Sammanfattning

Vi har arbetat med att konstruera en robotsimulator som skall kunna användas för att simulera flera olika typer av robotar som agerar med varandra och med sin omgivning. Simulatorn kan användas för att utveckla och testa olika inlärningsalgoritmer. Programmet skall också kunna styra robotar parallellt som de simuleras men det är inte implementerat än. Inledningsvis kan bara en robotmodell av typen Khepera hanteras men systemet kan lätt utökas med flera robottyper. Simulatorprogrammet har blivit lyckat och fungerar över förväntningarna.

Abstract

We have worked with the construction of a robot simulator. The programs main purpose is to simulate a number of robots and their interaction with each other and with the simulated environment. The simulator can be used to develop and test different learning algorithms. The program is also meant to control a real robot in parallel with the simulated robot but this feature is not implemented yet. First only one robot model, the Khepera, can be simulated but the system can easily be expanded with further robots. The simulation program is a success and fulfils our expectations.

Förord

Detta projekt blev möjligt tack vare det faktum att gruppen för artificiell intelligens vid Lunds Universitet skall starta en ny kurs som handlar om multiagentsystem och robotar. Till kursen behövs en robotsimulator och en utredning om de frågor som arbete med en robotsimulator ger upphov till.

Jag och Henrik Lernmark meddelade vårt intresse att vi gärna ville utveckla vårt examensarbete med inriktning på artificiell intelligens och hittade ramarna för vårt projekt i utvecklingsarbetet för den nya kursen.

Jag skulle vilja tacka min sammarbetspartner Henrik för hans återhållsamhet och hans tolerans mot mina vilda idéer. Utan hans hjälp och fokus hade detta arbete ej bli klart. Jag skulle också vilja tacka Johan Lindén, doktorand och kursledare, som först väckte mitt intresse för artificiell intelligens och som sedan har varit en språngbräda för nya idéer.

Sist men inte minst skulle jag vilja nämna Eric Astor, Robert Pallbo och Jacek Malec alla verksamma inom gruppen för artificiell intelligens i Lund, för utan dem skulle det inte finnas något projekt över huvud taget. Varje gång man var där och visade de senaste framstegen så var de positiva och sprudlade av ideér, många som säkert skulle kunna realiseras, givet ett stort antal superdatorer och en stab av programmerare.

Innehållsförteckning

1 INLEDNING 5

1.1 SIMULATORER 5

1.2 VAD ÄR EN ROBOT? 5

1.3 SVÅRIGHETERNA MED ATT SIMULERA EN ROBOT 6

1.4 MÅLET MED PROJEKTET 6

1.5 NÅGRA ANVÄNDNINGSOMRÅDEN 6

1.5.1 Robotlokaliseringsproblemet 6

1.5.2 Urskiljningsproblemet 7

1.5.3 Kartlägga omgivningen 7

1.5.4 Styrning 7

2 KHEPERA 8

2.1 SPECIFIKATIONER 8

2.2 KOMMUNIKATIONSALTERNATIV 9

2.3 KOMMUNIKATIONSNIVÅ 10

2.4 KHEPERANS SPRÅK 11

3 BESKRIVNING AV ARBETET 12

3.1 PROGRAMBESKRIVNING 13

3.1.1 Geometriska klasser 13

3.1.2 Skärmobjekt 13

3.2 EFFEKTORER OCH SENSORER 14

3.2.1 Effektorer 14

3.2.2 Tvåhjulsmotor 15

3.2.3 Lampa 15

3.2.4 Sensorer 15

3.2.5 Avståndssensor 16

3.2.6 Fotosensor 16

3.2.7 Närhetssensor 17

3.2.8 Kompass 18

3.2.9 GPS 18

3.2.10 Kartograf 18

4 SAMMANFATTNING 19

4.1 VAD SOM ÄR GJORT 19

4.2 FRAMTIDEN 19

5 REFERENSER 20

APPENDIX A – ORDLISTA 21

APPENDIX B – KHEPERANS KOMMANDON 22

APPENDIX C – KLASSER 25

APPENDIX D - ANVÄNDARMANUAL 28

Figurförteckning

Figur 1: Bild av Kheperaroboten _____ 8
Figur 2: Kompilerat program skickas till Kheperan_____ 9
Figur 3: Styrprogram kommunicerar med Kheperan _____ 9
Figur 4: Styrprogram och kommunikationsprogram laddas ner till Kheperan ___ 10
Figur 5: Skiss över kommunikationsstruktur för simulatorprogrammet _____ 12
Figur 6: Bild av användargränssnittet på serversidan _____ 13
Figur 7: Principbild av informationsflödet i en effektor _____ 14
Figur 8: Tvåhjulsmotorn och dess datafält _____ 15
Figur 9: Skiss av en lampa och dess datafält _____ 15
Figur 10: Ett sensorfält och de parametrar som styr dess utbredning _____ 16
Figur 11: Graf över ljussensorns mätvärden_____ 17
Figur 12: Graf över Närhetssensorns signalstyrka _____ 17
Figur 13: Bild över hur en kartografs värden kan representera en karta _____ 17

1 Inledning

Projektet går ut på att skapa en simulator med en sådan detaljrikedom att den skall kunna användas för att simulera riktiga robotar. Det är också meningen att man skall kunna representera många olika robotar. Varje robot skall kunna fatta beslut själv, utifrån vad den lär sig och vad den får för intryck (percepts). Fler än en robot skall kunna simuleras, och för att få datorkapaciteten att räcka så skall varje robot kunna köras på en separat dator och kommunicera med servern som berättar vad roboten ser.

Som språk valde vi Java för dess objektorienterade synsätt och dess flexibilitet. Vi behövde grafik, trådar och nätverksstöd. Allt detta finns färdigt i Java.

1.1 Simulatorer

Att uföra en simulering av ett verkligt förlopp med hjälp av en dator har varit en möjlighet sedan den första datorn skapades. Vad är det då för mening med att simulera en robot och dess omgivning? Svaret på det är att robotar är dyra och innan man kan arbeta mot en riktig robot så kan man arbeta mot en simulator. Detta gör att man kan ta in fler studenter på en kurs som handlar om robotar, robotstyrning och inlärningsalgoritmer och vid en sådan kurs kan en simulator räcka långt för att få en första förståelse av problematiken kring styrningen av en riktig robot. Vår simulator har också den fördelen att när man har utvecklat ett styrprogram för roboten i simulatorn kan man direkt styra den riktiga roboten utan extra arbete[1].

1.2 Vad är en robot?

Det som skiljer en robot från vilken annan mekanisk konstruktion som helst är dess förmåga att fatta egna beslut. Till sin hjälp har den sensorer som läser av omgivningen och effektorer som verkställer robotens beslut.

Vi har inte kommit så långt än att vi behöver ta hänsyn till de robotlagar som den skönlitteräre författaren Asimov [1] presenterat i sin litteratur.

1. En robot får inte skada en människa eller genom underlåtenhet skada en människa.
2. En robot måste lyda en människa så länge detta inte strider mot den första lagen.
3. En robot måste skydda sin egna existens så länge detta inte bryter mot första eller andra lagen.

Robotars nuvarande problemställning är snarare: var är jag[2], var ska jag[3], vad ser jag[4] och hur skall jag kombinera dessa till något vettigt.

[1] Detta är inte testat än, då vi inte har haft en riktig robot att tillgå.
[2] Se kapitel 1.5.1 Robotlokaliseringsproblemet och 1.5.3 Kartlägga omgivningen
[3] Se kapitel 1.5.4 Styrning
[4] Se kapitel 1.5.2 Urskiljningsproblemet

1.3 Svårigheterna med att simulera en robot

Det finns många svårigheter med att göra en robotsimulator. Varje objekt i den simulerade världen skall ha egenskaper som ett verkligt objekt har som t.ex. hastighet, riktning, materialtyp, tyngd och så vidare. Dessutom måste man simulera varje typ av sensor på roboten och konstruera artificiellt brus för dessa. Sedan måste robotens beslut kunna hanteras inte bara för en typ av robot utan för godtyckliga robotar. Detta leder till många val när det gäller vilken nivå som kommandona till roboten skall ligga på. Det gäller att göra kommunikationen generell så att det protokoll som skall användas fungerar både för en robot med två hjul och en annan robot med fyra ben.

För oss var det svårast att hantera kollisioner för kartobjekt på ett sätt som överenstämde med verkligheten. Verkligheten kräver mycket för att simuleras men det är inte där vi har lagt arbetets tyngdpunkt, även om vi har gjort mycket för att få det rätt.

1.4 Målet med Projektet

Målet är att skapa en simulator som kan användas i utvecklingen av inlärningsalgoritmer och styrprogram för riktiga robotar och som sedan kan kopplas upp direkt mot den riktiga roboten. Vi har gjort en del val för hur vi representerar världen i simulationen. Vi har en platt kvadratisk värld med en sida på 4m. Underlaget är jämt och vi har inget väder eller andra yttre faktorer som kan inverka på simulationen. Alla objekt representeras av tvådimensionella polygoner som kallas kartobjekt. Allting presenteras sett uppifrån med ett grafiskt gränssnitt.

1.5 Några användningsområden

Simulatorn kan användas för att simulera olika styrproblem för robotar men framför allt kan man simulera olika inlärningsalgoritmer. Det går utmärkt att använda simulatorn för att se hur flera robotar interagerar med varandra eller med sin omgivning. Här nedan tar jag upp några av de problem som man kan använda simulatorn till och problem som man bör ta upp inledningsvis för att lära sig hantera simulatorn.

1.5.1 Robotlokaliseringsproblemet

Ett väldigt känt problem är robotlokaliseringsproblemet som helt enkelt är att veta var roboten är. Problemet är att man kan lätt få ett allt större fel om man använder passiv referens[1] (dead reconing) speciellt om roboten kör in i något och bara står och slirar.

[1] Passiv referens är när man skall röra roboten från en punkt till en annan utan att få intryck från omgivningen.

1.5.2 Urskiljningsproblemet

Ett annat problem är att skilja på olika objekt. Först att se om någonting är ett rörligt objekt och sedan se skillnad på om det t.ex. är en boll eller en annan robot. Detta problem är av stor vikt om man skall simulera t.ex en robocup[4]. Man brukar fuska och ha bollen i en annan färg (givet att robotarna kan urskilja färger) eller göra bollen självlysande eller på något annat sätt särskilja bollen från alla andra objekt i omgivningen.

1.5.3 Kartlägga omgivningen

Att försöka kartlägga omgivningen är inte så lätt som det verkar, särskillt om det finns rörliga objekt. Kartläggning går in under robotlokaliseringsproblemet. Här kan man beräkna sannolikheter för de olika väggbitarna, se Markovprocesser [2]. Exempel på en uppgift är att åka igenom en labyrint på så kort tid som möjligt. Här ger Kheperaroboten hjälp då man kan ange en position den skall åka till och sedan sköter Kheperan om resten.

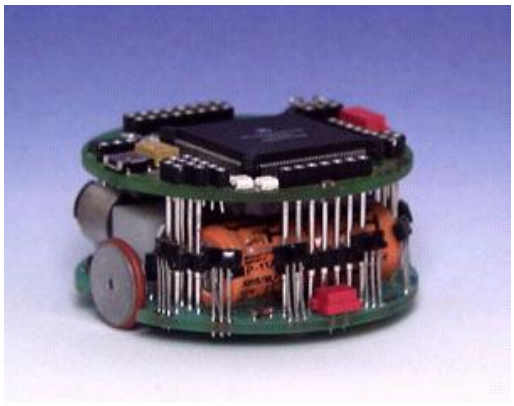
1.5.4 Styrning

När det gäller att planera robotens färdväg finns det många svårigheter, speciellt om omgivningen innehåller andra rörliga objekt. För att planera hur roboten ska röra sig använder man sig av rörelseplanering eller Robot Motion Planning[5]. Ett exempel på hur fel det kan bli är en enkel styrregel som säger att om du får en signal från de främre närhetssensorerna (se kapitel 3.2.7) så sväng åt det håll som har svagast signal. Denna regel verkar tillsynes göra att roboten kör omkring och undviker objekt, men i ett rektangulärt rum blev tyvärr resultatet att roboten sicksackade in i ett hörn och ställde sig där och slirade.

Som ett inledande moment i en kurs kan man göra en tournering, där studenterna får skapa styrprogram för roboten. Styrprogrammet skall köra runt roboten utan att krocka och roboten skall ha en viss minimihastighet. Studenterna kan då tävla mot varandra i simulatorn, där den som krockar sist får gå vidare och möta andra vinnare.

2 Khepera

Den robot som vi har implementerat är Khepera. Det är en miniatyrrobot som har den fördelen att den är liten, relativt billig, flyttbar och kan utökas med många olika funktioner. Den utvecklats av Labratoire de Microinformatique[3] i Schweiz.



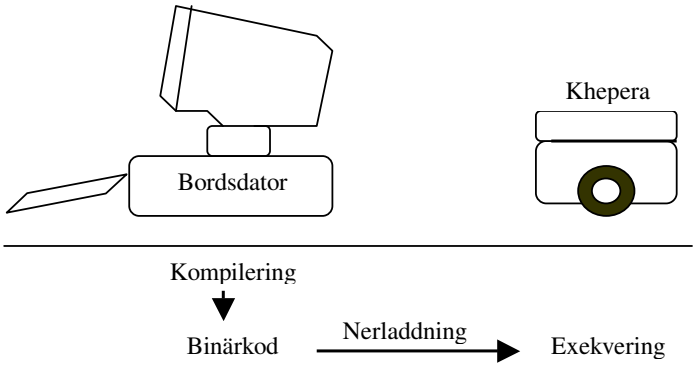
Figur 1: Bild av Kheperaroboten

2.1 Specifikationer

Roboten är bara 55mm i diameter och har åtta ljussensorer som den använder dels för att se ljuskällor och dels för att se om den kommer nära andra objekt. Hastigheten på de två hjulen kan sättas individuellt vilket ger både framdrivning och styrning på samma gång. Roboten har många utbyggnadsmöjligheter och ett relativt avancerat komunikationsprotokoll. Tack vare dess storlek kan man köra omkring roboten på ett bord eller annan plan yta där man kan ställa upp sin bana.

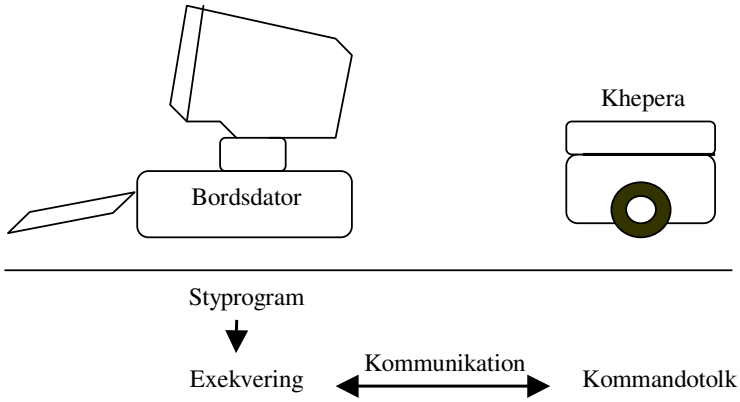
2.2 Kommunikationsalternativ

Det finns tre sätt att programmera och styra en Kheperarobot.



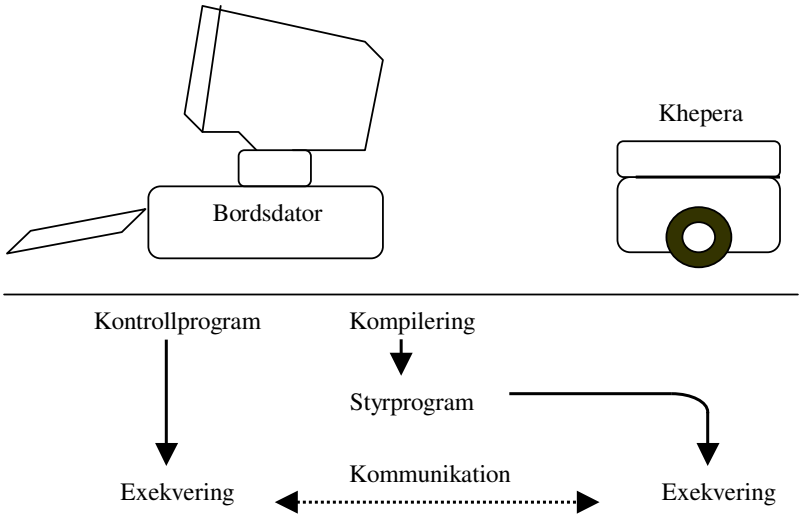
Figur 2: Kompilerat program skickas till Kheperan

Det första sättet ovan är att kompilera programmet för Kheperans motorola 68030 processor och flytta det färdiga programmet till roboten och sedan låta roboten köra självständigt. Detta sätt har den nackdelen att man inte får någon feedback när roboten kör så det blir svårt att göra felsökningar.



Figur 3: Styrprogram kommunicerar med Kheperan

Det andra sättet är att skicka kommandon till Kheperan och låta programmet som skall testas vara i bordsdatorn tillsammans med eventuell felsökningsalgoritmer. De (ASCII) kommandon som skickas till Kheperan tolkas av en inbyggd kommandotolk. Detta system är det vi använder.



Figur 4: Styrprogram och kommunikationsprogram laddas ner till Kheperan

Det tredje sättet att kommunicera är att skapa ett eget styrprogram för Kheperan inklusive ett kommunikationsprotokoll. Ladda ner detta till Kheperan och sedan låta Kheperan köra. Felsökningsprogram och övervakningsprogram kan då köras på bordsdatorn samtidigt som Kheperan följer sitt styrprogram. Detta är en kombination av de två tidigare metoderna.

2.3 Kommunikationsnivå

Den nivå på vilket kommunikationen sker sätter gränser för flexibiliteten men samtidigt fördelas arbetsbördan olika. Man kan antingen skicka ett kommando som tolkas av respektive robot som t.ex. Sätt hastighet=2 eller skicka mer detaljerade kommandon som inte behöver tolkas av roboten.

Vägning av fördelar och nackdelar beroende på val av kommunikationsnivå

Nivå	Generella kommandon	Detaljerade kommandon
Tid	Kan sträcka sig över längre tid	Utförs på kortast möjliga tid
Belastning	Kräver tolkning av roboten	Belastar nätverket
Språk	Samma kommandon för olika robotar	Specifika kommandon för varje typ av robot

Vi har valt att kombinera dessa tillvägagångsätt. Vi kan skicka både generella kommandon och detaljerade kommandon. Tolkningen från de generella till de detaljerade kommandona görs i den robotklass som hör till robottypen på

serversidan[1]. Vissa av robotens kommandon kan vara svåra att simulera och är för tillfället inte implementerade. Några av dessa handlar om att ladda ner styrprogram till roboten och kräver att man utökar simulatorn med en förmåga att tolka alla instruktionerna till en motorola 68030 processor. Denna typ av simulerad CPU behövs dock inte för att använda simulatorn till vad den är tänkt.

Vi har valt att huvudsakligen använda oss av de kommandon som specificeras av robottypen och att endast ha med robotoberoende kommandon för att kunna göra tidiga feltester under utvecklingsarbetet. De två robotoberoende kommandona vi har gjort är [**turn** grader] - där grader är antalet grader som den skall svänga nästa uppdatering och [**speed** v] - där v är hastigheten i bildpunkter per sekund.

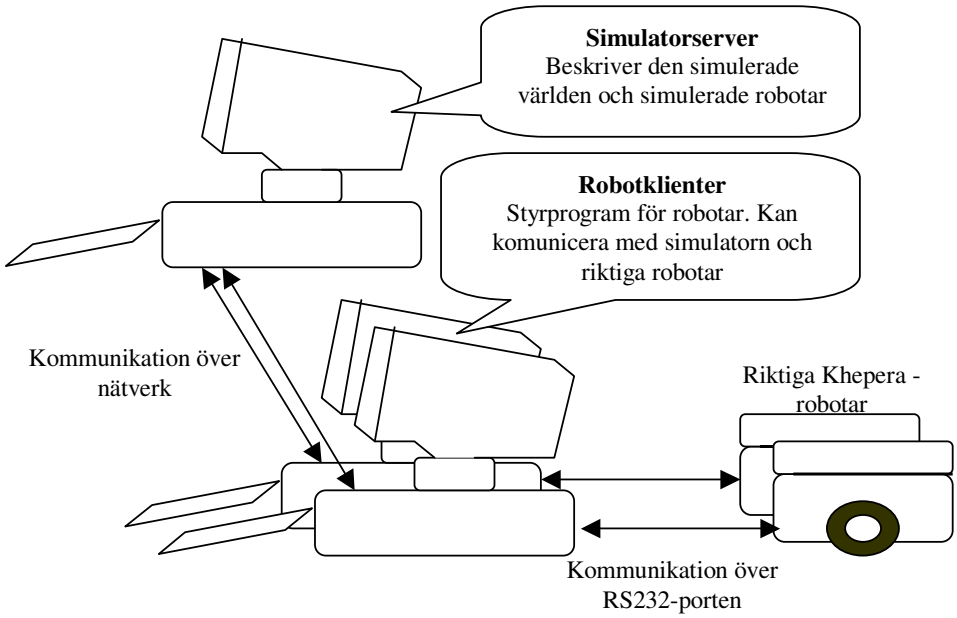
2.4 Kheperans Språk

Kheperan har en egen kommandotolk. Dess kommandon består av en versal bokstav följt av noll eller flera parametrar åtskiljda av kommatecken. Kheperan svara med samma bokstav i gemen form, när kommandot har tagits emot. En komplett lista av Kheperans kommandon ges i appendix B. Listan är ett utdrag från manualen till Kheperan och ges på Engelska.

[1] Man bör undvika att i styrprogrammet använda kommandon som kan användas i simulatorn men inte ingår i robotens språk.

3 Beskrivning av arbetet

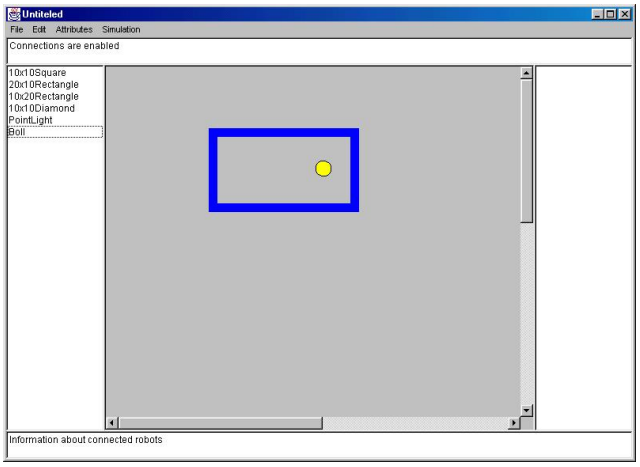
I vår simulator presenteras världen i en grafisk miljö uppifrån. Alla objekt på skärmen är kartobjekt. Varje kartobjekt har ett utseende som beskrivs av en polygon. De har också ett material som beskriver färg och friktion mot underlaget. Alla kartobjekt kan ges en hastighet eller en rotationsvektor. Dessutom kan man placera effektorer och sensorer på ett kartobjekt, se kapitel 3.2. Varje simulerad robot startas från en robotklient och kopplas upp mot simulatorservern över det lokala nätverket. Detta gör att vi kan utnyttja flera datorer för vår simulering. Här nedan ser ni en bild på hur simulatorprogrammets olika delar fungerar.



Figur 5: Skiss över kommunikationsstruktur för simulatorprogrammet

På serversidan presenteras simulationen. Här kan man ladda och spara olika banor där simuleringen sker och man kan placera ut olika kartobjekt på banan genom att dra och släppa dem där med musen. Figur fem visar en bild på hur det grafiska användargränssnittet ser ut. Man kan välja om man vill tillåta om robotar skall kunna ansluta sig eller inte och man kan markera robotar för att få reda på deras position, färg, riktning och hastighet. En mer detaljerad beskrivning av användargränssnittet och programmets användarmanual ges i appendix D.

Samma kommandon som skickas till simulatorn kan skickas till de riktiga Kheperaroboten. Nivån på kommandona, generella eller detaljerade, bestäms av robotens språk. Kheperan har till exempel ett kommando för att ange en position som den skall nå. Detta är ett generellt kommando som sträcker sig över en längre tid medan ett kommando som läser sensorvärden svarar så fort som möjligt.



Figur 6: Bild av användargränssnittet på serversidan

3.1 Programbeskrivning

Vi behövde ett antal grundklasser för att skapa grafiska objekt. Här ges en översikt av de olika klasserna. En sammanställning av klassernas alla datafält och metoder ges i appendix C.

3.1.1 Geometriska klasser

Punkt - Punkt med reella koordinater. Den kan translateras och jämföras med andra punkter.

Linje – Ett objekt med metoder för att flytta linjer, se om två linjer korsar varandra och rotera linjer.

Poly - Ett objekt som håller ett godtyckligt antal punkter vars *n* första ritas ut som en polygon. Alla punkter beskrivs av reella värden. En av metoderna (”overlap”) kollar om två polygoner har ett icke tomt snitt vilket behövs för att hantera kollisioner. De punkter som finns efter de *n* första kan användas för att beskriva speciella fästpunkter på polygonen, se *Sensorer* och *Effektorer*.

3.1.2 Skärmobjekt

Kartobjekt – Denna klass beskriver alla saker som kan ritas ut på skärmen. Alla kartobjekt har en polygon, se *Poly*, som beskriver dess utsträckning. Kartobjekt har även ett material, se *Material*, och en rörelse, se *Rörelse*. Gemensamt för alla kartobjekt är att de kan ritas ut och har metoder för att göra allt det som man kan göra med polygoner, material och rörelse. Kartobjekt är det man bygger upp sin *Arena* med och som klassen *Robot* ärver från.

Rörelse – Innehåller värden för rotationshastighet och hastighet. Polygonens riktning avgör färdriktningen. Metoden update() flyttar polygonen och beräknar ny hastighet.

Material – Materialet håller egenskaper som friktion mot underlag, färg, brus, vikt och om objektet är transparent.

Robot – En robot är ett kartobjekt som har sensorer och effektorer. Roboten har alla de egenskaper som ett kartobjekt har och kan läsa av sensorer och sätta värden till sina effektorer. Klassen robot har ingen egen intelligens utan styrs från klientsidan genom att skapa en ny klass som utökar klassen *Brain*. Med intelligens menas här adaptiv metod för att länka sensorvärden med effektorvärden.

Arena – Beskriver alla de kartobjekt och robotar som syns på skärmen och sköter om deras uppritning och hanterar händelsehantering. Arena är en komponent i användargränssnittet där simuleringen sker.

3.2 Effektorer och sensorer

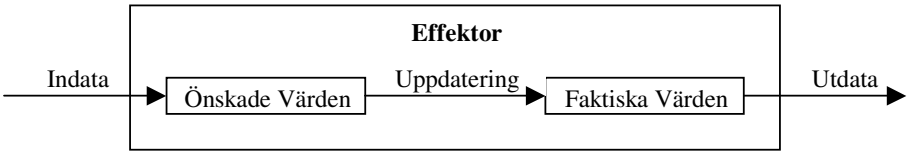
Som en genomgående strategi har vi baserat vårt projekt på idén att allt som definierar en robot kan delas upp i dels effektorer, och dels sensorer. Genom att hålla oss stringent till denna metodik har vi förenklat utvecklingen av programmet, förenklat för utbyggnad av nya robotar och framför allt gjort programmet mer generellt.

3.2.1 Effektorer

En effektor är ett verställande organ för objektet den sitter på. En effektor kan även påverka sin omgivning. Som exempel på effektor har vi motor, lampa och lysdiod. En effektor placeras på någon av ett kartobjekts fästpunkter som definieras av dess polygon[1].

En effektor kan ses som en vektor med önskade värden och en vektor med faktiska värden. När man konstruerar en effektor måste man i metoden update själv flytta de önskade värdena till de faktiska värdena för att ta hänsyn till tidsfördröjning, acceleration eller liknande tidsberoende aspekter.

Man kan utveckla många fler effektorer som; radiosändare, högtalare, griparmar eller olika framdrivningssystem.

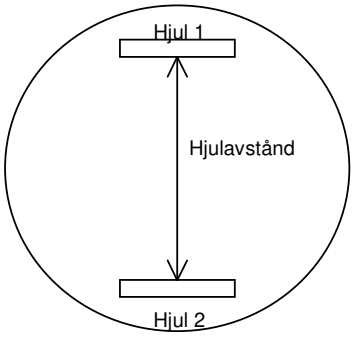


Figur 7: Principbild av informationsflödet i en effektor

[1] En polygon kan ha flera punkter än de som beskrivs av dess hörn. Dessa punkter är just till för att fästa effektorer och sesorer.

3.2.2 Tvåhjulsmotor

Tvåhjulsmotorn är en effektor som hanterar framdrivningen av objektet den sitter på. Den har sex datafält.



Datafält
0 Hastighet hjul 1
1 Hastighet hjul 2
2 Position x
3 Position y
4 Pulser hjul 1
5 Pulser hjul 2

Figur 8: Tvåhjulsmotorn och dess datafält

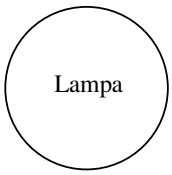
Tvåhjulsmotorn kan påverka två egenskaper på objektet den sitter på. Den kan ändra objektets hastighet och dess rotation. Dessa räknas ut från de första två datafälten d_0 och d_1, se nedan.

$$Hastighet = \frac{(d_0 + d_1)}{2}$$

$$Rotation = \arctan\left(\frac{hjulavst}{d_0 - d_1}\right)$$

3.2.3 Lampa

Lampa är en effektor som sitter på en robot. Lampan kan ha en färg och en styrka som bestäms av lampans enda datafält. Lampans styrka beskrivs av ett värde mellan 0 och 1.



Datafält
0 Ljusstyrka

Figur 9: Skiss av en lampa och dess datafält

Lampan är en enkel effektor som kan användas för att skilja vissa objekt från andra. T.ex kan man placera en lampa på en boll så underlättar man för spelarna att urskilja bollen från de andra spelarna.

3.2.4 Sensorer

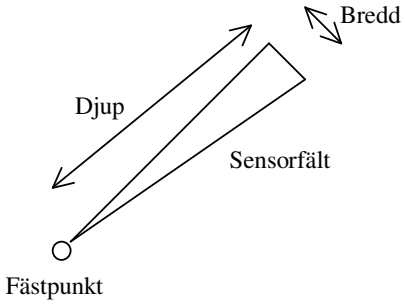
En sensor är något som läser av sin omgivning och ger en vektor med reella värden som resultat varje gång den anropas. Som exempel har vi avståndssensorn och ljussensorn.

En sensor placeras på ett kartobjekt på samma sätt som en effektor. Den definieras av ett cirkelsegment updelat i ett antal sensorfält.

Vi har tänkt att man i framtiden kan tänka sig vilja ha mikrofon, radiomottagare, tryckkännare eller LRF (Laser Range Finder).

3.2.5 Avståndssensor

Avståndssensorn, se klass **DistanceSensor**, används för att bestämma avståndet till det objekt som har någon del i sensorfältet, räknat från sensorns fästpunkt. Avståndet ges som ett tal mellan noll och ett, där noll är nära och ett betyder att inget är i sensorns synfält.



Figur 10: Ett sensorfält och de parametrar som styr dess utbreding

Genom att ändra på djup, bredd, antal fält och mängden brus så kan man skapa många olika typer av avståndssensorer[1]. Bredden ges som en vinkel i radianer.

3.2.6 Fotosensor

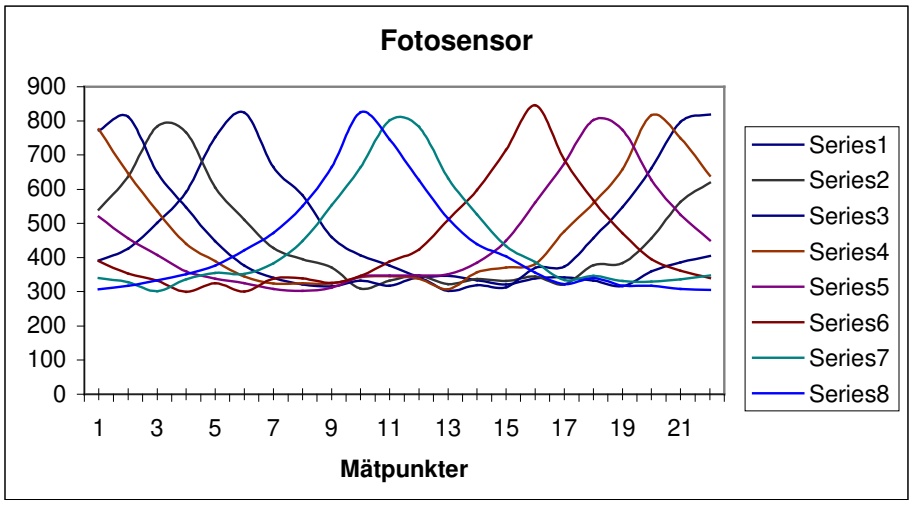
En fotosensor känner av ljuset från lampeffektorer och beräknar sitt sensorvärde utifrån lampornas avstånd och läge i förhållande till fotosensorns riktning. Fotosensorn halverar ljusstyrkan för varje objekt som finns mellan den och lampan så länge inte objektet är transparent. Objektet som lampan sitter på skymmer inte ljuset.

$$sensorvärd\ e = \sqrt{\frac{\sum_{\forall lampor}\left[\left(1 - ljusstyrka \times \frac{\Delta vinkel}{\pi} \times \prod_{\forall objekt\ _i_\ vägen \in synliga} 1/2\right)^2 \times \left(1 - \frac{avstånd}{sensordjup}\right)\right]^2}{antal\ _lampor}}$$

[1] Närhetssensorn på Kheperan mäter inte avståndet till objekten utan mängden reflekterat ljus. Man får inte en märkbar förändring på signalstyrkan förrän man är mellan en till sex cm från ett objekt.

Vid en första anblick kan formel ovan verka något komplicerad. Men det är en första approximering av addition av ljusstyrka, som är modifierad för att minska ljuset för de objekt som är i vägen. En mer avancerad formel behövs inte.

Detta är en av de mer krävande beräkningarna. Den är Ordo(skl), där s är antalet sensorer, k är antalet kartobjekt och l är antal lampor. Tack vare en del villkorssatser så sker bara uträkningen för de objekt som kan inverka på formeln.



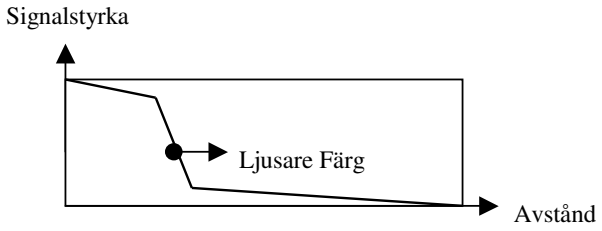
Figur 11: Graf över ljussensorns mätvärden

Figur 11 beskriver hur signalstyrkan varierar från åtta ljussensorer när roboten roterar ett helt varv (22 mätpunkter) ca. 150 bildpunkter från en lampa (dvs 6dm i verkligheten). Man ser helt klart att kurvorna överrensstämmer med en normalfördelning.

3.2.7 Närhetssensor

På kheperan sitter åtta närhetssensorer. De fungerar genom att mäta ljusmängden med och utan indirekt belysning. Skillnaden mellan de två mätningarna ger mängden reflekterat ljus. Denna typ av sensor reagerar på ett avstånd mellan en till sex cetimeter beroende på färg och yta på materialet. Vi har implementerat närhetssensorn genom att hitta det närmaste kartobjektet och sedan beroende på totala färgmänden och avståndet returnera ett värde som motsvarar mängden reflekterat ljus. Det innebär att vi förenklat beräkningarna från att summera ljusreflektionen från alla objekt till att göra en approximation av mängden reflekterat ljus från det närmaste objektet.

Figur 12 beskriver hur signalstyrkan varierar med avståndet och färgmängden hos det reflekterande objektet. Färgen påverkar avståndet där signalstyrkan börjar öka. Avståndet varierar med ca fem cm, från ett helt svart till ett helt vitt objekt.



Figur 12: Närhetssensorns signalstyrka

3.2.8 Kompass

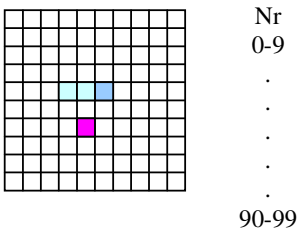
Denna sensor returnerar den riktning som objektet den sitter på har. Riktningen som returneras är helt oberoende av kompassens placering på objektet. Riktningen ges i radianer och varierar med ett brus på 0.01 radianer. Denna sensor är avsedd att placeras på en robot för nybörjare då det är en relativt svår uppgift att beräkna robotens riktning utifrån dess rörelser.

3.2.9 GPS

Denna sensor returnerar den position som centrumet på objektet den sitter på har. Positionen ges i bildpunkter och varierar med ett brus på en halv bildpunkt d.v.s. 2mm. Varje avläsning returnerar en x-koordinat och en y-koordinat. Sensorn är för närvarande inte placerad på någon robot men är avsedd en robot för nybörjare då positionering är ett komplex uppgift.

3.2.10 Kartograf

Denna sensor är fortfarande på experimentstadiet. Den returnerar en punktmatris n x n stor. Varje ruta täcker en lika stor del av världen sedd ovanifrån. Varje ruta har ett värde från 0 till 1 beroende på hur många objekt (hörnpunkter) som finns i rutan. Standardinställningen är en 10x10 matris men högre upplösning är möjlig. Användningen för denna sensor är att kunna få en absolut referens för hur värden ser ut. Denna typ av sensor kan ses som en signal från en stationär kamera i taket.



Figur 13: Bild av hur en kartografs värden kan representera en karta

4 Sammanfattning

4.1 Vad som är gjort

Uppgiften vi har arbetat med var att skapa ett simulatorprogram som dels kan användas för att utveckla olika styrprogram och inlärningsmetoder och dels kan användas för att styra en riktig robot. Varje robot skulle kunnas köras från en separat dator i ett lokalt nätverk och världen simuleras på en server. Vi har lyckats på många punkter. Det som fattas är en mer avancerad kollisionshantering som hanterar rotation och att räkna ut studsarna mer noggrant. Vi har inte heller implementerat komunikationen med en RS232-port för direktkommunikationen med Kheperan. Anledningen till det är att vi inte hade tillgång till en riktig Khepera under projektets gång. Av samma anledning så har vi inte testat att simuleringen av roboten överrensstämmer med styrning av den riktiga roboten. Men med tanke på arbetets omfattning är vi väldigt nöjda med vår insats.

4.2 Framtiden

Vad som behöver göras är att implementera fler sensorer och effektorer. Utveckla ännu en robot för att se att programmet har den flexibilitet som krävdes av handledarna. Testa programmet mot riktiga sensorer och riktiga robotar för att bekräfta funktionaliteten och testa att simuleringen överensstämmer med verkligheten.

Vi behöver också implementera den klass som skall komunicera med en RS232-port. Detta verkar inte vara speciellt svårt men vi har avvaktat då vi inte har någon hårdvara att testa på.

Gränssnittet kan utökas med fler olika kartobjekt och möjlighet att skala om och konstruera nya kartobjekt innifrån simulatorprogrammet. Framför allt skulle det vara bra att ha ett konstruktionsset för robotar där man kan placera ut fördefinierade effektorer och sensorer på robotkroppen (läs polygonen). Dessa egenkonstruerade robotar kunde sedan styras med ett språk som definierades av deras sensorer och effektorer.

För en mer åtkomlig produkt kan man byta det lokala nätverket mellan simulatorserver och robotklienter till det globala Internet genom att presentera servern som en applet och göra det möjligt att hämta hem och skapa ett robotklient program genom något javascript. Denna utbyggnad skulle dock kräva en omarbeting av gränssnittet och förmodligen sätta begränsningar på robotarnas komplexitet och sålunda programmets funktionalitet.

5 Referenser

[1] Död Robot av Isaak Asimov, Legenda Pocket, ISBN 91-582-1220-5

[2] Markov Processer Artificial Minds av Stan Franklin MIT Press A Bradford Book, ISBN 0-262-06178-3

[3] Labratoire de Microinformatique, 2000-01-01, http://lamiwww.epfl.ch/Khepera

[4] Robocup, 2000-01-20, http://www.robocup.org

[5] Robot Motion Planning av J.-C.Latombe och Kluwer, Academic Publishers, Boston, 1991

Appendix A – Ordlista

Applet	Javaklass som möjliggör att ett program kan köras i en webläsare
Effektor	Samlingsnamn på en apparat som uför något med omgivningen t.ex. motor eller gripklo
Khepera	Namn på liten robot som marknadsförs a Labratoire de Microinformatique
Klient	Dator i ett nätverk som arbetar mot en server
Passiv Referens	Eng. Dead Reconing; att hitta från en punkt till en annan utan att mäta omvärden
RS232	Standard för kommunikation över en 8-bitars seriell kabel
Sensor	Samlingsnamn för liten mätapparat som omvandlar någon aspekt av omgivningen till en mätbar kvantitet
Server	Värddator i ett nätverk. Servern förser klienter med tjänster
Trådar	Datorsammanhang; ett program som kan köra samtidigt som andra program. Ex: Om en robot styrs av ett program och man vill styra flera robotar kan varje robotprogram startas i en tråd.
Webläsare	Program som kan läsa och visa information från Internet.

Appendix B – Kheperans Kommandon

Lista av tillgängliga kommandon till Kheperan. Beskrivningen ges på engelska och är ett utdrag till Kheperans manual.

 # indicates CR (carriage return) or LF (line feed). ¶ indicates CR and LF.)

A Configure

Format of the command: A, Kp, Ki, Kd#
Format of the response: a¶
Effect: Set the proportional (Kp), integral (Ki) and derivative (Kd) parameters of the speed controller. At the reset, these parameters are set to standard values: Kp to 3800, Ki to 800, Kd to 100.

B Read software version

Format of the command: B#
Format of the response: b, version_of_BIOS, version_of_protocol¶
Effect: Give the version of the software present in the EPROM of the robot.

C Set a position to be reached

Format of the command: C,pos_left,pos_right#
Format of the response: c¶
Effect: Indicate to the wheel position controller an absolute position to be reached. The motion control perform the movement using the three control phases of a trapezoidal speed shape: an acceleration, a constant speed and a deceleration period. These phases are performed according to the parameters selected for the trapezoidal speed controller (command J). The maximum distance that can be given by this command is (2**23)-2 pulses that correspond to 670m. The unit is the pulse that corresponds to 0.08mm. The movement is done immediately after the command is sent. In the case another command is under execution (speed or position control) the last command replaces the precedent one. Any replacement transition follows acceleration and maximal speed constraints.

D Set speed

Format of the command: D, speed_motor_left, speed_motor_right#
Format of the response: d¶
Effect: Set the speed of the two motors. The unit is the pulse/10 ms that corresponds to 8 millimetres per second. The maximum speed is 127 pulses/10ms that correspond to 1m/s.

E Read speed

Format of the command: E#
Format of the response: e, speed_motor_left, speed_motor_right¶
Effect: Read the instantaneous speed of the two motors. The unit is the pulse/10 ms that corresponds to 8 millimetres per second.

F Configure the position PID controller

Format of the command: F,Kp,Ki,Kd#
Format of the response: f¶
Effect: Set the proportional (Kp), the integral (Ki) and the derivative (Kd) parameters of the position regulator. At the reset, these parameters are set to standard values: Kp to 3000, Ki to 20, Kd to 4000.

G Set position to the position counter

Format of the command: G, position_motor_left, position_motor_right#
Format of the response: g¶
Effect: Set the 32 bit position counter of the two motors. The unit is the pulse, that corresponds to 0,08 mm.

H Read position

Format of the command: H#
Format of the response: h, position_motor_left, position_motor_right¶
Effect: Read the 32 bit position counter of the two motors. The unit is the pulse, that corresponds to 0,08 mm.

I Read A/D input

Format of the command: I, channel_number#
Format of the response: i, analog_value¶
Effect: Read the 10 bit value corresponding to the channel_number analog input. The value 1024 corresponds to an analog value of 4,09 Volts.

J Configure the speed profile controller

Format of the command: J, max_speed_left, acc_left, max_speed_right, acc_right#
Format of the response: j¶
Effect: Set the speed and the acceleration for the trapezoidal speed shape of the position controller. The max_speed parameter indicates the maximal speed reached during the displacement. The unit for the speed is the pulse/10ms that corresponds to 8 mm/s. The unit for the acceleration is the ((pulse/256)/10 ms)/10 ms, that correspond to 3,125 mm/s2. At the reset, these parameters are set to standard values: max_speed to 20, acc to 64.

K Read the status of the motion controller

Format of the command: K#
Format of the response: k, T_left, M_left, E_left, T_right, M_right, E_right¶
Read the status of the motion controller. The status is given by three numbers for every motor: T (target), M (mode) and E (error). T=0 means that the robot is still on movement. T=1 means that the robot is on the target position. M=0 means that the motor control is in the speed mode. M=1 means that the control is in position mode. M=2 means that the control is in PWM mode. E indicates controller position or speed error.

L Change LED state

Format of the command: L, LED_number, action_number#
Format of the response: l¶
Effect: Perform an action on one of the two LEDs of the robot. Possible actions are: 0: turn OFF, 1: turn ON, 2: change status. The LED number 0 is the lateral one, the LED number 1 is the frontal one.

N Read proximity sensors

Format of the command: N#
Format of the response:
n,val_sens_left_90°,val_sens_left_45°,val_sens_left_10°,val_sens_right_10°,val_sens_right_45°,val_sens_right_90°,val_sens_back_right,val_sens_back_left¶
Effect: Read the 10 bit values of the 8 proximity sensors (section 2.1.6.2), from the front sensor situated at the left of the robot, turning clockwise to the back-left sensor.

O Read ambient light sensors

Format of the command: O#
Format of the response:
o,val_sens_left_90°,val_sens_left_45°,val_sens_left_10°,val_sens_right_10°,val_sens_right_45°,val_sens_right_90°,val_sens_back_right,val_sens_back_left¶
Effect: Read the 10 bit values of the 8 light sensors (section 2.1.6.1), from the front left sensor turning clockwise to the back-left sensor.

P Set PWM (pulse with modulation)

Format of the command: P, pwm_motor_left, pwm_motor_right#
Format of the response: p¶
Effect: Set the desired PWM amplitude (see "Motors and motor control" on page 6 for more details) on the two motors. The minimum PWM ratio is 0 (0%). The maximal forward ratio (100%) correspond to a value of 255. The maximal backwards ratio (100%) correspond to a value of -255.

T Send a message to an extension turret

Format of the command: T, turret_ID, command#
Format of the response: t, response¶
Effect: Send a command and return the response of the intelligent extension turret with turret_ID. The list of turrets connected and their ID can be requested with the tool "net". The command parameter takes the same format as a standard command, including an identification capital letter followed, if necessary, by numerical parameters separated by a comma and terminated by a line feed. The response takes the same format, starting with the same letter but in lower case, followed, if necessary, by numerical parameters separated by a comma and terminated by a line feed. The command and response format are specific for every module.

R Read a byte on the extension bus

Format of the command: R, relative_address#
Format of the response: r, data¶
Effect: Read the data byte available at the relative_address (0...63) of the extension bus.

W Write a byte on the extension bus

Format of the command: W, data, relative_address#
Format of the response: w¶
Effect: Write the data byte at the relative_address (0...63) of the extension bus.

Appendix C – Klasser

Beskrivning av de klassobjekt som ingår i vårt simulatorprogram.

Namn	Punkt	Linje	Poly
Data	double x,y	double x1,y1,x2,y2	double xp[], yp[] int number, length double x1,y1,x2,y2, direction
Konstruktorer	Punkt() Punkt(x,y) Punkt(punkt)	Linje() Linje(x1,y1,x2,y2) Linje(linje)	Poly(poly) Poly(x[],y[],number) Poly(x[],y[],number,dir)
Metoder	distance(punkt) distance(x,y) distance(poly) direction(punkt) direction(x,y) getPoint() translate(dx,dy) translate(punkt) translatePolar(ang, dist) toString()	translate(dx,dy) getCenter() rotate(ang) length() isVertical() isHorizontal() intersects(linje) toString() draw(graphics) direction() minimumDistance(punkt) minimumDistance(x,y)	translate(dx,dy) translate(punkt) getPolygon() getCenter() getPunkt(i) getBounds() updateBounds() intersects(poly) intersectsLinje(poly) overlaps(poly) rotate(ang) setDirection(dir) getDirection() getLinje(i)

Namn	Material	Rorelse	KartObjekt
Data	double weight, friction double brus Color color boolean transparent	KartObjekt sak double hastighet, rotation boolean moveable	Poly pol, Material mat, Rorelse mov KartObjekt connectedTo Int thispoint, topoint Vector sensors, effectors
Konstruktorer	Material() Material(material) Material(color) Material(transparent) Material(color, frict, wgt)	Rorelse(sak, Rorelse) Rorelse(sak) Rorelse(sak, hast, rotat)	KartObjekt(poly p) KartObjekt()
Metoder	isTransparent() getFriction() getWeight() getNoice() getColor() toString() setTransparency(trans) setFriction(frict) setWeight(weig) setNoice(brus) setColor(Color c)	setSpeed(hastighet) getSpeed() setRotation(rotation) getRotation() setDirection(dir) getDirection() update() toString()	setColor(col), getColor() setPoly(poly), getPoly() setConnection(denna, ko, där) clearConnection() getConnected(), connectedAt() connectedTo(), isConnected() getEffectors(), getEffector(i) getSensors() addEffektor(effektor) setEffector(i,values[]) overlaps(kartobj) getReading(i, arena) setMaterial(material), getMaterial() setMovement(rorelse) getMovement() setDirection(dir), getDirection() turn(), update() setSpeed(fart), getSpeed() setRotation(dang), getRotation() setMoveable(boolean), isMoveable() setWeight(w), getWeight() setFriction(f), getFriction() setTransparency(boolean) isTransparent() setNoice(n), getNoice() draw(graphics), erase(graphics, bkcol) translate(dx,dy), getKartObjekt() getColorName(), copy()

Namn	Effektor (super)	Lampa	TvaHjulsMotor
Data	String namn, text int fästpunkt, antal KartObjekt sak double[] value, wanted	*extends Effektor* int size Color base, color	*extends Effektor* final double maxspeed, acc double hjulavstånd, d0,d1
Konstruktorer	Effektor(sak, fästpunkt) Effektor(sak, fästpunkt, namn) Effektor(sak, fp, namn, antal)	Lampa(Kartobjekt, fp, color)	TvaHjulsMotor(kartobjekt, fp)
Metoder	setNamn(namn) getNamn() set(value) setDirect(index, val) set(text) getValues() getText() abstract update(ms) abstract draw(Graphics) abstract effekt() abstract fel()	update(ms) effekt() fel() draw(Graphics)	setHjulAvstånd(dist) update(ms) effekt() fel() draw(Graphics)

Namn	Sensor (super)	DistanceSensor	FotoSensor
Data	private String Namn private int strålar, fästpunkt private double djup, bredd, riktning, diroffset, ambience, brus Poly sak KartObjekt kartobjekt double delay double[] value	*extends Sensor*	*extends Sensor*
Konstruktorer	Sensor() Sensor(sak, fästpunkt)	DistanceSensor(sak, fp, dir)	FotoSensor(sak, fp, dir)
Metoder	adjustDirection(ang) setPoly(sak) getPoly() setKartObjekt(kartobjekt) getKartObjekt() setDelay(sec) getDelay() setName(Namn) getName() setFields(antal) getFields() setDepth(djup) getDepth() setWidth(bredd) getWidth() setDirection() getDirection() setOrigin(fästpunkt) getOrigin() setAmbience(ambience) getAmbience() setNoice(brus) getNoice() getPosition() getReading() getField(nr) getNewReading() abstract makeReading(items)	makeReading(items)	makeReading(items)

Namn	CollisionManager	Robot	Khepera
Data		*extends KartObjekt* private String[] types private String name, type private int size private arena	double scale int points double[] xp, yp
Konstruktorer	CollisionManager()	Robot() Robot (name, type, arena)	Khepera()
Metoder	adjust(kartobjekt1, kartobjekt2)	supported(type) getName() getType() getSpecs() react(recived) updateRobot() eraseSensors(Graphics) drawSensors(Graphics)	getShape() initSensors(poly) initEffektors(robot) protocol(recived, robot)

Namn	Lysdiod	GPS	Kompass
Data	*extends Effektor*	*extends Sensor*	*extends Sensor*
Konstruktorer	Lysdiod(sak, fp, col)	GPS(sak, fp, dir)	Kompass(sak, fp, dir)
Metoder	update(ms) effekt() fel() draw(Graphics)	makeReading(items)	makeReading(items)

Namn	Kartograf	NärhetsSensor	
Data	*extends Effektor*	*extends Sensor*	
Konstruktorer	Kartograf(sak, fp, col)	NärhetsSensor(sak, fp, dir)	
Metoder	update(ms) effekt() fel() draw(Graphics)	makeReading(items)	

Appendix D - Användarmanual

Inledning

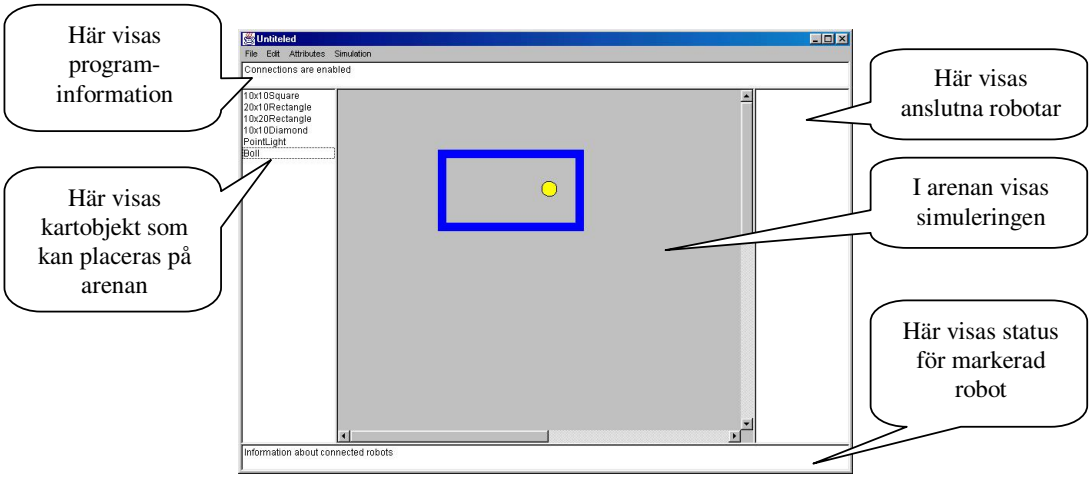
Den här robotsimulatorn har vuxit fram ur behovet att ge er en möjlighet att simulera styrprogram och inlärningsalgoritmer för olika robotar utan att ha tillgång till en riktig robot. Fördelen med att använda denna simulator är att när man har skapat ett program som fungerar i simulatorn är det tänkt att det skall fungera direkt mot roboten genom att ansluta roboten antingen genom en radiolänk eller genom en RS-232 port med kabelanslutning till roboten.

Simulatorn är skriven i Java och det rekommenderas att ni gör ert styrprogram i java, även om det inte är helt nödvändigt. Om ni gör det i något annat språk så får ni även ta hand om kommunikationen över det lokala nätverket.

Översikt

Robotsimulatorn är ett server-klient program där servern visar världen och robotarna styrs av klienterna. Varje robot har sin egen klient. Den enda roboten som är implementerad i dagsläget är modellen Khepera som är tänkt att användas vid en fortsättningskurs efter tillämpad artificell intelligens, men det är inte så svårt att beskriva en ny robot, se *Hur man gör en ny robot*.

När man startar simulatorn, och nu menar vi servern, så får man upp ett grafiskt gränssnitt med fem delar. Det görs genom att skriva **java RoboWorld** vid promten.



Alla kartobjekt och robotar kan flyttas omkring med musen och man kan markera flera objekt samtidigt genom att hålla shift-tangenten nedtryckt. Man kan klippa, kopiera och klistra kartobjekt eller grupper av kartobjekt till andra ställen på arenan genom att använda alternativen under menyposten *Edit*.

När man skall göra en simulering så skapar man en bana eller laddar in en tidigare skapad bana och startar anslutningen i menyn *Simulation-Connections*. I rutan för programinformation står nu *Connections enabled* vilket tillåter robotar att ansluta sig.

För att starta en robot så starta en robotklient genom att skriva **java RoboClient**, eller namnet på er egen robotklient**,** i ett terminalfönster. Ni får nu tre frågor. Var servern är, vilken typ av robot du vill simulera och vad du vill kalla din robot. Svaren på frågorna kan vara **orff-2**, **khepera** och **Roger**. Nu dyker roboten upp i övre vänstra hörnet på serverns arena och du kan flytta roboten vart du vill med musen.

Klasser

Programmet betstår av ett antal klasser som bildar ett kartobjekt. En robot är ett kartobjekt likaså en bit vägg. Ett kartobjekt har alla de egenskaper som man förknippar med ett rörligt objekt på skärmen. Den har en riktning, en hastighet, ett material och den beskrivs av en polygon. Materialet beskriver färgen, vikten och friktionen mot underlaget samt om den är transparent.

Alla koordinater i programmet beskrivs av typen double så det är möjligt att flytta ett kartobjekt ett avstånd som inte är ett heltal.

Robotar

En robot är ett kartobjekt som har sensorer, effektorer och ett styrprogram. Sensorer känner av sin omgivning och effektorer påverkar sin omgivning eller objektet det sitter på. För att en robot skall lära sig något så skall man ändra i klassen **Brain** på klientsidan. I klassen **Brain** skall man skapa en metod **react()** som automatiskt anropas regelbundet. Om man vill så kan man skicka förfrågningar till servern med metoden **action(String)** och om allt går som det skall får man en returnsträng med svaret.

Ett exempel på kod för att köra en Khepera robot runt i ett rum

```
public void react(){
        int i; String recived;
        recived = action("N");
        if(!recived.startsWith("n")) end();
        else{
            value = sensorValues(recived);
            String r;
            if(value[2] < 800&&value[3] < 800)r = action("D,4,4");
            else if (value[2] > value[3]) r = action("D,3,4");
            else r = action("D,4,3");
        }
}
```

Här görs en förfrågning om sensorvärden

Omvandlar mottagen sträng till en vektor med 8 heltal

Beroende på sensorvärde 2 och 3 kör eller sväng

Hur man skapar en effektor

En effektor är ett objekt som påverkar omgivningen eller det kartobjekt det sitter på. De effektorer som är skapade nu är tvåhjulsmotorn som sitter på en Khepera och lampor. En effektor beskrivs av ett antal datafält som alla innehåller ett värde av typen *double*. Det finns fyra metoder som man skall beskriva för att definiera en effektor. Man kan komma åt alla kartobjektets metoder ifrån referensen till det kartobjekt som effektorn sitter på och om det är en robot så kan man även komma åt alla andra objekt i världen genom dess referens till arenan. Här nedan beskrivs metoderna man skall definiera.

Dessa abstrakta metoder skall definieras i effektorn

```
public abstract void update(double ms); //Tid sedan förra update() i ms
public abstract void draw(Graphics g); // Ritar effektorn
public abstract boolean effekt(); // Här implementeras effektorn
public abstract void fel();
 // om man definierar feltillstånd i effekt så kan dessa tas om hand om här
```

Man har tillgång till alla de metoder man ärver från effektorsuperklassen. Den klassens metoder står beskrivna här nedan.

Metoder som ärvs från superklassen effektor

Datafält

```
String namn = "Effektor"; String text;
int fästpunkt;KartObjekt sak;
int antal = 1; // antal effektorvärden;
double[] value = new double[antal];
double[] wanted = new double[antal];
```

Konstruktorer

```
public Effektor(KartObjekt p, int fp)
public Effektor(KartObjekt p, int fp, String)
public Effektor(KartObjekt p, int fp, String n, int ant)
```

Metoder

```
public void setNamn(String n)
public String getNamn()
public void set(double[] val) // ger effektorn önskade värden
public void set(String s)   // ger effektorn en sträng
// set utför effekt() och om det blir fel (false)
// så anropas metoden fel().
public double[] getValues() // hämtar effektorvärden
public String getText() // hämtar en sträng I effektorn
```

Abstrakta metoder

```
public abstract void update(double ms);// Tid mellan update() i ms
public abstract void draw(Graphics g);
public abstract boolean effekt();
public abstract void fel();
}
```

Utöver dessa metoder så har man metoderna för det kartobjekt som effektorn sitter på. Dessa kan man komma åt från datafältet **sak**. För en beskrivning av **kartobjekt** se nedan.

Exempel på en effektor

Ett exempel på effektor är en lampa. Här nedan följer en beskrivning hur den är implementerad.

Lampans implementation

```
import java.awt.Color; import java.awt.Graphics;

public class Lampa extends Effektor{

Datafält
   int size = 3;
   Color base = new Color(255,255,245);
   Color color = new Color(255,255,245);

Konstruktorer
   Lampa (KartObjekt kartobjekt, int fästpunkt, Color color){
      super(kartobjekt, fästpunkt,"Lampa",1); this.base = color;}

   Lampa (KartObjekt kartobjekt, int fästpunkt){
      super(kartobjekt,fästpunkt,"Lampa",1);}

Metoder
   public void update(double ms) {
      if (wanted[0] > = 1)value[0] = 1; else if (wanted[0] < = 0)value[0] = 0;
      else value[0] = wanted[0]; effekt(); }

   public boolean effekt() {
      color = new Color((int)(base.getRed()*value[0]),(int)(base.getGreen()*
value[0]),(int)(base.getBlue()*value[0]));
      return true;  }

   public void fel(){}

   public void draw(Graphics g) {
      g.setColor(color);
      Punkt fp = sak.getPoly().getPunkt(fästpunkt);
      g.fillOval((int)(fp.x-size),(int)(fp.y-size),2*size,2*size);   }
}
```

En lampa har en färg och en storlek.

Lampan är en fylld cirkel med rätt färg placerad rätt över sin fästpunkt.

Effekten är att färgen och ljusstyrkan sätts rätt.

Hur man skapar en sensor

En sensor är någonting som tolkar sin omgivning till ett antal värden. En sensor kan placeras på ett kartobjekt men inte förrän kartobjektet är en robot kan värdena läsas och användas på något sätt[1]. De sensorer som finns just nu är avståndssensorn och ljussensorn. Båda klasserna ärver från den abstrakta superklassen **Sensor**. Här nedan följer en kort beskrivning av superklassen Sensor.

Superklassen Sensor

```
import java.util.Vector;
public abstract class Sensor{
```

Datafält

```
    private String Namn;
    private int strålar = 1; // antal strålar, djup långa, fördelade i bredd
    private double djup = 1000; private double bredd = 0.1; // radianer
    private double riktning = 0; private double diroffset = 0;
    private int fästpunkt = 0; // var på Poly sitter den
    private double ambience = 0.02; // bakgrundsstrålning
    private double brus = 0.02; // brus, variation kring sensorvärdet
    private Poly sak = null; // den sak som sensorn är fäst på
    KartObjekt kartobjekt = null;  double[] value; // mätvärden
    double  delay = 0.05; // Sensorns hastighet I ms
```

Konstruktorer

```
    public Sensor(){}
    public Sensor(Poly p, int fp){sak = p;fästpunkt = fp;setDirection();}
```

Metoder

```
    public void adjustDirection(double a){ diroffset = a;
        if (sak! = null) setDirection(); }
    public void setPoly(Poly p) {sak = p;}
    public Poly getPoly() {return sak;}
    public void setKartObjekt(KartObjekt ko){kartobjekt = ko;}
    public Poly getKartObjekt(){return kartobjekt;}
    public void setDelay(double t) {delay = t;}
    public double getDelay() {return delay;}
    public void setName(String n) {Namn = n;}
    public String getName() {return Namn;}
    public void setFields(int ant) {strålar = ant;value = new double[strålar];}
    public int getFields() {return strålar;}
    public void setDepth(double d) {djup = d;}
    public double getDepth(){return djup;}
    public void setWidth(double b) {bredd = b;}
    public double getWidth(){return bredd;}
    public void setDirection(){riktning = sak.getDirection() + diroffset;}
    public double getDirection(){setDirection();return riktning;}
    public void setOrigin(int fp){fästpunkt = fp;}
    public int getOrigin(){return fästpunkt;}
    public void setAmbience(double a){ambience = a;}
    public double getAmbience() {return ambience;}
    public void setNoice(double b) {brus = b;}
    public double getNoice(){return (Math.random()-0.5)*brus;}
    public Punkt getPosition() {return new Punkt(sak.getPunkt(fästpunkt));}
    public double[] getReading(){return value;}
    public Poly getField(int nr){   … kod … return new Poly(x,y,3);}
    public double[] getNewReading(Vector ){
        setDirection(); makeReading(k); return value;}
```

En sensor har ett eller flera sensorfält

En sensor är fäst vid en punkt på en Poly.

När man skapar en sensor måste den oftast veta var den är

Ändra riktningen på sensorn i förhållande till Polyn den sitter på

Ange antal mätvärden som sensorn skall registrera

Hur långt ser sensorn?

Returnera en triangel som motsvarar det fält som sensorn uppfattar saker i. (Beroende på sensor)

Denna metod skall implementeras

[1] Meningen är att en sensor direkt skall kunna påverka en effektor som är placerat på samma kartobjekt. Detta går att göra men kräver att man skapar en effektor som själv anropar sensorn.

Implementation av avståndssensorn

Hur gör man då för att skapa en ny typ av sensor ifrån denna superklass? Jo man gör en ny klass som ärver från Sensor och som implementerar **makeReading(Vector items)** metoden. Här nedan följer ett programexempel på hur avståndssensorn är gjord.

Implementation av avståndssensor

```
import java.util.Vector; import java.lang.Math;
public class DistanceSensor extends Sensor{
```

Konstruktor

```
    public DistanceSensor(Poly sak, int fp, double r){
        super(sak,fp); super.setName("Distance Sensor");
        setFields(1); setWidth(0.3); setDepth(250);
        adjustDirection(r); }
```

Metod

```
    public void makeReading(Vector items){
        int number = items.size(); KartObjekt item;
        Poly field; double dist, best; double d = getDepth();

        for (int j = 0;j < getFields();j++) {
            field = getField(j); dist = d; best = d;
            for (int i = 0;i < number;i++) {
                item = (KartObjekt)items.elementAt(i);
                if(this.getPoly()! = item.getPoly()) {
                    if (field.overlaps(item.getPoly()))
                        dist =  field.getPunkt(0).distance(item.getPoly());
                    if (best > dist) best = dist; }
            }
    value[j] = Math.max((d-best)/d + getNoice(), getAmbience() +  getNoice());
        }
    }
}
```

Här tilldelas namn och grundinställningar för denna typ av sensor

Här beräknas det närmsta avståndet *best* till alla kartobjekt som korsar sensorns fält

Här tilldelas det uträknade sensorvärdet med brus och bakgrundsnivå

Hur man gör en ny robot

När man gör en ny robot så gör man en klass som har ett par metoder som beskriver dess språk. Språket är ett antal strängar som antingen ger värden till robotens effektorer eller hämtar värden från dess sensorer. Det är lättast att ge ett exempel på hur en robot kan se ut.

Implementation av en ny robot

```
import java.util.*;
public class Khepera{
    static int points = 12;
    static double[] xp = {0,2,5,9,12,14,14,12, 9, 5, 2,0,7};
    static double[] yp = {5,2,0,0, 2, 5, 9,12,14,14,12,9,7};

    public static Poly getShape(){return new Poly(xp,yp,points);}

    public static Vector getSensors(Poly poly){
        Vector sensors = new Vector();
        sensors.addElement(new DistanceSensor(poly,3,1.57));
        sensors.addElement(new DistanceSensor(poly,4,0.78));
        sensors.addElement(new DistanceSensor(poly,5,0.17));
        sensors.addElement(new DistanceSensor(poly,6,6.11));
        sensors.addElement(new DistanceSensor(poly,7,5.50));
        sensors.addElement(new DistanceSensor(poly,8,4.71));
        sensors.addElement(new DistanceSensor(poly,11,3.32));
        sensors.addElement(new DistanceSensor(poly,0,2.97));
        return sensors;  }

    public static Vector getEffectors(Robot robot){
        Vector effectors = new Vector();
        effectors.addElement(new TvaHjulsMotor(robot,12));
        return effectors; }

    public static String protocol(String recived, Robot robot){
        int i;   double[] reading = new double[1];
        double[] wheels = new double[2];
        String reply = "OK";
        Double speed, angle, left, right;

        if(recived.startsWith("speed")){
            speed = new Double(recived.substring(5).trim());
            robot.setSpeed(speed.doubleValue()); }
        else if(recived.startsWith("turn")){
            angle = new Double(recived.substring(4).trim());
            robot.turn(Math.toRadians(angle.doubleValue())); }
        else if(recived.startsWith("A")){ reply = "NS";     }
        …
        else if(recived.startsWith("D")){
            reply = "d"; recived = recived.substring(2).trim();
            i = recived.indexOf(",");
            left = new Double(recived.substring(0,i).trim());
            right = new Double(recived.substring(i + 1).trim());
            wheels[0] = left.doubleValue();
            wheels[1] = right.doubleValue();
            robot.setEffector(0,wheels); }
        else if(recived.startsWith("E")){ reply = "NS";     }
        …
        else if(recived.startsWith("N")){
            reply = "n"; for(i = 0;i < 8;i + +){
                reading = robot.getReading(i);
                reply = reply + ", " + Math.round(reading[0]* 1024);}}
        else if(recived.startsWith("O")){ reply = "NS";     }
        …
        }
```

Skapar punkterna för robotens poly

Metod för att dela ut robotens poly

Definierar robotens sensorer

Definierar robotens effektorer

Definierar robotens språk.
Strängen *recived* är kommandot som ges till roboten ifrån klienten och beroende på robottyp så skall olika saker utföras.
Strängen *reply* returneras tilbaka till klienten.

Här avläses robotens sensorer och värdena returneras till klienten