Fotbollsspelande robothundars seende

Carl-Emil Kjellstrand
dat98cek@ludat.lth.se

David Pettersson
dat99dpe@ludat.lth.se

4 januari 2003

Sammanfattning

Denna rapport behandlar problem då en autonoma robot skall lokalisera sig med hjälp av seende. Den miljö roboten förutsätts arbeta i har känd utsträckning och kända landmärken. För att roboten skall kunna lokalisera sig i denna miljö krävs att den kan känna igen landmärkena. En utmärkande egenskapen hos dessa landmärken är deras schackrutiga textur som ger skarpa kontraster. Denna rapport redovisar utvecklingen och implementationen av en metod för att identifiera dessa landmärken i bilder tagna med robotens kamera. Den framtagna metoden lyckas med identifieringen i mer än hälften av dessa bilder.

Abstract

This paper covers problems with an autonomous robot's self-localisation using vision. The environment in which the robot will work and the landmarks therein are both known. In order for the robot to self-localise in this environment, they must be able to recognise the landmarks. The landmarks' main feature is their checkered pattern which gives sharp contrast. This paper describes the development and implementation of a method for identifying theis landmarks in pictures taken with the robot's camera. The developed method successfully identifies the landmarks in more than half of the pictures taken.

Förord

Vi vill rikta ett speciellt tack till vår handledare Jacek Malec samt Per Kjellstrand och Deanna Neal för deras hjälp med korrekturläsning, stöd och engagemang. De många sena kvällarna, långa nätterna och tidiga morgnarna skvallrar om att det varit ett roligt examensarbete. Till slut vill vi även rikta ett speciellt tack till Kevin Bacon och folket i Örebro för det ömsesidiga utbytet samt det stöd vi fått under arbetets gång.

Innehåll

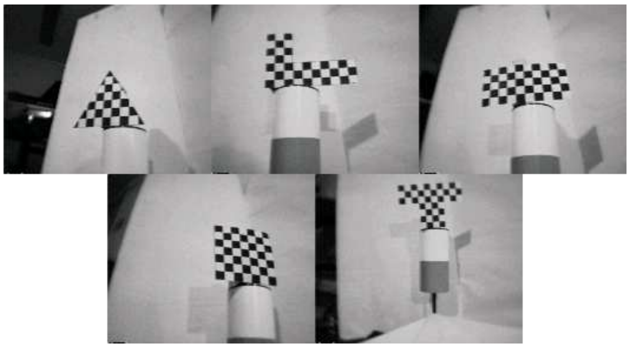
Kapitel 1

Introduktion

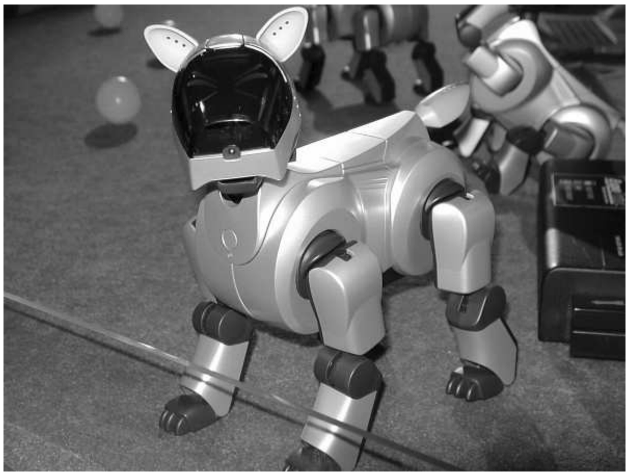
1.1 Syfte

RoboCup är en internationell robotfotbollstävling som årligen anordnas av RoboCup Federation[1]. Det lag Sverige ställer upp med heter Team Sweden[2] och har deltagit i RoboCups Sony Legged Robot League sedan 1999. I denna division använder alla lagen Sonys fyrbenta hundrobot AIBO[3].

Hittills har roboten positionerat sig på spelplanen med hjälp av olikfärgade markörer som sitter längs spelplanens sidor. Arrangörerna för RoboCup vill undersöka möjligheten för robotarna att positionera sig utan färgmarkörer och istället använda schackmönstrade figurer. Syftet med arbetet är att utreda om detta är praktiskt genomförbart.



Figur 1.1: De möjliga figurer som kan förekomma i bilden.



Figur 1.2: AIBO-hunden.

1.2 Problemställning

I en bild ska en av de möjliga figurerna (se figur 1.1) kunna urskiljas eller skall avsaknaden av en sådan figur i bilden identifieras. En mycket viktig aspekt i det hela är att identifieringen aldrig får vara felaktig, då det kan få drastiska konsekvenser. Exempelvis kan hundens uppfattning om var den befinner sig bli felaktig. Ytterligare en aspekt är att det måste göras i realtid. Det finns begränsad beräkningskraft i roboten och bara en liten del av den kan användas för att urskilja eventuella mönster i bilden. Detta på grund av att hundens huvuduppgift är att spela fotboll. Att känna igen dessa figurer är endast en deluppgift i fotbollsspelandet. Ett krav är att det skall gå att analysera minst ett par bilder i sekunden. Problemet är att finna en metod för att identifiera om någon av de möjliga figurerna finns i en given bild.

1.3 Om AIBO

Den hårdvaruplattform som användes var Sonys hundliknande robot AIBO, se figur 1.2. AIBO är hårdvarumässigt sett en komplex plattform. Den kan utföra diverse rörelser med benen och har kamera, mikrofon, accelerometer och radiokommunikation. För att utnyttja denna hårdvaruplattform har Sony även tagit fram en utvecklingsmiljö vid namn Open-R[4]. En av plattformens större brister är kameran som har dålig skärpa vilket gör det svårare att t ex urskilja skarpa kanter i ett schackmönster.

Kapitel 2

Metoder

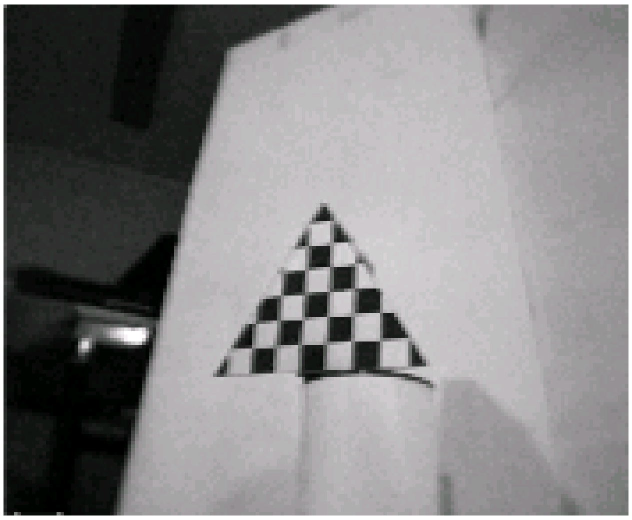
2.1 Inledning

Tiden från inläsning av bilden fram tills den tidpunkt då metoden slutförts delas in i fyra steg. Första steget utför kantigenkänning. Man mäter intensitetsskillnaderna mellan pixlarna i ursprungsbilden och från dessa bygger man en ny bild som illustrerar kontrastområdena i bilden. Denna bild görs sedan om till en svartvit bild med hjälp av ett tröskelvärde. Andra steget söker efter rutmönster i den svartvita bilden. För varje vit pixel undersöks om den ingår i en mängd av vita pixlar som är inneslutna av svarta pixlar. Om så är fallet är det förmodligen resultatet av en underliggande schackruta i ursprungsbilden. Centrum för sådana områden samlas i en mängd. Tredje steget skapar ett minsta uppspännande träd från denna mängd. Därefter klipps oönskade delar bort från trädet. Sedan skapas ett hölje från det klippta trädet. Fjärde steget försöker identifiera höljet som en figur i databasen över möjliga figurers höljen.

2.2 Införskaffandet av bilden

Den initiala bilden b_{yuv} hämtas från Open-R. Open-R är de bibliotek man använder för att komma åt hårdvaran i Sonys AIBO. Bilden som Open-R levererar är i YUV-format. I en YUV bild beskriver Y-komponenten ljusintensitetsinformation medan U och V komponenterna beskriver färginformation.

För att kunna hitta kontraster i b_{yuv} plockas Y-komponenten ut och sätts in i den nya bilden b_y som nu blir en ganska god representation av de intensitetsvärden som finns i b_{yuv}.



Figur 2.1: Bild b_y.

2.3 Bildernas storlek och format

Då det talas om bilder i denna rapport så menas normalt sett bilder av storleken 176×144 pixlar. En pixel består i ursprungsbilden b_{yuv} av tre värden. Dessa värden är Y, U och V. I senare bilder finns endast ett värde kvar. Värden i bilderna kan anta heltalsvärden mellan 0 och 255.

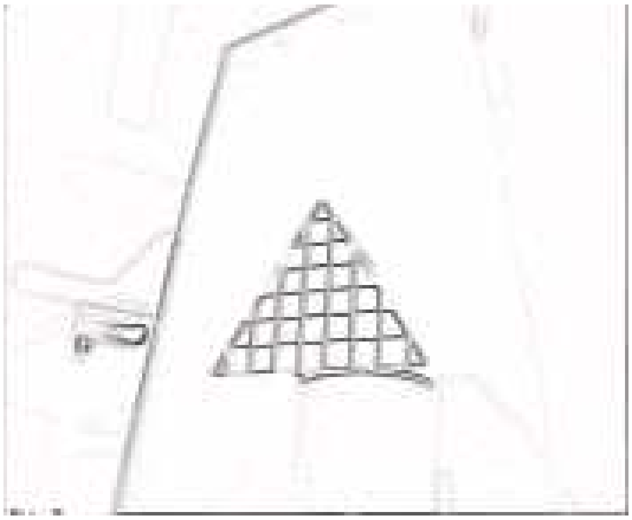
2.4 Raffinering av pixeldata

Ur b_y skall de kontraster som finns tas fram och markeras i den nya bilden b_i enligt formel 2.1. Figur 2.1 visar bild b_y och figur 2.2 b_i.

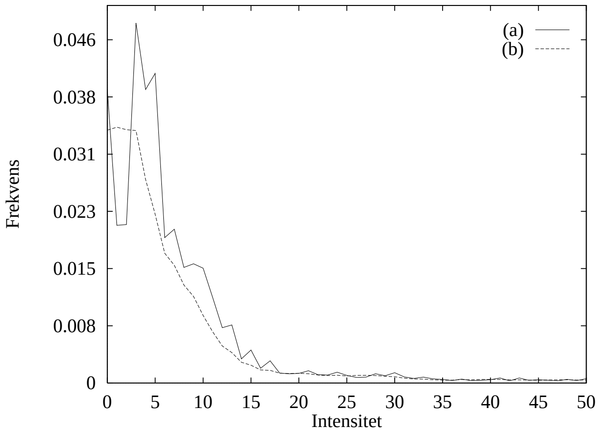
$$\begin{aligned} b_i[x][y] = \max(\quad & |b_{yuv}[x][y] - b_{yuv}[x+1][y]|, \\ & |b_{yuv}[x][y] - b_{yuv}[x][[y+1]|, \\ & |b_{yuv}[x][y] - b_{yuv}[x+1][y+1]| \quad) \end{aligned} \tag{2.1}$$

För att hitta ett gränsvärde skapas en frekvenstabell f innehållande antalet pixlar av en viss intensitet. Frekvenstabellen visar hur vanligt förekommande en viss intensitet är i bilden. För att inte nästa steg skall bli känsligt för lokala avvikelser i f så jämnas värdena i f ut enligt formel 2.2.

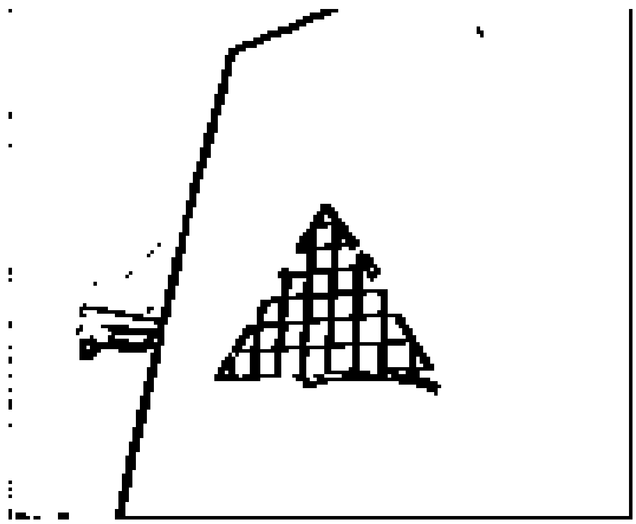
$$d_u[i] = \sum_{j=0}^{4} d[i+j], \quad i \in [0 \dots 255-4] \tag{2.2}$$



Figur 2.2: Bild b_i.



Figur 2.3: Frekvensdiagram över pixlarna i b_i.



Figur 2.4: Bild b_{sv}.

Frekvenstabellen f_u innehåller mycket jämnare värden och eventuella toppar eller dalar är utjämnade vilket visas av linje (b) i figur 2.3.

En stegning genom f_u görs nu från vänster till höger med början på den stapel som har index 0 i b_i. Derivatan för den raka linje som spänns upp mellan topparna på staplarna $f_u[i]$ och $f_u[i + k_h]$ beräknas i varje steg. Då derivatan är mindre än k_d sätts gränsvärdet $g = i$ och stegningen avbryts. Stegningen har då passerat de höga staplarna som alla de vita och ljusgrå pixlarna i b_i gett upphov till. Detta för att synliggöra de starka kontrasterna i bilden, däribland schackrutemönstret. Konstanterna k_h och k_d har valts till 40 respektive $\frac{1}{64}$, då detta ger ett g som i sin tur ger bra svartvita bilder utifrån de bilder som kameran levererar. Därefter används g för att skapa en ny bild b_{sv} enligt formel 2.3.

$$b_{sv}[x][y] = \begin{cases} \text{svart} & \text{om } b_i[x][y] > g \\ \text{vitt} & \text{om } b_i[x][y] \leq g \end{cases} \tag{2.3}$$

Detta ger en bild b_{sv} som är svart där pixlarna i b_i har höga värden och vit i övrigt, se figur 2.4. Om g ligger utanför intervallet $[k_s \ldots k_t]$ så avbryts hela figursökningen för den aktuella bilden. Den innehåller med stor sannolikhet för mycket brus eller saknar schackmönster. Konstanterna k_s och k_t ovan sattes till 12 respektive 102. Genom att välja lämpliga värden på konstanterna slipper man bilder som antingen innehåller för många eller för få svarta pixlar.

2.5 Sökning efter rutmönster

En av de egenskaper som utnyttjas hos figurerna är att de är schackrutiga. Vanliga linjeigenkänningsalgoritmer fungerar för att finna linjer i bilden, men är alltför tidsödande. För att lokalisera den önskade figuren söker man efter schackrutor. För att finna dessa rutor stegar man över alla pixlar och undersöker huruvida de ingår i en ruta.

För att avgöra om pixeln ingår i en sådan ruta utförs en sökning. Antal pixlar som hittas under sökningen måste ligga mellan $[s_{undre}, s_{ovre}]$ för att området skall anses vara en schackruta. Den övre gränsen s_{ovre} är till för att undvika att en enskild sökningen breder ut sig över hela bilden. Den undre gränsen s_{undre} hindrar att de små områden som kan uppkomma på grund av brus betraktas som ett resultat av schackmönstret. Varje pixel undersöks enligt följande steg.

Första steget kontrollerar att sökningen sker utifrån en pixel som hittills är osökt. Detta görs genom att en lista med redan sökta pixlar upprätthålls. Om pixeln är osökt går sökningen vidare till steg två, annars upprepas sökningen på nästa pixel i bilden.

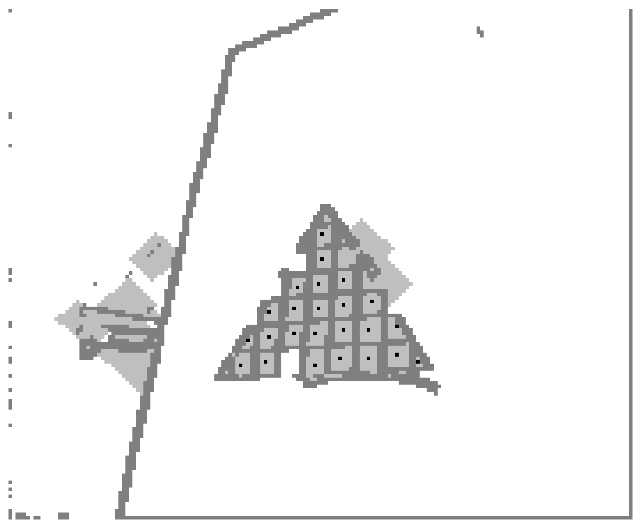
Andra steget kontrollerar om den aktuella pixeln är vit. Om så är fallet går sökningen vidare till steg tre, annars upprepas sökningen på nästa pixel i bilden.

Tredje steget kontrollerar att det finns en svart pixel strax till vänster om den aktuella pixeln. Om så är fallet går sökningen vidare till steg fyra, annars upprepas sökningen på nästa pixel i bilden.

Fjärde steget består i att stega uppåt 15 steg, nedåt 15 steg och åt höger 15 steg. Alla dessa stegningar görs utifrån samma utgångspunkt. Skulle svarta pixlar finnas i alla riktningar inom 15 steg är troligtvis området inneslutet och sökningen fortsätter till steg fem, annars upprepas sökningen på nästa pixel i bilden.

Femte steget gör en breddenförstsökning utifrån den aktuella pixeln. Sökningens utbredning begränsas av svarta pixlar. Sökningen avbryts då utbredning inte längre är möjlig eller då ett maximalt antal pixlar avsökts. Avsökta pixlar bokförs i listan som omnämns i steg ett. Skulle sökningen avstanna för att den når svarta pixlar har vi funnit ett inneslutet område som kan tänkas tillhör schackmönstret.

Då man funnit ett område beräknas en ungefärlig mittpunkt (x, y) i detta område. Denna punkt ligger mitt mellan de horisontella, respektive de vertikala extrempixlarna. Dessa mittpunkter samlas i en mängd M och kommer fortsättningsvis kallas för hörn och inte mittpunkter. Figur 2.5 visar var dessa hörn är belägna i bilden.



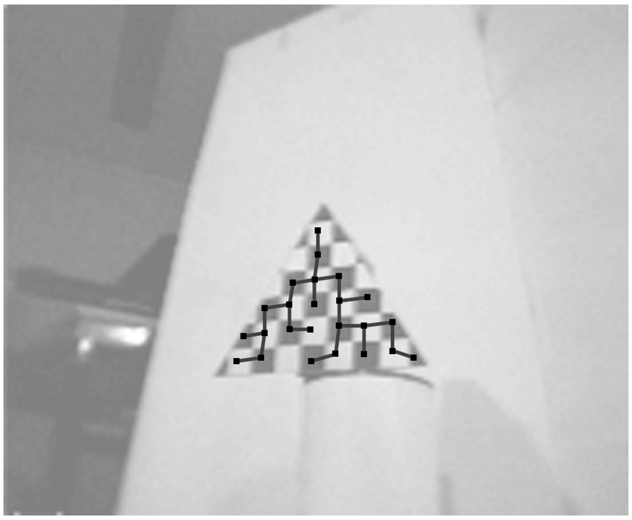
Figur 2.5: Mittpunkter funna under sökning efter rutmönster.

2.6 Graf- och vektorrepresentation

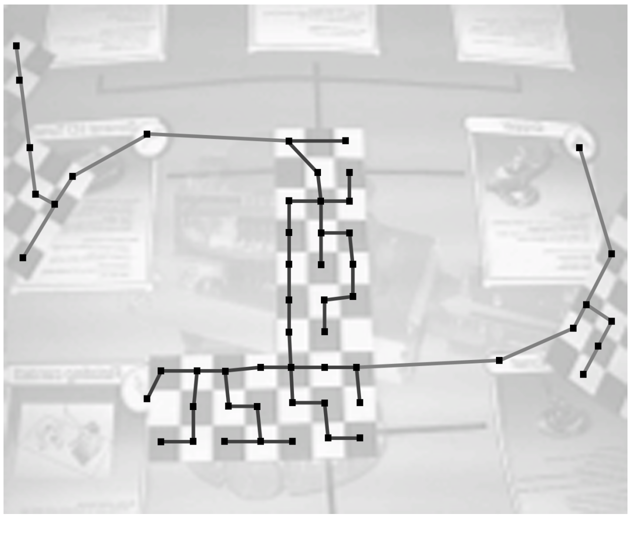
För att konstruera en vektorrepresentation av figuren används de hörn i mängden M som upptäcktes i föregående steg. En del av de hörnen härstammar inte från schackrutorna i den figur som eftersöks utan är resultatet av brus. De måste således avlägsnas. Ett problem som uppstår är att figuren ännu inte är känd och därför är det inte trivialt att avgöra vilka hörn i M som härstammar från figuren samt vilka som är resultatet av brus.

För att rensa ut de hörn som härstammar ur brus i bilden byggs ett minsta uppspännande träd T (figur 2.6). Medellängden av alla kanter i T beräknas och sedan delas dessa in i två mängder: en mängd K_n innehållande de kanter vars längd ligger nära medellängden och en mängd K_a med kanter vars längd avviker 70% från medellängden. De kanter som avviker visas som gråa i figur 2.7. Övriga kanter är svarta i figuren. Värdet 70% valdes då det är tillräckligt litet för att sortera bort det mesta bruset i figuren. Värdet ger även utrymme åt att bilden kan få vara tagen ur en sned vinkel. En sådan ger upphov till perspektivförvrängningar av figuren. Dessa leder till att de kanter som ingår i figuren blir av varierande längd. Ett för lågt värde hade kunnat begränsa i hur sned vinkel bilden hade kunnat tas.

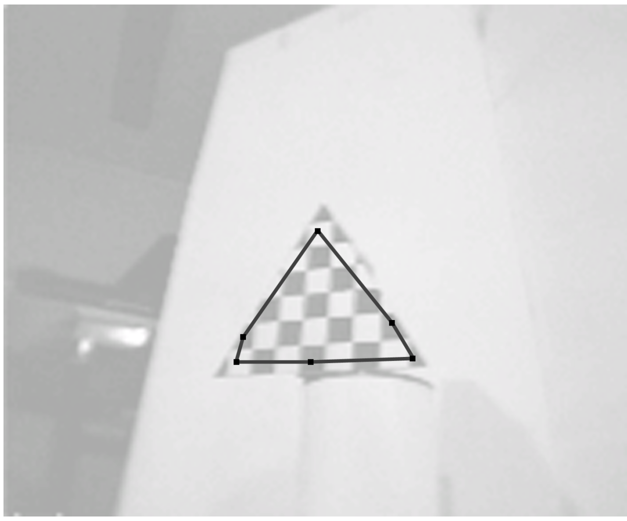
Ur K_a väljs en kant. Denna kant tas sedan bort ur T som då kommer att delas i två delar. T sätts att vara den större av dessa. Detta upprepas tills K_a är tom. Då antas det att återstående hörn i T är de som ingår i figuren. Alla hörn i T läggs i den nya mängden M_{fig}.



Figur 2.6: Minsta uppspännande träd.



Figur 2.7: Önskade och oönskade kanter i trädet.



Figur 2.8: Konvext hölje uppspänt av hörnen i M_{fig}.

För att få en inledande uppfattning om figurens utsträckning och form konstrueras ett konvext hölje H_{konvex} ur hörnen i M_{fig}. Figur 2.8 visar hur H_{konvex} kan se ut. H_{konvex} visar dock inte några konkava detaljer. För att få ett detaljrikare hölje kollapsas delar av H_{konvex}. Det nya höljet benämns H. Fler figurer kan identifieras unikt av sina höljen då höljena bättre följer de underliggande figurernas former. Böjningen av höljena görs på följande vis:

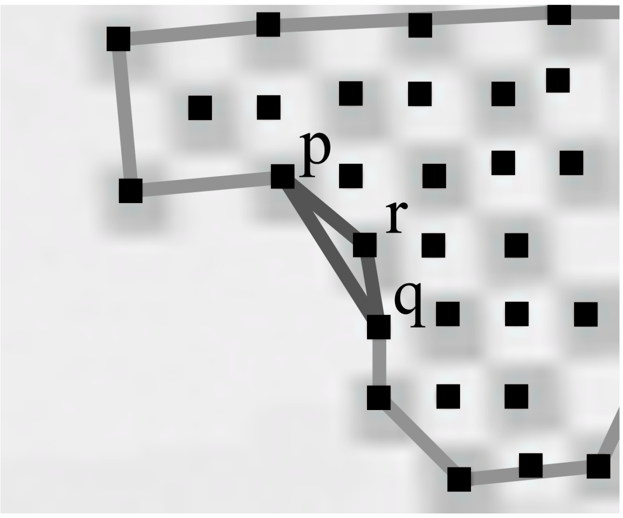
1. Ta godtycklig kant (p, q) i det konvexa höljet som är längre än medellängden gånger 2.12.

   Värdet 2.12 används beroende på att den kortaste kant (p, q) i figur 2.9 som skall böjas in är av längden 2.24 ($\sqrt[2]{1^2 + 2^2} = 2.24$). En kant som är 2.00 lång bör inte böjas in då detta skulle kunna ge upphov till det hack som visas i figur 2.10 av kant (r, q) och (r, q). På så vis fås det värde som ligger mellan 2.00 och 2.24, nämligen 2.12.

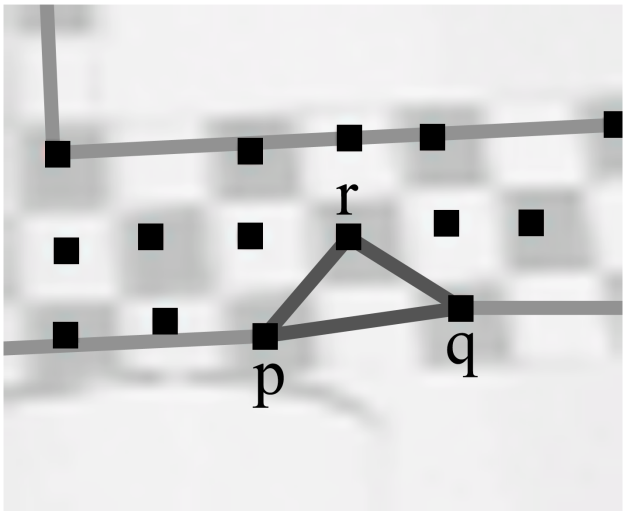
2. Välj det hörn r ur mängden M_{fig} där summan av längderna hos kanterna (p, r) och (r, q) är som minst.

3. Om $|180 - v| > \alpha$ där v är vinkeln mellan kanterna (p, r) och (r, q) tas kanten (p, q) bort ur höljet och ersätts med kanterna (p, r) och (r, q). Vinkeln v visas av figur 2.11.

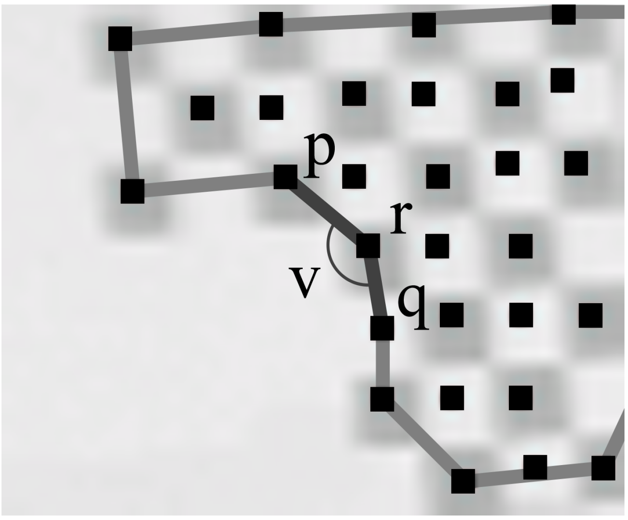
Villkoret $|180 - v| > \alpha$ finns till för att förhindra att en kollaps görs då den kommer leda till en vinkeln v inte är stor nog för att unikt identifiera figuren. Ett lämpligt värde för α är 10.



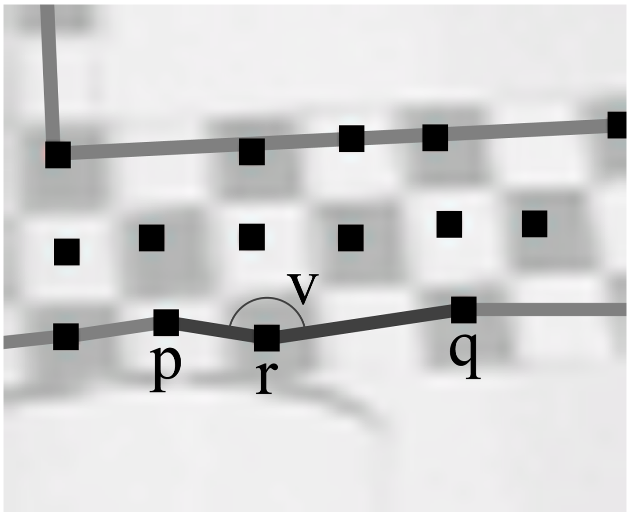
Figur 2.9: Kant (p, q) är här 2.24 rutor lång.



Figur 2.10: Kant (p, q) är här 2.00 rutor lång.



Figur 2.11: Vinkeln v vid kollaps.



Figur 2.12: Vinkeln v vid utjämning.

Som en sidoeffekt av kollapsen skapas en hel del korta kanter. Eftersom de inte tillför någon faktiskt information om figurens utseende slås två kortar kanter samman till en längre, om $|180 - v| < \beta$ där v är vinkeln mellan de två korta kanterna (p, r) och (r, q). Allt detta illustreras av figur 2.12. Av naturliga skäl måste $|180 - v| < \beta$ villkoret vara hårdare än kollapsvillkoret. Det omvända skulle leda till att utjämningen kunde ske på kanter som skall vara kollapsade och därmed omintentgöra korrekta kollapser. Konstanten β måste således vara mindre än α. Ett lämpligt värde visade sig vara 5.

2.7 Identifiering

Då höljet är kollapsat skapas en lista L över kanternas längd samt vinklarna mellan de på varandra följande kanterna i höljet. Listans första post håller information om hur många par listan består av. Efter detta följer information om kanterna i moturs ordning.

[*listlängd, längd, vinkel, längd, vinkel, ...*]

längd är i detta fallet längden hos den aktuella kanten och *vinkel* är den vinkel som bildas mellan den aktuella kanten och den efterföljande i höljet. Vinkeln kan även ses som vinkel v i figur 2.12. En ideallista för en triangel kan se ut på följande vis:

$$L_{triangel} = [3, 10, 120, 10, 120, 10, 120]$$

Själva identifieringen är en jämförelse där L jämförs mot ett antal på förhand bestämda listor som var och en motsvarar en figur, vid identifieringen tillåts att längden avviker med ± 2 rutor och vinklar med 5 grader. Om inte avrundningen gjordes skulle metoden nästan aldrig lyckas identifiera en figur som fanns i en given bild.

2.8 Utvecklingsmiljö

Eftersom det tidigt insågs att flera olika algoritmer behövde provas och att detta var tidskrävande, valdes ett programmeringsspråk för prototypkodning. Den ursprungliga koden var implementerad i Python och utgick ifrån enkla bilder som tagits med en webbkamera. Denna kodbas var strukturerad som en pipeline som i princip tog in rådata, behandlade denna i ett antal steg och slutligen matade ut ett resultat som berättade vilken slags bild som känts igen. Till hjälp har man ett mjukvaruutvecklingspaket vid namn Open-R som ger tillgång till den faktiska hårdvaran som servon, kamera, högtalare och mikrofon. Program som skrivs för Open-R exekveras parallellt.

Kapitel 3

Resultat

En uppsättning tester har gjorts och resultaten från dessa presenteras i tabellen nedan.

Figur	Antal	Lyckade	% lyckade	Felaktiga
L-form	182	100	55%	0
Rektangel	140	78	56%	0
Kvadrat	152	103	68%	0
T-form	192	124	65%	0
Triangel	155	64	41%	0
Totalt	821	469	57%	0

Det som kan utläsas av tabellen 3 är att metoden klarar av att identifiera 57% av de bilder i de serier den testats på. Dessa serierna innehöll endast bilder med figurer i där figuren ligger helt i bilden. Bakgrunden i dessa serier är densamma som ses i flera av figurerna i denna rapport. Man kan även utläsa att triangeln är den svåraste figuren att identifiera och kvadraten är den lättaste. Dessutom kan man dra slutsatsen att metoden är stabil med avseende på särskiljning. En bild med en L-formad figur i kommer med mycket stor sannolikhet inte att identifieras som någon annan figur än just en L-formad figur. Däremot går många bilder till spillo som förmodligen skulle kunnat identifieras med en mindre strikt metod.

Kapitel 4

Diskussion

4.1 Förbättringar av metoden

Raffinering av pixeldata

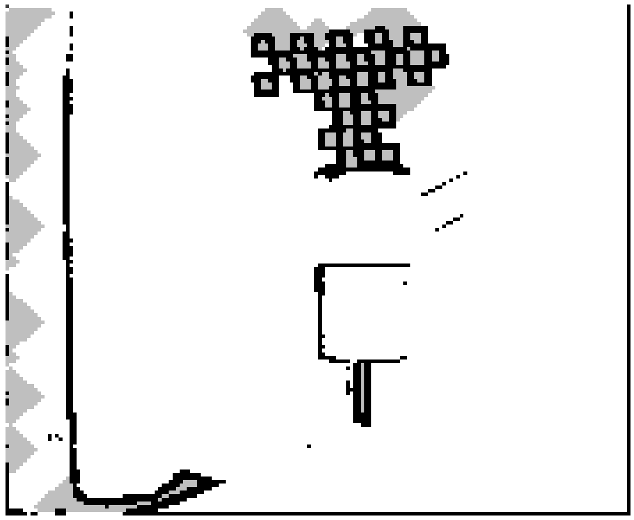
Den mask som använts för att hitta kontraster är en 2×2 mask. Denna mask fungerar men är inte så bra som man skulle kunna önska. En större mask skulle ge en bättre bild av det underliggande rutmönstret men skulle vara för tidskrävande.

Sökning efter rutmönster

Den del av igenkänningsmetoden som behandlar sökning efter rutmönster är uppdelad i fem steg. Fjärde steget i denna sökning kan förbättras tidsmässigt. Stegningen görs utifrån en punkt $p = (x, y)$ och stegningen görs 15 steg uppåt samt 15 nedåt. Det stegas alltså 30 steg i y-led men endast 15 i x-led. Stegning i x-led bara görs åt vänster eftersom punkten till höger om p är svart i b_{sv} . Således avstannar stegningen åt höger direkt. Detta leder även till att sökningen täcker av ett område som är dubbelt så stort i y-led som i x-led.

Sökningen bör inte ske uppåt eller nedåt från p. När den pixel som ligger precis till höger om p är svart i b_{sv} är det sannolikt att det finns en svart pixel i b_{sv} precis under eller över p. Förutom att sökningen söker ett område som inte är kvadratisk så utgår ju uppåt och nedåtsökningen från samma pixel som den i vänstersökningen då de egentligen borde utgå från en pixel som ligger $\frac{d_y}{2}$ steg till vänster om p. Om d_y är antalet steg som stegades innan en svart pixel hittades i vänsterstegningen.

Hur detta ser ut illustreras av figur 4.1 och de gråa områdena i bilden är de avsökta områdena. Mycket av sökningarna på vänstersidan av bilden görs



Figur 4.1: Sökta områden.

helt i onödan. Detta på grund av att kontrollsökningen i sin nuvarande form inte klarar av den typen av brus i bilden. Detta löses dock med ovan givna metod.

Houghtransformen

Den ursprungliga idén var att analysera bilden utifrån de linjer som uppkommer efter kantigenkänningen. En klassisk algoritm för att ta reda på vilka linjer som finns i en bild är Houghtransformen[5]. Houghtransformen provades, men visade sig vara för långsam för att kunna användas.

Graf- och vektorrepresentation

Misstankar om att kollapsen av höljet på det hela försämrade identifieringen av de möjliga figurerna finns men har inte bekräftats. Den försämrade identifieringen antas kunna bero på att kanter som inte kollapsat. Därigenom förstörs data. Huruvida kollapsen påverkar negativt totalt sett eller inte återstår att undersöka. Kollapsen hjälper emellertid till att identifiera mer generella figurer.

Identifiering

Flera metoder för att identifiera listor av vinklar och avstånd provades. En metod där listan endast består vinklar användes tidigt i projektet. Istället för att stoppa in både avstånden och vinklarna i listan sattes endast vinklarna in i listan. Vinklarna sattes in motsvarande så långt in som de hade funnits i höljet om man betraktar höljet som en linje med vinklar på. Exempel på en kvadrat:

$$L_{org} = [0, 0, 90, 0, 0, 0, 90, 0, 0, 0, 90, 0, 0, 0, 90, 0]$$

$$L_{mask} = [0.05, 0.20, 0.50, 0.20, 0.05]$$

Listans längd normaliseras och får en ny längd som är oberoende av hur långt höljet är. En ny utjämnad lista L_{utj} skapas med hjälp formel 4.1.

$$L_{utj}[i] = \sum_{j=0}^{4} L_{org}[i + j] * L_{mask}[j], \quad i \in [0 \ldots 255] \tag{4.1}$$

I detta exempel skulle L_{utj} komma att se ut som nedan efter att ovan nämnda tilldelning skett på alla positionerna. Observera att längden på listorna i detta exempel är kortare än 255 då ovan beskrivna utjämning fungerar lika bra på 16 som på 256 poster långa listor.

$$L_{utj} = [9, 18, 45, 18, 9, 18, 45, 18, 9, 18, 45, 18, 9, 18, 45, 18]$$

Idén var att sådana listor skulle vara enkla att särskilja. Listorna skulle särskiljas genom att en standardavvikelse mellan alla positioner i två sådana listor skulle kunna beräknas. Dels en testlista som skulle innehålla värden motsvarande en av de möjliga figurerna och dels L_{utj}. Den testlista som gav minst standardavvikelse borde då rimligtvis ange vilken figur som fanns i bilden. Resultaten var mycket dåliga och gick inte att lita på. Detta ledde till att andra metoder testades och denna metod förkastades.

4.2 Hantering av flera figurer

Som tidigare nämnts fungerar de metoder som implementerats på så vis att de klipper bort oönskade delar av bilden. Detta får effekten att partiella figurer kan klippas bort. Detta får konsekvensen att bara en figur av flera kan kännas igen i samma bild. Klippningen av det minsta uppspännande trädet syftar till att avgöra vilka hörnmängder som finns i bilden och ta bort alla utom det största. Ett alternativ vore att behålla alla hörnmängder som är stora nog för att representera en figur och försöka identifiera dessa mot de kända figurerna. På detta vis hade flera figurer kunnat identifieras i samma bild.

4.3 Avståndsbedömning

Avståndet från kameran till figuren beskrivs som en funktion av medellängderna på de kanterna i trädet T som också ingår i figuren. Även medellängden på kanterna på olika delar av figuren beräknas. Kvoten mellan de olika medellängderna gör det möjligt att avgöra bildens lutning i förhållande till kameran. På så vis kan t ex information fås om huruvida den vänstra sidan är närmare än den högra. Detta ger information om ungefär vilken vinkel figuren i bilden har i förhållande till kameran. Tillförlitligheten hos funktionen avtar med avståndet.

4.4 Kameror

Det finns två slags kameror som kan användas i dessa sammanhang. CMOS-kameror är en billig och ofta använd typ av kamera som dock ger stora problem genom sin långsamhet. CCD-kameror är betydligt dyrare, men är å andra sidan betydligt snabbare när det gäller uppdateringen. De kameror som använts har varit av typen CMOS. En konsekvens av detta är att bilden påverkas av skjuvning när kameran rörs snabbt i sidled. Detta får effekten att en lodrät linje ser ut att luta. En andra påverkan på bilden är att lysrör som blinkar med 50Hz lämnar varvat ljusa och mörka linjer på bilden. Dessa kan vara mycket kraftiga.

Kapitel 5

Slutsats

Målet med arbetet var att ta fram en metod för att känna igen väldefinierade figurer. Metoden skulle klara att hantera skräp och oskarpa bilder, men fick under inga som helst omständigheter ge upphov till en felaktig identifiering. För att lösa detta problem brukades en klassisk pipeline-liknande lösning där varje steg syftar till att antingen raffinera data inför nästa steg, eller avbryta om data är för dålig. Vårt examensarbete visar att det är möjligt att använda klassiska metoder för igenkänning på en plattform som AIBO.

Bilaga A

Terminologi

Hörn. Ett hörn är en punkt i planet. Exempelvis består ett hölje bland annat av en mängd hörn.

Kant. En linje som spänns upp mellan två hörn.

Konvext hölje. Det som skiljer ett konvext hölje från ett mer generellt hölje är följande. Om en stegning runt höljet görs medurs så kommer stegningen aldrig göra en vänstersväg för att kunna följa höljet utan bara högersvängar. Det konvexa höljet har således inga inbuktningar eller konkava ytor.

Minsta uppspännande träd. Kommer från det engelska Minimum Spanning Tree. Ett minsta uppspännande träd är en samling hörn och de kanter som förbinder alla dessa hörn med varandra på kortast möjliga sätt. Med kortaste avses den totala längden hos de kanter som behövs för att förbinda hörn med varandra. Två hörn är förbundna med varandra om det finns kanter mellan dessa två hörn antingen via andra hörn eller direkt mellan hörnen.

Rekursion. Se rekursion.

Litteraturförteckning

[1] RoboCups hemsida
http://www.robocup.org

[2] Team Swedens hemsida
http://www.aass.oru.se/Agora/RoboCup

[3] AIBOs hemsida
http://www.aibo.com

[4] Open-Rs hemsida
http://www.aibo.com/openr/

[5] Hough Transform
http://www.dai.ed.ac.uk/HIPR2/hough.htm