Från myror till samarbetande robotar

Henrik Öhman
Examensarbete vid Institutionen för datavetenskap
Lunds Universitet

31 augusti 2001

Sammanfattning

I den här uppsatsen presenteras det arbete som utförts för att få robotar av typen Khepera att genomföra kollektiva transporter. Problemet som robotarna ställdes inför var att få en låda i rörelse. Lådan var konstruerad så att det krävdes minst två robotar, som samtidigt knuffade från samma håll, för att få den i rörelse. Detta problem skulle lösas på ett sätt som var inspirerat av hur arbetsmyror förflyttar föda till stacken. Algoritmen som konstruerades med hjälp av beteendeprogrammering, blev både enkel och resurssnål. Resultatet visar att problemet går att lösa på ett effektivt sätt samtidigt som förbättringar är möjliga genom att göra robotarnas beteende ännu mer lik myrors.

Abstract

This report presents the work behind making Khepera robots transport an object collektivly. The problem the robots had to solve was making a box move. The box was constructed so that two robots had to push the box in the same direction at the same time to make it move. To solve this problem inspiration was gathered from ants and the way they transport food to the anthill. The algorithm which was constructed with behavior programming, was not only simple but also used minimal resources. The result shows that the problem can be solved in an efficent way and improvements are possible by making the robots behave more like ants.

Innehåll

1 Inledning

En av de största utmaningarna inom robotiken är att programmera en grupp robotar som tillsammans klarar av att interagera med en oförutsägbar omgivning. Ett tillvägagångssätt för att åstadkomma detta är att inspireras av vanliga arbetsmyror. Arbetsmyror som transporterar föda till stacken använder sig av ett antal grundläggande mekanismer. När alla dessa mekanismer används av ett stort antal myror samtidigt blir gruppens effektivitet större än summan av individernas effektivitet. Myrorna samarbetar alltså, troligen utan att vara medvetna om det själva, på ett mycket raffinerat och samtidigt enkelt sätt.
Med utgångspunkt för arbetsmyrors sätt att arbeta har jag programmerat ett antal Khepera [7] robotar till att leta upp och förflytta en stor låda. Lådan är konstruerad så att det krävs åtminstone kraften från två robotar som knuffar på samma sida för att den ska röra sig. Den här implementationen har sedan prövats i grupper av två, tre och fyra robotar.
Denna rapport är uppdelad som följer:

- Nästa kapitel handlar om kollektiv intelligens och kollektiv transport i naturen, och då främst hos myror. Här förklaras de grundläggande mekanismerna till hur den enskilda individen agerar och hur dess agerande gynnar gruppen.
- I kapitel 3 förklaras hur Kheperaroboten är uppbyggd och hur man går tillväga för att programmera den.
- Kapitel 4 beskriver experimentet och hur det har utförts. Här redovisas även de resultat som erhölls samt förslag på förbättringar och andra intressanta experiment som kan utföras inom samma genre.

2 Kollektiv intelligens hos sociala insekter

Kollektiva beteenden har studerats hos ett antal arter av insekter. Myror, termiter och vissa arter av bin lever alla i sociala samhällen där uppgifter utförs på ett sätt som i förhållande till individernas begränsade intelligens är mycket effektivt [2].
Ett insektsamhälles intelligens kan ses i två olika perspektiv, dels det kollektiva perspektivet där alla invånare i samhället är medverkande och dels ur den enskilda individens perspektiv. Många forskare tror att nivån av kollektiv intelligens är högre än summan av individernas intelligens [9]. Men detta kan vara svårt att tro för den som någon gång studerat en myrstack: den enskilda individen genomför en tillsynes slumpartad irrfärd mellan olika delar

av stacken. Men faktum kvarstår, samhället i stort lyckas ändå skapa byggnadsverk i form av sina stackar som är väl anpassade till de olika behov och krav som finns. Hur går detta egentligen ihop?
Ett förslag på lösning till denna problematik grundar sig på att den enskilda individens beteende är mycket mindre slumpmässig än vad det vid första ögonkastet tycks vara. Myrorna kommunicerar hela tiden med varandra. Kommunikationen sker dock inte direkt, utan indirekt genom doftspår, så kallade feromoner. Viss indirekt kommunikation sker också genom att myrorna påverkar sin omgivning. Ett barr som t.ex. ska förflyttas kan inte flyttas om någon annan myra redan tagit barret. Detta blir då en indirekt signal till myran att leta reda på ett annat barr.
Ett annat förslag är att tillåta en viss del slumpmässighet i myrornas beteende [10]. Vissa myror kommer då att vara sysselsatta med för samhället helt irrelevanta aktiviteter men statistiskt sett så kommer ändå den största delen av tiden ägnas åt aktiviteter som är gynnsamma för samhället.
Genom studier av sociala insekter kan vissa grundläggande mekanismer som är nödvändiga för ett effektivt kollektivt beteende härledas. Dessa mekanismer kan användas som grundstenarna i robotars beteende vid implementation av kollektiv intelligens och transport.

- Den första mekanismen innebär att samtliga individer strävar efter att nå ett gemensamt mål. Genom att använda ett väldefinerat mål och samtidigt undvika att individerna motverkar varandra i sin iver att nå målet, kan grundläggande uppgifter som t.ex. att få en grupp att samlas på en förutbestämd plats, lösas på ett enkelt sätt.

- En följa-efter mekanism är en enkel mekanism som bl.a. används av myror som ska välja vilken väg de ska gå när det finns två olika att välja mellan. När myror vill att andra myror ska följa efter dem lämnar de spår av feromoner som efterföljande myror luktar sig fram till. Då den kortare av de två vägarna besöks av fler myror per tidsenhet blir feromonlukten såpass mycket starkare att till slut inga myror väljer den längre vägen. Resultatet blir att enbart ett fåtal myror behöver gå den längre vägen.

- Den tredje mekanismen är påverkan från omgivningen. En viss speciell typ av stimuli leder alltid till samma aktivitet. Vissa typer av krigsmyror påbörjar t.ex. alltid sina rövartåg på jakt efter föda i gryningen och på samma sätt är skymningen en signal att avbryta och ta sig tillbaka till stacken. Detta beteende kan även lätt implementeras hos robotar genom att låta en viss aktivitet utlösas av en förändring i omgivningen.

- Den fjärde mekanismen bygger på att en aktivitet endast utlöses om individen befinner sig i en grupp. Om man vill få en grupp robotar att skotta snö på ett effektivt sätt skulle man endast tillåta dem att ha plogen nedfälld i snön när de är tillsammans i en grupp. Då skulle snöskottningen samordnas och stora ytor skulle kunna bli snöfria snabbare än om varje robot ensam irrade omkring och skottade individuellt.

- Den sista mekanismen kräver någon form av mer direkt kommunikation. Man kan tänka sig en grupp av robotar vilka letar efter ett visst föremål. När en av robotarna hittar det eftersökta föremålet så skickar han en signal till de övriga i gruppen, som då avbryter sitt letande och övergår till någon annan aktivitet. Detta beteende har studerats hos bin som letar efter blommor. När en större tillgång av blommor har hittats, flyger biet tillbaka till bikupan och genomför en dans i riktning mot blommorna. Detta blir en signal till andra bin att avbryta det de gör och hjälpa till att samla in nektaren som finns i de funna blommorna.

2.1 Kollektiv transport

Många olika arter av myror klarar av att kollektivt samla in byten som är för stora för en ensam myra att tranportera. Vanligtvis hittar en enskild myra ett byte och försöker ensam transportera det hem till stacken. Om uppgiften skulle vara övermäktig, rekryterar han andra myror till sin hjälp. Skulle gruppen inte heller klara uppgiften rektyteras i vissa fall speciella arbetare med extra stora käkar. Dessa får dela upp bytet i mindre bitar, som sedan med lätthet kan förflyttas[3]. Denna metod är dock mycket tidskrävande och undviks om möjligt.
När den enskilda myran finner ett lämpligt byte genomför den följande:

- Försöker lyfta föremålet.

- Försöker dra föremålet.

- Byter angreppsvinkel utan att släppa taget och försöker igen.

- Byter position och börjar om.

- Hämtar hjälp av andra myror.

När den enskilda myran har hämtat några kamrater så ändras schemat en aning:

- Gruppen greppar föremålet.

- Individerna byter angreppsvinkel och position ända tills de lyckas få föremålet i rörelse.
- Bytet transporteras till stacken.
- Om förflyttningen av någon anledning stoppas under transporten, börjar myrorna ändra angreppsvinkel och position ända tills föremålet åter är i rörelse.

Denna algoritm har visat sig vara mycket effektiv. Som exempel har en grupp av cirka 100 arbetsmyror av arten *Pheidologeton diversus* studerats när de transporterar en 1.92 gram tung mask i en hastighet av 0.41 cm/s. Varje enskild arbetsmyra väger mellan 0.3 och 0.4 mg och kan ensam maximalt bära 5 gånger sin egen kroppsvikt i en hastighet av 1 cm/s. I detta fallet bar varje myra en last av ungefär 50 gånger sin egen kroppsvikt utan att förlora mer än 59% av hastigheten [2].

3 Kheperan

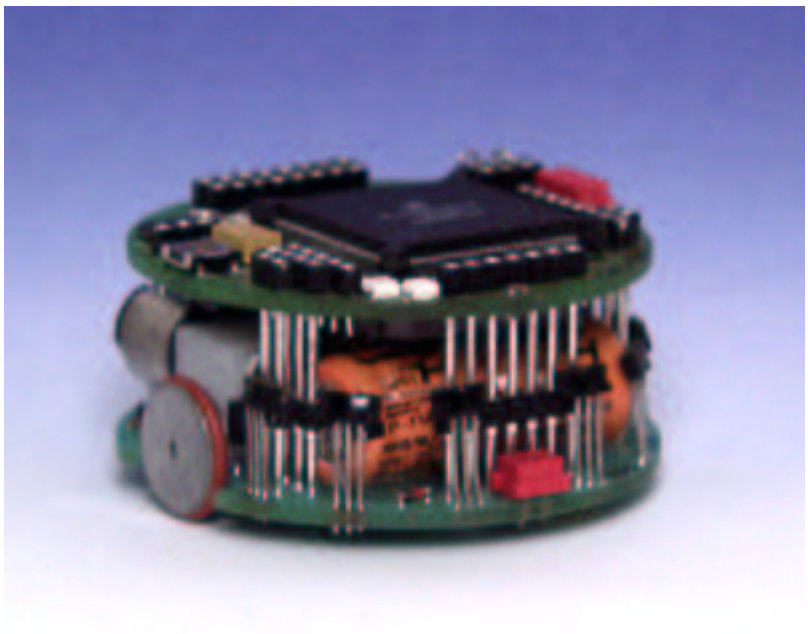
Robotarna som används i experimentet är av modellen Khepera. De utvecklades ursprungligen för att användas som ett forsknings och undervisningsverktyg av Laboratorie de Microinformatique i Lausanne. Numera distribueras och utvecklas de av företaget K-team [4].

3.1 Kheperans hårdvara

Kheperan är en liten robot, endast 55 mm i diameter, och i sitt basutförande 30 mm hög. Men styrkan sitter inte i storleken utan i dess kraftfulla beräkningskapacitet. Kheperan är utrustad med en Motorola 68331 processor som kan hantera upp till 32 parallellt exekverande processer.
För att förflytta sig använder sig Kheperan av 2 hjul som är kopplade till varsin likströmsmotor. På motorns drivaxel finns en räknare som uppdateras cirka 24 gånger per varv. Det blir ungefär 1 uppdatering per 0.08 mm körsträcka. Denna räknare är användbar till att bl.a. kontrollera Kheperans rörelser vid sväng.
Kheperans viktigaste kontakt med omvärlden är dess IR-sensorer. Den har 8 stycken som sitter placerade runtom roboten och var och en av dessa är både mottagare och sändare av infrarött ljus (IR).
Denna dubbla funktionalitet ger Kheperan förmågan att mäta två värden:

Ljusnivån i robotens direkta omgivning. Genom att endast använda IR mottagaren kan ljusnivån närmast Kheperan avläsas. Detta är mycket användbart om man t.ex. vill att Kheperan ska finna en ljuskälla. Denna mätning sker kontinuerligt var 20 ms och resulterar i ett värde mellan 500 och 0. Standardvärdet i ett mörkt rum är ungefär 450 och detta värde minskar ju ljusare omgivningen blir. Denna sensor kallas i fortsättningen för Kheperans ljussensor.

Reflekterande ljus. Genom att använda IR-sändaren kan föremål som befinner sig nära roboten (inom 5 cm) upptäckas. Mätningen av det reflekterade ljuset är, precis som för den allmänna ljusnivån, kontinuerlig och resultatet är differensen mellan det reflekterande ljuset och den allmänna ljusnivån. Detta gör att IR-sändarens värde är starkt beroende av ytmaterialet på föremålet, ett blankare föremål upptäcks tidigare än ett matt. Denna sensor kallas i fortsättningen för närhetssensor.



Figur 1: Khepera roboten. Bilden från [4].

3.2 Programmera Kheperan

För att med hjälp av egenutvecklad programvara styra Kheperan finns det två tillvägagångsätt: antingen exekveras programmet på en värddator eller

direkt i Kheperans processor. Den sista metoden ger en robot som är helt fysiskt autonom, men det finns stora utvecklingstekniska nackdelar som gör denna metod svårare att använda. Det är heller inte alltid nödvändigt att ha en helt fysiskt autonom robot.

3.3 Programmering med hjälp av en värddator

När man programmerar med hjälp av en värddator exekveras den kod man skrivit på värddatorn och under exekveringen skickas kommandon till Kheperan via en seriell kabel. Kommunikationen är uppbyggd kring ett enkelt ASCII protokoll där värddatorn skickar ett kommando till Kheperan följt av ett ny rad tecken. Kheperan skickar då antingen ett svar eller bara en konfirmation på att kommandot har blivit korrekt mottaget, allt beroende på kommandots natur. För en fullständig beskrivning av Kheperans seriella protokoll se [7]. Den här sortens programmering ställer inga stora krav på vare sig plattform eller programmeringsspråk. Det viktiga är att kommunikationen över datorns seriella port (/dev/ttyS0 i unix eller com1 i windows) stöds. Man behöver inte nödvändigtvis programmera allt från grunden själv, National Instruments (http://www.natinst.com) har producerat en utvecklingsmiljö kallad LabVIEW där man med hjälp av ett grafiskt gränssnitt kan programmera Kheperan. LabVIEW är ett snabbt och lätt sätt att komma underfund med hur Kheperans olika delar fungerar och hur de interagerar med omgivningen. Även mer avancerade program kan skrivas men det kräver specialkunskaper i hur LabVIEW fungerar.

3.4 Programmering för Kheperans processor

Det enda sättet att få Kheperan helt fysiskt autonom, d.v.s. slippa den seriella kabeln, är att skriva programmet direkt till Kheperans processor. Detta görs lämpligen genom att programmet skrivs i programmeringsspråket C på en värddator. Programmet kompileras sedan med den medföljande GNU C korskompilatorn [6]. Korskompilatorn kompilerar C koden men istället för att producera exekverbar kod som är körbar på värddatorn, produceras exekverbar kod till Kheperans processor. Denna måste sedan laddas ned till Kheperans RAM (random access memory) där den exekveras. Nedladdningen sker precis som tidigare via den seriella kabel men skillnaden nu är att kabeln kan tas bort så fort nedladdningen är färdig. Roboten är då helt autonom.
I Kheperans 256 kB stora ROM (read only memory) finns det ett antal bibliotek som hanterar den interna kommunikationen mellan programmet som

skall exekveras och Kheperans hårdvara. Dessa bibliotek kallas med ett samlingsnamn för Kheperans BIOS (Basic I/O System). BIOS är uppdelade i olika moduler där var och en är specialiserade på en viss del av hårdvaran. Här är en översikt över vad de olika modulerna i BIOS gör:

TIM hanterar systemets förmåga att hantera flera parallella processer, så kallad multitasking. 32 stycken processer kan maximalt köras samtidigt.

MOT hanterar alla de resurser som är kopplade till Kheperans motorer.

VAR hanterar diverse lågnivåresurser som t.ex. att läsa av kontaktbågarna, kontrollera ljusdioderna på Kheperans ovansida samt övervaka användarens användning av återställningsknapparna (från engelskans reset buttons).

SENS styr de 8 IR sensorerna.

STR är en stränghanterare. Den stödjer bl.a. formatering mellan ASCII och binära format och används vid seriell kommunikation för att översätta ASCII kommandon till processorinstruktioner.

SER hanterar alla kommunikation mellan Kheperan och en värddator via det seriella gränssnittet.

COM hanterar alla I/O kanaler i systemet.

MSG hanterar all kommunikation mellan processorn och olika tilläggsmoduler på Kheperan som t.ex. videokameran eller gripklon.

3.5 Ett enkelt exempel

Ett enkelt program för Kheperan kan se ut som följande:

```
#include <khesys.h>

void process_0() {
  while(1) {
    tim_suspend_task(500);
    mot_new_speed_2m(2,2);
    tim_suspend_task(500);
    mot_new_speed_2m(-2,-2);
  }
}
```

```
int main() {
  int32 status;
  static char process_name_0[] = "Process 0\n";

  var_reset();
  mot_reset();

  status = install_task(process_name_0, 800, process_0);
  if(status == -1) exit(0);
}
```

Det här programmet får Kheperan att röra sig framåt under en halv sekund, därefter bakåt en halv sekund sedan framåt igen och så vidare ända tills exekveringen avbryts. Här följer nu en kort förklaring av de centrala delarna av programmet:
För att anrop till Kheperans BIOS ska kunna göras måste filen khesys.h inkluderas:

```
#include <khesys.h>
```

De olika resetfunktionerna initierar de delar av BIOS som programmet kommer att använda.

```
var_reset();
mot_reset();
```

Anledningen till att inte TIM initieras beror på att den är så grundläggande att den initieras automatiskt av systemet när Kheperan startas. Delar som inte ska användas (i det här fallet t.ex. SENS) behöver ej initieras.
Processer startas genom ett anrop till BIOS som ser ut så här:

```
status = install_task(process_name_0, 800, process_0);
```

De olika parametrarna betyder:

process_name_0 en sträng som är associerad med den startande processen, i detta fallet:

```
static char process_name_0[] = "Process 0\n";
```

800 det antal bytes som den nya processen ska få på stacken. Stacken är den plats i minnet där bland annat lokala variabler lagras. Om den angivna minnesstorleken överskrids kan mycket svårförklarliga fel inträffa.

process_0 en adress till den kod som ska exekveras i processen. Om koden, som i vårt fall, innehåller en evig loop avbryts aldrig processen, i annat fall avslutas den när koden exekverat färdigt.

Processen i exemplet gör två stycken olika anrop till BIOS:

```
tim_suspend_task(500);
```

Ett anrop till BIOS:s TIM modul där en förfrågan görs om att låta den aktuella processen vila i 500 ms.

```
mot_new_speed_2m(2,2);
```

Här sätts hastigheten på Kheperans båda motorer. Den första parametern anger den högra motorn och den andra den vänstra. För en fullständig referens av de anrop till BIOS som finns, se [1].
Programmering för Kheperans ska ses som programmering av ett vanligt realtidssystem, där flera processer exekverar parallellt. Det är därför viktigt att skydda gemensamma resurser vid läsning och skrivning, exempel på en sådan resurs kan vara ljus- och närhetssensorenas värden. Man ska också tänka på att Kheperan har relativt små minnesresurser, om möjligt ska man undvika att använda för många lokala varibler och använda minnesrutinerna `malloc` och `free` flitigt.
En varning är på sin plats: Programmering för Kheperans processor är mycket komplicerat på grund av brist på respons från roboten. Det är alltså svårt att veta vad som egentligen pågår inuti roboten och därför rekommenderar jag att man inte börjar att utveckla direkt för Kheperans processor utan tar sig tid att lära känna roboten via det seriella gränssnittet. När man bemästrar detta är det relativt enkelt att skriva om sitt program så att det blir exekverbart direkt på Kheperan.

4 Experimentet

Till mitt förfogande vid experimentet stod fyra robotar av typen Khepera, beskrivna ovan. Arbetet inleddes med att studera tillgänglig litteratur om robotarnas funktion och andra egenskaper. I förberedelsearbetetet ingick även att skriva en enkel version av den kommande implementationen för det seriella gränssnittet. Detta skulle visa sig vara nyttigt då det gav en mer djupgående kännedom om Kheperans möjligheter och begränsningar.

4.1 Problemet

Målet med experimentet var att få ett antal Kheperarobotar att tillsammans förflytta en låda. Lådan var konstruerad så att det krävdes två eller fler robotar som samtidigt knuffade på samma sida för att få den i rörelse. Det här problemet kan utan några stora svårigheter lösas av två robotar som med hjälp av direkt kommunikation ställer sig på samma sida och knuffar åt samma håll. Men en sådan lösning skulle bli mycket mer komplicerad om vikten på lådan ökades så att det krävdes fler robotar. Det jag ville konstruera var en grupp av robotar som var flexibla nog att klara förändringar i omgivningen samtidigt som komplexiteten hos robotarna inte skulle öka om antalet robotar ökar. Lösningen blev att låta sig inspireras av hur arbetsmyror tillsammans hjälps åt att samla föda till myrsamhället. Målet med experimentet var att skapa en decentraliserad grupp av robotar som utan direkt kommunikation klarar av att förflytta ett givet föremål.
För att komma igång krävdes det att ett antal grundläggande problem löstes:

1. Robotarna var tvungna att kunna skilja på lådan som skulle förflyttas och andra hinder som skulle kunna komma i dess väg.

2. Väggarna runt övningsbanan borde undvikas för att inte onödiga kollisioner skulle inträffa.

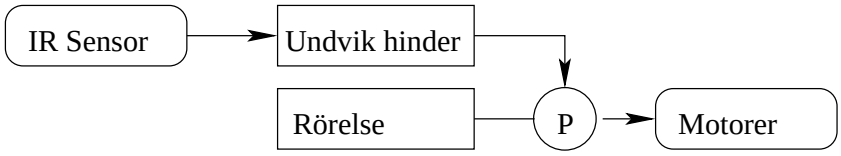
3. Robotarna skulle också undvika att krocka med varandra. Dels skulle detta minska effektiviteten och dels ville jag inte riskera att någon av robotarna skulle skadas.

Dessa problem löstes genom att en lampa monterades inuti lådan som skulle förflyttas. Detta gjorde att robotarna med hjälp av sina ljussensorer kunde se skillnad på föremål som avger ljus repektive föremål som inte avger ljus. För att skillnaden i ljusstyrka mellan ett lysande föremål och ett icke lysande skulle bli tillräckligt stor så krävdes att experimentet genomfördes i ett mörklagt rum.

4.2 Programmeringsmetodik

Traditionellt har robotprogrammerare använt sig av en programmeringsmetodik som går ut på att först skapa en intern uppfattning av omvärlden med hjälp av insignaler från robotens alla sensorer. Därefter planeras ett antal åtgärder som roboten ska utföra för att kunna uppnå ett förutbestämt mål och till sist utförs dessa åtgärder. Denna metodik var för beräknings-

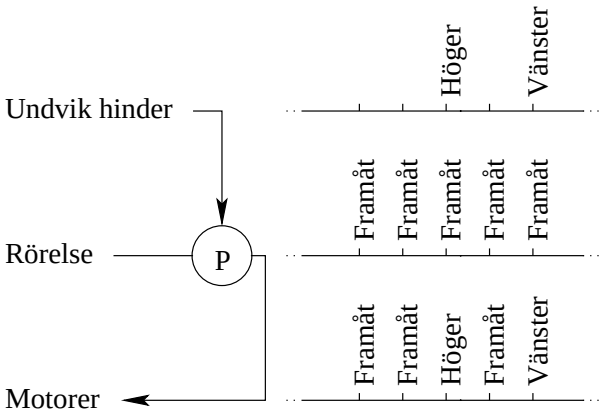
och utrymmeskrävande för Kheperans relativt små minnesresurser. Metoden som istället användes en programmeringsmetodik hämtad från ett arbete av Professor Rodney Brooks vid MIT Artificial Intelligence Laboratory, kallad *beteendeprogrammering* eller *beteendekontroll* [5]. I beteendeprogrammering har man ett antal kontrollsystem (eller beteenden) som var och ett är kopplade till robotens olika sensorer. När en sensor utlöses exekveras det kontrollsystem som är kopplat till den sensorn. Skulle flera sensorer utlösas samtidigt, används ett prioriteringssystem som avgör vilket kontrollsystem som ska få exekvera.
Ett enkelt exempel på en algoritm som undviker alla former av hinder med beteendeprogrammering skulle behöva två kontrollsystem. Dels ett som hela tiden manar roboten till rörelse och dels ett som utlöses så fort roboten känner av att något främmande föremål befinner sig i dess väg. När detta inträffar byter roboten färdriktning och en kollision undviks.



Figur 2: Översikt av algoritm i beteendeprogrammering. En robot programmerad med denna algoritm undviker att kollidera med ev. hinder. Bilden efter [5].

Resultatet vid en exekvering av ovanstående blir att beteendet som manar roboten i rörelse exekveras hela tiden eftersom det inte kontrolleras av någon sensor. Men skulle robotens närhetssensorer upptäcka ett hinder, exekveras det beteende som undviker hinder, eftersom det har högre prioritet. Detta beteende beräknar då, utifrån data från närhetssensorerna, vilken åtgärd som är lämplig för motorerna att göra. Därefter skickas denna åtgärd via prioritetssystemet till motorerna som utför åtgärden som i det här fallet blir att svänga roboten bort från hindret.

Den största fördelen med beteendeprogrammering är att programmeringen inte blir mer komplicerad av att fler beteenden läggs till. Inte heller prestandan påverkas i någon större grad negativt av att antalet processer ökar, detta tack vare Kheperans kraftfulla processor. Men man måste vara medveten om att det maximal antalet processer är 32 och att detta antal ej kan överskridas.



Figur 3: Översikt av prioriteringssystemets arbete vid exekvering av figur 1. Bilden efter [5].

4.3 Implementation

I implementationen av kollektiv transport användes två av de grundläggande mekanismerna från kapitel 2:

1. Robotarna ska ha ett gemensamt mål för sina aktiviter. Detta mål ska nås utan att robotarna påverkar varandra i någon större utsträckning.

2. Robotarna ska kunna anpassa sitt agerande utifrån förändringar i omgivningen.

Med dessa mekanismer som grund konstruerade jag fem olika beteenden som var kopplade till robotens sensorer och andra av omgivningen påverkbara faktorer. Beteenden är här angivna i prioritetsordning.

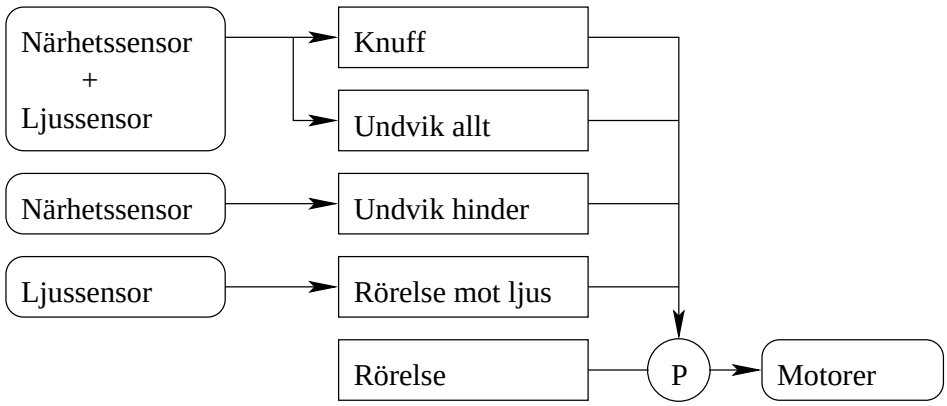
1. Knuffbeteende. Detta beteende exekveras då Kheperan står framför ett lysande föremål (lådan). När det inträffar försöker roboten flytta lådan genom att knuffa på den. Skulle denna manöver lyckas fortsätter den att knuffa tills lådan når närmaste vägg och därmed inte kan förflyttas ytterliggare.

2. Undvik allt. Detta beteende gör att Kheperan både undviker hinder av någon form men även att den undviker att röra sig mot en stark ljuskälla.

3. Undvik hinder. Skulle Kheperan hamna framför ett hinder undviker detta beteendet att en kollision inträffar och styr istället roboten åt ett annat håll.

4. Rörelse mot ljus. Om Kheperan upptäcker en ljuskälla som är avsevärt starkare än omgivningen ska den sträva efter att röra sig mot denna ljuskälla.

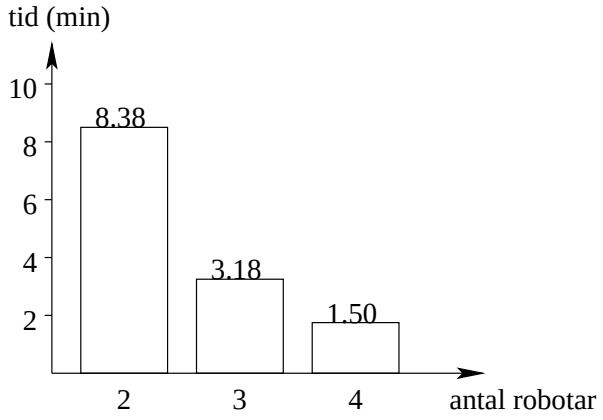
5. Rörelsebeteende. Om ingenting annat gäller ska roboten hela tiden röra sig framåt över övningsområdet.



Figur 4: Översikt av algoritmen som användes vid experimentet.

Tillsammans resulterade detta i att den enskilda roboten rörde sig framåt utan att krocka med väggarna i övningsområdet. När Kheperans ljussensorer upptäckte ljuset från lådan styrdes roboten mot denna tills en av lådans sidor upptäcktes av närhetssensorerna. I detta läget uppfylldes alla villkor som behövdes för att roboten skulle försöka förflytta föremålet som den stod intill. Knuffbeteendet aktiverades. Därefter kunde två saker inträffa, antingen fick roboten lådan i rörelse och då fortsatte roboten att knuffa lådan ända tills något hinder kom i dess väg, eller så orkade roboten inte få tillräcklig fart på lådan eftersom den var för tung eller redan låg mot en vägg. När Kheperan upptäckte att lådan inte längre gick att förflytta från den aktuella angrepspunkten skulle roboten, precis som arbetsmyror gör, byta angrepspunkt. I experimentet liknades detta vid att förflytta sig runt lådan och knuffa igen från ett annat håll. Detta genomfördes genom att beteendet som undvek både lådan och övriga hinder exekverades under en slumpmässigt vald tidsperiod (vanligen mellan två och tio sekunder). Då hade roboten med stor sannorlikhet förflyttat sig till en annan del av övningsområdet där den kunde återuppta sökandet efter lådan.
Skalbarheten hos en grupp av robotar programmerade på ovanstående sätt blir otroligt stor. Det behövs inga förändringar i algoritmen eller i hårdvaran om man skulle öka antalet robotar från fyra till 20.

Experimentet utfördes med två, tre och fyra robotar. För varje antal gjordes fem försök. Tiden mättes från det att robotarna placerades på övningsområdet tills det att lådan kom i rörelse.



Figur 5: Tid i sekunder för att få lådan i rörelse vid olika antal robotar. Tidsangivelserna är ett genomsnitt av fem försök.

För att få fram en mer pålitlig genomsnittstid borde åtminstone 10 försök med varje uppsättning av robotar genomförts. Vid varje försök krävdes det att robotarnas batterier återuppladdades. Att ladda en Khepera tog mellan 30 och 45 minuter och jag hade bara tillgång till två batteriladdare. Tidsbrist gjorde att fler försök inte var genomförbara.

4.4 Resultat och erfarenheter

Att vid implementation av robotars beteende snegla på hur sociala insekter löser uppgifter har i många fall varit en bra idé [2]. I det experiment som genomförts har det visat sig vara extra fördelaktigt, inte bara för att experimentet i fråga redan från början är hämtat från en företeelse som sker regelbundet hos olika arter av myror, utan även för att algoritmen i fråga är såpass väl anpassningsbar till olika förändringar i de yttre förutsättningarna. Implementationen blir enkel samtidigt som den är resurssnål, vilken är en stor fördel då Kheperans begränsade minnesresurser beaktas.
Implementationen som gjordes var dock inte helt felfri och förbättringar är möjliga på många olika ställen. Den förändring som skulle gett bäst resultat var att implementera ett mer aktivt gruppbeteende hos Kheperorna. Som det var nu agerade varje robot på egen hand och undvek hela tiden att komma alltför nära någon av de andra robotarna. Detta gjorde att det var mest i

undantagsfall som robotarna knuffade lådan i samma riktning, vilken var ett krav för att lösa uppgiften. Om man istället hade implementerat ett beteende som fick Kheperorna att röra sig som en grupp, skulle de flesta robotarna vara samlade på en begränsad yta under en stor del av tiden och därmed skulle chansen bli betydligt större att tillräckligt många robotar knuffar lådan i samma riktning samtidigt. Effektiviteten hade ökat avsevärt. Men denna förbättring är svår att implementera. Hur ska Kheperan se skillnad på en robot och ett annat icke lysande objekt, som t.ex. en vägg? Närhetssensornerna i kombination med ljussensorerna ger robotarna information om sin omgivning, men frågan är om den är tillräcklig. Kanske måste man använda sig av videokameran, som finns som tillbehör, för att ytterliggare förstärka robotens synförmåga.
En ren felaktighet i implementationen som jag aldrig lyckades finna lösningen på var att roboten i vissa fall hamnade i ett så kallat dödläge (eng. deadlock). Kheperan snurrade runt på stället tills något i robotens omgivning gav utslag på dess närhetssensorer.
De erfarenheter som jag har fått genom att arbeta med Kheperorna har gett mig många uppslag på andra experiment som skulle vara möjliga att genomföra med hjälp av Kheperor och beteenden hämtade från myror. En naturlig utveckling av det experiment som jag gjorde vore att få robotarna att placera lådan på en viss plats. Det experimentet skulle då bli ännu mer likt myrornas tranport av föda till stacken och myrornas effektivitet skulle på ett ännu mer tydligt sätt kunna åskådliggöras.

Referenser

[1] Edoardo Franzi. *Khepera BIOS 5.0 Reference Manual.* 1998.

[2] Marco Dorigo. *Ant Algorithms and Swarm Intelligence: An Introduction.* ECAI Tutorial Notes, 2000.

[3] Edoardo Bonabeu, Marco Dorigo, G. Theralaz. *From Natural to Artificial Swarm Intelligence.* York: Oxford Univerity Press, 1999.

[4] http://www.k-team.com/

[5] Joseph Jones, Anita Flynn, Bruce Seiger. *Mobile Robots - Inspiration to Implementation, second edition* A K Peters, 1999.

[6] K-team. *Khepera - GNU C based cross-compiler for the Khepera robot.* 1999.

[7] K-team. *Khepera - User manual.* 1998.

[8] Max Karlström. *Experiment med orientering och navigering av robotar.* Examensarbete vid Institutionen för datavetenskap, Lunds Universitet, 2000.

[9] Ronald Kube, Hong Zhang. *Collective Robotic Intelligens.* 1992.

[10] Robert Pallbo. *Myror i huvudet - Om kollektiv intelligens.* 1999.