ROBO-BITS-57

Jaargang15, nummer 2, juni 2012

Afz.hcc Robotica gg, p.a. Henk de Gans,Koelmanhof 2 3861GG Nijkerk..

hcc robotica

Robobits is een uitgave van de hcc!robotica interesse groep, en wordt vier keer per jaar als PDF beschikbaar gesteld aan de leden. hcc! robotica is een onderdeel van de hcc! (hobby computer club), een vereniging van bijna 90.000 leden.

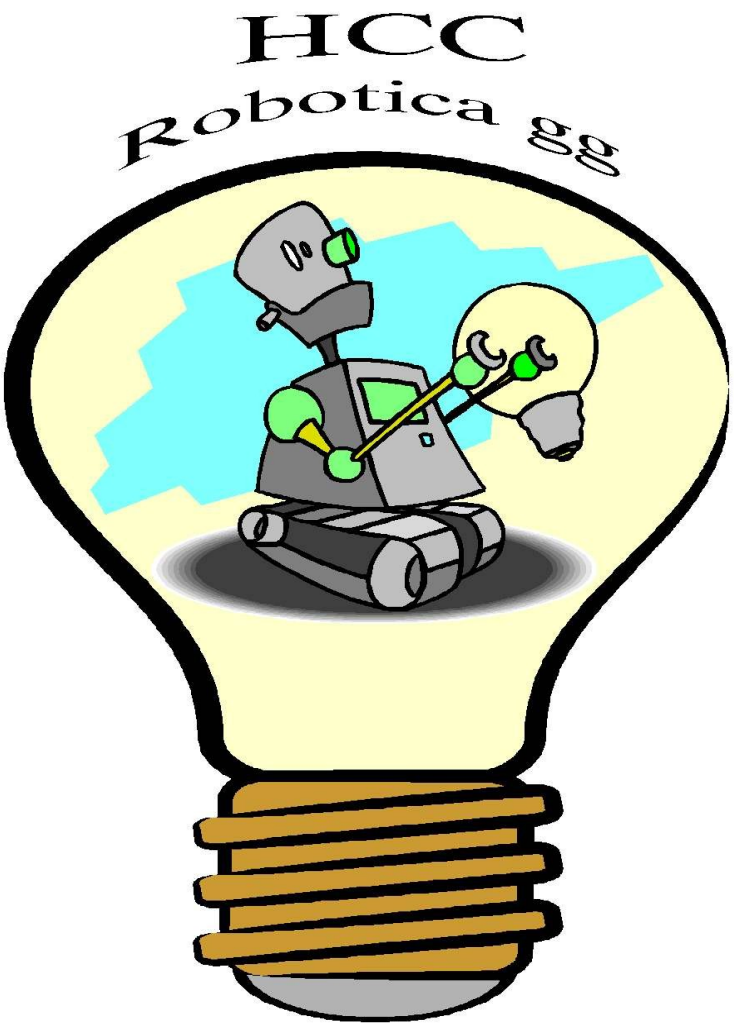
==========

Redactie adres: H.J. de Gans, Koelmanhof 2, 3816GG Nijkerk.
hj.de.gans@gmail.com Tekst aanleveren in WORD of platte tekst in ASCII.
Afbeeldingen los er bij in JPG, GIF of BMP formaat.

==========

Dagelijks bestuur:

Voorzitter:	E.F.O.Buzzi(Ed), Ed.Buzzi@net.hcc.nl
Technisch adviseur:	Z.Otten(Zeno), z.otten@chello.nl
Technisch adviseur:	H.M.P. van Sint Annaland (Hinnie) h.vansintannaland@xs4all.nl
Secretaris:	M.W.J. van Harmelen (Rien) r.van.harmelen@hetnet.nl
Penningmeester:	H.J. de Gans(Henk) hj.de.gans@gmail.com



inhouds opgave:

REDACTIE

Geachte lezer,

Voor u ligt de 57 ste Robobits! Een beetje verlaat, zoals u wellicht gemerkt hebt. Maar er was GEEN ENKELE kopij! ZONDER OOK UW INBRENG, GEEN ROBOBITS!!!! Denk niet te snel dat wat u schrijft of waar u mee bezig bent, dat dit voor niemand interessant is! Stuur ook eens wat in! Ook UW inbreng is van harte welkom!

Ik ben naar de EK robotvoetbal in Eindhoven geweest, dit was zeer de moeite waard! Nu ONS team van Techunited de WK gewonnen heeft, dacht ik ineens, laten we dan in ieder geval een van hun robots op het kaft zetten, en het laatste stuk van Iwan Tolboom over robot voetbal publiceren in de robobits.

Henk de Gans
redactie

==========

Beeld herkenning met MSRDS

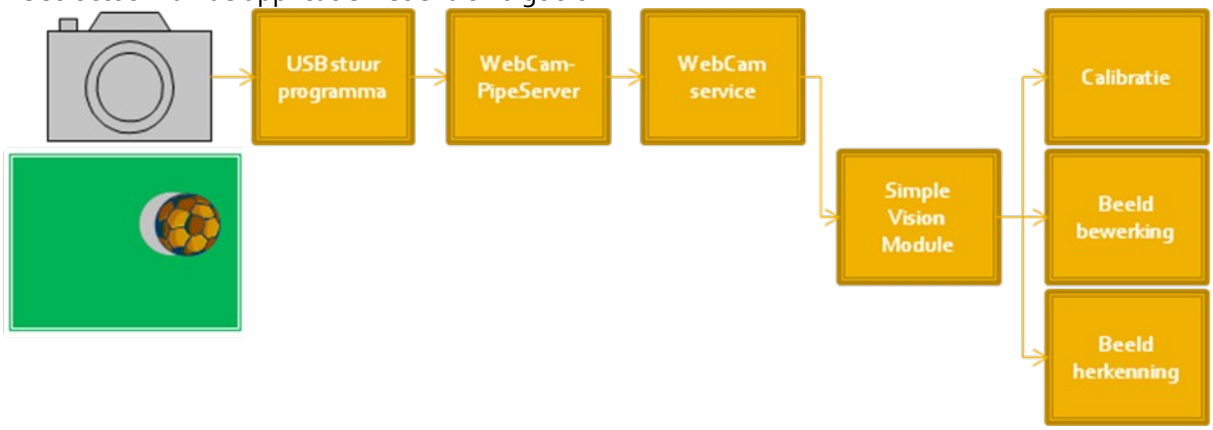
Voor het robotvoetbal hangt een een camera boven het veld.
Daarvoor is naast de HCCRefereeModule welke dient voor bijhouden van de tijd, scores en overtredingen, een HCCSimpleVisionModule geschreven met de volgende functies:

1. Vastleggen van de afmetingen van het veld en posities van de doelpalen
2. Instellen van de kleuren van bal, teamkleuren(geel en blauw) en labelkleuren(groen en roze)
3. Bepalen van de positie van de bal en positie van de spelers van ieder team

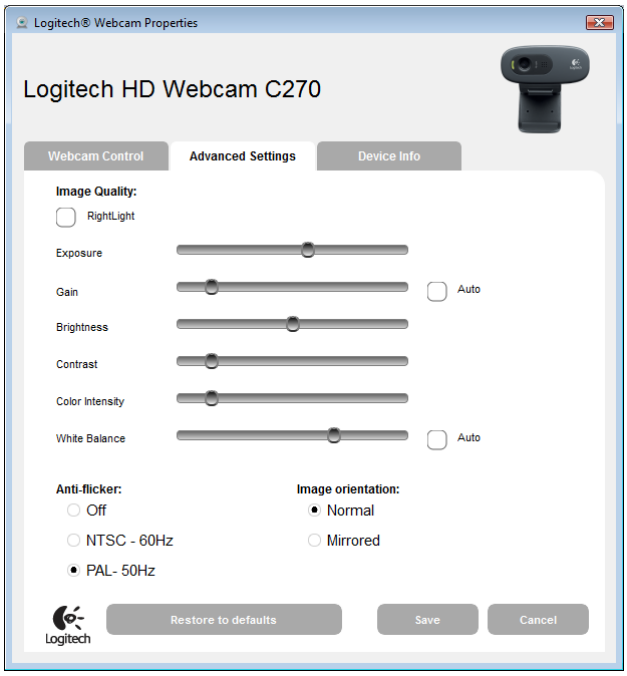
Hier wordt beschreven hoe het beeldherkennen voor robotvoetbal in zijn werk gaat. Dit document is heel erg gericht op de source code van het robotvoetbal project maar er staan ook stukken in over de basisprincipes van beeldherkenning.

Het systeem

De structuur van de applicatie ziet er als volgt uit:



De WebCam



De gebruikte camera is een "Logitech HD Webcam C270" 3.0 MegaPixels en 720P die samen met een headset met microfoon voor € 29,95 werd aangeschaft.

USB stuurprogramma

Voor het aansluiten van de WebCam via USB op de computer zijn er drivers van de leverancier. Bij camera's zit vaak ook door de fabrikant geleverde software om de camera in te stellen. **Het beste kunnen alle automatisch beeldcorrecties uitgezet worden.** De voetbal software moet zelf kunnen corrigeren en automatisch helderheids aanpassingen verstoren dat.
Om de in onderstaande hoofdstukken gegeven voorbeelden uit te proberen wordt ervan uitgegaan dat een standaard USB camera is aangesloten.

WebCam service

Microsoft Robotics Development Studio levert zijn eigen service om de beelden van WebCams beschikbaar te maken. Het programma kan zich dan abonneren op enkele van de door deze service geleverde diensten.

De WebCam service gebruiken

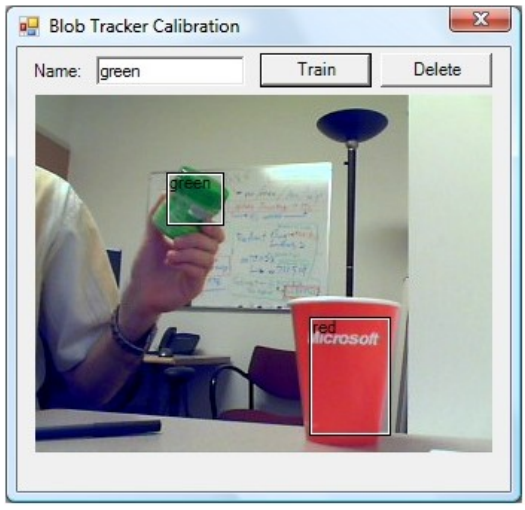
Door in het Start menu in de Microsoft Robotics Developer Studio folder het programma "Run DSS" te starten wordt de webbrowser geopend op adres http://localhost:50000. Door in het menu links in de brwoser op "Control Panel" te klikken is een overzicht te zien van alle beschikbare services. Type "webcam" in de search textbox en 3 verschillende webcam services worden getoond:

- Simulated Webcam: deze is alleen interessant wanneer een webcam gesimuleerd moet worden in de "Visual Simulation Environment"
- Webcam Replay: deze service kan gebruikt worden om opgenomen beelden af te spelen.
- WebCam: deze service wordt hieronder besproken.

Klik op de "Create" button om de "WebCam" service te starten en druk op de F5 button om de link te krijgen die naar de pagina met het camera beeld verwijst. Deze link is ook benaderbaar via het menu "Service directory" waar tevens andere opgestarte services te zien zijn.

Op de pagina van de webcamservice wordt in het begin alleen het eerste frame getoond want om live beelden te krijgen moet de framecapture gestart worden. Klik daartoe op de "Start" button of om slechts 1 nieuw frame te krijgen op de "Refresh" button.
Door de "Refresh Interval" te wijzigen kan het aantal beelden per seconde geregeld worden maar de performance van de computer heeft hier grote invloed op. Door het Capture Format naar bijvoorbeeld 160 * 120 te wijzigen kunnen zelfs tientallen frames per seconde (fps) gehaald worden terwijl dat in hogere resoluties misschien slecht enkele fps zijn. Hieronder een overzicht van typisch te halen waarden waarbij "Refresh Interval" op 10 ms of 100 fps is gezet, wat toch niet haalbaar is, omdat de maximale framerate van de gebruikte Logitech camera maar maximaal 30 fps of 33 ms is.

Capture Format	Display Format	FrameRate
JPEG	160 * 120	38
GIF	160 * 120	41
BMP	160 * 120	39
JPEG	640 * 360	18
GIF	640 * 360	16
BMP	640 * 360	16
JPEG	640 * 480	15
GIF	640 * 480	15
BMP	640 * 480	9
JPEG	1280 * 720	4.1
GIF	1280 * 720	3.7
BMP	1280 * 720	3.4



In TaakBeheer is ook te zien dat grotere Display Formats de computer meer belasten. Maar hoewel de breedte van het bovenstaande grootste formaat 8 keer (namelijk 1280 / 160) groter is en de hoogte 6 keer groter (namelijk 720 / 120) dus het oppervlak 48 keer groter is dan het kleinste, blijkt de FrameRate niet 48 maar slechts 12 maal zo klein te zijn.
Om de frames op te halen is de browser voortdurend bezig de pagina te verversen en wordt deze ook iedere keer weer opgebouwd. De HTML is hier dus de grootste vertragende factor.

Bij het beeldherkenningsprogramma gebeurd dit niet en wordt alleen het beeld opgehaald wat de performance aanzienlijk verhoogt.
"Create" ook de "BlobTracker" service eens met de "BlobTrackerCalibrate.manifest.xml". Als er geen dialoog verschijnt moet de DSS opnieuw opgestart worden. Zie plaatje rechtsboven.
Tijdens bovenstaande experiment kan het zijn dat er een rode error melding wordt getoond in de CMD console. Klik dan op de "Control Panel" pagina en klik, na de "Search" textbox leeg gemaakt te hebben en "Running Services" aangevinkt te hebben, op de "Drop" button naast de services die voor de webcam opgestart zijn. Start daarna weer opnieuw de "WebCam" service op.

De source code

De source code van de WebCam service is terug te vinden in de "C:\Users\JE_NAAM\Microsoft Robotics Dev Studio 4\samples\Sensors\WebCam" folder. Door te dubbelklikken op "WebCam.csproj" wordt het project automatisch in "Visual c#" geopend.
Soms is converteren noodzakelijk maar bij gebruik van de laatste Visual Studio 2010 C# en de laatste Microsoft Robotics Development Studio 4 zou dit niet moeten optreden. Laat anders het proces voor de zekerheid ook een backup maken.
Hieronder kort een beschrijving van de inhoud van dit project omdat de "SimpleVisionModule" hiervan uitgebreid gebruik maakt.
In de solution explorer zijn de volgende bestanden te zien:

- Properties: bevat properties voor dit DSS service project
- References: bevat alle door dit project gebruikte dlls
- Config: bevat verwijzingen naar gebruikte xml files met unieke namen voor elk van de door de WebCam service project beschikbaar gestelde services (zie verderop in deze lijst voor de .cs files van deze services)
- AssemblyInfo.cs: bevat assembly informatie over de gegenereerde dll welke de WebCam service gaat leveren binnen de DSS server
- DeployInfo.xml: hierin wordt de afhankelijkheid=dependency van WebCamPipeServer.exe (in C:\Users\JE_NAAM\Microsoft Robotics Dev Studio 4\bin te vinden) gemeld zodat deze serverapplicatie gestart kan worden om beelden van aanwezige WebCams beschikbaar te stellen aan de WebCam service
- PipeClient.cs: deze client initieert en start de WebCamPipeServer.exe en stuurt commando's (zoals besproken voor de webapplicatie in hoofdstuk De WebCam service gebruiken) via WebCamPipeServer naar de WebCams.
  De WebCamPipeServer stuurt de beelden naar PipeClient welke de beelden weer beschikbaar stelt aan WebCam.cs
- PlatformDependencies.xml: Dit bestand is een overblijfsel uit een eerdere versie van MSRDS
- Replay.cs: Kan eerder door de WebCam service opgeslagen beelden opnieuw afspelen
- SaveStream.cs: Biedt de mogelijkheid om beelden op te slaan als file en weer in te lezen
- WebCam.cs: Tijdens opstarten wordt StartPipeServer() in PipeClient aangeroepen zodat laatste gebruikt kan worden om de verschillende opdrachten aan de WebCam te geven. Ook zorgt WebCam dat benodigde data voor de webpagina gebouwd wordt en requests van de webpagina afgehandeld worden. Dit is de service die in de Visual Programming Language gekozen kan worden
- webcam.user.xslt: dit zorgt voor de opbouw van de webpagina zoals die in hoofdstuk De WebCam service gebruiken besproken werd
- WebCamService.Image.png: dit is het plaatje zoals gebruikt in VPL
- WebCamService.Thumbnail.png: idem
- WebCamState.cs: is bedoeld om de mogelijkheden die de camera biedt te registreren
- WebCamTypes.cs: definieert de types van data classes die gebruikt worden binnen dit project.

Gebruik van Services
Hoewel het gebruik van services niet heel makkelijk is, biedt de WebCam service toegang tot bijna alle mogelijkheden van WebCams. Hardware voor beeldherkenning moet bovendien meestal zo krachtig zijn als de reeds aanwezige PC dus is MSRDS een eenvoudige oplossing.

Alternatieven
Een andere manier om een WebCam te gebruiken is programmeren via DirectX. Alle genoemde functionaliteit zoals opslaan van beelden, instellen van resolutie en gebruik van meerdere camera's moet dan wel nog geprogrammeerd worden of uit een library gehaald worden als OpenCV.
Een robot is bijvoorbeeld "ViewPort Ultimate + Propeller Proto Board USB" van $179,- met een Parallax Propellor Multi Core in combinatie met de ViewPort library, welke Parallax's implementatie is van Opensource Computer Vision library (OpenCV) http://opencv.willowgarage.com/wiki/
Hier bijvoorbeeld een Parallax dansrobot http://www.youtube.com/watch?v=JmpPm2DioBo

De HCCSimpleVisionModule
Deze module doet het echte werk voor de voetbal beeldherkenning. Het hele project van kan van http://hccsoccer.codeplex.com/ gedownload worden. Volg de aldaar te vinden handleiding voor installatie, open het project in Visual C# en bekijk vervolgens de HCCSimpleVision Module. Deze module bestaat uit de volgende onderdelen:

- Properties: bevat properties voor dit DSS service project
- References: bevat alle door dit project gebruikte dlls
  Dat is onder andere de HCCGenericVisionModule.Y2011.M09.Proxy welke een verwijzing is naar de module met dezelfde naam. Van die module worden namelijk de HCCGenericVisionModuleTypes gebruikt met daarin de afmetingen van het veld en de goals en de positie van de voorlopig 4 spelers. Deze worden namelijk niet alleen voor beeldherkenning maar ook voor de HCCRefereeModule gebruikt om goals en overtredingen te kunnen vaststellen.
- Resources: kan afbeeldingen bevatten
- AssemblyInfo.cs: bevat assembly informatie over de gegenereerde dll welke binnen de DSS server de WebCam service gaat leveren
- HCCSimpleVisionModule.cs: zie paragraaf met deze naam
- HCCSimpleVisionModuleTypes.cs: zie paragraaf met deze naam
- VisionForm.cs: zie paragraaf met deze naam

VisionForm.cs
Zodra de applicatie start wordt in "HCCReferee.HCCSimpleVision.Manifest.xml" gezocht naar andere services die gestart moeten worden. In dit document worden achtereenvolgens: "HCCRefereeModule", "HCCSimpleVisionModule" en "WebCam" opgestart. Ook bijbehorende configuratie files worden geladen:
"hccrefereemodule.config.xml", "hccsimplevisionmodule.config.xml" en "webcam.config.xml" waarin allerlei waarden opgeslagen kunnen worden zoals callibratie gegevens. Omdat iedere omgeving zijn unieke calibratie nodig heeft is dit nog niet geimplementeerd.
Dus direct nadat de "HCCRefereeModule" start wordt ook de "HCCSimpleVisionModule" opgestart welke in zijn "start()" methode een "VisionForm" creeert.
De "VisionForm" bestaat eigenlijk uit 3 c# files:

- VisionForm.cs (de code hiervan is zichtbaar door "View Code" uit het context menu van VisionForm.cs te kiezen. VisionForm is de plek waar het callibreren plaatsvindt en de resultaten getoond worden)
- VisionForm.designer.cs (dit is sourcecode welke de coordinaten, afmetingen, teksten en tabposities van de elementen in het Form bevat)

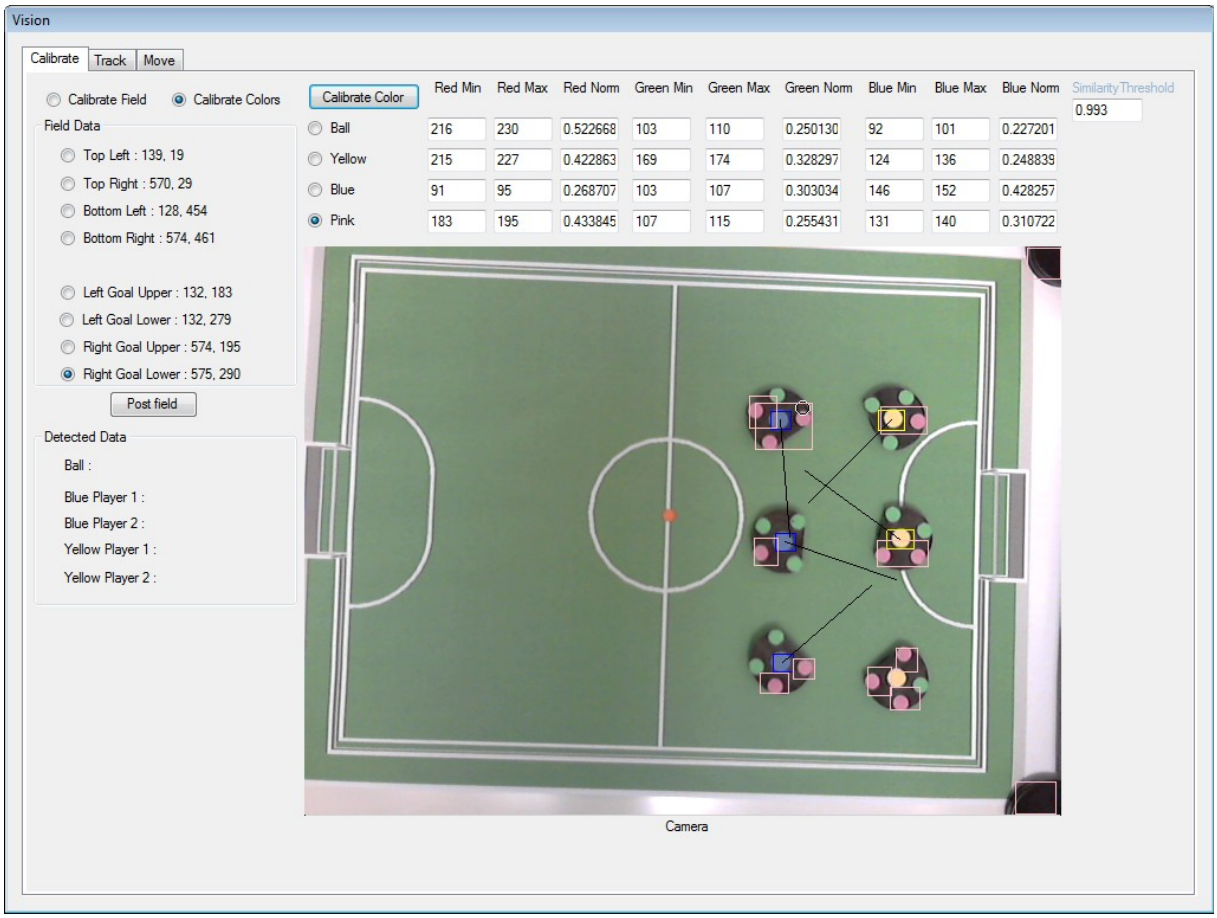
- VisionForm.resx (hierin kunnen teksten en waarden eenmalig staan en op meerdere plaatsen op het form getoond worden. Wanneer deze teksten dan in dit document aangepast worden wijzigen ze tegelijkertijd op alle plaatsen in het form).

Calibreren met de VisionForm

Zoals te zien is op onderstaand plaatje biedt het VisionForm de mogelijkheid om te kiezen tussen "Calibrate Field" en "Calibrate Colors".

Calibreren van afmetingen

Bij iedere beeldherkenning door robots of andere geautomatiseerde systemen is het nodig afmetingen te calibreren. Dit kan zijn het vastleggen van afmetingen van een ruimte waarin de robot zich beweegt of die hij in de gaten houdt en vormen van voorwerpen die de robot moet vasthouden.
Wanneer "Calibrate Field" wordt gekozen verandert de muiscursor in een kruis en zorgt klikken op de WebCam afbeelding dat de geklikte coordinaat bij de gekozen "Field Data" wordt gezet. De gevonden veld coordinaten worden door klikken op de button "Post field" doorgegeven aan de "HCCRefereeModule" noodzakelijk om de spelregels te handhaven. Bovendien hoeft beeldherkenning door "HCCSimpleVisionModule" alleen plaats te vinden binnen deze grenzen.



Calibreren van kleuren

Veel beeldherkenning vergt een special soort licht: ambient licht is het meest geschikt om ruimtes te herkennen, strijk licht van opzij om vormen te herkennen en infrarood is het meest geschikt in het donker maar wordt ook gebruikt om rottend fruit, bloedvaten onder de huid en patronen van de iris te detecteren. Kleuren bepalen of iets een gezicht is en helpen voorwerpen te onderscheiden.
Met "Calibrate Colors" verandert de muiscursor in een cirkel met een door slepen varierende grootte waarmee een kleurgebied geselecteerd kan worden. Dit biedt de mogelijkheid te calibreren zonder

dat de hoogte van de camera invloed heeft. Wanneer de "Calibrate Color" button wordt geklikt worden de gevonden kleurwaarden opgeslagen bij de gekozen kleur.
Het "Similarity Threshold" nummer rechtsboven bepaald welke mate (<1.0) een kleur overeen moet komen met een ingestelde kleur om als die kleur geteld te worden.

Calibreren van vervorming

Omdat iedere camera een lens heeft wordt het licht afgebogen en ontstaat er een vervorming aan de randen van het beeld. Hiervoor is te compenseren maar deze functionaliteit vergt veel rekenkracht en is voor een camera die altijd op dezelfde hoogte en hoek staat te verwaarlozen. Het feit dat de lijnen van het voetbalveld niet exact parallel aan de randen van het beeld lopen is echter wel eenvoudig te compenseren door een simpele matrix transformatie.

HCCSimpleVisionModuleTypes.cs

In dit bestand worden de classes gedefinieerd die de service gebruikt om met de buitenwereld te communiceren. Het is belangrijk om te weten dat een service niet werkt met het aanroepen van methodes en parameters maar door het via poorten (brievenbussen) overdragen van berichten met data (brieven). Deze data wordt door onderstaande classes automatisch geconverteerd naar xml-tekstberichten die een vergelijkbare structuur hebben als html documenten met "<" en ">".
Hieronder zijn de minder belangrijke classes in schuinschrift.

`public sealed class Contract` *Dit is de DSS identifier waarmee de HCCSimpleVisionModule als service bekend is.*
`public class HCCSimpleVisionModuleState` In deze class worden alle calibratie waarden opgeslagen om later opgezocht te kunnen worden.
`public class HCCSimpleVisionModuleOperations : PortSet<DsspDefaultLookup, DsspDefaultDrop, Get, Subscribe, SetObjectTrackingColor, UpdateFieldMeasures, NotifyObjectDetection>` *Dit is een lijst met alle door deze service beschikbaar gestelde classen welke voor doorgeven van data gebruikt worden.*
`public class Get : Get<GetRequestType, PortSet<HCCSimpleVisionModuleState, Fault>>` *Deze class implementeert iedere service om zijn status te kunnen retourneren.*
`public class Subscribe : Subscribe<SubscribeRequestType, PortSet<SubscribeResponseType, Fault>>` *Deze class implementeert iedere service zodat er op geabonneerd kan worden.*
`public class SetObjectTrackingColor : Update<ColorVector, PortSet<DefaultUpdateResponseType, Fault>>` Deze class wordt gebruikt om de TrackingColors op te slaan.
`public class UpdateFieldMeasures : Update<FieldMeasures, PortSet<DefaultUpdateResponseType, Fault>>` Deze class wordt gebruikt om de veldafmetingen te zetten.
`public class NotifyObjectDetection : Update<ObjectResult, PortSet<DefaultUpdateResponseType, Fault>>` Dit is een bericht dat ontvangen wordt wanneer een object gedetecteerd is.
`public class ColorVector` Dit is de belangrijkste class voor het bepalen of kleuren overeenstemmen.
`public class FieldMeasures` Dit is de class waarin de veld en goal afmetingen worden opgeslagen.
`public class ObjectResult` Deze class kan gegevens opslaan van een gedetecteerd object.
`public struct PointType` Deze class heeft een x en een y coordinaat en kan bepalen of twee punten exact hetzelfde zijn.
`public struct RectangleType` Dit is een class voor representeren van een rechthoek met x en y coordinaten, breedte en hoogte en kan bepalen of twee rechthoeken exact hetzelfde zijn.
`public struct BotTracksType` Hier wordt de positie van de voetbal speler opgeslagen en kunnen wederom de coordinaten met andere bots vergeleken worden.

ColorVector

De ColorVector legt de kleur van een pixel vast. Daartoe heeft deze 3 waarden: rood, groen en blauw met waarden van 0 tot en met 255.
Wanneer een kubus met 3 assen (breedte, hoogte en diepte) de kleurruimte zou voorstellen dan kan een punt in die ruimte met coordinaten rood, groen en blauw (r, g, b) bepaald worden.
Vanuit de oorsprong O, waarvoor het hoekpunt links/onder/voor van de kubus gebruikt wordt, bepaalt het aantal punten naar rechts de roodwaarde of r-coordinaat, naar boven de groenwaarde of g-coordinaat en naar achter de blauwwaarde of b-coordinaat.
De denkbeeldige pijl vanuit de oorsprong naar zo'n punt (r, g, b) heet een Vector. Hieronder een overzicht van een paar typische kleur vectoren:
Zwart (0,0,0); Rood (255,0,0); Groen (0,255,0); Blauw (0,0,255), Wit (255,255,255)

Twee kleuren stemmen het best overeen wanneer hun vectoren dezelfde richting hebben. Dus hoe kleiner de hoek die twee vectoren met elkaar maken hoe meer de kleuren overeenkomen. Het maakt dan niet uit of de ene vector iets meer groen en de andere iets meer blauw heeft als de vector waarmee zij vergeleken worden, zolang ze zich maar binnen een kegel rondom de referentie vector bevinden.

De lengte van een kleurvector geeft daarenboven de helderheid van de kleur aan.
Wit is rgb (255,255,255) en grijs rgb(128,128,128). Deze vectoren hebben echter exact dezelfde richting maar verschillen in helderheid. Voor kleurherkenning is de helderheid van minder belang omdat deze door omgevingslicht en schaduwen toch steeds verandert.

Inproduct

In de lineaire algebra (of vector meetkunde) wordt veel gebruik gemaakt van formules om in 3 dimensionale ruimtes te kunnen rekenen zoals roteren, verplaatsen en vergroten van objecten.

Een van die formules om de overeenstemming van 2 vectoren te bepalen is het Inwendig Product(ook dot product of scalar product): http://nl.wikipedia.org/wiki/Inwendig_product.
$\boldsymbol{a \cdot b = |a||\, b|\, cos\theta}$
Dit betekent: het produkt van 2 vectoren is gelijk aan het product van de lengtes van de vectoren maal de cosinus van hun hoek. Een andere schrijfwijze voor deze formule is:

$$\boldsymbol{cos\theta = \frac{a \cdot b}{|a||\, b|}}$$

Stel dat er twee kleurenvectoren zijn en er moet bepaald worden of deze bijna hetzelfde zijn dus dat hun hoek minimaal is. Wanneer twee vectoren loodrecht op elkaar staan (hun hoek is 90°) dan is de sinus van hun hoek 1 en de cosinus is dan 0. In dit geval zijn de kleuren totaal verschillend. Bij een onderlinge hoek van 0° is de sinus 0 en de cosinus 1 en zijn de kleuren dus exact hetzelfde.

Het inproduct wordt als volgt bepaald:

1. bepaal de lengte l van de vectoren volgens Pythagoras(wiens formule ook in 3D geldt)
   $l1^2 = r1^2 + g1^2 + b1^2$ of $l1 = \sqrt{(r1^2 + g1^2 + b1^2)}$
   $l2^2 = r2^2 + g2^2 + b2^2$ of $l2 = \sqrt{(r2^2 + g2^2 + b2^2)}$

2. bepaal het product p van de vectoren onderling. Het is door wiskundigen bewezen dat dit

   als resultaat geeft: de cosinus van de hoek van de twee vectoren maal de lengte van de ene

maal de lengte van de andere vector:

$$p = r1 * r2 + g1 * g2 + b1 * b2$$

3. Wanneer nu hetgeen in 1 en 2 gevonden wordt in de formule voor inwendig product wordt ingevuld is de cosinus van de hoek:

$$cos\theta = \frac{p}{l1 * l2}$$

Hieronder een aantal voorbeelden waarbij cosinus=1 betekent dat de kleuren exact hetzelfde zijn:

r1	g1	b1	lengte 1			
255	255	255	441.67	Grenswaarde		0.993
r2	**g2**	**b2**	**lengte 2**	**product**	**cosinus**	**Gelijk?**
255	255	255	441.67	195075	1	ja
200	255	255	412.37	181050	0.99405269	ja
200	200	255	380.82	167025	0.99302284	ja
200	200	200	346.41	153000	1	ja
255	0	0	255.00	65025	0.57735027	nee
0	255	0	255.00	65025	0.57735027	nee
0	255	255	360.62	130050	0.81649658	nee

Sourcecode van inproduct

Hieronder staat de code die bovenstaande weging uitvoert in HCCSimpleVisionModuleTypes.cs:

```
public double Magnitude()
{
    return Math.Sqrt(Red * Red + Green * Green + Blue * Blue);
}

public static double DotProduct(ColorVector cv1, ColorVector cv2)
{
    if (cv1 == null)
        return 0;
    if (cv2 == null)
        return 0;

    return (cv1.Red * cv2.Red) + (cv1.Green * cv2.Green) + (cv1.Blue * cv2.Blue);
}

public static double CompareSimilarity(ColorVector cv1, ColorVector cv2)
{
    if (cv1 == null)
        return 0;
    if (cv2 == null)
        return 0;

    double mag = (cv1.Magnitude() * cv2.Magnitude());
    if (mag != 0)
        return (DotProduct(cv1, cv2) / mag);
    return 0;
}
```

Met dank aan: http://freespace.virgin.net/hugo.elias/routines/r_dot.htm

HCCSimpleVisionModule.cs

Dit bestand bestaat uit twee classes:

```
class HCCSimpleVisionModuleService : DsspServiceBase
```

en

```
public class ImageProcessingResult
```

ImageProcessingResult, helemaal onderaan het bestand, wordt veelvuldig gebruikt voor het doorgeven van beeldanalyse resultaten tussen methoden van HCCSimpleVisionModuleService en zal indien nodig bij die methoden besproken worden. Dat proces ziet er als volgt uit:

HCCSimpleVisionModuleService

```
protected override void Start()
```

Hier begint het optuigen en opstarten van de service voor beeldbewerking.Dit is ook de eerste belangrijke methode in deze class. Erboven staat vooral intialisatie code.
Deze methode wordt maar een maal aangeroepen en voert de volgende stappen uit:

- de state opslaan van de service zoals
    1. Webcam beeldprocessing interval van 100 ms
    2. Instellen default detectie kleuren
    3. Minimale grootte van kleur detectie gebied instellen
- Aanmaken van een ImageProcessingResult voor opslaan van resultaten van detectie van ieder van de kleuren
- Registreren dat de kleurdetecties (NotifyObjectDetections) naar de methode ObjectDetectionHandler() gestuurd moeten worden via de _internalPort. De _internalPort is van het type HCCSimpleVisionModuleOperations(zie boven) en kent de diverse berichten
- Opstarten van het VisionForm via CreateVisionForm()
- `Activate(Arbiter.ReceiveWithIterator(false, TimeoutPort(3000), GetFrame));`
  Het opstarten van het GetFrame() proces dat de webcam beelden gaat leveren en de beelden bewerkt voor de detectie en de NotifyObjectDetections gaat leveren. De eerste keer wordt 3000 ms gewacht om alles tijd te geven om op te starten voordat dit intensieve proces per 100 ms gaat worden uitgevoerd

```
public void ObjectDetectionHandler(NotifyObjectDetection update)
```

In de update.Body staan gegevens wat ,object of beweging, is gedetecteerd en waar.
Dit wordt bijgehouden in de _state van HCCSimpleVisionModule Service.
Vervolgens wordt een Response verstuurd naar de verzender dat het bericht ontvangen is.
Bovendien wordt via de SubscriptionManager aan alle andere abonnees de NotifyObjectDetection gestuurd.

```
public IEnumerator<ITask> GetFrame(DateTime timeout)
```

Hier wordt eerst een frame van de camera opgehaald met `_webcamPort.QueryFrame()`en in _curFrame opgeslagen.Vervolgens wordt voor iedere te detecteren kleur de methode ProcessFrameHandler() aangeroepen met:

- de kleurinformatie voor detecteren van de oranje bal of labelkleuren roze, geel en blauw
- kleurgrenswaarde (+/- 0.99)
- _curFrame (het huidige beeld)
- _testFrame en _testFrame2 welke een zwart beeld representeren met alleen de gedetecteerde pixels ingekleurd
- _lastProcessingResult waarin alle gevonden gebieden met de overeenstemmende kleuren opgeslagen worden.

Met de resulterende kleurposities kan `AnalyseRobotPositionDirections(_lastProcessingResult)` de posities en de kijkrichting van de robotvoetballer bepalen en weer in _lastProcessingResult opslaan.
Met `_internalPort.Post(..(..(_lastProcessingResult)))` worden partijen op de hoogte gebracht van de resultaten.
`DrawImages(_curFrame)` zorgt dat de beelden in het geheugen opgebouwd worden en onderstaande zorgt dat VisionForm op de hoogte gebracht wordt van de nieuwe beelden.

```
FormInvoke setImage = new FormInvoke(
    delegate()
    {
        _form.CameraImage = bmp;
        _form.TestImage = testBmp;
        _form.TestImage2 = testBmp2;
    }
    WinFormsServicePort.Post(setImage);
```

In de VisionForm.designer.cs zorgt `this.picCamera.Paint += new System.Windows.Forms.PaintEventHandler(this.picCamera_Paint);` ervoor dat de methode `picCamera_Paint` () wordt aangeroepen om deze `FormInvoke` af te handelen. `picCamera_Paint`() tekent vervolgens de gekleurde vierkantjes en de richtingslijnen. Dat `picCamera_Paint` () toegang heeft tot de geanalyseerde data in _result komt omdat deze in de constructor aan een lokale reference(pointer) variabele is toegekend:

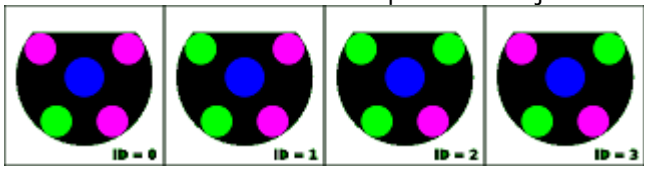
```
public VisionForm(HCCSimpleVisionModuleOperations mainPort, ImageProcessingResult[]
result)
{
    InitializeComponent();
    _mainPort = mainPort;
    _result = result;

}
```

Vervolgens stopt de `GetFrame` methode alle gevonden posities in een _visionData welke naar de HCCRefereeModule gestuurd worden.
Daarna volgt de herstart van GetFrame(): `Activate(Arbiter.ReceiveWithIterator(false, TimeoutPort(_state.WebCamPollingIntervalInMs), GetFrame));` en begint het process opnieuw.

`private BotTracksType findBotWithBotNumber(ImageProcessingResult processingResult, int numberOfCircles)` bepaalt aan de hand van het aantal roze cirkels (1,2 of 3) op het robotlabel het rugnummer van de speler wat nodig is om de richting van een speler te berekenen.

`private void AnalyseRobotPositionDirections(ImageProcessingResult[] result)`bepaalt voor ieder van de teams de posities en richting van de spelers. Hierbij wordt uitgebreid gebruik gemaakt van de resultaten verkregen door `SegmentationRegions()` welke verderop besproken wordt.

`private void compareRectangles(ImageProcessingResult[] result, ColorVector.TrackingColor trackingColor)` Hier wordt eerst het middelpunt van de blauwe of gele cirkels bepaalt. Daarna wordt gekeken hoeveel roze cirkels rond een blauwe of gele cirkel staan. Het aantal roze cirkels bepaalt namelijk in welke richting de robot kijkt.



Eerst wordt het middelpunt van de roze cirkels onderling bepaalt. De lijn dat dit middelpunt en het middelpunt van de blauwe cirkel verbindt hoort altijd een vaste hoek te maken met de loodlijn van de voorkant omlaag naar het middelpunt van de blauwe cirkel. Bij 3 cirkels is dat 75˚, bij 2 cirkels aan de rechterkant 105˚ en bij 1 cirkel 135˚.

Het label met ID=3 kan deze methode niet gebruiken omdat het middelpunt van de twee roze cirkels midden op de blauwe cirkel komt te liggen en de beeldherkenning is niet zo nauwkeurig. *Een betere methode voor de rijrichting van de speler is bepalen waar de rechte voorkant zich bevindt bijvoorbeeld met edgedetection.*

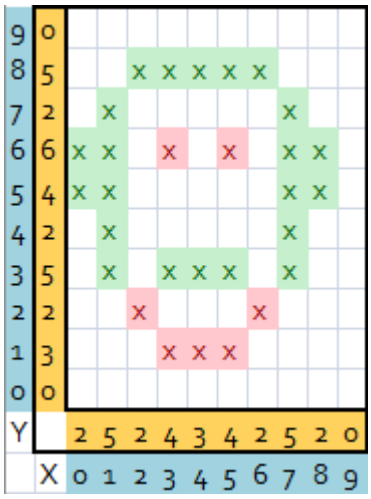
`private void DrawImages(byte[] _curFrame)` stuurt alle beelden voor de verschillende tab pagina's naar het VisionForm.

```
public void ProcessFrameHandler(ColorVector trackingObjectColor,int colorAreaThreshold
, byte[] rgbValues, byte[] testValues, byte[] testValues2, ImageProcessingResult
result)
```
Deze methode wordt dus eenmaal per gezochte kleur aangeroepen. Eerst wordt er een zwarte colorImg gemaakt waarin witte pixels gaan aangeven of pixels op het camerabeeld de gezochte kleur hebben door IsMyColor().
Door aanroepen van `SegmentationRegions()` worden de gevonden kleuren in rechthoeken opgedeeld. Als laatste wordt `DrawCurrentResult()` aangeroepen.

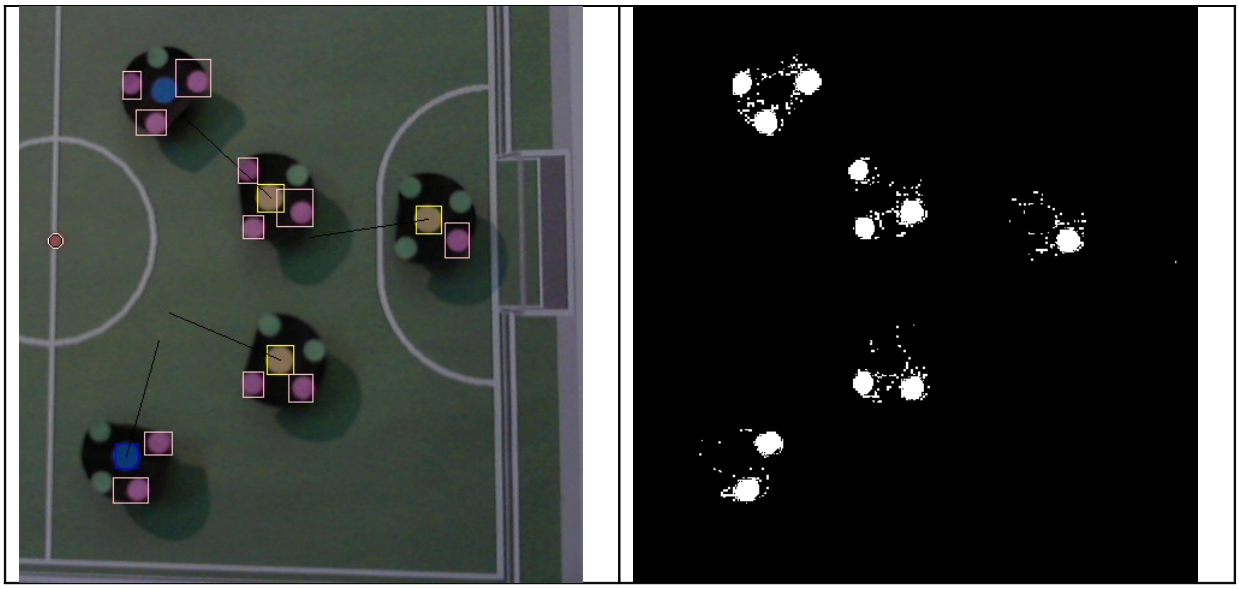
`private void DrawCurrentResult(byte[] testValues, byte[] testValues2)`
De gemaakte detectie beelden worden in het geheugen opgeslagen.

`static bool IsMyColor(int red, int green, int blue, ColorVector myColor)`
Hier wordt gekeken of de pixelkleur overeenkomt met de gezochte kleur. Daarbij wordt `ColorVector.CompareSimilarity` aangeroepen zoals in hoofdstuk ColorVector besproken.

`int SegmentationRegions(byte[] image, int sizeThresh, ImageProcessingResult result)`



In deze methode wordt het zwart/wit verkregen kleur detectie resultaat nader geanalyseerd.
Er wordt gekeken of een pixel hetzelfde is als de pixel links ervan of de pixel erboven. *(Omdat de image een 1 dimensionale byte array is met breedte*hoogte aantal elementen, bevindt de bovenpixel zich breedte posities terug).* Deze pixels vormen dan samen een regio. Volgende pixels waar deze 2 pixels links of boven zitten worden toegevoegd aan dezelfde regio. Nieuwe pixels die geen links of boven buren hebben vormen een nieuwe regio. Hierna worden de grootste segmenten, hier sizeTresh (=200) stuks, bewaard en de rest weggegooid. In het plaatje hiernaast is gesteld dat de 4 grootste bewaard moeten worden en daardoor vallen de ogen, wangen en onderkaak (van verbazing) weg.
Vervolgens worden nog de locaties en afmetingen van de area's bepaald. Al deze informatie wordt opgeslagen in de location[] en boundbox[] van result. Zie het rechtse plaatje hieronder voor de input van `SegmentationRegions`.



```
public static bool Scale(int pixelBytes, byte[] sourceImage, RectangleType sourceRect,
byte[] tarImage, int tarWidth, int tarHeight)
```

Deze methode schaalt binnenkomende frames naar de gespecificeerde size. Het is het beste om zowel de resolutie van de camera als de afbeelding op VisionForm dezelfde grootte te laten hebben om de processor zo min mogelijk te belasten. Hier is gekozen voor 640 * 480. De afmetingen van de images die de webcam levert kunnen ingesteld worden door de webcamservice via http://localhost:60000 te wijzigen.Vooral bij het kiezen van een kleine afmeting is het schalen goed waarneembaar. De beeldherkenning gaat dan ook snel achteruit.

Onderstaande methoden zijn allemaal voor afhandelen van service functies.

```
public void DropHandler(DsspDefaultDrop drop)
```

Al even genoemd in De WebCam service gebruiken wordt hier de service mee beeindigd.

```
public void SubscribeHandler(Subscribe subscribe)
```

Hier worden verzoeken om aan deze service deel te nemen afgehandeld.

```
public IEnumerator<ITask> SetObjectTrackingColorHandler(SetObjectTrackingColor update)
```

Via deze service functie kunnen de in VisionForm gekozen kleuren ingsteld worden.

```
public IEnumerator<ITask> UpdateFieldMeasuresHandler(UpdateFieldMeasures update)
```

Via deze service functie worden de afmetingen van het speelveld vastgelegd.

Conclusie

Zoals uit bovenstaande blijkt is beeldherkenning pittige materie. Omdat hierboven gebruik gemaakt wordt van services is alles nog een tikkeltje ingewikkelder. Daar zou een aparte library met alleen de beeldherkennings functies uitkomst in kunnen bieden. Aan de andere kant kan betere documentatie die bijvoorbeeld met Doxygen www.doxygen.nl is gegenereerd ook meer inzicht verschaffen. Houdt de www.twintellect.com website in de gaten voor deze documentatie.

Ook het programma kan nog wel enige verbetering gebruiken zoals:

- Kleuren in xml opslaan en uit xml halen bij het opstarten van VisionForm
- De Gemiddelde kleur van de afbeelding bepalen om te compenseren voor ambient licht
- Kleur waarden uit meerdere samples bepalen en dan middelen
- Iedere kleur zijn eigen grenswaarde geven
- Meer inzicht verschaffen door een histogram te tonen op een van de tabs
- Een hoogdoorlaatfilter gebruiken om het grasveld onzichtbaar te maken
- Eerst de kleuren wit en groen van het speelveld verwijderen

- Een analyse afbeelding maken met daarin de overeenkomst van iedere pixel met de gezochte kleur
- Gebruik maken van detectie van object randen
- De grootte van de te detecteren objecten vastleggen
- OpenCV libraries gebruiken

Verder lezen

Een goed boek (hoewel met veel wiskunde en in het engels) om wat meer te leren over beeldherkenning is: "Computer Vision: Principles and Practice" van Elektor:
http://www.elektor.nl/products/books/robotics/computer-vision-uk.428878.lynkx
Voor een voorbeeld van beeldherkenning met auto's is er onderstaande engelse presentatie:
http://www.cs.cmu.edu/~gunhee/publish/wacv08_gunhee.ppt

Colofon

Auteur: Iwan Tolboom, iwan.tolboom@chello.nl
Website: www.twintellect.com