Sonar för en Khepera

Av
Fredrik Winge

Sammanfattning

En mängd olika typer av sensorer finns för robotar. Det här examensarbetet gick ut på att utrusta en Khepera robot med sonarer och testa systemets prestanda. Sonarn är speciellt använd i robotsammanhang för att det är ett enkelt och billigt sätt att upptäcka objekt en bit ifrån roboten. Sonarn fungerar så att den sänder ut en ultraljudssignal. När den träffar ett objekt reflekteras den tillbaka till sonarn. Tiden det tar för signalen att komma tillbaka anger hur långt ifrån objektet finns. Kheperan har annars bara ir-sensorer som kan upptäcka objekt på upp till fem centimeters avstånd. Men med SRF04 sonarn, som den utrustades med, kan den upptäcka objekt ända upp till tre meter bort. Det finns dock ett antal nackdelar med sonarn. Framförallt sänds ultraljudsignalen ut i en kon. Så objektet som upptäckts kan finnas varsomhelst inom en viss vinkel. En annan nackdel är att ultraljudssignalen lätt kan reflekteras åt olika håll, vilket ger felaktiga mätningar. Dessa nackdelar märktes i testerna som gjordes med den färdiga konstruktionen. De visades även i testet där en bana skulle kartläggas. Men i det testet visades också fördelarna. En robot med sonarer kan undersöka ett område utan att behöva besöka hela området fysiskt.

Abstract

There are many different types of sensors for robots. In this thesis the Khepera robot was to be equipped with sonars and then its performance tested. For robots the sonar is a commonly used sensor. That is because it provides an easy and cheap way to discover objects not in the direct vicinity of the robot. The sonar sends out an ultrasound signal. When it reaches an object it reflects back to the sonar. The time for the signal to return tells how far away the object is. The Khepera otherwise only has IR-sensors that can discover objects five centimeters away or less. But with the SRF04 sonar that the robot was equipped with, it can discover objects as far as three meters away. However there are some disadvantages with the sonar. In particular the ultrasound signal is sent out in a cone. So the object that is discovered can be anywhere within a certain angle. Another disadvantage is that the ultrasound signal can easily be reflected in different directions, which gives erroneous measurements. These disadvantages were noticed in tests that were made with the final solution. The disadvantages were also noticed in the test where an environment was to be mapped. However in that test the advantages were also noticed. A robot with sonars can examine an area without the need to physically visit the entire area.

Förord

Det här examensarbetet gjordes på institutionen för datavetenskap på Lunds tekniska högskola under höstterminen 2003 och delar av vårterminen 2004.

Jag, författaren till rapporten, går på civilingenjörsutbildningen i elektroteknik och har framförallt inriktat mig i programmering och digital elektronik. Därför passade det här examensarbetet mig bra. Det innehöll både elektronik när sonarerna skulle integreras med Kheperan och programmering när den färdiga konstruktionen skulle testas.

Jag vill framförallt tacka min handledare Jacek Malec, som har varit mycket hjälpsam under arbetets gång.

Innehållsförteckning

Kapitel 1

Bakgrund

Kheperan är en robot utvecklad av K-Team Corporation i Schweiz. Den har bl.a. använts på LTH i kursen artificiell intelligens för robotar. Studenterna har fått programmera roboten så att den har kunnat kartlägga omgivningen och kunnat hitta väg i en i princip godtycklig bana. Roboten är liten och behändig vilket gör att den är lätt att använda i undervisning och en del forskning som inte kräver stora robotar. Men den har också ett antal svagheter. En av dem är att sensorerna bara når några cm. Man skulle kunna säga att roboten är blind och får känna sig fram i omgivningen. Det här examensarbetet gick ut på att utrusta roboten med ljudsensorer så att den får möjligheten att känna av omgivningen på mycket längre avstånd. Lämplig sonar skulle letas upp, sedan skulle den fås att fungera tillsammans med roboten och slutligen skulle den nya konstruktionens styrkor och begränsningar testas.

1.1 Kheperan

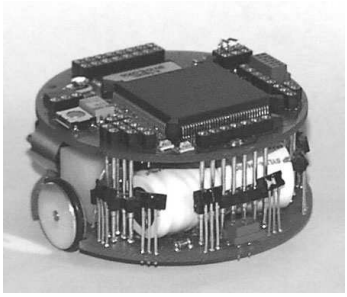

Bild 1.1 Kheperan

Följande beskrivning om Kheperan är baserade på manualerna [3, 4, 5, 7] och K-Teamets hemsida [14]. Kheperan är en liten robot som är 5.5 cm i diameter och 3 cm hög. Se bild 1.1 ovan. Robotens styrsystem är baserad på en Motorola 68331 processor på 16 MHz, 256 KB RAM. Roboten har åtta st. infraröda sensorer som når upp till fem cm, två hjul som drivs av DC-motorer och fyra nickel-cadmium batterier som räcker för ca 45 minuters drift.

DC-motorerna har en upplösning på 600 pulser per varv för hjulen, d.v.s. 12 pulser per mm. Hastigheten för roboten kan variera mellan 2 och 60 cm/s. Roboten vet var den är och är riktad genom att hålla reda på hur mycket hjulen har snurrat. Fram och bak på undersidan av roboten finns två små plastploppar som stödjer upp roboten eftersom den bara har två hjul. Om underlaget är jämnt fungerar det bra eftersom hjulen är gummiklädda och har bra grepp. Om däremot underlaget är ojämnt kan plastplopparna haka fast i underlaget. Roboten förlorar då snabbt uppfattningen om var den är och åt vilket håll den är riktad.

Roboten kan antingen köras helt autonomt genom att man laddar upp ett program till RAM minnet och låter batterierna stå för strömförsörjningen. Eller så kan den kopplas till seriellporten på en dator via en speciell adapter. Då kan både strömförsörjningen komma utifrån och programmet köras på den externa datorn. Nackdelen är då att räckvidden för roboten begränsas eftersom sladden bara är två meter lång. Roboten styrs utifrån genom att skicka ASCII-kommandon via seriellbussen. T.ex. skickar man;
D,vänster_motor_hastighet,höger_motor_hastighet
för att ställa in hastigheten på roboten. Se [7] för mer information. I ett program man laddar upp använder man sig istället av speciella bios kommandon för att styra roboten. Språket som används är C och för att kunna använda bios kommandona måste en korskompilator användas. För ytterliggare information om det se [3,5]. De flesta viktiga funktionerna kan användas med ASCII-kommandona, men inte alla. Ibland måste man ladda upp ett program för att få tillgång till alla funktionerna.

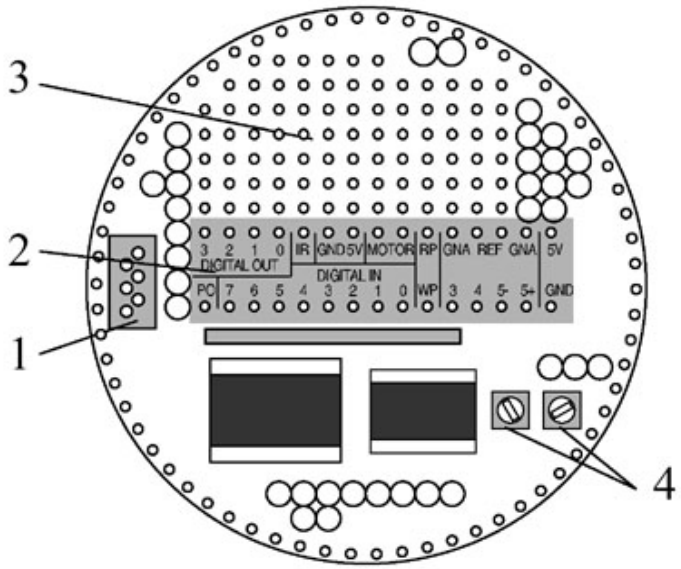

Bild 1.2 Skiss över I/O-Turreten: 1. Seriell kontakten 2. I/O kontakterna 3. Fritt område att använda 4.Reglering av Analoginsignalsförstärkningen.

Till Kheperan finns ett antal extra moduler eller ”turrets” som K-Teamet kallar dem. De sätts ovanpå roboten och vissa av dem kan kombineras så att flera kan användas samtidigt om bara den fysiska platsen finns. Det finns t.ex. en modul med griparm och en med kamera som kan användas tillsammans. Det här projektet baseras på I/O-Turreten. Det är en modul som är tänkt ska kunna användas för att på ett enkelt sätt kunna koppla på valfria elektroniska komponenter på roboten, som t.ex. en sonar. Bild 1.2 ovan är hämtad ur [4]. Den visar hur modulen ser ut ovanifrån och vilka ut- och insignaler som finns. Sett från robotens perspektiv finns det åtta stycken digitala enbitars ingångar, fyra stycken digitala enbitars utgångar, en avbrottsingång (IR) som genererar avbrott i processorn. Det finns läspuls och skrivpuls utgångar för att generera pulser (RP,WP), en utgång för strömförsörjning som kan stängas av eller sättas på (PO) och två utgångar för att styra en motor framåt eller bakåt. Sedan finns det även utgångar för +5 Volt och jord för digitala ändamål och jord och referensspänning (+4.096 V) för analoga ändamål (REF, GNA). De digitala utgångarna är lågeffektutgångar

och kan alltså inte användas för strömförsörjning till yttre komponenter. För att läsa och skriva till de digitala in- och utgångarna läser man resp. skriver till bussadressen 32. Läspuls och skrivpuls utsignalerna är höga tills läsning resp. skrivning till adress 36 sker. Då blir läspuls signalen låg under ca 250 ns och skrivpuls signalen blir låg under ca 200 ns. Slutligen kan I/O-Turreten högst leverera en ström på 200 mA. Så det begränsar valet av vilka elektriska komponenter som kan användas med I/O-Turreten.

1.2 Sensorer för avståndsbestämning

Precis som vi människor behöver sinnen, så behöver en robot sensorer för att kunna känna av och agera i sin omgivning. En sorts sensorer är de som behövs för att roboten ska kunna upptäcka och avgöra avstånd till objekt och på så sätt veta var de befinner sig i förhållande till dem. En mängd olika sådana sensorer finns för robotar. Följande beskrivningar om ett antal olika sensorer är baserade på [1,11,12].

1.2.1 Sonarn

Sonarn är den mest populära sensorn för mobila robotar. Den sänder ut en ultraljudssignal. När signalen träffar ett objekt reflekteras den tillbaka till sonarn. Avståndet kan då räknas ut ifrån hur lång tid det tar för signalen att komma tillbaka till sonarn. Om *d* är avståndet, *t* är tiden från signalen sändes till att den togs emot och *h* är hastigheten för ljudet, kan avståndet beräknas ur formeln $d=h/2*t$. Man delar avståndet med två eftersom signalen både ska fram till objektet och sen tillbaka till sonarn igen. Ljudets hastighet i luft vid 18°C är ca 342 m/s. Det gör att tiden *t* oftast är i storleksordningen millisekunder. En av anledningarna till sonarns popularitet är att det är mycket enklare och billigare att räkna tiden i millisekunder än om man t.ex. skulle använda en laser för avståndsberäkning, som räknar tiden i storleksordningen nanosekunder. En annan anledning är att sonarn är väl beprövad som sensor och att det finns många produkter på marknaden att välja mellan.

1.2.1.1 Egenskaper

Från sonarn sänds ljudet ut i princip i en halvsfär. Det gör att den mottagna ljudsignalen kan komma varsomhelst ifrån inom sfären. Men ljudsignalen är starkast rakt fram och har blivit så svag långt före de 90 graderna från en linje rakt fram i sfären att vanligtvis inga objekt kan upptäckas där. Det gör att objektet kan antas finnas inom en kon som är betydligt mindre än de 180 graderna som halvsfären upptar. Bild 1.3 nedan är hämtat från [11] och visar ett exempel på hur känsligheten för Polariod 600 sonarn påverkas av vinkeln.

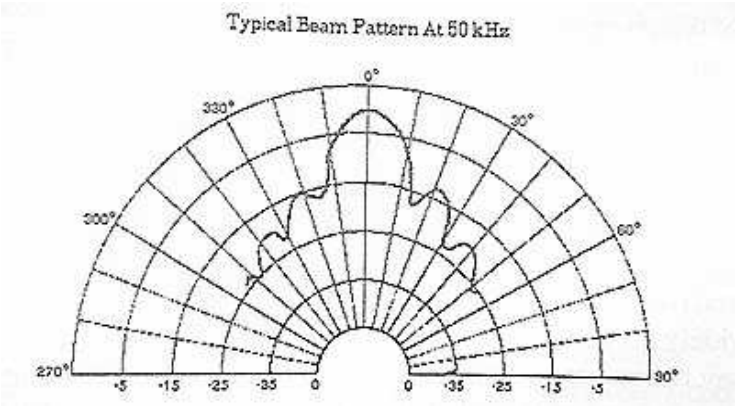

Bild 1.3 Känsligheten för Polaroid 600 sonarn i dB.

Vid en mätning brukar oftast konen delas in i tre olika områden. I det första området finns objektet någonstans. Sonarn kan inte avgöra avståndet exakt. Så det finns en osäkerhet både vad gäller avståndet och vinkeln till objektet. Det första området täcker hela området där objektet kan antas vara med tillräckligt stor sannolikhet. Det andra området är området mellan det första området och sonarn. Eftersom sonarn inte har registrerat något i det området är det antagligen tomt. Det tredje området är området bakom område ett. Där vet man inte om det finns något eller ej. I bild 1.4 nedan finns en skiss var områdena finns i konen.

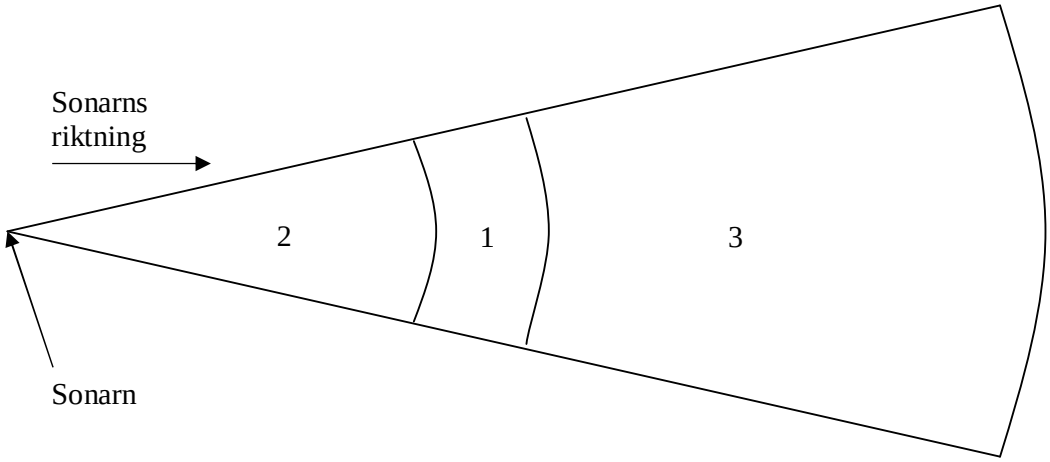

Bild 1.4 Sonarkonens olika områden.

1.2.1.2 Sonarns nackdelar

Det finns ett antal nackdelar med sonarn. För det första så finns det som diskuterats ovan en osäkerhet både vad gäller vinkeln och avståndet till objektet. Framförallt vinkelns osäkerhet gör att man vid en mätning egentligen enbart kan uppskatta avståndet till objektet. För att ta reda på objektets position måste flera mätningar göras från olika positioner. Olika algoritmer har tagits fram för att samordna mätningarna från sonarn och göra en karta över omgivningen utifrån dem. Se avsnitt 1.3 för mer information om det. Osäkerheten i avståndet beror dels på att ljudhastigheten varierar. Den beror både på temperatur och luftfuktighet. Så ljudhastigheten måste uppskattas på nytt inför varje ny mätning i ny miljö. Ännu mer problematiskt blir det om miljön varierar i tid och/eller rum. Avståndets osäkerhet beror även på fel som kan uppkomma då tiden fås fram. För en sonar är detta ett mindre problem än för t.ex. en laser som har betydligt kortare tider att handskas med.

Andra nackdelar illustreras i bild 1.5 nedan. De har att göra med hur ljudet reflekteras mot ytor. Om vinkeln till en platt yta är för stor, kommer ljudet inte kunna reflekteras tillbaka till sonarn. Det reflekteras då istället bort från sonarn ungefär som ljuset reflekteras mot en spegel. Det medför att objektet inte kommer att upptäckas. Dessutom kommer ljudet antagligen att reflekteras vidare tills det slutligen träffar sonarn efter ett antal reflektioner. Det medför att sonarn kommer att tro att det finns objekt där det egentligen inte finns något. Den ser alltså i syner. Bild 1.5 a visar hur signalen reflekteras bort från sonarn. Bild 1.5 b visar ett exempel på hur sonarn kan se i syner p.g.a. reflektionsproblemet. Vid hörn tror sonarn att det finns en vägg bakom hörnet (punktstreckade linjen).

Ett liknande problem är något som i litteraturen kallas cross-talk. Det inträffar då flera sonarer används samtidigt på en robot. Då kan signalen från en sonar reflekteras och träffa en annan sonar. Det medför att den andra sonarn kan upptäcka objekt som egentligen inte finns. Ett sätt att undvika det är att bara använda en sonar åt gången, vilket görs i det här projektet. Bild 1.5 c visar cross-talk problemet.

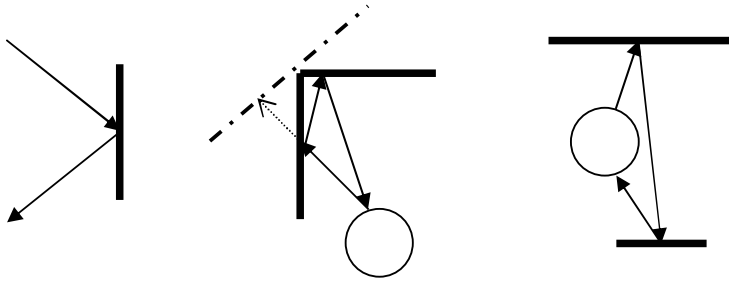

Bild 1.5 Nackdelar med sonarn, i a reflektion, i b ett exempel på hur den ser i syner p.g.a. reflektionen och i c s.k. ”cross-talk”.

1.2.2 Andra sensorer

En annan typ av sensor som känner om det finns objekt i närheten är ir-sensorn. Det är den typen av sensor som Khepera roboten är utrustad med i sin grundkonfiguration. Sensorn sänder ut infrarött ljus och mäter hur mycket som reflekteras tillbaka. Ju längre ifrån objektet är desto mindre ljus reflekteras tillbaka. Men räckvidden är inte speciellt lång, oftast i storleksordningen cm. Sedan beror även hur mycket ljus som reflekteras tillbaka på materialets färg och ytegenskaper. Så för att kunna avgöra avståndet måste man känna till de egenskaperna för objektet. Fördelen med den är att den inte upptäcker objekt som inte finns. Däremot kan den missa objekt som är för mörka.

Lasern fungerar på liknande sätt som sonarn. Tiden det tar för ljuset att återvända bestämmer avståndet. Däremot sänds ljuset inte ut i en kon utan i en punkt. Det gör att man vet direkt var objektet befinner sig. Den högre frekvensen gör också att risken för reflektion bort från sensorn (bild 1.5 a) är betydligt mindre. Men den höga hastigheten gör att tidmätningen blir betydligt svårare. Även att generera en laserpuls är mer komplicerat än att generera en ljudpuls. Så lasersensorn är betydligt dyrare än sonarn. Lasern kan även fås att svepa över ett område som elektronstrålen i en bildskärm vilket gör den ännu dyrare.

En annan sensor för att upptäcka objekt i omgivningen är känselspröt eller liknande. Då gör en mekanisk retning av känselsprötet att ett objekt har upptäckts. Två eller fler kameror kan även användas för att erhålla stereo seende. Nackdelen är att det tar lång tid att analysera en bild eftersom det krävs mycket beräkningar.

1.3 Kartläggningsalgoritmer

För att få fram en karta av omgivningen från sensordatan har en mängd olika algoritmer tagits fram. I [13] beskrivs översiktligt olika grupper av algoritmer. Det som alla har gemensamt är att de på något sätt använder sig av sannolikhetsteorin. Det beror på att sensordatan alltid har någon form av osäkerhet. Därför måste objekten på kartan även innehålla information om hur stor sannolikheten är att de verkligen finns där. Det gäller framförallt när olika data som är

motsägelsefulla ska samordnas. Att sensordatan är osäker beror både på osäkerheten i själva sensorn och på osäkerheten i var roboten befinner sig och vart den är riktad.

Speciellt en grupp av algoritmer som blivit väldigt populära är de som delar in världen i ett tvådimensionellt rutnätsmönster. Varje kartruta innehåller ett sannolikhetsvärde på om den är ockuperad eller inte. Det är enkelt att förstå eftersom kartrepresentationen påminner mycket om hur våra vanliga kartor är uppbyggda. Den har även andra fördelar. Kartan går att bygga upp efterhand vilket är viktigt om roboten själv ska göra kartan från grunden. Det blir också detaljerade kartor om rutorna är små. En nackdel är att det krävs att robotens riktning är känd vid varje sensoravläsning.

I [12] beskrivs tre olika algoritmer där kartan är uppbyggd som ett rutnät och där de är speciellt framtagna för sonarn som sensor. De är Bayesiska, Dempster-Shafer och HIMM (Histogrammic in Motion Mapping) algoritmerna. De första två är ganska lika. Enligt boken [12] är de även nästan identiska resultatmässigt. Så därför kommer Bayesiska algoritmen, som är lite enklare, att prövas på den färdiga lösningen.

1.3.1 Bayesiska algoritmen

Bayesiska algoritmen använder sig av Bayes regel från sannolikhetsteorin som är:

$$P(H \mid s) = \frac{P(s \mid H)P(H)}{P(s \mid H)P(H) + P(s \mid \neg H)P(\neg H)} \tag{1.1}$$

H är hypotesen ”ockuperad” och *s* är sensoravläsningen. På det sättet kan gamla sensoravläsningar kombineras med nya eftersom *P(H|s)* anger sannolikheten att en viss ruta är ockuperad efter nuvarande sensoravläsning, *P(H)* anger det gamla värdet på sannolikheten att rutan är ockuperad och *P(s|H)* är en ny beräkning på sannolikheten att rutan är ockuperad från informationen från den senaste sensoravläsningen. *P(s|H)* tar alltså inte hänsyn till gamla sensoravläsningar. $\neg$ är tecknet för negation.

P(s|H) fås fram ur formler som beror på i vilken region i sonarkonen rutan befinner sig i. Se avsnitt 1.2.1 och figur 1.1 för mer information om de olika regionerna. För region ett är formlerna:

$$P(s \mid H) = \frac{\frac{R-r}{R} + \frac{\beta-\alpha}{\beta}}{2} \times Max_{occupied} \tag{1.2}$$

$$P(s \mid \neg H) = 1 - P(s \mid H) \tag{1.3}$$

R och β anger sonarns maximala räckvidd resp. maximala vinkeln den kan upptäcka objekt på. *r* och α anger avstånd resp. vinkel till rutan som beräknas för tillfället. $Max_{occupied}$ används om man vill att sannolikheten inte ska kunna bli ett. Det är alltså ett tal större än noll och mindre än ett. Man ser att ju längre ifrån rutan är desto mindre är sannolikheten att rutan innehåller något. Även ju större vinkeln är desto mindre är sannolikheten att rutan innehåller något. Det beror ju på att sonarn har mindre chans att upptäcka objekt ju längre ifrån och ju

större vinkeln är. För region två är formlerna istället det omvända förutom att $Max_{occupied}$ inte används:

$$P(s \mid H) = 1 - P(s \mid \neg H) \tag{1.4}$$

$$P(s \mid \neg H) = \frac{\frac{R-r}{R} + \frac{\beta - \alpha}{\beta}}{2} \tag{1.5}$$

Region tre påverkas inte eftersom man inte vet något om det området.

1.3.2 HIMM algoritmen

För HIMM algoritmen innehåller kartrutorna ett värde på mellan noll och femton. Varje kartruta representeras alltså bara av fyra bitar, jämfört med Bayesiska algoritmens kartrutor som vanligtvis representeras av 64 bitar (double). För att förenkla kartuppdateringen antar HIMM algoritmen att enbart rutorna rakt fram från sonarn påverkas, eftersom sonarn har starkast signal rakt fram. Beräkningarna för varje kartruta är också väldigt enkla. Rutorna i sonarkonens område ett uppdateras med plus tre och rutorna i sonarkonens område två uppdateras med minus ett. HIMM algoritmen uppdaterar alltså både färre rutor än Bayesiska algoritmen och beräkningarna för varje ruta är betydligt enklare. De tillsammans medför att HIMM algoritmen både är betydligt snabbare och tar mindre minne än Bayesiska algoritmen. Men det medför också att det blir mer fel med den algoritmen. Därför lämpar den sig inte till att pröva hur bra sonarn är för kartläggning. Istället kommer en modifierad variant av HIMM algoritmen att användas för kartläggningen. Mer om den går att läsa i avsnitt 3.2. Bild 1.6 nedan är hämtad från [12] och illustrerar hur HIMM algoritmen uppdaterar kartrutorna.

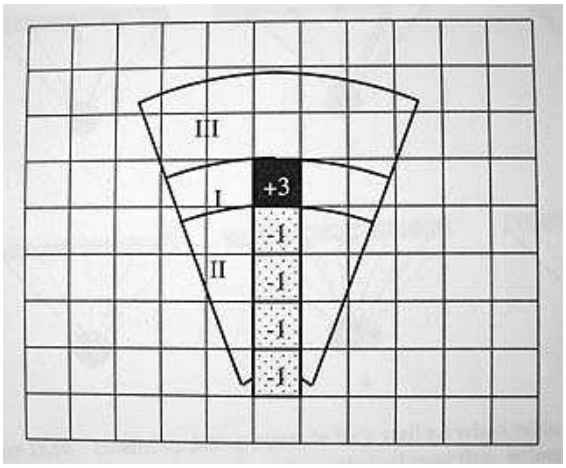

Bild 1.6 Illustration hur HIMM algoritmen uppdaterar kartrutorna.

Kapitel 2

Implementation

Själva arbetet kan delas upp i att först hitta en passande sonar för roboten. Sedan ska sonarn fås fungera tillsammans med roboten. Slutligen ska konstruktionen testas.

2.1 Vilken sonar?

Det finns en mängd olika sonarer på marknaden. Det finns både möjligheten att köpa sonarpulssändaren för sig och då bygga elektroniken omkring själv. Eller så kan man köpa färdiga system att använda direkt. Fördelen med de färdiga systemen är att det mesta redan är färdigt, testat och ska fungera. Däremot kan man själv inte skräddarsy dem för sina egna ändamål och det är inte säkert att gränssnittet mellan sonarsystemet och roboten passar ihop. Men om det går bra så är färdiga system att föredra.

I litteraturen nämns speciellt ett färdigt system som används mycket till robotar. Det är Polaroid ultrasonic ranging module [15]. Den består av en färdig modul som driver en ultraljudssändare/mottagare. Den har en räckvidd på 10.7 meter. Eftersom sändaren även används som mottagare måste den hinna sluta svänga av innan det går att ta emot några signaler. Det gör att minsta mätavståndet är 20 cm. Det är en nackdel för en liten robot som Kheperan som oftast arbetar med korta avstånd och som inte behöver så stora avstånd att mäta som 10.7 meter. Polariodmodulen drar också mycket ström, ungefär 100 mA i genomsnitt i viloläge och så mycket som 2000 mA när den sänder. Kheperan har en maximal strömförsörjning på 200 mA vilket innebär att den bara kan utrustas med maximalt en sonar, samt att det antagligen behövs en extra buffert för att klara av den höga sändningsströmmen. Polariods lösning passar alltså mer in på större robotar än en liten som Kheperan.

Devantech har två färdiga sonarsystem som är byggda för mindre robotar [16]. Det är SRF04 och den nyare SRF08. Skillnaderna mellan dem är främst att SRF08 har längre räckvidd, 6 meter istället för SRF04:ans 3 meter, den drar mindre ström, 15 mA istället för 30 mA i genomsnitt och den använder sig av I2C bussen för att kommunicera med omvärlden. SRF04:an kommunicerar istället med ett eget system som beskrivs mer detaljerat nedan. SRF08:an är alltså bättre i vissa avseenden. Men Kheperan har inget I2C gränssnitt så i så fall måste en mjukvaruversion av I2C bussen utvecklas på Kheperan för att kunna använda SRF08:an. Det verkar fullt möjligt om man tittar på I2C buss specifikationen. Men det verkar bli mer jobb och arbetet skulle då nästan enbart handla om programmering. När jag valde det här examensarbetet hoppades jag ju att det både skulle handla om elektronik och programmering. Därför valdes SRF04:an för vidare experiment.

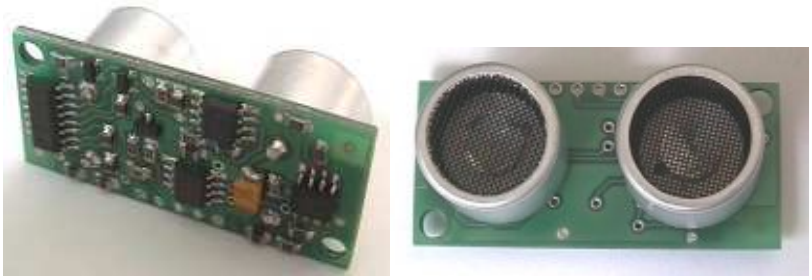

Bild 2.1 SRF04:an.

Bild 2.1 ovan, som är hämtad från Devantechs hemsida [16], visar SRF04 sonarn från olika vinklar. Den använder sig av TTL nivåer precis som Kheperan. Som man kan se på bilderna har sonarn både en sändare och en mottagare. Det gör att avståndsbestämningen speciellt på nära avstånd försämras. Men det gör också att den inte behöver vänta på att sändaren ska sluta svänga. Därför är minsta avståndet den kan upptäcka objekt på så låg som tre cm. Maximala avståndet är tre meter och angivna upplösningen är 2-3 cm. Ultraljudssignalen har en frekvens på 40 kHz. Modulen är ganska liten, 43x20x17 mm, så den får bra plats på Kheperan som är 55 mm bred. Strömförsörjningen är också låg, typiskt 30 mA eller maximalt 50 mA i genomsnitt. En annan detalj som är värd att nämna är att sändar- och mottagarsensorerna är avskärmade av ett hölje. Det verkar göra att maximala vinkeln med vilket den kan upptäcka objekt begränsas till ett visst värde. Det kan tyckas vara en försämring. Men det är en fördel för de algoritmer som ska tolka sensordatan eftersom det då är säkert att objektet inte befinner sig utanför en viss vinkel. T.ex. fås ett exakt värde på β i formel 1.2 och 1.5 i avsnitt 1.3.1, istället för att anta att inget objekt kan upptäckas efter en viss vinkel.

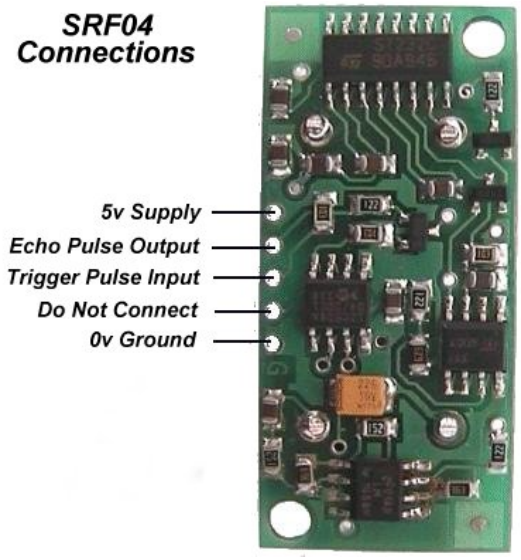

Bild 2.2 SRF04:ans ut- och insignaler.

Bild 2.2 ovan, som är från Devantechs hemsida [16], visar sonarns ut- och insignaler. För att göra en mätning sätts Trigger pulse input (TPI) hög under minst 10 µs. Vid neråtgående flank kommer sedan sonarpulsen att skickas ut. Echo pulse output (EPO) går samtidigt från låg till hög och förblir så tills ultraljudssignalen reflekterats tillbaka efter maximalt 18 ms (tre meter) eller tills 36 ms har gått. Upptäcks inget objekt blir utsignalen hög under 36 ms. Innan nästa

sonarpuls kan sändas måste det gå minst tio ms. Bild 2.3 nedan, som också har hämtats från Devantechs hemsida [16], visar detta. Första signalen är TPI, nästa visar ultraljudssignalen och den nedersta signalen är EPO.

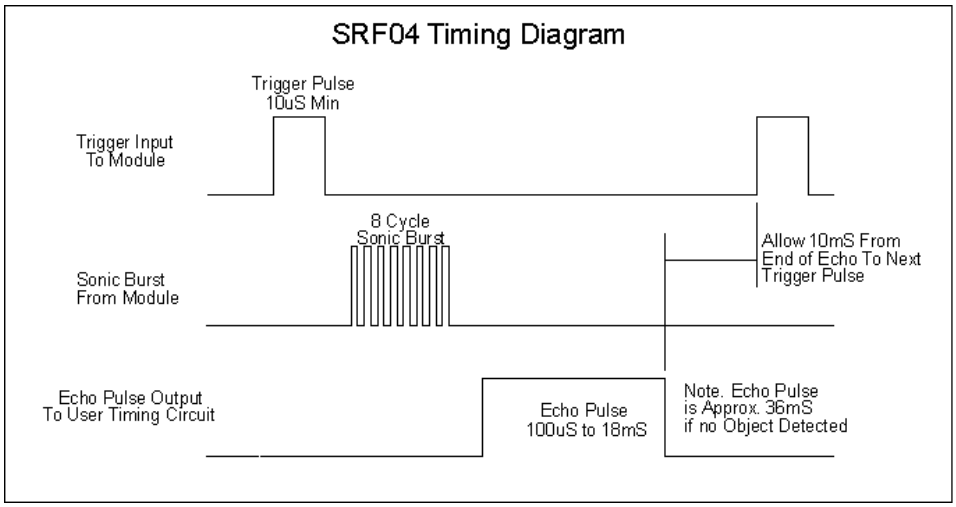

Bild 2.3 Beskrivning av hur man kommunicerar med SRF04 sonarn.

Det gör att kommunikationen med SRF04 sonarn är ganska enkel. Även övriga egenskaper som att den har låg strömförbrukning, liten fysisk storlek, att den kan mäta även korta avstånd och att den använder sig av TTL nivåer precis som Kheperan gör att det är den sonarn som passar bäst till roboten.

2.2 Första lösningen

Hur fås då SRF04 sonarn att fungera tillsammans med Kheperan? Jag hade tre olika förslag:

1) EPO-signalen kopplas till avbrottsinsignalen på IO-Turreten. I avbrottsrutinen som startas vid sonaravläsning läses kontinuerligt EPO-signalen av från en av de digitala ingångarna på I/O-Turreten. Sedan mäts tiden från start av avbrottsrutinen tills insignalen blivit låg igen vilket ger avståndet. Nackdelen är att avbrottsrutinen enbart kan hanteras med bios kommandona. Så då kan man inte göra en sonaravläsning från ett externt program. Den digitala ingången måste även läsas av hela tiden, vilket gör att inget annat kan göras på roboten under tiden.
2) Det kan stället ordnas så att två olika avbrottssignaler genereras. En när EPO går hög och en när EPO går låg igen. Tiden mellan avbrotten ger avståndet. Det blir som i 1) ovan förutom att roboten är fri att göra annat under tiden som den väntar på att sonarpulsen ska komma tillbaka.
3) Det tredje alternativet är att undvika att använda sig av avbrottet alls. Om EPO-signalen läses av kontinuerligt hela tiden efter att en sonaravläsning startat, kan man mäta hur lång tid EPO är hög, vilket ger avståndet. Det blir liknande 1) ovan förutom att roboten måste stanna upp ytterligare en aning tidigare och inte göra annat än att läsa av digitala insignalen. Men en sonaravläsningen kan då göras från externa program.

2.2.1 Elektroniken

Alternativ två ovan verkar bäst eftersom roboten är fri att göra annat under mätningen. Antagligen blir även tiden mer exakt med den metoden eftersom avbrottet genereras direkt när EPO-signalen ändras. Därför valdes alternativ två. För att kunna få en avbrottssignal för uppgåendeflank och en för nedgående flank på EPO kan en lösning enligt bild 2.4 nedan användas. Flip-floparna (FF) är minneselement som uppdateras av en klocksignal (CLK). Så de gör att det även behövs en klocksignal. Avbrottsinsignalen (IR) på I/O-Turreten är aktiv låg, vilket gör att en inverterare (NOT) behövs. Med kopplingen nedan kommer två pulser genereras av EPOs uppgående och nergående flank. Längden på pulserna är en klockcykel.

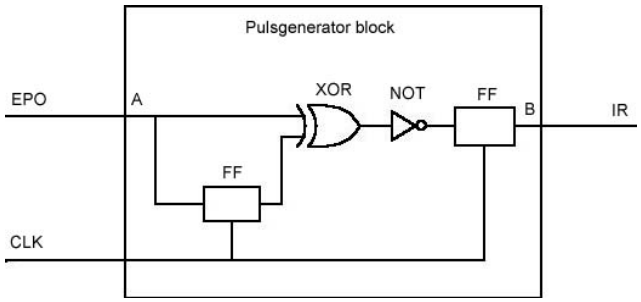

Bild 2.4 Koppling för att få två avbrottssignaler från EPO-signalen.

Eftersom strömförsörjningen bara är ca 30 mA i genomsnitt skulle flera sonarer kunna användas. Problemet är att bara en avbrottsinsignal finns att tillgå. Så därför kan bara en sonar användas åt gången. Eftersom flera sonarer även drar mer ström gör det att drifttiden minskar om den körs med enbart batteri. Det finns trots det en fördel med fler än en sonar. Det gör att roboten inte behöver vända sig lika mycket om den ska söka av området omkring sig och den sparar då tid. Två stycken sonarer är en bra kompromiss. Så det bestämdes att konstruktionen skulle ha en sonar fram och en bak.

Hur fås då två sonarer att fungera tillsammans med roboten? Bild 2.5 nedan visar hur det kan åstadkommas. Pulsgeneratorblocket är precis detsamma som kopplingen i bild 2.4. I/O-Turreten har fyra digitala utgångar. En av dem styr multiplexorn (MUX), två andra kopplas till sonarernas TPI och styr alltså när sonaravläsning ska starta. Med multiplexorn väljs vilken av EPO utgångarna från de två sonarerna som ska skickas vidare till pulsgeneratorblocket.

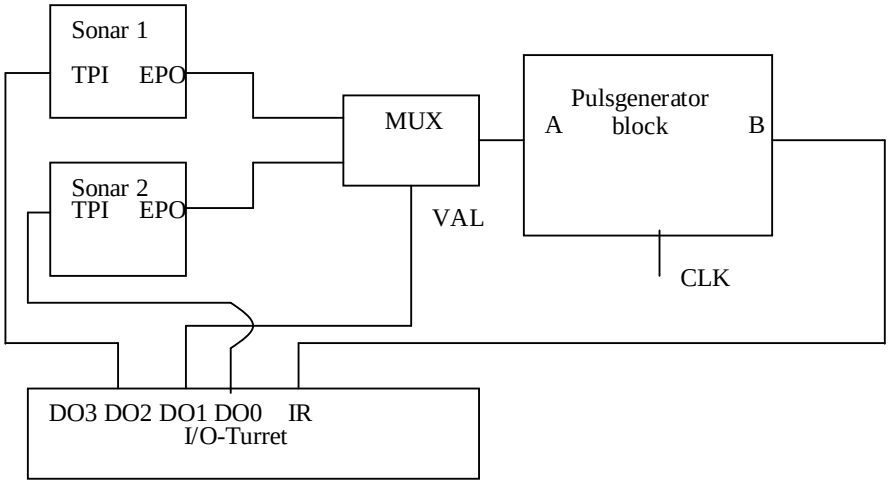

Bild 2.5 Koppling för att få två sonarer att fungera ihop med Kheperan.

Det gör att det behövs två flip-flopar, en multiplexor, en inverterare och en XOR grind. D.v.s. en IC för varje komponent typ. För att minimera strömåtgången och utnyttjandet av ytan kan XOR grinden (det utnyttjas bara ¼ av ICn) ersättas med en ledig multiplexor. Bild 2.6 nedan visar hur det går till. Två st. fyraingångars multiplexrar används. I komponenten så används samma väljare, A och B, till båda multiplexorna. Därför var det tvunget att dubblera val av EPO i den första multiplexorn. I första multiplexorn väljer B från vilken sonar EPO-signalen väljs, medan A inte har någon betydelse. I andra multiplexorn är det istället A som väljer om EPO-signalen ska vara inverterad eller ej, medan B inte har någon betydelse. Det gör att andra multiplexorn hela tiden väljer en nolla som utsignal. Men eftersom signalen till ingångarna är fördröjd en klockcykel mer än till väljaren A, kommer en etta att väljas under en klockcykel varje gång EPO-signalen ändras. T.ex. kommer en etta på B och en nolla på A att välja ingång 10 på båda multiplexrarna. Med den konstruktionen ska Kheperan kunna kommunicera med sonarerna.

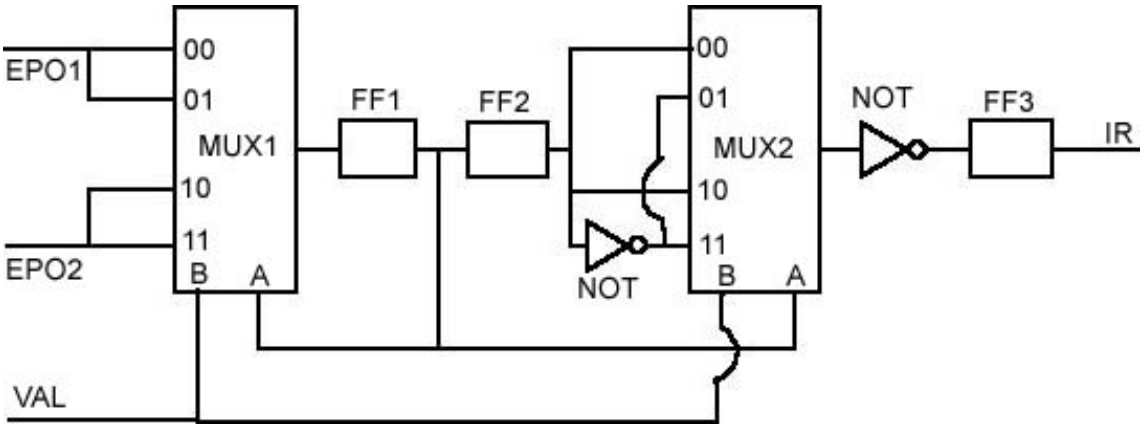

Bild 2.6 Kopplingen med 2 multiplexrar istället för en multiplexor plus en XOR grind.

2.2.2 Klocksignalen

Då framkom ett problem. Bioskommandot tm_get_ticcount() [3] var tänkt att användas för att mäta tiden. Men den ger bara nuvarande systemtid i millisekunder. Eftersom tre meter för sonarn ungefär motsvarar 18 ms blir bara upplösningen då 300/18≈17 cm, vilket är betydligt sämre än de 2-3 cm som sonarn ska klara av. För att få den upplösningen borde tiden kunna räknas i ungefär 300/3=100 st. steg istället för 18. Det finns åtta st. digitala ingångar på I/O-Turreten som skulle kunna användas för att läsa av en extern räknare. De skulle då kunna läsa av upp till 2^8=256 nivåer, vilket räcker bra. Det skulle dock innebära att en räknare som åtminstone kan räkna upp till 100 behövs läggas till, en klocka till den, logik för att starta och nollställa räknaren samt något som kan göra om klocksignalen till rätt frekvens (18 ms för 100 steg ger en frekvens på 100/0.018≈5556 Hz). En betydligt enklare lösning, om större upplösning behövs, är att istället göra flera mätningar och tar sedan medelvärdet av dem. Det skulle vara fullt möjligt såvida det inte finns ett beroende mellan när avbrottsrutinen startar och systemtiden. Alltså valdes det senare alternativet.

Då återstår bara problemet med var klocksignalen som ska användas till flip-floparna ska komma ifrån. På I/O-Turreten finns en klocksignal som Kheperan använder internt för att kommunicera med modulerna. Men ingen klocksignal kunde mätas där. I [6] står det att klocksignalen enbart används när Kheperan kommunicerar med modulen och är avstängd annars. Alltså går den inte att använda. Därför fick en mjukvaruklocka duga tills vidare.

Skrivpuls utsignalen (WP) på I/O-Turreten är hög tills man skriver till adress 36 [4]. Då går den låg under 200 ns och går sedan hög igen. Genom att låta en process köra en loop som hela tiden skriver till adress 36 får man en klocksignal, fast med asymmetrisk fyllning.

2.2.3 Den färdiga konstruktionen

Komponenterna som valdes till kopplingen var 74LS175, vilket är en fyra grindars flip-flop som även har negerade utgångar och 74LS153, som är en två grindars mux med fyra ingångar var. LS står för ”low power Schottky”. De är snabba, har 9 ns fördröjningstid och drar bara 2 mW eller 0.4 mA per grind. Senare ska de bytas ut mot HCT varianten istället som bygger på CMOS teknologi och drar ännu mindre ström. På I/O-Turreten kopplades digital out 0 till IPO på första sonarn och digital out 2 till IPO på andra sonarn. Digital out 1 användes till väljarsignalen (VAL). Så för att generera en signal för första sonarn så skriver man en etta (0001) till adress 32 och skriver kort därefter en nolla (0000) till samma adress. Den adressen har hand om de digitala in- och utgångarna på I/O-Turreten. För andra sonarn skriver man istället sex (0110) resp. två (0010) till adress 32. Utgången från den egna elektroniken kopplades till avbrottsingången (IR) och slutligen kopplades Skrivpuls utsignalen (WP) till klockinsignalen på flip-floparna. Se [6] samt avsnitt 1.1.1 och bild 1.2 för mer information. Bild 2.7 nedan visar hur första lösningen såg ut. På bilden kan även en transistor och ett motstånd anas. De användes som en väldigt enkel förstärkare till klocksignalen. Anledningen till det var att när konstruktionen testades så dog klockan varje gång avbrottssignalen gick låg, vilket medförde att avbrottsrutinen kördes om och om igen. Eftersom klocksignalen är en lågeffektutgång så kunde det bero på att den inte klarade att driva flip-floparna, varför en förstärkare kanske skulle behövas. Men det visade sig att det berodde på att avbrottsrutinen har företräde framför alla andra processer inklusive klockgenerator processen. Så det var alltså avbrottsrutinen som stängde av klocksignalen. Genom att generera en extra klockcykel i själva avbrottsrutinen löstes problemet och alltså behövdes egentligen inte förstärkaren.

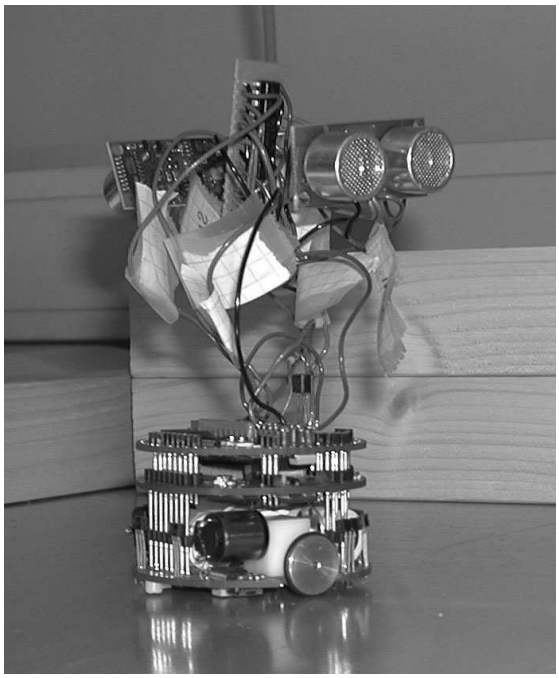

Bild 2.7 Första lösningen.

Tester visade att konstruktionen fungerade och kunde mäta avstånd med ungefär upplösningen tre cm som sonarerna ska klara av. Det fanns dock en del saker som behövdes förbättras. För det första så innebar användandet av mjukvaruklocka att stora resurser användes enbart för att generera en klocksignal. Motorola processorn kör varje process i fem ms. Att stänga av klockan under så lång tid skulle kunna innebära att någon avbrottssignal fördröjs, vilket inte var acceptabelt. Så därför kunde inga andra processer köras under tiden som en sonarmätning utfördes, vilket även stred mot anledningen till att använda lösningen med avbrott. Sedan så behövde sonarerna fixeras på roboten och det skulle ha varit lite snyggare om en del av sladdarna kunde ersättas med ett kretskort. Det kan också nämnas att det var lite av en tillfällighet att lösningen fungerade så långt. Motorolaprocessorn hanterar avbrott och processbyten på ett sätt som fungerade bra ihop med mjukvaruklockan. Men när den skulle bytas ut mot en hårdvaruklocka så framkom det nya problem. Mer om det i avsnitt 2.3.1.

2.3 Den slutgiltiga lösningen

Tre saker skulle förbättras. Mjukvaruklockan skulle bytas ut mot en hårdvaruklocka, ett kretskort skulle byggas för att ge lösningen den mekaniska pålitligheten och snygga till designen lite och slutligen skulle sonarerna fixeras så att de förblir rätt riktade.

2.3.1 Hårdvaruklockan

Då var frågan vilken hårdvaruklocka som skulle väljas. Mjukvaruklockan kunde med ett oscilloskop uppskattas till att vara på ca 50 kHz. Jag prövade att sänka hastigheten med hjälp av att lägga till bioskommandot switch_fast() i loopen som skriver till adress 36. Den fungerar så att den direkt byter till nästa process i kön. Men eftersom inga andra processer kördes så bytte den tillbaka till klockgeneratorprocessen igen. Klockfrekvensen kunde då uppskattas till att vara ca 10 kHz istället. Men med den frekvensen blev mätresultaten dåliga. De verkade bara ha en upplösning på ca 17 cm i det fallet. Utan att tänka så mycket mer på det så tolkade jag det som att det behövdes en frekvens på i alla fall 50 kHz. Egentligen berodde det på hur processorn hanterar processbyten. Så egentligen skulle frekvensen som räknats ut i avsnitt 2.2 på 5556 Hz kunnat räcka, men mer om det i avsnitt 2.3.1.1.

En viktig egenskap klockan ska ha är att den ska vara strömsnål. Om roboten ska köras med enbart batterierna så vill man ju att den ska kunna användas så länge som möjligt. I Elfakatalogen var den strömsnålaste kristalloscillatorn på 15 mA och den långsammaste av dem låg på 5 MHz. Snabbare än så behövs inte och de långsammare drar mer ström. Därför valdes den.

2.3.1.1 Undantagsproblem

Men efter att ha bytt ut mjukvaruklockan mot 5 MHz kristalloscillatorn så slutade konstruktionen att fungera. Ett undantag genererades så fort det blev ett avbrott. Från utskriften från programmet framgick det att det var spurius interruption (se [9,10]) som orsakade det. Frågan var varför det genererades ett avbrott bara för att klocksignalen blivit snabbare? Första tanken var att strömförsörjningen kanske hade överskridits när kristalloscillatorns 15 mA lagts till. Det genomsnittliga värdet på strömförbrukningen ligger på max 117 mA, 50 mA var för sonarerna plus 15 mA för kristalloscillatorn plus 0.4 mA var för de fem komponenterna, vilket är långt under de 200 mA som kan förses av I/O-Turreten. Men i manualen till den nyare sonarn från Devantech SRF08, så nämns det att den

momentana strömförbrukningen kan vara betydligt större än så. Alltså skulle kanske en kondensator som extra buffert kunna lösa problemet. Hur stor buffert skulle behövas då? SRF08:an drar extra mycket ström (275 mA) under ca tre µs. Så man skulle kunna avrunda det till att under ungefär fem µs ska bufferten kunna leverera ström. Man kan också säga att under den tiden får kondensatorn inte förlora mer spänning än 0.5 V. Slutligen för enkelhetens skull så kan man anta att den max behöver leverera konstant 100 mA under den tiden. Beräkningarna blir då som följer i formel 2.1 nedan. Kondensatorn införskaffades och sattes mellan jord och matspänning. Men det gjorde ingen skillnad alls.

$$U = 0.5V = \frac{1}{C}\int_0^{5\mu s} Idt = \frac{1}{C}\left[It\right]_0^{5\mu s} = \frac{1}{C}100mA \cdot 5\mu s \leftrightarrow C = 1\mu F \qquad (2.1)$$

Kanske kunde det vara ett mjukvaruproblem istället. Det kunde vara så att avbrottsrutinen läste av avbrottsingången för att se om det var den som genererade avbrottet. Problemet var då att avbrottsingången hade hunnit gå hög igen långt innan det testet gjordes. Eftersom klockan var på 5 Mhz hann det bara gå ungefär tre klockcykler internt i Kheperan. Jag hade tre förslag på hur de tre klockcyklerna kunde utökas och förhoppningsvis lösa problemet:

1) Det fanns en oanvänd flip-flop kvar som kunde användas. I kopplingen i bild 2.6 skulle den kunna läggas till direkt efter den andra flip-flopen. Det skulle dubbla tiden till ungefär sex interna klockcykler. Men det är antagligen inte tillräckligt, så det är nog inte en ändring som löser problemet.
2) En annan lösning vore istället att skaffa en långsammare kristalloscillator. Den långsammaste färdiga kristalloscillatorn är på 1 MHz. Det skulle bara öka antalet interna klockcykler till ungefär 15 och dessutom ökar då strömförbrukningen till 30 mA för kristalloscillatorn. Alltså var inte det heller en lösning som verkade bra.
3) Ett tredje sätt var att lägga till en räknare som kan räkna till förslagsvis 100. Kristalloscillatorn räknar då upp den och istället för kristalloscillatorn fungerar istället räknarens mest signifikanta bit som klocksignal. Det gör att klockfrekvensen sänks till 50 kHz vilket ju fungerade bra när mjukvaruklockan användes.

Alltså prövade jag det tredje alternativet och införskaffade en räknare. Efter det fungerade sonarerna bättre. Men avståndet hade bara en upplösning på ca 17 cm. Och det verkade som den missade avbrottssignaler ibland och mätte tiden mellan sonaravläsningarna istället. Dessutom hände det även någon gång ibland att samma undantag som tidigare kom. I [3] står det i slutet bland exempelkoden att man ska undvika att sätta innterruptsignalen till jord manuellt. Det föreslås att en flip-flop ska kopplas till ingången och sedan rensas i avbrottsrutinen. Först verkade det oklart vad de menade med det. Men vad var egentligen skillnaden på den fungerande mjukvaruklockan och den icke fungerande hårdvaruklockan? Jo, skillnaden ligger i avbrottsrutinen. Som det nämndes tidigare så avbröt avbrottsrutinen alla andra processer inklusive klockgeneratorn. Klockgeneratorn startade sedan inte igen förrän avbrottsrutinen var klar. Hårdvaruklockan kördes däremot hela tiden. Från det kan man sluta sig till att avbrottssignalen alltså måste vara låg under precis så lång tid som det tar för avbrottsrutinen att starta, vilket varierar. Det är ju inte ett bra sätt att sköta avbrotten på, om externa komponenter, som inte ska ha något med Kheperan att göra, ska kunna kopplas till I/O-Turreten. Men det finns en lösning på problemet som visas i bild 2.8 nedan. Som det nämndes

om i exempelkoden så kan en extra flip-flop kopplas mellan den gamla avbrottssignalen och avbrottsingången på I/O-Turreten. Men den gamla signalen kopplas istället till klockinsignalen på flip-flopen. Till insignalen kopplas istället fem volt och efter utsignalen en inverterare. Slutligen kopplas skrivpuls signalen (WP), som tidigare användes som klocka, till clear ingången på flip-flopen. Clear nollställer utgången. Det går bra att använda WP direkt eftersom clearingången är aktiv låg. Tanken är att flip-flopen ska vara nollställd. När sedan en avbrottspuls genereras släpps ettan på ingången igenom och ett avbrott genereras. Avbrottssignalen är sedan aktiv ända tills flip-flopen nollställs igen, vilket görs i avbrottsrutinen genom att skriva till adress 36 varpå WP går låg under ca 200 ns som tidigare.

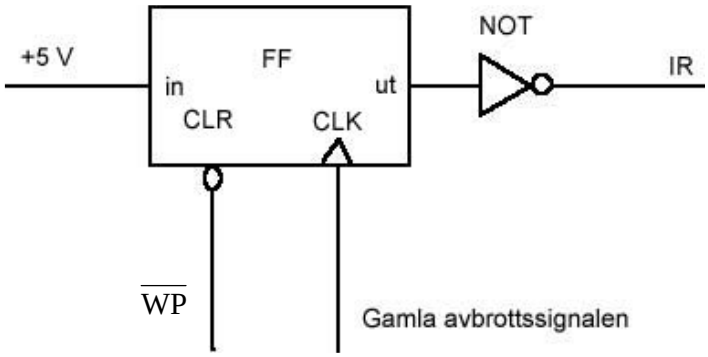

Bild 2.8 Hur en extra flip-flop ska kopplas till konstruktionen för att lösa avbrottsproblemet

Tabell 2.1 Komponenter i slutgiltiga lösningen.	
Komponentnamn	Förklaring
Två st. 74LS175.	Fyra st. tvåingångars flip-flopar med inverterade och oinverterade utgångar.
En 74LS153.	Två st. fyraingångars multiplexrar.
En 14 pin DIL 5 MHz kristalloscillator.	Klocka.

En extra flip-flop komponent fick läggas till. När det var klart försvann undantagsproblemen helt men fortfarande kvarstod problemet med att upplösningen bara var omkring 17 cm oavsett hur många mätningar det gjordes medelvärde på. Lösningen på det problemet fanns även det att finna i varför mjukvaruklockan fungerade. I lösningen med hårdvaruklockan kördes inga processer alls medan programmet väntade på avbrott. I versionen med mjukvaruklockan kördes däremot klockgeneratorprocessen hela tiden. Genom att lägga till en process med en tom for sats till hårdvaruklocksversionen så fungerade även den. Motorolaprocessorn kontrollerar alltså bara varje hel ms om det finns någon process som ska köras, även avbrottsrutinen, när ingen process körs alls för tillfället. Körs däremot minst en process så avbryter avbrottsrutinen alla processer direkt. Det var även anledningen till att det blev så mycket sämre upplösning när switch_fast() lades till i klockgeneratorprocessen. När den skulle byta till nästa process fanns inga andra. Då vilade processorn tills nästa hela ms då den såg att klockgeneratorprocessen ville köra igen. Sammanfattningsvis berodde problemen dels på hur avbrottshanteringen fungerar. Avbrottssignalen måste vara aktiv ”lagom” länge. Sen berodde problemen även på hur processbytena görs. Man måste bara tänka på att alltid ha någon process igång för att avbrottsrutinen ska köras igång direkt vid avbrott och då få en

bättre upplösning på sonarerna än 17 cm. I tabell 2.1 nedan finns en förteckning över vilka komponenter som användes i den slutgiltiga lösningen.

2.3.2 Snyggare utformning

Då återstod det bara att snygga till konstruktionen och fixera sonarerna. För att göra ett kretskort som minskar ner på antalet sladdar så användes programmet Eagle. Programmet innehåller mängder med färdiga komponenter att använda. Först gjorde jag en schematisk design som bara visar vilka komponenter som används och hur in- och utgångarna är kopplade till varandra. Bild 2.9 nedan visar den. Därefter användes den för att göra layouten som är själva kretskortet. Man får själv bestämma var komponenterna ska ligga. Men det finns en färdig funktion som drar de nödvändiga kopplingarna, var nödvändiga hål ska göras o.s.v. Den funktionen klarade dock inte av att lägga alla ledningarna på samma sida på kretsen. P.g.a. begränsade resurser valdes ETF laboratoriet för att egentillverka kretskortet och då krävdes det. Jag lyckades dock själv få alla kopplingar på samma sida och kunde då alltså göra kretskortet själv. Figur 2.10 nedan visar den färdiga layouten med komponenterna i vitt. Som man kan se finns det även plats för en kondensator som var tänkt som buffert. I appendix B finns layouten med enbart ledningssidan om man skulle vilja göra ett nytt kretskort.

Efter det var kretskortet också klart. Då visade sig det att det inte gick att montera sonarerna framåt och bakåt på ett enkelt sätt, vilket var tänkt från början. Men eftersom det egentligen inte har så stor betydelse, så monterades de åt sidorna istället. Bild 2.11 nedan visar den slutgiltiga lösningen.

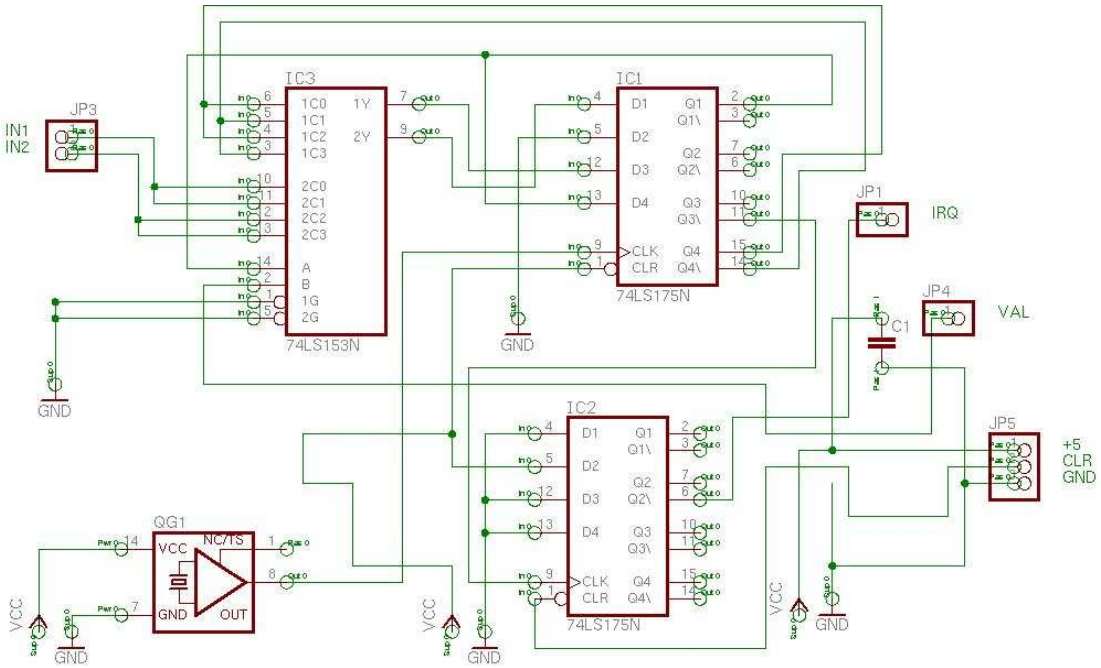

Bild 2.9 Schematisk bild över kretskortet.

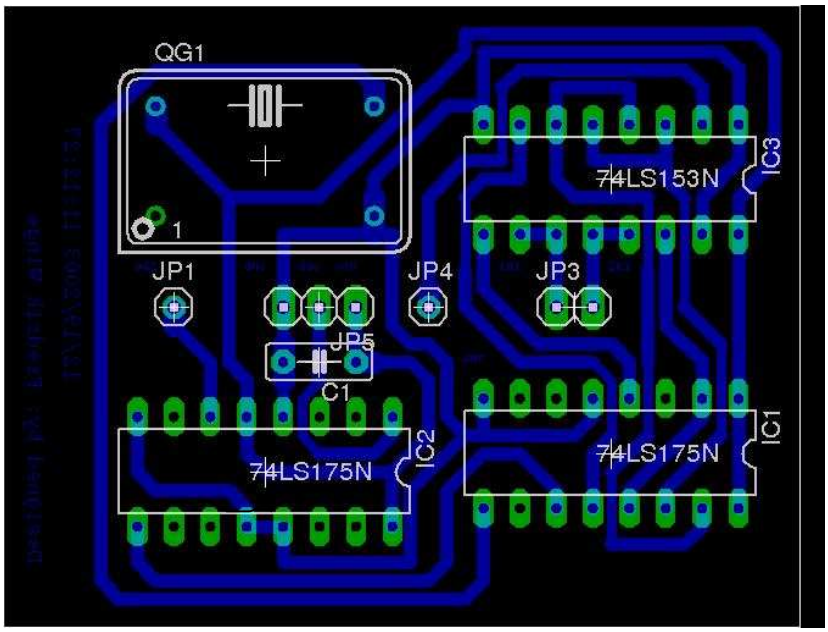

Bild 2.10 Den färdiga layouten till kretskortet.

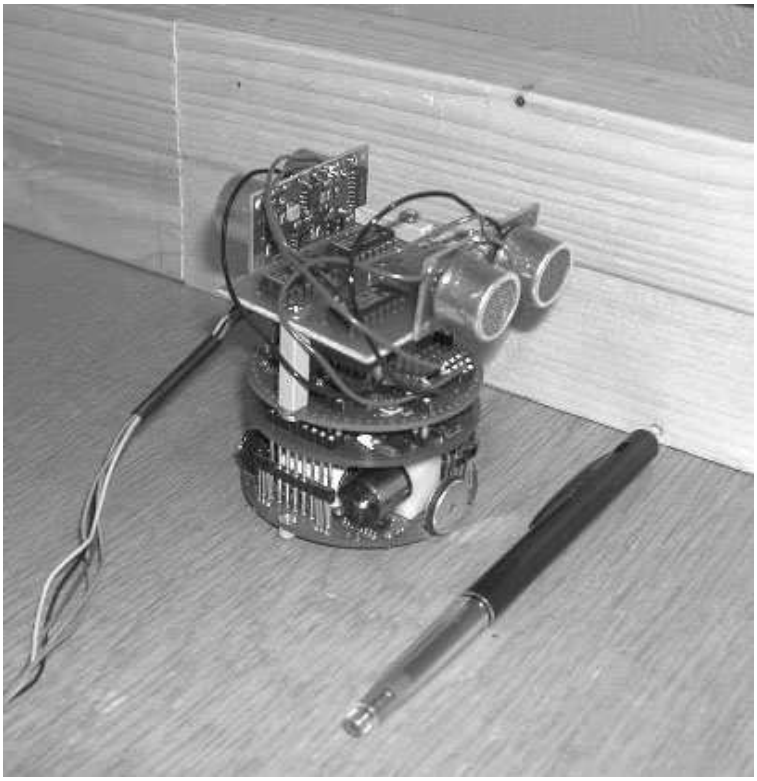

Bild 2.11 Den slutgiltiga lösningen sett snett bakifrån.

Sammanfattningsvis finns det tre huvudfördelar med den slutgiltiga lösningen. Med kretskortet har den mekaniska hållbarheten ökats jämfört med när bara sladdar användes i den första lösningen. Sedan har införandet av hårdvaruklockan medfört att roboten kan utföra andra beräkningar under tiden som sonarmätningarna utförs. Slutligen håller sig sonarerna i samma riktning och vinkel sedan de har fixerats på roboten.

Kapitel 3

Resultat

När själva konstruktionen var klar återstod det att testa den. Först visas testerna på hur pass bra prestanda SRF04 sonarn har. Sedan visas ett litet kartläggningstest där hela systemet med robot och sonarerna testades tillsammans.

3.1 Sonarns egenskaper

Hur stora objekt kan upptäckas på olika avstånd? Hur ändras känsligheten när vinkeln till objektet ändras? Och hur bra upplösning kan man få med sonarerna? Det var några frågor som skulle försöka besvaras.

3.1.1 Känslighet

För att ta reda på hur stora objekt som kan upptäckas och på vilka avstånd så valdes några olika typer av objekt ut. De ställdes sedan rakt framför en av sonarerna på roboten och sonarns avståndsbestämning jämfördes med det verkliga avståndet som erhölls med hjälp av ett måttband. Själva koden för att göra mätningen var ganska enkel. Appendix A visar exempel på kod för att göra sonarmätningar med Kheperan och skriva ut resultaten på skärmen. En detalj som kan nämnas om koden är att roboten för säkerhets skull ställdes in på att göra en sonaravläsning var 80:e ms. Teoretiskt borde den klara en sonaravläsning var 48:e ms eftersom den maximalt väntar 38 ms på att sonarpulsen ska komma tillbaka och sedan behöver tio ms innan nästa sonarpuls kan skickas.

Tre olika objekt testades för att ta reda på känsligheten rakt fram. Det var två bitar hård skumplast med storlekarna 3.2 cm*3.8 cm=12.2 cm^2 och 2.4 cm*2.5 cm=6 cm^2 samt botten på en tuschpenna med diametern 1.9 cm eller arean 2.8 cm^2. Testet utfördes så att sonarn gjorde ett medelvärde av 100 mätningar för att få en bra upplösning. Varje sådant test utfördes flera gånger för varje valt avstånd och objekt. Ljudets hastighet antogs vara 342 m/s. Resultaten för varje test varierade, så tabell gjordes över mellan vilka värden den antog. Tabell 3.1 och bild 3.1 nedan visar hur bra den stora skumplastbiten kunde upptäckas. Bilden visar diagrammet från tabelldatan i tabell 3.1 där X-axeln visar det verkliga avståndet. Y-axeln visar största och minsta uppmätta medelvärde samt en hjälplinje som visar det verkliga avståndet. Tabell 3.2 och bild 3.2 ger motsvarande data för den lilla skumplastbiten och slutligen visar tabell 3.3 och bild 3.3 datan för tuschpennan. Man kan se i diagrammen att de uppmätta avstånden stämmer rätt bra med de riktiga avstånden fram till en viss punkt. Då sticker det största uppmätta avståndet iväg medan det minsta stannar kvar. Det beror på att ljudsignalsstyrkan har minskat och då också chansen att upptäcka objektet. I avståndsdatan som används för medelvärdesbildningen ingår då avstånd till objekt som ligger bortom objektet, vilket gör att avståndet verkar längre än vad det är. Däremot är det inte möjligt att

upptäcka andra objekt närmare än testobjektet om där inte är några. Det gör att felet i medelvärdesbildningen bara kan göra det uppmätta avståndet större än det verkliga avståndet. Det förklarar varför det minsta värdet på avståndsbestämningen stannar kvar vid rätt värde längre än det största värdet. Det största värdet är alltså värsta fallet.

Tabell 3.1 Hård skumplastbit (12,2 cm^2)		
Verklig längd (m)	Minsta uppmätta medelvärde (m)	Största uppmätta medelvärde (m)
0,2	0,249	0,256
0,3	0,328	0,333
0,4	0,42	0,427
0,8	0,808	0,819
1,2	1,217	1,226
1,5	1,532	1,537
1,6	1,627	1,636
1,7	1,715	1,751
1,75	1,79	1,795
1,78	1,824	1,908
1,8	1,86	1,98
1,85	1,881	2,081

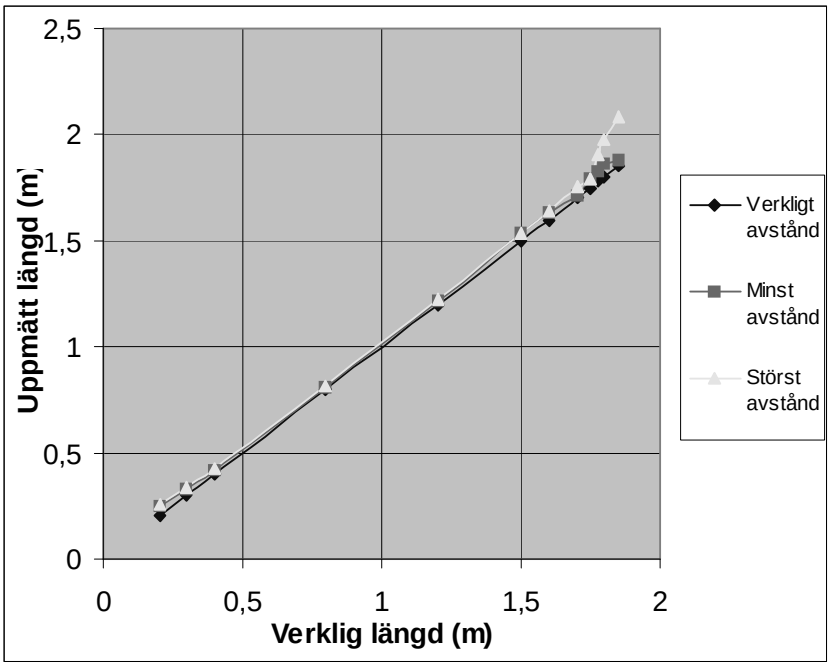

Bild 3.1 Diagram som visar de minsta resp. största uppmätta avstånden jämfört med det verkliga avstånden för den stora skumplastbiten.

En annan sak som kan noteras är att även vid korta avstånd, när objektet alltid borde upptäckas, så är det uppmätta avståndet alltid omkring två till tre cm längre än det verkliga avståndet. En delförklaring till det kan vara att avståndet till objektet mättes från framkanten av sonarn när egentligen mottagaren och sändaren sitter en bit in. Men det kan inte förklara mer än omkring en centimeter av felet. Inte heller verkar problemet vara i genereringen av avbrotten. Båda avbrotten genereras på samma sätt vilket gör att eventuella fördröjningar borde ta ut varandra. Så det tyder på att felet sitter i sonarn.

Något annat som är intressant är hur storleken på objektet påverkar hur långt ifrån den kan upptäckas. Den stora skumplastbiten kunde upptäckas ganska bra upp till ca 175 cm medan den lilla kunde upptäckas ganska bra upp till ca 138 cm. Teoretiskt avtar ljudsignalstyrkan proportionellt med roten ur avståndet. Så den stora skumplastbiten borde då kunna upptäckas på ett avstånd som är ca 1,43 gånger större än avståndet den lilla skumplastbiten kan upptäckas på ((12,2/6)^0,5≈1.43). Men i det här fallet var avståndet bara ca 1,27 gånger större (175/138≈1,27). En anledning till det kan vara att det var svårt att sätta skumplastbiten helt vinkelrätt mot sonarn, vilket har större inverkan ju längre ifrån den är.

Tabell 3.2 Hård skumplastbit (6 cm^2)		
Verklig längd (m)	Minsta uppmätta medelvärde (m)	Största uppmätta medelvärde (m)
0,8	0,813	0,817
1,1	1,113	1,118
1,2	1,214	1,217
1,3	1,316	1,32
1,35	1,368	1,371
1,38	1,405	1,41
1,4	1,421	1,691

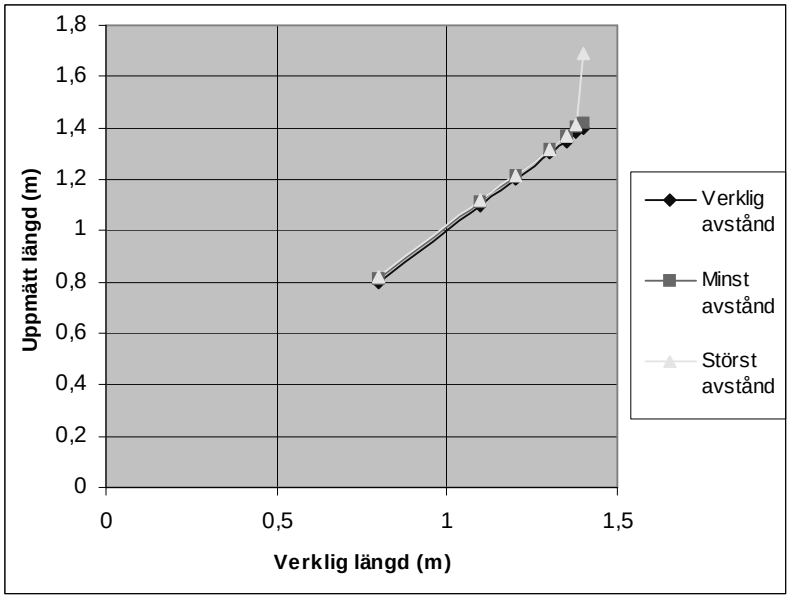

Bild 3.2 Diagram som visar de minsta resp. största uppmätta avstånden jämfört med det verkliga avstånden för den lilla skumplastbiten.

Man kan även se att materialet har stor betydelse. Botten på tuschpennan, som är av hårdare material än den hårda skumplasten, kan upptäckas på upp till ca 185 cm. Det är bättre än den betydligt större biten av skumplast. Så mjuka saker som t.ex. kläder och textiler är svårare att upptäcka med sonarn än hårda saker som trä och metall. Det kan också nämnas att mjuk skumplast inte gick att upptäckas alls med sonarn.

Slutligen kan man se på bild 3.1 att det uppmätta avståndet verkar skilja sig mer och mer från det verkliga ju närmare objektet kommer sonarn. Det beror på att sonarn inte var i samma plan som testobjektet utan en bit ovan. Så det är inte ett fel på sonarn. Egentligen klarar hela

systemet med sonar och robot av att mäta avstånd ända ner till tre cm, precis som sonarn enligt specifikationen ska klara av.

Tabell 3.3 Botten på tuschpenna (2,8 cm^2)		
Verklig längd (m)	Minsta uppmätta medelvärde (m)	Största uppmätta medelvärde (m)
1,4	1,432	1,436
1,5	1,535	1,539
1,6	1,629	1,633
1,7	1,718	1,739
1,8	1,834	1,841
1,85	1,829	1,911
1,9	1,923	2,072

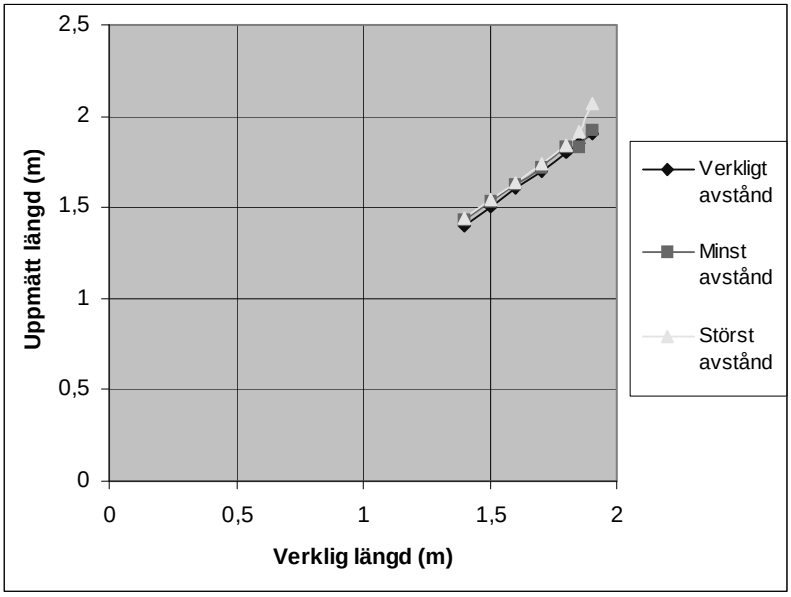

Bild 3.3 Diagram som visar de minsta resp. största uppmätta avstånden jämfört med det verkliga avstånden för botten på tuschpennan.

3.1.2 Vinkelberoende

För att ta reda på hur känsligheten påverkas av hur vinkel till sonarn ändras, så användes den lilla skumplastbiten. Den flyttades ifrån sonarn längs en viss vinkel. När den inte längre upptäcktes så visade det hur långt bort den kunde upptäckas för den vinkeln. Var linjen för en viss vinkel fanns fick jag fram genom att använda två måttband. Den ena mätte avståndet rakt fram precis som i känslighetstestet i avsnitt 3.1.1. Den andra lades vinkelrätt ut från det första måttbandet. Ur förhållandet $tan(v)=y/x$, där v är vinkeln, x är avståndet rakt fram och y är avståndet vinkelrätt från det första måttbandet, kunde y lösas ut. Punkten där y slutar och basen på sonarn bildar då en linje där vinkeln är densamma. x valdes till ett stort värde, 1,5 meter, för att linjen för vinkeln skulle bli så exakt som möjligt. Bild 3.4 illustrerar hur linjen erhölls. Längs den linjen sattes sedan ett måttband som skumplastbiten flyttades på.

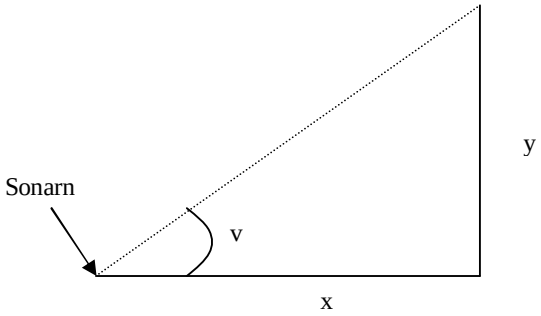

Bild 3.4 Bild som visar hur linjen med vinkeln *v* erhölls.

Tabell 3.4 nedan visar hur långt ifrån skumplastbiten kunde upptäckas beroende på vinkeln till den. Som man kan se i tabellen så minskar bara avståndet till ca 85.6% av maximala avståndet då vinkeln ändras från noll till 30 grader. Efter 31 grader upptäcks däremot inte skumplastbiten alls. Det beror på det som nämndes i avsnitt 2.1, att sändar- och mottagarsensorerna är avskärmade av ett hölje. Tydligen är sonarn konstruerad att upptäcka objekt inom en kon på 60 grader. Det gör att β värdet i formel 1.2 och 1.5 kan sättas till 30 grader. Det är en viktig upptäckt när sedan den Bayesiska algoritmen ska prövas i avsnitt 3.2.

Tabell 3.4 Vinkelberoende	
Vinkel (grader)	Maximalt avstånd (m)
0	1,38
10	1,38
20	1,34
30	1,18
31	Gick ej

3.2 Ett kartläggningstest

Det mest intressant testet är hur systemet med robot och sonarerna tillsammans kan kartlägga en testbana. Bild 3.4 nedan visar hur banan såg ut. Bild 3.5 är en skiss över banan med avståndsangivelser och robotens färdväg angiven. Banan är tänkt att den både ska testa hur pass bra långa raka väggar och mindre detaljer som pelarna kan kartläggas.

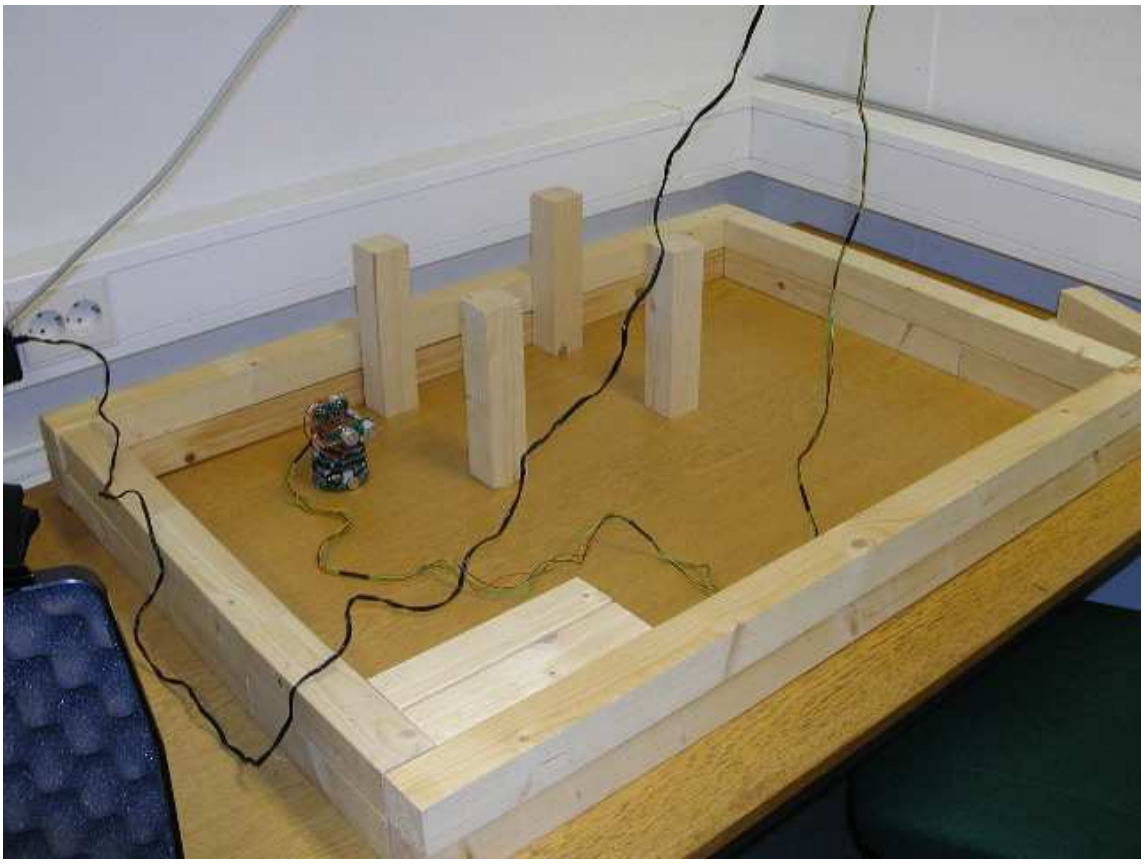

Bild 3.4 Bild över testbanan.

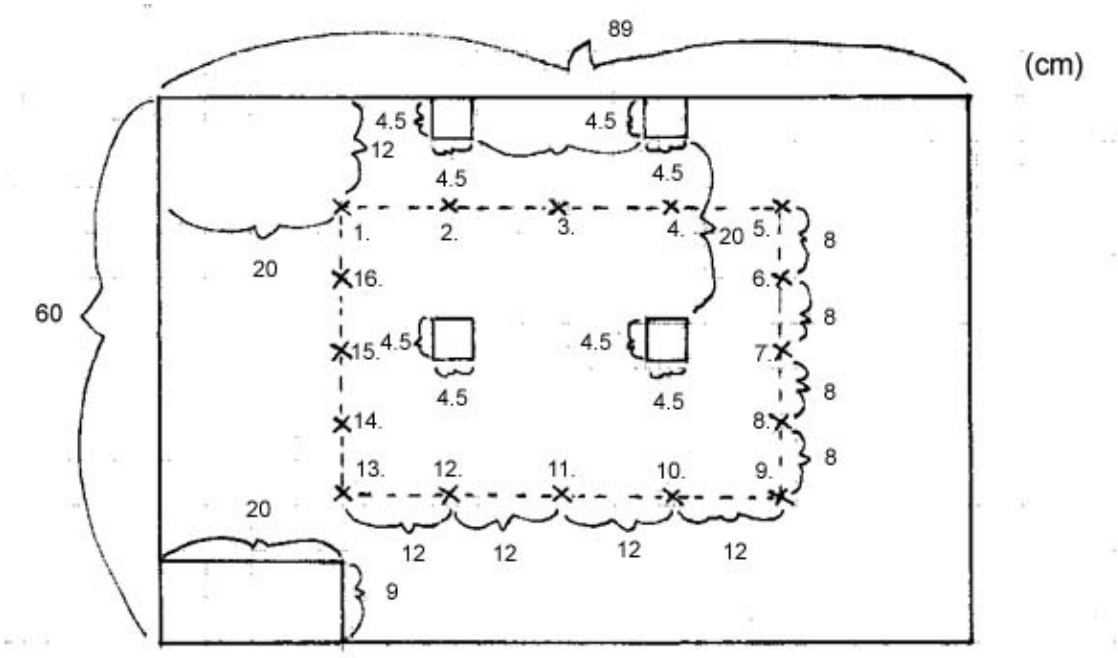

Bild 3.5 Skiss över testbanan inklusive avståndsangivelser och robotens färdväg.

Kartläggningsalgoritmerna som användes var Bayesiska algoritmen från avsnitt 1.3 och en egen jag ville pröva. Bayesiska algoritmen är bra för det här projektet eftersom den är speciellt utvecklad för sonarn som sensor. Men det finns svagheter med den:

- Den tar mycket minne eftersom varje ruta i rutnätmönstret representeras av ett flyttal. Egentligen behöver roboten som ska använda kartan (oftast samma robot som kartlägger) enbart veta om rutan kan anses ockuperad eller inte.
- Det krävs ganska mycket beräkningar för varje ruta. Enklare beräkningar skulle snabba upp algoritmen.
- Den kan ge konstiga resultat. Skulle ett objekt upptäcka på långt avstånd så är även kanterna av område två i sonarkonen (bild 1.3) ockuperade enligt algoritmen. Man kan tänka sig scenariot att roboten redan har fått en karta över omgivningen som nästan är perfekt. T.ex. kan alla rutor som är tomma ha sannolikheten 0,1 att de är ockuperade och de ockuperade ha sannolikheten 0.9 för det. Även om roboten alltid skulle veta var den är och åt vilket håll den är riktad samt att sonarn alltid skulle ge rätt avstånd, så skulle algoritmen bara göra kartan sämre i de flesta fall. Det visar att den inte är optimal.

HIMM algoritmen (avsnitt 1.3.2) tar både mindre minne och har enklare beräkningar. Den är dock för enkel för att pröva hur bra sonarerna är på kartläggning. Men om HIMM modifieras så att den uppdaterar hela konen istället, så är det sannolikt att algoritmen blir bättre. Att enbart ändra så att alla rutorna i sonarkonens område två uppdateras med –1 och alla rutor i område ett uppdateras med +3 duger dock inte. För område två går det bra eftersom hela området kan anses tomt, men den reflekterade signalen kommer nästan aldrig från alla rutorna i område ett. Antagligen för det mesta enbart från en av dem. Enda sättet att kunna gissa från vilken av rutorna signalen kom från är att se tillbaka på gamla mätningar. Det kan göras genom att se på kartan som har erhållit hittills. Om någon av rutorna i område ett har ett större värde än de andra innebär det att den har större chans att vara ockuperad. Det kan då antas att signalen antagligen kom därifrån och att alltså enbart den rutan ska uppdateras med +3. Skulle flera rutor ha samma största värde så uppdateras alla dessa med +3. Signalen måste ju ha kommit någonstans ifrån. Djupet på sonarkonens område ett är tänkt att vara en ruta. Större djup är möjligt men resultatet blir antagligen sämre. Nedan är en sammanfattning av den egna algoritmen som prövades på testbanan:

- Kartan består av ett rutnät med tal mellan noll och femton. Ett högt tal innebär att rutan antagligen är ockuperad och ett lågt tal att den antagligen inte är det.
- Vid en sonarmätning uppdateras alla rutor i område två med –1 (se bild 1.3 och avsnitt 1.2.1.1). I område ett uppdateras rutan med störst värde med +3. De övriga påverkas inte. Skulle flera rutor i område ett ha samma största värde så uppdateras alla dessa med +3.
- Djupet på sonarkonens område ett bör vara en ruta.

Den algoritmen tar både mindre minne än Bayesiska algoritmen (vanligtvis 16 gånger mindre, se avsnitt 1.3.2) och är snabbare eftersom rutorna enbart uppdateras med en addition eller en subtraktion. För scenariot ovan där roboten startar med en redan färdig karta, så skulle till skillnad från Bayesiska algoritmen denna inte göra kartan sämre.

Experimentet utfördes enligt följande. Tre olika tester gjordes. Roboten färdades medurs längs den streckade linjen i bild 3.5 i alla tre testerna. I första testet stannade den och gjorde sonaravläsningar i alla X-punkter med ojämna nummer. I varje punkt gjordes åtta mätningar

jämnt fördelade i alla rikningar. Eftersom den stannade i åtta olika punkter blev det totalt 8*8=64 olika sonaravläsningar. I det andra testet gjordes istället 16 sonaravläsningar i varje punkt men annars var det som i första testet. Så i det testet blev det istället totalt 128 olika sonaravläsningar. I det sista testet stannade roboten även i alla X-punkter med jämna nummer. Precis som i första testet gjordes åtta olika sonaravläsningar i varje punkt vilket gav totalt 128 st. Det finns inget kommando att vända t.ex. 45 grader. Istället finns kommandon för att snurra hjulen. Så innan testet fick roboten kalibreras för att ta reda på hur mycket hjulen skulle snurras, för att vrida roboten en viss vinkel. Felet i positionen visade sig litet i alla testerna. När roboten hade slutfört hela varvet och återkommit till startpositionen, så skiljde det bara någon centimeter och riktningen var nästan densamma. Det gör att eventuella fel i mätningarna till största delen beror på sonarerna. Båda sonarerna användes i testerna. De gjorde varannan mätning vilket gjorde att mätningarna kunde snabbas upp väsentligt. Normalt mätte ju en sonar var 80:e ms. Men med två sonarer kunde väntetiden efter varje sänkas till 40 ms. Totalt sett väntade då varje sonar 80 ms som tidigare, men två mätningar hade då gjorts under den tiden. Det gjorde att mätningarna kunde göras dubbelt så snabbt med två sonarer. För att snabba upp mätningen gjordes bara medelvärdesbildning av tio mätningar istället för hundra som använts i tidigare experiment. Det medförde att avståndsbestämningen blev lite sämre. Informationen om avstånd till objektet, robotens position och riktning skrevs ut på skärmen för varje sonaravläsning och lades sedan in i en fil. När testerna var klara så lästes informationen in av ett javaprogram gjorts för ändamålet. Programmet bearbetade informationen, gjorde alla beräkningarna som krävdes för de två algoritmerna och skrev sedan ut resultatet i form av en fil med alla värdena i rutnätet som utgör kartan. Den filen lästes sedan in i matlab där en kartbild kunde göras. Bild 3.6 till 3.8 nedan visar resultatet för Bayesiska algoritmen. Första bilden är för test ett o.s.v. Bild 3.9 till 3.11 visar motsvarande resultat för den egna algoritmen. Varje ruta på kartorna är tre gånger tre centimeter. Totalt består kartorna av 60 gånger 60 rutor. En ljus ruta innehåller ett högt värde och har alltså stor sannolikhet att vara ockuperad. Tvärtom för svarta rutor. När testet började så antogs alla rutorna vara ockuperade med sannolikheten 50 %. För Bayesiska hade alltså rutorna värdet 0.5 av ett och för den egna algoritmen hade rutorna värdet 8 av 15 från början (korrigerad till 8/15 för att få maximalt ett). Kartorna var alltså gråa från början. Roboten började i nedre vänstra hörnet på kartorna, 50 cm ifrån kanterna på kartan.

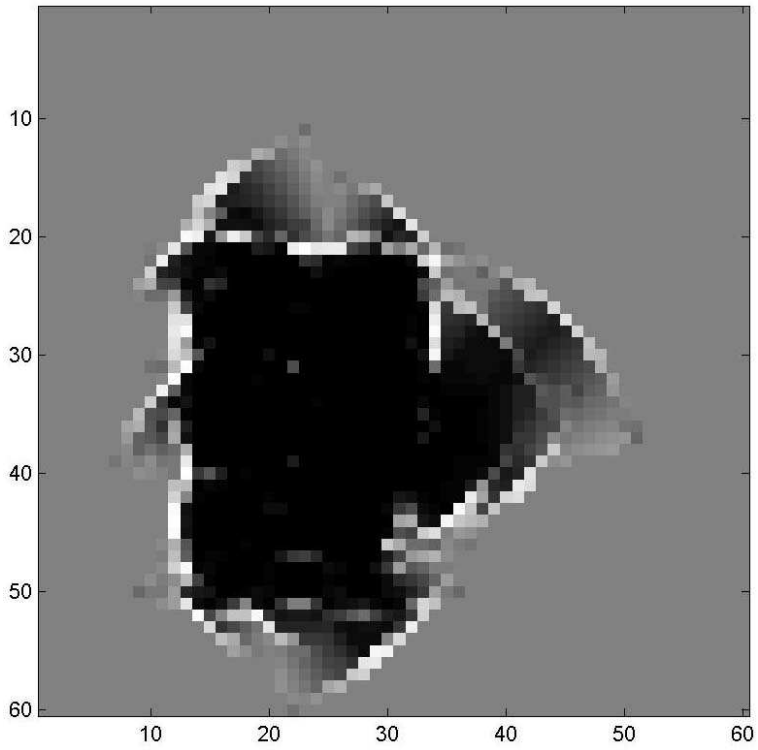

Bild 3.6 Resultatet för Bayesiska algoritmen för test ett.

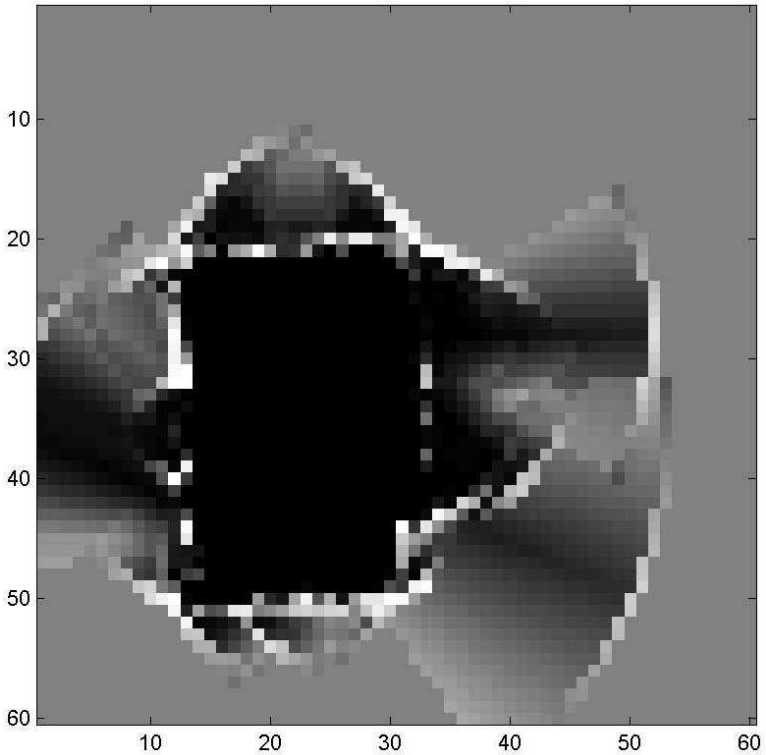

Bild 3.7 Resultatet för Bayesiska algoritmen för test två.

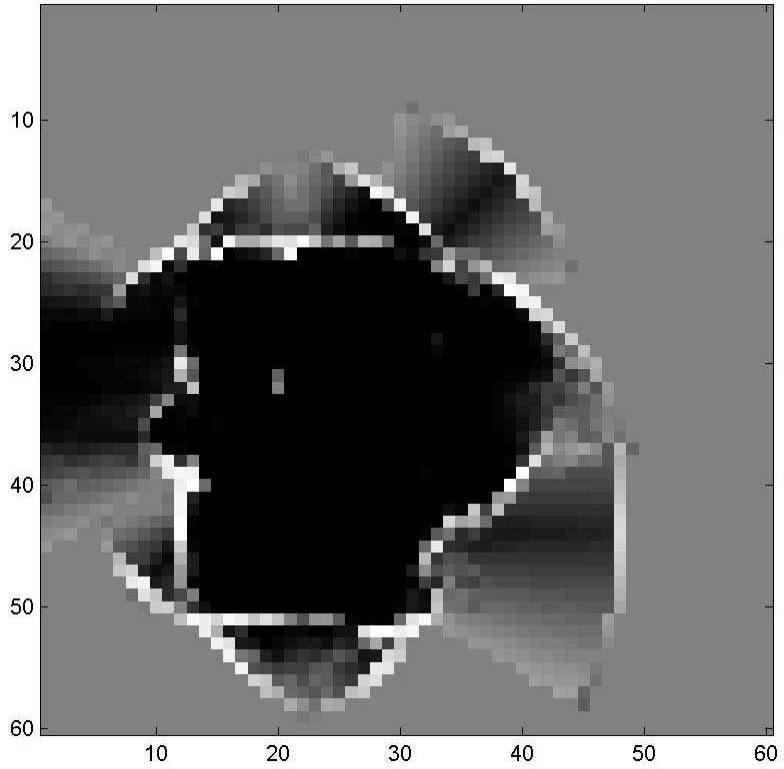

Bild 3.8 Resultatet för Bayesiska algoritmen för test tre.

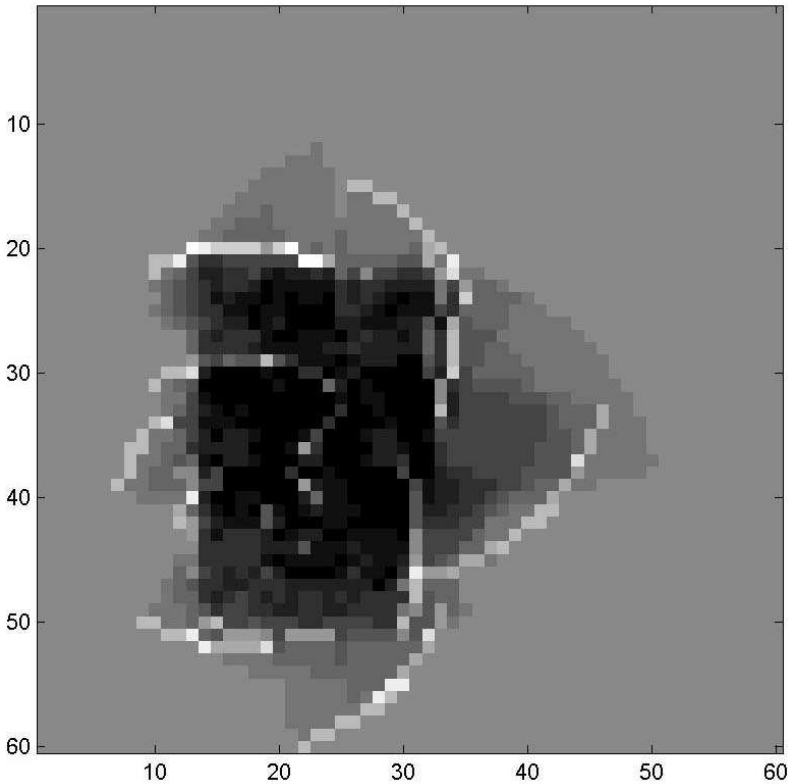

Bild 3.9 Resultatet för den egna algoritmen för test ett.

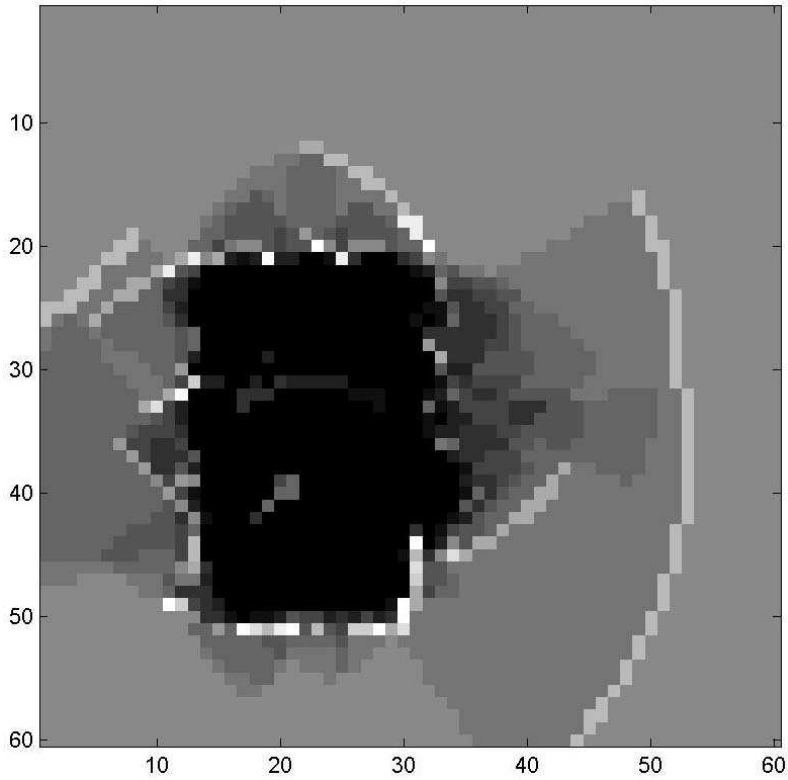

Bild 3.10 Resultatet för den egna algoritmen för test två.

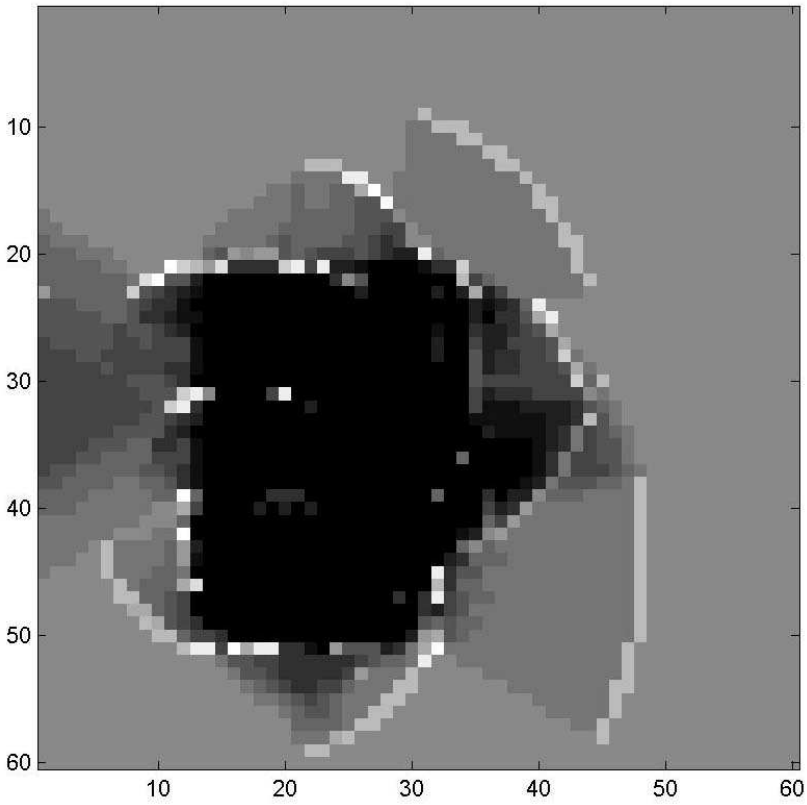

Bild 3.11 Resultatet för den egna algoritmen för test tre.

På alla kartorna verkar det vara problem att hitta framförallt den högra väggen. Det beror på problemet med reflektion som diskuterades i avsnitt 1.2.1.2. När roboten gör en sonarmätning snett uppåt höger eller snett neråt höger på kartan från positionerna omkring 8-14, så ser man att den har upptäckt något betydligt längre bort än var väggen ligger. Antagligen har ultraljudssignalen studsat från högra vägen till den övre eller undre väggen och sedan tillbaka till roboten. Även på de andra väggarna kan man ana det problemet. Problemet som illustreras i bild 1.5 b syns också. På alla kartorna finns väggar bakom hörnen som egentligen inte finns där i verkligheten. Teoretiskt borde det sista testet ge bäst resultat eftersom mätningarna gjorts från fler positioner. Totalt sett verkar det dock inte vara så stor skillnad mellan kartorna för det andra och tredje testet.

Kartorna för den egna algoritmen verkar brusigare, men å andra sidan behåller den mer av detaljerna som försvunnit i kartorna för den Bayesiska algoritmen. Pelarna syns t.ex. bättre i kartorna för den egna algoritmen. Den egna algoritmen verkar istället ha problem med de långa raka väggarna. Kartan för det första testet (bild 3.9) är speciellt brusig. Det verkar som den egna algoritmen behöver fler mätningar för att stabilisera kartan. Ett sätt att ta bort mycket av bruset på den karta skulle kunna vara att rensa upp kartan snabbare, genom att minska rutorna i område två med två istället för ett. Men att ökningen av rutorna i område ett är större än minskningen för område två, verkar vara bra mot problemet med reflektion och andra felmätningar. Man kan se på kartorna att där Bayesiska algoritmen rensat bort väggarna helt på grund av reflektionen, finns det fortfarande ofta spår kvar av väggen för den egna algoritmen. Det är en anledning till att kartorna för den egna algoritmen är brusigare men innehåller mer detaljer. En annan anledning till att den ser mer detaljer är att den antar att det reflekterade ljudet kommer från bara en ruta medan Bayesiska algoritmen i princip antar att den kommit från flera. Det är också en anledning till att väggarna är bättre för Bayesiska algoritmen. Den markerar helt enkelt flera av rutorna i taget som ockuperade. Så väggar är ett exempel där antagandet att bara en av rutorna i område ett kan vara ockuperad, inte stämmer så bra. Men skulle det t.ex. vara ett hål någonstans i väggen, så skulle det vara sämre att anta att flera rutor är ockuperade. Den egna algoritmen ”fyller inte igen” hål som upptäckts i en vägg. Det kan den Bayesiska algoritmen däremot göra. Totalt sett verkar kartorna vara lite bättre för den Bayesiska algoritmen. Men det ska också nämnas att avstånden var korta, oftast inte längre är 20 cm. Den Bayesiska algoritmen kan ju tänkas ge lite konstigare resultat för längre avstånd. Att mätningarna oftast gjordes vinkelrätt mot objekten är också något som gynnar den Bayesiska algoritmen lite, eftersom den antar att det är störst sannolikhet att objektet är rakt fram. Eftersom de båda algoritmerna skiljer sig mycket från varandra och kartorna ändå är ganska lika, så tyder det på att det är den prestandan man kan förvänta sig.

Kapitel 4

Slutsats

Delmålen för det här examensarbetet var att hitta en lämplig sonar för Kheperan, montera den och få den att fungera tillsammans med roboten och slutligen testa konstruktionen. Alla målen nåddes. Den färdiga konstruktionen är gjord för att inte ta onödig processorkraft medan sonarerna används. Det gör att den ska kunna användas i sammanhang där området den befinner sig i är för stort för den korta räckvidden hos Kheperans egna ir-sensorer. Men som man kan se på de färdiga kartorna i avsnitt 3.2, så blir inte kartläggningen med sonarerna tillräckligt bra när roboten befinner sig långt från objekten. För att få en bra kartläggning av omgivningen och för att kunna upptäcka detaljer måste roboten fortfarande undersöka objekten på nära håll. Så ett förslag är att använda sonarerna för att få en grov bild av omgivningen och sedan undersöka intressanta delar av omgivningen på närmre håll. Den blir då effektivare än om enbart ir-sensorerna används, eftersom den då skulle vara tvungen att fysiskt besöka alla delar av området. Om roboten används på det sättet kan jag rekommendera att prova att använda algoritmen jag utvecklade från HIMM algoritmen, och som står beskrivet i avsnitt 3.2. Den ger brusigare kartor men där också mer detaljer har behållits än för Bayesiska algoritmen. Det gör att tomma områden är mer säkra att de faktiskt är tomma och behöver då inte undersökas närmare. En nackdel med konstruktionen är att det måste göras flera sonarmätningar för att få en bra längdupplösning. Ett bättre sätt att mäta tiden skulle snabba upp mätningarna betydligt.

Framtida arbeten skulle kunna vara att istället utrusta Kheperan med den bättre SRF08 sonarn. Eftersom den använder I2C bussen för kommunikationen med roboten så kan även andra komponenter som t.ex. en kompass kopplas på roboten. En kompass skulle vara till stor hjälp då osäkerheten i robotens riktning inte längre skulle vara något problem. En dyrare förbättring skulle vara att utrusta Kheperan med en laser istället för sonar. Med en laser försvinner osäkerheten om i vilken riktning från roboten objektet som upptäckts finns. Men eftersom Kheperan är så liten så är det nog fysiskt omöjligt. En laser drar mycket ström och är antagligen är för stor och tung. Slutligen skulle man kunna gå djupare in på kartläggningen med hjälp av sonarerna. Man skulle kunna undersöka algoritmer för att kartlägga omgivningen, samt algoritmer för att kunna bestämma var och åt vilka håll sonarmätningarna ska göras på. Det så att kartläggningen blir så effektivt som möjligt.

Litteraturförteckning

[1] J. Borenstein, H. R. Everett och L. Feng, *Where am I? Sensors and Methods for Mobile Robot Positioning*, 1996.

[2] K. Demirli och I. B. Türksen, *Sonar based mobile robot localization by using fuzzy triangulation*, 1999.

[3] Franzi Edo (K-Team S.A.), *Khepera BIOS 5.0 Reference Manual*, 1998.

[4] K-Team, *General IO-Turret user manual version 2.1*, 1999.

[5] K-Team, *GNU C based cross-compiler for the Khepera robot version 5.0*, 1999.

[6] K-Team, *Khepera bus and turret specifications*, rev 1.2, 2000.

[7] K-Team, *Khepera user manual version 5.0*, 1998.

[8] Gisbert Lawitzky, Wendelin Feiten och Marcus Möller, *Sonar Sensing for low-cost indoor mobility*.

[9] Motorola, *CPU32 Central Processor Unit Reference Manual*, rev. 1, 1990.

[10] Motorola, *MC688331 User´s Manual*, rev. 1, 1993.

[11] Ulrich Nehmzow, *Mobile Robotics: A Practical Introduction*, Springer-Verlag.

[12] Robin R. Murphy, *Introuktion to AI Robotics*, MIT Press, 2000.

[13] Sebastian Thrun, *Robotic Maping: A Survey*, 2002.

[14] www.k-team.com (verifierad 31/3 2004)

[15] www.polaroid-oem.com (verifierad 31/3 2004)

[16] www.robot-electronics.co.uk (verifierad 31/3 2004)

Appendix A

Exempelkod

Nedan följer ett exempel på kod för att göra ett medelvärde av 100 st. mätningar och sedan skriva ut resultatet på skärmen i enheten millimeter. Det görs kontinuerligt nya mätningar hela tiden. Konfigurationen är sådan att programmet kompileras på en dator. Det laddas sedan upp på roboten från datorn genom en seriellkoppling. Direkt när programmet laddats upp körs det av roboten. Eventuella utskrifter skickar roboten tillbaka via seriellkopplingen. För att se dessa utskrifter så måste minicom eller liknande program vara igång. För mer information se [3,5].

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

static void ir(){
  static int32 st=0;
  static uint32 mtime0=0;
  static uint32 tot=0;
  static uint32 antal=0;
  uint32 mtime1;
  uint32 ctime;
  ctime=tim_get_ticcount();
  var_disable_irq();
  var_put_extension(36,1); //Nollställer flip-flopen mot avbrottsingången (avsnitt 2.3.1.1 och bild 2.8).
  if( st==0) {
    st=1;
    mtime0=ctime;
  } else {
    st=0;
    mtime1=ctime-mtime0;
    if(antal<99) {
      antal++;
      tot=tot+mtime1;
    }
    else {
      antal=0;
      tot=tot+mtime1;
      tot=tot*342/2/100; //Gör om ms till mm.
      RESERVE_COM;
      printf("%d \n",tot); //Skriver ut resultatet på skärmen.
      RELEASE_COM;
      tot=0;
    }
  }
  var_enable_irq();
}
```

```
static void sensor(){  //Ny mätning var 81:e ms.
  for(;;){
    tim_suspend_task(80);
    var_put_extension(32,1);  //Gäller för första (vänstra) sonarn. För andra ändra 1 till 6.
    tim_suspend_task(1);
    var_put_extension(32,0);  // Gäller för första (vänstra) sonarn. För andra ändra 0 till 2.
  }
}

static void empty(){ //Tom process behövs. Se avsnitt 2.3.1.1 för mer information.
  for(;;){
  }
}

void main(void){
  int32 s;
  com_reset();
  var_reset();
  var_put_extension(32,0); //Nollställer de digitala utgångarna.
  var_set_irq_vector(ir);
  tim_suspend_task(10);
  var_put_extension(36,1); //Nollställer flip-flopen mot avbrottsingången (avsnitt 2.3.1.1 och bild 2.8).
  tim_suspend_task(1);
  var_enable_irq();
  tim_suspend_task(10000);
  s=install_task("empty_p",800,empty);
  s=install_task("sensor_p",800,sensor);
}
```

Appendix B

Kretskortslayouten

Nedan i bild B.1 finns layouten med enbart ledningssidan i skala 2:1. I bild B.2 finns istället komponenternas positioner markerade, inklusive hålen.

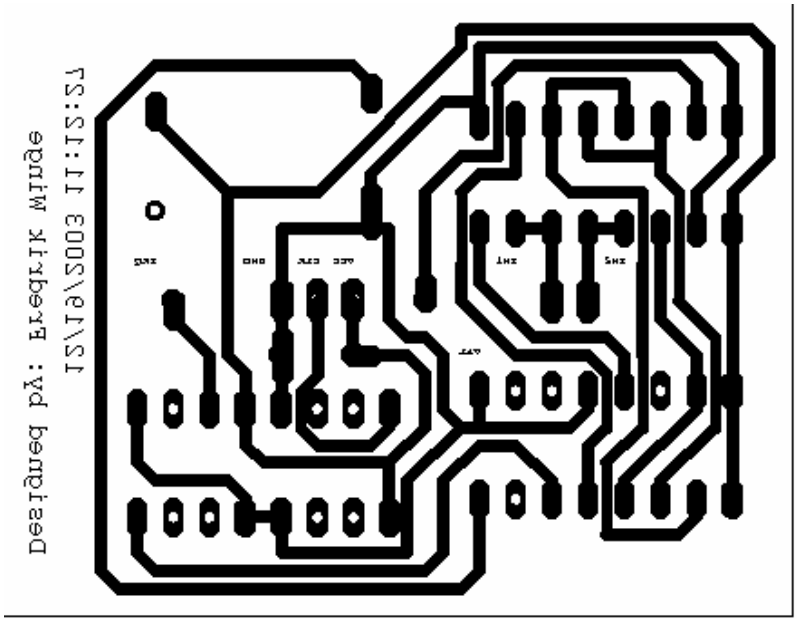

Bild B.1 Layouten med enbart ledningssidan i skala 2:1.

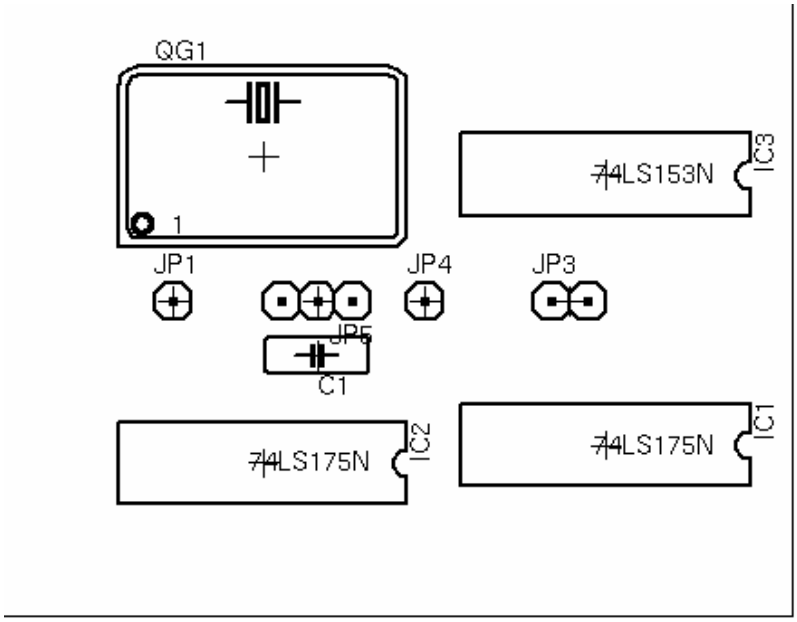

Bild B.2 Komponenterna och hålens positioner i skala 2:1.