Kooperativ kartkonstruktion för autonoma robotar

Daniel Påhlstorp
dat98dph@ludat.lth.se

Dannie Ronnovius
dat98dro@ludat.lth.se

Mars 2003

Kooperativ kartkonstruktion för autonoma robotar

Sammanfattning

Denna rapport behandlar problemet global kartkonstruktion för autonoma mobila robotar i en RoboCup-miljö. Den miljö robotarna förutsätts arbeta i har känd utsträckning och kända landmärken. Objekt i denna miljö är rörliga och består av flera olika men kända typer. Det är dessa objekt som skall kartläggas. Robotarna använder sig av seende för identifiering av objekt och landmärken. Då robotarnas syn är begränsad, föreligger osäkerhet rörande igenkännande av objekt samt självlokalisering. Kartläggning är därför ett svårt problem. Denna rapport undersöker huruvida det med hjälp av informationsutbyte är möjligt att ge en mer fullständig och precis kartläggning. I rapporten redovisas utveckling, implementation och resultat av tre olika metoder med ökande beräkningsmässig komplexitet, som alla avser att lösa kartläggningsproblemet.

Cooperative map construction in autonomous robots

Abstract

This report investigates problems of the global map construction for autonomous mobile robots in a RoboCup environment. The environment in which the robot works, and the landmarks therein, are both known. Objects in this environment are movable and belong to one of several known types. Construction of the map is based on these objects. The robots use vision to recognize objects and landmarks. Since the robot´s vision is limited there exist uncertainties about object recognition and also self localization, and therefor the map construction is a hard problem. This report investigates the possibility of increasing the accuracy in map construction by use of information exchange between robots. In the report we will present the development and implementation of three different methods of increasing computational complexity. These methods all aim at solving the map construction problem.

Förord

Det har varit en lång väg från start till det färdiga resultatet, men det har varit ett otroligt intressant och givande arbete. Ett arbete som inte skulle ha varit möjligt utan hjälpen från flera personer. Vi vill rikta ett alldeles speciellt tack till vår handledare Jacek Malec som med stor kompetens väglett och hjälpt oss under arbetets gång. Vi vill dessutom tacka Alessandro Saffiotti och de i Team Sweden engagerade i Örebro, för den oumbärliga hjälp vi fått av dem. Slutligen vill vi tacka Katarina Bendtz för hennes hjälp med korrekturläsning, stöd och engagemang.

Innehåll

1 Introduktion **11**

1.1 Bakgrund 11

1.1.1 RoboCup 11

1.1.2 Sony Legged Robot League 12

1.1.3 Sonys AIBO 13

1.1.4 Team Sweden 13

1.2 Problemformulering 14

1.3 Syfte med rapporten 15

2 Team Sweden-arkitekturen **17**

2.1 Överblick 17

2.1.1 Rörelsekontroll 17

2.1.2 Perception 18

2.1.3 Självlokalisering 19

2.1.4 Beteenden och strategi 19

2.2 Kartrepresentation 20

2.2.1 Lokal karta 20

2.2.2 Global karta 20

2.3 Oskarp självlokalisering 22

2.4 Oskarp logik 23

3 Problemanalys **27**

3.1 Definition av problem 27

3.1.1 Datarepresentation 27

3.1.2 Utbyte av kartinformation 28

3.1.3 Association av objekt 28

3.1.4 Sammanslagning av kartinformation 29

3.2 Relevanta arbeten 30

3.3 Val av kartdelningsmetoder 30

4 RoboSimulatorn **31**
4.1 Vad är RoboSimulatorn? . . . 31
4.2 Hur fungerar RoboSimulatorn? . . . 31
4.3 RoboSimulatorns uppbyggnad . . . 32
4.3.1 Server . . . 32
4.3.2 Klient . . . 34

5 Metoder för kartdelning **37**
5.1 Inledning . . . 37
5.2 Metod 1: Viktat medelvärde . . . 37
5.2.1 Översikt . . . 37
5.2.2 Representation och konstruktion av data . . . 38
5.2.3 Sammanvägning av kartinformation . . . 40
5.3 Metod 2: Oskarpa positionslådor . . . 41
5.3.1 Översikt . . . 41
5.3.2 Representation och konstruktion av data . . . 42
5.3.3 Sammanvägning av kartinformation . . . 44
5.3.4 Extrahering av position . . . 45
5.4 Metod 3: Oskarpa positionsrutnät . . . 46
5.4.1 Översikt . . . 46
5.4.2 Representation och konstruktion av data . . . 46
5.4.3 Sammanvägning av kartinformation . . . 48
5.4.4 Extrahering av position . . . 49
5.5 Association av objekt . . . 50
5.5.1 Algoritm . . . 51
5.5.2 Värderingsfunktioner . . . 53
5.6 Distribution av kartinformation . . . 53

6 Resultat **55**
6.1 Tester av kartdelningsmetoder . . . 55
6.2 Körtider för metoder . . . 56
6.3 Självlokalisering . . . 58

7 Diskussion **61**
7.1 Inledning . . . 61
7.2 Förväntat resultat . . . 61
7.3 Analys av resultat . . . 62
7.3.1 RoboSimulatorn . . . 62
7.3.2 RoboCup 2002 . . . 65
7.4 Möjliga förbättringar . . . 65
7.4.1 Minne över tidigare positioner . . . 66

7.4.2 Förbättrad objektassociation 66
7.4.3 Förbättrad självlokalisering 67
7.5 Vidare utvärdering i verklig miljö 67

8 Slutsats **69**

Litteraturförteckning **71**

A Terminologi **73**

B RoboSimulatorn **75**
B.1 Manual för RoboSimulatorn 75

C Tabeller och diagram **79**

Kapitel 1

Introduktion

1.1 Bakgrund

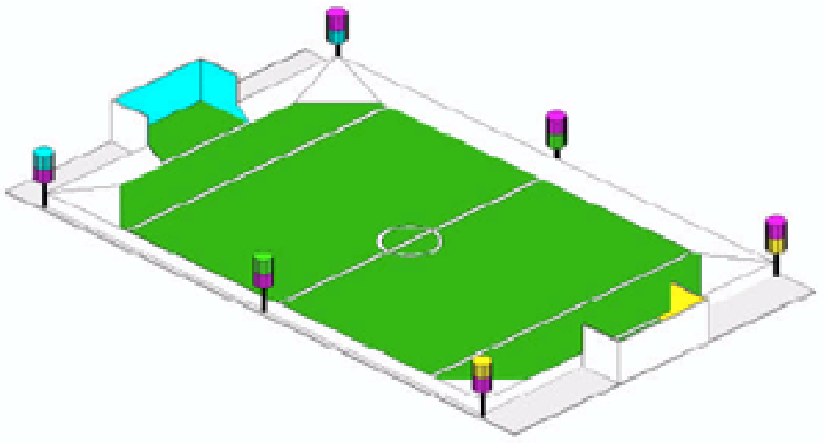
Denna rapport beskriver utveckling av mjukvara för Sonys hundrobot AIBO [4] i samarbete med Team Sweden [2].

1.1.1 RoboCup

RoboCup [1] (ursprungligen kallad Robot World Cup Initiative) är en internationell organisation för forskning och undervisning. Det är ett försök till att stödja AI (artificiell intelligens) och robotforskning genom att erbjuda ett standardproblem där olika teknologier kan integreras och utvärderas. I detta syfte valde RoboCup fotboll för robotar som sin primära domän. RoboCup är en utmaning i form av en tävling för lag av robotar, som kan förflytta sig snabbt i en dynamisk omgivning. Flertalet teknologier måste inkorporeras för att ett robotlag skall kunna prestera bra, inkluderande robotik, design av autonoma agenter, multi-agent-samarbete och realtidsresonemang.

RoboCup-tävlingen är uppdelad i flera olika divisioner. Dessa ställer olika krav på de deltagande lagen i form av hårdvara samt utformning av robotar. Divisionerna är:

- Simulation League (Simuleringsdivision)
- Small Size Robot League (Division för små robotar)
- Middle Size Robot League (Division för medelstora robotar)
- Sony Legged Robot League (Division för Sonys fyrbenta hundrobot AIBO)



Figur 1.1: Bild över spelplanen för SLRL.

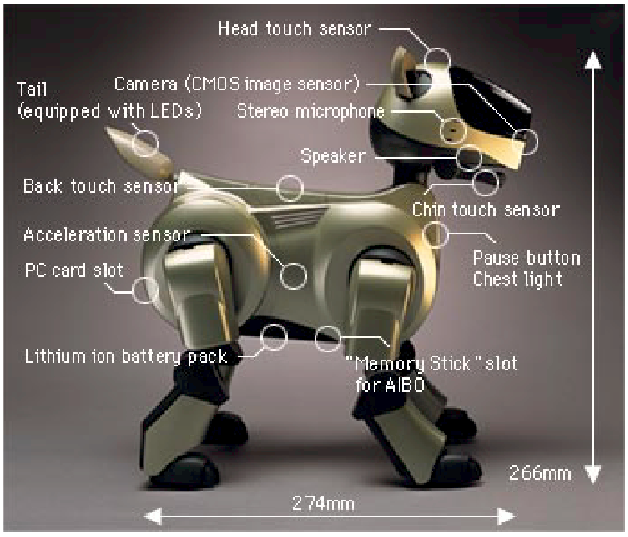
- Humanoid League (Division för humanoida robotar)

RobCup-organisationen har föreslagit att deras ultimata mål skall lyda som följer:

> **År 2150 skall ett lag av fullt autonoma humanoida robotar vinna i en fotbollsmatch mot de för tillfället regerande världsmästarna i fotboll enligt de gällande officiella FIFA reglerna.**

1.1.2 Sony Legged Robot League

Sony Legged Robot League, eller kort SLRL, är en division av RoboCup där alla lagen använder sig av samma fysiska plattform. Plattformen är en fyrbent hundliknande robot som heter Aibo. I SLRL tävlar lag om fyra stycken hundrobotar mot varandra på en ca 4×5 meter stor spelplan (se figur 1.1). Runt spelplanen är sex stycken landmärken i olika färger placerade. Med hjälp av dessa, samt de två målen som också är olikfärgade, kan hundarna lokalisera sig själva på spelplanen. För att robotarna skall kunna skilja på hundar från samma lag respektive andra laget, är de två lagen märkta med olika färger (blå/röd). I SLRL tillåts radiokommunikation för distribution av information mellan hundarna. Bandbredden för kommunikation är dock begränsad, år 2002 var gränsen satt till 0.5Mb.



Figur 1.2: AIBO hunden.

1.1.3 Sonys AIBO

Den hårdvaruplattform som används i RoboCups SLRL är Sonys hundliknande robot AIBO ERS-210, se figur 1.2. AIBO är hårdvarumässigt sett en komplex plattform. AIBO har 3 frihetsgrader för vardera ben och totalt 20 frihetsgrader för alla leder. Den har färgkamera, mikrofon och radiokommunikation, för att nämna några av dess attribut. För att utnyttja denna hårdvaruplattform har Sony även tagit fram en utvecklingsmiljö vid namn OPEN-R [5]. En av plattformens större brister är kameran som har dålig skärpa, vilket medför att dess synfält är kraftigt begränsat. Ett annat problem är att roboten är fyrbent vilket medför att odometrisk[1] information är väldigt opålitlig (eftersom roboten glider på underlaget).

1.1.4 Team Sweden

Team Sweden är resultatet av Sveriges ansträngningar att producera ett lag av fotbollsspelande fysiska robotar. Målsättningen är att Team Sweden skall deltaga i SLRL. Team Sweden är ett samarbete mellan flera universitet i Sverige. År 2002 bestod laget av Örebro Universitet (Koordinator), Lunds Universitet och Blekinge Tekniska högskola. Laget har deltagit i SLRL sedan 1999. Team Sweden har som mål att skapa lösningar som är generellt applicerbara på varje system som uppvisar följande karakteristika:

[1]Information om hur robotens position förändras baserat på robotens rörelser.

- Behandling av stora osäkerheter som är genomgående i sin domän.
- Integrering av högre nivåers kognitiva processer med lägre nivåers sensor och motorprocesser.
- Integrering av handlingar och perception i en komplex och dynamisk omgivning.

Huvudmålet är att utveckla generella principer och algoritmer för autonom robotstyrning i oförutsägbara omgivningar, som kan återanvändas i olika robotar och domäner.

1.2 Problemformulering

I många situationer som involverar robotar och AI uppstår problem gällande klassificering och kartläggning av dynamiska objekt. I SLRL förekommer många rörliga objekt såsom boll, medspelare och motspelare. I en sådan omgivning föreligger problemet att vid varje tidpunkt bestämma respektive objekts position. Eftersom en väl utförd kartläggning dessutom är beroende av att självlokalisering fungerar väl, är kartläggningen inte ett trivialt problem. Genom att låta robotar kommunicera och utbyta information sinsemellan, erbjuds en möjlighet att ge en mer täckande och precis kartläggning av robotarnas omgivning. Problemet består då i att avgöra hur detta informationsutbyte skall gå till. Problemet kan förenklas genom att det delas upp i följande delproblem:

- **Datarepresentation**
  Vilken information skall utbytas och hur kan denna representeras?
- **Utbyte av kartinformation**
  Hur kan kartinformation utbytas? (Exempelvis hur och när information skall skickas etc.)
- **Association av objekt**
  Hur kan en mängd av objekt på bästa sätt, väljas att associeras med en annan mängd av objekt? (Då det finns flera objekt av samma typ föreligger problemet för varje robot att matcha sin information om observerade objekt med den motsvarande informationen från andra robotar)
- **Sammanslagning av kartinformation**
  Hur kan information från flera robotar kombineras, dvs. hur skall en

robot behandla den information den får från andra robotar? (Dessutom, hur kan en positionsuppskattning extraheras från den sammanvägda informationen?)

Ytterligare ett komplikation av kartläggningsproblemet är att det måste göras i realtid, samt att endast begränsat med information kan skickas robotarna emellan. Det finns endast begränsat med beräkningskraft i roboten, och bara en liten del av den kan användas till att behandla kartinformation. Detta på grund av att robotens huvudsakliga uppgift är att spela fotboll. Att göra en kartläggning är bara en, om dock viktig, deluppgift i fotbollsspelandet. Det är önskvärt att en cykel av utbyte samt behandling av kartinformation endast skall ta en bråkdel av en sekund. Det är också önskvärt att precisionen i kartläggning har en felmarginal som i medelfallet inte är större än några tiotal centimeter (för den givna storleken på planen, dvs. 4×5m). Problemet är således att finna en metod som ger ett säkert resultat, men som samtidigt är så billig[2] som möjligt beträffande beräkningar och informationsutbyte.

1.3 Syfte med rapporten

Denna rapport avser att beskriva utvecklingen och resultatet av tre olika metoder som alla har som mål att lösa det ovan beskrivna kartläggningsproblemet i en RoboCup SLRL-miljö. De tre metoderna är av ökande beräkningsmässig komplexitet och kräver dessutom olika mycket bandbredd för överföring av information. Genom att jämföra de olika metoderna syftar rapporten till att:

1. Uppskatta skillnad i säkerhet för kartläggning med respektive utan informationsutbyte.

2. Uppskatta skillnad i säkerhet för kartläggning med metoder av olika tidskomplexitet.

3. Finna en metod som är beräkningsmässigt billig men som också uppfyller kraven på god säkerhet gällande kartläggning.

Huvudmålet med arbetet är att redovisa en metod som faktiskt uppfyller den sistnämnda punkten ovan och som är praktiskt applicerbar i RoboCup-domänen.

[2]Termer som billig och dyr används i datasammanhang för att beskriva hur kostsam en algoritm är ur ett antingen beräknings- eller lagringsmässigt perspektiv.

Kapitel 2

Team Sweden-arkitekturen

De metoder som redovisas i denna rapport är baserade på den arkitektur och de tekniker som Team Sweden använder sig av [14]. För att ge en ökad klarhet i kommande resonemang, redovisas de kortfattat nedan.

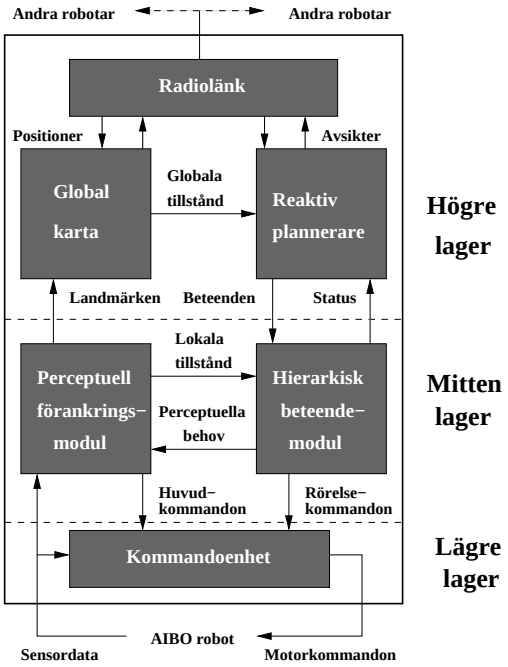
2.1 Överblick

Team Sweden använder sig av en kognitiv och lagrad arkitektur som har inspirerats av Thinking Cap [3]. Detta är en arkitektur för autonoma robotar som använder sig av oskarp logik för kunskapsrepresentation. Team Sweden-arkitekturen är uppdelad i tre lager, se figur 2.1. Den lägsta nivån erbjuder ett abstrakt interface till de fysiska funktionerna på roboten. Mittenlagret är ansvarigt för att upprätthålla en lokal representation av omgivningen runt roboten, samt för att implementera robusta taktiska beteenden. Den högsta nivån håller en global karta över spelplanen och fattar på grundval av denna strategiska beslut i realtid. Oskarp logik används i alla nivåer för att hantera osäkerheter.

Det är främst informationen från den globala kartenheten (i viss grad även den perceptuella förankringsmodulen) som kommer att behandlas i denna rapport. Denna modul kommer därför att beskrivas mer ingående.

2.1.1 Rörelsekontroll

Det är kommandoenheten som ansvarar för robotens rörelser. Kommandoenheten tar emot rörelsekommandon från HBM, Hierarchical Behavior Module, och översätter dessa till lämpliga gångstilar. Kommandoenheten kontrollerar dessutom rörelse för huvud samt, ansvarar för olika typer av sparkar.



Figur 2.1: Team-Sweden arkitekturen

Det finns en stor mängd gångstilar implementerade, som kombinerar framåt-, sido- och roterande rörelser. Dessutom finns en alternativ parametrisk gång [9] som har utvecklats av New South Wales Universitet.

2.1.2 Perception

Enheten som ansvarar för perception kallas PAM, Perceptual Anchoring Module. PAM fungerar som ett korttidsminne över positioner för objekten[1] runt roboten. Vid varje tillfälle innehåller PAM de bästa tillgängliga positionsuppskattningarna för objekten. Positionsuppskattningen uppdateras regelbundet genom en kombination av tre mekanismer

- *Perceptuell förankring*(varje gång ett objekt detekteras av kameran)
- *Odometri* (varje gång roboten rör på sig)
- *Global information* (varje gång roboten gör en ny självlokalisering)

[1]När det talas om objekt i denna rapport åsyftas om inget annat nämnts landmärken, medspelare, motspelare och boll.

Förmågan att kunna identifiera objekt korrekt är en väldigt viktig egenskap för robotarna. En felaktig klassificering av ett landmärke leder ofta till en felaktig positionsuppskattning i självlokaliseringen. Klassificering av objekt i PAM bygger på tre tekniker:

- *Färgsegmentering* genom en snabb algoritm som använder sig av utdata från färgdetektionen i hårdvaran [10]
- *Modellbaserad regionssammanslagning* för att indentifiera utmärkande former i färgklickar
- *Kunskapsbaserat filter* för att eliminera felaktiga klassificeringar.

PAM styr också hur roboten skall titta. Den gör detta genom att styra kameran efter de för tillfället största perceptuella behoven. Dessa erhålles från HBM [11].

2.1.3 Självlokalisering

Självlokalisering i SLRL är ett svårt problem. Anledningen till det är bl.a. som tidigare nämnts att odometrisk information är väldigt osäker och att kameran har ett begränsat synfält. För att kunna hantera dessa osäkerheter har Team Sweden utvecklat en ny självlokaliseringsteknik som använder sig av oskarp logik [6]. Denna teknik kombinerar idéer dels från den oskarpa landmärksbaserade lokaliseringen som beskrivits av Saffioti och Wesley [7], men också från den teori om positionsmöjlighetsrutnät som lagts fram av Burgard [8]. Fördelarna med denna teknik är att den endast använder sig av kvalitativa rörelse- och sensormodeller. Den är dessutom beräkningsmässigt billig samt kan återhämta sig från godtyckligt stora fel. Denna teknik har visat sig vara väldigt effektiv i SLRL-sammanhang.

Resultatet från självlokaliseringen används av RP, Reactive Planner, för att fatta strategiska beslut.

2.1.4 Beteenden och strategi

Modulen HBM håller en mängd beteenden för navigering samt kontroll av bollen. Dessa beteenden är konstruerade med hjälp av oskarp logik och är organiserade i en hierarkisk struktur. Genom att kombinera enkla beteenden kan mer komplexa beteenden skapas. Beteenden inkorporerar dessutom perceptuella regler som används för att delge PAM vilka perceptuella behov

som för tillfället finns.

Spelstrategier genereras dynamiskt av RP. Denna modul använder sig av ett beteendeurvalsschema som är baserat på tekniken med artificiella elektriska fält (EFA)[12]. Metoden använder sig av positiva och negativa elektriska laddningar som appliceras på målen samt på robotarna. För en given spelplanssituation kan man sedan uppskatta ett heuristiskt värde genom att undersöka den elektriska potentialen för någon av positionerna på spelplanen (exempelvis bollens position). Detta värde används sedan för att välja det beteende som antas ge bäst resultat i denna situation.

2.2 Kartrepresentation

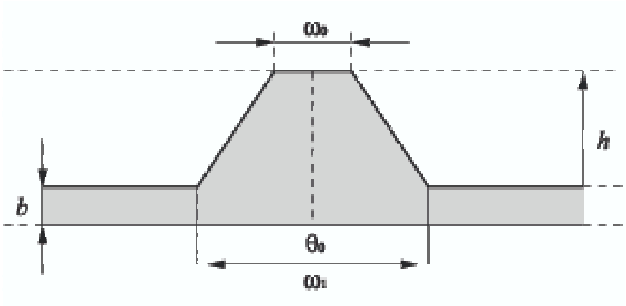
I Team Sweden-arkitekturen använder sig robotarna av två typer av kartor för att representera sin omgivning. Den ena är en lokal karta som tillhandahålls av PAM. Denna karta håller information om objekt i robotrelativa koordinater. Den andra är den globala kartan som konstrueras utifrån informationen från den lokala kartan. Här är positioner för objekt representerade i ett globalt kartesiskt koordinatsystem. I fortsättningen kommer kortformer som *Gk* och *Lk* användas för att benämna den globala respektive lokala kartan.

2.2.1 Lokal karta

Den lokala kartan uppdateras av den perceptuella enheten PAM med jämna intervall (nuvarande 5Hz). Objekt i Lk är beskrivna i robotrelativa polära koordinater. Positioner för objekt är här representerade som ett avstånd $\rho \in \mathbb{R}$, och en orientering $\phi \in [-\pi, \pi]$ som är relativa robotens position och orientering. Lk håller dessutom för varje objekt ett förankringsvärde (anchoring) $a \in [0, 1]$ som talar om hur färsk informationen är. Ett värde $a = 1.0$ innebär att informationen är helt färsk och ett värde $a = 0.0$ innebär att informationen är väldigt gammal. Informationen om ett objekt i Lk är representerat som en trippel (ρ, ϕ, a) där ρ, ϕ, a är variablerna beskrivna ovan. Lk håller för varje objekt information av den här formen.

2.2.2 Global karta

Den globala kartan Gk håller en rutnätsbaserad karta av positioner där varje cell c representerar en specifik position (x, y, θ) för roboten i globala plankoordinater. Värdet i varje cell c representerar hur möjligt det är på en skala



Figur 2.2: Oskarp mängd $(\theta_0, w_0, w_1, h, b)$

från 0.0 till 1.0, att roboten befinner sig på positionen som representeras av c. Om roboten är osäker på var den befinner sig, kan många celler vara möjliga, det vill säga, de kan alla ha höga värden. Den idealiska situationen är att endast en cell är möjlig och att alla andra har ett värde på 0.0.

Den ovan beskrivna rutnätskartan kan med fördel representeras som en funktion:

$$u_r : M \rightarrow [0, 1] \tag{2.1}$$

där mängden M är en tredimensionell *array* av celler, och varje cell representerar en position precis som ovan. Att representera en $3D$ array kräver dock mycket lagringsutrymme och är kostsam att behandla. En bättre lösning är att förenkla representationen till en $2D$ array P. Varje cell $p \in P$ motsvarar en position (x, y). För att dessutom inräkna riktningen i varje position så håller p inte bara ett värde utan istället en funktion

$$u_p : [-\pi, \pi] \rightarrow [0, 1] \tag{2.2}$$

För varje vinkel $\theta \in [-\pi, \pi]$ anger $u_p(\theta)$ hur möjligt det är att roboten är orienterad med vinkeln θ förutsatt att den befinner sig på koordinaten p. Vi har då att $u_r(c) = u_p(\theta)$ där $c = (x, y, \theta)$ och $p = (x, y)$, detta förutsatt att inga restriktioner satts på u_p. Låt nu u_p vara en parametrisk representation som beskriver en trapezoid med bias[2] (se figur 2.2) en så kallad oskarp mängd (eng. fuzzy set). Då behövs endast 5 värden för att representera $u_p : (\theta_0, w_0, w_1, h, b)$. Jämför med den tidigare nämnda representationen där det behövs ett rutnät med 360 celler, en per grad. För ett helt rutnät är i

[2]Snedvridning, används här som ett värde på hur tillförlitlig informationen är.

praktiken biasen samma för varje cell och representationen kan därför minskas från 5 till 4 värden. Observera att ett högt värde på biasen (nära 1.0) innebär att informationen inte är tillförlitlig.

Den ovan beskrivna typen av rutnätskarta kallar vi i fortsättningen för ett oskarpt positionsrutnät (OPR), eller bara ett oskarpt rutnät. Observera dessutom att möjligheten för en robot att befinna sig på en viss position $p = (x, y)$, oberoende av riktningen, är den maximala möjligheten att den befinner sig på p för någon riktning.

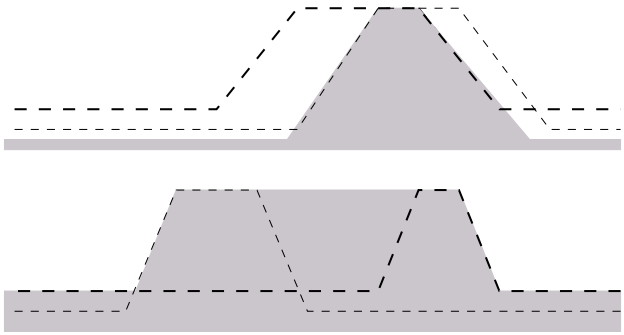
$$u_r(p) = \|u_p(\theta)\|_{[-\pi,\pi]} \tag{2.3}$$

Här utläses $\|f\|_E$ som suprenumnormen av f på E, dvs. $\|f\|_E = \sup_{x \in E} |f(x)|$, där f är en funktion definierad på E. Om det är uppenbart från sammanhanget vilken mängd E som avses skriver vi bara $\|f\|$. Den maximala möjligheten ges av höjden h från trapezoiden på position p.

Genom att kombinera informationen från den lokala kartan, med informationen angående robotens position från den ovan beskrivna rutnätskartan, OPR, är det möjligt att konstruera ett globalt kartesiskt system över objekt i robotens omgivning.

2.3 Oskarp självlokalisering

Robotens säkerhet på sin position kan beskrivas som en möjlighetsdistribution över mängden av alla möjliga positioner. Denna representeras av det virtuella 3D-rutnätet, OPR, som beskrivits ovan. Underhåll av positionsinformation följer cykeln *förutse-observera-uppdatera*: vid observation av ett landmärke på en känd position konverteras denna information till en möjlighetsdistribution över rutnätet. För varje cell i rutnätet mäter denna möjligheten att roboten befinner sig på den cellen, givet observationen. Denna möjlighetsdistribution används sedan för att uppdatera den föregående positionsuppskattningen genom ett så kallat oskarpt snitt. Proceduren upprepas för varje tillgänglig observation. Observera att tidigare distribution, kan vara homogen, som vid fallet då roboten först blir placerad på planen. I steget *förutse*, i cykeln, används en enkel modell där rörelse information vägs in i för att återkasta möjligheten att roboten förflyttat sig. Detta åstadkoms genom att helt enkelt låta distributionen bli lite *suddigare* i alla positioner. Att göra en distribution suddigare, innebär att möjligheten för varje cell i rutnätet ökas lite. Denna suddning görs i alla riktningar, och graden är



Figur 2.3: Oskarpt snitt med vinkelinformation.

bestämd av den maximala hastigheten som roboten kan ha. Robotens riktning och faktiska hastighet tas inte hänsyn till i beräkningen. Det innebär att modellen är helt fri från odometri. Observera att möjligheten att roboten fortfarande befinner sig på samma position inte minskar.

Vid varje tillfälle ges den mest troliga positionen för roboten av centrum, till den region i OPR som har högst möjlighet. Denna position beräknas efter varje cykel med en form av tyngdpunktsberäkning.

2.4 Oskarp logik

Betrakta fallet då vi vill tyda observationen av ett känt landmärke vid tiden t. Låt G_t vara ett oskarpt positionsrutnät vid tiden t så att $G_t(x, y)$ beskriver en oskarp mängd (figur 2.2) över positionen (x, y). Låt $\vec{r}$ vara vektorn mellan robot och det observerade landmärket. (Denna innehåller information rörande både längd och riktning). Beteckna med S en möjlighetsdistribution så att $S_t(x, y \mid \vec{r})$ beskriver en oskarp mängd över möjligheten att roboten befinner sig på positionen (x, y) givet observationen $\vec{r}$. För varje observation uppdateras G_t som

$$G'_t(x, y) = G_t(x, y) \times S_t(x, y \mid \vec{r})$$

där $\times$ avser en oskarp operator för ett snitt mellan två oskarpa mängder. Beräkning av ett oskarpt snitt C för två oskarpa mängder A och B delas upp på två fall:

1. A och B överlappar varandra.

2. A och B överlappar inte varandra.

Om A och B överlappar varandra så väljs C som den inre trapezoid som skapas av intersektionen(se övre figur 2.3). Värdena för denna är lätta att beräkna utifrån parametrarna för A och B. Speciellt så beräknas den nya biasen: $b_C = b_A \cdot b_B$. Om A och B inte överlappar varandra dvs. när de två mängderna innehåller motsägande information, så är den ovan beskrivna beräkningen inte fullgod, eftersom information går förlorad. Istället låter vi C vara den yttre trapezoid som omsluter både A och B (se undre figur 2.3). Den uppskattade positionen kommer då att minska i precision och säkerhet, vilket är en naturlig följd av den motsägande informationen.

En enklare form av oskarpt snitt kan beräknas om man bortser från robotens riktning, och således endast betraktar dess position. Denna form av oskarpt snitt kommer att användas senare i rapporten. Låt då istället G_t vara ett oskarpt 2D-rutnät vid tiden t, där $G_t(x, y)$ är ett värde $p \in [0, 1]$ för cellen på position (x, y) i rutnätet som mäter hur möjligt det är att roboten befinner sig på den motsvarande positionen i omgivningen. För att tyda observationen av ett känt landmärke vid tiden t skapar vi en möjlighetsdistribution $S_t(\cdot \mid r)$ där $S_t(x, y \mid r) \in [0, 1]$ anger möjligheten att roboten befinner sig på positionen (x, y) givet att avståndet till det observerade landmärket är r. På liknande vis som tidigare uppdateras G_t för varje observation som:

$$G'_t(x, y) = G_t(x, y) \times S_t(x, y \mid r)$$

Även här avser $\times$ en oskarp operator. Denna dock väljas enklare än tidigare, ty operanden är nu en funktion $f : [0, 1] \times [0, 1] \to [0, 1]$. Notera skillnaden: tidigare var operanden en funktion av två oskarpa mängder, nu är det en funktion av två reella tal i intervallet 0.0 till 1.0. Eftersom observationerna är oberoende kan operatorn $\times$ väljas som någon T-norm. En T-norm är en funktion:

$$T : [0, 1] \times [0, 1] \to [0, 1] \tag{2.4}$$

med egenskaperna:

$$T(0, 0) = 0, T(a, 1) = T(1, a) = a \tag{2.5}$$

$$a \leq c \text{ och } b \leq d \Rightarrow T(a, b) \leq T(c, d) \tag{2.6}$$

$$T(a, b) = T(b, a) \tag{2.7}$$

$$T(a, T(b, c)) = T(T(a, b), c) \tag{2.8}$$

Exempel på T-normer är min ($T_{min}(x, y) = \min(x, y)$), algebraisk produkt ($T_{ap}(x, y) = x \cdot y$) och Łukasiewicz norm ($T_L(x, y) = \max(x + y - 1, 0)$).

Kapitel 3

Problemanalys

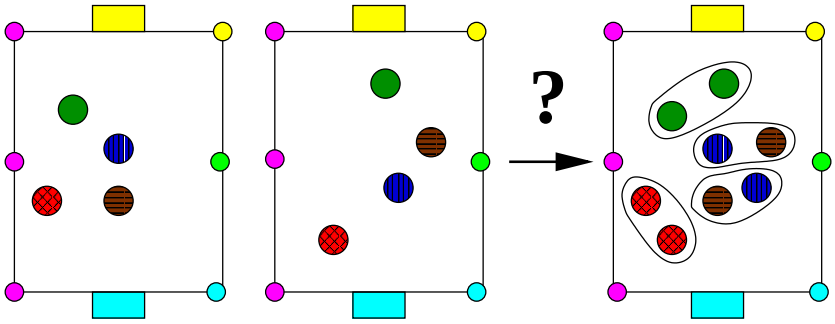
3.1 Definition av problem

I inledningen valde vi att dela upp problemet i fyra viktiga delproblem: **Datarepresentation**, **Utbyte av kartinformation**, **Association av objekt** och **Sammanvägning av kartinformation**. Genom att göra denna uppdelning är det möjlig att förenkla problemet till att lösa mindre delproblem. Det bör nämnas att detta bara är ett sätt att dela upp problemet och att det kan finnas andra bättre tillvägagångssätt. Delproblemen beskrivs närmare nedan.

3.1.1 Datarepresentation

Vilken kartinformation skall utbytas mellan robotarna och hur skall denna information representeras? Det finns många möjligheter att lösa detta problem på. (Vilken representation man än väljer måste man självklart också ha tillvägagångssättet för de övriga punkterna i åtanke). Att exempelvis endast skicka information om ett objekt som en positionsuppskattning (x, y) vore en enkel lösning. Nackdelen är att denna representation säger väldigt lite om objekten; Den talar inte alls om med vilken säkerhet objektet kan antas befinna sig på den givna positionen. Det vore således önskvärt att även väga in osäkerhetsfaktorer i den information som skall utbytas.

Problemet består i att hitta en representation som är så informativ som möjligt. Dessutom bör den vara billig att konstruera samt billig i lagringsutrymme eftersom information skall skickas mellan robotar.



Figur 3.1: Associering av informationen från två robotar. Objekt på bilden har ihopparats så att det totala avståndet mellan de ihopparade objekten minimerats.

3.1.2 Utbyte av kartinformation

Hur skall kartinformationen utbytas? Denna fråga är av mer övergripande och organisatorisk karaktär. Problem som kan tänkas behöva behandlas är:

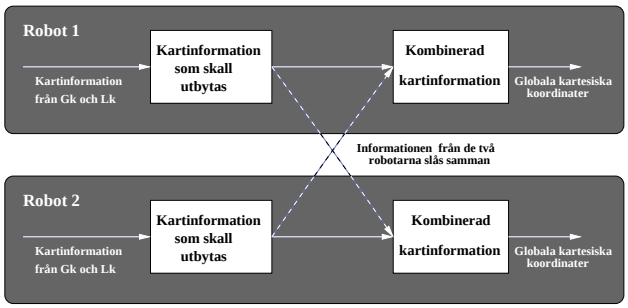
- **Hur skall information utbytas?**
  Bör robotarna utbyta information i en viss ordning? Bör informationen från särskilda robotar prioriteras? Bör en robots egen information särbehandlas när information kombineras? Bör information behandlas i den ordning den kommer in, eller bör den istället buffras för senare gemensam sammanvägning? etc.

- **När skall information utbytas?**
  Bör robotar skicka information endast då de har ny information från PAM? Bör robotar skicka information endast då de gjort en ny självlokalisering? Bör robotarna själva fråga om information när de anser sig vara i behov den? Bör robotar undvika att skicka information om den inte är tillförlitlig? etc.

I korthet, behandlar detta problem frågor angående hur ett förlopp av utbyte av information mellan robotarna skall ske. Problemet består i att hitta ett effektivt schema för utbyte av information.

3.1.3 Association av objekt

Då det finns mer än ett objekt av en typ, t.ex. medspelare och motspelare, är det ett problem att veta hur objekten från olika robotar skall matchas (observera att detta problem även finns då ingen information utbytes dvs.



Figur 3.2: Bilden visar kombinering av kartinformation för två robotar.

varje gång roboten får ny information från kameran). Betrakta följande exempel för fallet två robotar: robotarna R_1 och R_2 observerar samma fyra objekt. Pga. osäkerhet från kamera och lokalisering får de olika positionsuppskattningar för dessa objekt. R_2 skickar nu sin information till R_1 som skall väga samman denna med sin egen information. Hur skall R_1 på bästa sätt para ihop sina objekt med de erhållna från R_2, se figur 3.1. Det finns två uppenbara lösningar till detta problem. Den första är att försöka hitta den bästa parvisa matchningen av objekten från R_1 med objekten från R_2 och på så vis para ihop dem. Detta kan göras på många sätt, exempelvis genom att: för vart och ett av objekten para det med det närmsta icke matchade objektet (girig algoritm), para objekt så att det totala avståndet mellan de parade objekten minimeras, etc. Den andra lösningen är att helt enkelt ignorera problemet, genom att betrakta de fyra objekten som *en* enhet. Det innebär att vi inte betraktar de fyra objekten var för sig, utan istället väger samman två mängder av objekt. Det förutsätter dock att man har en representation som möjliggör detta. För denna lösning uppstår dock det senare svåra problemet att från den sammanslagna mängden extrahera de fyra troligaste positionerna för objekten.

Problemet är här att finna en lösning som matchar objekt på ett effektivt sätt och som dessutom är billig.

3.1.4 Sammanslagning av kartinformation

Det sista delproblemet, efter att representation, distribution och association av kartinformation har behandlats, är sammanvägning av kartinformation. Förhoppningen är att en robot, med hjälp av kartinformation från sig själv och andra robotar skall kunna göra en säkrare positionsuppskattning för ett objekt. För att möjliggöra det, krävs en metod som på ett *adekvat* sätt väger

samman informationen från de flera robotarna (se figur 3.2): den skall på ett effektivt sätt utnyttja den befintliga informationen om objekten. Det är också viktigt att denna metod inte är allt för kostsam. Problemet består i att finna en sådan metod.

3.2 Relevanta arbeten

Detta arbete är unikt på så sätt att den givna problemdomänen är relativt outforskad. Det finns många arbeten som behandlar olika typer av problem gällande kartläggning för robotar [15, 16, 17]. De flesta av dessa skiljer sig dock ganska markant från problemet beskrivet i denna rapport. Exempelvis så behandlar flertalet av dessa arbeten enbart kartläggning av statiska objekt, som t.ex. väggarna i ett rum och/eller andra föremål [15, 16, 17]. Vi däremot avser att kartlägga dynamiska objekt, objekt som kan förflytta sig. Ofta använder sig robotarna av någon typ av närhetssensorer, som t.ex. IR-sensorer, för utforskning av sin omgivning [15, 17], emedan våra robotar använder sig av synen från en kamera. De robotar som vi jobbar med skiljer sig också på så vis att det är fyrbenta, till skillnad från robotar beskrivna i många andra arbeten. Dessa är ofta hjulförsedda [15, 16, 17]. Denna skillnad påverkar den odometriska informationen och med stor sannolikhet också valet av kartrepresentation. Slutligen behandlar merparten av arbeten kartläggning för enbart en robot, utan någon form av samarbete [15, 16].

Vid den tidpunkt vi påbörjade detta arbete, fanns det till vår kännedom ingen litteratur som behandlar problem av den exakta karaktären som vi avser att lösa.

3.3 Val av kartdelningsmetoder

Med hjälp av uppdelningen i mindre delproblem är det dock möjligt att använda resultat även från närliggande problemområden. Kombination av kartinformation har stora likheter med landmärksbaserad självlokalisering för en robot. Även här handlar det om att förbättra en positionsuppskattning genom att kombinera information från flera, från varandra skilda, observationer. För den givna kartrepresentationen (OPR) är det också naturligt att dra en parallell till det material som finns om oskarp logik. Mot bakgrund av detta har vi valt att använda delar från dessa båda områden. Framförallt har våra idéer inspirerats av den oskarpa självlokaliseringsmetoden som föreslagits av Buschka et al. [6]. Metoderna redovisas i kapitel 5.

Kapitel 4

RoboSimulatorn

4.1 Vad är RoboSimulatorn?

RoboSimulatorn är ett verktyg vi utvecklat för att kunna testa idéer rörande informationsdelning även utanför robotarna. Simulatorn erbjuder en utvecklingsmiljö, där idéer enkelt kan implementeras och utvärderas. Det är en server/klientbaserad applikation som är försedd med ett grafiskt gränssnitt, samt metoder för statistikföring. Simulatorn tillhandahåller en miljö som är nästan identisk med den riktiga SLRL-miljön. Det innebär att proportioner på spelplan och objekt är konstruerade för att ge en så realistisk bild som möjligt av den verkliga miljön. Likaså eftersträvas att perception, rörelse etc. för simulerade robotar skall överensstämma proportionellt med egenskaperna för de riktiga Aibo-hundarna. Storleken på klienternas synfält och säkerheten i positionsuppskattningar gällande observerade objekt, är justerbara och tillåter därför att påverkan av olika synförhållanden kan utvärderas.

RoboSimulatorn skapades för att påskynda och underlätta utvärdering av informationsdelningsmetoder till Aibo-hundarna. Programmeringen av hundarna är ett långsamt och problematiskt arbete. Anledningen till detta är bland annat att redskap för felsökning är svåra och tidskrävande att använda. En annan orsak är svårigheten att utvärdera resultatet av programvaran direkt på hundarna. Möjligheten att kunna utveckla och utvärdera metoder redan innan de provas i hundarna sparar mycket arbete.

4.2 Hur fungerar RoboSimulatorn?

RoboSimulatorn är uppdelad i en serverdel samt en klientdel där kommunikation mellan delarna sker via ett nätverk. Mer exakt så sker kommu-

nikationen via TCP och kan äga rum dels mellan server och klient, dels klienter emellan. Servern simulerar den del av roboten som hanterar perception. Utöver detta upprätthåller och styr den fysikaliska egenskaper i världen (rörelse, friktion, kollision av objekt etc.), samt simulerar rörelser för ett av lagen (motståndarlaget). Servern håller en global karta över alla objekts faktiska position i världen. En typisk cykel av utbyte mellan server och klienter ser ut som följer:

1. Vid varje tidpunkt t beräknar och sänder servern den lokala kartan ${Lk_t}^i$ till varje klient K_i rörande objekt som klienten ser.

2. Klienten behandlar den information som servern sänt och gör utifrån denna en självlokalisering enligt den tidigare beskrivna metoden.

3. Klienten utbyter någon form av kartinformation med de andra klienterna.

4. Klienten behandlar information rörande objekten. Baserat på den kartinformation klienten har, skickar den därefter information angående sin nästa förflyttning, till servern.

4.3 RoboSimulatorns uppbyggnad

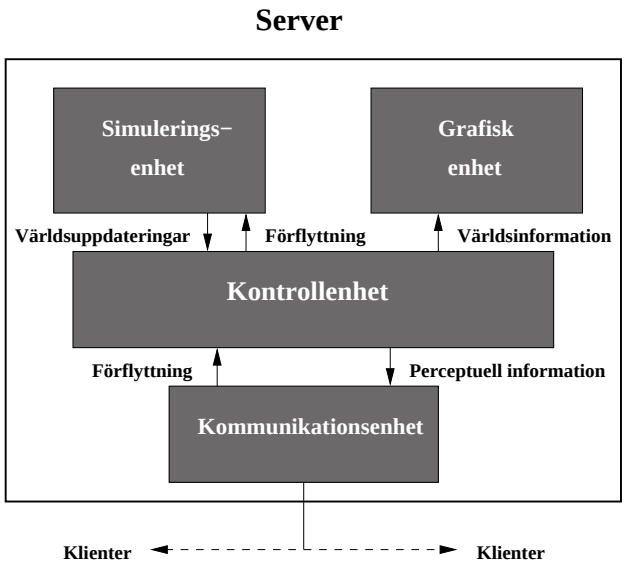
4.3.1 Server

Serverns huvudsakliga uppgift är att simulera perceptuell information för klienten. Det innebär att servern genererar den information som på den riktiga roboten skulle genererats av PAM. Denna information skickas normalt till klienten i exakt form. Det innebär att klienten erhåller de exakta robotrelativa koordinaterna för varje objekt. Möjlighet finns dessutom att addera olika grader av brus[1] till informationen. Positionsangivelserna är då inte längre helt tillförlitliga, utan kan skilja sig från de verkliga positionerna. Servern är uppdelad på ett antal enheter (se figur 4.1).

- *Kontrollenheten* agerar huvudprogram för servern. Det är också denna enhet som koordinerar arbetet mellan de andra enheterna.

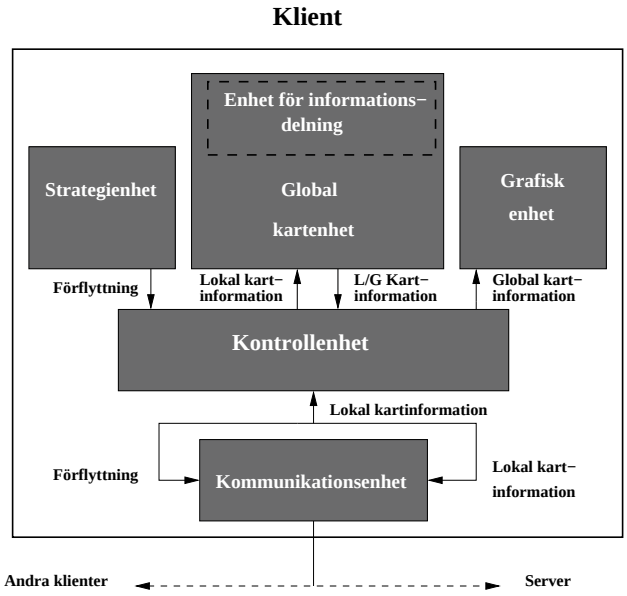
- *Simuleringsenheten* fungerar som själva hjärnan i simuleringsprocessen. Denna är bland annat ansvarig för perceptionsberäkningar för klienterna, strategier för motståndarlag, positionsuppdateringar av objekt samt beräkning av kollisionsdetektioner mm.

[1] Störningar i perception som orsakar felaktigheter i informationen.



Figur 4.1: Programstruktur över serverprogrammet.

Figur 4.2: Serverns grafiska gränssnitt.



Figur 4.3: Programstruktur över klientprogrammet.

- *Kommunikationsenheten* sköter kommunikationen mellan servern och klienterna. Den förser klienterna med lokal kartinformation samt tar emot rörelsekommandon från dessa.

- *Grafikenheten* används för att ge en grafisk representation av simuleringen under en körning. Denna använder sig av den globala kartan och visar vid varje tillfälle den faktiska kartbilden i simuleringsvärlden (se figur 4.2).

Strategier för motståndarlaget är för nuvarande baserade på ett enkelt reaktivt beteende.

4.3.2 Klient

Klienten är liksom servern uppbyggd av flera samverkande enheter (se figur 4.3).

- *Kontrollenheten* är den enhet som styr arbetet för klienten. Denna enhet koordinerar arbetet mellan samtliga andra enheter.

- *Strategienheten* planerar agentens nästa drag baserat på den för tillfället tillgängliga globala kartinformationen. Denna enhet är endast reaktiv.

- *Kommunikationsenheten* sköter kommunikation med server samt de andra klienterna. Enheten sänder rörelsekommandon till servern samt tar emot lokal kartinformation från denna. Enheten sköter dessutom utbytet av kartinformation mellan klienterna.

- *Grafikenheten* ansvarar för att ge en grafisk representation av simuleringen. Denna har som huvudsyfte att visa den av klienten uppbyggda globala kartan. Andra funktioner är: visa objekts riktiga positioner, visa förankring för objekt, visa synliga landmärken, visa oskarp lokaliseringskarta för robot etc. (se figur 4.4).

- *Den globala kartenheten* sköter självlokalisering samt konstruktion av den globala kartan (i samarbete med informationsdelningsenheten).

- *Enheten för informationsdelning* är inkorporerad i den globala kartenheten och sköter allt arbete som har med informationsdelning mellan klienter att göra. Den är också delaktig i konstruktionen av den globala kartan.

Den globala kartenheten använder sig av lokal kartinformation om landmärken för att göra en oskarp självlokalisering. Metoden som används är den exakt samma som den som beskrevs i avsnitt 2.3, och som också används på de riktiga robothundarna. Klienten är uppbyggd så att den på ett enkelt vis skall tillåta att olika informationsdelningsmetoder testas och utvärderas. Vid test av ny en informationsdelningsmetod är informationsdelningsenheten den enda enhet som behöver modifieras. Formen på informationsdelningsenheten måste dock följa en viss mall för att så skall vara fallet. Då är dessutom enheten kodmässigt direktöverförbar till roboten för att köras där.

Det bör nämnas att, förutom lokal kartinformation, får klienten även den verkliga globala kartinformationen från servern. Denna används dock bara till grafiken samt för att göra statistikberäkningar. Som det nämndes i inledningen till detta kapitel så håller klienten metoder för statistikföring. Vid en körning beräknas för varje positionsuppskattning av ett objekt, dess avvikelse från den verkliga positionen. Denna information beräknas för samtliga objekt och lagras till fil tillsammans med maxvärde, medelvärde och standardavvikelse för mätningarna.

Figur 4.4: Klientens grafiska gränssnitt. Den övre bilden visar klientens världsbild (till vänster), samt dess OPR (till höger). Den undre bilden visar information för kartdelningsmetoder.

Kapitel 5

Metoder för kartdelning

5.1 Inledning

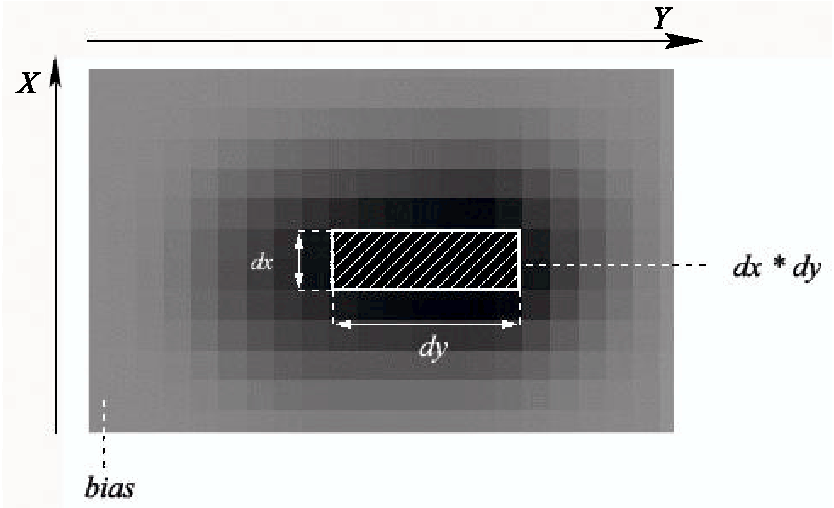
Detta kapitel ger en beskrivning av de kartläggningsmetoder, som vi utvecklat och implementerat. För vardera metod beskrivs representation av kartinformation, samt tillvägagångssätt för sammanvägning av information. Metoderna är uppställda efter beräkningsmässig komplexitet, i stigande ordning. Sist i kapitlet ges algoritmer för association av objekt samt ett distributions- och kombinationsschema för utbytet av information.

I detta kapitel avses med objekt endast boll, medspelare och motspelare.

5.2 Metod 1: Viktat medelvärde

5.2.1 Översikt

Denna metod använder sig av viktat medelvärde för sammanvägning av information om objekt. Metoden kan, för uppskattning av ett objekts position, liknas vid att robotarna fäster varsin elastisk snodd (av samma längd) kring en rörlig punkt P och att de sedan sträcker snodden till den position där de tror att objektet befinner sig. Låt varje robots snodd ha en tröghet som är proportionell mot den säkerhet med vilken de tror att ett objekt befinner sig på den givna positionen. För enkelhetens skull antar vi att snoddarna från början är oändligt korta och att de därefter kan sträckas oändligt mycket. Dessutom antar vi att trögheten i en snodd är konstant (oberoende av hur mycket snodden sträcks). Det resulterande läget för P kommer då att representera den uppskattade positionen för objektet. Denna position är en viktad sammanvägning av positionsangivelserna från de olika robotarna. Fördelen



Figur 5.1: Beräkning av precision respektive tillförlitlighet för positionsangivelser. Graden av svart i cellerna beskriver graden av möjlighet att roboten befinner sig på den givna positionen.

med metoden är att den beräkningsmässigt är väldigt billig. Representationen av data är också enkel och endast mycket lite information behöver utbytas mellan robotarna.

5.2.2 Representation och konstruktion av data

I den rutnätskarta OPR som hålles i Gk finns utförlig information angående robotens säkerhet på sin egen positionsuppskattning. Det skulle vara önskvärt att på något vis kunna överföra denna säkerhet till objekt i robotens omgivning. Dessa objekt finns i Lk representerade som en trippel (ρ, ϕ, a) där, som tidigare, $\rho \in \mathbb{R}$ och $\phi \in [-\pi, \pi]$ beskriver längd och orientering av vektorn mellan robot och objekt. Det tredje värdet $a \in [0, 1]$ beskriver hur färsk informationen är.

Vi börjar med att förenkla robotens OPR genom att bortse från information rörande robotens orientering. Detta kan göras genom att överföra rutnätet till den enklare distributionen $v_r : M \rightarrow [0, 1]$, där M är en tvådimensionell array av celler och varje cell representerar en position p = (x,y). Låt oss som tidigare beteckna robotens OPR som u_r så att u_p är den oskarpa mängden på position p. Som redan nämnts, är en sådan oskarp mängd en funktion $u_p : [-\pi, \pi] \rightarrow [0, 1]$. Vi kan beskriva distributionen v_r så att för varje cell p

är $v_r(p) = \|u_p(\theta)\|_{[-\pi,\pi]}$, vilket är höjden h av trapezoiden på positionen p. I denna distribution drar vi nu en minsta omslutande rektangel[1] (se figur 5.1) runt de positioner som håller MAX-värdet dvs. värdet 1.0. Detta är möjligt pga. att distributionen är normaliserad [2].

Enkelt uttryckt har vi bara valt ut de mest troliga cellerna i rutnätet och uppskattat området av dessa med en rektangel. Låt oss kalla den kompakta mängden av celler innanför den omslutande rektangeln för L. Man kan se storleken på L i förhållande till hela rutnätet som en angivelse på precisionen i robotens egen positionsuppskattning. Likaså kan MIN-värdet ($bias_r$) i distributionen ses som en angivelse på hur tillförlitlig informationen är.

Denna information kan nu överföras till de observerade objekten genom att projicera rektangeln L från roboten till respektive objekt. Observera att det nu kan vara av vikt att även ta hänsyn till robotens orientering. Låt oss därför införa några nya beteckningar. Beteckna med trippeln (x, y, θ) robotens mest troliga position och orientering i globala kartesiska koordinater. Här är $x, y \in \mathbb{R}$ och $\theta \in [-\pi, \pi]$. Dessa värden beräknades vid robotens självlokalisering. Beteckna dessutom med $dx, dy \in \mathbb{R}$ längden av sidorna i rektangeln L och slutligen $d\theta \in [-\pi, \pi]$ som ett värde:

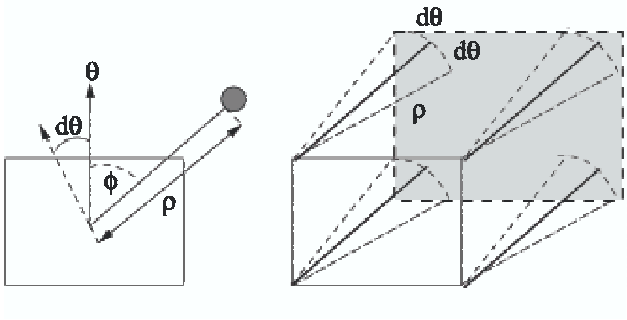
$$d\theta = \| |\theta - {\theta_0}^{ij}| + (\frac{{w_0}^{ij}}{2}) \|_L$$

Talen ${\theta_0}^{i,j}$ och ${w_0}^{ij}$ är mittpunkt respektive bredd på trapezoiden till cellen på index (i, j). Ytterligare är $\| \cdot \|_L$ normen definierad på sidan 22. Värdet på $d\theta$ kan ses som en uppskattad övre gräns på hur stort felet på robotens förmodade orientering kan vara. (Observera att så behöver inte vara fallet, felet kan vara större).

Genom att kombinera informationen från Lk med informationen ovan är det möjligt att överföra rektangeln L från roboten till ett objekt. Låt E vara det kompakta intervallet $E = [((\theta + \phi) - d\theta), ((\theta + \phi) + d\theta)]$. Beräkning av en rektangel L_o för ett objekt kan då informellt beskrivas som:

[1] Vi kommer i nästa metod att kalla en sådan rektangel med en bias för en oskarp positionslåda (OPL). Anledningen till det är att den kan ses som en approximation av en oskarp rutnätskarta.

[2] Att en möjlighetsdistribution är normaliserad innebär att det högsta värdet alltid är 1.0. Jämför med en sannolikhetsdistribution, där det är summan som skall vara lika med 1.0.



Figur 5.2: Beräkning av omslutande rektangel för ett objekt.

> Plotta kanterna till de cirkelsektorer som fås genom att translatera var och en av punkterna till L:s fyra hörn med (ρ, α), för alla värden på α där $\alpha \in E$. Då är L_o den minsta rektangel som omsluter alla punkter (se figur 5.2).

Rektangeln L_o är enkel att beräkna, exempelvis fås längden av sidorna i rektangeln genom följande räkning:

$$dx_o = dx + \rho \cdot \| \cos(\alpha) - \cos(\beta) \|_E$$
$$dy_o = dy + \rho \cdot \| \sin(\alpha) - \sin(\beta) \|_E$$

Detta förutsatt att det globala koordinatsystemet är positivt orienterat. För ett givet objekt med en omslutande rektangel kan vi nu beräkna ett värde för precisionen i positionsuppskattningen. Beteckna med A arean av spelplanen. Då kan precisionen (fokusen) på positionsuppskattningen uttryckas som $f = \frac{dx_o \cdot dy_o}{A}$ så att $f \in [0, 1]$. Som ett värde på tillförlitlighet väljes $t = \min(a, 1 - bias_g)$ så att $t \in [0, 1]$. Enligt tidigare är a ett förankringsvärde som anger hur färsk informationen är och $bias_r$ är biasen från robotens OPR. Genom att välja t som ovan kombineras tillförlitlighet från självlokaliseringen med tillförlitlighet om objektet i form av dess förankring.

Ett objekt kan då representeras som en 4-tupel (x, y, f, t) enligt beteckningarna ovan.

5.2.3 Sammanvägning av kartinformation

Information från flera robotar kan vägas samman genom att beräkna ett viktat medelvärde. Som vikter väljs värdena f och t. Dessa kan tillsammans

ses som en indikation på säkerheten i positionsuppskattningen. Beteckna med $(x, y, f, t)_i$ tupeln av information om ett objekt från robot i. Genom att kvotera vikterna från två robotar mot varandra får vi fram två nya värden $w_0, w_1 \in [0, 1]$ som beskriver förhållandet mellan vikterna:

$$w_0 = \frac{(f_0 \cdot t_0)}{(f_0 \cdot t_0) + (f_1 \cdot t_1)}$$
$$w_1 = \frac{(f_1 \cdot t_1)}{(f_0 \cdot t_0) + (f_1 \cdot t_1)}$$

Observera att sambandet $w_0 = 1 - w_1$ gäller för dessa värden. Med dessa värden kan informationen från två robotar enkelt vägas samman. Den sammanvägda positionsuppskattningen är en punkt p' på den räta linjen mellan punkterna $p_0 = (x_0, y_0)$ och $p_1 = (x_1, y_1)$. Koordinaterna för denna punkt beräknas som:

$$x' = (w_0 \cdot x_0) + (w_1 \cdot x_1)$$
$$y' = (w_0 \cdot y_0) + (w_1 \cdot y_1)$$

För att kunna upprepa samma typ av sammanvägning för informationen från flera robotar är det nödvändigt att beräkna vikterna för den nya positionen p'. Dessa nya sammanvägda vikter beräknas enligt samma metod som:

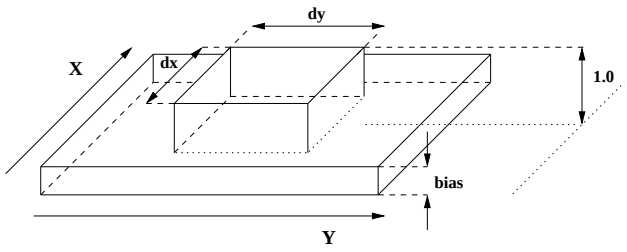
$$f' = (w_0 \cdot f_0) + (w_1 \cdot f_1)$$
$$t' = (w_0 \cdot t_0) + (w_1 \cdot t_1)$$

Processen upprepas för samtliga robotar vilket resulterar i en sammanvägd position som kan ses som ett viktat medelvärde av samtliga robotars positionsuppskattning.

5.3 Metod 2: Oskarpa positionslådor

5.3.1 Översikt

En av nackdelarna med metod 1 är att den endast till en liten del använder sig av informationen från självlokaliseringen rörande robotens positionsuppskattning. Säkerheten för positionsuppskattningen finns representerad som en möjlighetsdistribution i robotens OPR och det är önskvärt att kunna



Figur 5.3: Approximation av en oskarp rutnätskarta i form av en oskarp låda.

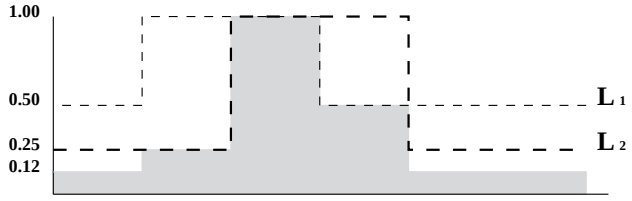
överföra denna till objekten. Enligt tidigare resonemang, är det möjligt skapa en omslutande rektangel så att denna beskriver precision i positionsuppskattningen av ett objekt. Tillsammans med en bias kan denna representation ses som en approximation av ett oskarpt rutnät (se figur 5.3. Genom att behålla denna representation och använda oskarp logik för att kombinera distributioner från flera robotar kan en ny sammanslagen distribution beräknas. Ur denna kan sedan en positionsuppskattning extraheras. Denna metod kräver mer beräkningskraft än den föregående men den är fortfarande relativt billig. Datarepresentationen kan göras enkel och mycket lite information behöver utbytas mellan robotar.

5.3.2 Representation och konstruktion av data

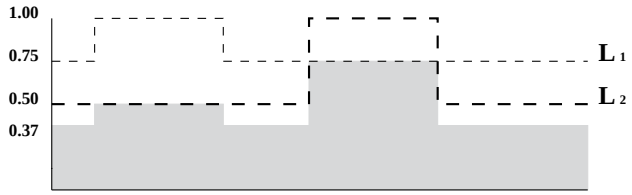
Konstruktionen av data är i stort densamma som för föregående metod. En omgivande rektangel beräknas för de celler i robotens OPR som anses mest möjliga (celler i vilka trapezoiden har höjden 1.0). Denna rektangel projiceras sedan på objekten enligt tekniken från föregående metod. Som tidigare betecknar vi längden och bredden av denna rektangel i globala koordinater med dx_o och dy_o. Istället för att beräkna storleken på rektangeln i förhållande till storleken på spelplanen, väljer vi att behålla denna representation. Detta är en approximation av det oskarpa rutnätet. Geometriskt kan det tolkas som att området som täcks av rektangeln har en höjd på 1.0 och området utanför har en höjd som ges av biasen (figur 5.3). Vi kallar en sådan representation för en oskarp positionslåda (OPL) , eller bara en oskarp låda. Som bias väljs ett värde $\min(a, 1 - bias_r)$ där a som tidigare är ett förankringsvärde som anger hur färsk informationen är och $bias_r$ är biasen från robotens OPR. Som tidigare innebär ett lågt värde på a att informationen är gammal och således inte tillförlitlig.

Representationen ser då ut som en 5-tupel $(x, y, dx, dy, bias)$ där x och y är

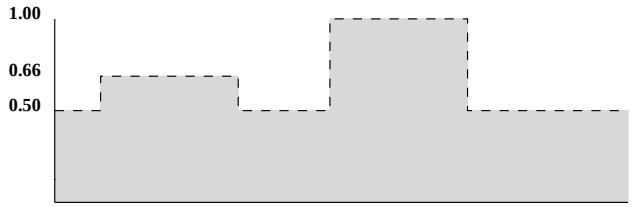
koordinaterna för mittpunkten av den oskarpa lådan.



Figur 5.4: Två i x-led överlappande oskarpa lådor. Det grå fältet beskriver den resulterande distributionen från sammanvägningen.



Figur 5.5: Två i x-led icke överlappande oskarpa lådor. Det grå fältet beskriver den resulterande distributionen från sammanvägningen.



Figur 5.6: Resultatet av en normalisering för de två icke överlappande oskarpa lådorna.

5.3.3 Sammanvägning av kartinformation

Sammanvägning av kartinformation i form av oskarpa lådor kan utföras på lite olika sätt beroende på hur mycket beräkningskraft och lagringsutrymme som står till förfogande. Lösningen som presenteras här är relativt billig och baseras på att sammanvägning görs för x och y-axel var för sig. Tillvägagångssättet är detsamma för båda axlarna. Metoden beskriven nedan har tid-

skomplexiteten $O(n)$ för sammanvägning över en axel av längden n. Tekniken upprepas för informationen från samtliga robotar.

Överlappande lådor

Betrakta figur 5.4, denna beskriver hur två lådor överlappar varandra i x-led. Den vertikala axeln beskriver ett möjlighetsvärde i intervallet [0,1] och den horisontella beskriver en global x-koordinat. Sammanvägningen kan göras genom att beräkna det oskarpa snittet för de två lådorna. Denna beräkning görs koordinat för koordinat längs hela axeln. Det oskarpa snittet för en given koordinat är en funktion $f : [0,1] \times [0,1] \rightarrow [0,1]$. Enligt avsnitt 2.4, sid. 24 kan en sådan typ av oskarpt snitt beräknas med en T-norm. Detta förutsatt att observationerna kan ses som oberoende, vilket är fallet här. För vårt ändamål har vi valt den vanliga algebraiska produkten. Det grå fältet i figur 5.4 visar resultatet från sammanvägningen. Observera att resultatet av sammanvägningen inte är en enkel rektangulär mängd (såvida inte de två lådorna är likformiga). Ett sätt att handskas med det, är att konvertera resultatet till en låda igen. Det medför dock att information går förlorad. Ett annat bättre sätt är att behålla den resulterande distributionen i form av en array, där varje position representerar en koordinat på axeln. Denna representation är betydligt mer kostsam än lådrepresentationen, men eftersom det bara är en intern struktur som inte skall delas mellan robotar, så är den extra kostnaden acceptabel. Vi har valt att använda denna senare, mer kostsamma representationen.

Icke överlappande lådor

För oskarpa lådor som inte överlappar, uppstår problemet att den sammanvägda strukturen inte längre är normaliserad. (Det högsta värdet är inte längre 1.0, se figur 5.5). Det finns flera lösningar till hur man kan normalisera distributionen. En lösning är att man till varje cell i arrayen adderar värdet $(1.0 - MAX)$. Här avser MAX det största värdet i distributionen. En annan lösning är att multiplicera värdet från vardera cell i arrayen med $\frac{1}{MAX}$. Metoderna skiljer sig på så sätt att den sista resulterar i en mer tillförlitlig distribution. Vi har valt den sistnämnda metoden. För exemplet i figuren 5.5 innebär det att varje cell multipliceras med $\frac{1}{0.75}$. Resultatet av normaliseringen kan ses i figur 5.6.

5.3.4 Extrahering av position

Från den sammanslagna distributionen kan en positionsuppskattning extraheras. Som vid sammanvägningen delar vi upp problemet så att x respektive y-axel behandlas var för sig. Låt X vara den array som representerar den sammanvägda fördelningen i x-led och anta att den har n_x element. Beteckna med $x(i) \in \mathbb{R}$ den globala x-koordinaten för cellen på index i och låt $u_x(i) \in [0, 1]$ vara värdet för denna cell. Den globala x-koordinaten kan beräknas från X genom en tyngdpunktsberäkning som liknar den som görs vid självlokaliseringen. Som namnet antyder kan metoden liknas vid att man söker tyngdpunkten för fördelningen. Metoden kan visualiseras genom följande liknelse:

> Låt arrayen X motsvara ett 1-dimensionellt linjesegment uppdelat på ett interval bestående av n_x celler och låt varje cell på linjen motsvara en viss vikt som är proportionell mot värdet i denna cell. Anta att detta segment nu skall balanseras på en spets som har storleken av en cell. Tyngdpunkten på fördelningen och således positionsuppskattningen för objektet ligger då i den cell för vilken balans av segmentet erhålles.

Positionsuppskattningen för x-koordinaten av ett objekt beräknas med ekvationen i 5.1. På samma vis kan den sammanvägda positionen i y-led beräknas (ekvation 5.2). Här är n_y antalet element för den array Y som representerar den sammanvägda fördelningen i y-led. Likaså är $y(i) \in \mathbb{R}$ den globala y-koordinaten för cellen på index i. $u_y(i) \in [0, 1]$ är värdet för denna cell.

$$x_{global} = \frac{\sum_{k=1}^{n_x}(x(k) \cdot u_x(k))}{\sum_{k=1}^{n_x}(u_x(k))} \tag{5.1}$$

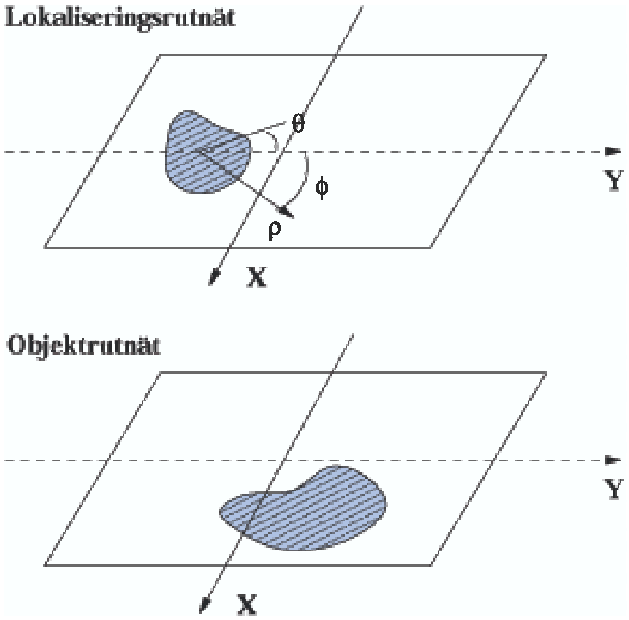
$$y_{global} = \frac{\sum_{k=1}^{n_y}(y(k) \cdot u_y(k))}{\sum_{k=1}^{n_y}(u_y(k))} \tag{5.2}$$

Tidskomplexiteten för extrahering av position för en array av längden n är $O(n)$.

5.4 Metod 3: Oskarpa positionsrutnät

5.4.1 Översikt

I metod 2 har vi approximerat informationen från robotens OPR till oskarpa positionslådor. I denna sista metod väljer vi att använda den fulla distributionen. På så vis drar vi nytta av all information rörande robotens säkerhet



Figur 5.7: Konstruktion av ett OPR för ett objekt.

gällande självlokaliseringen. Robotens rutnätskarta kan överföras från robot till objekt genom en form av projektion lik den som använts tidigare. Genom att använda oskarp logik för att kombinera möjlighetsdistributioner från flera robotar kan en ny sammanslagen distribution beräknas. Ur denna kan en positionsuppskattning sedan extraheras. Denna metod kräver mycket beräkningskraft och mycket information behöver utbytas mellan robotar.

5.4.2 Representation och konstruktion av data

Vi vill representera varje objekt från Lk som en möjlighetsdistribution i form av en oskarp rutnätskarta. En sådan representation är möjlig genom en teknik som påminner om konstruktionen av oskarpa lådor från metod 2. Kortfattat går tekniken ut på att man för varje cell i robotens OPR projicerar värdet från denna på en motsvarande cell i det objektrutnät som skall konstrueras.

Beteckna som tidigare robotens OPR som u_r och den oskarpa mängden på position p som $u_r(p)$ eller bara u_p. Låt u_o vara den oskarpa rutnätskarta som skall konstrueras för objektet. Detta rutnät innehåller för varje position p ett värde $[0, 1]$ som talar om hur möjligt det är att objektet befinner sig på positionen. Anledningen till att vi inte som för roboten använder en oskarp mängd för varje position, men det är att vi inte är intresserade av

Algoritm 1

for (alla celler p) **begin**
    $u_o(p) := 1 - a$
    for (alla celler q som uppfyller $\|\overrightarrow{pq}\| = \rho$) **begin**
        $u_o(p) := \max(u_o(p), u_q(\angle(p,q) + \phi)$
    end for
end for

Figur 5.8: Algoritm 1: konstruktion av ett objektrutnät.

ett objekts riktning. (Det kan tyckas vara av intresse att veta orienteringen av spelare, det är dock i nuvarande läget inte är möjligt att bestämma en spelares riktning. Framförallt beror detta på kamerans dåliga upplösning). Givet två positioner p och q i ett rutnät, betecknar vi med $\overrightarrow{pq}$ vektorn mellan p och q och med $\|\overrightarrow{pq}\|$ den euklidiska normen (längden) av denna vektor. Låt dessutom $\angle(p,q)$ vara orienteringen på $\overrightarrow{pq}$. Vi sätter punkten p att vara fix och låter M vara mängden av alla positioner q så att $\|\overrightarrow{pq}\| = \rho$. Här är som tidigare ρ värdet från robotens Lk som betecknar avståndet mellan robot och objekt. Vi kan då beräkna värdet av en position p i objektrutnätet som:

$$u_o(p) = \|u_q(\angle(p,q) + \phi)\|_M$$

Denna beräkning kan upprepas för varje cell i objektrutnätet vilket resulterar i en möjlighetsdistribution för objektet. Konstruktionen kan göras med algoritmen i figur 5.8. Initialiseringen $u_o(p) := 1-a$ på andra raden innebär är att om informationen är gammal, vill vi ha en hög bias för distributionen. Detta för att visa att informationen inte är helt tillförlitlig. Den beräkningsmässiga kostnaden för algoritmen är relativt hög. Om vi för enkelhetens skull bara betraktar kvadratiska rutnät där sidlängden är n, består ett helt rutnät av n^2 celler. Algoritmen kräver då $O(n^4)$ operationer. Genom att modifiera algoritmen är det dock möjligt att optimera den något. Lösningen går ut på att först gå igenom robotens OPR och att då endast betrakta de celler som har tillräckligt högt värde. Med tillräckligt högt värde avses (de celler som har) ett värde som är högre en ett visst satt tröskelvärde ϵ. Det kan åstadkommas genom att:

1. Lägga till en rad mellan rad 2 och 3: if $(\|u_p(\theta)\|_{[-\pi,\pi]} > \epsilon)$
2. Byta rad 4 till: $u_o(p) := \max(u_o(q), u_o(\angle(p,q) - \theta)$

Algoritm 2

for (alla celler p) **begin**
    $u_o(p) := 1 - a$
    if $(\|u_p(\theta)\|_{[-\pi,\pi]} > \epsilon)$ **then**
        for (alla celler q som uppfyller $\|\overrightarrow{pq}\| = \rho$) **begin**
            $u_o(q) := \max(u_o(q), u_p(\angle(p,q) - \theta)$
        end for
    end if
end for

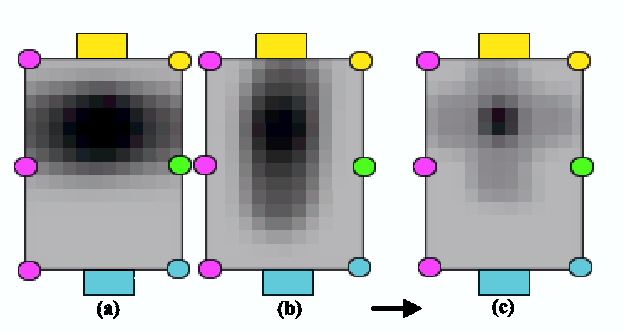
Figur 5.9: Algoritm 2: modifierad algoritm för konstruktion av ett objektrutnät.

Den modifierade algoritmen finns beskriven i figur 5.9. Tidskomplexiteten på den modifierade algoritmen är beroende av utseendet på distributionen från robotens OPR. Då roboten är osäker på sin position (många celler med höga värden i robotens OPR) är algoritmen långsam. Omvänt så är algoritmen snabb när roboten är säker på sin position (få celler med höga värden i robotens OPR). Metoden har samma värstafall som den föregående, dvs. $O(n^4)$, men är betydligt snabbare i medelfallet.

5.4.3 Sammanvägning av kartinformation

Sammanvägning av kartinformation från två oskarpa rutnätskartor påminner mycket om den sammanvägning som görs för roboten vid självlokaliseringen (beskriven i avsnitt 2.3 och 2.4). Problemet består i att väga samman två oskarpa distributioner för en given observation, i detta fall för en spelare eller bollen (vid lokaliseringen ett landmärke). Sammanvägningen kan göras koordinat för koordinat och resulterar då i en ny sammanslagen rutnätskarta. Metoden beskriven nedan har tidskomplexiteten $O(n^2)$ för kvadratiska rutnät med sidlängden n.

Rutnätskartorna som skall vägas samman kan ses som funktioner $u_o : M \rightarrow [0,1]$, där M är en 2-dimensionell array av celler. Eftersom cellerna i varje rutnät endast innehåller värden i intervallet $[0,1]$, är det möjligt att använda en T-norm som oskarp operator för att väga samman värden från celler i två olika rutnät. Som för de oskarpa lådorna har vi valt den algebraiska produkten för detta ändamål. Om ${u_o}^1$ och ${u_o}^2$ är två olika rutnätskartor för ett givet objekt så beräknas värdet för cellen på position (i,j) i det nya sammanslagna



Figur 5.10: Sammanvägning av två objektrutnät (a) och (b). Ju mörkare fälten i rutnäten är, desto större möjlighetsvärde. Figuren i (c) visar resultatet av sammanvägningen.

rutnätet som ${u_o}^1(i,j) \cdot {u_o}^2(i,j)$. Figuren 5.10 visar sammanvägningen av två oskarpa rutnätskartor.

Liksom vid sammanvägning av oskarpa boxar kan det inträffa att den sammanvägda distributionen inte blir normaliserad (dvs. om rutnäten innehåller motsägande information). Om så är fallet måste fördelningen normaliseras. Även här väljer vi att göra det genom att helt enkelt multiplicera alla celler med $\frac{1}{MAX}$, där MAX är det största värdet i den sammanvägda fördelningen.

5.4.4 Extrahering av position

Extrahering av positionsuppskattning från den sammanvägda rutnätskartan kan göras på liknande vis som för de oskarpa lådorna. Beteckna med u_o en rutnätskarta av storleken $n_x \times n_y$ och beteckna värdet av cellen i rutnätskartan på index (i,j) med $u_o(i,j) \in [0,1]$. En positionsuppskattning för ett objekt kan göras genom en tyngdpunktsberäkning. Denna beräkning liknar den som gjordes för de oskarpa lådorna. Skillnaden är att vi nu har två dimensioner att ta hänsyn till mot tidigare enbart en. Tyngdpunktsberäkningen kan som tidigare visualiseras genom en liknelse:

> Låt rutnätet u_o motsvara en 2-dimensionell yta uppdelad som ett raster bestående av $n_x \times n_y$ celler, och låt varje cell på ytan motsvara en vikt som är proportionell mot värdet i denna cell. Anta att denna yta nu skall balanseras på en spets som har storleken av en cell. Tyngdpunkten på fördelningen och således positionsuppskattningen för objektet ligger då i den cell för vilken balans av ytan erhålles.

Som för de oskarpa lådorna görs denna uppskattning enklast genom att hitta x respektive y-koordinat var för sig. Om $x(i) \in \mathbb{R}$ är den globala x-koordinaten för cellen på index $(i, \cdot)$ så kan en uppskattning av x-koordinaten beräknas med hjälp av ekvation 5.3. Beräkningen görs på samma vis för y-axeln (ekvation 5.4). Här är $y(i) \in \mathbb{R}$ den globala y-koordinaten för cellen på index $(\cdot, i)$.

$$x_{global} = \frac{\sum_{i=1}^{n_x}(\sum_{j=1}^{n_y}(x(i) \cdot u_o(i,j)))}{\sum_{i=1}^{n_x}(\sum_{j=1}^{n_y} u_o(i,j))} \tag{5.3}$$

$$y_{global} = \frac{\sum_{i=1}^{n_y}(\sum_{j=1}^{n_x}(y(i) \cdot u_o(i,j)))}{\sum_{i=1}^{n_y}(\sum_{j=1}^{n_x} u_o(i,j))} \tag{5.4}$$

Om vi som tidigare enbart betraktar kvadratiska rutnät med sidlängden n så har beräkningen tidskomplexiteten $O(n^2)$.

5.5 Association av objekt

För de objekt som det finns flera av dvs. medspelare och motspelare uppstår problemet att veta hur man skall associera objekt med ett annat. Vi väljer att betrakta problemet som en uppgift att hitta den optimala matchningen för ett antal mängder av objekt. Problemet har då många likheter med klassiska optimeringsproblem som t.ex. TSP-grafproblemet. Lösningen som presenteras här använder sig av en form av *brute force*[3] algoritm som helt enkelt testar alla kombinationer av matchningar och väljer den bästa. Denna lösning innebär dock två problem

1. Lösningen är beräkningsmässigt kostsam.
2. Hur avgör man om en matchning är bra?

5.5.1 Algoritm

Vi förutsätter att antalet robotar som utbyter informationen är detsamma som antalet objekt (robotar) som skall observeras. För det allmänna fallet ger det att information om n objekt från n robotar skall matchas. Förutsatt att vi har en funktion *evaluera*, som avgör hur bra olika mängder av objekt från flera robotar matchar, kan en algoritm som genererar och

[3]En metod som genererar mängden av alla tänkbara lösningar och från denna mängd väljer den bästa.

Associeringsalgoritm 1

```
max := 0
bästamängden := {∅}
for(alla ordnade mängder S_k från Robot_1) begin
   for(alla ordnade mängder S_l från Robot_2) begin
      ...
        for(alla ordnade mängder S_m från Robot_n) begin
            matchvärde := evaluera(S_k, S_l, ... , S_m)
            if(matchvärde > max) then
              bästamängden := {S_k, S_l, ... , S_m}
              max := matchvärde
            end if
        end for
      ...
    end for
end for
```

Figur 5.11: Algoritm 1: associering av objekt från n robotar. Algoritmen genererar och testar alla permutationer av objekt.

testar alla permutationer skrivas som i figur 5.11. Med S_k avses här en ordnad mängd (permutation) av objekten som roboten skall göra positionsberäkningar för. När algoritmen terminerar finns den bästa matchningen representerad i mängdvariabeln *bästamängden*. Denna håller för var robot den permutation av objekten som enligt algoritmen ger den bästa matchningen.

Den beskrivna algoritmen har tidskomplexiteten $O((n!)^{(n-1)})$. Algoritmen är alltså väldigt kostsam. I SLRL-miljön, där lag om fyra spelare tävlar, innebär det att det finns $(4!)^3 \approx 14000$ kombinationer. För fallet med fyra spelare är det kanske möjligt (om än kostsamt) att göra en komplett matchning, förutsatt att vi har en metod för att avgöra huruvida en matchning är bra, och att denna inte innebär allt för tunga beräkningar. Ökar vi istället antalet spelare från fyra till fem st, blir antalet kombinationer $(5!)^5 \approx 210$ miljoner och det är då inte längre praktiskt möjligt att genomföra en total matchning. Vi väljer därför att modifiera metoden så att vi bara matchar information från två robotar åt gången. Detta kan göras på två sätt. Antingen väljs en robot som alla får matcha mot, eller så matchar robotarna sin information mot redan sammanvägd information. Metoden innebär att vi nu

Associeringsalgoritm 2

matchmängd := någon mängd $M_1{}^l$ från $Robot_1$
bästamängden := matchmängd $\cup$ $\{\emptyset\}$
for(alla n robotar $Robot_k$ utom $Robot_1$) **begin**
  max := 0
  for(alla ordnade mängder $M_k{}^l$ från $Robot_k$) **begin**
    matchvärde := evaluera(matchmängd, $M_k{}^l$)
    if(matchvärde > max) **then**
      temporär := $\{M_k{}^l\}$
      max := matchvärde
    end if
  end for
  bästamängden := bästamängden $\cup$ temporär
end for

Figur 5.12: Algoritm 2: associering av objekt från n robotar. I algoritmen matchas permutationer av objekt från de olika robotarna mot en godtyckligt vald permutation av objekten från robot 1.

kan göra parvisa matchningar för objekten från två robotar. Denna modifierade metod kan ses som en approximation av den fullständiga matchningen. Tidskomplexiteten för den modifierade algoritmen är $O((n!) \cdot (n-1))$, vilket fortfarande är mycket, men innebär ändå en stor förbättring. För fallet med 4 spelare som utbyter information om 4 objekt, blir det $(4!) \cdot 3 = 72$ kombinationer jämfört med tidigare 14000 och för fallet 5 blir det $(5!) \cdot 4 = 480$ mot tidigare ca 210 miljoner. Det är en klar förbättring. Det är möjligt att optimera algoritmen ytterligare, men vi nöjer oss med den här lösningen. Den modifierade algoritmen finns beskriven i figur 5.12, där $M_1{}^l$ är en godtycklig permutation av objekten från robot 1, som övriga robotar matchar sin information mot. Även här håller *bästamängden* för vardera spelare, den permutation, som enligt algoritmen ger den bästa matchningen.

5.5.2 Värderingsfunktioner

Det andra problemet är att avgöra hur bra en matchning är. Eftersom vi bara tittar på två objekt åt gången kan vi använda en värderingsfunktion som talar om hur bra informationen om två objekt matchar. En sådan värderingsfunktion kan konstrueras på lite olika sätt beroende på vilken information som finns om objekten, med andra ord hur information om objekten finns

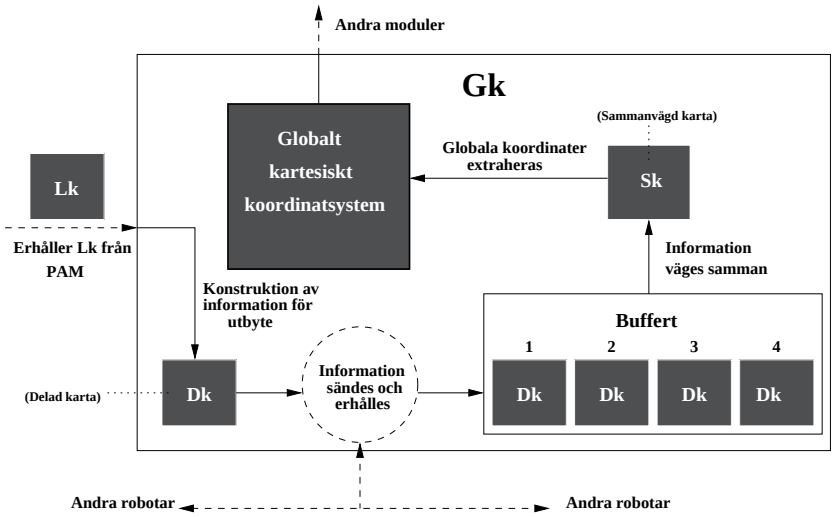
representerad. Det enklaste sättet är att helt enkelt enbart titta på avståndet mellan de två objekten och låta ett litet avstånd innebära en bra matchning och ett stort avstånd en dålig matchning. Det inses dock ganska snabbt att denna metod inte alltid är optimal. En teknik som vi har använt och som har visat sig vara effektiv, är att göra en simulerad sammanvägning av informationen från de två objekten som skall matchas. Sammanvägningen görs enligt någon av de tre metoder som beskrivits i kapitlet beroende på vilken representation som valts. Hur resultatet av den simulerade sammanvägningen skall tolkas är också beroende av vilken metod som har valts. För fallen med de oskarpa lådorna och de oskarpa rutnätskartorna har vi använt samma teknik. Tekniken finns för fallet med oskarpa lådor kortfattat redovisad nedan.

Betrakta två oskarpa lådor ${u_o}^1$ och ${u_o}^2$, var och en innehållande information om ett objekt, möjligtvis samma. Antag att vi vill ha ett värde för hur bra informationen från de två lådorna överensstämmer. Vi löser det genom att väga samman informationen och normalisera den resulterande distributionen. Utifrån denna summerar vi värdet från varje cell på vardera axeln (x och y) och delar summan med det totala antalet celler. Det beräknade värdet kommer att vara ett tal $v \in [0, 1]$ så att ett lågt värde (v nära 0.0) innebär en bra matchning och ett högt värde (v nära 1.0) innebär en dålig matchning.

Denna metod kan användas på liknande vis för rutnätskartorna. För fallet med rutnätskartor är metoden dock aningen kostsam, nämligen $O(n^2)$ för varje parvis matchning. För 4 spelare upprepas denna beräkning 72 gånger. I detta fall är processen *kanske* möjlig, men ökar man antalet spelare så är den inte längre realistisk. Metoden kan dock optimeras genom att man konverterar rutnätet till ett mindre finmaskigt nät, alternativt en oskarp låda. I det sista fallet får vi linjär tidskomlexitet för varje parvis matchning.

5.6 Distribution av kartinformation

Utbytet av information mellan robotar följer schemat i figur 5.13. Lokal kartinformation erhålles från PAM med jämna intervall. Utifrån denna information konstrueras enligt någon av de tidigare beskrivna metoderna den information som skall skickas mellan robotarna. Låt oss kalla strukturen innehållande denna information, för den *delade* kartan (Dk). Roboten skickar sin delade karta till samtliga andra robotar samt erhåller motsvarande strukturer från de andra. Varje robot håller en intern buffert i vilken Dk-strukturer lagras. Bufferten håller inte bara strukturer från de andra robotarna utan även robotens egen. När en Dk erhålls från en annan robot sätts en flag-



Figur 5.13: Schema för distribution.

ga för att visa att denna information är ny. Informationen från bufferten vägs med jämna intervall ihop till en ny struktur, den sammanvägda kartan (Sk). Från den sammanvägda kartan extraheras positionsuppskattningar för samtliga objekt och den globala kartan konstrueras.

Processen beskriven ovan upprepas så ofta det behövs. Vi har valt att upprepa processen varje gång roboten gör en ny självlokalisering (1Hz cykel).

Kapitel 6

Resultat

Resultatet i detta kapitel är endast baserat på körningar i RoboSimulatorn. Anledningen till det, är den stora svårigheten att utvärdera resultatet av programvaran på hundarna. Metod 2 (oskarpa positionslådor), användes på de riktiga robothundarna på RoboCup 2002. Resultatet tas upp i diskussionen (se kapitel 7).

6.1 Tester av kartdelningsmetoder

De olika metoderna beskrivna i föregående kapitel, har i flera tester granskats ingående. Faktorer som har varit av intresse att titta på är: Påverkan av olika synförhållanden och påverkan av olika grader av brus i perceptionen. Att addera brus till perceptionen innebär att positionsangivelser för objekt och landmärken från servern riskerar att vara lite felaktiga. Graden av brus är angiven i % och har betydelsen av hur felaktigt avståndet till ett objekt som mest kan vara, i förhållande till det verkliga avståndet. Resultaten av testerna för de olika metoderna har därefter jämförts med varandra. Metoderna har också jämförts med en lösning som inte använder sig av informationsutbyte. Värdena i testerna är angivna i enheten simulatorcentimeter och skall inte misstas för verkliga centimeter. Värdena ger dock en fingervisning om hur resultatet borde vara på de riktiga hundarna och kan framförallt ge ett mått på hur metoderna skiljer sig åt. Under *normala* synförhållanden har de simulerade robotarna ett synfält bestående av en cirkelsektor med vinkeln 180^o grader och räckvidden 2 meter. Spelplanen de spelar på har måtten 4×5 meter. I de tester då olika synförhållanden utvärderats, har räckvidden på synfältet modifierats (vinkeln har lämnats oberörd). I de tester då påverkan av brus har utvärderats, har robotarna i övrigt normala synförhållanden. Nedan redovisas resultaten från de tester som utförts under normala synförhållanden

Tabell 6.1: jämförelse av metoder

Metod	Maxvärde	Medelvärde	Standardavvikelse
Utan	442/299	89/122	107/54
Metod 1	212/413	31/93	26/44
Metod 2	241/221	33/100	29/31
Metod 3	595/386	76/126	102/52

Tabell 6.1: Maxvärde, Medelvärde och standardavvikelse för 10000 uppskattningar av bollens samt spelares positioner. Värden i tabellen avser avståndet mellan den uppskattade respektive den faktiska positionen för bollen. I tabellen redovisas resultat för boll/spelare.

Tabell 6.2: jämförelse av metoder

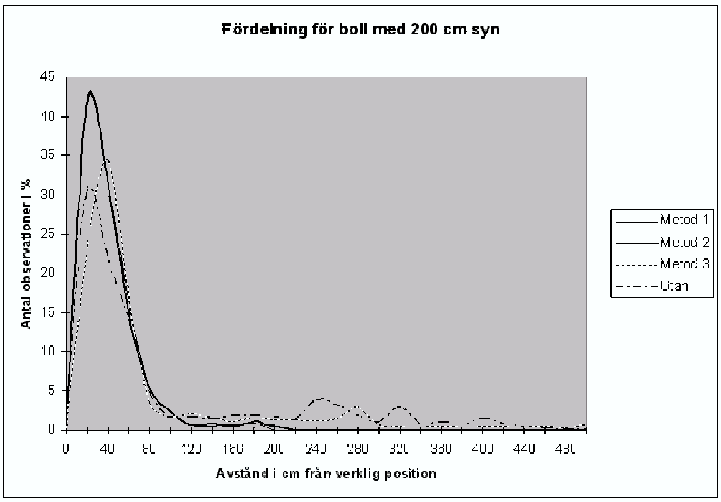
Metod	0-100 cm	100-200 cm	200-300 cm	400- cm
Utan	7246/3467	840/5352	1099/1179	815/0
Metod 1	9742/6157	256/3583	2/218	0/42
Metod 2	9630/4801	367/5149	3/50	0/0
Metod 3	7941/3735	774/5180	770/1015	515/70

Tabell 6.2: Fördelning av positionsuppskattningsfel för de olika metoderna. Fördelningen bygger på 10000 uppskattningar av bollens samt spelares positioner. Värdena i tabellen avser antalet observationer inom den givna felmarginalen. I tabellen redovisas resultatet för boll/spelare.

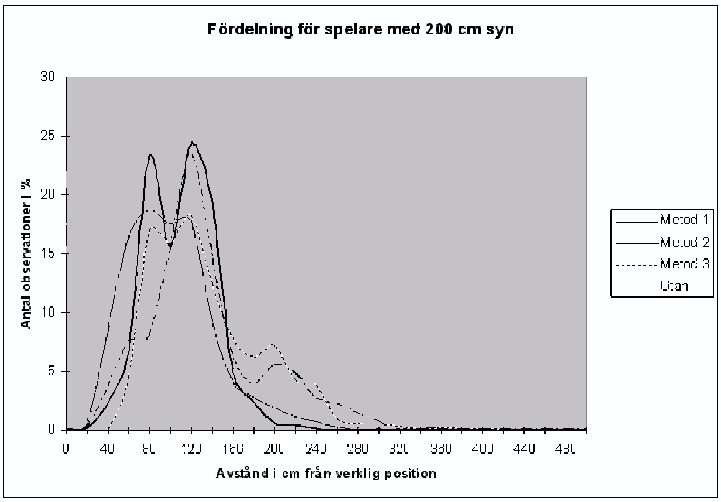
och utan brus. Den fullständiga förteckningen över tabeller och diagram finns i bilaga C.

6.2 Körtider för metoder

Vardera metod har körts på en 450Mhz SPARC Ultra-80 CPU. De genomsnittliga körtiderna för de respektive metoderna finns listade i tabell 6.3. Värdena i tabellen avser körtiderna för en cykel av informationsutbyte och sammanvägning av kartinformation. Värdena är uppmätta under normala synförhållanden och utan brus. Observera att metod 3, har olika körtider beroende på utseendet på robotens OPR (se avsnitt 5.4.2, sid 48). En hög



Figur 6.1: Diagram över fördelning för boll. Fördelningen bygger på 10000 uppskattningar av bollens position. Värdena i tabellen avser antalet observationer inom den givna felmarginalen.



Figur 6.2: Diagram över fördelning för spelare. Fördelningen bygger på 10000 uppskattningar av spelares positioner. Värdena i tabellen avser antalet observationer inom den givna felmarginalen.

Tabell 6.3: körtider för metoder

Metoder	tid i sek
Metod 1	0.0007
Metod 2	0.0017
Metod 3	0.0973

Tabell 6.3: Genomsnittliga körtider för en cykel av kartdelning på en 450Mhz SPARC Ultra-80 CPU. Testerna är gjorda under normala synförhållanden och utan brus.

osäkerhet i distributionen innebär en längre körtid och omvänt (I de fall då roboten är helt ovetande om sin position kan metod 3 ha en körtid på upp mot ett par sekunder).

I tabellen ser vi att metod 1 och metod 2 har korta körtider (ca 1 millisekund). Metod 2 tar cirka dubbelt så lång tid som metod 1. Vi ser också att metod 3 är avsevärt mycket långsammare än både metod 1 och 2. Den har en körtid på ca 0.1 sekund, vilket innebär att den (under gynnsamma förhållanden) tar drygt 100 ggr så lång tid som metod 1.

6.3 Självlokalisering

För att underlätta utvärderingen av resultaten, har vi även undersökt robotens självlokalisering. Vi har i tester låtit roboten lokalisera sig själv under samma förhållanden som i de övriga testerna. Resultatet finns i tabell 6.4. Med hjälp av resultatet hoppas vi kunna göra en mer ingående bedömning av egenskaperna för kartdelningsmetoderna.

Tabell 6.4: självlokalisering

Syn	Maxvärde	Medelvärde	Standardavvikelse
Normal syn	132	27	13
Syn 130 cm	139	44	16
Syn 550 cm	132	27	11
Brus 25%	128	53	14
Brus 50%	141	68	16

Tabell 6.4: Maxvärde, Medelvärde och standardavvikelse för en robots självlokalisering. Värdena i tabellen avser avståndet mellan den uppskattade och den faktiska positionen för bollen. Testerna är baserade på 10000 observationer.

Kapitel 7

Diskussion

7.1 Inledning

Metoderna för kartutbyte har utförligt testats i RoboSimulatorn. Resultatet av testerna finns redovisade i föregående kapitel samt bilaga C. Metod 2, som använder sig av oskarpa lådor, har dessutom prövats i den verkliga SLRL-miljön på RoboCup 2002.

I rapporten har vi behandlat medspelare och motspelare lika. I det verkliga fallet skiljer sig dessa från varandra på så sätt att en medspelare har möjligheten att själv ge information om sin position. Detta fenomen har vi avsiktligt och genomgående nonchalerat i rapporten för att förenkla problemställningen. I den programvara som användes på RoboCup utnyttjade vi dock denna möjlighet. Vi valde då enbart att använda positionsuppskattningar från vardera spelare, utan att göra någon sammanvägning med informationen om observerad objekt. För att förbättra informationen ytterligare kan denna kombineras med perceptionsinformation.

7.2 Förväntat resultat

De tre metoderna för kartdelning beskrivna i kapitel 5, använder sig av information med varierande grad av mängd och detaljrikedom. Ett rimligt antagande är att en metod som använder sig av mer detaljerad information, ger ett bättre resultat än en som använder sig av mindre detaljerad information (förutsatt att informationen utnyttjas rätt). Metod 3, beskriven i avsnitt 5.3, använder sig av rutnätskartor för att göra positionsuppskattningar för objekt. Informationen om vardera objekt finns utförligt beskriven i form av ett oskarpt rutnät. Denna metod är kostsam, men bör

ha goda kartläggningsegenskaper. För metod 2, används oskarpa positionslådor istället för ett helt rutnät. Representationen är en approximation av rutnätskartorna från metod 3, och metoden borde därför ha liknande egenskaper som denna. Resultaten kan dock antas skilja sig aningen med avseende på exakthet. Där det i metod 3 finns ett helt rutnät av celler med möjlighetsvärden, finns nu bara två nivåer av värden: En bias (lågt värde), samt en låda (högt värde). Information har alltså bortprioriterats, och det bör ha inverkan på resultatet. Slutligen har vi metod 1, som helt avstår från att göra beräkningar med oskarp logik. Istället konverteras robotens rutnät till två vikter som används för att beräkna ett medelvärde. Vid denna övergång förloras mycket information, vilket bör ha en uppenbar inverkan på resultatet.

7.3 Analys av resultat

Innan man kan dra några slutsatser av resultat från testerna, är det nödvändigt att sätta upp riktlinjer för vad som utmärker ett bra resultat. Det finns flera faktorer som man kan titta på, däribland medelvärde för feluppskattningar, standardavvikelse för feluppskattningar, maxvärde för feluppskattningar etc. Tar man dessutom i beaktande de tester som gjorts för varierande synförhållanden och olika grader av brus i perceptionen, så inför man ytterligare faktorer. Valet är naturligtvis också beroende på applikationsområdet för metoderna. Vi har valt att titta på en kombination av flera faktorer, varav de ovan nämnda. Det innebär att ett lågt medelvärde, en låg standardavvikelse samt ett lågt maxvärde alla är egenskaper som bidrar positivt till en metod. Omvänt påverkar höga värden resultatet negativt. Vi har dessutom undvikit att dra för mycket direkta slutsatser från resultaten när ovissheter föreligger.

7.3.1 RoboSimulatorn

Orsaken till att vi i resultatet redovisat värden för både boll och spelare, är att dessa skiljer sig från varandra angående behovet av association. Det innebär att skillnaden mellan resultatet för boll och spelare för en given metod och kan ses som ett värde på hur bra associationen av spelarna lyckats. En liten skillnad innebär i så fall att matchningen var lyckad och omvänt. Vad gäller resultatet för uppskattning av spelarnas positioner då ingen kommunikation utnyttjas, så kan detta vara aningen missvisande. Anledningen till det är att klienterna alltid erhåller information om spelare från servern. Servern tillhandahåller information både om position och identifiering av objekten på spelplanen, vilket medför att en klient då inte behöver göra någon

objektassociation för spelarna själv. Observera att i verkligheten, dvs. utanför simulatorvärlden, finns associeringsproblemet även då ingen information utbytes mellan robotarna. Då en robot får ny perceptuell information från den visuella enheten måste denna kombineras med tidigare erhållen information. Vi ger ett kort exempel: En klient har observerat ett antal spelare. Strax därefter tappar den kontakten med dessa. En liten stund senare observerar den återigen ett antal spelare (möjligtvis samma). Den vill nu inte förkasta informationen om de tidigare spelarna, utan istället skall informationen om de tidigare observerade spelarna matchas med de nu observerade spelarna. Problemet är att veta vilken spelarinformation som skall uppdateras. Detta problem löses automatiskt i simulatorvärlden av servern. Av detta skäl kan värden för observationer om spelare i simulatorvärlden vara lite bättre än vad de egentligen borde vara i den verkliga miljön.

Kartläggning av bollen

Från resultatet i kapitel 6 och bilaga C, kan man ganska omgående urskilja att samtliga kartdelningsmetoder tenderar att ge ett bättre resultat än det som erhålls då ingen kartinformation delas. Resultaten från metod 1 och metod 2 ser dessutom riktigt bra ut. I tester under normala synförhållanden är den genomsnittliga felmarginalen för positionsuppskattningen av bollen jämförbar med den från robotens självlokalisering (jämför tabell 6.1 med 6.4). Från resultatet kan man möjligtvis också uttyda att metod 1 i de flesta testerna visar ett lite bättre resultat än metod 2. Skillnaden är dock så marginell att den kan bero på andra faktorer, (implementation, slump etc.), än just metodernas kartläggningsegenskaper.

Likaså kan man uttyda att metod 3 tenderar att ge ett något sämre resultat än de andra två metoderna. I detta fallet är tendensen dock ganska tydlig och verkar vara genomgående för samtliga tester. Undantaget är när den simulerade roboten har god perception; då har denna metod värden som motsvarar de andra två metoderna. Detta kan få en att misstänka att skillnaden beror av andra faktorer än just metodens kartläggningsegenskaper. Vi vet att robotarna ibland blir osynkroniserade, vilket kan antas vara en orsak till det sämre resultatet. Frånvaron av synkronisering i sin tur orsakas dels av metodens höga beräkningsmässiga kostnad (vilken leder till fördröjningar). Dels orsakas den av att metoden har varierande körtider beroende på dess säkerhet från självlokaliseringen. (Kostnad för konstruktion av objektrutnät är beroende av utseendet på robotens OPR). I de tester då roboten antingen har väldigt begränsat synfält eller mycket brus i informationen, får metod 3

osedvanligt dåligt resultat (se figur C.3 och C.9). Här ger resultatet intrycket av att vara nästan slumpmässigt. I dessa situationer har roboten dock en väldigt osäker distribution i sin OPR, vilket kan resultera i körtider på upp mot ett par sekunder för en cykel i värstafallet (se under 6.2). Resultatet av det är att robotarna kommer i kraftig otakt. Möjligtvis kan det även finnas mindre brister i implementationen, som förstärker dessa negativa egenskaper. Metod 3 borde ur ett teoretiskt perspektiv ge ett bra resultat. Det skulle därför, för att få mer klarhet, vara önskvärt att undersöka programvaran för metod 3 närmare. Man kan möjligtvis dessutom försöka optimera algoritmer ytterligare.

Kartläggning av spelare

De övergripande egenskaperna för kartläggning av spelare för de respektive metoderna verkar i stort vara desamma som för kartläggning av bollen. Vi kan dock notera några skillnader här. Det är inte längre lika stor skillnad mellan metod 3 och de andra två metoderna. Likaså tenderar metod 2 nu att vara lite bättre än metod 1 (marginell skillnad) till skillnad mot tidigare. En trolig orsak till dessa fenomen är att metod 2 och metod 3 använder sig av mer komplexa och således också bättre värderingsfunktioner för objektassociation, än vad metod 1 gör. Om man tittar på diagrammen i bilaga C, för observationer av spelare, kan man urskilja att kurvorna för varje metod oftast har två mer framträdande toppar; jämför med kurvor för bollen, dessa har oftast bara en framträdande topp. Denna skillnad antar vi återspegla den enda skillnaden som finns mellan kartläggning av boll och kartläggning av spelare: associationsproblemet. En misstanke är att då robotarna matchar rätt, får de väldigt bra värden och att dessa centreras runt en punkt. Då robotarna matchar fel, får de dåliga värden vilka centreras runt en annan punkt. Detta är ett särdrag som i så fall borde vara mer framträdande då roboten har god perception (färre andra felkällor). Tittar man närmare på diagram C.6 så får man detta bekräftat. Här kan man tydligt se att vardera metod har två toppar och att dessa är fördelade på ungefär samma områden. Dessutom ser man i diagram C.5, om man jämför med C.6, att en av topparna på spelarkurvorna är belägen på samma område på den horisontella axeln, som toppen för bollen, vilket ytterligare bekräftar misstanken nämnd ovan. Ytterligare kan man se att för metod 2 och 3 är den högsta av topparna den som representerar liten felmarginal, medan topparna för metod 1 är ungefär lika. Detta stärker det tidigare förmodandet att metod 2 och metod 3 har något bättre egenskaper vad gäller objektassociation.

Som helhet kan man säga att resultatet för kartläggningen av spelarna var något sämre än förväntat. Samtliga metoder har för normala synförhållanden en genomsnittlig felmarginal på runt 100 cm, vilket är tämligen mycket. Jämför denna med den genomsnittliga felmarginalen hos bollen som ligger på ca 30 cm. Denna skillnad är är troligtvis direkt orsakad av dålig objektassociation.

7.3.2 RoboCup 2002

Metod 2, som använder sig av oskarpa lådor, användes av Team Sweden på RoboCup 2002. Metoden användes dock bara för bollen och för medspelarna användes istället en enkel version av informationsutbyte. Denna innebär att spelarna rakt av utbyter sina egna positionsuppskattningar (globala kartesiska koordinater) med varandra; information vägdes alltså inte samman med den perceptuella informationen. För motspelarna utbyttes ingen information.

Det är svårt att utvärdera exakt hur väl metoden kartlade objekt i den verkliga miljön. En av orsakerna till det är svårigheten att avgöra hur mycket robotens kartbild bidrar till dess spelegenskaper. Om någon annan del i programmet felar, så är en bra världsbild inte till så stor nytta och vice versa. Detta till trots fanns det många situationer då robotarna helt klart uppvisade ett beteende av att på ett effektivt sätt utnyttja informationen från de andra spelarna. Sådana situationer innebar t.ex. att roboten kunde gå mot bollen trots att den hade kameran riktad åt något annat håll. I sådana situationer kunde man istället se en annan robot stå vänd mot bollen och observera den.

Robotarna gav intrycket av att största delen av tiden ha en god uppfattning om bollens position. För att få en mer exakt uppfattning av metodens egenskaper skulle det vara nödvändigt att utvärdera metoden ytterligare i verklig miljö.

7.4 Möjliga förbättringar

Metoderna beskrivna i denna rapport har visat prov på goda egenskaper gällande kartläggning av dynamiska objekt. Nedan redovisas några förändringar och utökningar som vi anser skulle kunna förbättra kartläggningen ytterligare.

7.4.1 Minne över tidigare positioner

Det vore intressant beakta huruvida man skulle kunna förbättra kartläggningen genom låta roboten hålla ett minne över tidigare positioner för objekten. Det verkar rimligt att anta att så är fallet, frågan är hur detta skall göras.

Genom att hålla ett minne över tidigare positioner skulle exempelvis färdriktningen för ett objekt kunna uppskattas. Denna information kan ge en eventuell indikation på objektets position även då ingen ny information för objektet erhållits. Troligtvis skulle denna information även kunna användas för att eliminera risken för totala feluppskattningar för objekt (t.ex. som en följd av dålig självlokalisering). Slutligen är det en förhoppning att sådan information skall kunna användas för att förbättra association av objekt.

En form av *virtuellt* minne kan uppnås genom att för varje cykel väga samman information med redan tidigare sammanvägd information. Det innebär att den resulterande distributionen efter en sammanvägning sparas undan (observera att hittills har vi kastat informationen efter varje cykel). Vid nästa sammanvägning kombineras informationen om de observerade objekten med informationen i den undansparade strukturen. Denna procedur kan upprepas för varje cykel. På så vis används vid varje uppdatering även tidigare information om respektive objekt. Denna teknik har testats på de oskarpa lådorna. Resultatet visade sig dock vara sämre än vi hoppats på. En av anledningarna till det är att den undansparade distributionen har blivit föråldrad. Vid sammanvägningen leder det till att uppskattade positioner för objekt släpar efter. För att motverka detta fenomen prövade vi att föråldra den sammanvägda informationen genom att göra en form av *suddning* som liknar den som görs vid robotens självlokalisering (avsnitt 2.3, sid 22). Denna suddning innebär att vi adderar osäkerhet till distributionen genom att öka värdet i varje cell. Denna modifikation gav ett betydligt bättre resultat, men var fortfarande sämre än de metoder som kastade informationen efter varje cykel. Vad detta beror på är osäkert; metoden bör ha goda egenskaper och det är av intresse att undersöka det vidare.

7.4.2 Förbättrad objektassociation

I testerna i RoboSimulatorn har positionsuppskattningar för spelare visat sig vara mindre tillförlitliga än de för bollen. En orsak till det är att informationen inte alltid kombineras på rätt vis. Med det avser vi problemet med att korrekt associera två mängder av objekt med varandra. Det vore intressant att se hur denna process skulle kunna förbättras.

En tanke är att man helt enkelt avstår från att associera informationen för objekten (denna idé berördes kort i avsnitt 3.1.3). Istället kombineras informationen som den är, och problemet övergår till att i efterhand isolera information för respektive objekt. Förutsättningen för denna metod skulle vara att man valt en lämplig representation för kartinformationen. Här följer ett konkret exempel: För de oskarpa rutnätskartorna finns möjligheten att hålla ett rutnät för varje *grupp* av objekt (t.ex. motspelare), istället för ett rutnät för varje objekt. Rutnät för två robotar kan kombineras på vanligt vis, vilket resulterar i en ny rutnätskarta som även den håller information om samtliga objekt i gruppen. Från denna kan sedan vart och ett av objektens positioner extraheras. Den svåra frågan är på vilket sätt positioner skall väljas att separeras från den kombinerade mängden.

7.4.3 Förbättrad självlokalisering

Egenskaperna för självlokaliseringen för en robot som den är idag är tämligen goda. Den har en genomsnittlig felmarginal på knappt 30 cm (se tabell 6.4) för en 4×5 meters spelplan. Förutsättningarna för det, är att normala synförhållanden råder. Ökar man storleken på spelplanen, ökar dock denna felmarginal snabbt (jämför med de värden i tabellen då synen antingen är begränsad eller påverkad av brus). Av detta skäl kan det vara av intresse att försöka förbättra lokaliseringen något.

I den nuvarande versionen av robotens självlokalisering används endast informationen från landmärken i robotens omgivning. Självlokaliseringen skulle med fördel kunna utökas till att använda sig av information även om objekt, i första hand informationen om egenobserverade objekt, men senare även den information om objekt som erhålles från andra spelare. Ett enkelt sätt vore att använda informationen för att utesluta positioner för roboten som är orimliga. Det kan exempelvis göras genom att de positioner i robotens rutnät som medför att objekt skulle placeras utanför spelplanen ges en lägre möjlighet. Att bestämma exakt hur information skall användas för att förbättra självlokaliseringen är ett omfattande jobb och hör hemma i ett separat arbete.

7.5 Vidare utvärdering i verklig miljö

Simuleringsmiljön är utmärkt för att testa grundläggande egenskaper hos de olika kartdelningsmetoderna beskrivna i rapporten. Det finns dock flera

egenskaper för robotarna som inte är möjliga att återge på ett bra sätt i en simulerad miljö. Perceptionen är en av dem. I simulatorn introduceras osäkerhet i perceptionen genom att positioner för objekt ges en slumpmässig risk att vara felaktiga. I den verkliga miljön finns det dock ingen slump. Från kameran erhålls en bild av den faktiska omgivningen och det är sedan robotens uppgift att göra en tolkning av verkligheten utifrån denna bild. I den verkliga miljön finns möjligheten att en robot helt missklassificerar ett objekt. I den simulerade miljön inträffar detta inte. Även detta är ett fenomen som skulle kunna simuleras genom införandet av slump. Det är dock i det här fallet väldigt svårt att avgöra vad som vore en rimlig slump. Vi har därför valt att avstå från denna lösning. Andra fenomen som är svåra att återge i en simulerad miljö är robotarnas rörelse (fyrbent gång) och yttre fysikaliska faktorer (friktion, ljusförhållanden etc.).

Det är svårt att veta hur de fenomen som inte går att simulera påverkar resultatet för metoderna, och det skulle därför vara värdefullt att testa och utvärdera metoderna ytterligare på det riktiga robotarna. Problemet är att avgöra hur bra en metod kartlägger objekt i den verkliga SLRL-miljön. I simuleringsmiljön är det inga problem. Här finns de riktiga positionerna för objekten att jämföra positionsuppskattningar med. Denna information finns inte för de verkliga robotarna i SLRL-miljön. Troligtvis kan detta problem lösas på något annat vis, alternativt genom att man gör sådana mätningar manuellt.

Kapitel 8

Slutsats

Kooperativ kartläggning av dynamiska objekt är en problemdomän som är relativt outforskad. Denna rapport har redovisat tre metoden av olika beräkningsmässig komplexitet som behandlar problemet. Samtliga tre metoder har i simulerade tester visat sig ha positiva egenskaper gällande precision i kartläggning. Två av dessa metoder har visat sig ha särdeles bra kartläggningsegenskaper och dessutom goda beräkningsmässiga egenskaper.

De oskarpa positionslådorna har användts med goda resultat även i den verkliga SLRL-miljön i RoboCup 2002. Metodens egenskaper har i hög grad visat sig vara tillämpliga i den snabbt föränderliga miljön i SLRL. På ett adekvat sätt adresserar metoden de utmaningar som ställs för SLRL, där rörelseinformation är ytterst osäker, observationer är osäkra och beräkningar måste göras i realtid. Fördelarna med metoden är att den har låg beräkningsmässig kostnad och hanterar osäker information på ett robust sätt.

Litteraturförteckning

[1] RoboCups hemsida
http://www.robocup.org
(Verifierad 2003-03-15)

[2] Team Swedens hemsida
http://www.aass.oru.se/Agora/RoboCup
(Verifierad 2003-03-15)

[3] Thinking Caps hemsida
http://aass.oru.se/ asaffio/Software/TC/
(Verifierad 2003-03-15)

[4] Aibos hemsida
http://www.aibo.com
(Verifierad 2003-03-15)

[5] Open-Rs hemsida
http://www.aibo.com/openr/
(Verifierad 2003-03-15)

[6] P. Buschka, A. Saffiotti och Z. Wasik. *Fuzzy landmark-Based Localization for a Legged Robot.* Proc. of the IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS) pp. 1205-1210. Takamatsu, Japan, 2000.

[7] A. Saffiotti och L.P. Wesley. *Perception based selflocalization using fuzzy locations* In M. van Lambalgen L. Dorst and F. Voorbraak, editors, Reasoning with Uncertainty in Robotics, pages 368–385. LNAI 1093, Springer, 1996.

[8] W. Burgard, D. Fox, D. Hennig, och T. Schmidt. *Estimating the absolute position of a mobile robot using position probability grids.* In Proc. of AAAI, pages 896-901, 1996.

[9] B. Hengst, B. Ibbotson, P. Pham och C. Sammut. *Omnifirectional locomotion for quadruped robots.* Birk, Coradeschi, Tadokoro (eds), RoboCup 2001, SpringerVerlag, 2002.

[10] Z. Wasik och A. Saffioti. *Robust color segmentation for the RoboCup domain.* IEEE Int. Conf on Pattern Recognition (ICPR), 2002.

[11] A. Saffioti and K. LeBlanc. *Active perceptual anchoring of robot behavior in a dynamic environment.* IEEE Int. Conf Robotics and Automation (ICRA), 2000.

[12] A. Johansson och A. Saffiotti. *Using the Electric Field Approach inte the RoboCup Domain.* Birk, Coradeschi, Tadokoro (eds) RoboCup 2001, Springer-Verlag, 2002.

[13] Russel, J. Stuart och P. Norvig. *Artifical Intelligence: A Modern Approach.* Prentice-Hall, 1995.

[14] Gal A. Kaminka, Pedro U. Lima, Raul Rojas (Eds.) *The 2002 International RoboCup Symposium*, 2002.

[15] Vassilis Varveropoulos. *Robot Localization and Map Construction Using Sonar Data*, 2001.

[16] Takehisa Yairi, Kosuke Hirama and Koichi Hori. *Fast and Simple Topological Map Construction Based on Cooccurrence Frequency of Landmark Observation*, 2001.

[17] Ioannis M. Rekleitis, Gregory Dudek och Evangelos E. Milios. *Multi-Robot Collaboration for Robust Exploration*, 2000.

Bilaga A

Terminologi

Algoritm. En uppsättning bestämda regler eller följd av instruktioner för att lösa ett visst problem i ett bestämt antal steg. En algoritm kan t.ex. vara en matematisk formel eller en logisk struktur.

Bandbredd. (Överföringskapacitet) Mått på hur stor mängd data som kan överföras genom ett nätverk vid ett givet tillfälle .

Brute force. En lösningsmetod som genererar mängden av alla tänkbara lösningar och från denna mängd väljer den bästa.

Odometri. Information om hur en robots position förändras baserat på robotens rörelser.

Oskarp logik. En utbyggnad av vanlig (Boolsk) logik i avsikt att hantera koncept som är delvis sanna, dvs värden som ligger mellan ”helt sant” och ”helt falskt”.

TSP. (Handelsresandeproblemet) Problemet att besöka ett antal städer i en ordning som minimerar den totala sträckan för resan. Varje stad ska besökas precis en gång och den handelsresande skall återvända till startpunkten.

Bilaga B

RoboSimulatorn

RoboSimulatorn är en fri programvara (open source). Programmet är utvecklat för att köras under UNIX/Linux. Källkoden samt de exekverbara filerna finns att ladda hem från *http://ai.cs.lth.se*.

För att kompilera källkoden i UNIX/Linux krävs en C++ kompilator (gcc), samt att grafikbiblioteket SDL (http://www.libsdl.org) är installerat.

Aktuell version av källkoden är v1.0 (2003-03-24).

B.1 Manual för RoboSimulatorn

För att starta en enkel körning av RoboSimulatorn skrivs:

```
start -c [antal] -s [tid]
```

Parametern -c anger hur många klienter som skall aktiveras. Den andra parametern -s är till för statistikföring och anger i sekunder hur länge programmet skall köras. Om denna parameter utelämnas eller sätts till 0 så kör programmet tills dess att det avslutas av användaren. Ingen statistik lagras då till fil. Om statistikparametern är satt, visas ingen grafik.

För en detaljerad beskrivning av alla parametrar i start skrivs:

```
start -help
```

då visas texten:

```
usage: start [options]
-c #nbrClients     start clients (max 4)
-s #statTime       time to run stat program
-k                 terminate program
-help              this message
```

Det går också att starta server- respektive klientprogram manuellt. Servern startas genom att skriva:

```
server -c [antal] -g
```

där parametern -c anger hur många klienter som kommer att begära uppkoppling till servern. Parametern -g anger att det grafiska gränssnittet skall aktiveras. För en detaljerad beskrivning av alla parameterar till serverprogrammet skrivs:

```
server -help
```

Då visas texten:

```
usage: server [options]
-c #nbrClients      number of clients (max 4)
-g                  start graphic interface
-k                  terminate program
-help               this message
```

Klienter startas genom att skriva:

```
client -p [port] -s [tid] -g
```

Parametern -p anger vilken port klienten skall använda för att lyssna på kommunikation från andra klienter. Parametern -s anger att statistikföring skall köras en viss tid. Som ovan så gäller att om denna parameter utelämnas eller sätts till 0 så kör programmet tills dess att det avslutas av användaren. Parametern -g anger att grafiken skall aktiveras. För en detaljerad beskrivning av alla parametrar till klientprogrammet skrivs:

```
client -help
```

Då visas texten:

```
usage: client [options]
-p \#portNbr        port number (10001 - 10004)
-s \#statTime       time to run stat program
-g                  start graphic interface
-k                  terminate program
-help               this message
```

Bilaga C

Tabeller och diagram

Jämförelse av metoder: 200 cm (normal syn)

Metod	Maxvärde	Medelvärde	Standardavvikelse
Utan	442/299	89/122	107/54
Metod 1	212/413	31/93	26/44
Metod 2	241/221	33/100	29/31
Metod 3	595/386	76/126	102/52

Tabell C.1: Maxvärde, Medelvärde och standardavvikelse för 10000 uppskattningar av bollens samt spelares positioner. Värdena i tabellen avser avståndet mellan den uppskattade och den faktiska positionen. I tabellen redovisas resultat för boll/spelare.

Jämförelse av metoder: syn 130 cm

Metod	Maxvärde	Medelvärde	Standardavvikelse
Utan	513/452	200/206	113/85
Metod 1	433/461	111/165	98/81
Metod 2	403/290	145/161	104/46
Metod 3	504/273	160/156	94/46

Tabell C.2: Maxvärde, Medelvärde och standardavvikelse för 10000 uppskattningar av bollens samt spelares positioner. Värdena i tabellen avser avståndet mellan den uppskattade och den faktiska positionen. I tabellen redovisas resultat för boll/spelare.

Jämförelse av metoder: syn 550 cm

Metod	Maxvärde	Medelvärde	Standardavvikelse
Utan	334/277	42/100	53/48
Metod 1	152/165	24/68	14/39
Metod 2	160/171	26/44	16/29
Metod 3	293/234	29/52	26/39

Tabell C.3: Maxvärde, Medelvärde och standardavvikelse för 10000 uppskattningar av bollens samt spelares positioner. Värdena i tabellen avser avståndet mellan den uppskattade och den faktiska positionen. I tabellen redovisas resultat för boll/spelare.

Jämförelse av metoder: 25% brus

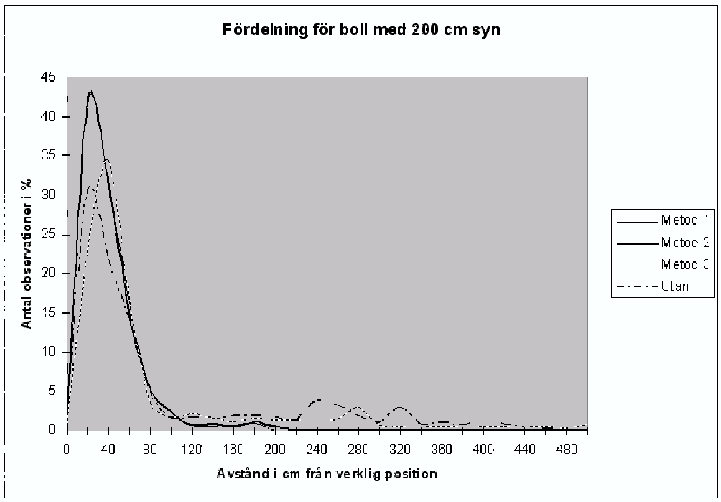
Metod	Maxvärde	Medelvärde	Standardavvikelse
Utan	466/381	78/115	100/68
Metod 1	195/413	37/115	29/50
Metod 2	217/226	42/96	31/37
Metod 3	695/328	90/120	103/58

Tabell C.4: Maxvärde, Medelvärde och standardavvikelse för 10000 uppskattningar av bollens samt spelares positioner. Värdena i tabellen avser avståndet mellan den uppskattade och den faktiska positionen. I tabellen redovisas resultat för boll/spelare.

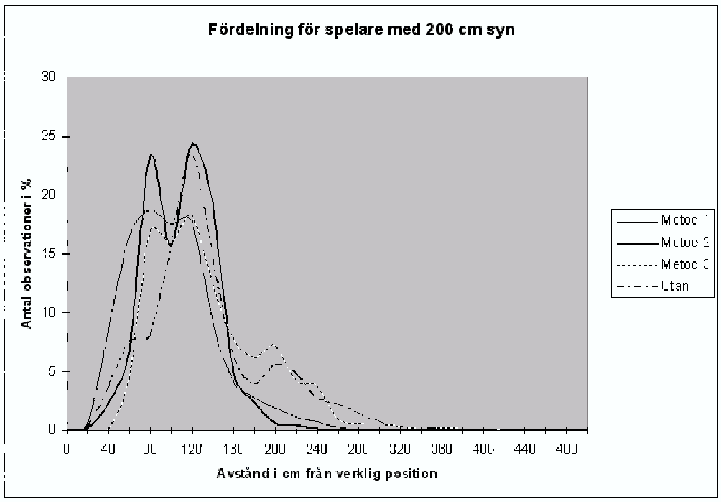
Jämförelse av metoder: 50% brus

Metod	Maxvärde	Medelvärde	Standardavvikelse
Utan	519/411	143/133	133/93
Metod 1	418/405	52/106	40/39
Metod 2	295/329	61/111	44/37
Metod 3	649/375	124/143	106/53

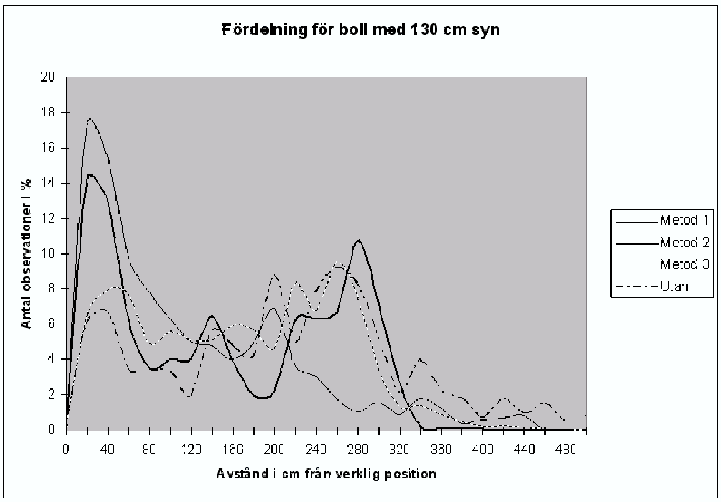
Tabell C.5: Maxvärde, Medelvärde och standardavvikelse för 10000 uppskattningar av bollens samt spelares positioner. Värdena i tabellen avser avståndet mellan den uppskattade och den faktiska positionen. I tabellen redovisas resultat för boll/spelare.



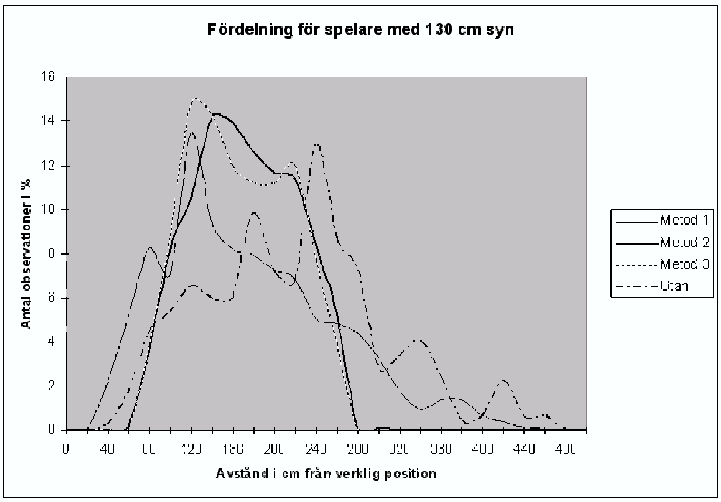
Figur C.1: Diagram över fördelning för boll, med avseende på avstånd från korrekt position. Värdena i tabellen avser det procentuella antalet observationer inom den givna felmarginalen. I testerna har roboten ett synfält på 200 cm.



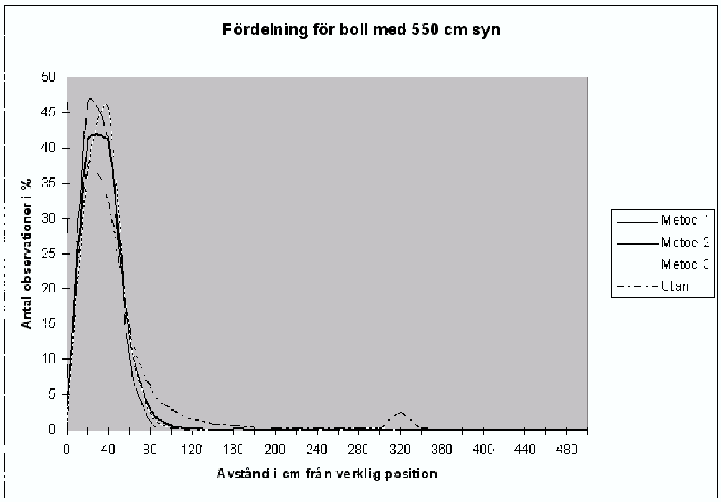
Figur C.2: Diagram över fördelning för spelare, med avseende på avstånd från korrekt position. Värdena i tabellen avser det procentuella antalet observationer inom den givna felmarginalen. I testerna har roboten ett synfält på 200 cm.



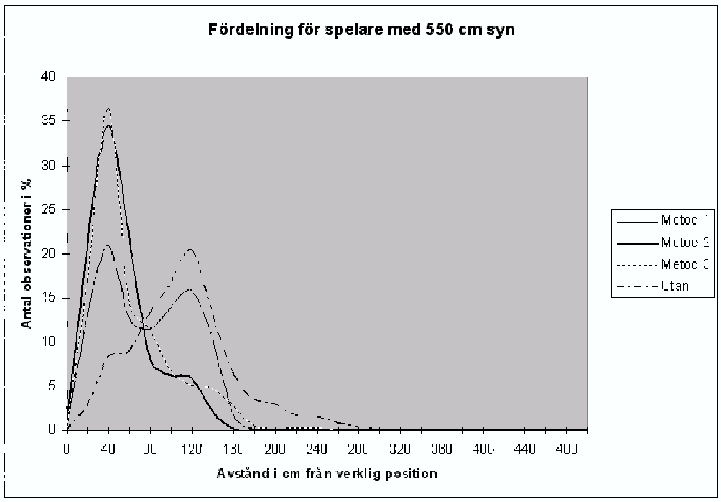
Figur C.3: Diagram över fördelning för boll, med avseende på avstånd från korrekt position. Värdena i tabellen avser det procentuella antalet observationer inom den givna felmarginalen. I testerna har roboten ett synfält på 130 cm.



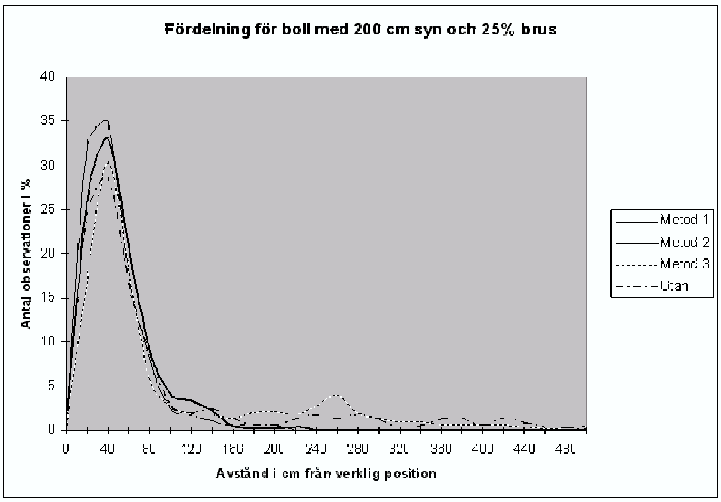
Figur C.4: Diagram över fördelning för spelare, med avseende på avstånd från korrekt position. Värdena i tabellen avser det procentuella antalet observationer inom den givna felmarginalen. I testerna har roboten ett synfält på 130 cm.



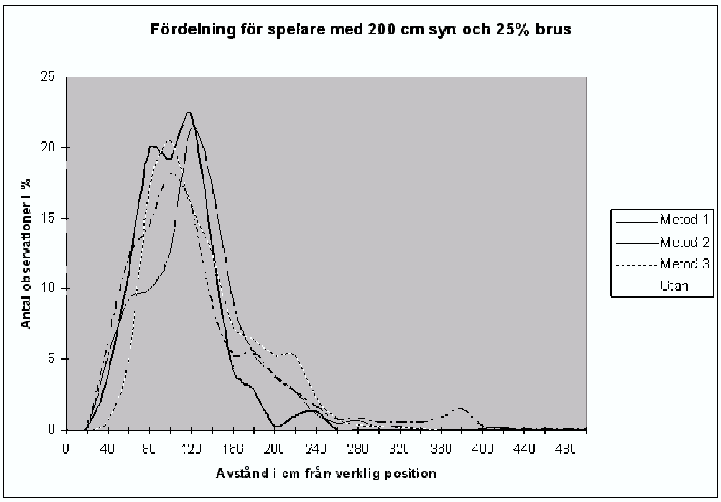
Figur C.5: Diagram över fördelning för boll, med avseende på avstånd från korrekt position. Värdena i tabellen avser det procentuella antalet observationer inom den givna felmarginalen. I testerna har roboten ett synfält på 550 cm.



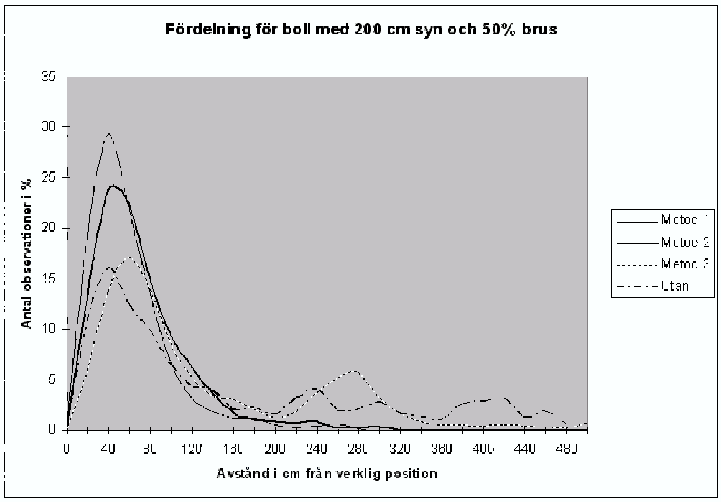
Figur C.6: Diagram över fördelning för spelare, med avseende på avstånd från korrekt position. Värdena i tabellen avser det procentuella antalet observationer inom den givna felmarginalen. I testerna har roboten ett synfält på 550 cm.



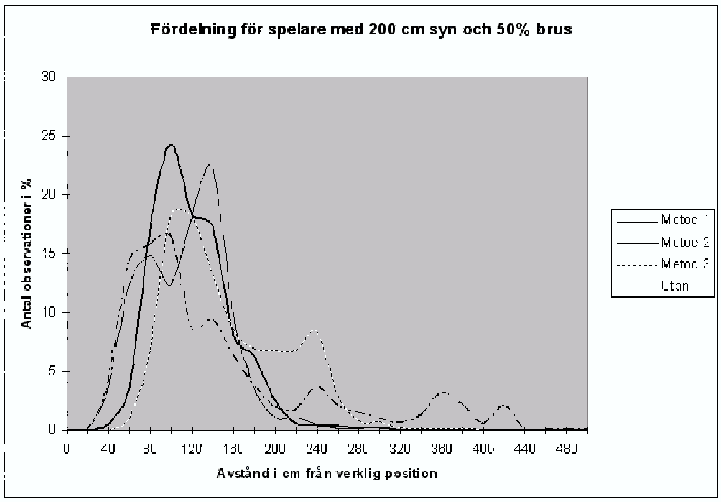
Figur C.7: Diagram över fördelning för boll, med avseende på avstånd från korrekt position. Värdena i tabellen avser det procentuella antalet observationer inom den givna felmarginalen. I testerna förekommer 25% brus.



Figur C.8: Diagram över fördelning för spelare, med avseende på avstånd från korrekt position. Värdena i tabellen avser det procentuella antalet observationer inom den givna felmarginalen. I testerna förekommer 25% brus.



Figur C.9: Diagram över fördelning för boll, med avseende på avstånd från korrekt position. Värdena i tabellen avser det procentuella antalet observationer inom den givna felmarginalen. I testerna förekommer 50% brus.



Figur C.10: Diagram över fördelning för spelare, med avseende på avstånd från korrekt position. Värdena i tabellen avser det procentuella antalet observationer inom den givna felmarginalen. I testerna förekommer 50% brus.