Sammanfattning

I den här uppsatsen har jag studerat den mobila robot som kallas Khepera. Uppgiften har sitt ursprung i Lunds universitets robotikprojekt som startades hösten 1999. Arbetet har bestått i ett antal olika experiment där jag programmerat Kheperan att utföra olika uppdrag. De har främst handlat om navigering med hjälp av intryck från den omedelbara omgivningen. I de flesta fall har jag använt mig av beslutträdsliknande algoritmer med flera olika tillstånd. Resultaten har jag sedan utvärderat. Slutsatserna blev flera, de kanske viktigaste var upptäckten av den linjära kamerans begränsning och närhetssensorernas användbarhet. Arbetet fungerar som en utvärdering av Kheperans möjligheter och begränsningar och innehåller flera uppslag till ytterligare utveckling.

Abstract

In this report I have studied the mobile robot called Khepera. The task has its origin in the robotics project started at the University of Lund in the fall of 1999. My work has consisted of various experiments where I have programmed the Khepera to perform different tasks. They have mainly been orientated towards navigation using sensory input from the immediate surroundings. In most cases I have used algorithms based on decision trees with different states. I have then evaluated the results. There are many conclusions. Perhaps the most important was the discovery of the limitations of the linear camera and the capabilities of the proximity sensors. My paper is an evaluation of the possibilities and limitations of the Khepera. It also contains several ideas concerning further development.

Förord

Det här examensarbetet växte fram ur den satsning på robotik som gruppen för Artificiell Intelligens vid Lunds Universitet gjorde med start hösten 1999. AI–gruppen består av en samling eldsjälar som nu tar upp denna högst tidsenliga gren av den artificiella intelligensen. Det har varit roligt att kunna deltaga i detta projekt, och jag hoppas att mitt arbete kan vara till nytta för gruppen i deras fortsatta forskning.

Jag vill här passa på att tacka de personer som speciellt hjälpt mig under mitt arbete. Först och främst vill jag tacka Jacek Malec, min handledare, som visat intresse för mitt arbete och stor förståelse för de problem som uppstått. Jag vill även tacka Eric Astor som varit som en andra handledare och en riktig idéspruta. Ett stort tack går även till Johan Lindén som hjälpte mig att komma igång och sedan flitigt följt utvecklingen och till Robert Arnholm som gav viktiga syn–punkter på rapporten. Slutligen vill jag tacka alla de som följt mitt arbete genom upprepade besök i robotlabbet. Det kändes verkligen som om jag var på rätt väg när så många visade sitt intresse. Tack.

Innehållsförteckning

1 Inledning

1.1 Bakgrund

Under våren år 2000 köpte AI–gruppen vid institutionen för datavetenskap in två stycken robotar av modellen Khepera. De skall användas till undervisning, främst i kursen "AI för robotar" som startar på hösten samma år.

Redan under hösten 1999 gjordes två examensarbeten med anknytning till Khepera. Det ena heter "Objektorienterad robotsimulator" och skrevs av Henrik Lernmark [1] och det andra "Objektorienterad robotstyrning och simulering i realtid" och skrevs av Tim Portnoff [2]. Tillsammans gjorde de "RoboWorld" som är en simulator för mobila robotar och med speciellt stöd för Khepera.

1.2 Målet

I samband med robotiksatsningen fanns flera förslag till examensarbeten som handlade om robotar. Jag fastnade för ett som var orienterat kring robotseende och hur roboten kunde orientera sig och navigera med hjälp av detta. Målet är att tolka synintryck och med hjälp av dessa interagera framgångsrikt med om–givningen. Jag ska studera robotarnas möjligheter och begränsningar med speciell vikt på seendet.

1.3 Fakta

Det här examensarbetet handlar om autonoma mobila robotar. En robot är en maskin som i någon mån kan fatta egna beslut och utifrån dessa interagera med sin omgivning. En mobil robot är en robot som har förmågan att förflytta sig av egen kraft. En autonom robot är en robot som efter att den startats fattar alla beslut själv. Den mobila robotiken är fortfarande i sin vagga. Den nytta vi hittills haft av mobila robotar är begränsad. Det har konstruerats autonoma mobila robotar som kan klippa en gräsmatta själv eller dammsuga ett rum. En annan kommersiell tillämpning är att få robotarna att spela en variant av fotboll. Detta är ganska utbrett och det ordnas årliga mästerskap i olika ligor [3]. Vissa kanske skulle hävda att en bombrobot (som kan desarmera bomber) är ett bra exempel på en användbar autonom mobil robot, men en sådan robot är inte autonom eftersom den i större eller mindre grad fjärrstyrs av en människa och alltså inte fattar alla sina beslut själv.

Under de flesta experiment som jag utförde var Kheperan inte helt autonom. Jag skickade kommandon till den genom en seriell kabel som också förság roboten med ström. Detta tillvägagångssätt är praktiskt då man gör flera experiment och vill kunna felsöka resultaten. Roboten kan även gå på sitt eget batteri men styrprogrammet måste då finnas ombord på roboten och det begränsas då av Kheperans prestanda.

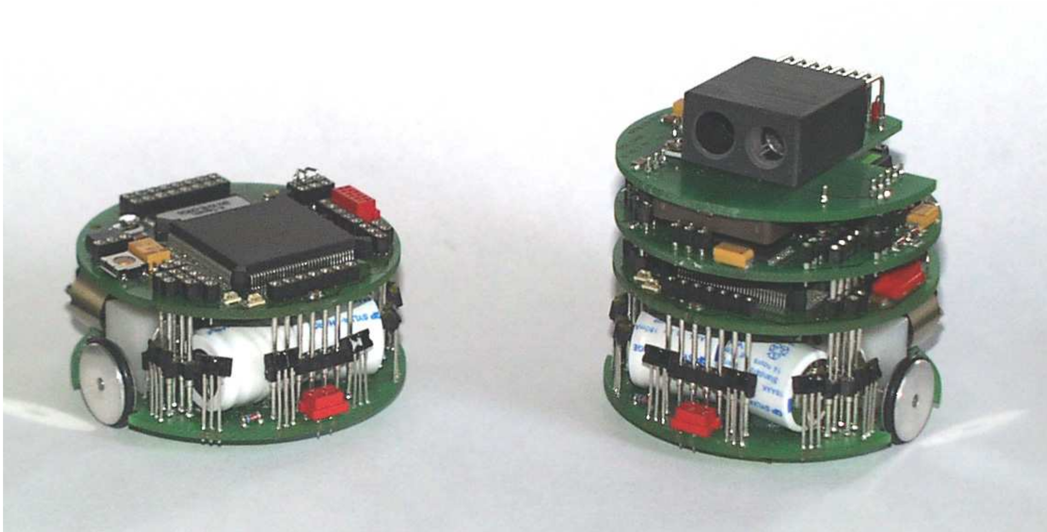
2 Robotarna

2.1 Specifikation

De mobila robotarna som examensarbetet kretsar kring är av modellen Khepera. Roboten har utvecklats av Laboratorie de Microinformatique [4] och distribueras av ett företag som heter K–team [5], båda ligger i Schweiz. Kheperan designades i forsknings– och undervisningssyfte och är vitt spridd över världen.

Kheperan är cylindrisk, 55 mm bred och 30 mm hög i sitt basutförande. Inuti sitter en Motorola 68331 processor med en hastighet på 16 MHz. Roboten har ett RAM på 256 Kbytes och ett ROM på lika mycket. I basmodulen sitter fyra uppladdningsbara Nickel–Kadmium batterier som kan förse Kheperan med ström i 45 minuter. När jag här skriver om någon av robotarna kommer jag att omväxlande kalla den för "roboten" eller "Kheperan" för att undvika upprepningar.

Till basmodulen finns flera olika tilläggsmoduler. Dessa monteras ovanpå basmodulen. Här finns två olika sorters kameror, en griparm, en radiomodul och en "I/O"–modul där man kan skicka in signaler från egna påbyggnader.



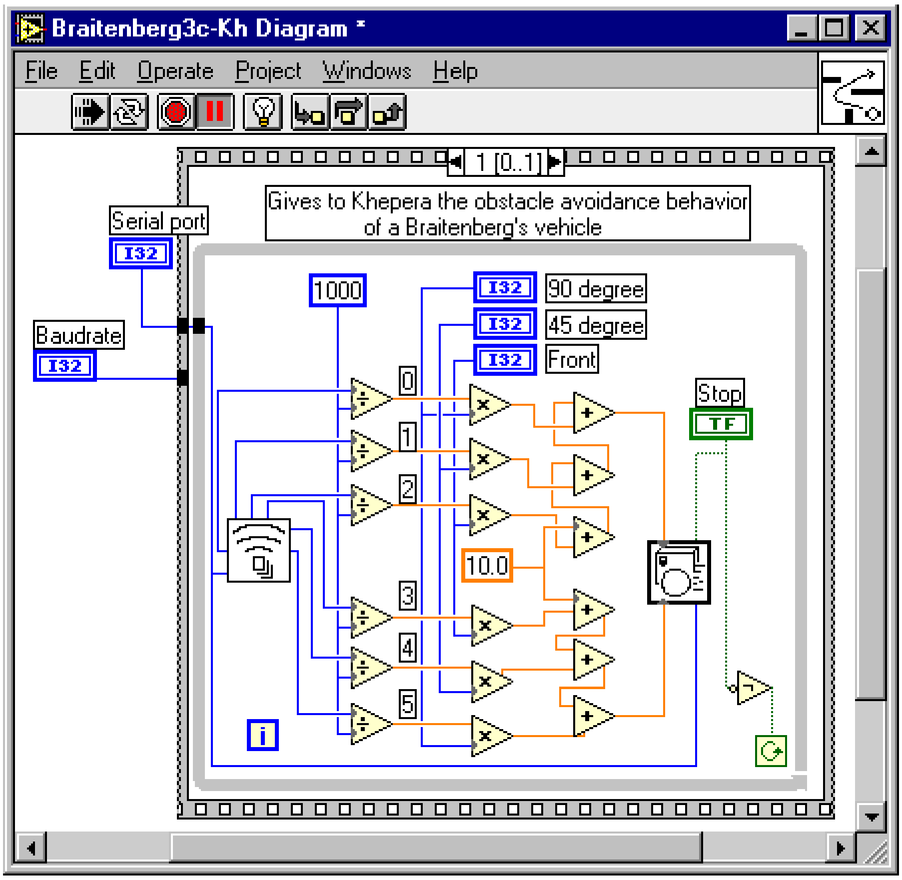
Figur 2.1: Institutionens båda robotar. Till vänster, endast basmodulen. Till höger, en basmodul med kameramodul.

2.2 Programvara

För att kunna initiera robotarna och göra dem mottagliga för kommandon, använder jag minicom för att ställa in bps, paritet m.m. Med Kheperorna levererades flera program för att underlätta experimenterandet med dem. Ett var den så kallade "cross–compilern", i fortsättningen kallad korskompilatorn, som innehåller stöd för Kheperans egna funktioner. Med denna kan man kompilera ett program (skrivet i C) på värddatorn som sedan kan laddas ner på roboten i körbart format [6, 7].

Ett annat program som följde med robotarna var ett stöd för LabVIEW. LabVIEW är ett grafiskt programmeringsgränssnitt som till exempel kan illustrera sensorernas utslag. Detta är högst användbart då man lär känna

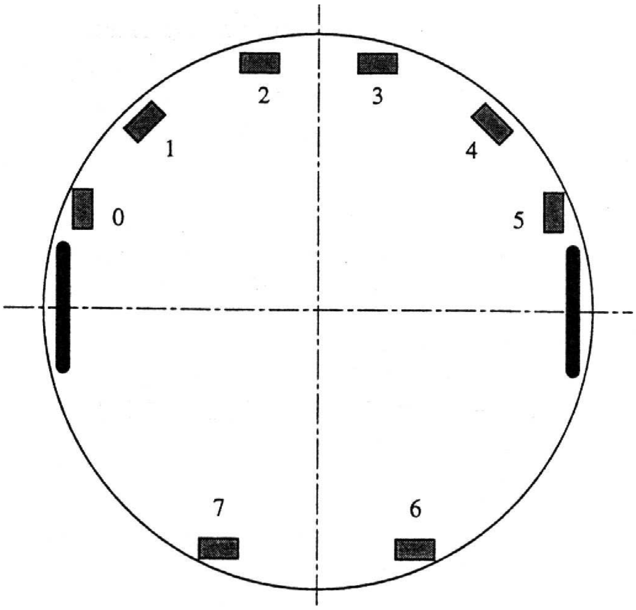
Kheperorna. I LabVIEW kan man även "bygga" sina egna program, något som dock kräver en del erfarenhet av applikationen.



Figur 2.2: Bild ur LabVIEW. Programmeringsgränssnitt med flödesschema för Braitenbergs navigeringsalgoritm.

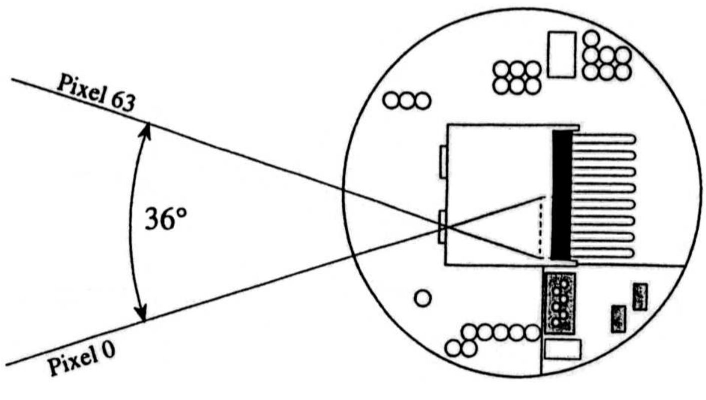
2.3 Sensorerna och hjulen

Runt robotkroppen sitter åtta infraröda sensorer, fyra är riktade framåt, två åt sidorna och två bakåt. De fungerar så att de först läser av bakgrundsljuset, sedan skickar de ut en ljusimpuls med infrarött ljus och mäter igen. Den första mätningen subtraheras från den andra. Finns det nu föremål i närheten reflekterar dessa delar av det utsända ljuset och skillnaden blir ett mått på avståndet till dessa. Givetvis blir en sådan mätning mycket osäker. Det reflekterade ljuset beror inte bara på avstånd utan även materialens reflexivitet och övriga ljusförhållanden. Ljusa ytor reflekterar bättre än mörka. Matta ytor registreras generellt på större avstånd än blanka. Ir–sensorerna har en räckvidd på 20 – 30 mm. Vissa material, såsom svart plast, registreras på mindre än 10 mm. Sensorerna returnerar ett värde mellan 0 och 1023, där 0 betyder att inget registrerats och 1023 är fullt utslag. Ibland kallar jag dessa sensorer för närhetssensorer.



Figur 2.3: Översiktsbild på sensorernas placering på robotkroppen. Märk även hjulens placering på vardera sida om kroppen.

Det finns som jag nämnt ovan, ett antal olika på byggnadsmoduler. Den jag använde mig av var en "vision turret" [8] som är en toppmodul med en svartvit linjär kamera, som har en upplösning på 1 x 64 pixlar och en gråskala på 256 nyanser. Kameran täcker en vinkel på 36°. På modulen sitter även en ljusdetektor som mäter den allmänna ljusintensiteten hos omgivningen. Kameran använder sedan detta för att ställa in lämplig slutartid. Slutartiden påverkar antalet bilder som kameran kan ta varje sekund. Under optimala förhållanden är bildfrekvensen 5 Hz.



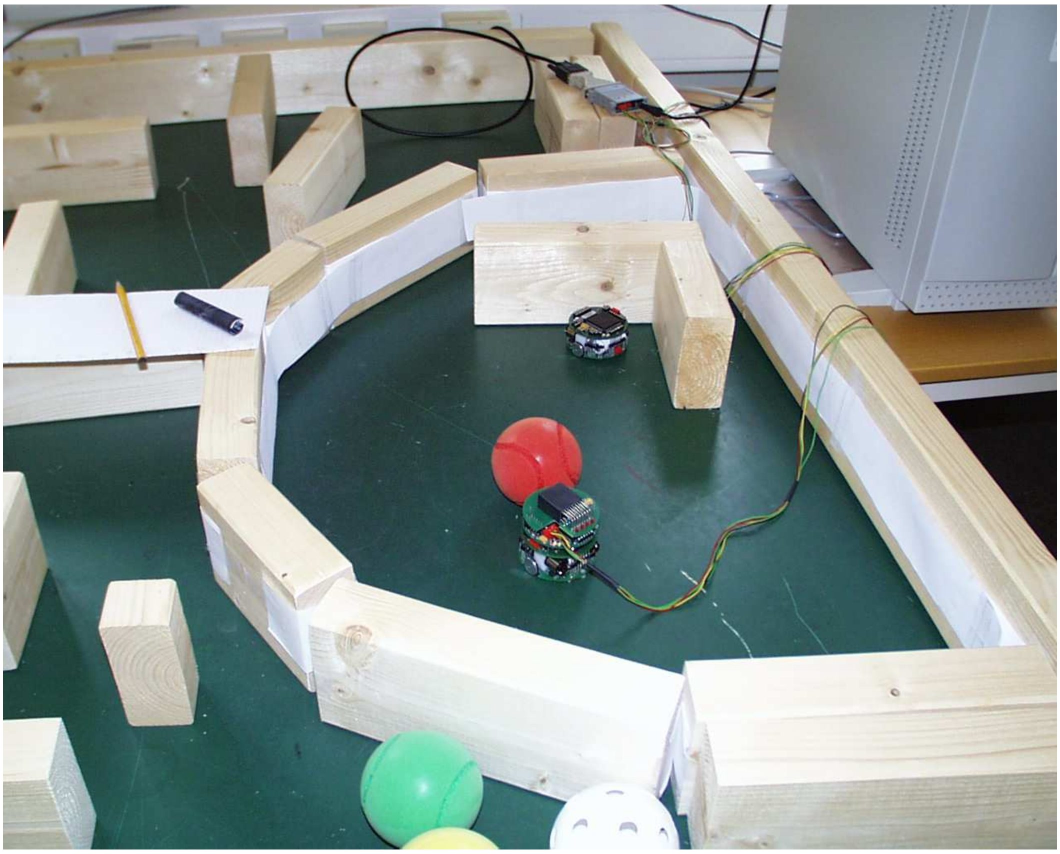
Figur 2.4: Översiktsbild på kameramodulen sedd uppifrån. Den vänstra kameraöppningen tar bilden, den högra reglerar slutartiden med hjälp av det totala ljusinsläppet.

Roboten förflyttar sig med hjälp av två hjul, ett på vardera sida om kroppen. För att inte roboten ska släpa eller tippa finns två stödpunkter på undersidan som håller roboten upprätt. Hjulen är kopplade till två motorer som kan driva hjulen oberoende av varandra. Roboten kan alltså både rotera på plats eller svänga medan den färdas. Jag kommer i fortsättningen att skilja mellan dessa. Vid rotation snurrar robotens båda hjul lika fort fast åt olika håll, resultatet blir att roboten svänger utan att förflytta sig. Detta kan vara smidigt att använda då roboten är i ett trängt läge och inte vill riskera att kollidera. Det är också bra då man vill vända sig en bestämd vinkel. Då roboten svänger går båda hjulen vanligtvis i samma riktning, men det ena fortare än det andra, alltså förflyttas roboten samtidigt som den ändrar orientering. Till varje hjul hör en räknare som räknar hjulets position sedan den senaste nollställningen. En enhet på räknaren motsvarar 0,08 millimeter.

2.4 Kommunikation

Man kan kommunicera med Kheperan antingen genom en seriell kabel eller genom radiosignaler [9, 10] (det sistnämnda alternativet kräver dock att roboten utrustas med en radiomodul). Då kan man skicka kommandon till robotens kommandotolk. Ett kommando består av en versal som eventuellt följs av en eller flera tal åtskilda av kommatecken [11]. Kommandona är av typen "rulla höger–hjulet med hastighet x" eller "läs av närhetssensorerna". Roboten svarar med en gemen följt av tal. På detta sätt kan man ha styrprogrammet på en värddator och enbart skicka kommandon till Kheperan. En fördel med detta är att man får en bra bild av vad som sker i programmet. Är roboten ansluten till värddatorn med en kabel kan den dessutom få sin strömförsörjning genom denna och man sparar batterierna. Denna metod lämpar sig bra vid längre tids experimenterande och var också den som jag använde mest. Ett annat sätt är att kompilera styrprogrammet på datorn och ladda ner det till robotens minne. Roboten blir då helt "själv–ständig" och exekverar programmet tills det tar slut eller roboten stängs av. Nackdelen är att möjligheten till felsökning är minimal.

2.5 Miljö



Figur 2.5: Robotarna i robotlabbet. Den ena roboten är ansluten till värddatorn som skymtar till höger.

För att framgångsrikt kunna laborera med robotar krävs en del av omgivningen. Det viktigaste är kanske att skydda robotarna. Deras ringa storlek och känsliga elektronik gör dem ömtåliga för yttre påverkan. Vid försök med Kheperorna vill man att miljön ska vara enkel och konsistent. Ett uttjänt pingisbord försågs med kantväggar i form av rejäla träbjälkar, så att inte robotarna skulle kunna falla ner. Mindre klossar framställdes för att agera som väggar inne på banan. Även ett antal skumgummibollar införskaffades. Värddatorn, kallas i bland ”robotdatorn”, placerades alldeles bredvid bordet så att robotarna skulle kunna vara anslutna till den med en ledning. Denna omgivning kallar jag i fortsättningen "robotlabbet".

3 Experimenten

3.1 Allmänt

Inför mitt arbete med Kheperorna studerade jag flera olika texter om robotseende och navigering [12, 13, 14, 15, 16, 17]. Metoderna att styra en robot är många. En variant är att använda sig av inlärning där det finns flera olika modeller. Jag tyckte inte att detta passade riktigt i det här fallet då de ofta kräver att man kör programmet många gånger i inlärningssyfte och eftersom programmets hastighet begränsas av roboten skulle detta ta mycket tid i anspråk. En annan anledning är att jag anser att man inte riktigt kommer i kontakt med problematiken i de olika stegen, man är inte alltid tvungen att sätta sig in i detaljerna.

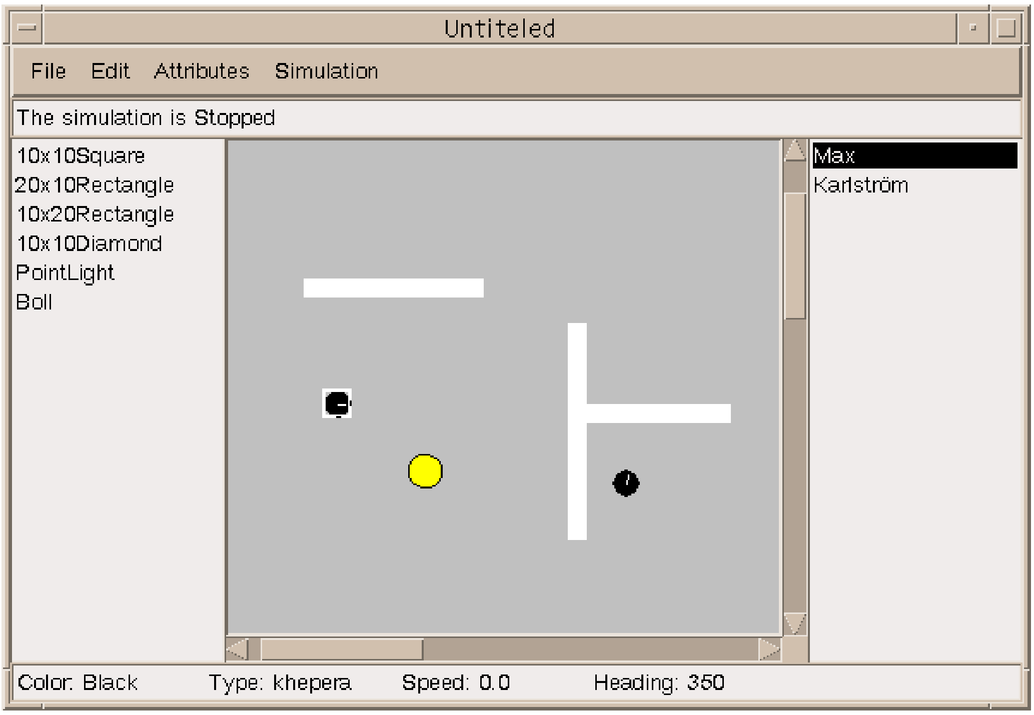
Jag har i mina större experiment använt mig av en metod som mest kan liknas vid beslutsträd [18]. Roboten kontrollerar om vissa kriterier är uppfyllda och väljer sitt handlande utefter detta. Ofta tar den också hänsyn till vad den gjorde i sitt senaste beslut (och ibland ännu flera steg tillbaka). I några experiment använde jag mig också av tillstånd. Då vissa krav uppfylls försätts programmet i ett nytt tillstånd och plötsligt gäller andra regler. Jag valde att programmera i C, främst för att korskompileraren kompilerar C–kod. På detta sätt behövs inga stora modifikationer om jag vill anpassa koden till att laddas ner till Kheperan. Ofta börjar jag med en enklare uppgift och bygger sedan på fler finesser efter hand.

För att inte detta kapitel ska bli för långt innehåller det ingen kod, pseudokod eller funktionsnamn. Vill man fördjupa sig i koden kan den nås via adressen http://www.cs.lth.se/~ai. Bollhantering finns även kommenterad i Appendix. Jag har i största möjliga mån försökt dela upp programmen i små moduler. Till exempel finns det i flera av programmen en funktion för att rotera, med riktning och grader som invariabler. Ett annat genomgående drag är att många funktioner kan presentera mätdata. Vill man se dessa talar man om detta i funktionsanropet. Detta underlättar för dem som vill förstå vad som sker vid exekveringen, eller för den som vill felsöka.

3.2 RoboWorld

Under hösten 1999 gjorde examensarbetarna Henrik Lernmark och Tim Portnoff "RoboWorld" som är en simulator för robotar i allmänhet och Khepera i synnerhet. Då en sådan simulator skulle kunna vara ett utmärkt hjälpmedel, tog jag mig tid att studera RoboWorld närmare. Jag fann ett högst användbart program som väl åskådliggjorde hur Kheperorna kunde agera. Framförallt möttes jag av ett smidigt användargränssnitt. Tyvärr fanns inget stöd för den linjära kamera som jag avsåg att använda. RoboWorld är dock gjort så att det är mycket lätt att själv lägga till nya sensorer. Jag gjorde ett försök att utifrån specifikationen implementera kameran ifråga.

Med facit i hand, i form av erfarenhet från de riktiga robotarna, får jag konstatera att simuleringen av kameran inte blev så lyckad. En av svårigheterna ligger onekligen i att RoboWorld endast stöder två dimensioner, en annan i att stödet för nya sensorer till Kheperan inte fullt lämpar sig för kameror. Anledningen är att varje pixel i kameran (64 stycken) måste implementeras som en egen sensor. Dessa sensorer skulle ha mycket lång räckvidd. Kort sagt blir kameran oerhört beräkningstung. Färgerna måste översättas till en gråskala som normaliseras av det totala ljusinsläppet. Jag bedömer dock att man skulle kunna göra en ganska god simulering av den linjära kameran. Arbetet bakom en sådan är omfattande och skulle kanske lämpa sig som ett examensarbete i sig. Även de färdiga simulerade sensorerna i RoboWorld skulle behöva en del finjustering för att bättre stämma överens med verkligheten. Till exempel skulle bruset och räckvidden kunna anpassas bättre. En annan nyttig påbyggnad skulle vara en omgivning som bättre stämde överens med den som robotlabbet ger.

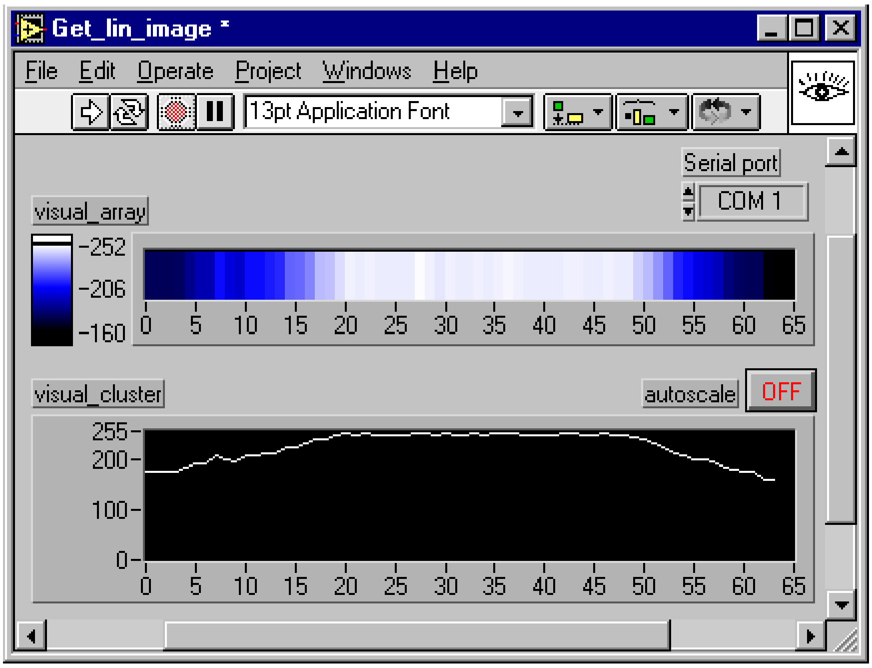


Figur 3.1: Användargränssnittet i RoboWorld. Två svarta Kheperor vid namn "Max" och "Karlström", en boll och ett par väggar syns på planen. Till höger ser man robotarnas namn, nedan deras status och till vänster kan man välja ytterligare objekt att lägga in på planen.

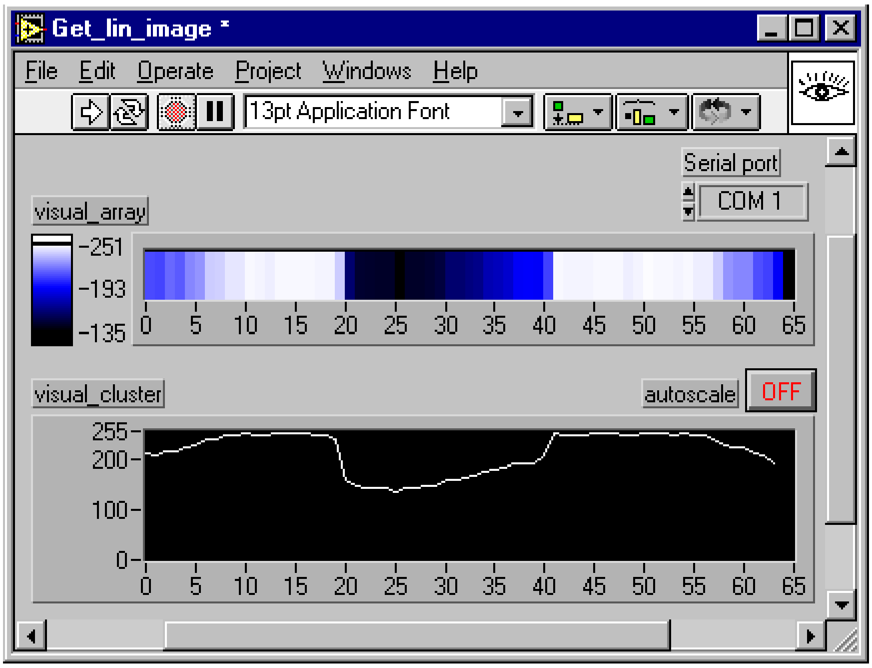
3.3 Identifiering

Förvisso var lokalisering och identifiering av föremål i Kheperans omgivning först inte tänkt som ett separat experiment, men det visade sig vara ett inte helt trivialt projekt och innehöll flera iakttagelser jag skulle vilja redogöra för. Den första iakttagelsen var att den linjära kameran var riktad nästan helt horisontellt. Eftersom den monteras på toppen av roboten betyder detta att den inte ser föremål som är lägre än sig själv. Om man plottar gråskalan (y–axeln) mot pixelnumret (x–axeln), och riktar kameran mot en enfärgad, jämn och ljus yta, får man en knagglig böjd kurva. Den registrerar alltså mörkare nyanser i kanterna av kameran. Jag kan inte vara säker på vad detta beror på, men en teori är att kameraöppningen faktiskt skymmer sikten något. Det kan också helt enkelt vara en följd av att ytan rakt fram är närmare och reflekterar mer ljus än de i kanterna.

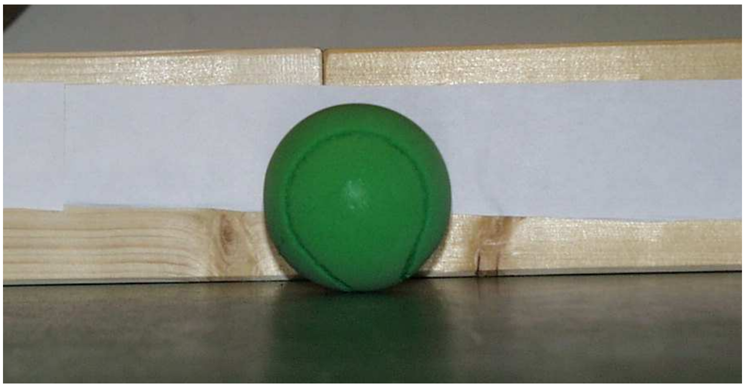
Om man nu ritar ett svart streck på ytan syns detta som ett "dip" i plotten. Detta dip har dock inga skarpa konturer i kamerabilden (trots att så är fallet i verklig–heten), utan det verkar som om pixlarna "överlappar" varandra. Detta innebär i sin tur att om strecket är smalt, så att bara 2 pixlar täcks, kanske det inte alls registreras av kameran.



Figur 3.2: Vy ur LabVIEW där kameran riktats mot en jämn vit yta. Överst är ljusintensiteterna illustrerade med en gråskala. Underst illustreras de i en graf.



Figur 3.3: Kameran är riktad mot samma yta som i figur 3.2, men nu med en grön boll framför. Observera hur bollen är ljusare mot den ena sidan. Detta beror på att ljuset faller från höger och skuggar bollen.



Figur 3.4: Vad roboten egentligen ser i figur 3.3. Kamerablixten gör att bollens skuggning inte längre syns.

För att registrera huruvida det befinner sig ett föremål i kamerans synfält måste man finna konturerna i bilden. Genom att derivera mätvärdena från kameran två gånger, kan man finna de brantare lutningarna i plotten. Då man studerar andraderivatan undviker man jämna ljusskiftningar såsom skuggning och den allmänna kurvaturen i plotten. Jag fick ta hänsyn till flera faktorer för att sortera ut de intressanta skiftningarna. En faktor var känslighet, hur stor skulle andra–derivatan vara för att betraktas som en kontur? Sedan kan jag sortera ut några genom att studera den oderiverade grafen, där det måste vara en minsta skillnad i intensitet före och efter konturen för att det inte ska betraktas som brus. Är konturen riktigt kraftig kan andraderivatan överstiga tröskelvärdet i samma kontur i flera punkter i rad, därför kräver jag att konturen ska understiga ett visst tröskelvärde innan den kan räknas som en ny. Jag vill påpeka att dessa kan vara mycket svåra att skilja från det allmänna bakgrundsbruset och kurvaturen hos plotten. Jag klädde därför väggarna med vitt papper (som ju ger tydligare kontraster än det naturliga träet) och kamouflerade kanterna med böjt papper, för att gömma skarvarna. Jag försökte använda mig av mörka föremål som skulle synas tydligt mot det vita papperet.

Efter att ha funnit konturerna i bilden kan man med hjälp av deras inbördes avstånd ana storleken hos föremålet. Givetvis påverkas bildens storlek av avståndet, men om man antar att allt man ser befinner sig inom en halvmeter från Kheperan kan man ändå göra en grov uppskattning. Sedan kan man använda styrkan på konturen för att uppskatta gråtonen hos föremålet relativt bakgrunden. Ett problem uppstår då man bara ser en del av ett föremål och andra halvan alltså befinner sig utanför bilden. Då kan man inte längre använda sig av storleken hos föremålet för att avgöra vad det är. Vanligtvis klassar jag det då bara som om "något" befinner sig till höger/vänster, och vill man veta vad det är måste man svänga eller rotera i den riktningen.

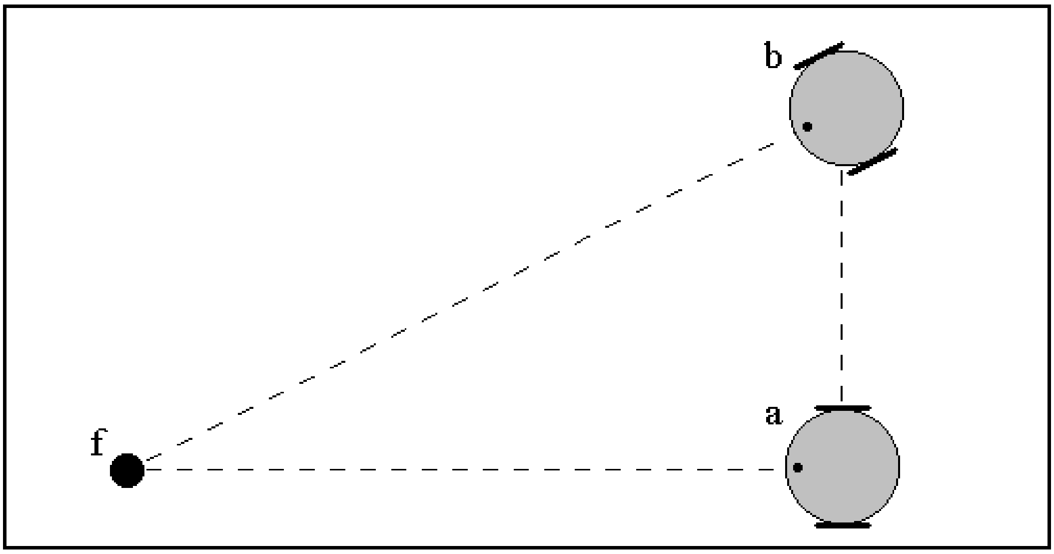
Med hjälp av dessa egenskaper kan man sedan identifiera objektet. Det är ganska tydligt att man utifrån detta inte kan skilja mellan så många olika naturliga föremål. Med naturliga föremål menar jag sådant som en boll, en pelare eller en annan robot. Med konstgjorda kännetecken, kan man skilja mellan betydligt fler (jämför med streckkoder nedan). Många gånger klassade den ändå felaktigt. Detta löste jag genom att när den registrerade något, så stannade den i det läget, och väntade på nästa bild, gav den då samma resultat, accepterades det. Detta kan göras i olika varianter.

Kameran ger ganska olika bilder beroende på bakgrundsljuset. Bäst resultat fick jag med starkt ljus utan kraftiga skuggor. Eftermiddagssolen i robotlabbet lämpade sig bra eftersom det inte satte banan i direkt solljus. Senare på dagen var jag tvungen att tända taklamporna. I bland fick jag även ändra inställningarna i programkoden för att uppnå optimalt resultat, till exempel öka känsligheten för registrering av konturer.

3.4 Avståndsmätning

Ett av de mer grundläggande experimenten vid hanteringen av Kheperorna, var att avgöra deras förmåga att uppskatta avståndet till föremål i deras omgivning. Jag ville göra detta utan att roboten faktiskt tog sig fram till föremålet ifråga. Den kanske mest självklara metoden var att stega upp en sida i en rätvinklig triangel och utifrån denna och två vinklar beräkna avståndet. Det var också den metod jag använde mig av.

Jag började med att tillverka en liten pelare av papper som jag målade svart. Roboten programmerades till att rotera åt vänster tills den fick pelaren i synfältet. Sedan centrerar den pelaren inom de fem mellersta pixlarna i kameran. Roboten roterar nu 90° åt höger, varpå den färdas rakt fram en förutbestämd sträcka. Den roterar nu åter 90°, denna gång åt vänster och centrerar igen föremålet som beskrivet ovan. Enkel trigonometri ger nu avståndet till föremålet.



Figur 3.5: Föremålet är märkt f, Kheperans första position a och den andra b. Pricken i roboten illustrerar framåtriktningen.

Resultatet visade sig dock vara en besvikelse. Det returnerade resultatet hade en felmarginal på nära 50 %. Samma sträcka kunde alltså vid två mättillfällen ge vitt skilda resultat. Den största felkällan spårar jag till centreringen av föremålet. Trots att jag centrerar det till de fem mellersta pixlarna, kan det ger ett fel på nästan 2°. Kameran har 64 pixlar på 36°, och med suddigheterna i konturerna hos föremålet, kan felet uppgå till 2°. Eftersom denna centrering sker två gånger kan det totala felet bli ännu större. Att försöka centrera föremålet exaktare bedömer jag som mycket svårt. En annan felkälla är hjulen. Hjulen har en minsta möjliga steglängd och kan inte rulla hur små steg som helst, dessutom blir felet större ju fler justeringar man måste göra. Sätter man dem båda till samma framåthastighet finner det uppmärksamma ögat att roboten svänger lite åt höger. Troligen beror detta på att vänsterhjulet har en något större radie än högerhjulet. Även skräp på hjulen och ojämnheter i banan kan påverka riktningen. Försök att korrigera detta visade att felet från hjulen var relativt litet. Hur lång roboten färdas i sidled påverkar också exaktheten. Mäter man på ett avlägset föremål får man bättre resultat om roboten färdas längre. Är föremålet nära, kan det vara bättre att inte färdas så långt i sidled. Detta beror på att osäkerheten i rotationerna ger större relativa fel vid små vinklar. Eftersom roboten inte på förhand känner till avståndet till föremålet, måste man använda ett förutbestämt avstånd. I de flesta fall använde jag 16 cm och mätte på föremål på 5 till 35 cm avstånd.

Ett annat sätt att bedöma avståndet till ett föremål är att titta på storleken på bilden av föremålet. Om man i förväg vet hur stort föremålet är kan man, om man har hela objektet i bild, göra en grov uppskattning av avståndet till det. Det erhållna felet blir cirka 25 %. Om föremålet är långt bort (mer än 30 cm) blir felet större eftersom upplösningen är begränsad och bredden kanske bara blir ett par pixlar. Denna metod är mycket snabb. Ingen förflyttning krävs. Det enda som krävs är att hela det aktuella föremålet är i bild.

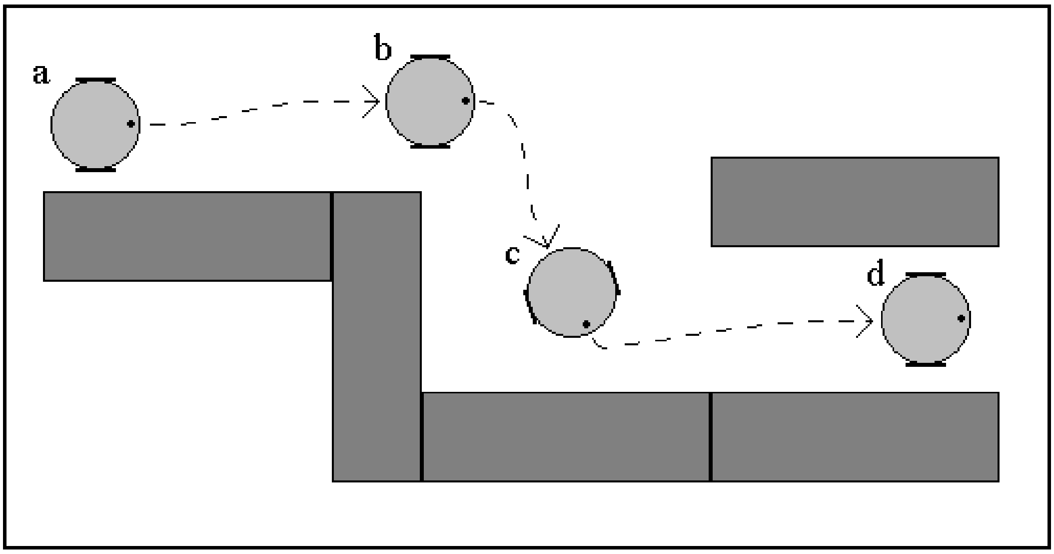
En lite mer generell metod som jag också provade är att närma sig ett föremål och på detta sätt studera den relativa ökningen av storlek hos objektet. Ett närbeläget föremål ökar snabbare i storlek än ett avlägset. Jag lät Kheperan fokusera föremålet, färdas 8 cm mot det och sedan fokusera det igen. Den relativa storleksökningen hos föremålet beräknades. Jämfört med den förstnämnda metoden är den här snabbare, den är heller inte lika beroende av motorernas och hjulens exakthet. Resultaten blev ändå nedslående. Kamerans begränsningar gjorde breddmätningen mycket osäker och eftersom varje bedömning innefattade två sådana fortplantade sig felet. Ett mindre fel kom av att roboten inte alltid färdades rakt mot objektet ifråga. Jag uppskattar från mina mätningar att felet även här är av storleksordningen 50 %. Återigen blir mätningar på avlägsna föremål ännu osäkrare.

Det gav alltså inga exakta resultat att med den här metoden bedöma avstånd med Kheperan. Märk dock att de uppmätta avstånden blir av rätt storleksordning, man kan alltså använda metoden för att avgöra om något är nära eller långt borta. För en snabb uppskattning av avstånd till föremål i omgivningen kan man använda två kameror. Utifrån parallaxen kan man beräkna avståndet på samma sätt som i mänskligt seende.

3.5 Väggföljning

I detta experiment använde jag mig av de infraröda sensorerna i stället för kameran. Målet var att få roboten att följa en vägg. Jag ville att den skulle klara av att följa såväl en högervägg som en vänstervägg samt kunna tackla hörn och andra trånga utrymmen. Jag valde att implementera det som ett slags beslutsträd, där den följde olika agerande beroende på sensorutslagen.

Att få Kheperan att följa en rak vägg som den redan står parallellt med bjöd inte på några större svårigheter. Det finns en sensor på varje sida. Gav denna för högt utslag (väggen nära) korrigerar roboten kursen från väggen (beroende på utsla–get), gav den för lågt (väggen långt borta) korrigerar den åt andra hållet. Eftersom sensorerna har en ganska begränsad räckvidd måste roboten hålla sig inom 2 cm från väggen. Skulle roboten helt tappa väggen ur "sikte" minns den på vilken sida den senast var och korrigerar kursen kraftigt ditåt. Detta fungerar även då väggen svänger bort från Kheperan. På den korrigerade kursen kan den snabbt finna väggen igen. Då sensorerna som var riktade framåt gav utslag (och roboten samtidigt hade en vägg på sin ena sida) korrigerar den med en kraftig sväng längs den nya väggen. Skulle Kheperan få väggar på båda sidor gäller särskilda regler för att roboten inte ska röra sig ryckigt eller fastna. Då roboten kommer nära den ena väggen skulle den normalt ha kompenserat genom att svänga bort från den. I stället uppmärksammar den väggen på andra sidan och försöker att färdas rakt, mitt emellan väggarna. När väggarna sedan skiljs, följer den en godtycklig vägg.



Figur 3.6: Exempel på hur Kheperan följer en vägg. Vi ser samma robot i flera situationer. I "a" har den hamnat för nära väggen. Sidosensorn känner detta och roboten korrigerar utåt. I "b" tappar roboten väggen ur sikte. Den korrigerar med en skarp högersväng och finner strax väggen igen. Vid "c" har Kheperan redan börjat svänga, då den skymtat väggen med de främre sen– sorerna. Den blir dock tvungen att göra ett tvärstopp och rotera för att inte kollidera med väggen. På väg mot "d" registrerar den också en vägg på vänster sida. Roboten minskar nu på margina– lerna och styr framåt i mitten av gången.

Jag ville uppnå en jämn rörelse hos roboten, helst utan stopp. I största möjliga mån lät jag den svänga medan den färdades framåt, och bara då den nästan kolliderade med en vägg stannade jag den och lät den rotera i stället. Kheperan kunde färdas relativt snabbt under dessa förhållanden. Onekligen är det här Kheperans riktiga styrka sitter. Irsensorerna ger snabba och pålitliga utslag. Betänk dock att väggarna i det här experimentet är av trä och vitt papper, som reflekterar det infraröda ljuset väl.

3.6 Bollhantering

Nu hade jag testat både närhetssensorerna och den linjära kameran. Nästa steg var att försöka kombinera dessa. En klassisk uppgift för robotar av den här typen är att lära dem spela fotboll. Till detta använde jag mig av en mörkgrön skumgum– miboll. Bollen var lika hög som roboten, vilket är en förutsättning för att den ska kunna ses av kameran. Den lämpade sig bra eftersom den gav bra kontraster och den matta ytan gav inga reflexer. Ett minus var dock att den inte var särskilt rund utan full av ojämnheter. För att bättre kunna knuffa på bollen klistrade jag på en liten platta längst fram på Kheperan med dubbelhäftande tejp. På detta sätt minskade jag friktionen mellan roboten och bollen som annars var ganska avsevärd och ställde till problem.

Kheperan sattes att rotera i vänstervarv tills bollen kom i sikte. Därpå satte roboten kurs mot bollen, en kurs som ständigt korrigerades så att bollen hamnade så nära mitten av synfältet som möjligt. Då roboten hunnit nära bollen fylls hela dess synfält upp av den (kameran hade ju bara en synvidd på 36°). Att roboten nu står och tittar på bollen och inte på en enfärgad vägg är näst intill omöjligt att avgöra utifrån kamerabilden. Lyckligtvis står den det avstånd från bollen som gör det möjligt för närhetssensorerna att känna av den. När de gör det, sätter Kheperan fart och knuffar till den. Att bollen nu inte går i en rak bana beror nog mest på dess ojämnhet.

I nästa steg ville jag att roboten skulle rikta skottet. Jag använde en svart pappers–pelare (samma som i avståndsmätningen) som "mål". Roboten skulle alltså skjuta bollen i riktning mot pelaren och med lite tur välta den. Det svåraste i den här delen var för roboten att skilja mellan boll och mål. Detta behandlades mer ingående ovan i kapitlet "identifiering". Det händer ibland att roboten tolkar målet som en avlägsen boll, dock tar den aldrig bollen för ett mål. Detta experiment ställde höga krav på belysningen i rummet. Bollsökningen går till på samma sätt som ovan. Då roboten når bollen roterar Kheperan trekvarts varv och söker efter målet. Ser den målet minns den riktningen. Finner den inte målet slår den bollen rakt fram (under antagandet att bollen skymmer målet), annars använder den närhetssensorerna för att runda bollen och sedan knuffa den i rätt riktning. Slutligen byggde jag in en finess som letade efter målet samtidigt som den sökte efter och närmade sig bollen. Såg den målet till vänster om bollen riktade den skottet i den riktningen, såg den målet till höger sköt den ditåt. Detta kunde också påverka hur den sökte efter målet då den nått bollen.

Bildserien nedan illustrerar ett möjligt scenario. Kheperan är ljusgrå, med de båda hjulen något överdimensionerade för tydlighetens skull. En prick visar framåt–riktningen. Pilarna indikerar hur roboten just rört sig. Bollen är mörkgrå (grön i verkligheten), och målet är en mindre svart, fylld cirkel.

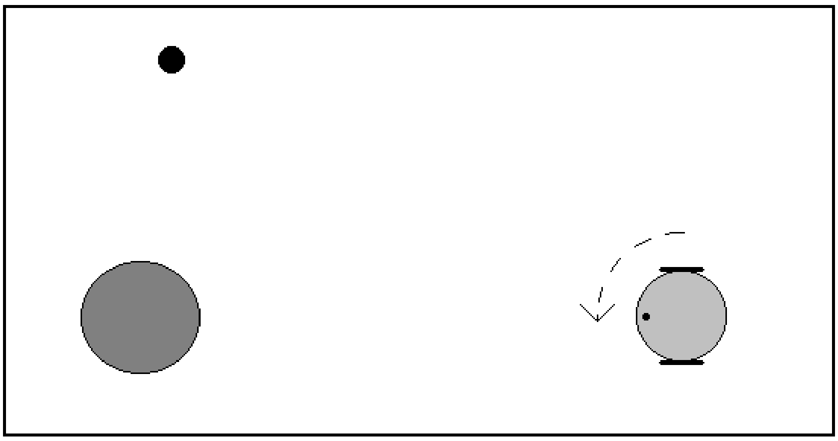


Fig. 3.7: Kheperan söker efter bollen och roterar i vänstervarv. På vägen ser den målet, och registrerar att det befinner sig till höger om bollen. Då den har bollen mitt i synfältet stannar den.

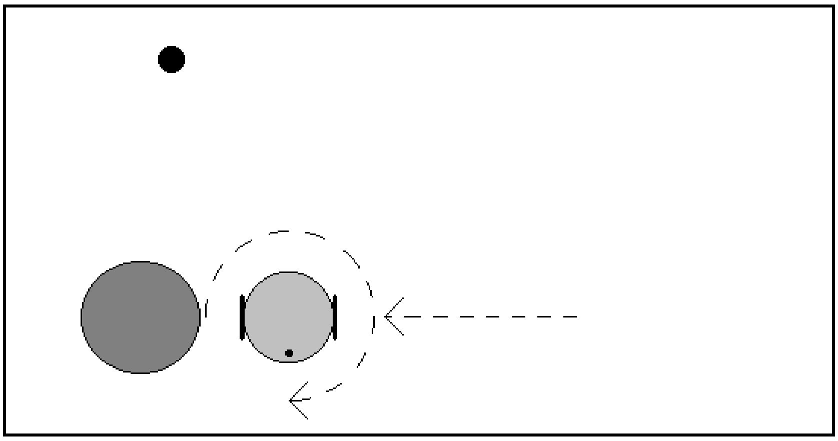


Fig. 3.8: Roboten närmar sig bollen och korrigerar kursen om den håller på att tappa bollen ur sikte. När närhetssensorerna känner av bollen stannar Kheperan. Roboten väljer nu att rotera i högervarv eftersom den tidigare sett målet till höger. Återigen ser roboten målet och registrerar vinkeln. Efter trekvarts varv stannar den.

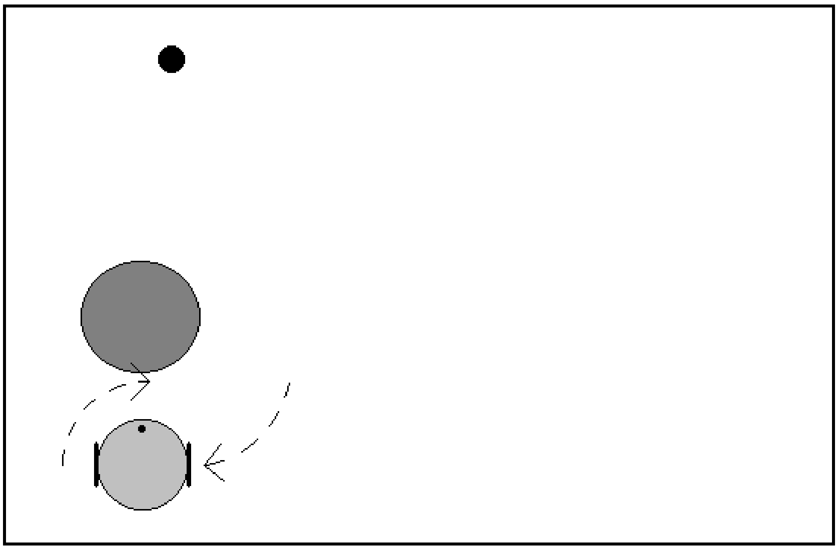


Fig. 3.9: Den rundar nu bollen och håller avståndet med hjälp av sensorn på sidan. Då den kommit i rätt position stannar den och vänder sig mot bollen.

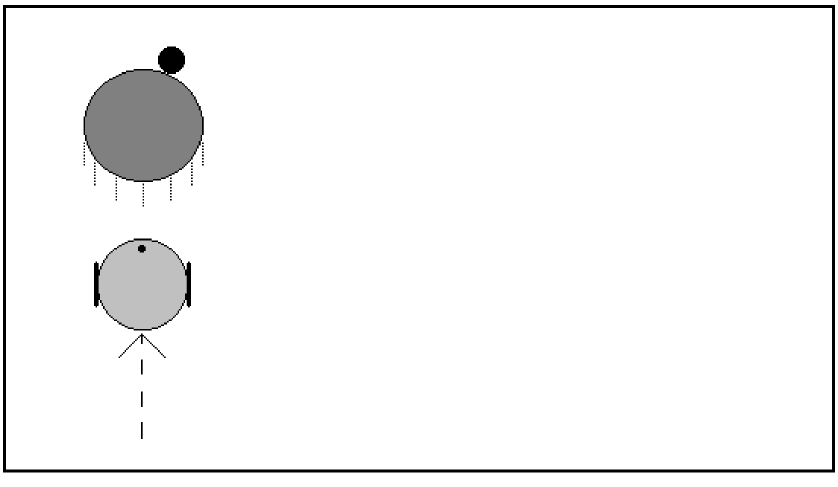
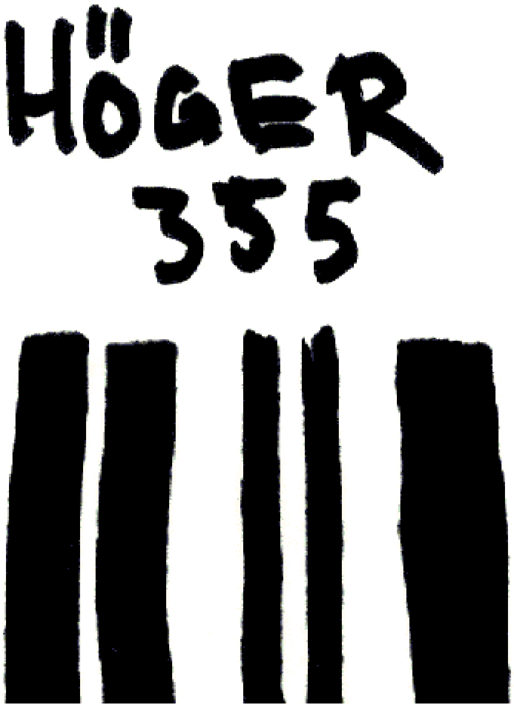


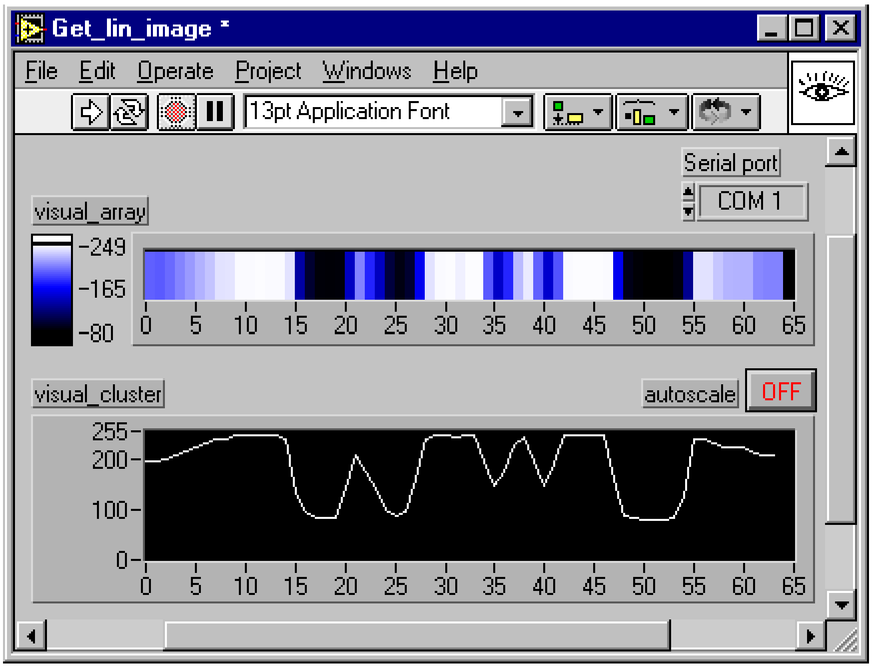
Fig. 3.10: Kheperan färdas nu rakt fram i hög hastighet för att ge bollen en knuff. När roboten stannar fortsätter bollen att rulla och träffar förhoppningsvis sitt mål.

3.7 Streckkodsläsning

De flesta av mina experiment hade hittills fallit ut väl. Jag tyckte dock inte riktigt att den linjära kameran kommit till sin rätt. Jag hade märkt att kameran bäst kunde urskilja intensitetsskiftningar i kamerans vågräta synfält. Jag beslöt därför att försöka använda den till att läsa streckkoder. Efter att har tittat på "vanliga" streckkoder genom kameran och läst på teorin bakom dem förstod jag att det skulle vara omöjligt för roboten att tyda en sådan. En "riktig" streckkod består av 12 siffror som var och en anges av sju svarta eller vita fält. Detta innebär att kameran måste kunna urskilja minst 84 fält, vilket är omöjligt med en upplösning på 64 pixlar. Man skulle kunna tänka sig att kameran läste delar av koden var för sig, men en streckkod är helt enkelt för detaljerad för att kunna läsas av roboten. Jag uppskattar att man skulle vara tvungen att förstora upp en streckkod till A4–format för att den skulle bli läsbar för kameran. I stället tillverkade jag egna koder som skulle översättas till binära tal. Koden består av omväxlande svarta och vita fält som är antingen smala (en binär nolla) eller breda (en binär etta). Här är det viktigt att fälten håller vissa mått. Ett för smalt streck kanske inte registreras av kameran, ett streck som är mitt emellan smalt och brett kan tolkas olika vid olika avläsningar. Hela koden ska vara mellan 4 och 4,5 cm bred. Ett smalt steck bör vara av storleksordningen 3 mm, ett brett måste vara större än 6 mm, men har ingen övre begränsning. Jag visade dessa koder för roboten och översatte dem sedan till olika kommandon, till exempel "sväng 90° åt höger" eller "gå 8 cm framåt".



Figur 3.11: Streckkoden såsom den ser ut i verkligheten. Läst från vänster blir den 1 0 1 1 0 0 0 1 1, vilket blir 355 i decimala siffror. Detta översätts i sin tur till en 90° högersväng.



Figur 3.12: Så här ser streckkoden i figur 3.11 ut i kameran.

Det är av viss vikt att koden hålls på rätt avstånd från kameran. Är den för nära kanske den kommer utanför bilden, är den för långt borta kan inte fälten urskiljas. Detta kontrollerade roboten genom att bara försöka sig på en avläsning då avståndet mellan ytterkanterna var inom vissa ramar. Ramarna kunde dock breddas lite genom att "förstora upp" en för liten streckkod. För att vara säker på att inte misstolka några koder accepterade jag bara en avläsning om den var densamma flera gånger i rad. När programmet avläser streckkoden söker jag inte efter konturer som i de övriga experimenten med kameran. I stället undersöker jag huruvida intensiteten för en pixel underskrider ett visst värde, i så fall är det ett svart fält. De översatta koderna tolkas helt enkelt i en vanlig "switch"–sats. Försöken föll väl ut, och Kheperan lydde. Jag utvecklade det hela genom att tillverka en kod som betydde "utför ej följande koder direkt, utan spara anvisningarna och vänta på startkod". På detta sätt kunde man "programmera" Kheperan, och sedan starta den med startkoden. För att visa att den lyckats tolka en instruktion lät jag den rotera kort fram och tillbaka.

I streckkodsläsningsförsöket kommer kameran verkligen till sin rätt. Den linjära kameran är bra på att uppfatta lodräta kontraster. Min hemmagjorda streckkod innehåller i sin nuvarande utformning nio binära siffror. Teoretiskt sett kan man alltså uppnå 512 olika kombinationer. Kombinationen med bara nollor utgår eftersom streckkoden då blir för smal, eventuellt bör man också undvika koder som lätt kan förväxlas. Med koder som kan förväxlas menar jag de som endast skiljer i en eller två binära siffror. Detta går att undvika genom att endast använda en utvald fjärdedel av talen. Jag beräknar att man kan konstruera drygt 100 olika koder utan risk för förväxling.

3.8 Korskompilatorn

I alla mina experiment exekveras själva programmet på värddatorn och kommandon skickas till Kheperan genom en seriell kabel. Roboten kan då inte räknas som 100%–igt autonom. Som jag antytt tidigare var autonomitet något jag ville uppnå. Jag utnyttjade mig då av den tidigare nämnda korskompileraren som genererar ett program som kan laddas ner och exekveras av Kheperans egen processor. Då exekveringen påbörjats kan man dra ur den anslutningskabeln och roboten blir helt autonom. När man skriver ett program som exekveras av Kheperan själv krävs lite andra rutiner. Till exempel måste man initiera funktioner innan de används. Felsökningen blir mycket komplicerad eftersom man inte på samma sätt kan se när programmet exekveras.

Jag skrev en förenklad variant av väggföljningsalgoritmen (baserad på ett med–följande exempelprogram) och lyckades ladda ner och exekvera denna. Roboten agerade som förväntat. Programmet kan lätt byggas ut till att motsvara den väggföljningsalgoritm jag tidigare beskrivit. Eftersom jag ansåg att detta nu var uppenbart beslöt jag att inte utveckla detta vidare utan koncentrerade mig i stället på andra frågeställningar.

Jag anser utan att faktiskt ha testat det att samtliga mina program kan laddas ner och exekveras på samma sätt. Då ett program körs av Kheperan, begränsas det ju av dess minne (RAM och ROM är båda 256 Kbytes) och hastighet (16 MHz). Inga av mina program tar så mycket minne i anspråk att de skulle påverkas av begränsningen. Vad hastigheten beträffar ligger flaskhalsen i kamerabilderna och deras behandling. Då kameran har en maxhastighet på 5 Hz och endast har 64 pixlar i 256 nyanser (varje pixel kan lagras i en byte) tycker jag att det är tydligt att detta inte är en begränsning.

4 Sammanfattning och diskussion

4.1 Vad som gjorts

Jag hade ganska fria händer i mitt arbete, något jag upplevde som positivt. Rikt–linjerna var få men idéerna och uppslagen var många. Den teori jag studerade var till viss hjälp, men vissa saker som till exempel den linjära kameran fanns det mycket litet skrivet om. De flesta robotar som fanns omnämnda hade dessutom viss form av djupseende med hjälp av två kameror. Robotarna var helt nya på institutionen, och även om vissa hade erfarenhet av robotar tidigare var det få som hade haft möjlighet att sätta sig in i Kheperorna. Min uppgift var att studera deras begränsningar och möjligheter med vikt på seendet. I ett inledande skede gjorde jag en kort bedömning av ”RoboWorld”. Ett väldesignat användargränssnitt tillåter användaren att simulera Kheperans basmodul. I stort sett fungerar det bra, även om vissa justeringar skulle öka realismen. Att simulera kameran i Robo–World är svårt, men säkert inte omöjligt.

Jag försökte göra experimenten med de ”riktiga” robotarna skilda i karaktär och så heltäckande som möjligt. Försöken föll ut med olika resultat och dessa kan kanske vara riktlinjer för uppgifter i kursen "AI för robotar". Särskilt nöjd är jag med väggföljningen och steckkodsläsningen som visar på de olika sensorerna styrka. Kameran passar bäst i en stiliserad miljö med tydliga kontraster och så lite brus som möjligt. Belysningen är speciellt viktig när man använder kameran. Bildbehandlingen (så som jag har gjort den) går snabbt och begränsas egentligen bara av kamerans hastighet. Att skilja mellan olika naturliga föremål är mycket svårt. Med min metod begränsas man till tre till fyra olika föremål om man vill behålla säkerheten. Avståndsbedömning med hjälp av kameran föll inte så väl ut. Osäkerheten är stor och orsakas till stor del av kamerans begränsningar. Här påverkas resultaten även av hjul och hjulräknare. De infraröda sensorerna är inte lika beroende av ljus och en enkel miljö. De fungerar alldeles utmärkt även i en tätt belamrad omgivning. Likaså är de snabba och kan göra många avläsningar per sekund. Deras svaghet ligger i den korta räckvidden. De räcker dock för att i god hastighet undvika föremål och följa väggar. Med korskompileraren kan roboten göras helt autonom. Ett program som skrivs för denna kräver lite andra rutiner. I den utsträckning som jag använde mig av korskompileraren lyckades jag åstadkomma vad jag gjort med den seriella kabeln och värddatorn i väggföljningsexperimentet.

4.2 Vad som kan göras

Jag skulle gärna velat fördjupa mig i flera av experimenten och utveckla dem ytterligare. Jag prioriterade dock att hinna med flera olika för att få ett mer heltäckande underlag. Några experiment skulle kunna kombineras. Till exempel kan man tänka sig en robot som aktiverar väggföljning då den visas en viss streckkod och bollhantering då den visas en annan. I stället för streckkoder skulle man eventuellt kunna använda tydliga attribut i omgivningen, kanske i kombina–tion med närhetssensorerna. En variant som jag inte alls testat är orientering med hjälp av landmärken, kanske i kombination med kartritning [19]. Jag studerade faktiskt möjligheterna men insåg att den linjära kamerans begränsningar skulle bli stora. Jag bedömer att igenkänning av olika landmärken är mycket svårt med den linjära kameran som dessutom är svartvit. Färgseende och höjdseende skulle betydligt underlätta denna uppgift. Även stereoseende skulle vara en god tillgång, men enligt min vetskap finns ännu inget sådant tillbehör till Kheperan. Slutligen har jag inte hunnit prova radiomodulen. Jag har sett att den fungerar och jag tror att den skulle vara ett mycket användbar till kommunikation mellan robotarna. Jag har räknat lite på kapaciteten och har kommit fram till att Kheperan skulle hinna med att skicka kamerabilder (fem i sekunden) via radion. Sannolikt skulle det dock påverka hastigheten. Desto tveksammare är det med en större kamera. Huruvida bilder från en sådan skulle kunna skickas via radio vill jag låta vara osagt.

Det finns onekligen många uppslag till fortsatta experiment med Kheperorna. Jag hoppas att mitt arbete kan vara till hjälp för de som vill laborera med robotarna.

5 Litteraturförteckning

[1] ”Objektorienterad Robotsimulator” av Henrik Lernmark, Examensarbete i datalogi mars 2000.

[2] ”Objektorienterad robotstyrning” och simulering i realtid av Tim Portnoff, Examensarbete i datalogi mars 2000.

[3] http://www.robocup.org/

[4] http://diwww.epfl.ch/lami/robots/

[5] http://www.k−team.com/

[6] ”Khepera, GNU C based cross−compiler for the Khepera robot” av K−Team, mars 1999.

[7] ”Khepera BIOS 5.0 Referens Manual” av Franzi Edo.

[8] ”Khepera, K213 Vision Turret User Manual av K−Team, mars 1999.

[9] ”Khepera, Radio Turret User Manual” av K−Team, mars 2000.

[10] ”Khepera, Radio Base User Manual” av K−Team, mars 2000.

[11] ”Khepera, User Manual”, appendix A av K−Team, november 1998.

[12] ”Mobile Robots, Inspiration to Implementattion” av Joseph L. Jones och Anita M. Flynn, 1993.

[13] ”The RoboCup−NAIST: A Cheap Multisensor−Based Robot with Visual Learning Capability” av T. Nakamuara, K. Terada, A. Shibata och H. Takeda.

[14] ”Development of Self−Learning Vision−Based Mobile Robots for Acquiring Robots Behaviors” av T. Nakamuara.

[15] ”Real−Time Vision Processing for a Soccer Playing Mobile Robot” av Gordon Cheng och Alexander Zelinsky.

[16] ”A $1000 Active Stereo Vision System” av Ian Horswill, November 1993.

[17] ”Understanding Object Motion: Recognition, Learning, and Spatiotemporal Reasoning” av M. Mohnhapt och B. Neumann.

[18] ”Artificial Intelligens, A Modern Approach” kapitel 18.3 av S. Russel och P. Norvig.

[19] ”Where am I?” av J. Borenstein, H. R. Everett och L. Feng.

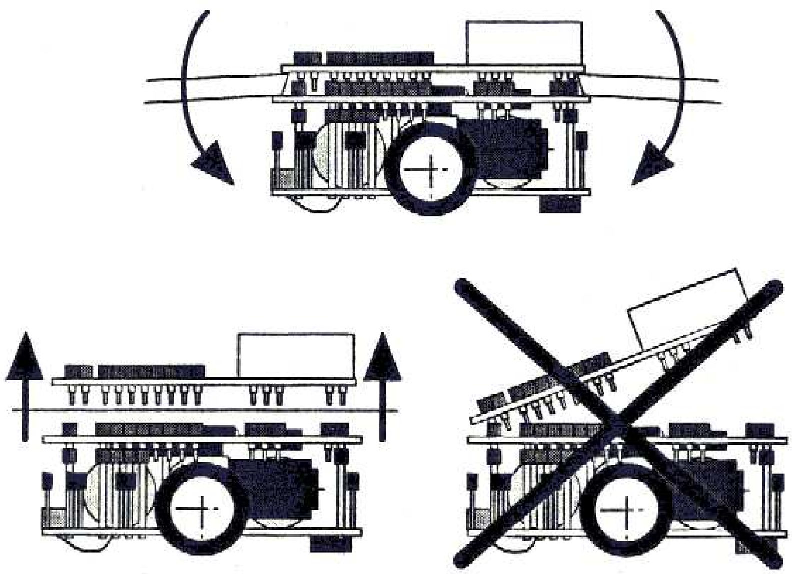
Appendix A – Handledning

Detta är tänkt som en kort manual med nödvändiga och praktiska tips för den som vill börja laborera med Khepera. Mycket av detta finns i Kheperans manual.

A.1 Viktigt!

Var rädd om roboten. Tänk på att den innehåller mycket känslig elektronik. När du håller i den, greppa den på sidorna och se till att du i varje fall håller den i basmodulen. Undvik att utsätta den för stötar och väta. Behöver du peta på den med något redskap, undvik metallföremål. Använd till exempel en plastskruv–mejsel.

Undvik om möjligt att ta isär modulerna. Varje gång du gör detta slits kontakter–na. Kontrollera om inte den andra roboten har rätt sammansättning. Måste du ändå göra det, iakttaga stor försiktighet. Försök att dra med samma kraft runt hela modulen. Använd gärna verktyg. När du sätter ihop två moduler, tryck försiktigt ner alla sidor samtidigt. Ta aldrig isär själva modulen utan speciell tillåtelse och rätta verktyg.



Figur A.1: Bänd försiktigt med plastverktyg i varje sida. Då modulen lossnar, lyft rakt upp.

Tvinga inte hjulet till rörelse. Undvik också att hindra roboten i sin rörelse eller att under en längre tid gå in i en vägg. Detta kan skada motorn. När du inte använder roboten, stäng av den. Dra även ut elförsörjningen till laddnings–interfacet, det kan annars bli mycket varmt. När du vill ladda upp robotens batterier se till att den inte är ansluten till något annat och att den är avstängd. Undvik att ladda upp fulla batterier, det kan orsaka skador. Om möjligt, låt roboten gå (eller rotera) tills batterierna är helt tömda.

A.2 Att komma igång

Det första du gör när du ska börja laborera med Kheperaroboten bör vara att studera sensorutslagen i LabVIEW. LabVIEW finns under NT på robotdatorn (Asimov). Kheperans applikationer finns under katalogen i LabVIEW märkt "Khepera". Prova att hålla föremål av olika färg och storlek på olika avstånd och studera hur sensorerna reagerar. Detta gäller såväl ir–sensorer som kameran. Detta ger en god och nödvändig förståelse för hur omvärlden ter sig för roboten.

När du sedan vill styra roboten bör du inte direkt använda korskompileraren. Som jag tidigare nämnt är möjligheterna till felsökning små. Använd i stället den seriella kabeln eller radiomodulen för att sända och ta emot information. Fördelen med detta är att du kan använda vilket programspråk du vill. Tänk dock på att om du i framtiden vill modifiera din kod och ladda ner programmet på roboten så att den blir självgående måste detta göras i C. Det går även att programmera Kheperan med LabVIEW men detta rekommenderas inte om man inte har tidigare erfarenhet inom något liknande. Korskompileraren är tillgänglig under LINUX. Läs i manualen om hur du använder den. Tänk på att du måste lägga in ditt eget program sist i ”Makefilen”.

A.3 Praktiska tips

Tycker du att roboten inte reagerar alls eller reagerar oväntat på ditt program, kontrollera först att den har strömförsörjning. Se till att kontaktbågarna är rätt inställda och att du har initierat roboten till ditt användningsområde. Kontrollera sedan att du nollställt stegräknarna till hjulen om du använder en metod som kräver detta.

När du experimenterar och roboten inte är frigående, se till att Kheperan har tillräckligt med kabel. En för kort kabel kan påverka hastighet, sträckor och rotationer och därmed ge felaktiga resultat. Se till att hålla experimentytan ren och likaså robotens hjul. Skräp kan annars påverka sträckor och riktningar. Ha alltid god och jämn belysning. Kameran kan tolka samma föremål olika i olika riktningar på grund av skuggor. Även ir–sensorerna påverkas av ljuset. Om du tycker att ir–sensorerna ger utslag fastän omgivningen är fri kan detta bero på att sensorerna har böjts. Jag har märkt att förvaringen i boxarna pressar sensorerna bakåt. Åtgärda problemet genom att mycket försiktigt böja dem utåt. Undvik att ta direkt på sensorytan.

Så länge roboten har ström utför den det senaste kommandot. Detta kan medföra ett problem. Se alltid till att avsluta dina program med att stanna roboten. Då ett program inte exekverar som man tänkt sig, det kanske hänger sig eller avbryts, kan detta resultera i att roboten far utom kontroll över bordet. Se till att alltid ha ett stopprogram redo som när det exekveras helt enkelt stannar och nollställer roboten. När jag använde kameran hade jag en funktion som när den registrerade mer än sex konturer stannade roboten och avbröt programmet. Sex konturer förekommer sällan naturligt, men genom att rita ett antal svarta streck på ett vitt papper och visa detta för kameran kunde man avbryta exekveringen utan oförutsedda konsekvenser.

När du inte längre ska använda Kheperan, lägg tillbaka den i sin förvaringsbox och lägg boxen på sin plats (i skrivandets stund i respektive nedersta låda i de båda skrivborden i robotlabbet). Här skyddas den mot stötar, dessutom blir det lätt för nästa användare att finna den. De olika handledningarna ska finnas i flera exemplar i labbet. Om inte ta ett par kopior på originalen. De ligger för tillfället i ett kuvert i den nedersta lådan i det vänstra skrivbordet.

Appendix B – Förkortningar

AI	Artificiell Intelligens
IR	InfraRött
BPS	Bits Per Second (Bitar Per Sekund)
RAM	Random Access Memory
ROM	Read Only Memory

Appendix C – Programkod

I detta stycke har jag bifogat programkoden för bollhanteringen. Den ivriga läsaren kan alltså direkt se hur ett fungerande program ser ut och hur olika problem tacklats. Av utrymmesskäl finns programkoden till de övriga experi– menten inte här. Dessa (tillsammans med bollhanteringen) kan nås via http://www.cs.lth.se/~ai.

Bollhantering

```
#include <stdio.h>
#include <errno.h>
#include <ctype.h>

/* Note: Before using this prog, use minicom to set up /dev/cua0, to
19200 baud. */


/* Globals */

FILE* dv;
char buf[1024];
int bufsize=0;
int a[64]; /* place to store the numbers...*/
int last_action=0;
int nbr_actions=0;
int search=0;
int stop=0;
int dir=0;
int goal;

/* Check for errors, clear given file after the check. */
void erch(FILE *f){
  if(errno){
    perror("ERROR");
    clearerr(f);
  }
}

/* Open the /dev/cua0 for khepera communication. */
void init_khep_comm(){
  if(!dv){
    if((dv=fopen("/dev/cua0","r+"))!=NULL) printf("Communication
channel is OPEN\n");
    else erch(dv);
  }
}

/* Sends a nullterminated string to the khepera,
   stores the returned string in the global buffer buf.
   Returns the number of characters valid in buf.
   buf is null-terminated.
*/
int say_to_khep(char *s){
  char *bufp;
  if(dv){
    fprintf(dv,s);
  }
  bufp=buf;
  while((*bufp++=getc(dv))!='\n')
    ;
  *bufp='\0';
  return bufsize=bufp-buf;
}

/* Nullyfies the wheelcounters. Allways call before step. */
void step_init(){
  say_to_khep("G,0,0\n");
}

/* Set the left motor to speed l, and the right to r.
   If info is true, the order is printed. */
void run(int l, int r, int info){
  char saybuf[256];
  if (info) {
    printf("Run:%i,%i\n",l,r);
```

```
  }
  sprintf(saybuf, "D,%i,%i\n\0", l, r); /* &saybuf??? */
  if (info) {
    printf("SAYBUF:%s", saybuf);
  }
  say_to_khep(saybuf);
}

/* Parse count comma-separated numbers from the string numbuf,
   writing them into array ary. */
int nums2ary(char *numbuf, int *ary, int count){
  int i=0, consumed=0;
  while(i<count){
    sscanf(numbuf, "%i%n",&ary[i++],&consumed);
    numbuf+=consumed+1;
  }
}

/* Detects edges using certain limits */
void edges(int *d2, int *edge_pos){
  #define SENSITIVITY 20 // Limit for detection
  #define ELIMINATION 9 // Minimum difference between levels
  #define TOLERANCE 16 // Limit to reduce flag
  int flag=0;
  int nbr_edges=0;
  int i;
  for(i=0; i<62; i++){
    if((abs(d2[i])>SENSITIVITY) && (flag==0)) {
      if (abs(a[i]-a[i+2])>ELIMINATION) {
            edge_pos[nbr_edges]=i+1;
            nbr_edges++;
            flag=2; // Flag must be zero for new edge to be detected.
      }
    }
    else if((abs(d2[i])<TOLERANCE) && (flag>0)) {
      flag--;
    }
  }
  printf("%i egdes detected\n", nbr_edges);
  edge_pos[9]=nbr_edges; // The last position is the number of edges.
}

/* Identify object using the egdes and their properties. */
int identify(int *d2, int *edge_pos){
  int id;
  int result;
  int i;
  int final=0; // Default is "nothing detected".
  int ball_pos=0;
  int nbr_edges=edge_pos[9];
  goal=0;
  if (nbr_edges==0) {
      printf("Nothing in sight.\n");
      return 0;
  }
  else if (nbr_edges==1) { // If only one edge is detected...
    result=rec1(d2, edge_pos[0]); // ...check with rec1
    if ((result>0) && (result<6)) {
      final=result;
    }
    return final;
  }
  else if (nbr_edges>1) { // If several edges are detected...
    for (i=1; i<nbr_edges; i++) { // ...check them pair by pair...
      result=rec2(edge_pos[i-1], edge_pos[i]); // ...using rec2.
      if (result==6) { // If nothing can be deducted...
            result=rec1(d2, edge_pos[i-1]); // ...check each edge...
            if ((nbr_edges==(i+1)) && (result==6)) {
              result=rec1(d2, edge_pos[i]); // ...using rec1.
            }
      }
      if ((result>0) && (result<6)) {
            if (final!=3) { // Some results are more accurate than
others.
              final=result;
              ball_pos=edge_pos[i-1];
            }
      }
    }
    if ((ball_pos) && (goal) && (!search)) { // If not in search
mode...
```

```
      if (ball_pos>goal) { // ...remember the relative position of the
goal.
            dir=-5;
      } else {
            dir=5;
      }
    }
    return final;
  }
}

/* Analyses one edge */
int rec1(int *d2, int pos) {
  int mid=0;
  if (a[pos-1]<a[pos+1]) { // Check the intesity around the edge.
    mid=(pos)/2;
    if ((a[mid]<(a[pos+1]-10)) && (d2[pos]<35)) { // Is it the ball?
      if (pos<28) { // Where is it?
            printf("Ball to the far left!\n");
            return 1;
      }
      else {
            printf("Ball to the left!\n");
      return 2;
      }
    } else if (d2[pos]>27) { // Is it the goal?
      printf("Goal in sight!\n");
      goal=pos;
      return 0;
    }
  }
  else {
    mid=(pos+62)/2;
    if ((a[mid]<(a[pos-1]-10)) && (d2[pos]<35)) {
      if ((pos>34) && (d2[pos]<35)) {
            printf("Ball to the far right!\n");
            return 5;
      }
      else {
            printf("Ball to the right!\n");
            return 4;
      }
    } else if (d2[pos]>27) {
      printf("Goal in sight!\n");
      goal=pos;
      return 0;
    }
  }
  return 6;
}

/* Analyses two edges together */
int rec2(int left_pos, int right_pos){
  int mid=0;
  int width=0;
  int left;
  int right;
  mid=(left_pos+right_pos)/2;
  width=right_pos-left_pos;
  left=right_pos>3 ? left_pos-3:0;
  right=right_pos<60 ? right_pos+3:63;
  if ((a[left]>a[mid]) && (a[mid]<a[right])) {
    if (width<12) {
      printf("Goal in sight!\n");
      goal=left_pos;
      return 0;
    }
    else if ((mid>26) && (mid<36)) {
      printf("Ball ahead!\n");
      return 3;
    }
    else if (mid<27) {
      printf("Ball to the left!\n");
      return 2;
    }
    else if (mid>35) {
      printf("Ball to the right!\n");
            return 4;
    }
  }
  else {
```

```
    printf("Too much disturbance...\n");
    return 6;
  }
}

/* Chooses an action depending on per and previous actions */
int action(int per){
  if (last_action==per) {
    nbr_actions++;
  }
  else {
    last_action=per;
    nbr_actions=1;
  }
  switch (per) {
  case 0: if (nbr_actions>3) { // If nothing is detected in 4 rounds...
    run(-1,1,1); // ...rotate (and keep looking).
  }
  else if (nbr_actions>1) { // If nothing if detected in 2 rounds...
    run(0,0,1); // ...stop (and check again).
  }
  break;
  case 1: run(3,4,1); break; // Choose a speed depending on...
  case 2: run(4,5,1); break; // ...where the ball is detected.
  case 3: run(5,5,1); break;
  case 4: run(5,4,1); break;
  case 5: run(4,3,1); break;
  case 6: if (nbr_actions>4) { // If there is too much disturbance for
5 rounds...
    run(-1,1,1); // ...start rotating.
  }break;
  default: break;
  }
}

/* Turns the Khepera angle degrees, right if direction is 0, left if it
is 1.
   If info is true, angle and direction is printed. */
int turn(int direction, int angle, int *d2, int *edge_pos, int info) {
  int finished=0;
  step_init();
  if (info) {
    printf("Direction: %i\n", direction);
    printf("Angle: %i\n", angle);
  }
  if (direction) {
    run(-2,2,0);
  } else {
    run(2,-2,0);
  }
  while(!finished) {
    if (search) { // If in search mode...
      goal=0;
      look(1); // ...look for goal.
      derive(d2,0);
      edges(d2, edge_pos);
      identify(d2, edge_pos);
    }
    say_to_khep("H\n");
    nums2ary(buf+2, a, 2);
    if (search && goal>20 && goal<42) { // If search and goal in
center...
      dir=(a[direction]*20/2048)+1; // ...set the direction.
      printf("a: %i\n",a[direction]);
    }
    if ((int)(a[direction])>(angle*2048/360)) {
      finished=1;
    }
  }
  run(0,0,0);
}

/* Puts the robot in position, and kicks! */
int kick(int *d2, int *edge_pos) {
  int prox[8];
  int right;
  int left;
  int finished=0;
  search=1; // Enter (goal) search mode.
  if (dir>=0) { // Chooses direction depending on previous goal
sightings.
```

```
    turn(0,270,d2,edge_pos,1); // Rotate and look.
    step_init();
    while(!finished) {
      say_to_khep("N\n");
      nums2ary(buf+2, prox, 8); // Skip first two characters.
      right=prox[5];
      if (right>400) { // Put Khepera in position...
            run(5,4,1);
      }
      else if (right<150) { // ...using proximity sensors...
            run(7,2,1);
      }
      else { // ...to keep the right distance.
            run(6,3,1);
      }
      if (prox[2]>500) { // If you detect something ahead, stop.
            finished=1;
      }
      say_to_khep("H\n");
      nums2ary(buf+2, a, 2);
      if ((int)(a[0])>(4150/10*dir)) {
            finished=1;
      }
    }
    search=0;
    turn(0,90,d2,edge_pos,1);
  }
  else {
    turn(1,270,d2,edge_pos,1);
    dir=abs(dir);
    step_init();
    while(!finished) {
      say_to_khep("N\n");
      nums2ary(buf+2, prox, 8);
      left=prox[0];
      if (left>400) {
            run(4,5,0);
      }
      else if (left<150) {
            run(2,7,0);
      }
      else {
            run(3,6,0);
      }
      if (prox[3]>500) {
            finished=1;
      }
      say_to_khep("H\n");
      nums2ary(buf+2, a, 2);
      if ((int)(a[1])>(4150/10*dir)) {
            finished=1;
      }
    }
    search=0;
    turn(1,90,d2,edge_pos,1);
  }
  finished=0;
  step_init();
  run(23,23,0); // Full speed ahead!
  while(!finished) {
    say_to_khep("H\n");
    nums2ary(buf+2, a, 2);
    if ((int)(a[1])>1000) {
      run(0,0,0);
      finished=1;
    }
  }
}

/* Collects the vision data from Khepera */
int look(int graphics) {
  int i;
  int y;
  int asize=0;
  int resolution=16;
  int maxvalue=256;
  char * bufp=0;
  say_to_khep("T,2,N\n");
  for(i=0, bufp=buf; *bufp++; i+=*bufp==','); // Counting commas.
  asize=i-2;
  nums2ary(buf+6, a, asize); // Skip first six characters.
```

```
  if (graphics) {
    for(y=resolution; y>0; y--){
      for(i=0; i<64; i++){
            printf("%c",(a[i]/(double)maxvalue>y/(double)resolution)?'_
':'*');
      }
      printf("\n");
    }
    printf("\n");
  }
}

/* Derives a[] twice, and puts the result in d2 */
int derive(int *d2, int info) {
  int d1[63]; /* derivate */
  int i;
  for(i=0; i<63; i++){ /* Diff once */
    d1[i]=a[i]-a[i+1];
  }
  for(i=0; i<62; i++){ /* Diff again */
    d2[i]=d1[i]-d1[i+1];
  }
  if (info) {
    for(i=0; i<62; i++){
      printf("%i,",d2[i]);
    }
    printf("\n");
  }
}

int main(int argc, char **argv){
  int prox[8];
  int proxsum=0;
  int d2[62];
  int *myD2=&d2[0];
  int edge_pos[10];
  int *myEdge_pos=&edge_pos[0];
  init_khep_comm();
  step_init();
  while(!stop){
    erch(dv);
    say_to_khep("N\n");
    nums2ary(buf+2, prox, 8); // Skip first two characters.
    proxsum=0;
    proxsum=prox[2]+prox[3];
    if (proxsum>500) {  // If there is an object close ahead...
      kick(myD2, myEdge_pos); // ...kick it!
      stop=1;
    } else if (proxsum>100) { // If there is something further on...
      run(2,2,0); // ...aproach it.
    } else { // Otherwise...
      look(1);
      derive(myD2,1);
      edges(myD2, myEdge_pos);
      action(identify(myD2, myEdge_pos)); // ...do what action tells
you.
    }
  }
  fclose(dv); // Finish nicely.
}
```