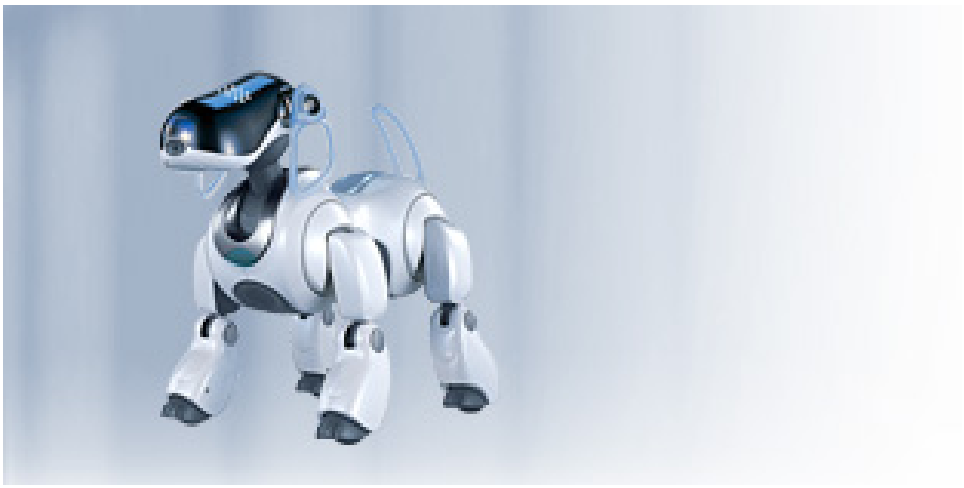
Kleur invariantie voor de Robocup challenge

Onderzoek naar een belichtings invariante methode voor object herkenning.



C.A.M. Pieterse,
St. nr. 0155071
Hoorn, 20-6-2004
UvA

Inhoudsopgave

1 Inleiding

Deze opdracht is naar aanleiding van het komende toernooi Robocup. Deze keer zijn er 3 technische competities, waaronder ‘Variable Lightning Challenge’. Hierin is het doel om de gebruikte Aibo’s robuust te maken tegen veranderingen in de belichting. De huidige methode van het team van de UvA is uiterst gevoelig voor de belichting en moet ook iedere keer worden gekalibreerd. Om mee te doen moet er een andere methode komen.

Het doel van dit onderzoek was het implementeren van de kleur invariante methoden voorgesteld door Jan-Mark Geusebroek en kijken of deze geschikt zijn voor toepassing in de Robocup challenge. In zijn proefschrift ‘Color and Geometrical Structure in Images’ beschrijft hij verschillende kleur invarianten die hiervoor gebruikt kunnen worden.

De opbouw van dit rapport is als volgt. In hoofdstuk 2 worden verschillende kleurmodellen besproken die gebruikt worden in dit onderzoek. Vervolgens wordt in hoofdstuk 3 worden verschillende toepassingen van object herkenning in computer vision besproken. In hoofdstuk 4 worden de resultaten gepresenteerd en conclusies weergegeven.

2 Kleurmodellen

Kleur laat zich op vele verschillende manieren omschrijven. De meest bekende manier is het RGB model dat gebaseerd is op de menselijke kleur beleving. Dit is echter lang niet de enige manier en is ook niet het beste model.
Kleur is namelijk een omschrijving van de energie verdeling van het licht. Licht omvat oneindig veel verschillende golflengten met ieder een eigen energie inhoud. Kleur modellen zijn een projectie van deze oneindig dimensionale ruimte met energie verdelingen op een N dimensionale ruimte.
Hieronder bespreek ik een aantal van de modellen die in de praktijk worden gebruikt.

2.1 Het RGB kleurmodel

Het RGB model is ontwikkeld in 1931 en beschrijft het kleuren spectrum op basis van de energie verdeling op 3 golflengten. Grofweg zijn dit de golflengten die overeenkomen met het menselijk rood (700nm), groen (546nm) en blauw (436nm). Dit model wordt vaak gebruik voor het additief mengen van kleuren. Denk hierbij aan het maken van een kleur op een beeldscherm of TV.
RGB wordt vaak weergegeven als een kubus, met de waarden van R, G en B respectievelijk langs de x-, y- en z-as. Een specifieke kleur is dus een punt op deze kubus. Rood is dan bijvoorbeeld (bij een schaling van de assen van 0 – 1) het punt P (1, 0, 0). Wit is dan het punt (1, 1, 1).
RGB is dan ook een 3D omschrijving van de kleur. Als we stellen dat de invalshoek de kleur niet veranderd, maar de energie verdeling schaalt met een constante factor, dan is de formele omschrijving van de waarden[1]:

$$R(\vec{x}) = m_b(\vec{n},\vec{s})\int e(\vec{x},\lambda)s(\vec{x},\lambda)f_R(\vec{x},\lambda)d\lambda$$
$$G(\vec{x}) = m_b(\vec{n},\vec{s})\int e(\vec{x},\lambda)s(\vec{x},\lambda)f_G(\vec{x},\lambda)d\lambda$$
$$B(\vec{x}) = m_b(\vec{n},\vec{s})\int e(\vec{x},\lambda)s(\vec{x},\lambda)f_B(\vec{x},\lambda)d\lambda$$

Hierbij is m_b een functie die de geometrische afhankelijkheid van het object omschrijft. Dit is een functie van de normaal $\vec{n}$ van het object en de invalshoek $\vec{s}$.
De functie e geeft de kleur (energie verdeling) van de lichtbron als functie van golflengte λ.

2.2 rgb: Genormaliseerd RGB

Het rgb model is een afgeleid model van RGB. Het is in feite een vereenvoudiging die geen rekening houdt met de totale intensiteit van de kleuren. Er wordt gezorgd dat de som van de afzonderlijke kleuren altijd op 1 uitkomt. Dit vereenvoudigt het model en is slechts een 2D omschrijving van de kleur. De waarden voor de r, g en b componenten worden als volgt bepaald op basis van de RGB waarden:

[1] Tetelepta, S., 'Object herkenning op basis van invarianten van kleurendistributies', p. 11, 6-2004

$$r = \frac{R}{R+G+B}$$
$$g = \frac{G}{R+G+B}$$
$$b = \frac{B}{R+G+B}$$

Het model is 2 dimensionaal, omdat de waarde van blauw volledig bepaald wordt door de waarden van rood en groen:

$$r+g+b = \frac{R}{R+G+B}+\frac{G}{R+G+B}+\frac{B}{R+G+B} = \frac{R+G+B}{R+G+B} = 1$$

Hieruit volgt:

$$r+g+b=1$$
$$b=1-(r+g)$$

Bij de veronderstelling gedaan bij het RGB model, dat de invalshoek de kleur niet veranderd, zijn de rgb waarden niet meer afhankelijk van de geometrie van het object[1].

2.3 Gaussisch kleur model

Dit kleurmodel wijkt iets af van de modellen die hierboven staan beschreven. In plaats van de lokale waarden van het spectrum weer te geven, probeert dit model het spectrum lokaal te beschrijven. Dit wordt gedaan in verschillende stappen.
De energie verdeling E(λ) van het licht kan worden omgeschreven naar een Taylor reeks benadering[2]:

$$E(\lambda) = E^{\lambda_0} + \partial\lambda E_{\lambda}^{\lambda_0} + \frac{1}{2}\partial\lambda^3 E_{\lambda\lambda}^{\lambda_0} + \ldots$$

Als we het spectrum meten met een sensor met een gevoeligheid volgens een Gaussische verdeling rond basis golflengte λ_0, dan kunnen wij dit model gebruiken om het model lokaal te omschrijven:

$$\hat{E}(\lambda) = \hat{E}^{\lambda_0,\sigma_\lambda} + \partial\lambda \hat{E}_{\lambda}^{\lambda_0,\sigma_\lambda} + \frac{1}{2}\partial\lambda^3 \hat{E}_{\lambda\lambda}^{\lambda_0,\sigma_\lambda} + \ldots$$

Hierin staat σ_λ voor de spectrale schaal van de Gaussische verdeling $G(\lambda_0, \sigma_\lambda)$ van de sensor. Hier zijn de geschatte waarden een functie van de gemeten waarden en de verdeling G.

[1] Tetelepta, S., 'Object herkenning op basis van invarianten van kleurendistributies', p. 14, 6-2004
[2] Geusebroek, J.M., 'Color and Geometrical Structure in Images', p. 15, 11-2000

$$\hat{E}^{\lambda_0,\sigma_\lambda} = \int E(\lambda) G(\lambda, \lambda_0, \sigma_\lambda) d\lambda$$
$$\hat{E}_{\lambda}^{\lambda_0,\sigma_\lambda} = \int E(\lambda) G_\lambda(\lambda, \lambda_0, \sigma_\lambda) d\lambda$$
$$\hat{E}_{\lambda\lambda}^{\lambda_0,\sigma_\lambda} = \int E(\lambda) G_{\lambda\lambda}(\lambda, \lambda_0, \sigma_\lambda) d\lambda$$

Met deze 3 schattingen is het mogelijk om de menselijke kleuren ruimte te modelleren. Deze waarden zijn ook te halen uit de RGB waarden van een beeld:

$$\begin{bmatrix} \hat{E}^{\lambda_0,\sigma_\lambda} \\ \hat{E}_{\lambda}^{\lambda_0,\sigma_\lambda} \\ \hat{E}_{\lambda\lambda}^{\lambda_0,\sigma_\lambda} \end{bmatrix} = \begin{bmatrix} 0.06 & 0.63 & 0.27 \\ 0.3 & 0.04 & -0.35 \\ 0.34 & -0.6 & 0.17 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

Met dit model is het mogelijk om kleur invarianten te berekenen voor objecten. Een kleur invariant die ik gebruikt heb is N, waarbij de laagste orde invariant in 1D omschreven wordt als:

$$N_{\lambda x} = \frac{E_{\lambda x} E - E_\lambda E_x}{E^2}$$

Hierbij nemen wij aan dat:

- de kleur van het licht op het object niet afhangt van de plaats op het object.
- het object van een mat en dof materiaal is.

2.4 Genormaliseerde kleurendistributies[1]

Genormaliseerde kleurendistributies is net als het rgb model een afgeleid model. Het maakt bij dit model echter niet uit welk model als basis wordt genomen. In tegenstellling tot de voorgaande modellen, gaat dit model uit van een verzameling punten met verschillende kleurwaarden. De verschillende kleurwaarden worden over alle punten verzameld en hier wordt de verdeling van uitgerekend. Dit leidt bijvoorbeeld tot een histogram weergave van het object.
Toepassing van genormaliseerde kleurendistributies zorgt ervoor dat de kleurverdeling van het object robuust wordt tegen veranderingen in de belichting. De verdeling wordt totaal invariant voor een vermenigvuldiging of verschuiving van de losse kleurwaarden (diagonale transformatie). Het model is echter niet invariant voor de situatie waar de kleurwaarden elkaar gaan beïnvloeden (affine transformatie).

Een formele omschrijving van een diagonale transformatie:

$$\begin{bmatrix} R' \\ G' \\ B' \end{bmatrix} = \begin{bmatrix} a_1 & 0 & 0 \\ 0 & a_2 & 0 \\ 0 & 0 & a_3 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

[1] Tetelepta, S., 'Object herkenning op basis van invarianten van kleurendistributies', p. 34, 6-2004

Een formele omschrijving van een affine transformatie:

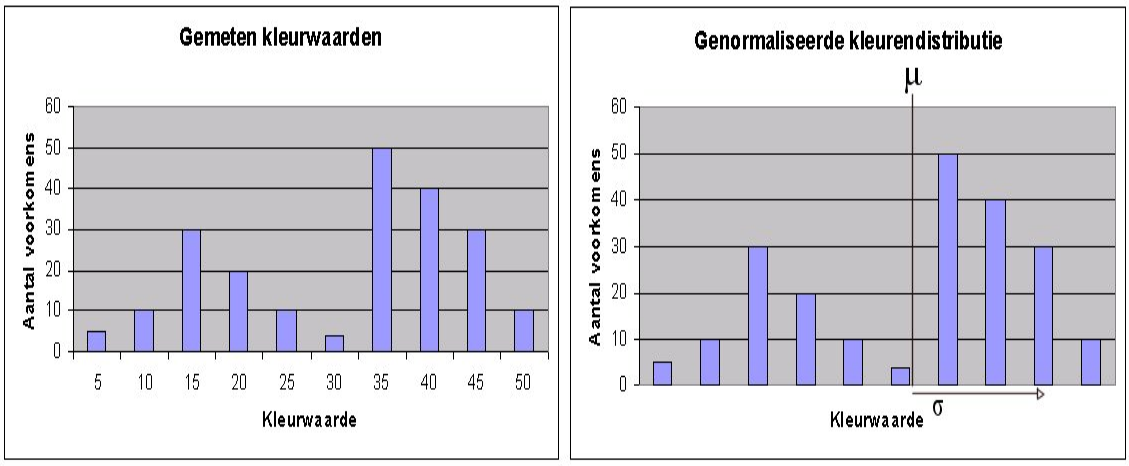
$$\begin{bmatrix} R' \\ G' \\ B' \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

De invariantie voor diagonale transformaties wordt bereikt door alle punten uit het beeld te normaliseren. De normalisatie stap ziet er als volgt uit:

$$\forall_{p \in object, k \in kleruen}\, p(k)' = \frac{p(k) - \mu(k)}{\sigma(k)}$$

Oftewel: Voor ieder punt p in het beeld van het object wordt iedere kleur van ieder punt verminderd met de gemiddelde waarde van die kleur over alle punten (μ) en gedeeld door de standaard afwijking van die kleur over alle punten (σ).

Dit houdt in dat de verdeling van de verschillende kleuren over het beeld nu gecentreerd is rond 0 en dat de eenheidsmaat voor deze distributie nu de standaard afwijking is geworden.



Het figuur hierboven geeft een voorbeeld van een histogram dat wordt genormaliseerd. Links zijn de gelezen kleurwaarden die lopen van 0 tot 50. Rechts zijn de kleurwaarden genormaliseerd en lopen de waarden van -2 tot 1,5.
Een nadeel van de methode is dat de onder- en bovenwaarden van de kleuren niet direct bekend zijn. Dit kan tot gevolg hebben dat details uit het histogram worden weggelaten als de grenzen handmatig worden bepaald.

3 Object herkenning in computer vision

3.1 De huidige methode gebruikt door het UvA team

Tot nu toe maakt het UvA team van een heel eenvoudige methode om objecten te herkennen. Voor de wedstrijd wordt eenmalig de kleur bepaalt van het object dat we willen herkennen, in dit geval de bal. Als deze kleur bekend is, dan wordt het beeld zo bewerkt dat alleen punten met dezelfde kleur over blijven. Omdat de kleuren die gebruikt worden op het veld allemaal vast staan, heeft alleen de bal dezelfde kleur.
Na de eerste stap blijft er altijd ruis over. Deze ruis komt onder andere door de camera zelf en door de afbuiging van het licht op de verschillende objecten. Hierdoor moet er eerst nog gefilterd worden om de ruis weg te werken. Hierna blijft de bal over. Het midden kan bepaald worden door bijvoorbeeld het gemiddelde te nemen van alle overgebleven punten.

Deze methode heeft als voordeel zijn eenvoud en de geringe rekenkracht die nodig is voor de verschillende operaties. Het grote nadeel van de methode is dat deze erg gevoelig is voor het soort licht dat wordt gebruikt. Andere lampen kunnen al tot gevolg hebben dat de kleur van de bal anders is. Dit kan ook gebeuren door een andere invalshoek van het licht.

3.2 Object herkenning op basis van invarianten uit het Gaussisch kleuren model[1]

Er zijn veel invarianten die bepaald kunnen worden uit het Gaussisch kleurenmodel. Deze invarianten zijn allemaal afhankelijk van de lokale kleurstructuur van een beeld. Hieronder zal ik de invariant N afleiden uit het model dat besproken wordt in hoofdstuk 2.3.
Voor matte doffe materialen met lokaal constante belichting (onafhankelijk van x), kunnen we de weerkaatste kleur omschrijven als:

$$E(\lambda,\vec{x}) = e(\lambda)i(\vec{x})m(\lambda,\vec{x}) \qquad (1)$$

Hierin is e de lokaal constante belichting, i de intensiteit van de belichting op een lokatie. Als laatste staat m voor de kleur absorptie coëfficiënt op locatie x.
De afgeleide naar λ van deze formule is:

$$E_\lambda = i(\vec{x})m(\lambda,\vec{x})\frac{\partial}{\partial\lambda}e(\lambda) + e(\lambda)i(\vec{x})\frac{\partial}{\partial\lambda}m(\lambda,\vec{x}) \qquad (2)$$

Als we het resultaat weer delen door vergelijking (1), dan krijgen we:

$$\frac{E_\lambda}{E} = \frac{1}{e(\lambda)}\frac{\partial}{\partial\lambda}e(\lambda) + \frac{1}{m(\lambda,\vec{x})}\frac{\partial}{\partial\lambda}m(\lambda,\vec{x}) \qquad (3)$$

[1] Geusebroek, J.M., 'Color and Geometrical Structure in Images', p. 29-31, 11-2000

Hier valt de intensiteit factor i weg. Als we vervolgens differentiëren naar x, dan is de uitkomst slechts afhankelijk van het materiaal:

$$N_{\lambda x} = \frac{E_{\lambda x}}{E} - \frac{E_{\lambda} E_x}{E^2} \tag{4}$$

Deze maat kan geïnterpreteerd worden als de eerste afgeleide van de kleur, puur afhankelijk van het materiaal in de richting x. Hogere orde afgeleiden zijn ook te berekenen uit deze formule.

Omdat edges in beelden ook worden gedetecteerd door de lokale afgeleiden te berekenen, kunnen we deze theorieën gebruiken. In de praktijk wordt de Canny edge detector[1] toegepast. Volgens de Canny edge detector is iets een edge als geldt:

$$f_{ww} f_w^2 >> 0 \tag{5}$$

Waarin f_w staat voor de afgeleide naar de hoofdrichting van de edge van de continue beeldfunctie f. Een afgeleide van het beeld f in een hoofdrichting (i) van het beeld wordt benaderd door:

$$f_i(i,j) = \sum_{x,y \in beeld} F(x,y) G_i[(x,y) - (i,j)]$$

waarbij F de discrete beeldfunctie is, ofwel het ingelezen beeld en G een normale verdelingsfunctie. De kracht van de edge in richting w is nu gedefinieerd als:

$$f_w = \sqrt{f_x^2 + f_y^2} \tag{6}$$

En de tweede afgeleide waarde in richting w is gedefinieerd als:

$$f_{ww} = \frac{1}{f_{ii}^2 + f_{jj}^2}\left(f_i^2 f_{ii} + f_j^2 f_{jj} + f_i\ f_j f_{ij}\right) \tag{7}$$

Deze formules kunnen we nu gebruiken in combinatie met de kleur invariant N. In plaats van de afgeleide van de beeldfunctie f, nemen wij de afgeleide van de invariant N_λ. Als we dit doen, dan volgt hieruit:

$$\begin{aligned} N_{\lambda w} &= \sqrt{N_{\lambda x}^2 + N_{\lambda y}^2} \\ N_{\lambda ww} &= \frac{1}{N_{\lambda ii}^2 + N_{\lambda jj}^2}\left(N_{\lambda i}^2 N_{\lambda ii} + N_{\lambda j}^2 N_{\lambda jj} + N_{\lambda i} N_{\lambda j} N_{\lambda ij}\right) \end{aligned} \tag{8}$$

Een punt wordt nu herkend als deel van een edge als geldt:

$$N_{\lambda ww} N_{\lambda_w}^2 > drempelwaarde \tag{9}$$

[1] http://carol.science.uva.nl/~rein/mp/files/book-4-localstruct.pdf

Uit de punten die uit formule (9) komen, kan nu de bal worden gedetecteerd. Een methode die kan worden gebruikt is om een cirkel te trekken met minimale kwadratische fout ten opzichte van de overgebleven punten. Alleen als er weinig ruis is en geen andere objecten in het beeld, zal dit leiden tot het gewenste resultaat.
Er zijn betere methoden voor het lokaliseren van de bal uit de edge punten, maar het bleek al snel dat er veel te veel rekentijd nodig was om de punten te bepalen. Om formule (9) voor alle punten te berekenen in Matlab was ik al snel 10 seconden kwijt. Dit is onacceptabel voor realtime toepassingen zoals de AIBO league.

3.3 Histogram gebaseerd EM algoritme[1]

De laatste methode die ik heb bekeken is een soort EM algoritme. Het is gebaseerd op het zogenaamde 'mean-shift tracking' dat veel wordt gebruikt om objecten te kunnen volgen in computer vision. Bij een mean-shift algoritme wordt er gezocht naar een gebied dat het beste overeenkomt met een model van het gewenste object.
De methode die ik heb onderzocht maakt eerst een model van een object. Dit gebeurt door er eerst het object te vangen in een normale verdeling en met deze verdeling een soort gewogen histogram te maken. Punten dichter bij het midden tellen hierbij zwaarder mee, dan punten verder weg. Vervolgens wordt er in het beeld gezocht naar een zo goed mogelijke fit van het object in het beeld.
Dit zoeken gebeurt iteratief, volgens 5 stappen, waarbij we uitgaan van Θ^k, het centrum van de normale verdeling op tijdstip k en V^k, de covariantie matrix van de verdeling op tijdstip k. O_m staat voor de waarde van het histogram model van het gezochte object in bin m.

1: Maak een histogram model voor het object dat nu omvat wordt door de verdeling. Voor ieder punt in het object wordt de bin van de overeenkomstige kleurwaarde opgehoogd met de waarde van de verdelingsfunctie op dat punt.
Geef nu ieder punt x_i een gewicht dat aangeeft hoe goed het past in het model van het gewenste object:

$$w_i = \sum_{bins \in x_i} \sqrt{\frac{o_m}{r_m}}$$

Voor iedere bin die hoort bij de kleur van punt x_i wordt de wortel genomen van de waarde voor die bin van het gezochte object o, gedeeld door de waarde voor die bin van het huidige object r.

2: Vervolgens worden alle q waarden bepaald, die een gewogen gemiddelde gaan bepalen en een nieuwe gewogen covariantie matrix. De q waarden worden berkend als volgt:

$$q_i = \frac{w_i G\left(\vec{x}_i, \Theta^k, V^k\right)}{\sum_i w_i G\left(\vec{x}_i, \Theta^k, V^k\right)}$$

[1] Z. Zivkovic en B. Kröse, An EM-like algorithm for color-histogram-based object tracking, 2004

3: Bepaal uit de q waarden het nieuwe centrum Θ^{k+1}:

$$\Theta^{k+1} = \sum_i q_i \vec{x}_i$$

4: Bepaal de nieuwe covariantie matrix uit de q waarden:

$$V^{k+1} = \beta \sum_i q_i \left(\vec{x}_i - \Theta^k\right)\left(\vec{x}_i - \Theta^k\right)^T$$

De factor β zorgt ervoor dat het algoritme geen bias heeft bij waarde 2. Dit kan er echter ook voor zorgen dat het object veel te snel groeit.

5: Als Θ^k niet meer veranderd, dan is het object gevonden en kan eventueel het volgende beeld worden ingeladen. Ga anders weer terug naar 1.

De modellen van het object worden op de volgende manier gemaakt:

$$r_m = \sum_i G\left(\vec{x}_u, \Theta^k, V^k\right)\left[m \in b(\vec{x}_u)\right]$$

Hierin staat m voor de bin van het histogram model. b(x) kent een punt toe aan 1 of meerdere bins. Bij grijswaarden van 0 tot 1, kan dit bijvoorbeeld zijn: rondaf(waarde(x) * 10).
Dit is een redelijk eenvoudige methode, die toch goede resultaten kan boeken. De methode is niet afhankelijk van de oriëntatie van het gezochte object en er worden geen hoge orde afgeleides bepaald. Ook in Matlab bleek dat in het gunstigste geval een iteratie van 1 tot en met 5 in ongeveer 0,1 seconde werd uitgevoerd.

3.4 Combinatie EM met genormaliseerde kleurendistributies

Het algoritme besproken in hoofdstuk 3.3 gebruikt histogram waarden om objecten te vergelijken. De genormaliseerde kleurendistributies is een omschrijving die kleur invariante histogrammen kan maken. Ik heb daarom besloten om deze te combineren tot een enkel algoritme.
In pseudo code ziet dit er als volgt uit:

```
function EMtrack:
    o = laad( objectmodel );
    beeld = laad_nieuw_beeld;
    theta = midden( beeld );
    // pak slecht maxPoints punten uit het object
    maxPoints = 500;

    while( theta niet veranderd )
        [theta, V] = iterate( theta, V, beeld, maxPoints );
    end while
end function
```

```
function [theta, V] = iterate( theta, V, beeld, maxPoints )
    // zie hoofdstuk 2.4 voor de normalisatie stap
    beeld = normalize( beeld );

    // zie hoofdstuk 3.3 voor de te volgen stappen
    [theta, V] = iterate_zie_3.3
end function

function beeld = normalize( beeld );
    mr = gemiddeld_rood(beeld)
    sr = deviatie_rood(beeld)
    mg = gemiddeld_ groen (beeld)
    sg = deviatie_groen(beeld)
    mg = gemiddeld_ blauw (beeld)
    sg = deviatie_blauw(beeld)
    for all point in beeld
        rood( point ) = ( rood( point ) – mr ) / sr
        groen( point ) = (groen( point ) – mg ) / sg
        blauw( point ) = (blauw( point ) – mb ) / sb
    end for
end function
```

Deze methode kostte door de normalisatie stap wel iets meer tijd, maar het algoritme deed er nu even lang over, ook na het schalen van de kleurwaarden. Dit is een algoritme dat zeker gebruikt kan worden voor de Robocup Legged League.
Wat wel bleek is dat het algoritme nu gevoeliger is voor de schaal factor β, genoemd in stap 5 van hoofdstuk 3.3. Bij een te hoge waarde omvat het object al snel het hele beeld en is het gedrag niet meer goed te voorspellen.

4 Resultaten en conclusies

4.1 Experimenten

Als experiment heb ik geprobeerd om een rood object te vinden tegen een witte achtergrond. Om verschillende belichtingen te testen heb ik de foto's gemaakt vanuit verschillende hoeken, zowel met als zonder flitser.
Met het Matlab programma 'CIedge.m' wordt geprobeerd de bal uit de foto te filteren met de kleurconstanten uit het Gaussisch kleurmodel. Dit wordt besproken in hoofdstuk 2.3 en 3.2. Het geschatte middelpunt wordt vervolgens aangegeven met een stip.
Het programma 'EMtrack.m' vindt de bal op de manieren besproken in hoofdstuk 3.3 en 3.4. Het EM algoritme probeert iteratief de beste overeenkomst te vinden van het gezochte object. In het programma kan het aantal samples van het object worden gekozen en de methode (rgb of NCD, genormaliseerde kleurendistributies (hoofdstuk 2.4)). Als eerste stap wordt een model ingeladen van het object. Dit object wordt gemaakt met de programma's 'emModelNCD.m' voor een NCD model en 'emModel.m' voor een rgb model. Het model wordt gemaakt van een foto van alleen het gewenste object.
Als extra experiment wordt hier gekeken naar de gevoeligheid van veranderingen in kleur. De ingelezen kleurwaarden wordt per kleurgroep met een willekeurige factor geschaald, alvorens het programma wordt uitgevoerd. Het ingelezen model blijft hetzelfde.

4.2 Gebruikte beelden

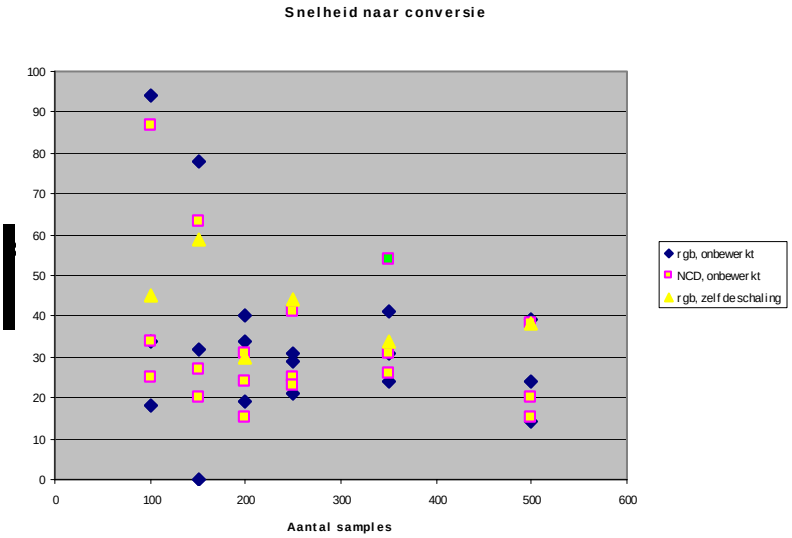
Hieronder staan de gebruikte beelden voor de experimenten. De eerste 3 zijn onbewerkte beelden:



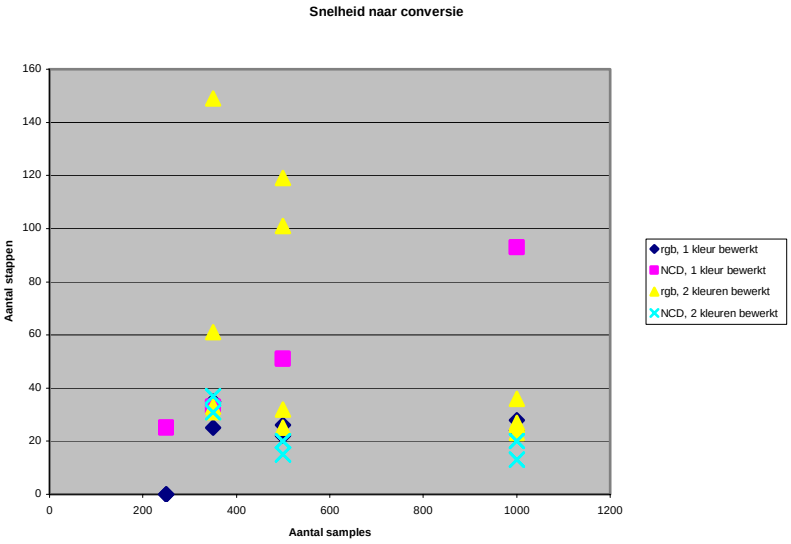
Hieronder is een voorbeeld van een foto waarbij de kleuren zijn bewerkt:

4.3 Resultaten

In dit hoofdstuk bespreek ik de resultaten van het EM algoritme met gebruik van een rgb histogram en genormaliseerde kleurendistributies.
Hieronder staan het aantal stappen tot conversie uitgezet voor de 2 algoritmen op 3 onbewerkte beelden. 2 beelden zijn genomen met flits, het andere beeld zonder.



Op de x-as zijn het aantal samples uitgezet dat per stap werd genomen van het nieuwe object. Er is te zien dat de procedure met genormaliseerde kleurendistributies iets constanter is in het aantal stappen dat het nodig heeft om het object te vinden. Voor rgb zijn er 2 series gemaakt. Dit komt omdat rgb kon werken met een grotere schaal factor en daarmee sneller het object vond.
Hieronder zien we echter het grote voordeel van de nieuwe methode. Hier zijn de beelden bewerkt. 1 of meerdere kleuren is met een random waarde geschaald. De rgb serie werkt nu met een grotere schaal factor en heeft hierdoor een snelheids voordeel.
Wat niet in de grafiek te zien is, is dat rgb niets meer vond als er weinig samples van het object werden genomen. Als het rgb algoritme wel iets vond, dan was dit altijd na meer stappen dan bij het onbewerkte plaatje. Als we 3 kleuren veranderden dan was rgb niet meer bruikbaar. De genormaliseerde kleurendistributies bleef echter constant, onafhankelijk van de verandering in kleur, zoals we uit de theorie verwachtten. Ook de rekentijd viel mee, gemiddeld 4,5 seconden om het object te vinden. Dit kan sneller buiten Matlab en er kan met minder samples gewerkt worden.
Dit alles is een mooi resultaat en verdiend dus ook een aanbeveling voor de Aibo League.



Hieronder geef ik een voorbeeld van het resultaat van het algoritme. De stip geeft het gevonden centrum aan.

5 Programma code

5.1 CIedge.m

```
% color independence by Coen Pieterse,
% st. nr. 0155071

% close all images still open
close all;

% set edge filter standard deviation to .7 ( gaussian filter )
sigma = .35;

% initialize cell array to store all images
images = cell( 0,1 );
edgeImages = cell( 0,1 );

% set number of bins
bins = 10;

% set number of groups for EM
groups = 3;

% read images as doubles
moreImages = 1;
imageNr = 34;
while moreImages == 1
   if imageNr < 20
      %moreImages = 0;
      iamgeNr = imageNr + 1;
      continue
   end
   if imageNr > 36
      moreImages = 0;
   end

   if imageNr < 10
      name = strcat( 'Image0', num2str( imageNr ), '.jpg' );
   else
      name = strcat( 'Image', num2str( imageNr ), '.jpg' );
   end % if imageNr < 10
   name
   try
      % load an rgb image and convert to a gaussian image
      image = loadGaussImage( name, 1 );

      % create illumination independent edge image from original image
      %edges = gaussEdgeDetect( image, sigma );
      % add image and edge image
      images( end + 1, 1 ) = { image };
      disp( 'Image loaded' );
      %edgeImages( end + 1, 1 ) = { edges };
      %disp( 'edges created' );
      % temporary
      %figure( imageNr );
      %subplot( 2,1,1 );
```

```
        %imshow( image );
        %subplot( 2,1,2 );
        %imshow( edges );
    catch
        imageNr
        moreImages = 0;
    end % try catch
    imageNr = imageNr + 1;
end % while
% clear loop variables
clear image edges moreImages imageNr;

close all;

for i = 1:size( images, 1 )
    figure( i );
    gi = images{i};
    [ edges, edgepoints, directions, strengths ] = gaussEdgeDetect( gi, 1.5 );
    [m,r] = detectBall( edges, edgepoints, directions, strengths )
end
```

5.2 GaussEdgeDetect

```
% detect edges with Canny edge detector on illumination invariant edge
% detector
% input gaussImage, an image converted to gaussian image values
% sigma, the deviation of the gaussian filter to detect the edges
% out
% edges, the edge image derived from the input
% edgepoints, the points in the image where an edge is detected
% directions, the normalized line through of the edges
% strengths, the edge strength
% function [ edges, edgepoints, directions, strengths ] = gaussEdgeDetect( gaussImage, sigma )

function [edges, edgepoints, directions, strengths ] = gaussEdgeDetect( gaussImage, sigma )

% store all edge points
edgepoints = zeros( 0, 2 );
% store their directions (not normalized)
directions = zeros( 0, 3 );
% store the edge strengths
strengths = zeros( 0, 1 );

% set maximum value of an edge
maximum = 10;
% minimum value
minimum = 0.1;


% check if sigma is passed and correct
try
    if sigma <= 0
        sigma = 1;
    end
catch
    sigma = 1;
end
```

```
filterSize = round(2 * 3 * sigma );
% construct filters
gaussF = zeros( filterSize, filterSize );
dXgaussF = zeros( size( gaussF ) );
dYgaussF = zeros( size( gaussF ) );
dXXgaussF = zeros( size( gaussF ) );
dYYgaussF = zeros( size( gaussF ) );
% construc edges matrix
edges = zeros( size( gaussImage, 1 ), size( gaussImage, 2 ) );

middle = ( size( gaussF, 1 ) + 1 ) * 0.5;

% fill filter values
for i = 1:size( gaussF, 1 )
   for j = 1:size( gaussF, 2 )
      x = i - middle;
      y = j - middle;
      gaussF( i, j ) = gauss( [ x, y ], [ 0, 0 ], [ sigma * sigma, 0 ; 0, sigma * sigma ] );
      dXgaussF( i, j ) = - x * gaussF( i, j ) / (sigma^2);
      dXXgaussF( i, j ) = gaussF( i, j ) * ( x^2 - sigma^2 ) / (sigma^4 );
   end % for j
end % for i

dYgaussF = dXgaussF';
dYYgaussF = dXXgaussF';


% calculate edge strengths as
% strength = Nlw^2 * Nlww
% for all points
%dIm = imfilter( gaussImage, gaussF );
dIm = gaussImage;
dxIm = imfilter( dIm, dXgaussF );
dyIm = imfilter( dIm, dYgaussF );
dxxIm = imfilter( dIm, dXXgaussF );
dyyIm = imfilter( dIm, dXXgaussF );
dxyIm = imfilter( dxIm, dYgaussF );

% edge image
edges = zeros( size( gaussImage, 1 ), size( gaussImage, 2 ) );


for i = 1:size( gaussImage, 1 )
   for j = 1:size( gaussImage, 2 )
      % original image
      E( 1 ) = gaussImage( i, j, 1 );
      E( 2 ) = gaussImage( i, j, 2 );
      E( 3 ) = gaussImage( i, j, 3 );
      % deriviate to x
      Ex( 1 ) = dxIm( i, j, 1 );
      Ex( 2 ) = dxIm( i, j, 2 );
      Ex( 3 ) = dxIm( i, j, 3 );
      % deriviate to y
      Ey( 1 ) = dyIm( i, j, 1 );
      Ey( 2 ) = dyIm( i, j, 2 );
      Ey( 3 ) = dyIm( i, j, 3 );
      % twice to x
      Exx( 1 ) = dxxIm( i, j, 1 );
      Exx( 2 ) = dxxIm( i, j, 2 );
      Exx( 3 ) = dxxIm( i, j, 3 );
```

```
      % twice to y
      Eyy( 1 ) = dyyIm( i, j, 1 );
      Eyy( 2 ) = dyyIm( i, j, 2 );
      Eyy( 3 ) = dyyIm( i, j, 3 );
      % to x, to y
      Exy( 1 ) = dxyIm( i, j, 1 );
      Exy( 2 ) = dxyIm( i, j, 2 );
      Exy( 3 ) = dxyIm( i, j, 3 );


      % Nlx = ( Elx * E - ElEx ) / E^2
      Nlx = ( Ex(2) * E(1) - E(2) * Ex(1) ) / E(1)^2 ;
      Nly = ( Ey(2) * E(1) - E(2) * Ey(1) ) / E(1)^2 ;
      % Nlxx = Elxx / E - ( 2 * ExElx + ExxEl ) / E^2 + 2 * ElEx^2 / E^3
      Nlxx = Exx(2) / E(1) - ( 2 * Ex( 1 ) * Ex( 2 ) + Exx(1) * E(2) ) / (E(1)^2) + 2 * E(2) * ( Ex(1)^2 ) / (E(1)^3);
      Nlyy = Eyy(2) / E(1) - ( 2 * Ey( 1 ) * Ey( 2 ) + Eyy(1) * E(2) )/ (E(1)^2) + 2 * E(2) * ( Ey(1)^2 ) / (E(1)^3);

      % Nlxy = Elxy / E - EyElx/E^2 -Ely*Ex/E^2 - ElExy / E^2 + 2*ElExEy/E^3
      Nlxy = Exy(2)/E(1) - ( Ey(1)*Ex(2) + Ey(2)*Ex(1) + E(2)*Exy(1) )/(E(1)^2) +
2*E(2)*Ey(1)*Ex(1)/(E(1)^3);

      Nlw = sqrt( Nlx^2 + Nly^2 );
      if ( Nlxx^2 + Nlyy^2 ) > 0
         Nlww = ( Nlx^2 * Nlxx + 2 * Nlx * Nly * Nlxy + Nly^2 * Nlyy ) / ( Nlxx^2 + Nlyy^2 );
         % calculate edge strength
         edges( i, j ) = Nlw^2 * Nlww;
         if edges( i, j ) > minimum
            % set edge strength to maximum

            edges( i, j ) = 256;

         else
            edges(i,j) = 0;
         end
      else % check for division by zero
         Nlww = 0;
      end


   end % for j
end % for i

% dilate and erode image
dilateElement = [ 1 1 1 1 1 ...
            ; 1 1 1 1 1 ...
            ; 1 1 1 1 1 ...
            ; 1 1 1 1 1 ...
            ; 1 1 1 1 1];

% filter image
edges = imerode( imdilate( edges, dilateElement ), dilateElement );

for i = 1:size( gaussImage, 1 )
   for j = 1:size( gaussImage, 2 )
      if edges( i,j) > 0
         % store this edge
         edgepoints( end + 1, : ) = [ i, j ];
         line = [ Nlx, -Nly, - i * Nlx + j * Nly ];
         directions( end + 1, : ) = line / norm( line );
         %strengths( end + 1, 1 ) = edges( i, j );
```

```
        strengths( end + 1, 1 ) = 256;
      end % if edge > 0
   end % for j
end % for i

size(edgepoints)

% normalize edges / mean = edges * n / mean
%edges = edges / (mean( strengths ));
%strengths = strengths / (mean( strengths ));

%edges = imfilter( edges, cnct4 );

imshow( edges )
```

5.3 DetectBall

```
% this function will try to detect the ball from the collection of edges
% input:
% edges, an edge image
% edgespoints: points detected in an image which are part of an edge
% directions : direction of the edge of which the points are part
% strengths  : the relative strength of the edge
% thresshold, the thresshold value to consider a point as part of an edge.
function [ m, r ] = detectBall( edges, edgepoints, directions, strengths, threshold )


% get number of edgepoints
nPoints = size( edgepoints, 1 )

% store middlepoints
m = zeros( 0, 2 );
% store radius
r = zeros( 0, 1 );

% formulate matrix A and b for equation Ax = b
A = zeros( 0, 3 );
b = zeros( 0, 1 );

% for all combinations of three points
for i = 1:nPoints
   A( end+1, : ) = [ 2 * edgepoints( i, : ), 1 ];
   b( end+1, 1 ) = edgepoints( i, : ) * edgepoints( i, : )';
end % for i

% use \ instead of inv(A) * b
% to solve the equation
C = A \ b

m = [ C( 1 ) C( 2 ) ];
r = sqrt( C(3) + C(2)^2 + C(1)^2 ) ;

dot = ones( 7, 7 );
edges( m(1) - 3:m(1) + 3, m(2) - 3:m(2) + 3 ) = dot;

imshow( edges )
```

5.4 EMtrack

```
% use algorithm presented by Zoran Zivkovic and Ben Krose  to track the
% ball in the game
% using the naming conventions from the paper

imageStr = 'Image35.jpg';
%imageStr = 'LeggedLeague.jpg';
%imageStr = 'AIBOS_wide.jpg';

% include gaussian function
addpath 'scripts_and_data' '-begin'

% set maximum points to process in one run
maxPoints = 500;

% set margin for theta ( center )
margin = 1;

% set minimal color value for normalized color distibution
minColor = -10.0;
% set maximal color value for normalized color distibution
maxColor = 10.0;


% import object model ( histogram ) of the ball
load ballmodel;
%load compmodelrgb;
load ballmodelNCD;
% load ball model from AIBO site
%load compmodelNCD;

%object = 'Object.jpg';
%object = 'CompBall.jpg';

% or create the image and save it (rgb)
%o = emModel( im2double( imread( object ) ), 5000, 40 );

% construct object with normalized color distributions
%oNCD = emModelNCD( im2double( imread( object ) ), 5000, 30, minColor, maxColor );

%save ballmodel o;
%save compmodelrgb o;
%save compmodelNCD oNCD;
%save ballmodelNCD oNCD;

%return

% for each frame load the image
image = im2double( imread( imageStr ) );

% blur image:
factor = .5 * rand(1)
image( :,:,1 ) = ( image( :,:,1 ) +  factor ) / ( 1 + factor );
factor = .8 * rand(1)
image( :,:,2 ) = ( image( :,:,2 ) +  factor ) / ( 1 + factor );
factor = .8 * rand(1)
image( :,:,3 ) = ( image( :,:,3 ) +  factor ) / ( 1 + factor );
```

```
clear factor

% set center to half the image
thetaN = size( image ) / 2; thetaN = thetaN( 1, 1:2 );
% set direction V to ball with radius 20
VN = [ 400 0;0 400 ];
% set different values for V and theta
theta = 0;
V = 0;

% count same position
steadyCount = 0;
%%%%%%%%%%%%

% count number of steps
steps = 0;
% count total elapsed time
totaltime = 0;

% iterate until stable center
while steadyCount < 5
   % copy values from previous iteratino
   V = VN; theta = thetaN;
   % store time before iteration
   tic;
   % iterate

   %[thetaN, VN ] = EMiterateNCD( oNCD, theta, V, image, maxPoints, minColor, maxColor );
   [thetaN, VN ] = EMiterate( o, theta, V, image, maxPoints );
   % show time of iteration step
   totaltime = totaltime + toc;

   imageAcc = image;
   % show white dot
   dotCenter = round( theta );
   imageAcc( dotCenter(1) - 3:dotCenter(1) + 3, dotCenter(2) - 3:dotCenter(2) + 3 , : ) = ones( 7,7,3 );
   %hold off;
   imshow( imageAcc );
   % increase number of steps
   steps = steps + 1;
   if mod( steps, 20 ) == 0
      totaltime
   end
   % check for same position ( 1 pixel margin )
   if ( theta(1) >= thetaN(1) - margin && theta(1) <= thetaN(1) + margin ) ...
         && ( theta(2) >= thetaN(2) - margin && theta(2) <= thetaN(2) + margin )
      steadyCount = steadyCount + 1;
   else
      steadyCount = 0;
   end
end
% display statistics
disp( '--------------------' );
% make pixel values
theta = round( thetaN )
imageStr
maxPoints
steps
totaltime
disp( '--------------------' );
```

5.5 emModel

```
% create a model for the EM like algorithm proposed by
% Zoran Zivkovic and Ben Krose  to track the
%
% model = emModel( region, bins )
% model , the output histogram, 1 x 3*bins vector
% region, the region to include in the histogram
% points, the number of points to use
% bins , the number of bins per color
function model = emModel( image, points, bins )

% initialize model
model = zeros( 1, 3 * bins );

% initialize center of the image
theta = size( image ) / 2;
% ignore color dimension
theta = theta( 1, 1:2 );


% set Covariance matrix for the shape of the ball ( circular of course )
V = [ (.5*size( image, 1 ))^2 0 ...
   ; 0 (.5*size( image, 2 ))^2 ];

% set maximum points in x and y directin
maxI = sqrt( points ) * V(1,1) / V(2,2);
maxJ = sqrt( points ) * V(2,2) / V(1,1);

% naive implementation
% start and end of convolution
% don't go outside the image
startI = max( theta(1) - round( 2.5 * sqrt(V(1,1)) ), 1 )
endI = min( theta(1) + round( 2.5 * sqrt(V(1,1)) ), size( image, 1 ) )

startJ = max( theta(2) - round( 2.5 * sqrt(V(2,2)) ), 1 )
endJ = min( theta(2) + round( 2.5 * sqrt(V(2,2)) ), size( image, 2 ) )

for i = startI:ceil( (endI - startI)/maxI ):endI
   for j = startJ:ceil( (endJ - startJ)/maxJ ):endJ
      % color values range from 0.0 to 1.0 so..
      redBin = 1 + round( image( i, j, 1 ) * (bins-1) );
      greenBin = 1 + round( image( i, j, 2 ) * (bins-1) );
      blueBin = 1 + round( image( i, j, 3 ) * (bins-1) );
      % calculate value of gaussian function
      % this could also be done outside the loop to profit from
      % symetry in the gaussian function
      G = gauss( [i, j], theta, V );
      % add to the histogram model
      model( redBin ) = model( redBin ) + G;
      model( bins + greenBin ) = model( bins + greenBin ) + G;
      model( 2 * bins + blueBin ) = model( 2 * bins + blueBin ) + G;
   end % for j
end % for i
```

5.6 emModelNCD

```
% create a model for the EM like algorithm proposed by
% Zoran Zivkovic and Ben Krose  to track the
% augmented with normalized color distributions proposed by
% Salmon Tetelepta
%
% model = emModel( region, bins )
% model , the output histogram, 1 x 3*bins vector
% region, the region to include in the histogram
% points, the number of points to use
% bins , the number of bins per color
% minColor, the minimum value of the color in normalized space
% maxColor, the maximum value of the color in normalized space
% function model = emModel( image, points, bins, minColor, maxColor )
function model = emModel( image, points, bins, minColor, maxColor )

% initialize model
model = zeros( 1, 3 * bins );

% initialize center of the image
theta = size( image ) / 2;
% ignore color dimension
theta = theta( 1, 1:2 );



% set Covariance matrix for the shape of the ball ( circular of course )
V = [ (.5*size( image, 1 ))^2 0 ...
   ; 0 (.5*size( image, 2 ))^2 ];

% set maximum points in x and y directin
maxI = sqrt( points ) * V(1,1) / V(2,2);
maxJ = sqrt( points ) * V(2,2) / V(1,1);

% naive implementation
% start and end of convolution
% don't go outside the image
startI = max( theta(1) - round( 2.5 * sqrt(V(1,1)) ), 1 )
endI = min( theta(1) + round( 2.5 * sqrt(V(1,1)) ), size( image, 1 ) )

startJ = max( theta(2) - round( 2.5 * sqrt(V(2,2)) ), 1 )
endJ = min( theta(2) + round( 2.5 * sqrt(V(2,2)) ), size( image, 2 ) )


% cut out section of the image to process
section = zeros( size( startI:ceil( (endI - startI)/maxI ):endI, 2 ) ...
   , size( startJ:ceil( (endJ - startJ)/maxJ ):endJ, 2 ), 3 );
position = [1,1];
for i = startI:ceil( (endI - startI)/maxI ):endI
   for j = startJ:ceil( (endJ - startJ)/maxJ ):endJ
      section( position(1), position(2), : ) = image( i, j, : );
      position( 2 ) = position( 2 ) + 1;
   end % for j
   position = [ position( 1 ) + 1, 1 ];
end % for i

% use normalized color distributions, range from 0.0 to 1.0
```

```
section = ( normalize( section ) - minColor ) / ( maxColor - minColor );

% reset the position
position = [1,1];

% for all points to process
for i = startI:ceil( (endI - startI)/maxI ):endI
   for j = startJ:ceil( (endJ - startJ)/maxJ ):endJ
      x = position( 1 ); y = position( 2 );
      redBin = 1 + round( max(min( section( x, y, 1 ), 1), 0) * (bins-1) );
      greenBin = 1 + round( max(min( section( x, y, 2 ), 1), 0) * (bins-1) );
      blueBin = 1 + round( max(min( section( x, y, 3 ), 1), 0) * (bins-1) );
      % calculate value of gaussian function
      % this could also be done outside the loop to profit from
      % symetry in the gaussian function
      G = gauss( [i, j], theta, V );
      % add to the histogram model
      model( redBin ) = model( redBin ) + G;
      model( bins + greenBin ) = model( bins + greenBin ) + G;
      model( 2 * bins + blueBin ) = model( 2 * bins + blueBin ) + G;
      position( 2 ) = position( 2 ) + 1;
   end % for j
   position = [ position( 1 ) + 1, 1 ];
end % for i
```

5.7 normalize

```
% normalize an image color histogram
function normalized = normalize( image )
% store mean temporary to correct for 3D image properties
tempMu = mean( mean( image, 1 ), 2 );
% mean color values
mu = zeros( 3,1 );
% convert 3D image values
mu( 1, 1 ) = tempMu( :,:,1 );
mu( 2, 1 ) = tempMu( :,:,2 );
mu( 3, 1 ) = tempMu( :,:,3 );

% clear temporary values
clear tempMu;

% store standard deviance
sigma = zeros( 3, 1 );

% calculate deviance for every color
sigma( 1,1 ) = max( std( std( image( :,:,1 ), 1 ) ), 0.00000001 );
sigma( 2,1 ) = max( std( std( image( :,:,2 ), 1 ) ), 0.00000001 );
sigma( 3,1 ) = max( std( std( image( :,:,3 ), 1 ) ), 0.00000001 );

% initialize normalized image
normalized = zeros( size( image ) );

% for all colors normalize the values
normalized( :, :, 1 ) = ( image( : , :, 1 ) - mu( 1, 1 ) ) / sigma( 1, 1 );
normalized( :, :, 2 ) = ( image( : , :, 2 ) - mu( 2, 1 ) ) / sigma( 2, 1 );
normalized( :, :, 3 ) = ( image( : , :, 3 ) - mu( 3, 1 ) ) / sigma( 3, 1 );
```

5.8 EMiterate

```
% use algorithm presented by Zoran Zivkovic and Ben Krose  to track the
% ball in the game
% using the naming conventions from the paper
% iterate to timestep k + 1
% input:
% o, the object model
%   theta, the current mean of the gaussian distriution ( position in image )
%   V, orientation of the distribution ( covariance matrix )
%   image, the image in which to search for the object
%   maxPoints, max. nr. of points to use to iterate
% output:
%   thetaN, mean on next timestep
%   VB, orientation on next timestep
function [thetaN, VN ] = EMiterate( o, theta, V, image, maxPoints )

% normalization factor
beta = 1.1;

% bins per color
bins = size( o, 2 ) / 3;

% calculate current histogram model
% ignore blue color ( depends on red and green )
r = zeros( 1, 2 * bins );


% set maximum points in x and y directin
maxI = sqrt( maxPoints ) * V(1,1) / V(2,2);
maxJ = sqrt( maxPoints ) * V(2,2) / V(1,1);

% naive implementation
% start and end of convolution
% don't go outside the image
startI = max( floor( theta(1) - 2.5 * sqrt(V(1,1)) ), 1 );
endI = min( floor( theta(1) + 2.5 * sqrt(V(1,1)) ), size( image, 1 ) );

startJ = max( floor( theta(2) - 2.5 * sqrt(V(2,2)) ), 1 );
endJ = min( floor( theta(2) + 2.5 * sqrt(V(2,2)) ), size( image, 2 ) );

% initialize gaussian value for every point
G = zeros( 0, 1 );
% store every point
points = zeros( 0, 2 );
% store bins for every point
pointColor = zeros( 0, 2 );

for i = startI:ceil( (endI - startI)/maxI ):endI
   for j = startJ:ceil( (endJ - startJ)/maxJ ):endJ

      % normalize to rgb values (ignore blue)
      image( i, j, : ) = image( i, j, : ) / sum( image( i, j, : ) );
      %image( i, j, 1:2 ) = image( i, j, 1:2 ) / sum( image( i, j, : ) );

      % color values range from 0.0 to 1.0 so.. for rgb
      redBin = 1 + round( image( i, j, 1 ) * (bins-1) );
      greenBin = 1 + round( image( i, j, 2 ) * (bins-1) );
      %blueBin = 1 + round( image( i, j, 3 ) * (bins-1) );
```

```
        % calculate value of gaussian function
        % this could also be done outside the loop to profit from
        % symetry in the gaussian function
        G( end + 1, 1 ) = gauss( [i, j], theta, V );
        % add to the histogram model
        r( redBin ) = r( redBin ) + G(end);
        r( bins + greenBin ) = r( bins + greenBin ) + G(end);
        %r( 2 * bins + blueBin ) = r( 2 * bins + blueBin ) + G(end);
        % store histogram positions

        %pointColor( end + 1, : ) = [ redBin, bins + greenBin, 2 * bins + blueBin ];
        pointColor( end + 1, : ) = [ redBin, bins + greenBin ];
        % store point
        points( end + 1, : ) = [i,j];
    end % for j
end % for i

% initialize weights for every point
weights = zeros( size( points, 1 ), 1 );

% take square roots of o and r
sqrto = sqrt( o(1, 1:2*bins) );
sqrtr = sqrt( r );
% for all points calculate the weight
for i = 1:size( points, 1 )
    weights( i, 1 ) = sqrto( pointColor( i, 1 ) ) / sqrtr( pointColor( i, 1 ) ) ...
                + sqrto( pointColor( i, 2 ) ) / sqrtr( pointColor( i, 2 ) ) ; %...
                %+ sqrto( pointColor( i, 3 ) ) / sqrtr( pointColor( i, 3 ) ) ;
end % for all points

% take product of gaussian and weights ( division in (7) )
q = weights .* G;

% sum this equation
sumProduct = sum( q );

% calculate q values from eq. (7)
q = q / sumProduct;

% 2 x n * n x 2
VN = beta * ( [q, q] .* ( [points(:,1) - theta(1),points(:,2) - theta(2)] ) )' * ( [points(:,1) - theta(1),points(:,2) -
theta(2)] );

maximum = max( max( VN, [], 1 ), [], 2 );
maxSize = ( min( size( image, 2 ), size( image, 1 )  ) / 6 ) ^ 2;
if maximum > maxSize
    VN = VN * maxSize / maximum;
end
% for all points
thetaN = (q' * points  );
```