Real life Pong met BB-8: van digitaal naar tastbaar spel

Patrick Spaans, Joris Jonkers Both, Kylian van Geijtenbeek en Geerten Rijsdijk

Inleiding

Sinds het verschijnen van de computer zijn er verscheidene klassieke spellen ontwikkeld die nog tot op de dag van vandaag voor veel mensen tijdloos zijn. Ondanks het gigantische succes van simpele spellen zoals *Pong*, zijn ze altijd beperkt gebleven tot enkel de computer; besturing uitsluitend via muis, toetsenbord en tegenwoordig een enkele keer via *touchscreen*. In een tijd waarin men steeds meer tijd doorbrengt achter de computer, komt er langzaam maar zeker ook weer steeds meer vraag naar activiteit die mensen bij de computer vandaan haalt.

Rajaratnam et al. [2] heeft een theorie geschreven over hoe een kunstmatig intelligente robot spelregels van spellen kan leren en vervolgens die spellen kan spelen, maar het huidige onderzoek naar robots is nog niet ver genoeg om dit te kunnen realiseren. Als dit in de toekomst wel mogelijk is, is het ook mogelijk om computerspellen in *real-life* te reconstrueren. Dit heeft als groot voordeel dat het mensen, met name de jeugd, weghaalt van de computer en er daarbij voor zorgt dat mensen minder lang zullen zitten op dagbasis. Te veel zitten is volgens onderzoek van Owen et al. [1] een groot probleem, aangezien het resulteert in veel gezondheidsklachten. Daarom is het belangrijk dat er meer vormen van vermaak komen waarbij mensen actiever bezig zijn. Door klassieke computerspellen te reconstrueren buiten de digitale omgeving worden mensen gestimuleerd om meer tijd bij de computer vandaan door te brengen en meer interactie aan te gaan met elkaar.

Op wetenschappelijk niveau is dit onderzoek interessant omdat dit onderzoek de mogelijkheden en problemen in kaart brengt als het aankomt op robotica toepassen op real-life spelomgevingen. Naar aanleiding van de problemen die tijdens dit onderzoek naar voren komen, kan eventueel vervolgonderzoek gedaan worden naar hoe men kleine robotische apparatuur dusdanig kan ontwikkelen zodat het ook multifunctioneel is. Zo zou een robotballetje zoals de *Sphero BB-8* met de juiste aanpassingen mogelijk voor meerdere spellen inzetbaar zijn.

Onderzoeksvraag

Hoe kan de Sphero BB-8 gebruikt worden om Pong in het echte leven na te maken?

De eerste vraag die bij dit onderzoek beantwoord moet worden is hoe de *Sphero BB-8* via een computer bestuurd kan worden. Aangezien de *Sphero BB-8* in staat is om met bluetooth via een app bestuurd te worden, wordt verwacht dat het ook mogelijk zal zijn om via een computer commando's te geven. Hierbij zal gebruik gemaakt worden van de javascript bibliotheek Cylon.js om de *BB-8* aan te drijven. Een andere vraag is hoe de positie van *BB-8* en de balkjes gedetecteerd kunnen

worden. De Sphero *BB-8* komt met een ingebouwde odometer en een ingebouwd digitaal kompas. Verwacht wordt dat deze gebruikt kunnen worden om de positie van *BB-8* in het veld te bepalen. De balkjes kunnen gedetecteerd worden middels collision detection. Het volgende obstakel is om ervoor te zorgen dat de *Sphero BB-8* automatisch binnen een veld blijft. Er wordt verwacht dat dit op meerdere manieren gedaan zal kunnen worden. Als de positie van de *Sphero BB-8* bekend is, zowel als de grootte van het veld, dan kan gezorgd worden dat *BB-8* zich omdraait als het zich bij de rand van het veld bevindt. Verder is het ook mogelijk om de randen van het veld af te bakenen en gebruik te maken van collision detection om *BB-8* binnen het veld te houden. Als laatste moet er gekeken worden naar hoe de voortgang van het spel kan worden bijgehouden en kan worden aangegeven door de *Sphero BB-8*. De verwachting is dat de voortgang van het spel bijgehouden zal kunnen worden door de scores van de individuele spelers bij te houden. Als er een punt gescoord wordt of het spel gewonnen wordt, zal *BB-8* dit kunnen doorgeven aan de computer en dit kunnen aangeven door optische signalen af te geven.

Methode

De robot die gebruikt is bij dit onderzoek is de *Sphero BB-8* (figuur 1). Deze robot bestaat uit een bolvormig, bewegend lichaam met een magnetisch hoofd. De robot laat zich besturen door middel van bluetooth low energy (ble) signalen. Vanuit de fabriek wordt de robot geleverd met een smartphone app. Met deze app is het eenvoudig de robot te besturen. Een gebruiker kan virtuele stuurpookjes gebruiken om *BB-8* te navigeren. Deze app kan deze gebruiker signalen omzetten in bluetooth signalen die de robot op kan vangen en kan verwerken tot daadwerkelijke bewegingen.

Figuur 1: de meegeleverde app bij *BB-8* waarmee consumenten de robot kunnen besturen.

Voor dit onderzoek was het noodzakelijk de app te omzeilen en directe toegang tot de robot te verkrijgen. Hiervoor is gebruik gemaakt van code in *Javascript* en *Python 2.7*. Javascript is gekozen door de aanwezigheid van de *Cylon.js* bibliotheek. Met deze bibliotheek is het mogelijk om via Node.js functies aan te roepen die omgezet worden in bluetooth signalen die *BB-8* kan begrijpen. *Python* is gekozen omdat er al eerdere ervaring met de taal was en omdat de software *OpenCV* hiermee gemakkelijk te gebruiken was. Initieel werd geprobeerd *Pong* te maken door de *Sphero BB-8* in een afgesloten veld te plaatsen met balkjes gemaakt van karton. De coördinaten van de robot zouden berekend worden uit de informatie die *BB-8* kan geven met zijn odometer. Deze coördinaten zouden bijgehouden worden om *BB-8* zijn positie in het veld te kunnen bepalen. Aan de hand hiervan zou *BB-8* door het veld kunnen bewegen en door middel van collision detection zou het, wanneer het een rand of het balkje van een speler raakt, op gepaste momenten kunnen omdraaien.

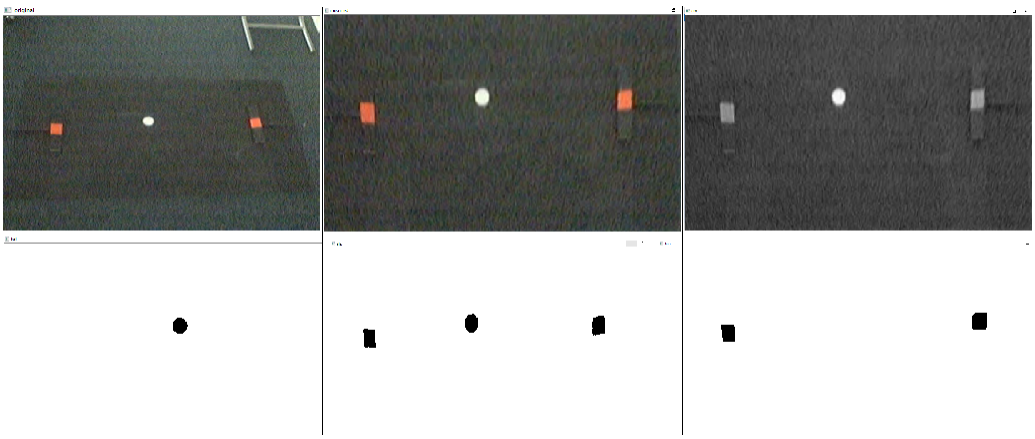
Na enig testen bleek echter dat de *Sphero BB-8* niet in staat was om accuraat zijn bewegingen door te geven. Ook bleek dat de *collision detector* niet sensitief genoeg was om botsingen met zachtere materialen zoals karton te detecteren en ook niet betrouwbaar werkte als er botsingen met harde materialen voorkwamen. Hierdoor was een andere aanpak noodzakelijk.

Voor deze nieuwe aanpak werd gebruik gemaakt van een camera. Deze camera werd op een hoogte van ongeveer 2 meter aan de muur opgehangen. Hoewel de exacte hoogte niet zozeer van belang was, moest de camera wel hoog genoeg hangen zodat het volledige speelveld in beeld was. Ook was het noodzakelijk de camera langs een lange zijde van het speelveld te plaatsen om te voorkomen dat spelers het beeld van de camera zouden kunnen blokkeren.

De *BB-8* werd in een afgezet veld onder de camera geplaatst. Door middel van een perspectief translatie werd een bovenaanzicht gegenereerd dat gebruikt zou worden om de coördinaten van *BB-8* bij te houden en te zien wanneer het tegen een rand zou botsen. De perspectief translatie werd gedaan door de vier hoekpunten van het speelveld aan te geven in coördinaten, en door daar een *openCV* functie op los te laten. De hoekpunten werden verkregen door bij het runnen van de *python* code het camerabeeld weer te geven, en door te klikken in het camerabeeld werden de coördinaten van dit aangeklikte punt teruggegeven als hoekpunt. Dit werd gedaan in een apart *python* file, want zolang het speelveld en de camera op dezelfde positie blijven, is het niet nodig om iedere keer opnieuw de vier hoekpunten aan te geven. Het detecteren van de robot zou met *openCV* blob detectie gedaan worden. Hierbij werd *OpenCV 2.4.13* gebruikt. Helaas was het op deze manier technisch niet mogelijk om de robot betrouwbaar te detecteren. Dit werd veroorzaakt door een te klein contrast tussen de kleur van de robot en de ondergrond. Hierdoor was, mede door de lage resolutie camera, *openCV* niet in staat blob detectie uit te voeren en kon de robot dus niet gevolgd worden.

Om dit op te lossen is het ingesloten veld vervangen door een rechthoekige ondergrond, gemaakt van zwart papier. Hierdoor werd een hoog contrast gecreëerd tussen de voornamelijk witte *Sphero BB-8* en de zwarte ondergrond. Dit RGB-beeld werd in *OpenCV* eerst getransformeerd naar een afbeelding met enkel grijswaarden, waarna vervolgens door middel van een *threshold* elke pixel onder een bepaalde grijswaarde wit gemaakt werd en elke pixel boven de aangegeven grijswaarde zwart gemaakt werd. Dankzij deze beeldbewerking konden lichte objecten worden weergegeven als zwarte vormen op een witte achtergrond, waardoor alle lichte objecten makkelijk herkenbaar waren

voor de blob detectie. Hoewel er door de grootte en de positie van elk object te bepalen was welk van de drie objecten een balkje en welk object *BB-8* was, ontstond er een volgend probleem doordat de herkenning alleen klopte wanneer er daadwerkelijk drie objecten gedetecteerd werden. Onder de invloed van de belichting gebeurde het regelmatig dat één of meerdere objecten niet herkend werden, waardoor de posities van *BB-8* en de balkjes met elkaar verward konden worden. Om dit op te lossen werd besloten de balkjes gedeeltelijk rood te kleuren. De grijstint van rood kon onderscheiden worden van de grijstint van wit, waardoor met verschillende *threshold* bewerkingen een beeld verkregen kon worden wat alleen de *Sphero BB-8* en een beeld van zowel de *Sphero BB-8* als beide balkjes bevatte. Door vervolgens de inverse van het beeld van de *Sphero BB-8* op te tellen bij het beeld van alle drie de objecten, kon een beeld verkregen worden dat enkel de beide balkjes bevatte. Op deze aparte beelden werden vervolgens verschillende blob detecties toegepast en de coördinaten berekend. Hierna konden de *Sphero BB-8* en de balkjes niet meer met elkaar verward worden wanneer één of meerdere objecten tijdelijk niet meer te herkennen waren met behulp van blob detectie. (zie figuur 2).



Figuur 2: v.l.n.r. van boven naar beneden de stadia die het beeld van de camera ondergaat.

Eén van de problemen die regelmatig voorkwam bij het herkennen van *BB-8* met behulp van blob detectie is dat de *Sphero BB-8* niet bolvormig genoeg was om goed herkend te worden door de blob detectie. Een simpele oplossing voor dit probleem was om *BB-8* meer bolvormig te maken, door het hoofd te scheiden van het lichaam. Aangezien het hoofd van *BB-8* geen toegevoegde waarde heeft voor de werking van de *Sphero BB-8* en het alleen maar een hoofd heeft voor cosmetisch effect (zonder hoofd is het niet meer *BB-8*), kon dit worden gedaan zonder enige consequenties. Daarnaast kon de blob detectie voor *BB-8* aanzienlijk verbeterd worden door *BB-8* ten alle tijden licht, het liefst fel licht zoals wit of blauw, te laten geven. Dit zorgde ervoor dat de blob detectie een stuk consistenter was in het herkennen van de *Sphero BB-8*, wat de werking van de *openCV* code bevorderde.

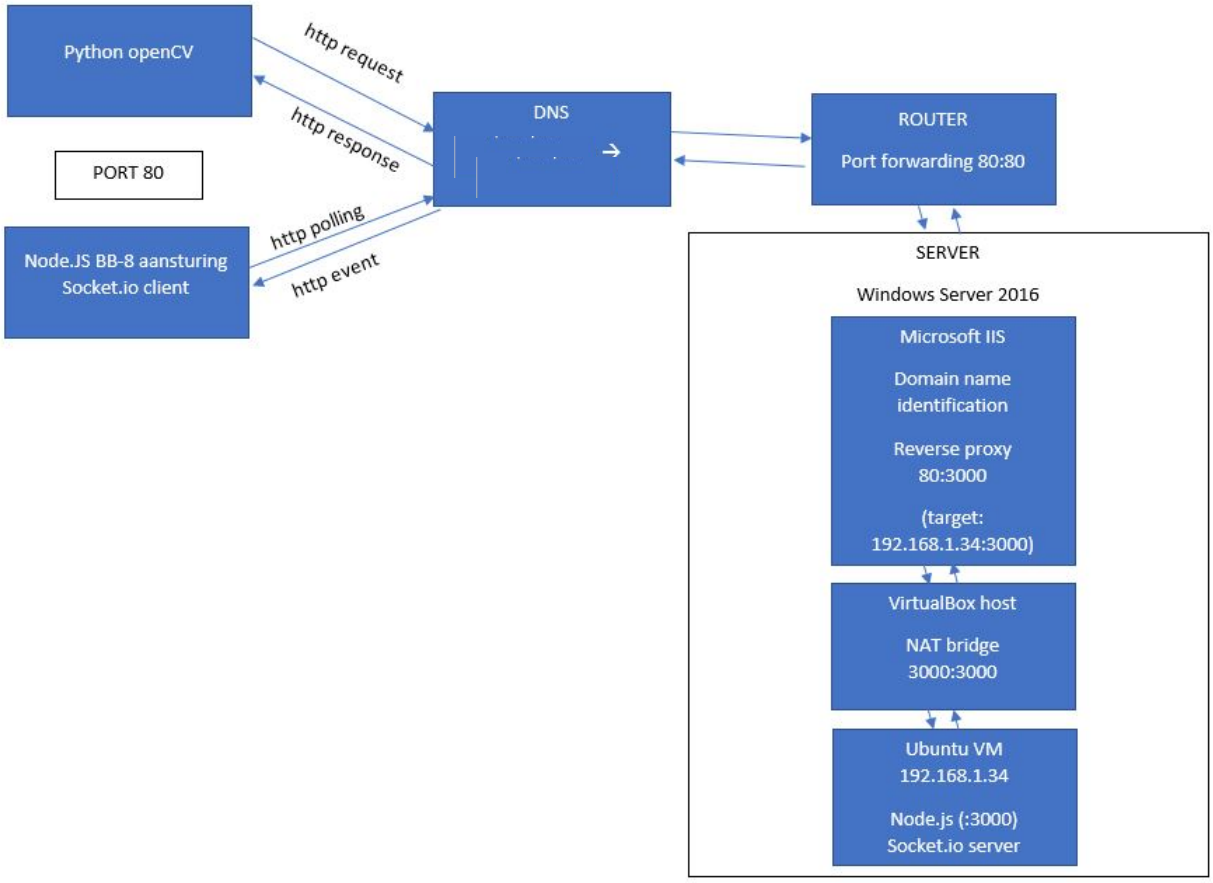
Een ander probleem dat te maken had met blob detectie was het herkennen van de balkjes die door de spelers verplaatst worden (te zien in figuur 3), om op die manier het spel *Pong* te kunnen spelen. De balkjes waren namelijk rechthoekig, en rechthoekige vormen zijn erg ingewikkeld te herkennen met behulp van blob detectie. Een oplossing voor dit probleem is door in het midden

van ieder balkje een rood vierkant te maken, door er rood papier om heen te wikkelen. Dit rode vierkant is met behulp van *openCV* wel te herkennen, en de rode kleur zorgt ervoor dat *openCV* het balkje kan onderscheiden van de *Sphero BB-8* zelf. Het deel van het balkje wat niet ingepakt is met het rode papier, wordt omwikkeld met zwart papier, waardoor het wegvalt in de zwarte achtergrond, en *openCV* het niet herkend. Vervolgens wordt met behulp van de coördinaten van het middelpunt van het rode vierkant bepaalt waar het balkje zich bevindt, waardoor de python code kan bepalen of de *Sphero BB-8* het balkje aanraakt of niet. Verder is er een opgerold zwart a4'tje aan ieder balkje vastgemaakt, wat dient als bedieningsstaaf van het balkje. Hiermee kunnen de balkjes verplaatst worden zonder dat er handen door het camerabeeld heengaan, waarmee je voorkomt dat handen per ongeluk door *openCV* herkent wordt als een balkje of de *Sphero BB-8* zelf.

Figuur 3: het balkje dat bedient wordt door de speler. Het zwarte papier is voor de camera onzichtbaar.

Toen de locatie van de robot succesvol bepaald kon worden, werd een manier bedacht om de *openCV* software, die onder python draaide, te verbinden met de besturing van de robot, die op Node.js draaide. Om beide systemen te verbinden werd besloten een centrale server te gebruiken en zo een API te bewerkstelligen. De centrale server bestaat uit een Windows Server 2016 installatie waar een instantie van *virtualbox* op draait. Deze virtualisatie draait een versie van *Ubuntu* die is uitgerust met een instantie van *Node.js* uitgebreid met *socket.io*. Een uitgebreid traject schema is hieronder weergegeven. Deze *Node.js* instantie luistert naar inkomende http requests. Deze bepaalt dan de actie die bij de opgevraagde url hoort en stuurt dan de aanvraag via *socket.io* door naar alle *BB-8* controller clients. Deze clients luisteren naar deze events en roepen de juiste functie aan die de robot aanstuurt. Bijvoorbeeld: op het moment dat *openCV* detecteert dat de robot een rand van het veld raakt, roept deze de url aan die hoort bij dit type event met bijbehorende parameter (de

richting van de wand die de robot raakt, in dit geval example.nl/collision?id=north). De Node.js instantie op de server detecteert dit en broadcast een signaal naar alle verbonden *Node.js* clients. Deze detecteren vervolgens dat het gaat om een collision event bij de noordwand en sturen een signaal naar *BB-8* om deze te laten omkeren. De *openCV* instantie ontvangt bij een succesvolle aanroep van de API een *http status response* 200. (zie figuur 4).



Figuur 4: de netwerk infrastructuur gebruikt voor communicatie tussen de verschillende computersystemen.

Een van de onverwachte gebreken aan de *Sphero BB-8* was dat de ingebouwde gyroscoop vrij inconsistent was, dus wanneer je het de opdracht geeft om naar het noorden te rijden (dus naar 0 graden), het regelmatig niet naar het noorden reed, en regelmatig een andere positie koos om naartoe te rijden. Om dit op te lossen was het nodig om de *Sphero BB-8* te kalibreren, maar omdat het niet mogelijk was om *BB-8* zelf te kalibreren, was de enige andere oplossing om met behulp van de camerabeelden en *openCV* te bepalen hoever het noorden van *BB-8* afwijkt van de noordelijke

rand van het speelveld. Aan de *Sphero BB-8* werd de opdracht gegeven om een klein stukje naar zijn noorden te rijden, en vervolgens weer terug. Van de camera werden twee frames van deze beweging genomen, één van voor de beweging naar het noorden en één van na de beweging naar het noorden, en daarmee was het mogelijk om aan de hand van het verschil in coördinaten de hoek te berekenen van de afwijking van *BB-8*. Deze hoek werd vervolgens doorgegeven aan de javascript code, die deze hoek als variabele vastzette, en iedere hoek die de *Sphero BB-8* zou moeten nemen volgens de hoekberekeningen werden gecorrigeerd met deze afwijkingshoek.

Data: Camera feed
Result: Berekenen van afwijkingshoek BB-8
$calibrate1 \leftarrow None$
$calibrate2 \leftarrow None$
$x1 \leftarrow 0$
$x2 \leftarrow 0$
$x3 \leftarrow 0$
$x4 \leftarrow 0$
BB-8 rolt noord;
for *i in range 100* **do**
  $frame \leftarrow getframe()$
  if $i == 1$ **then**
    $calibrate1 \leftarrow frame$
    Herken BB-8 in calibrate1;
    $x1 \leftarrow BB-8\ x\ positie$
    $y1 \leftarrow BB-8\ y\ positie$
  end
  if $i == 99$ **then**
    $calibrate2 \leftarrow frame$
    Herken BB-8 in calibrate2;
    $x2 \leftarrow BB-8\ x\ positie$
    $y2 \leftarrow BB-8\ y\ positie$
  end
end
$x\ difference \leftarrow x2 - x1$
$y\ difference \leftarrow y2 - y1$
$angle \leftarrow degrees(atan(x\ difference\ /\ y\ difference))$
if $angle < 0$ **then**
  $angle \leftarrow angle\ +\ 360$
end
BB-8 rolt zuid;
return angle

Algorithm 1: Kalibreren van BB-8

Resultaten

Het is mogelijk gebleken om de *Sphero BB-8* robot te besturen met behulp van de *Cylon.js* bibliotheek en *bluetooth* (low energy). Na het toepassen van de juiste thresholds kunnen de *Sphero BB-8* en de balkjes die gebruikt worden om te spelen goed gedetecteerd worden. Na enige

Tabel 1: Afwijking in graden na calibratie van BB-8

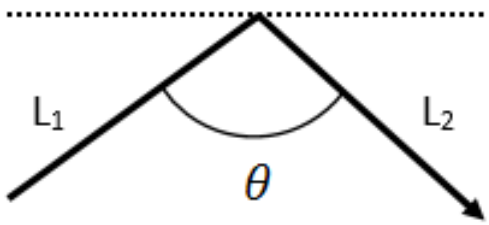
Meting	Afwijking in graden
1	4
2	0
3	0
4	4
5	2

verandering van de lichtcondities echter, kan het voorkomen dat de toegepaste thresholds niet meer correct zijn, waardoor de blob detectie ofwel *BB-8*, ofwel de balkjes niet meer juist kan herkennen, waardoor het zo nu en dan noodzakelijk is om opnieuw de juiste thresholds te vinden en toe te passen.

Tabel 2: Metingen van de hoek θ in graden en afstanden L_1 en L_2 na de *Sphero BB-8* 1 seconde met een snelheid van 100 te laten rijden, waarna het een draaiing van 90 graden maakt en nog 1 seconde met een snelheid van 100 rijdt.

Meting	Hoek θ in graden	Afstand L_1 in mm	Afstand L_2 in mm	Afstandsverhouding heen/terug
1	87	215	190	1.13
2	92	220	210	1.05
3	96	230	220	1.05
4	85	265	180	1.47
5	84	240	210	1.14

In tabel 1 zijn een aantal metingen van de nauwkeurigheid van de kalibratie te zien. Hoewel deze kalibratie zeer nauwkeurig lijkt te zijn met een gemiddelde afwijking van 2 graden, gebeurt het regelmatig dat wanneer de code op verschillende momenten wordt uitgevoerd met dezelfde beginhoek, *BB-8* desalniettemin met een afwijking van 90, 180, of 270 graden begint met rollen. Na een kleine reeks testen waarbij *BB-8* een seconde lang met een snelheid van 100 vooruit rolt, vervolgens 90 graden draait en weer een seconde lang met dezelfde snelheid rolt, bleek dat er bij een dergelijke draaiing nauwkeurigheid verloren gaat. De zojuist beschreven manoeuvre is schematisch weergeven in figuur 5. Hierbij is L1 de waargenomen afstand van de eerste rol, θ de waargenomen hoek waarmee gedraaid werd, welk geprogrammeerd stond als 90 graden en L2 de waargenomen afstand van de tweede rol, die theoretisch gezien even groot zou moeten zijn als L1. Uit de resultaten, te zien in tabel 2, blijkt echter dat de draaihoek tot wel 6 graden afweek van de verwachte 90 graden, en de afstanden L1 en L2 vaak verschillen.



Figuur 5: schematische weergave van de uitgevoerde tests betreffende de behouding van de nauwkeurigheid na het draaien van de robot.

De *Sphero BB-8* botst over het algemeen op de juiste manier met de balkjes. Wanneer de *Sphero BB-8* echter veel licht geeft en in de buurt komt van het rode gedeelte van een van de balkjes kan het gebeuren dat de blob detectie op het balkje niet meer werkt, waardoor *BB-8* pas na een vertraging van zo'n twee tot vier seconden van het balkje af botst. Hoewel het botsmechanisme tussen *BB-8* en de randen van het veld over het algemeen naar behoren werkt, laat de reactietijd van BB-8 vaak nog te wensen over. Het gebeurt regelmatig dat *BB-8* pas van richting verandert als hij zich al een paar centimeter buiten, of juist nog een paar centimeter binnen het veld bevindt. Voor de noordelijke rand geldt dat *BB-8* hier gemiddeld drie centimeter te vroeg of te laat corrigeert en voor de zuidelijke rand is dit gemiddeld 8 centimeter, uitsluitend te laat.

Bij aanraking van de balkjes geeft de *Sphero BB-8* de juiste lichtsignalen af: blauw voor het westelijke balkje, oranje voor het oostelijke balkje. Ook wanneer er een punt wordt gescoord geeft *BB-8* dit correct aan door te gaan knipperen met zijn licht en vervolgens tien seconden niet te bewegen zodat hij in die periode terug in het veld geplaatst kan worden. Nadat drie punten zijn gescoord stopt *BB-8* definitief met bewegen.

Discussie

Tijdens dit onderzoek zijn problemen gevonden, waarvoor oplossingen bedacht moesten worden. Een van de grootste problemen was dat de *Sphero BB-8* een stuk minder kon doen op het gebied van gegevens bijhouden dan gehoopt werd. Hierom werd gekozen om een camera te gebruiken om het spel bij te houden. Deze oplossing heeft uiteindelijk goed gewerkt, maar leidde zelf weer tot nieuwe obstakels. Eén zo'n obstakel was het vinden van *BB-8* via blob detectie. Lichtere objecten, zoals *BB-8*, zijn zeer moeilijk te detecteren op lichte of gekleurde achtergronden vanwege een gebrek aan contrast. Dit is vervolgens opgelost door het speelveld een zwarte achtergrond te geven, het beeld te converteren naar grijswaarden, hier vervolgens met behulp van een *threshold* de lichte objecten laten weergeven als zwarte objecten op een witte achtergrond, waarna de blob detectie weinig moeite had met het herkennen van de *Sphero BB-8*. Deze oplossing heeft vooralsnog goed gewerkt, met als enige nadeel dat het zwarte papier erg gevoelig was voor de weerkaatsing van licht. Wanneer het veld namelijk dicht bij een felle lichtbron lag, bijvoorbeeld een raam, zag de camera het zwarte papier als veel lichtere kleur dan dat het daadwerkelijk was.

Om de balkjes te detecteren, is de keuze gemaakt om ze helemaal zwart te maken met een rood middenstuk. Het zwarte gedeelte is door de camera niet te zien op een zwarte achtergrond, maar het rode deel wel. Hoewel de positie van het balkje hierdoor uiteindelijk goed te detecteren was, is het mogelijk dat, door een andere kleur dan rood te gebruiken, een beter resultaat verkregen had kunnen worden. Ook zou het detecteren van de hele balk, in plaats van alleen het midden, waarschijnlijk geleid hebben tot het makkelijker opmerken van botsingen tussen *BB-8* en een balkje. Er is gekozen om dit niet te doen door de limitaties van blob detectie in *OpenCV 2.4.13*.

Tijdens de uitvoering van het onderzoek zijn er een aantal zaken naar voren gekomen die verbeterd zouden kunnen worden.

Ten eerste zou de camera verbeterd kunnen worden. De gebruikte camera was een *Logitech QuickCam Pro 4000* uit het jaar 2002. De resolutie van de gebruikte camera is 640x480 pixels. Door deze lage resolutie was het detecteren van objecten door *openCV* vaak erg lastig. In veel

lichtomstandigheden was er te veel ruis in het beeld om een duidelijk contrast te kunnen zien. Bij gebruik van een meer moderne camera met hogere resolutie, zou de detectie van objecten significant kunnen verbeteren. Dit zou ook kunnen leiden tot een verbeterde spelervaring van het spel.

Een andere omstandigheid die tijdens het onderzoek af en toe tot problemen leidde was verschil in belichting. Verschillende locaties waar het onderzoek uitgevoerd werd hadden verschillende belichting door o.a. de aanwezigheid van ramen, de felheid van de lampen of het weer buiten. Dit leidde tot problemen bij het herkennen van de *Sphero BB-8* en de herkenningspunten van de balkjes. Om dit op te lossen moest er veel gespeeld worden met de waarden van de thresholds die gebruikt werden om een kleurenafbeelding om te zetten naar een afbeelding met grijswaarden. Het uitvoeren van dit onderzoek in één standaard locatie met constante belichting zou dit probleem verhelpen en leiden tot makkelijkere detectie van zowel *BB-8* als de balkjes van de spelers.

Ook een probleem wat zich tijdens het onderzoek voordeed was dat de *Sphero BB-8* vaak niet herkend kon worden door de onegale kleur van de robot. Dit kon deels worden opgelost door het onderzoek in schemer uit te voeren en de robot fel wit te laten schijnen. Dit zou volledig opgelost kunnen worden door de robot een egale contrasterende kleur te geven. Hierbij zou het beste voor een zwarte kleur op een witte achtergrond gekozen kunnen worden. De reden hiervoor is dat *openCV* bij de gebruikte blob detectie op zoek gaat naar zwarte objecten op een witte achtergrond. In het experiment kwam naar voren dat ondanks het inverteren van de kleuren (het experiment werd immers uitgevoerd met een witte robot op een zwarte achtergrond), het patroon op de robot de detectie door openCV sterk verstoord. Hierdoor zou voor een vervolgonderzoek beter een egale kleur gekozen kunnen worden.

De computer vision software die bij dit onderzoek gebruikt is is *OpenCV 2.4.13*. Deze software is helaas niet in staat om rechthoekige objecten te herkennen met blob detectie. Om dit op te lossen is het midden van het balkje bedekt met rood papier, dat door de vierkante vorm wel gedetecteerd kon worden. Door te kijken of *BB-8* zich een bepaalde afstand links of rechts van het balkje bevond, kon een botsing met het balkje gedetecteerd worden. Als een andere software, die wel de detectie van rechthoekige objecten aankan, gebruikt zou worden, zou het hele balkje gedetecteerd kunnen worden. Hiermee zou preciezer kunnen worden gezien of het balkje en de *BB-8* botsen, wat het spel soepeler zal laten verlopen.

Tot slot zou voor een robot gekozen kunnen worden die meer diagnostische informatie kan terugsturen naar de bediening. De gebruikte *Sphero BB-8* was uiteindelijk enkel in staat een signaal te sturen wanneer een obstakel werd geraakt. Door deze weinige informatie was de besturing van de robot vrijwel geheel afhankelijk van *openCV*. Door voor een robot te kiezen die bijvoorbeeld zijn snelheid en coördinaten zou kunnen terug communiceren, zou het spel preciezer en responsiever kunnen worden. Dit omdat het gehele *openCV* gedeelte van de opstelling hierdoor overbodig zou kunnen worden. Er zouden minder berekeningen uitgevoerd hoeven worden, evenals de verbinding tussen beide systemen zou volledig opgeheven kunnen worden, wat de responsiviteit van het algehele systeem zou bevorderen.

Referenties

[1] N. Owen, G. N. Healy, C. E. Matthews, and D. W. Dunstan. Too much sitting: the population-health science of sedentary behavior. *Exercise and sport sciences reviews*, 38(3):105, 2010.

[2] D. Rajaratnam and M. Thielscher. Towards general game-playing robots: Models, architecture and game controller. In *Australasian Conference on Artificial Intelligence*, pages 271–276. Springer, 2013.