDRE og TEND samt skrive data til TDR. TDRE og TEND kan overvåges enten ved busy-wait eller ved en interrupthandler til TXI interupt. Opgaven er i begge tilfælde at skrive den næste byte af data i TDR, når TDRE bliver sat til 1. Interrupthandleren er at foretrække, da afsendelse af en bit tager 416 μs. Med 8 bit data, 1 start, 1 paritet og 1 stopbit tager afsendelsen af 1 byte $4,6$ ms. svarende til 73600 clock cykler ved 16 mhz. En kodestump, der afsender en byte ved hjælp af busy-wait, kan ses i figur B.5.

B.3 Modtagelse af data

Modtagelse af data fungerer på mange måder som et spejlbillede af afsendelsen. Den primære forskel er håndteringen af fejl. Ved afsendelse af data opstår der ikke fejl. Men under modtagelsen kan der opstå flere fejl. De tre

```
#include "Buttons.h"
#define TDRE (1<<7)
#define SCI_TDR *((volatile byte *) 0xffdb) /*TDR*/
#define SCI_SSR *((volatile byte *) 0xffdc) /*SSR*/
static byte aborted;

byte TDRempty(){
  return (SCI_SSR & TDRE);
}

void writeTDR(byte i) {
  SCI_TDR=i;
}

void clearTDRE() {
  SCI_SSR &= (~TDRE);
}

void busysend(byte i) {
  while(!TDRempty()){
    if(!Running) {
      aborted=1;
      return;
    }
  }
  writeTDR(i);
  clearTDRE();
}
```

Figur B.5: Busy-Wait afsendelse af data

fejl, som bliver håndteret af SCI, er: overrun error(ORER), parity error(PER) og framing error (FER). Modtagelsen af data foregår altså som følger:

1. SCI overvåger RxD-pinnen og synkroniserer, når en startbit opdages.
2. Data modtages til RSR en bit af gangen
3. Paritets - og stopbit modtages og kontroleres. Det undersøges også, om RDRF-feltet i SSR er sat til 0. Er RDRF = 1, så er der opstået en overrun error.
4. (a) Hvis der er opstået fejl, sætter SCI det rette felt i SSR til 1 og afsender en ERI-interrupt. Ved overrun error skrives de nye data ikke i RDR. Det gør de derimod ved parity errror og framing error. Modtagelse af data kan ikke fortsætte, før ORER, PER og FER felterne igen er sat til 0 fra CPU'en.
   (b) Hvis der ikke er opstået nogen fejl, flyttes data fra RSR til RDR, RDRF sættes til 1 og SCI afsender en RXI-interrupt.

Modtagelsen kan ligesom afsendelsen foregå både som busy-wait og afbrydelsesstyret. CPU'ens opgave i forbindelse med modtagelse af data er at læse RDR, når RDRF bliver sat til 1. Opstår der en fejl, er det også CPU'ens opgave at reagere, når ORER, PER eller FER bliver sat. CPU'en skal således beslutte, hvordan fejlen skal håndteres og sætte ORER, PER og FER i SSR til 0. En kodestump, der modtager en byte ved hjælp af busy-wait, kan ses i figur B.6.

B.4 RCX'ens transceiver

SCI som beskrevet udgør et interface, der kan benyttes til at give kontrolprogrammet til RCX'en mulighed for at afsende og modtage infrarøde beskeder. Som nævnt er dette interface forbundet til en infrarød transmitter og en infrarød receiver. Disse infrarøde komponenter udgør begrænsninger i brugen af SCI'en. Det er vigtigt, at man holder sig disse begrænsninger for øje, når man skriver programmer, der benytter den infrarøde kommunikation på RCXen.

B.4.1 Infrarød receiver

Den infrarøde receiver er beskrevet i [Vis00]. Den består af et analogt kredsløb, der detekterer infrarødstråling i et relativt smalt frekvensinterval. Det, som receiveren detekterer, er blink af infrarødt lys i dette frekvensinterval. Disse

```
#define SCI_RDR *((volatile byte *) 0xffdd) /*RDR*/
#define SCI_SSR *((volatile byte *) 0xffdc) /*SSR*/
#define RDRF (1<<6)
#define ORER (1<<5)
#define FER (1<<4)
#define PER (1<<3)
#define ERROR (SCI_SSR & (ORER | FER | PER))
static byte receiveerror;
static byte aborted;

byte readRDR() {
  return SCI_RDR;}

byte RDRfull() {
  return (SCI_SSR & RDRF);}

void clearRDRF() {
  SCI_SSR &= (~RDRF);}

void clearERRORs() {
  SCI_SSR &= ~(FER | PER | ORER);}

byte busyreceive() {
  byte i;
  while(!RDRfull() && !ERROR) {
    if(!Running) {
      aborted=1;
      return 0;
    }
  }
  if(ERROR) {
    receiveerror=1;
    clearERRORs();
  }
  i=readRDR();
  clearRDRF();
  return i;
}
```

Figur B.6: Busy-Wait modtagelse af data

infrarøde blink bliver afsendt med 38,5 khz. Receiveren kan detektere signaler, der består af minimum 6 cykler. Efter 6 signalcykler skal der være en pause på mindst 10 cykler før næste signal. Dette giver følgende begrænsning på den maximale bitrate, som signaler kan modtages med:

$T_{cycle} = \frac{1}{38{,}5khz} = \frac{1}{38500}s = 2.6 \times 10^{-5}s$

$T_{bit} = 16 \times T_{cycle} = 415\mu s$

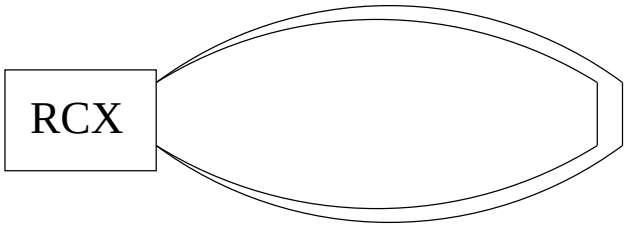
$Bitrate = {T_{bit}}^{-1} = 2406\frac{bit}{s}$

Den maximale mulige bitrate er altså omkring 2400 bit/s. Dermed kan det ikke forventes, at RCX'ens kommunikation er stabil ved højere hastigheder. Der kan dog kommunikeres ved højere hastigheder, men der vil med tiden blive ophobet støj i det analoge kredsløb, og kommunikationen vil blive ustabil.

B.4.2 Infrarød transmitter

Den inrarøde transmitter er koblet til TxD på SCI samt en timer fra H8 CPU'en. Timeren fra CPU'en leverer et signal med frekvens på 38,5 khz. Signalet fra TxD negeres og amplitude moduleres [Sta00] med signalet fra timeren som bærebølge. Signalet bliver på denne måde kodet således, at 1 er en periode uden infrarødt signal, mens 0 er en periode med infrarød signal med puls på 38,5 khz.

Den infrarøde transmitter kan operere i to tilstande - longrange og shortrange. Forskellen mellem de to tilstande er den effekt, som signalet sendes med. Ved longrange benyttes højere effekt. Dermed opnås større rækkevidde end ved shortrange. For en mere præcis beskrivelse af udbredelsen af de infrarøde signaler se Appendix C.



Figur C.1: RCX'ens fremadrettede kommunikationskegle

C Rækkevidden for RCX'ens infrarøde kommunikation

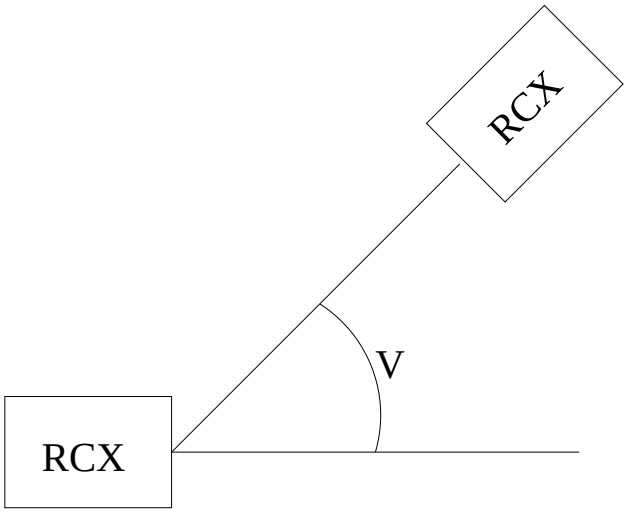
RCX'ens kommunikation bliver anvendt meget i dette speciale. Mange af de valg, der er foretaget, er en direkte følge af kommunikationens fysiske udbredelse. Derfor er det vigtigt at undersøge, hvor lang rækkevidde RCX'ens infrarøde kommunikation har.

En vigtig faktor, når man måler kommunikationsradius, er lys. [GVSM02b] viser, at skarpt lys kan reducere antallet af pakker, der bliver modtaget fra omkring 95% til omkring 50%. Den store overfølsomhed over for lys skyldes, at RCX'en benytter lys i det nært synlige infrarøde område til at sende signaler. I det nært synlige infrarøde område er der i forvejen meget lys i vores omgivelser, for eksempel indeholder sollys infrarødt lys i dette område [Hec98]. Denne faktor vil ikke blive undersøgt i dette eksperiment, dels fordi resultatet allerede er kendt og dels, fordi det ofte er muligt at holde lysmængden konstant i indendørs omgivelser.

Dette eksperiment vil derimod udforske de horizontale karakteristika ved udbredelsen af RCX'ens infrarøde signaler, under forudsætning af at lyssætning, reflektion og skygger fra omgivelserne holdes konstant.

Fra kvalitative målinger vides det, at RCX'en kommunikerer stabilt inden for et fremadrettet kegleformet område, figur C.1. Dette område er omgivet af en tynd kant, hvor kommunikationen bliver ustabil. Det er denne kant jeg ønsker at bestemme.

Kommunikationsområdet er retningsafhængigt, derfor måles kommunikationen i forskellige vinkler. I hver af disse vinkler forsøges det at finde grænsen, hvor kommunikationen bliver ustabil og grænsen, hvor kommunikationen forsvinder helt.



Figur C.2: Eksperimentel opstilling

0x55	0xFF	0x00	0x00	i	0xFF	~i

Figur C.3: Format af pakkerne

C.1 Ekseprimentel opstilling

På figur C.2 ses den eksperimentelle opstilling til udmåling af den fremadrettede kommunikationskegle. For at gøre dette foretages der målinger for forskellige værdier af V og med forskellig afstande mellem afsender og modtager.

Værdierne for V er valgt til $0°, 10°, 20°, 35°, 45°, 65°$ og $90°$. For hver af disse værdier findes den afstand, hvor kommunikationen bliver ustabil. Der vælges nogle punkter omkring denne afstand, og der tages 5 målinger i hver af disse punkter.

Hver måling består i at lade afsenderen afsende 100 pakker, der har formatet angivet i figur C.3. Modtageren undersøger, om pakkerne overholder dette format ved at undersøge om de to sidste bytes er hinandens bitvise negationer. Hvis dette er tilfældet, tælles pakken som modtaget.

Disse målinger bliver kun foretaget i det horizontale plan. Dette er valgt som en begrænsning, da robotterne i dette speciale befinder sig på en bane, hvor deres transceivere altid befinder sig i det samme horizontale plan.

Afstand / cm	Målinger				
	1	2	3	4	5
70	100	100	97	100	100
72	91	94	93	98	98
75	48	47	57	63	67
77	25	19	26	35	21
80	2	3	3	1	1

Tabel C.1: V=0

Afstand / cm	Målinger				
	1	2	3	4	5
65	100	97	97	94	91
68	67	51	34	52	35
70	39	17	28	14	6
72	12	3	2	2	10

Tabel C.2: V=10

C.2 Resultat

Målinger for vinklerne $0°, 10°, 20°, 35°, 45°, 65°$ og $90°$ er angivet i tabellerne C.1 til C.7. Dette er med en antagelse om symmetri plottet i figur C.4. Signalet ser ud som forventet - nemlig som en kegle omgivet af en smal kant.

Der kan opleves store udsving i størrelsen af kommunikationsområdet afhængigt af forhold. Det primære forhold, der spiller ind, er lys. Sollys giver således store mængder støj på signalet, det samme gør visse typer elektrisk lys. Andre faktorer, der gør sig gældende, er skyggeforhold, reflektion samt robotternes batteriniveau. Det er altså ikke nødvendigvis muligt at sige præcist, hvordan RCX'ens kommunikation udbreder sig. Formen af udbredelsen bliver påvirket af reflektion og skygge, mens rækkevidden bliver kraftigt påvirket af lys og batteriniveauet.

Afstand / cm	Målinger				
	1	2	3	4	5
50	99	98	85	83	87
52	56	41	44	33	48
55	0	0	1	0	2

Tabel C.3: V=20

Afstand / cm	Målinger				
	1	2	3	4	5
25	100	100	100	100	100
28	21	55	52	69	41
30	0	2	1	0	0

Tabel C.4: V=30

Afstand / cm	Målinger				
	1	2	3	4	5
18	97	97	95	99	97
20	5	0	0	2	0

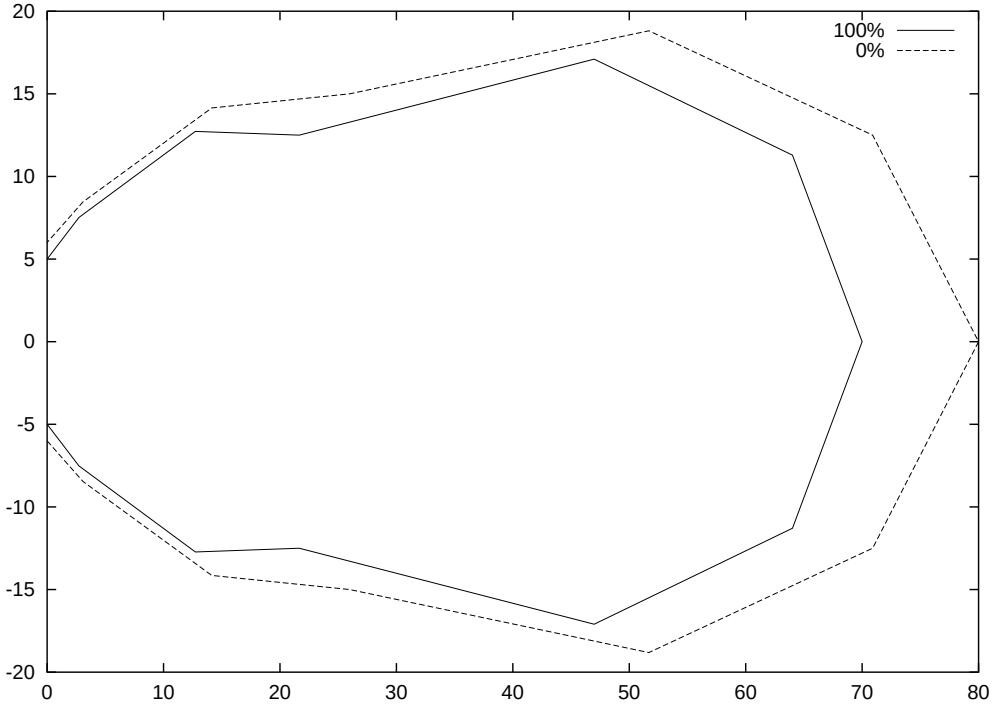
Tabel C.5: V=45

Afstand / cm	Målinger				
	1	2	3	4	5
8	100	100	100	99	100
9	0	4	15	0	1

Tabel C.6: V=70

Afstand / cm	Målinger				
	1	2	3	4	5
5	100	100	99	100	100
6	24	56	7	80	37

Tabel C.7: V=90



Figur C.4: Signalets vandrette udbredelse. Afsenderen er placeret i (0,0)

D Forsøgsresultater

D.1 Standstill forsøg

Dette er resultaterne fra afsnit 6.3. I vinklerne 90° og 270° har det stor betydning, om søgeren drejer den korte eller den lange vej rundt. Det er derfor blevet noteret i hvert forsøg om søgeren drejede den korte eller den lange vej rundt.

T_{seek} med brug af lyssensorer

V	1	2	3	4	5	6	7	8	9	10
0	0.128	0.180	0.110	0.146	0.122	0.085	0.110	0.124	0.134	0.274
90	0.968	0.576	0.612	0.618	0.828	0.526	0.568	0.524	0.566	0.562
180	1.760	2.266	1.730	1.802	1.686	1.708	1.756	1.654	1.828	2.024
270	0.504	0.806	0.560	0.578	0.612	0.700	0.776	0.428	0.532	0.464

T_{seek} uden brug af lyssensorer

V	1	2	3	4	5	6	7	8	9	10
0	0.146	0.112	0.110	0.080	0.146	0.118	0.172	0.108	0.136	0.138
90	2.544	0.588	2.670	2.580	2.686	0.524	2.754	2.678	0.462	2.732
180	1.640	1.598	1.694	1.616	1.676	1.642	1.700	1.622	1.598	1.710
270	0.458	0.424	2.772	2.728	2.786	2.838	0.420	2.756	2.726	0.552

T_{total} med brug af lyssensorer

V	1	2	3	4	5	6	7	8	9	10
0	0.256	0.308	0.240	0.276	0.556	0.213	0.238	0.252	0.262	0.402
90	2.428	2.036	2.072	2.078	2.288	1.986	2.028	1.984	2.026	2.022
180	3.220	3.726	3.192	3.262	3.146	3.168	3.216	2.986	4.628	3.482
270	1.964	2.516	2.020	2.038	1.944	2.160	2.230	2.144	1.994	2.308

T_{total} uden brug af lyssensorer

V	1	2	3	4	5	6	7	8	9	10
0	0.272	0.238	0.206	0.200	0.272	0.244	0.298	0.234	0.262	0.264
90	4.286	1.982	4.060	3.974	4.080	1.920	4.148	4.074	1.856	4.126
180	3.034	2.866	3.088	3.010	3.070	3.036	3.094	3.018	2.994	3.104
270	2.150	2.116	4.166	4.122	4.180	4.232	1.814	4.150	4.122	1.946

E Filer fra implementationen

Dette appendix indeholder en liste af de filer, som indgår i implementationen i forbindelse med eksperimenterne i dette speciale. Alle filerne findes på den vedlagte CD-ROM

E.1 Kode til standstill forsøg

Koden til dette forsøg ligger i biblioteket *Standstill*, der indeholder følgende filer:

IR.h: Der indeholder interruptshandlers til SCI.

MotorState.h: Der indeholder en struct, der repræsenterer output til motorer fra de forskellige behavior producing modules.

Seek.h: Der implementere søgeadfærden med brug af lyssensorer.

Seekblind.h: Der implementerer søgeadfærden uden brug af lyssensorer.

pinger.c: Der implementerer fyrtårnet.

seeker.c: Der er hovedfilen til søgeren, der anvender lyssensorer.

seekerblind.c: Der er hovedfilen til søgeren, der ikke anvender lyssensorer.

E.2 Kode til wander forsøg

Koden til dette forsøg ligger i biblioteket *Wander*, der indeholder følgende filer:

IR.h: Samme som ovenfor.

MotorState.h: Samme som ovenfor.

Semaphore.h: Der indeholder en enkelt semafor.

sensorinput.h: Der håndterer multiplexing af tryk- og lyssensorer.

Avoid.h: Der implementerer obstacle avoidance.

Wander.h: Der implementerer en Wander-adfærd.

Communication.h: Der implementerer oprettelse og nedlæggelse af forbindelser samt acknowledgements.

net.h: Der implementerer buffering af pakker på netværksniveau.

com2.c: Der indeholder hovedprogrammet til robotten med adresse 2.

E.3 Kode til måling af rækkevidden for RCX'ens infrarøde kommunikation

Koden til dette forsøg ligger i biblioteket *Range*, der indeholder følgende filer:

IR.h: Samme som ovenfor.

busysender.c: Der implementerer afsenderen af pakkerne.

busyreceiver.c: Der implementerer modtagelsen af pakker.

F Litteratur

Litteratur

[Ark98] Ronald C. Arkin. *Behavior Based Robotics*. Massachusetts Institute of Technology, 1st edition edition, 1998.

[BBI+98] Rodney A. Brooks, Cynthia Breazeal, Robert Irie, Charles C. Kemp, Matthew Marjanovic, Brian Scassellati, and Matthew M. Williamson. Alternative essences of intelligence. In *AAAI/IAAI*, pages 961–968, 1998.

[BMJ+98] Josh Broch, David A. Maltz, David B. Johnson, Yih-Chun Hu, and Jorjeta Jetcheva. A performance comparison of multi-hop wireless ad hoc network routing protocols. In *Mobile Computing and Networking*, pages 85–97, 1998.

[BP02] Tucker Balch and Lynne E. Parker, editors. *Robot Teams, From diversity to polymorphism*. A K Peters, Natick Massachusetts, 2002.

[Bro86] Rodney A. Brooks. A robust layered control system. *IEEE Journal of robotics and automation*, vol. RA-2 NO 1 march, 1986.

[Bro90] Rodney A. Brooks. Elephants don't play chess. *Robotics and Autonomous Systems*, 6(1&2):3–15, June 1990.

[Bro91a] Rodney A. Brooks. Intelligence without reason. *Computers and Thought*, 1991.

[Bro91b] Rodney A. Brooks. New approaches to robotics. *Science*, September 1991.

[BÅN89] Sven Bruel and Niels Åge Nielsen. *Gyldendals Fremmedordbog*. Gyldendalske Boghandel, Nordisk Forlag A.S., Copenhagen, 10 edition, 1989.

[Cap02] Ole Caprani. *RCX Manual*. University of Aarhus, Department of Computer Science, Ny Munkegade, Bldg. 540, 8000 Aarhus C, Denmark, 2002.

[CDK01] George Coulouris, Jean Dollimore, and Tim Kindberg. *Distributed Systems, Concepts and Design*. Addison-Wesley, third edition edition, 2001.

[CFJ+02] Ole Caprani, Jakob Fredslund, Jens Jacobsen, Line Kramhøft, Rasmus B. Lunding, Jørgen Møller Ilsøe, and Mads Wahlberg. *Evolution of Computer Bugs - an Interdisciplinary Team Work*, chapter 10. Springer Verlag, 2002.

[DJM02] Gregory Dudek, Michael Jenkin, and Evangelos Milios. A taxonomy of multirobot systems. In *Robot Teams*, chapter 1. A K Peters, Natick Massachusetts, 2002.

[FBKT02] Dieter Fox, Wolfram Burgard, Hannes Kruppa, and Sebastian Thrun. Collaborative multirobot localization. In *Robot Teams*, chapter 6. A K Peters, Natick Massachusetts, 2002.

[Fer99] Jack Ferber. *Multi-Agent Systems -An Introduction to Distributed Artificial Intelligence*. Addision Wesley Longman, 1999.

[GAGPK03] T. Goff, N. B. Abu-Ghazaleh, D. S. Phatak, and R. Kahvecioglu. Preemptive maintenance routing in ad hoc networks. *Journal of Parallel and Distributed Computing*, 63(2):123–140, February 2003. Special Issue on Wireless Mobile Communication and Computing.

[Gas98] Gasperi. Mindstorms rcx sensor input page. http://www.plazaearth.com/usr/gasperi/lego.htm, 1998. Hentet 27/1 2004.

[GM02] Brian P. Gerkey and Maja J Mataric. Sold!: Auction methods for multi-robot coordination. *IEEE Transactions on Robotics and Automation*, special issue on Advances in Multi-Robot Systems, 18(2):758–786, oct 2002.

[GVSM02a] A. E. Gonzalez-Velazquez, L.E. Sacks, and I. W. Marshall. Ad hoc sensor network experimentation using the rcx by mindstorms. In *EUNICE*, 2002.

[GVSM02b] A. E. Gonzalez-Velazquez, L.E. Sacks, and I. W. Marshall. Simple spontaneous mechanism for flexible data communication in wireless ad hoc sensor networks. In *London Communications Symposium*, 2002.

[Hec98] Eugene Hecht. *Optics*. Addison-Wesley, third edition edition, 1998.

[HEK+04] Klaus Marius Hansen, Toke Eskildsen, Lars Kristensen, Kenneth-Daniel Nielsen, Rolf E. Thorup, Jack Fridthjof, Ulrik Merrild, and Jørn Eskildsen. The ex hoc infrastructure - enhancing traffic safety through life warning systems. Technical report, Computer Science Department, University of Aarhus, DK-8200 Aarhus N, liwas@isis.alexandra.dk, 2004.

[Hit98] Hitachi. *Hitachi Single-Chip Microcomputer H8/3297 Series Hardware Manual*. Hitachi, 1998.

[HMR91] David W. Hogg, Fred Martin, and Mitchel Resnick. Braitenberg creatures. Technical Report TR, MIT Media Labs, 1991, 10 pages, Massachusetts Institute of Technology, 1991.

[HMS01] A. Howard, M. Matadd, and G. Sukhatme. An incremental self-deployment algorithm for mobile sensor networks, 2001.

[JMB01] D.B. Johnson, D.A. Maltz, and J. Broch. Dsr: The dynamic source routing protocol for multi-hop wireless ad hoc networks. In *Ad Hoc Networking*, chapter 5, pages 139–172. Addison-Wesley, 2001.

[Joh95] David B. Johnson. Scalable support for transparent mobile host internetworking. In *Mobile Computing*, chapter 3. Kluwer Academic Publishing, 1995.

[JZ02] David Jung and Alexander Zelinsky. Symbol grounding for communication. In *Robot Teams*, chapter 8. A K Peters, Natick Massachusetts, 2002.

[Kar90] Phil Karn. Maca - a new channel access method for packet radio. *ARRL/CRRL Amateur Radio 9th Computer Conference*, september 1990.

[Kri00] Thiemo Krink. Motivation networks - a biological model for autonomous agent control. *Journal of Theoretical Biology*, 2000.

[LK02a] Maxim Likachev and Sven Koenig. Improved fast replanning for robot navigation in unknown terrain. Technical report, Georgia Tech, Mobile Robot Lab, 2002.

[LK02b] Maxim Likachev and Sven Koenig. Incremental replanning for mapping. Technical report, Georgia Tech, Mobile Robot Lab, 2002.

[Mae89] Patti Maes. How to do the right thing. *Connection Science Journal*, 1(3):291–323, 1989.

[Mae90] P. Maes. Situated agents can have goals. *Journal for Robotics and Autonomous Systems*, 6(1), 1990.

[Mae94] Pattie Maes. Modeling adaptive autonomous agents. *Artificial Life Journal*, 1 & 2(1-2), April 1994.

[Mat92] Maja J Mataric. Integration of representation into goal-driven behavior-based robots. *IEEE Transactions on Robotics and Automation*, 8(3):304–312, June 1992.

[Mat97] Maja J Mataric. Behavior-based control: Examples from navigation, learning, and group behavior. *Journal of Experimental and Theoretical Artificial Intelligence, special issue on Software Architectures for Physical Agents*, 9(2-3):323–336, 1997.

[Met99] Riku Mettala. Bluetooth protocol architecture. White Paper 1.C.120/1.0, Nokia Mobile Phones, 1999.

[Mic92] Z. Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs*. Springer, Berlin, 1992.

[NAS03] NASA. Mars exploration rover mission. http://mars.jpl.nasa.gov/mer/mission/spacecraft_surface_rover.html, 2003. Hentet 24/9 2003.

[Nil98] Nils J. Nilsson. *Artificial Intelligence, A New Synthesis*. Morgan Kaufmann, 1998.

[O'H02] Keith J. O'Hara. Active routing for mobile robot teams. Technical report, College of Computing, Georgia Institute of Technology, november 2002.

[PAI02] Srinath Perur, P. Abhilash, and Sridhar Iyer. Router handoff: A preemptive route repair strategy for aodv. In *IEEE Intl. Conference on Personal Wireless Computing, New Delhi, December 2002.*, 2002.

[Par98] Lynne E. Parker. Alliance: An architecture for fault tolerant multi-robot cooperation. *IEEE Transactions on Robotics and Automation*, Vol. 14(2), 1998.

[PB94] C. E. Perkins and P. Bhagwat. Highly dynamic destination-sequenced distance-vector routing (dsdv) for mobile computers. *ACM SIGCOMM'94 Conference on communications Architectures, Protovols and Applications, IEEE*, pages 90–100, 1994.

[Per98] Charles E. Perkins. Mobile networking in the internet. *Mobile Networks and Applications*, 3:319–334, 1998.

[Pfe96] Rolf Pfeifer. Teaching powerfull ideas with autonomous mobile robots. *Journal of Computer Science*, vol 7, No. 2, 1996.

[Rey87] Craig W. Reynolds. Flocks, herds, and schools: A distributed behavioral model. *Computer Graphics*, 21(4):25–34, juli 1987.

[RK03] Miklós Aurél Rónai and Eszter Kail. A simple neighbour discovery procedure for bluetooth ad hoc networks. In *proceedings of the GlobeCom 2003 conference, San Francisco, CA, USA, 1-5 December 2003*, 2003.

[Son] Sony. Sony global - aibo global link. www.sony.net/Products/aibo/index.html. Hentet 11/5 2004.

[Sta00] Willam Stalling. *Data & Computer Communications*. Prentice-Hall, sixth edition edition, 2000.

[Stø01] Kasper Støy. Using situated communication in distributed autonomous mobile robots. In *proceedings of the 7th Scandinavian Conference on Artificial Intelligence(SCAI-7), pages 44-52, Odense, Denmark, Feb 19-21, 2001.*, 2001.

[SV02] Peter Stone and Manuela Veloso. A survey of multiagent and multirobot systems. In *Robot Teams*, chapter 3. A K Peters, Natick Massachusetts, 2002.

[Sys] Cisco Systems. *Routing Basics*, chapter 5. Cisco Systems, ??

[Tec02] LEGO Product Technology. *LEGO Spybotics ROM documentation*, 2002.

[Teh97] LEGO Product Tehcnology. *Robotic Invention System 2.0*, 1997.

[Thr02] Sebastian Thrun. Probabilistic robotics. *Communication of the ACM*, 45(3), march 2002.

[VFC99] Richard Vaughan, Andy Frost, and Stephen Cameron. Robot sheepdog project archieves automatic flock control. http://www-robotics.usc.edu/ṽaughan/projects/RSP/, 1999.

[Vis00] Vishay. *Photo Modules for PCM Remote Control Systems*. Vishay, document 82006, rev. 6 edition, Sep 2000.

[Wer00] Barry Brian Werger. Ayllu: Distributed port-arbitrated behavior-based control. In George Bekey Lynne E. Parker and Jacob Barhen, editors, *Distributed Autonomous Robotic Systems 4*, pages 25–34. Springer, 2000.

[Win00] Alan Winfield. Distributed sensing and data collection via broken ad hoc wireless connected networks of mobile robots. *Distributed Autonomous Robotic Systems 4*, pages 273–282, 2000.

[WM01] Barry Brian Werger and Maja J Mataric. From insect to internet: Situated control for networked robot teams. *Annals of Mathematics and Artificial Intelligence*, Vol 31:1-4:173–198, 2001.

[Woo02] Michael Wooldridge. *An Introduction to MultiAgent Systems*. John Wiley & Sons ltd., 2002.

[ZSDT02] Robert Michael Zlot, Anthony (Tony) Stentz, M Bernardine Dias, and Scott Thayer. Market-driven multi-robot exploration. Technical Report CMU-RI-TR-02-02, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, January 2002.

[ØMS01] Esben H. Østergaard, Maja J. Matatric, and Gaurav S. Sukhatme. Distributed multirobot task allocation for emergency handling. *Proc. of the IEEE/RSJ intl conf. in Intelligent Robot and Systems (IROS)*, 2001.