Beweegvloer

Interactieve & educatieve vloerprojecties voor de kinderdagopvang

Springlab
Iedereen in beweging!

Auteur: Timon Nap
Begeleider: Alexander Mulder

De beweegvloer
Interactieve & educatieve vloerprojecties voor de kinderdagopvang

Auteur: A.H. Nap
Studentnummer: 500697750
Telefoonnummer: 06-41886302
Plaats: Utrecht
Datum: Februari-Augustus 2017

Opleiding: Informatica / Game Development
Onderwijsinstelling: Hogeschool van Amsterdam
Begeleider: Alexander Mulder

Bedrijf: Springlab Bv.
Heidelberglaan 2
3584 CS, Utrecht
Tel. 06-11350423
Bedrijfsbegeleider tot eind April: Imara Speek
Bedrijfsbegeleider vanaf Mei: Salko Joost Kattenberg

Stageperiode: 20 februari 2017 – 20 augustus 2017

Inhoudsopgave

Samenvatting

De afstudeerstage heeft plaatsgevonden bij Springlab Bv. in Utrecht. Springlab is een innovatiebureau waar aan ideeën gewerkt wordt welke beweging moeten stimuleren. Tijdens mijn stage is de bedrijfsvoering van Springlab veranderd van het maken van prototypes voor verschillende probleemoplossingen naar een focus op bewegend leren. Springlab ziet het huidige onderwijs met veel stilzitten als een groot probleem en ziet de oplossing in bewegend leren. Leren in combinatie met bewegen door middel van technisch slimme oplossingen. Het idee dat Springlab hiervoor heeft bedacht is de beweegvloer, interactieve vloerprojecties waar jonge kinderen spelenderwijs mee kunnen leren.

Om te bewijzen dat de beweegvloer technisch mogelijk is wil Springlab intern onderzoeken of het maken van een prototype mogelijk is. Dit is een onderdeel van het groter geheel, het haalbaarheidsonderzoek betreffende de beweegvloer. Met behulp van een prototype kan Springlab de volgende fase van ontwikkeling ingaan. Het technische deel van de opdracht is het onderzoeken van technieken waarmee men spelers van de beweegvloer kan detecteren en volgen. Dit detecteren en volgen noemen we tracking. Doordat spelers gedetecteerd worden op de vloer wordt het mogelijk kinderen interactief, bewegend en spelenderwijs te laten leren. De probleemvraag luidt: *Welke mogelijke techniek voor tracking gaan we gebruiken voor de beweegvloer en hoe implementeren we deze mogelijke technieken in Unity3D?*

Het onderzoek is uitgevoerd door middel van het implementeren van computer vision oplossingen in prototypes. Ideeën hier voor zijn opgedaan door het onderzoeken van reeds bestaande voorbeelden, het ontrafelen van algoritmes en het interactief testen van mogelijke oplossingen. Hiermee zijn onder andere de randvoorwaarden vastgesteld en is er verder gegaan met gericht zoeken naar implementeerbare oplossingen. Per onderdeel binnen computer vision is bekeken wat een mogelijke oplossing, of een weg tot een oplossing was, en is er een implementatie gemaakt. Deze implementaties zijn getest op bruikbaarheid en uiteindelijk samengevoegd tot één oplossing. Gezien de beweegvloer games moet gaan aanbieden zijn de computer vision oplossingen uiteindelijk samengekomen binnen een game ontwikkeling omgeving genaamd Unity3D.

Dit onderzoek resulteerde daarmee in een aantal opties om de spelers van de beweegvloer te detecteren op videobeeld. Vanuit deze resultaten is een software prototype gemaakt om de detectie data bij te houden en de gedetecteerde objecten te volgen. Dit is overgezet binnen een game omgeving in Unity3D inclusief een prototype.

Aan de hand van de gemaakte prototypes is gebleken dat het detecteren van de spelers van de beweegvloer het beste gedaan kan worden door middel van het gebruik van een dieptesensor. Dit zorgt voor minimale afwijking in het bronbeeld. Een duidelijk bronbeeld is belangrijk voor de toegepaste algoritmes. Op het gebied van de volgingsalgoritmes is nog ruimte voor dooronderzoek. De opgeleverde implementatie toont potentie voor een speelbaar prototype, maar er is nog ontwikkelruimte en tijd nodig om dit te kunnen testen met daadwerkelijke spelers.

Inleiding

Dit verslag beschrijft de resultaten van mijn stage bij Springlab. Ik heb deze stage gelopen aan het einde van mijn vier jarige hbo-studie aan de Hogeschool van Amsterdam. Hier heb ik de studie Informatica Game Development gevolgd.

Ik heb gekozen voor de stage bij Springlab, omdat de opdracht mij aansprak. Binnen de opleiding hoor je vaak van medestudenten dat ze bij een game development bedrijf willen werken, maar ik vind het juist interessant om buiten dat gebied te zoeken.
Bij Springlab heeft men bewegend leren als hoofdthema en willen ze de beweegvloer op de markt brengen. De beweegvloer moet interactieve vloerprojecties gaan maken waarop kinderen interactief en spelenderwijs kunnen leren. De stageopdracht was om onderzoek te doen naar de technische mogelijkheden voor de beweegvloer. Dit sprak mij aan, omdat het een heel nieuw gebied voor mij was.

Aan het begin van mijn stage was de werkwijze van Springlab nog om tot een prototype te komen voor een product. Dit prototype werd dan gebruikt om het project van investeringen te voorzien waarna samen met een technische partner tot een uitwerking van een uiteindelijk product werd gewerkt. Na ongeveer twee maanden tijdens mijn stage ging Imara Speek weg bij Springlab. Imara was mijn begeleidster en tevens ook ontwikkelaar & programmeur bij Springlab. De reden hier voor was dat Springlab steeds minder richting snel prototypes ontwikkelen ging en juist meer op het idee bewegend leren wilde gaan focussen. Voor de prototypes worden dan ook al in het beginstadium technische partners gebruikt in plaats van deze intern te ontwikkelen. Hierdoor viel Imara haar functie deels weg en ging ze op zoek naar een ander bedrijf. Mijn stagebegeleider is vanaf toen Salko Joost Kattenberg geworden; game designer bij Springlab.

Gezien de verandering van bedrijfsvoering binnen Springlab heb ik mijn onderzoek ook voornamelijk binnen de grenzen gehouden van wat ik zelf uit kon voeren. Er is uiteindelijk geen volledig prototype van de beweegvloer uitgekomen, ook mede door de complexiteit van het onderwerp en de beperkte tijd. Wel heb ik bruikbare resultaten behaalt voor de tracking en detectie van spelers. Persoonlijk heb ik hierbij enorm veel geleerd over computer vision en daarbij over object detectie en tracking.

In dit verslag wordt mijn werk bij Springlab beschreven, welke (softwarematige) prototypes ik heb ontwikkeld en onderzocht en wat de resultaten hiervan zijn.

Qua opbouw beschrijf ik in dit verslag het bedrijf Springlab, daarna het onderwerp computer vision en vervolgens het onderzoek doen naar bestaande methodes binnen tracking. Vanuit dit vooronderzoek ben ik opbouwend prototypes gaan maken van computer vision technieken om uiteindelijk tot een tracking oplossing te komen.
Als laatste is er nog een korte aanbeveling over wat ik zou doen om tot een compleet prototype te komen voor de beweegvloer.

1 Context van de opdracht

1.1 Het bedrijf

1.1.1 Algemeen

Mijn afstudeerstage heb ik gelopen bij Springlab B.V. te Utrecht.
Springlab is een klein bedrijfje gehuisvest binnen de Universiteit Utrecht.
Bij Springlab werkt men aan innovatieve oplossingen om beweging te stimuleren. De website is www.springlab.nl

Springlab bestaat uit een klein team met 6 werknemers en nog 2 stagiaires (waar ik er eentje van ben). Een gedeelte werkt parttime bij Springlab op freelancebasis.

Figuur 1 The Office; gefaciliteerd door de Universiteit Utrecht

1.1.2 Historie

Springlab is opgericht in 2013 door Jan-Paul de Beer en Thom Rutten met als doel het stimuleren van alledaagse beweging. Als innovatiebureau komt Springlab met slimme, innovatieve oplossingen om meer bewegen voor elkaar te krijgen.
Het eerste project was de *"active aging challenge"* waarbij het doel was senioren meer in beweging te krijgen naarmate ze ouder worden. Later werd ook gewerkt aan de *"active aging challenge"*, waar kantoor werk actiever gemaakt moest worden. Hiervoor is een gezonde staan tafel ontworpen, de Upyi.

Eén van de meest succesvolle projecten die Springlab afgerond heeft is de zogeheten Tovertafel. *"De Tovertafel is een speels product dat ouderen in de midden- tot late stadia van dementie prikkelt tot fysieke activiteit en sociale interactie met minimale begeleiding!".*

1.1.3 Bedrijfscultuur

Springlab is een klein bedrijf met een hele soepele sfeer. Er zijn bijvoorbeeld geen vaste werktijden; het enige vaste is een teammeeting op de woensdagochtend waarin ieders maanddoelen besproken worden. Ook is er veel openheid wat betreft inbreng in de gang van zaken. Niemand staat boven een ander wat zorgt voor veel feedback van collega's op je werk en de gang van zaken wordt vaak met iedereen besproken.

1.1.4 Visie en missie

"Het is het tijdperk van het stilzitten. We hebben ons leven zo ontworpen dat beweging geen noodzaak meer is. De technologie doet het zware werk waardoor wij steeds passiever worden. Dat lijkt misschien lekker, maar het is vooral ongezond en zelfs gevaarlijk. En daarom zijn wij er. Wij willen deze trend tegengaan en daarvoor is innovatie en ondernemerschap nodig. Onze missie is om samen met jou lichamelijke beweging weer terug te brengen in het dagelijks leven door het realiseren van duurzame innovaties." –springlab.nl

1.1.5 Projecten

Hier een overzicht van een aantal projecten waar Springlab aan heeft gewerkt of welke nog steeds actief zijn binnen Springlab. Dit geeft een mooi beeld waar Springlab zich allemaal mee bezig houdt. Quotes van springlab.nl

Rolstoelgaming

"Stimuleert beweging en verbetert de zelfredzaamheid van kinderen in een rolstoel op een leuke en uitdagende manier."

FietsFlo

"Jouw persoonlijke snelheidsadvies om vaker groen licht te halen. Dankzij de FietsFlo hoeft een fietser aanzienlijk minder te wachten voor een verkeerslicht. FietsFlo maakt fietsen leuker."

Active Cues Tovertafel

"De Tovertafel is een speels product dat ouderen in de midden- tot late stadia van dementie prikkelt tot fysieke activiteit en sociale interactie met minimale begeleiding!"

Moon Trees

"Een real life game voor in klimbossen speelbaar voor alle leeftijden."

Way2Go

Om drukte in de spits te voorkomen wil gemeente Utrecht ouders en kinderen stimuleren naar school te lopen in plaats van deze te brengen met de auto. Springlab ontwikkelt een game in de vorm van een app wat dit stimuleert.

Figuur 2 Monster Rebellion. De GPS game voor Way2Go waar ik ook aan gewerkt heb.

1.2 De opdracht

1.2.1 Aanleiding

Springlab wil doormiddel van een haalbaarheidsonderzoek de mogelijkheid verkennen om bewegend leren in de kinderopvang (leeftijd 2-4 jaar) te stimuleren doormiddel van interactieve projecties op de vloer. De werktitel van dit project is 'De Beweegvloer'. De beweegvloer moet het mogelijk maken om kinderen door middel van interactieve projecties op een speelse manier de basis van de *21st century skills* en de huidige leerdoelen van de kinderdagopvang bij te brengen. Er zijn wel al systemen die interactieve beelden op de vloer projecteren, maar deze projecties zijn niet gericht op leren, bewegen én ontwikkelen.

Springlab bekijkt of de interactieve projectietechnologie passend gemaakt kan worden voor toepassing in de kinderopvang. In dit project werkt Springlab samen met Partou kinderopvang. Daarnaast is Radboud Universiteit actief betrokken met promovendus Eva van de Sande en de twee hoogleraren Ludo Verhoeven en Eliane Zegers. Zij fungeren als kennispartners en gaan in een latere fase onderzoek uitvoeren naar het effect van de beweegvloer. Inmiddels is Spelenderwijs betrokken, en hebben gemeente Utrecht en Eindhoven geïnvesteerd in een eerste oriënterende fase.

1.2.2 Omschrijving van de opdracht

Er zijn diverse projectiesystemen op de markt (bijv. LumoPlay, Magixsense) die interactieve beelden kunnen projecteren. Deze systemen bieden nog geen interactieve projecties aan gericht op leren bij jonge kinderen. De bestaande systeem zijn ook niet geschikt voor beweegvloer projecties. Ze detecteren slecht, kunnen slechts 1-2 personen tegelijkertijd detecteren of bieden geen platform aan voor gameontwikkeling.
In het haalbaarheidsonderzoek wil Springlab kijken vooral kijken naar hoe digitale interactie bij 2-4-jarigen bijdraagt aan leren. Daarnaast toetst Springlab de geschiktheid van beschikbare detectietechnologie.

Springlab wil interactieve projecties (games) ontwikkelen die op de vloer geprojecteerd worden. Door de interactie al bewegend aan te gaan leren kinderen. Kinderen gaan ontdekken, verbinden, creëren, werken samen, oefenen bewegingen etc. Voordat Springlab deze interacties/ games kan ontwerpen, moet bekeken worden welke technologie geschikt is.

Intern is al kort vooronderzoek gedaan naar manieren om mensen te detecteren. Een losse infrarood sensor is bekeken als optie maar dit is afgevallen. Een infrarood detectiesysteem vereist compleet nieuwe algoritmes om mensvormen te volgen. Er zijn geen bekende goed ondersteunde open source *libraries* hiervoor. Een 3D camera zoals Kinect is naar voren gekomen als mogelijke oplossing.

Door het afvallen van infrarood oplossingen is mijn haalbaarheidsonderzoek in een beginfase al beperkt tot (3D) camera's. 3D camera's zoals de Kinect hebben wel een breed ondersteunde functionaliteit. Daarom is de opdracht voor mij bij Springlab om dit scenario te onderzoeken. Allereerst het tracken uit videobeelden zelf zodat dit later eventueel toegepast kan worden op een prototype samen met bijvoorbeeld Kinect.

1.2.3 Probleemstelling

Kinderen komen steeds vaker met een leerachterstand binnen op de basisschool en ook te weinig beweging is een groeiend probleem bij jonge kinderen. Bewegend leren is de perfecte oplossing om dit tegen te gaan. Daarom wil Springlab de Beweegvloer uitbrengen.

1.2.4 Doelstellingen

Om tot een uiteindelijk verkoopbaar product te komen moet Springlab veel onderzoek verrichten. Een deel van het onderzoek is de technische haalbaarheid: Welke technieken zijn een optie voor de beweegvloer?
Dat is wat dit onderzoek behandelt; Het onderzoek gaat dieper in op manieren om uit camera/sensor beeld tracking data te halen. Vervolgens worden deze technieken gebruikt in Unity3D om te koppelen aan de uiteindelijk educatieve games.

De hoofdvraag luidt als volgt:
Welke mogelijke techniek voor tracking gaan we gebruiken voor de beweegvloer en hoe implementeren we deze technieken in Unity3D?

Om deze hoofdvraag te kunnen beantwoorden behandelt deze scriptie de volgende deelvragen:

- *Wat verstaan we onder Computer Vision?*
- *Wat is OpenCV?*
- *Manieren om object te detecteren in OpenCV.*
- *Unity3D & OpenCV; hoe combineren we dit?*
- *Welke algoritmes zijn nodig om gedetecteerde objecten te volgen?*

1.2.5 Technieken

De volgende technieken zijn gebruikt tijdens het onderzoek:

- OpenCV Computer Vision Library i.c.m. C++
- OpenCV For Unity3D in C#
- Unity3D

Het onderzoek begon intern met het vergaren van bestaande data (Interne Desk Research). Mijn eerste stagebegeleidster (Imara Speek) had al kort onderzoek gedaan naar mogelijke technieken voor de beweegvloer en had al enige aanknopingspunten. Uit dit vooronderzoek ben ik op het spoor gekomen van OpenCV; "*an open source computer vision and machine learning software library*" (OpenCV team, 2017). Aan de hand van de interne eisen voor de beweegvloer is duidelijk dat de games met Unity3D ontwikkeld moeten worden. Het uitgangspunt was daardoor: OpenCV + Unity3D. Vanaf hier kon het onderzoek beginnen. Door middel van externe deskresearch is er data verzameld over OpenCV. Aan de hand van deze data is er gestart met de ontwikkeling van de eerste prototypes. Bij het ontdekken van nieuwe mogelijkheden van OpenCV zijn er over de tijd heen meerdere iteraties ontstaan met steeds betere functionaliteiten. Deze prototypes zijn getest op bruikbaarheid voor de beweegvloer. Aan de hand van de eisen van de beweegvloer zijn er keuzes gemaakt om bepaalde technieken wel en bepaalde technieken niet te kiezen.

Uiteindelijk zijn er aan functionaliteiten binnen OpenCV de volgende functies/algoritmes toegepast en deze worden ook uitgebreid behandelt binnen het onderzoek:

- Image & Video Processing
- SimpleBlobTracker
- FindContours Methode
- Kalman Filter
- Hungarian Algoritme

Voor elke nieuwe methode is een werkende demo ontwikkelt om de functionaliteit te testen en te bepalen of het bruikbaar is voor de beweegvloer.

1.2.6 Oplevering

Van mij wordt verwacht dat ik aan het einde van mijn stage een duidelijk beeld heb gecreëerd welke methodes binnen computer vision toe zijn te passen op een mogelijk prototype i.c.m. Unity3D voor de volgende ontwikkelfase van de beweegvloer. Welke technieken bruikbaar zijn en in welke programmeertaal deze zijn geschreven.

2 Deelvragen

2.1 Computer Vision

2.1.1 Wat is computer vision?

Computer vision is het analyseren van beelden en video door computers en uit deze beelden informatie halen. Voor ons mensen is het verwerken van beelden die we zien vanzelfsprekend. Geef een kind een foto met een aantal dieren erop en vraag het kind om het aantal dieren te tellen; je zult een antwoord krijgen. Kijk naar een foto van bijvoorbeeld familie en noem alle mensen die je ziet bij naam, ook dat is makkelijk te doen. Dingen zoals kleur, diepte, transparantie etc. zijn onderwerpen die voor ons mensen vanzelfsprekend zijn. Voor een computer zijn deze handelingen niet vanzelfsprekend (Szeliski, 2010, p. 3). We moeten de computer deze handelingen aanleren. Waarom is het zo lastig voor computers om beelden te analyseren? Dit komt, omdat de computer vanuit één beeld of afbeelding informatie moet halen welke (voor de computer) onbekend is. Wij mensen hebben een zeer complex brein, met allerlei verschillende functies alleen al om te kunnen zien, laat staan het ontcijferen van beeld. Voor de computer is een afbeelding slechts een reeks nummers waar hij een algoritme op los kan laten (Dawson-Howe, 2014, p. 1&2). Waar de mens zonder moeite een plaatje beschrijft, wat erin afspeelt en wat we herkennen, hebben computer vision algoritmes hier veel moeite mee en geven vaak foute informatie terug. Nu komt dit allemaal erg negatief over, maar gelukkig is computer vision niet onbruikbaar. Integendeel zelfs, het produceert zeer bruikbare data en wordt ook breed toegepast in de industrie (Szeliski, 2010, p. 5). Bijvoorbeeld bij de kassa waar producten gescand worden, bij verkeerssystemen welke auto's tellen, bij fruitbedrijven waar fruit op grootte en zelfs al kwaliteit gesorteerd wordt en ook recent bij de zelfrijdende auto om de omgeving te analyseren en aan de hand daarvan keuzes te maken. (Szeliski, 2010, pp. 3-6), (Dawson-Howe, 2014, pp. 1-3)

2.1.2 Geschiedenis

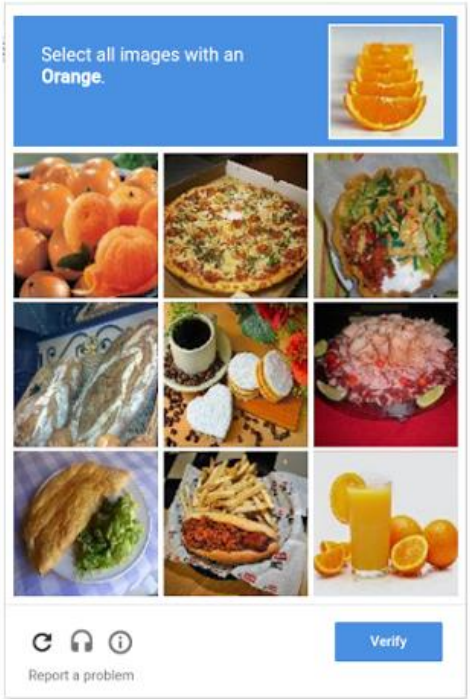
Computer Vision is begonnen in de jaren 70. Het werd toen gezien als het deel visuele waarneming voor de artificiële intelligentie, in de robot industrie. Men was hier mee bezig en dacht dat het visuele input gedeelte zeker niet het lastigste obstakel zou zijn op de weg naar slimme robots. Een schijnbaar bekende uitspraak uit 1966 geeft dit mooi weer waar een professor zijn studenten het volgende vroeg: *"spend the summer linking a camera to a computer and get the computer to describe what it saw".* Nu weten we dat het allemaal wat ingewikkelder is dan dat (Szeliski, 2010, p. 11).

Digital Image Processing, het verwerken van beeld, bestond al binnen de informatica. Waar computer vision in verschilde was dat het de volledige 3D scene uit een beeld wilde halen en dit gebruiken om de wereld in dat beeld te begrijpen. (Szeliski, 2010, pp. 11-13)

In de jaren tachtig ging men dieper in op het gebied van wiskundige oplossingen om beelden te verwerken. Er werd onderzoek gedaan naar betere *edge detection* (het detecteren van hoeken en randen in beeld) en ook het vinden van contouren werd onderzocht. Deze gebieden bleven onderzocht worden in de jaren negentig. De ontwikkeling van computer vision bleef door gaan. Onder andere werd er ondertussen gezichtsherkenning uitgevoerd door overheden. (Leone, 2017). Ook de tracking werd beter, zo werd er gebruikt gemaakt van *active contours* waar in beeld naar contouren gezocht werd om deze te gebruiken bij tracking. (Szeliski, 2010, p. 17).

Computer vision kwam ook regelmatiger in aanraking met computer graphics, bij het modelleren van 3D computerbeelden aan de hand van afbeeldingen bijvoorbeeld. Ook werden animaties gecreëerd aan de hand van echte beelden (Szeliski, 2010, p. 17).

Afgelopen decennium is er vooral veel ontwikkeling geweest op het gebied van *learning*. Omdat er meer rekenkracht beschikbaar is en er veel data opgeslagen wordt kunnen we computersystemen beelden laten herkennen aan de hand van opgeslagen data. Denk bijvoorbeeld aan een *captcha* waar je alle afbeeldingen met een verkeersbord moet markeren; Deze data wordt gebruikt om systemen te verbeteren en 'slimmer' te maken. (Google, 2017)



Figuur 3 Google's ReCaptcha, de data wordt weer gebruikt om machines te laten leren. (Google, 2017)

2.1.3 Object detectie/tracking binnen computer vision

Het detecteren van objecten is makkelijker dan het herkennen van objecten. Van alle visuele taken die een computer kan doen is het analyseren van een beeld en het herkennen van alle objecten hierbinnen één van de lastigste problemen om op te lossen.
Herkennen is lastig voor een computer gezien we enorm veel objecten kennen in de wereld en daar komt nog bij dat een enkele opname van een object per afbeelding weer heel anders kan zijn. Er kan iets voor het object in de weg staan of door lichtinval is de kleur misschien heel anders. Ook is niet elk object simpel om te identificeren. Denk bijvoorbeeld aan een hond; vaststellen dat het een hond is, is mogelijk met computer vision (Bergevin, 2010), maar om daarbinnen dan ook nog het goede ras vast te stellen dat is zeer lastig.
Object detectie en herkenning is een breed onderwerp, waar je per sub onderwerp diep op de theorie in kan gaan. Objectherkenning valt buiten de scope van mijn onderzoek. Mijn onderzoek gaat in op het detecteren van een object en het tracken daarvan. Voor het detecteren van objecten wordt in dit onderzoek OpenCV toegepast. OpenCV wordt behandelt in het aankomende hoofdstuk.

2.2 OpenCV

2.2.1 Wat is OpenCV

Voor het toepassen van computer vision maak ik gebruik van de open source computer vision *library* OpenCV. Dit is een pakket met computer vision code, algoritmes & methoden welke je kan gebruiken binnen een eigen project. OpenCV is begonnen als onderzoeksproject binnen Intel in 1999 (Willow Garage, 2015). Het is geschreven in C & C++ en draait onder Linux, Windows & Mac OS X. Ook is er actieve development van versies voor o.a. de programmeertalen Java & Python. Eén van de doelen van OpenCV is het helpen met het snel opstellen van relatief geavanceerde computer vision applicaties. Gezien OpenCV honderden functies bevat welke een breed scala aan computer vision onderwerpen aangaan biedt het veel opties om applicaties binnen computer vision tot stand te brengen (Kaehler, 2008). OpenCV wordt breed toegepast en heeft ondertussen meer dan 14 miljoen downloads (OpenCV team, 2017); de meest recente, grote release was versie 3.2 in december 2016.

2.2.2 Waarom zou je OpenCV kiezen?

Via mijn eerste stagebegeleidster, Imara Speek, kwam ik uit bij OpenCV. Zelf heb ik nog nooit met computer vision gewerkt dus voor mij was het een heel nieuwe gebied. Ik kwam er al snel achter dat er enorm veel te vinden is over OpenCV. Gezien OpenCV open source is en gratis wordt het veel gebruikt. Er is kort naar alternatieven gekeken, maar niets kwam in de buurt van de toepasbaarheid en grootte van OpenCV. De vele online *tutorials* en de vele *ports* naar andere omgevingen en talen maakt OpenCV zeer aantrekkelijk.
Een groot bekend alternatief om computer vision berekeningen te doen is Matlab. Matlab is geschreven in Java en beschikt over een computer vision *toolbox*. Het is mogelijk code geschreven in Matlab te exporteren naar C/C++ code welke je kan toepassen binnen je eigen project. Met behulp van een geschreven overzicht van (Thakkar, 2012) heb ik een overzicht gemaakt tussen de verschillen van OpenCV en Matlab.

	OpenCV	**Matlab**

Snelheid/efficiëntie	Aanroepen van functies geschreven in C/C++, zit directer op de machinetaal.	Je schrijft Matlab functies, welke naar Java worden vertaald. Deze Java code wordt vertaald naar machinetaal.
Geheugengebruik	Laag geheugengebruik, rond de honderd mb voor een video programma.	Meer dan één gigabyte voor een videoprogramma.
Kosten	OpenCV is gratis!	€1250,- voor één licentie van de computer vision toolbox + €2000,- voor Matlab zelf.
Toepasbaarheid	OpenCV zelf draait op Linux, OSX en Windows met veel beschikbare ports naar andere platformen.	Matlab draait op Linux, OSX en Windows met de mogelijkheid C code te exporteren om dit zelf te porten naar andere platformen.

Bron voor de tabel: (Thakkar, 2012)

Het was niet lastig om de keuze te maken voor OpenCV. Online lees je veel (o.a. op Stackoverflow, een grote vraag & antwoord site voor informatica gerelateerde vragen) dat Matlab beperkt is binnen de toolbox, lastig te porten naar andere platformen en omgevingen en daarnaast is het ook nog eens een stuk langzamer dan OpenCV. Het feit dat OpenCV gratis is en Matlab om te beginnen al 2000 euro kost zonder computer vision toolbox maakte de keuze definitief.
Een ander alternatief wat ik gevonden heb was AForge. Ik heb wat interessante video's gezien met tracking binnen AForge, maar na verder onderzoek bleek dat de laatste versie in 2013 uitgekomen is (AForge.NET, 2013) en de online informatie die beschikbaar is over AForge valt in het niet met de vele informatie over OpenCV wat onder andere te zien is aan de programmadocumentatie (AForge.Net, 2013). Ook op Stackoverflow zijn voor AForge 1627 resultaten en voor OpenCV ruim 94 duizend.

2.2.3 Tracking & OpenCV

Om te beginnen is de term "object tracking" onderzocht binnen OpenCV. Ik kwam hierover een interessante en recente blogpost tegen van Satya Mallick, een onderzoeker op het gebied van computer vision welke zijn PhD. gehaald heeft aan de University of California, San Diego. Mallick beschrijft hier een aantal tracker functies binnen OpenCV en het verschil tussen tracking en detectie komt aan bod. Om tracking te kunnen doen moet je eerst een object hebben om te tracken; hier voer je detectie voor uit. Tracking is sneller dan detectie; wanneer je een object aan het tracken bent dan heeft de *tracker* al informatie over het object. Waar het zicht bevindt, visuele kenmerken etc. Terwijl wanneer je detectie uitvoert je met niets begint. Tracking helpt ook wanneer het detecteren van een object faalt. Denk aan het uit beeld gaan van je te detecteren object, als het daarvoor al gedetecteerd was dan kan de tracker met behulp van algoritmes voorspellen wat het object doet totdat het weer terug in beeld komt. Ook behoudt tracking de identiteit van de te tracken objecten. Wanneer je meerdere objecten detecteert kan de volgorde per detectie veranderen, bij

tracking houdt je bij elk object de eigen identiteit vast. (Mallick, Object Tracking using OpenCV, 2017) In Mallick zijn blogpost gaat hij nog verder in op een aantal tracking methodes welke los te vinden zijn in OpenCV. Deze zijn voor dit onderzoek nu (nog) niet interessant, omdat de detectie nog niet geïmplementeerd is. Je geeft bij Mallick zijn implementatie handmatig in een video aan wat je gaat tracken, terwijl voor de beweegvloer automatische detectie vereist is. Deze bevindingen hebben de scope van mijn haalbaarheidsonderzoek verlegt naar object detectie en kunnen gezien worden als de eerste bevindingen van mijn onderzoek. Detectie is dan ook waar het onderzoek mee verder gaat.

2.3 Implementatie van tracking/detectie

2.3.1 Vooronderzoek implementatie

Het is mogelijk vele kanten op te gaan met computer vision. Het is maar net wat je uiteindelijk wilt bereiken met het toepassen van computer vision binnen je applicatie. Szeliski zegt in zijn boek daarom ook dat je niet de eerste de beste computer vision techniek moet pakken waar je bekend mee bent of waar je van gehoord hebt. Ga vanuit het probleem dat je wilt oplossen naar geschikte computer vision technieken kijken. Implementeer een techniek, bepaal of je deze kunt gebruiken door te testen en ga anders voor een alternatieve techniek. Het kan ook geen kwaad verschillende technieken te gebruiken, vaak is een combinatie van verschillende technieken zelfs de beste oplossing. (Szeliski, 2010, p. 9) Voor dit onderzoek is bekend dat er object detectie uitgevoerd moet worden alvorens er tracking plaats kan gaan vinden.

Alvorens een keuze te maken voor een computer vision toepassing kun je jezelf de volgende vragen stellen:

- Welke data heb je nodig?
- Wat wil je met de data doen?
- Hoe wil je aan de data komen?

Dit zijn belangrijke vragen welke je moet beantwoorden voordat je een keuze maakt voor een bepaalde toepassing en ook voordat je aan de slag gaat.
Voor mijn opdracht willen we spelers van de beweegvloer kunnen volgen. We weten aan de hand van het korte onderzoek naar tracking binnen OpenCV dat we eerst object detectie uit moeten voeren. We willen de spelers kunnen volgen, omdat de beweegvloer interactief moet zijn. Als bijvoorbeeld een speler over een geprojecteerd object loopt moet dit te detecteren zijn en moet er met het geprojecteerde beeld iets gebeuren (interactie). Hiervoor hebben we de posities van elke speler op de vloer nodig. Om spelers hun positie op te halen en te volgen moeten we ze eerst detecteren. Het speelbare veld waarin we de spelers willen detecteren is relatief klein, denk aan een vlak van ongeveer 3 bij 3 meter. Ook willen we meer dan één speler tegelijk op de vloer kunnen volgen. Meer dan zes wordt lastig gezien het gelimiteerde speelveld, maar het moet mogelijk zijn om met bijvoorbeeld 2 spelers tegelijk interactie te hebben op dezelfde beweegvloer. Als we de positie van elke speler hebben willen we hier iets mee doen in Unity3D. Het spel of de applicatie die draait (en geprojecteerd wordt op de vloer) moet aan de hand van de doorgegeven posities interactie teweegbrengen. Deze positie data willen we vergaren door middel van detectie en daarna tracking. OpenCV heeft verschillende opties voor object detectie welke behandelt worden in dit onderzoek.

Bestaande mogelijkheden verkennen
Een mogelijke optie om het tracken van de spelers makkelijker te maken voor de computer is het toevoegen van een accessoire welke de te tracken spelers met zich dragen of ophebben. Een voorbeeld hier van is de PlayStation Move controller (Figuur 4) welke een LED gekleurde bol bovenop heeft. De PlayStation Eye Camera volgt deze bol op kleur (niet op infrarood zoals de Wii dat doet (Lowe, 2010)) en ook, omdat de gekleurde bol afsteekt van de omgeving is deze makkelijker te tracken. De camera kan daarnaast d.m.v. de grootte van de bol zien hoe ver de speler verwijderd is van de camera. (Shuman, 2010)

Figuur 4 De PlayStation Move controller. (Sony Computer Entertainment Inc., 2010)

Naast het tracken van de gekleurde bol doet de PlayStation Eye nog meer:
"The Eye is capable of using edge detection, facial recognition, and basic motion tracking to map a room." (Lowe, 2010)
Hieruit kan je opmaken dat je door meer dan één tracking methode toe te passen je tot een betere tracking oplossing kan komen. Het is dus niet zo dat je maar één methode gebruikt en dan klaar bent. Het is mogelijk om meerdere algoritmes en/of detectie methodes te gebruiken, de data hiervan te combineren en zo tot een snelle en precieze tracker te komen.

Een wat gerichter voorbeeld voor de beweegvloer is het bestaande SMALLab Learning. Dit is een compleet product voor interactief leren. Het gebruikt een beamer, heeft tracking en heeft als doel het verbeteren van het leerproces op scholen. Nu is SMALLab Learning voornamelijk gericht op het basisonderwijs en het voortgezet onderwijs (SMALLab Learning, 2017). De beweegvloer wil de focus juist leggen op de kinderopvang. Het is interessant om te kijken hoe SMALLab de tracking aangepakt heeft. Hiervoor heb ik o.a. de video van ze bekeken op hun website. Allereerst valt meteen op dat ze gebruik maken van een tracking hulpmiddel. Aanvankelijk dacht ik dat het tracken bij SMALLab op kleur ging, maar verderop komt het tracking systeem in beeld.

SMALLab maakt gebruik van OptiTrack camera's welke markers volgen. De markers zijn makkelijk te zien door de camera door een specifieke setup (SMALLab, 2011). Waarschijnlijk reflecteren de markers heel goed infrarood waardoor ze makkelijk te tracken zijn.

Figuur 5 SMALLab Learning tracking oplossing

Randvoorwaarde naar aanleiding van het vooronderzoek
Uit het vooronderzoek blijkt dat er genoeg oplossingen beschikbaar zijn welke gebruik maken van een tracking hulpmiddel zoals de PlayStation Eye en de oplossing van SMALLab. De mogelijkheden om met een tracking hulpmiddel de beweegvloer te maken zijn intern binnen Springlab besproken. We hebben gekeken naar de bestaande oplossingen en besproken hoe dit zou kunnen gaan werken met de beweegvloer. De conclusie is dat Springlab niet met een tracking hulpmiddel de beweegvloer maken wilt. Het grootste probleem is het verkopen van de beweegvloer met een hulpmiddel/accessoire. Het vereist extra opzettijd wat de beweegvloer een stuk minder toegankelijk maakt. Dit zorgt er ook voor dat de beweegvloer lastiger te verkopen is. Kinderopvangcentra, welke de initiële klanten vormen voor de beweegvloer, hebben een druk programma. Als je aankomt met een nieuw product wat onnodige opzettijd heeft dan is dit gelijk een drempel om tot de aanschaf te komen. Wat we ook niet willen is dat personeel onnodig lang bezig is met het opzetten van de beweegvloer, dit gaat ten koste van de kwaliteit van de kinderopvang. Daarnaast willen we niet met een product komen dat extra componenten heeft welke stuk kunnen gaan, opgeladen moeten worden etc. Idealiter hangen we een kast op met een projector en een tracking oplossing welke aan een computer zit waar de spellen op draaien en zijn we klaar.
Springlab wil als innovatiebureau ook niet komen met een ouderwetse, bestaande oplossing.

Bestaande mogelijkheden binnen randvoorwaarden
Met nieuwe informatie, de randvoorwaarden, ben ik verder gegaan met het bekijken van bestaande oplossingen. Via Jan Paul de Beer, chief bij Springlab, ben ik in contact gekomen met Ronald Poppe. Ronald is onderzoeker en heeft meegewerkt aan een project genaamd: *Interactive Tag Playground/Tikkertje 2.0*. Jan Paul had van dit project gehoord en heeft contact opgenomen met de onderzoekers die hier bij betrokken waren. Deze zijn op 2 mei langs gekomen op kantoor en hebben uitgebreid verteld over Tikkertje 2.0

Op donderdag 4 mei 2017 heb ik een informerend gesprek gehad met Ronald over de technologie achter het Tikkertje 2.0 project. Het maakt gebruik van OpenCV + Unity3D en interactieve projecties op de vloer.
Wat ze gedaan hebben bij Tikkertje 2.0 is gebruik maken van vier Microsoft Kinect sensoren (Universiteit Twente, 2015) welke een diepte beeld kunnen teruggeven. Hierdoor heb je geen last van kleur afwijkingen welke de tracking kunnen verstoren. Aangezien het speelveld een stuk groter is hebben ze vier Kinect sensoren gebruikt. Op deze manier dekken ze een groter oppervlak. Voor detectie/tracking is gebruik gemaakt van een algoritme welke op het dieptebeeld kijkt naar hoofden + schouders en dit markeert als een speler (*mexican hat wavelet*). (Poppe, 2017)

Figuur 6 Interactive Tag Playground/Tikkertje 2.0 (Universiteit Twente, 2015)

Conclusie vooronderzoek

Er zijn dus veel manieren om een mogelijke tracking oplossing te maken voor de beweegvloer. Om het tracken makkelijker te maken is het mogelijk een accessoire te gebruiken wat elk te tracken object (in ons geval elke speler van de beweegvloer) beet/om heeft. Gezien onze doelgroep (kinderopvangcentra) valt deze oplossing af. Het maakt het aan de man brengen van de beweegvloer lastiger en zorgt ook voor meer onderhoud en verantwoordelijkheid (je levert immers meer spullen mee).
Na verder onderzoek is gebleken dat het gebruik van bijvoorbeeld een diepte sensor zoals in de Microsoft Kinect te vinden is een goed alternatief kan zijn. Bijkomend positief punt hiervan is dat deze sensoren ook werken in slecht belichte of juist overbelichte omgevingen; omgevingslicht is immers iets wat voor elk kinderopvangcentrum weer anders is.

Figuur 7 Kinect diepte beeld. (Davison, 2012)

2.3.2 Image & Video Processing

OpenCV is iets waar ik nog niet eerder mee gewerkt heb. Het is een nieuwe library voor mij en ook met C++ heb ik niet heel veel ervaring. Het helpt dat OpenCV redelijk *high level* geschreven is. Dit houdt in dat je voornamelijk methodes aan kan roepen zonder dat je daadwerkelijk helemaal moet weten hoe het precies werkt vanbinnen.
Om bekend te raken met OpenCV en de werkwijze daarvan heb ik veel geëxperimenteerd met basisfuncties. Hieronder valt image processing & video processing. De reden dat hier mee begonnen is, is dat het redelijk essentieel is voor het maken van een tracking oplossing en het is één van de meest basis dingen die je binnen OpenCV kan doen. Wat ik gedaan heb is wat tutorials gevolgd om image processing onder de knie te krijgen.

Image processing is belangrijk, omdat wanneer je tot object tracking wilt komen je namelijk een bruikbaar beeld nodig voor de computer om te verwerken. Bij bijvoorbeeld blob tracking wordt gekeken naar een groep pixels die aan bepaalde waarde voldoen (bijvoorbeeld zwart en de groep pixels moet rond zijn) (Mallick, Blob Detection Using OpenCV, 2015). Als je de computer een live camera beeld voedt in RGB dan kan hij hier bar weinig mee, dit omdat praktisch elke pixel wel anders is en er geen groepen pixels gevonden kunnen worden. Om deze reden pas je image processing toe op het beeld waar je computer vision op los wilt laten. Je filtert je bronbeeld om dit te kunnen gebruiken bij bijvoorbeeld detectie algoritmes. Het volgende figuur is een voorbeeld van image processing op kleur uit mijn onderzoek.

Figuur 8 Image Processing, filteren op de kleur geel.

De meest voorkomende manieren van filteren zijn op kleur, diepte of beweging. Er zijn verschillende mogelijkheden om dit te doen. Binnen dit onderzoek is een testopstelling gebruikt met webcam camerabeeld en vooraf opgenomen beelden.

Filteren op kleur

Normaal beeld komt binnen als RGB. Rood, groen & blauw. Dit beeld is voor ons mensen goed te interpreteren, maar voor computer vision lastig te verwerken. Je wilt een zo simpel mogelijk beeld creëren waar alleen het essentiële (het te tracken) zichtbaar is. Op die manier kan je makkelijker computer vision algoritmes toepassen; sterker nog: de methodes

die ik gebruik vragen om een binaire afbeelding, wat inhoudt dat de afbeelding alleen witte en zwarte pixels heeft. Een mogelijkheid hiervoor is het RGB-beeld filteren op kleur. Stel: je wilt een gele bal tracken, dan wil je dat alleen de gele bal zichtbaar is in het beeld. Alles wat niet de gele bal is doet er namelijk niet toe. Een veelgebruikte manier om dit voor elkaar te krijgen is het beeld converteren naar HSV en hier op bepaalde HSV-waardes te filteren.
HSV staat voor *Hue, Saturation & Value*.
Hue is de kleur, dit is dus een enkele waarde voor de kleur in plaats van 3 bij RGB.

Figuur 9 De Hue schaal in graden. In OpenCV is het een waarde tussen de 0-180. (Wikimedia)

De volgende waarde is Saturation. Dit is hoe 'vol' de kleur is. Een hele diepe kleur heeft een hoge saturation en een hele fletse kleur heeft een lage saturation. De laatste waarde is de value. Deze geeft aan hoe fel de kleur is. Een felle kleur heeft een hoge value een niet felle kleur heeft een lage value. Deze waardes worden in OpenCV opgeslagen als 3 *unsigned* 8 bits (Solderspot, 2014). Dit houdt in dat ze van 0-255 gaan. Aangezien Hue als 0-360 graden aangegeven wordt en je maar plek hebt voor 256 waardes loopt de Hue van 0-180 (de helft van de waardes 0-360). Het voordeel van HSV gebruiken is dat je één waarde hebt voor de kleur van een pixel en dat je daarnaast nog 2 waardes hebt waarmee je de intensiteit en de volheid van de kleur van de pixels aan kan geven. Als je een bepaalde reeks van HSV-waardes geeft waar je te detecteren object binnen valt is dit makkelijker te filteren dan wanneer je een kleur probeert vast te liggen met alleen RGB-waardes. Dit filteren van waardes buiten een gedefinieerde reeks waarden gebeurd in OpenCV met de inRange methode. Wat deze functie doet is voor elke pixel bepalen of hij binnen de opgegeven reeks valt. Valt hij ertussen dan wordt de pixel wit (255) en valt hij niet tussen de gegeven reeks dan wordt de pixels zwart (0). Op deze manier krijg je alleen te zien wat jij aangeeft met bepaalde HSV-waardes. (Solderspot, 2014)

Implementatie kleurfilter op video
De bovengenoemde filteringsmethode (op kleur) is toegepast op het beeld van mijn webcam in combinatie met een geel balletje. Dit gele balletje zou makkelijk te detecteren moeten zijn, gezien zijn afwijkende kleur. De implementatie volgt op de volgende pagina. De code opgesteld aan de hand van de tutorial van Solderspot, 2014

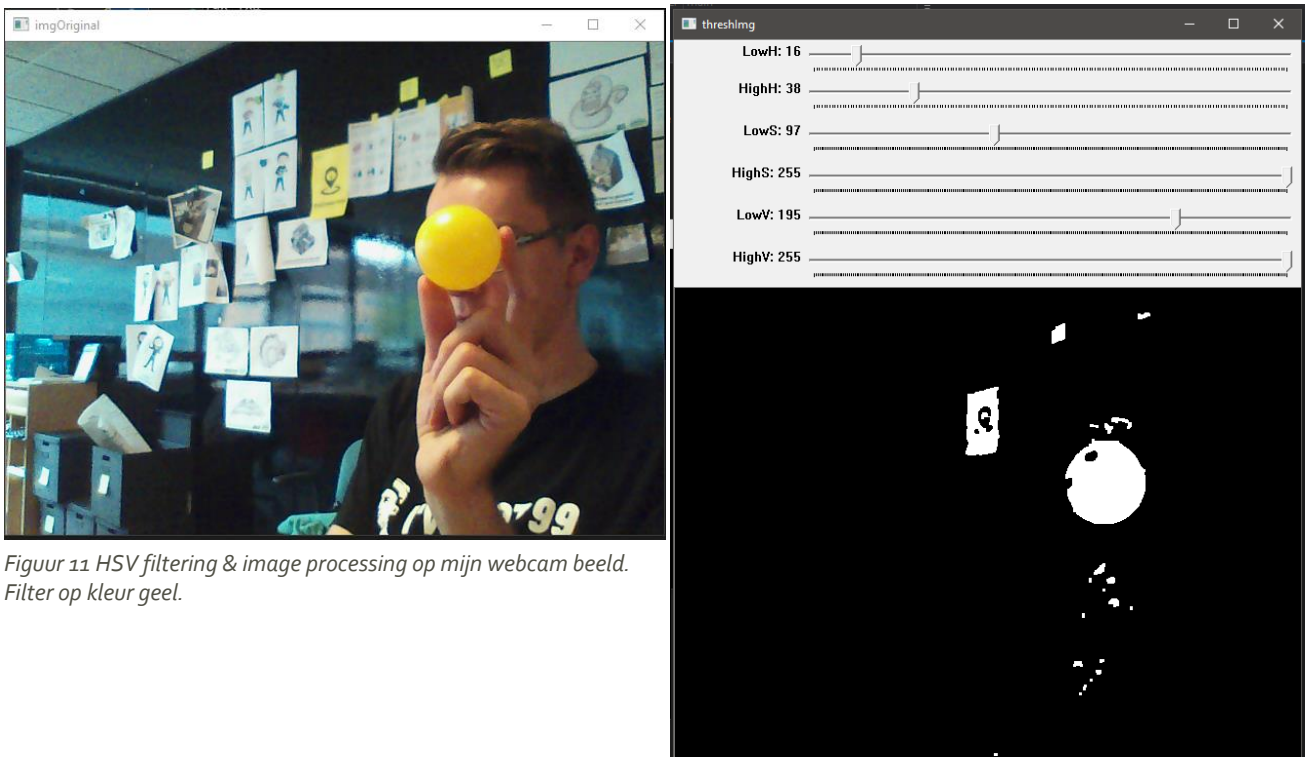
De code ziet er als volgt uit:

```
1.  cv::cvtColor(imgOriginal, hsvImg, CV_BGR2HSV);      // Convert Original Image to HSV Thresh Image

2.  cv::GaussianBlur(hsvImg, hsvImg, cv::Size(3, 3), 0);   //Blur Effect
    cv::dilate(hsvImg, hsvImg, 0);         // Dilate Filter Effect
    cv::erode(hsvImg, hsvImg, 0);          // Erode Filter Effect

3.  cv::inRange(hsvImg, cv::Scalar(lowH, lowS, lowV), cv::Scalar(highH, highS, highV), threshImg);
```

Figuur 10 Code voor het filteren van een beeld op een bepaalde reeks HSV waarden

1. Allereerst converteren we het beeld van mijn webcam (wat binnenkomt als RGB) naar HSV met een simpele conversiefunctie.
2. Vervolgens passen we drie bestaande OpenCV functies toe welke het beeld bewerken namelijk: *Gaussian Blur* maakt het beeld waziger, *dilate* groeit pixels uit en *erode* brokkelt deze juist weer wat af (OpenCV Development Team, 2017). Een combinatie van deze beeld bewerkende functies zorgt ervoor dat het beeld simpeler wordt en meer pixels dezelfde waardes hebben.
3. Als laatste passen we de inRange functie toe welke pixels binnen de high en low HSV-waardes op wit zit en de rest op zwart. De high en low HSV-waardes zijn te bepalen met 3 *sliders*. Met behulp van de sliders is aan te geven wat de waardes zijn waar binnen je filteren wilt.

Vervolgens wordt het bronbeeld en het resultaat van het uiteindelijk gefilterde plaatje getoond. Met behulp van de sliders zoek ik tijdens het draaien van het programma de kleur op die ik hebben wil. Deze waardes worden ook getoond en zijn dus op te slaan en vast te zetten als je telkens één en hetzelfde scenario hebt.

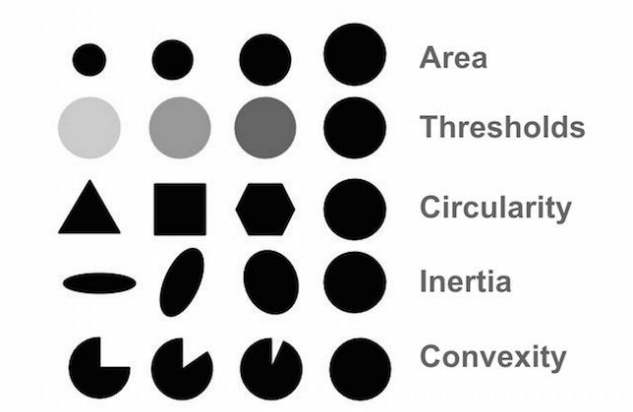


Figuur 11 HSV filtering & image processing op mijn webcam beeld. Filter op kleur geel.

Conclusie filteren op kleur
Mijn ervaring met het filteren op deze manier is dat als je een vast scenario hebt het heel goed te doen is om één bepaald object op te zoeken. Wel moet dit object een duidelijke, het liefst felle, kleur hebben. Het probleem zit hem in een vast scenario. Hiermee doel ik voornamelijk op licht. Zodra buiten de zon even wegvalt achter een wolk of het omgevingslicht verandert op een andere manier dan moet je de waardes aanpassen om het object weer te vinden. Dit zorgt voor een onpraktische ervaring. Stel je voor dat je dit toe wilt passen voor de beweegvloer waarbij je bijvoorbeeld elke speler een felgele pet op wilt zetten om te tracken dan moet je dit per kinderopvang locatie kalibreren. Als vervolgens iets aan de lichtinval verandert dan werkt het tracking systeem al niet meer. Ook bepaalde reflecties op objecten zorgen voor inconsistenties. Kijk bijvoorbeeld in mijn voorbeeld naar het balletje. Deze heeft een hoge glans, waardoor het witte deel met de meeste glans buiten de waardes valt en wegvalt in de detectie.

Al met al heb ik hier wel nuttige technieken aangeleerd en ga ik dit filteren later ook toepassen om bepaalde detectie en tracking methodes te testen, maar als uiteindelijke filter methode (op kleur) is dit niet praktisch voor de beweegvloer gezien de variabele die invloed hebben op de uitkomst.

2.3.3 Simple Blob Detection
Nu we beeld kunnen filteren gaan we door naar de volgende stap: het detecteren van objecten binnen het beeld. Een mogelijke manier is het gebruik van blob tracking.
Blob tracking houdt in dat je in een plaatje op zoek gaat naar een groep pixels met dezelfde kenmerken. Als je bijvoorbeeld een zwart plaatje hebt met drie witte vierkanten in het plaatje dan kan je deze drie witte vierkanten als drie blobs identificeren. Wat je wilt bereiken is dat je over meerdere plaatjes (video) deze blobs trackt. Nu is blob tracking op zich een hele brede term. Meer dan het tracken van een herkenbare groep pixels zegt het niet. Om tot goede blob tracking te komen moet je een combinatie van oplossingen gebruiken. Allereerst willen we de blobs kunnen identificeren. We willen dat dit consistent gebeurd. Binnen OpenCV bestaat de klasse *SimpleBlobDetector*. Deze simpele klasse is bedoelt om blobs in een plaatje te detecteren. Bij het aanmaken van een SimpleBlobDetector object is het mogelijk een aantal parameters mee te geven. Het liefste wil de blob detector ronde blobs zien, maar met de parameters valt hier iets aan te sleutelen. De verschillende parameters zijn te zien op Figuur 12.



Figuur 12 Parameters waarop je de SimpleBlobDetector kan filteren. (Mallick, Blob Detection Using OpenCV, 2015)

Aan de hand van een tutorial (Mallick, Blob Detection Using OpenCV, 2015) is de SimpleBlobDetector geïmplementeerd. Het is een goede start voor een tracking oplossing bij de beweegvloer. Mallick zijn tutorial behandelt de SimpleBlobDetector in combinatie met een plaatje, uiteindelijk is het in dit onderzoek ook op video toegepast om te kijken hoe de SimpleBlobDetector zich gedraagt op video.

Implementatie SimpleBlobDetector op een afbeelding

Als input plaatje is een afbeelding gebruikt met mensen van bovenaf welke met groot contrast op een witte achtergrond geplaatst zijn. Het resultaat is te zien op Figuur 14.

De code om de SimpleBlobDetector te gebruiken is als volgt:

```
1. // List of KeyPoints; Storage for blobs
   vector<KeyPoint> keypoints;

2. // Set up detector with paramaters
   Ptr<SimpleBlobDetector> detector = SimpleBlobDetector::create(params);

3. // Detect blobs, put detected blobs in the list of KeyPoints
   detector->detect(im, keypoints);

4. // Draw detected blobs as red circles.
   Mat im_with_keypoints;
   drawKeypoints(im, keypoints, im_with_keypoints, Scalar(0, 0, 255), DrawMatchesFlags::DRAW_RICH_KEYPOINTS);
```

Figuur 13 Implementatie van OpenCV C++ SimpleBlobDetector.

1. Eerst maken we een lijst om de gevonden blobs in op te slaan. De variabele *KeyPoint* bevat data zoals de positie, rotatie etc. Met de SimpleBlobDetector komt hier alleen positie en grootte data in; meer krijg je niet terug.
2. Vervolgens maken we de SimpleBlobDetector aan met de meegegeven paramaters (variabele *params*, zie ook Figuur 12) waar de blobs aan moeten voldoen.
3. Daarna roepen we de methode *detect* aan van de SimpleBlobDetector. Deze methode gaat als volgt te werk:
   - Het aangeleverde plaatje wordt naar binair geconverteerd. Dit gebeurt een aantal keer opnieuw. Hoeveel stappen geef je aan met een variabele. Stel dat je zegt max stappen = 100, stap grootte = 10 dan krijg je 10 stappen tot je aan de 100 zit dus 10 plaatjes in totaal. Dit gebeurt allemaal op de achtergrond.
   - De FindContours methode wordt gebruikt om groepen witte pixels te identificeren. Het midden van elke groep wordt gecalculeerd.
   - Het midden van de groepen wordt over alle plaatjes bij elkaar bekeken. Liggen deze over verscheidene plaatjes binnen een aangegeven afstand dan is dit één blob.
   - Van elke gevonden groep wordt het uiteindelijke midden berekend en de grootte van de blob. Dit is wat je uiteindelijk terugkrijgt in de *keypoints* list. (OpenCV Development Team, 2017).
4. Vervolgens tekenen we de gedetecteerde blobs als rode cirkels over de afbeelding heen.

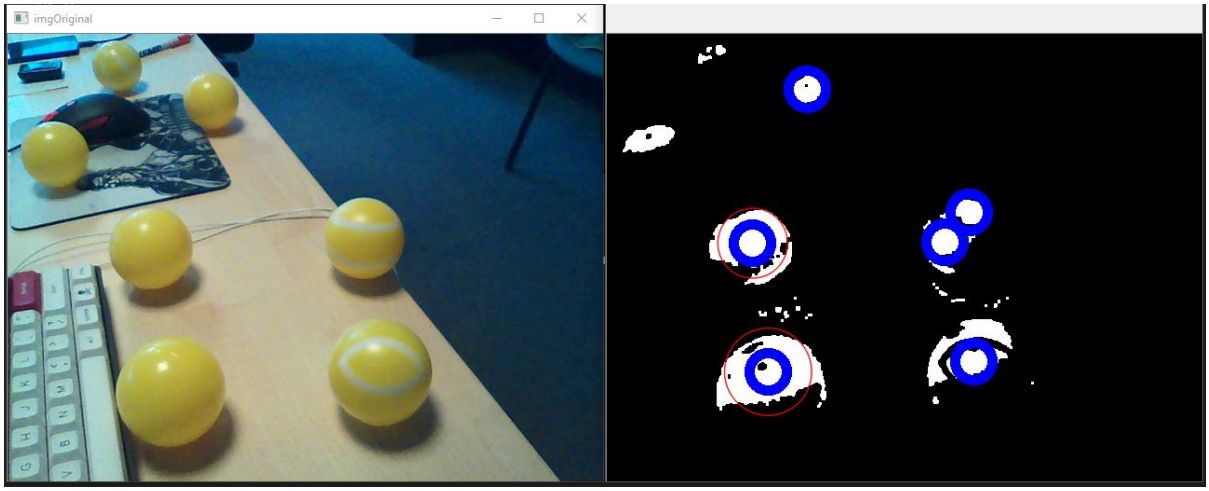
Figuur 14 Het resultaat van de SimpleBlobTracker implementatie, gevonden blobs zijn rood omcirkeld.

Zoals te zien is op Figuur 14 werkt de SimpleBlobDetector prima als je een tijdje met de parameters speelt en je een goed, duidelijk plaatje aanlevert. Het meest interessante wat uit de werking van de SimpleBlobDetector te halen valt is dat het eigenlijk de FindContours methode is die aan het werk is (OpenCV Development Team, 2017). Aan de hand van de parameters zal hij hiermee kijken of de gevonden contouren voldoen aan de parameters. Om meer controle te krijgen over wat je wilt detecteren en om meer informatie te krijgen over de gevonden objecten is het interessant om naar de FindContours methode te kijken. Deze wordt behandeld in 2.3.4.

Implementatie SimpleBlobDetector met videobeeld

Na een succesvolle test met een van tevoren doelgericht gemaakte afbeelding is de volgende stap toepassing op video. De werkwijze voor video implementatie is het verwerken van de kleurenfilter uit 2.3.2 in de SimpleBlobDetector implementatie.
Qua code is het niet meer dan de code van de filter en de code van de blob detector samenvoegen. Eerst wordt het videobeeld gefilterd zodat we uitkomen bij het resultaat wat te zien is in Figuur 11. Vervolgens maken we de detector aan zoals te zien is in Figuur 13.
Bij het opstarten blijkt de detector goed te werken op video; wel valt op te merken dat het allemaal wat traag is. Het beeld loopt achter en de beweging lopen op minder dan 30 frames per seconde. Zodra er meer gele ballen in beeld komen om te tracken merken we dat het echt heel traag wordt. Twee ballen is nog te doen qua performance, maar bij 3 of meer gaat het echt heel erg haperen. Het filter heeft ook moeite met de verschillende lichtinvallen op de ballen, zoals te zien is op Figuur 15 zijn een deel van de ballen verderop in beeld slecht te zien op het gefilterde beeld. Het is wel duidelijk dat het detecteren van de

blobs prima gaat, zolang het filteren goed gaat. Het is alleen erg traag allemaal met meerdere blobs op video. Het volledige resultaat is te zien in de volgende afbeelding:



Figuur 15 SimpleBlobDetector op video. HSV Kleurenfilter op geel.

Conclusie Simple Blob Detection

De SimpleBlobDetector is een fascinerende klasse binnen OpenCV. Wat er goed aan is, is dat hij simpel is toe te passen. Het is duidelijk dat de klasse opgesteld is om snel tot een blob detectie te komen. Dat de klasse simpel werkt heeft tegelijk ook nadelen. Veel komt aan op het hebben van een goed bronbeeld; maar het hebben van een duidelijk bronbeeld waar je de functies en algoritmes op los kan laten blijft belangrijk bij computer vision. Verder is de SimpleBlobDetector beperkt. Op de achtergrond wordt de FindContours methode gebruikt, welke uitgebreid is, maar omdat deze gesloten gebruikt wordt binnen de SimpleBlobDetector heb je hier naast de parameters die je mee kan geven geen invloed op. Data zoals de rotatie/de hoek van de gedetecteerde objecten is niet beschikbaar binnen de SimpleBlobDetector; terwijl dit wel beschikbaar is wanneer je zelf de FindContours methode implementeert. Om deze reden gaan we hier na verder met de FindContours methode. Door deze methode zelf te implementeren hebben we meer controle over de data die we terugkrijgen.

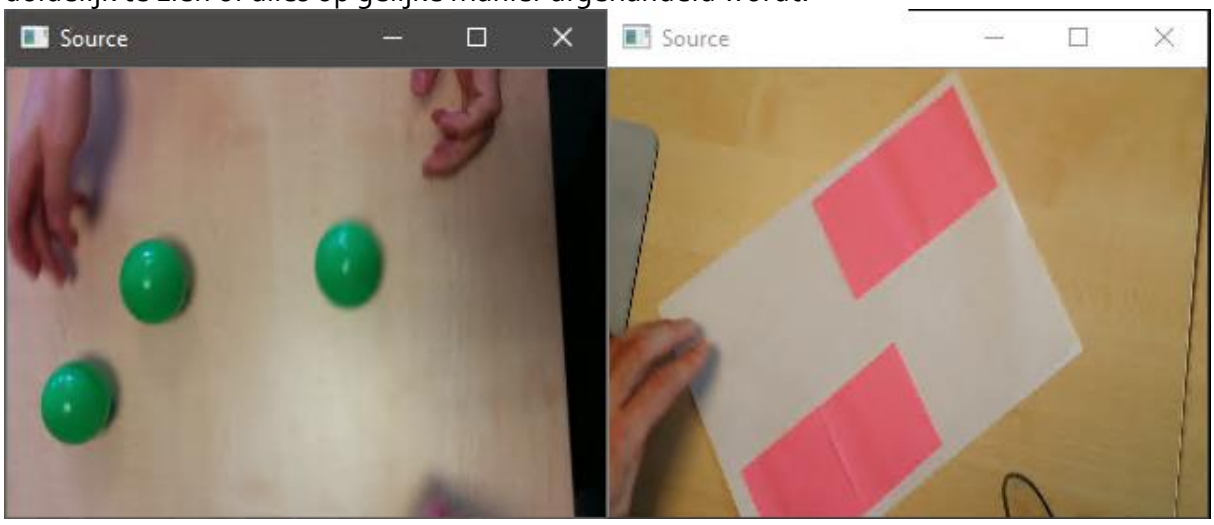
Verder komt performance al snel om de hoek kijken bij het gebruiken van de SimpleBlobDetector i.c.m. videobeelden. Wat ik merkte was dat zodra er meer dan 2 gele ballen in beeld kwamen de boel erg slecht draaien ging. Dusdanig slecht dat wanneer je dit bij een interactieve applicatie zou willen gebruik het storend wordt. Wat we niet willen hebben bij de beweegvloer is dat de interactiebeleving ondermaats is door vertragingen in de tracking. Exact gemeten is het niet aangezien dit de eerste oplossing is die geïmplementeerd is. Het plan is om verder te gaan met de FindContours methode. Ik hoop door deze los toe te passen een winst op performance te boeken en ook meer controle te krijgen over wat er precies gedetecteerd gaat worden. Zoals bevonden converteert de SimpleBlobDetector ook het bronbeeld meerdere malen naar binair wat allemaal losse stappen zijn welke tijd/rekenkracht kosten. Op een afbeelding is dit geen probleem aangezien het één keer uitgevoerd wordt. In dit onderzoek zijn we op zoek naar een oplossing voor een interactieve applicatie (de beweegvloer) dus het moet allemaal wel soepel draaien.

2.3.4 Detectie met FindContours

Bij de SimpleBlobDetector wordt op de achtergrond de FindContours methode van OpenCV aangeroepen. Ook als je op zoek gaat naar blob detection i.c.m. OpenCV op internet dan kom je vaak de FindContours methode tegen. Wat deze functie binnen OpenCV doet is gebruik maken van een algoritme (Suzuki, 1985) om de contouren binnen een plaatje te vinden. De contouren zijn erg handig om vormen te analyseren en voor het doen van object detectie (OpenCV Development Team, 2017). Wat je deze methode moet voeden is een binaire afbeelding. Wat inhoudt dat de pixels in je afbeelding of video wit of zwart zijn. Vervolgens gaat het algoritme de pixels langs en vindt op deze manier de contouren. Je krijgt de contouren terug (mits er contouren gevonden zijn) als een lijst van punten. Omdat de FindContours methode ook weer een binair plaatje nodig heeft is het belangrijk dat je een goede processing doet over het beeld wat je wilt gaan gebruiken. Het filteren van het bronbeeld tot een bruikbaar beeld voor de algoritmes blijft terugkomen. Gezien we eerder met kleur filteren bruikbare resultaten (beelden) hebben geproduceerd gaan we voor de volgende tests hier verder mee.

Implementatie FindContours

Allereest beginnen we zoals bij alle voorgaande implementaties weer met het filteren van het bronbeeld. Ditmaal heb ik zelf een vaste video opgenomen waar ik een A4 met roze notitie blaadjes beweeg en deze ook deels bedek met mijn hand tijdens de video. Ook heb ik een tweede video gemaakt waar ik groene ballen heen en weer laat rollen. Ik heb deze video's opgenomen om het proces deels te versnellen; zo hoef ik niet telkens mijn webcam te gebruiken. Ik kreeg op kantoor al de bijnaam "de tovenaar" omdat ik telkens met objecten voor mijn webcam zat te zwaaien. Ook is een video handig om te kijken hoe de code zich gaat gedragen wanneer het herschreven is binnen Unity3D. Met dezelfde video is duidelijk te zien of alles op gelijke manier afgehandeld wordt.



Figuur 16 De twee video's waarmee ik FindContours test.

De implementatie wordt doorgenomen aan de hand van de broncode. De code is opgesteld met behulp van een tutorial welke te vinden is binnen de online documentatie van OpenCV, specifiek die van de FindContours methode (OpenCV Development Team, 2017).

```
1. //Lower video resolution for performance reasons.
   resize(frame, frame, Size(320, 240), 0, 0, INTER_CUBIC);

   //Tresholding
   Mat hsvImg, threshImg;
2. cv::cvtColor(frame, hsvImg, CV_BGR2HSV);
   cv::inRange(hsvImg, cv::Scalar(lowH, lowS, lowV), cv::Scalar(highH, highS, highV), threshImg);

   //Morphing to make the image even easier to process
   Mat erodeElement = getStructuringElement(MORPH_RECT, Size(3, 3));
   //dilate with larger element so make sure objects are nicely visible
3. Mat dilateElement = getStructuringElement(MORPH_RECT, Size(8, 8));
   erode(threshImg, threshImg, erodeElement);
   erode(threshImg, threshImg, erodeElement);
   dilate(threshImg, threshImg, dilateElement);
   dilate(threshImg, threshImg, dilateElement);
```

Figuur 17 Eerste deel code van mijn implementatie van de FindContours methode.

Figuur 17:

1. Gezien we dit keer wat meer processing loslaten op de video verklein ik eerst het beeld naar een lagere resolutie. Om de blokken uit de video te halen hebben we niet heel veel pixels nodig. 320*240 is scherp zat en het zorgt ervoor dat de berekeningen sneller gaan (er zijn immers minder pixels om langs te lopen).
2. Hier wordt het beeld geconverteerd naar HSV en vervolgens met behulp van de inRange methode naar binair omgezet (alleen witte en zwarte pixels).
3. Processing wordt losgelaten op het plaatje. Het is mogelijk beter om de processing voor het converteren naar binair plaats te laten vinden, zodat dit converteren sneller gaat. Zelf heb ik bij mijn video's geen last gehad van ruis en dergelijke vandaar dat ik het achteraf laat gebeuren. Het gebeurd immers alsnog voordat er naar contouren gezocht wordt. (Figuur 17)

```
1. // List with list of points, to store the contours in.
   vector<vector<Point> > contours;

   /// Find contours
2. findContours(threshImg, contours, CV_RETR_TREE, CV_CHAIN_APPROX_SIMPLE, Point(0, 0));

   /// Find the rotated rectangles and ellipses for each contour
3. vector<RotatedRect> minRect(contours.size());
   vector<RotatedRect> minEllipse(contours.size());

   for (int i = 0; i < contours.size(); i++)
   {
       //Only find rotatedRect if the contour is big enough (exceeds minimum size).
       if (contours[i].size() > 19)
       {
4.         minRect[i] = minAreaRect(Mat(contours[i]));
           //cout << contours[i] << "Point" << endl;
           if (contours[i].size() > 5)
           {
               minEllipse[i] = fitEllipse(Mat(contours[i]));
           }
       }
   }
```

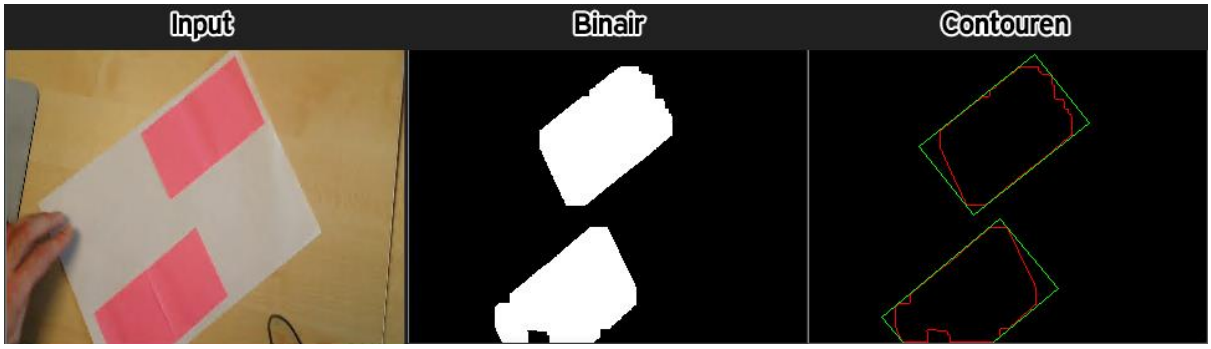
Figuur 18 Tweede deel code van mijn implementatie van de FindContours Methode.

Figuur 18:

1. Een lijst van punten om de contouren in op te slaan.
2. Hier wordt de FindContours methode aangeroepen welke in het gegeven plaatje (de eerste variabele: *threshImg*) op zoek gaat naar contouren. De tweede variabele (*contours*) is de lijst welke we bij 1. aangemaakt hebben. Hier worden de gevonden contouren in opgeslagen. De derde variabele is de manier waarop de contouren opgehaald worden. CV_RETR_TREE haalt de contouren op en alle subcontouren van een contour worden onder dit contour opgeslagen. Daarna komt CV_CHAIN_APPROX_SIMPLE wat aangeeft hoe de contouren opgeslagen worden. Bij deze SIMPLE methode worden rechte lijnen gecomprimeerd naar alleen de eindpunten. Een vierhoek zal dus opgeslagen worden met alleen de 4 hoekpunten. Als laatste is het mogelijk een *offset* mee te geven, waar ik geen gebruik van maak. (OpenCV Development Team, 2017)
3. Hier worden twee lijsten gemaakt om de omliggende vierhoeken en ellipsen in op te slaan welke we bij 4. op gaan zoeken.
4. Bij dit punt gaan we per contour op zoek naar de omliggende vierhoek en omliggende ellips. Ik geef hier aan dat er alleen een vierhoek moet komen als de contour een bepaalde grootte heeft. Op deze manier pas ik al wat filtering toe op de gedetecteerde objecten. In dit geval detecteren we dus vierhoeken als de gevonden contouren groot genoeg zijn.

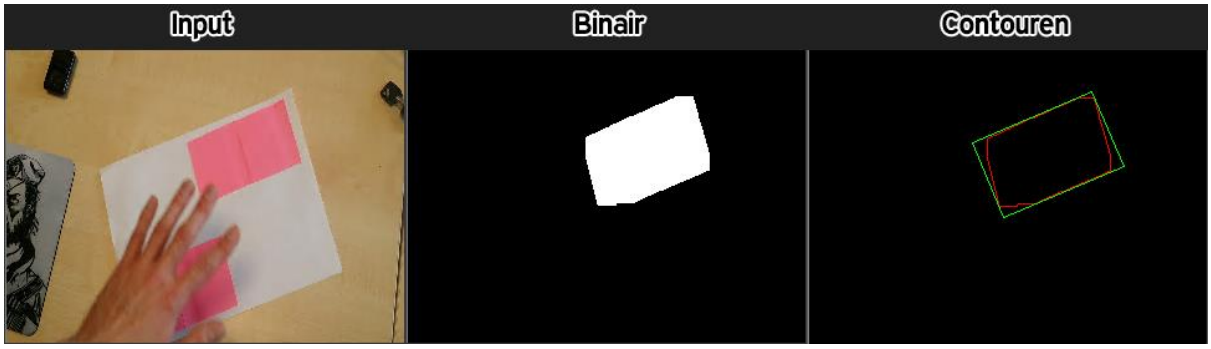
Verderop in de code worden de gevonden vierhoeken op het scherm getekend.

Het resultaat van de code is dat er op beide van mijn opgenomen video's (die met groene ballen en die met roze blokken) contouren gevonden worden en dat er ook per contour een vierhoek omheen komt.
Gezien er twee video's zijn, hebben we meerdere resultaten. Per video zijn er twee scenario's welke per stuk doorgenomen worden.



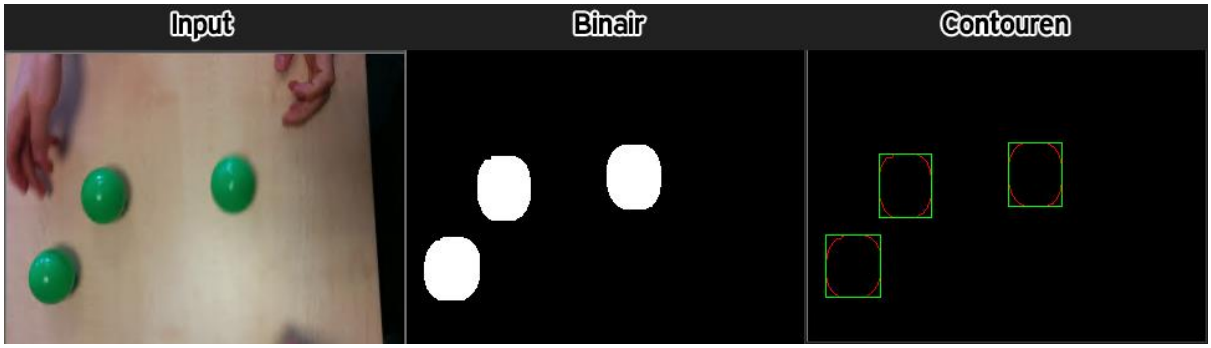
Figuur 19 Scenario 01 – Roze blokken zonder obstructie

Figuur 19 - We beginnen bij de video met de roze blokken. Zoals te zien werkt de conversie naar binair goed. Er zitten wat vlekken in, dit komt door lichtinval etc. Het probleem met op kleur filteren is dat wanneer er een schaduw over je object valt of iets dergelijks dan veranderen de HSV waardes. Je wilt ook je kalibratie niet te breed zetten, want dan worden er objecten opgepakt welke je niet wilt tracken. Bij het veld contouren vallen er rode en groene lijnen te zien. De rode lijnen zijn de gevonden contouren, deze komen overeen met de randen van de witte pixels in het binaire plaatje. De contouren worden opgehaald vanaf het binaire beeld. De groene lijnen zijn de gevonden rechthoeken aan de hand van de contouren.



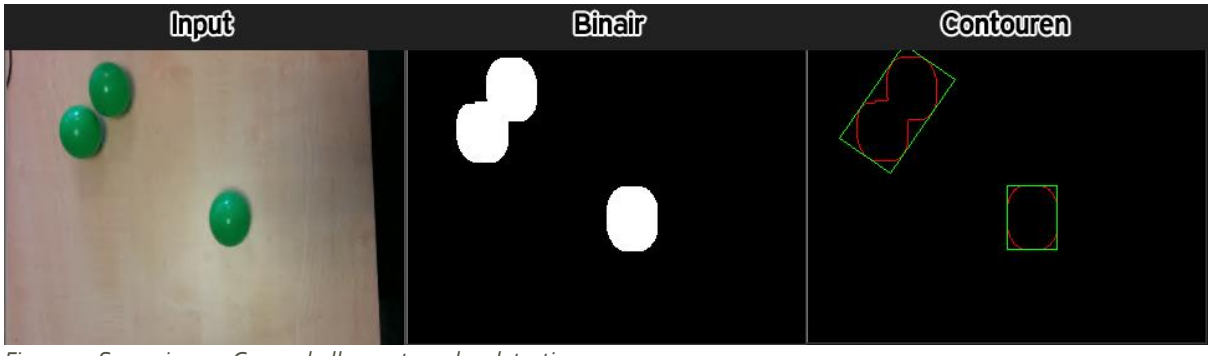
Figuur 20 Scenario 02 – Roze blokken met obstructie

Figuur 20 – Hier bedek ik één van de twee blokken met mijn hand. De HSV waardes van het bedekte blok veranderen hierdoor (het wordt donkerder) waardoor het niet meer opgepakt wordt door de kleurenfilter. Het resultaat is dat het blok ook verdwijnt in de conversie naar binair (deze gebeurd aan de hand van de HSV kleurenfilter) en er worden dus ook geen contouren gevonden.



Figuur 21 Scenario 03 – Groene ballen zonder obstructie of overlap

Figuur 21 – Ditmaal de groene ballen video. We zien hier het ultieme scenario. De ballen worden perfect opgepakt door de kleurenfilter wat resulteert in een perfect conversie naar binair. De contouren worden mooi opgepakt en het resultaat is drie rechthoeken rondom de gevonden contouren.



Figuur 22 Scenario 04 – Groene ballen met overlap detectie

Figuur 22 – We zien hier dat de conversie van kleur naar binair nog steeds prima gaat. Het groen steekt genoeg af om ook onder schaduw nog goed geconverteerd te worden. Interessant is om te zien dat wanneer de contouren overlappen de objecten gedetecteerd worden als één. De twee ballen in de linkerbovenhoek smelten als het ware samen tot één object.

Conclusie FindContours
Wanneer je bronbeeld perfect is dan is FindContours in combinatie met het vinden van de vierhoeken genoeg, gezien deze een positie en rotatie hebben welke je om kan zetten naar objecten in bijvoorbeeld een game. Dit perfecte scenario valt te zien in Figuur 21.
Waar FindContours niet voor zorgt, is daadwerkelijke tracking. De contouren worden gedetecteerd en in een lijst gestopt waar je vervolgens vierhoeken van kan berekenen; maar de volgorde is willekeurig en er kan storing optreden wanneer bijvoorbeeld objecten overlappen of wanneer een object even buiten beeld is zoals in Figuur 20.
Dat je bronbeeld constant perfect is, is eigenlijk nooit het geval. Een voorbeeld voor de beweegvloer is dat een speler van bijvoorbeeld uit het beeld kan lopen waardoor de FindContours methode de volgorde van de contouren opschuift en je hele tracking een rommel wordt. Dit gebeurd ook bij één van mijn scenario's. Op Figuur 22 valt te zien dat wanneer twee te tracken objecten dicht bij elkaar komen de contouren overlappen waardoor het samensmelt tot één object. Dit scenario is ook goed mogelijk bij de beweegvloer; denk aan twee kinderen die vlak naast elkaar staan.

Het blijkt dat er ook nog algoritmes nodig zijn om de gedetecteerde objecten te blijven volgen over meerdere frames. Wanneer een object even verdwijnt uit het beeld wil je zijn laatste positie bewaren en wanneer het object binnen korte tijd weer terug komt wil je dat het tracken door gaat en dat dit object weer zijn vorige *id* toegewezen krijgt. Denk bijvoorbeeld aan een speler van de beweegvloer die eventjes weg loopt. We hebben als doelgroep jonge kinderen bij de kinderopvang, je kan hier niet van verwachten dat die 100% van de speeltijd altijd perfect in het speelveld blijven lopen.

2.4 Unity & OpenCV

2.4.1 Integratie OpenCV in Unity (verschillende opties)

De beweegvloer moet uiteindelijk interactieve, leerzame games gaan projecteren. Het is namelijk de bedoeling dat de spelers van de beweegvloer spelenderwijs leren. Er moeten dus games voor de beweegvloer ontwikkeld worden. Een veel toegepast pakket om dit te doen is Unity3D. Unity is een game engine welke gebundeld komt met een uitgebreide *editor*. Gezien de brede toepasbaarheid van Unity, eerdere ervaring met Unity van mensen binnen Springlab en het feit dat het veel gebruikt wordt in de industrie is voor Unity gekozen als ontwikkelpakket voor de beweegvloer games. Een voordeel van het feit dat Unity veel gebruikt wordt in de industrie is dat wanneer de beweegvloer groter wordt het mogelijk is om externe partijen games te laten ontwikkelen voor de beweegvloer.
Unity beschikt helaas niet over een computer vision module. Tot zover hebben we OpenCV los gebruikt binnen een eigen opgezette omgeving. Dit houdt in dat ik OpenCV zelf gecompileerd heb en dit gebruikt heb binnen een C++ omgeving in combinatie met Visual Studio. Wat we nu moeten hebben is het alreeds gemaakte werk kunnen gebruiken binnen een Unity omgeving. Wat we in essentie willen bereiken is dat OpenCV zijn werk doet met het detecteren en tracken van objecten en we vervolgens in Unity de positie data van de gevonden objecten hebben. Deze data is dan toe te passen op objecten binnen Unity waarmee een game uitgewerkt kan worden.

Mogelijkheden om OpenCV & Unity samen te laten werken
Ik ben op internet op zoek gegaan naar het integreren/samen laten werken van OpenCV met Unity. Gelukkig ben ik niet de eerste die dit voor elkaar wil krijgen dus er zijn verschillende mogelijkheden te vinden. Hieronder neem ik de mogelijkheden door welke ik gevonden en bekeken/geprobeerd heb.

Zelf compileren als plugin
OpenCV is een open source library. Dit betekend dat alle broncode beschikbaar is en je het ook zelf kunt samenstellen tot één pakket. Het is mogelijk dit pakket als plugin te gebruiken binnen Unity. Ik heb dit geprobeerd aan de hand van een tutorial (Berge, 2017). Wat er in de tutorial gemaakt word is een plugin voor Unity waarin alle OpenCV functies die je nodig hebt gedefinieerd staan. Je programmeert alle functies die je uit wilt laten voeren door OpenCV voor, markeert deze als te exporteren naar de plugin en met de juiste implementatie kan je deze vervolgens aanroepen binnen de Unity omgeving.
Het voordeel van deze oplossing is dat alle OpenCV code binnen de C++ omgeving draait. Dit is efficiënter dan oplossingen waar de C++ code van OpenCV vertaald is naar een taal welke binnen de Unity omgeving werkt. Er zit geen extra laag tussen, de code draait in C++ en wordt direct aangeroepen vanuit de Unity omgeving.
Uiteindelijk heb ik deze methode niet gebruikt. Ik kreeg het tutorial gedeelte werkende, maar zodra ik begon met video input liep ik tegen fouten aan. Het gebruik van video binnen OpenCV is weer afhankelijk van een andere plugin. Dit gaf foutmeldingen binnen de Unity omgeving. Ik heb besloten deze implementatie links te laten liggen gezien er alternatieven zijn en ik niet al mijn tijd kwijt wilde raken aan het werkend krijgen van OpenCV binnen Unity. Naast het feit dat ik tegen problemen aan liep bij het werkend krijgen van deze oplossing is het ook een tijdrovende oplossing. Je moet elke functie van OpenCV welke je binnen Unity wilt gebruiken apart definiëren en aanroepen. Wil je dus een aanpassing doen aan je OpenCV tracking code dan moet je terug naar de C++ omgeving, de code aanpassen, dit exporteren als plugin om dit vervolgens te kunnen testen. Werkt het niet naar behoren kun je al deze stappen weer opnieuw doorlopen.

OpenCV C# port gebruiken als plugin
Er is een C# conversie beschikbaar van OpenCV onder de naam EmguCV (Emgu Corporation, 2017). Deze hebben een eigen uitgebrachte versie voor Unity welke beschikbaar is voor $399 (Emgu Corporation, 2017). Dit vind ik teveel geld voor waar ik mee bezig ben dus heb dit om die reden afgeslagen. Verder heb ik gekeken naar het zelf compileren van EmguCV voor Unity. Dit komt deels op hetzelfde neer als de hierboven genoemde methode. Na hier tijd aan besteed te hebben liep ik tegen dezelfde problemen aan als bij de C++ plugin methode. Ik wilde hier op dat moment niet meer tijd aan besteden zonder eerst naar de nog beschikbare alternatieven gekeken te hebben.

Via het netwerk laten communiceren
Een alternatief op het gebruiken van een plugin is OpenCV gewoon zijn werk laten doen in C++ en de data die je nodig hebt in Unity versturen via het netwerk. Het voordeel van deze oplossing is dat je niets hoeft te veranderen aan je OpenCV opzet. Ik zou in theorie gewoon mijn huidige code kunnen laten draaien en de data van de gevonden contouren + rechthoeken door sturen naar Unity over een netwerkpoort. Het probleem van deze oplossing is dat ik weinig tot geen kennis van netwerk gerelateerde zaken heb. Nu had ik de

keuze kunnen maken dit te onderzoeken en implementeren maar ik kon slecht een inschatting maken hoeveel tijd dit mij zou gaan kosten.

OpenCV For Unity Asset in de store
EmguCV is niet de enige computer vision *asset* in de *assetstore* van Unity. Een goedkoper alternatief is *OpenCV For Unity* voor $95 (Enox Software, 2017). Dit is een *port* van de Java versie van OpenCV. Deze asset sprak mij aan gezien de redelijke prijs, het feit dat hij regelmatig updates ontvangt en de vele positieve reviews. Uiteindelijk ben ik voor deze oplossing gegaan.

2.4.2 Verantwoording keuze *OpenCV for Unity* asset
Ik was de *OpenCV for Unity* asset al op het spoor voordat ik alle alternatieven geprobeerd had. Ik heb toen besproken met mijn vorige stage begeleidster (Imara Speek) of ik meteen de asset aan zou schaffen of dat ik eerst naar alternatieven zou kijken. Mij is toen aangereden om eerst te proberen de kosteloze alternatieven te implementeren alvorens mijn uitweg te zoeken in deze betaalde oplossing.
Toen ik na zo'n 2 weken stoeien met losse OpenCV integraties niet verder kwam heb ik serieus gekeken naar deze asset. De grootste vraagtekens werden gezet bij de ondersteuning. Je koopt een asset waardoor je vervolgens afhankelijk bent van wat dit pakket kan. Mist er een functie die cruciaal is voor wat we eerder gemaakt hebben? Dan is het weggegooid geld. Om hier achter te komen heb ik de documentatie van *OpenCV for Unity* bekeken. Zit er image processing in? Beschikt het over de FindContours methode? In de documentatie (Enox Software, 2016) vond ik deze methodes en technieken, waardoor ik er van overtuigd was dat deze asset voldoende was voor wat er tot nu toe geïmplementeerd is. Nog een voordeel is dat de asset geïntegreerd is in Unity. Dit betekend dat wanneer je de game later wilt *builden* voor het systeem waar de beweegvloer op gaat draaien, dan hoef je geen extra processen te draaien. OpenCV draait met deze asset mee in de Unity game. Je levert de beweegvloer applicatie als Unity .exe bestand en alles draait hier in mee.

Ik heb het aanschaffen van de *OpenCV for* Unity asset voorgelegd aan Imara en Jan-Paul de Beer, de baas bij Springlab. Ze waren het met mij eens dat deze asset een goede oplossing was, omdat ik daardoor door kon met mijn werk in plaats van nog langer tijd te besteden aan een implementatie van OpenCV. De prijs was geen probleem, het is een eenmalige aanschaffing wat ook hielp bij de keuze.

2.4.3 Implementatie FindContours code in Unity
Na het aanschaffen van de *OpenCV for Unity* asset ben ik de documentatie ingedoken om te kijken wat de verschillen zijn qua *syntax* met de C++ variant van OpenCV. De verschillen zijn gelukkig niet heel groot. Er worden wat andere variabelen gebruikt met soms net een andere naamgeving, maar in essentie is de structuur hetzelfde. De gehanteerde aanpak was daarom het kopiëren van de code welke eerder in het onderzoek geschreven is (in o.a. 2.3.4) en dit regel voor regel omzetten. Het image processing gedeelte was snel overgezet. Per regel was het opzoeken hoe het binnen de C# omgeving aangeroepen moest worden, maar functioneel is het hetzelfde. Op Figuur 23 is te zien hoe vergelijkbaar de stukken code zijn.

OpenCV for Unity C# Code

```
1.  //Downsize video; results in faster processing.
    Imgproc.resize(sourceImg, sourceImg, new Size(320, 240), 0, 0, Imgproc.INTER_CUBIC);

    //Convert to HSV Image and threshold it within range of the given target colour.
    Imgproc.cvtColor(sourceImg, hsvImg, Imgproc.COLOR_BGR2HSV);
2.  Core.inRange(hsvImg, new Scalar(lowH, lowS, lowV), new Scalar(highH, highS, highV), threshImg);

    //Morph HSV Image to make it more 'readable' (erosion and dilation are applied).
    Mat erodeElement = Imgproc.getStructuringElement(Imgproc.MORPH_RECT, new Size(3, 3));
    //dilate with larger element to make sure the object is clearly defined.
    Mat dilateElement = Imgproc.getStructuringElement(Imgproc.MORPH_RECT, new Size(8, 8));
3.  Imgproc.erode(threshImg, threshImg, erodeElement);
    Imgproc.erode(threshImg, threshImg, erodeElement);
    Imgproc.dilate(threshImg, threshImg, dilateElement);
    Imgproc.dilate(threshImg, threshImg, dilateElement);
```

OpenCV C++ Code

```
1.  //Lower video resolution for performance reasons.
    resize(frame, frame, Size(320, 240), 0, 0, INTER_CUBIC);

    //Tresholding
    cv::cvtColor(frame, hsvImg, CV_BGR2HSV);
2.  cv::inRange(hsvImg, cv::Scalar(lowH, lowS, lowV), cv::Scalar(highH, highS, highV), threshImg);

    //Morphing to make the image even easier to process
    Mat erodeElement = getStructuringElement(MORPH_RECT, Size(3, 3));
    //dilate with larger element so make sure objects are nicely visible
    Mat dilateElement = getStructuringElement(MORPH_RECT, Size(8, 8));
    erode(threshImg, threshImg, erodeElement);
3.  erode(threshImg, threshImg, erodeElement);
    dilate(threshImg, threshImg, dilateElement);
    dilate(threshImg, threshImg, dilateElement);
```

Figuur 23 Vergelijking van het image processing gedeelte tussen C++ en C# binnen Unity met de OpenCV for Unity asset.

Het FindContours gedeelte is op eenzelfde manier als Figuur 23 vergelijkbaar. De code was dus relatief snel over te zetten. Het meeste werk waren bepaalde specifieke functies die net een ander type variabele nodig hadden in C#. De keuze voor de *OpenCV for Unity* asset is geen verkeerde geweest. Gezien de snelheid van het overzetten van de C++ code naar C# met Unity. Alle methodes die nodig waren zaten in de asset.

Met toegang tot Unity & OpenCV samen ben ik begonnen aan het overzetten van de tracking data naar objecten binnen Unity. Bij het observeren van de rechthoeken, welke te zien zijn in de scenario's in Figuur 19 t/m Figuur 22 was mijn eerste assumptie dat wanneer deze rechthoeken gelijk staan aan spelers dit tot een bruikbaar prototype moest kunnen leiden. Wat volgt is de positie en rotatie data van deze rechthoeken gebruiken en overzetten op zogeheten *game objecten* binnen Unity.

In code is dit als volgt gedaan:

```
//Only assign a rotatedRect if the contour is big enough (exceeds minimum size).
if (contours[i].size().area() > 25)
{
    minAreaRects[i] = Imgproc.minAreaRect(new MatOfPoint2f(contours[i].toArray()));

    if(i == 0)
        player1Rect = minAreaRects[i].clone();

    if(i == 1)
        player2Rect = minAreaRects[i].clone();
}
```

Figuur 24 Kopiëren van de gevonden vierhoeken naar twee spelers vierhoeken.

Hier (Figuur 24) kopieer ik de gevonden vierhoeken, mits ze groot genoeg zijn (area > 25) naar een lijst met twee vierhoeken voor de spelers. Voor deze eerste implementatie ben ik uitgegaan van twee spelers. Mijn voorbeeldvideo heeft namelijk ook twee te tracken objecten, de roze vierkanten op het papier. Voor de uiteindelijk beweegvloer willen we hebben dat het aantal spelers variabel is. Iemand kan uit de beweegvloer lopen of er in stappen.
De data van de vierhoeken van de spelers, welke alleen bijgewerkt worden zolang er twee vierhoeken zijn die groot genoeg zijn, zet ik vervolgens over op twee game objecten binnen Unity.

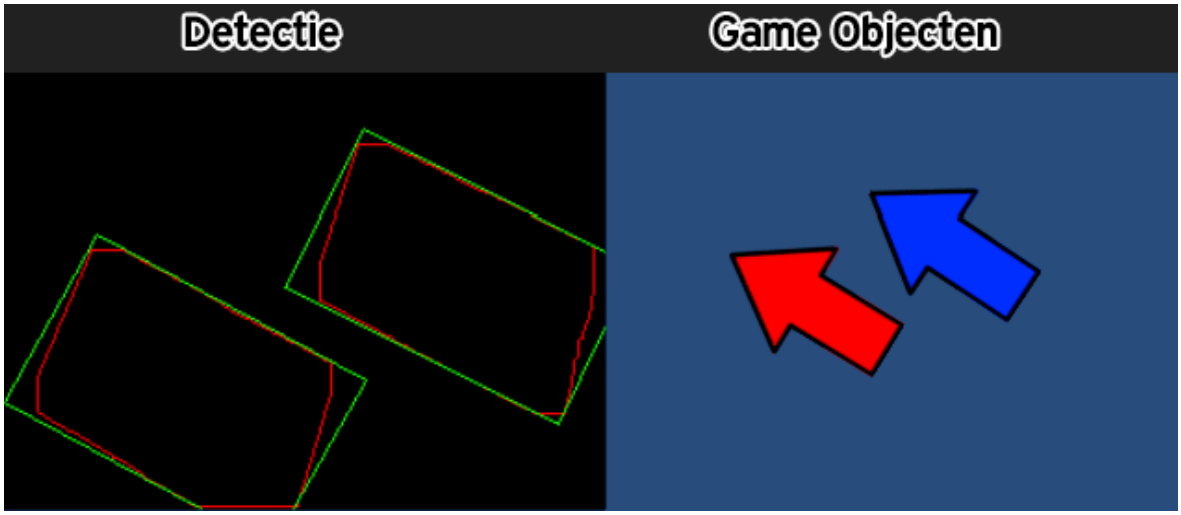
```
//Set gameObjects position
players[0].transform.position = new Vector3((float)player1Rect.center.x, -(float)player1Rect.center.y, 0);
players[1].transform.position = new Vector3((float)player2Rect.center.x, -(float)player2Rect.center.y, 0);

//Set gameObjects rotation
players[0].transform.rotation = Quaternion.Euler(0, 0, (float)player1Rect.angle+90f);
players[1].transform.rotation = Quaternion.Euler(0, 0, (float)player2Rect.angle+90f);
```

Figuur 25 Overzetten van de detectie data naar game objecten welke in Unity zichtbaar zijn.

Wat hier (Figuur 25) gebeurt is dat de data uit de spelerslijst, welke ik bijwerk in Figuur 24, overgezet wordt naar game objecten welke in de Unity Editor zichtbaar zijn. Ik heb gekozen voor een rode en een blauwe pijl. De keuze voor een pijl is gemaakt om te kijken hoe de rotatie data van de vierhoeken zich gedraagt. De reden dat ze twee aparte kleuren hebben is om te kijken of de twee gedetecteerde vierhoeken hun eigen positie behouden.

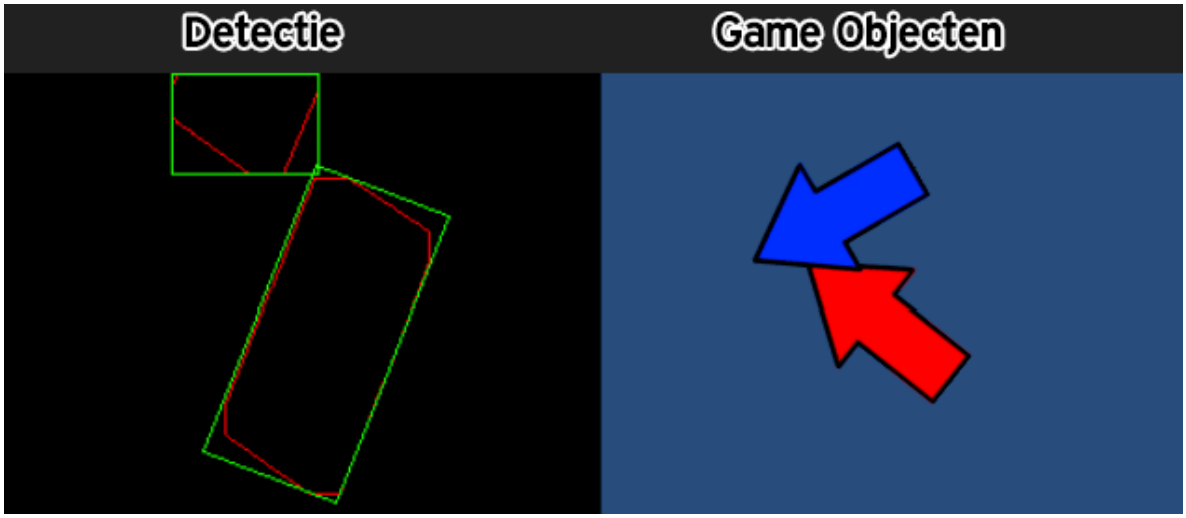
Resultaten FindContours in Unity op game objecten

Met de rotatie en positie vertaald naar twee duidelijk te volgen objecten binnen Unity heb ik voor het eerst de detectie data echt gebruikt. Mijn doel hier van was om te zien wat de ruwe detectiedata opleverde. In het ideale scenario blijven de game objecten op de plek van waar in het bronbeeld de roze blokken liggen.
Het bronbeeld is niet zichtbaar in de volgende voorbeelden; Dit is, omdat het tonen van videobeelden in Unity welke uitgelezen worden binnen OpenCV een conversie nodig heeft. De beelden worden uitgelezen en gebruikt binnen OpenCV, maar om deze in de Unity editor/speler te kunnen zien moet je deze pixel voor pixel omzetten naar een *texture*. Dit kost rekenkracht en tijd, vandaar dat ik alleen de detectie omzet, omdat dit het belangrijkste beeld is om bij te houden.



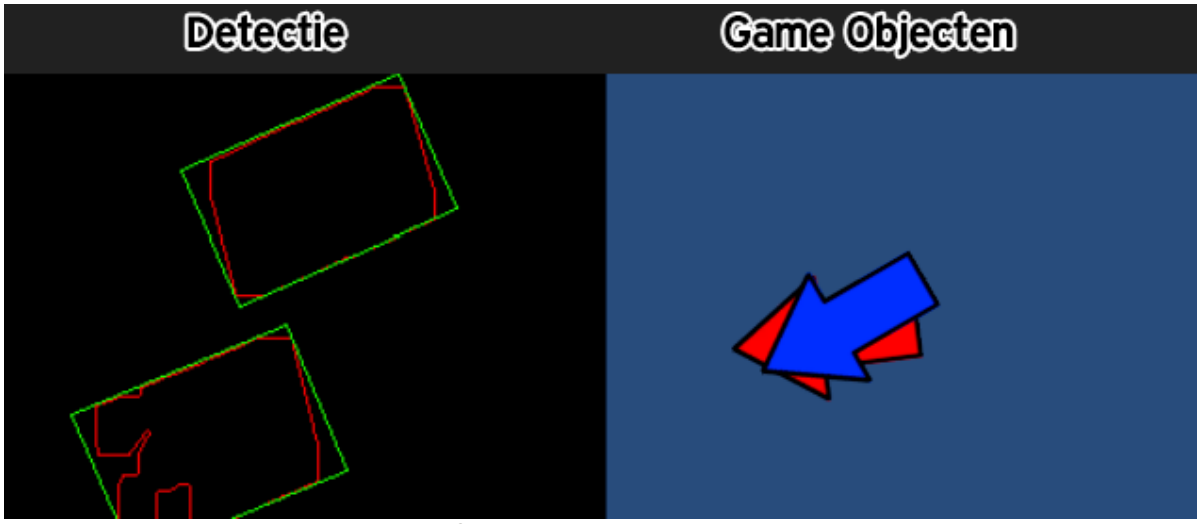
Figuur 26 De applicatie bij het opstarten.

Bij het opstarten van de applicatie (Figuur 26) ziet alles er goed uit. De game objecten (rechts) nemen de positie van de gedetecteerde vierhoeken aan en blijven ook bij de eerste bewegingen goed het bronbeeld volgen.



Figuur 27 De applicatie na een korte tijd draaien.

Even verder in de video verdwijnt één rechthoek deels uit beeld (Figuur 27). Dit zorgt er meteen voor dat de rotatie van de rechthoek niet meer overeenkomt met de rotatie van het blok in het bronbeeld. Ook al is het verdwijnen uit beeld maar kort, omdat het ruwe tracking data betreft wordt dit gelijk overgezet op de game objecten.



Figuur 28 De applicatie waarbij de video bijna afgelopen is

Verderop in de video treedt er nog meer storing op. De gedetecteerde rechthoeken verliezen vaak voor een paar *frames* hun positie en/of rotatie en dat wordt direct overgezet op de game objecten.

Conclusie FindContours in Unity op game objecten

Zoals bij het implementeren van de FindContours methode binnen C++ zonder enige verder gebruik al bleek is storing binnen de tracking data een probleem. Ook al lijken de rechthoeken bij het bekijken van het detectie beeld (Figuur 26 t/m Figuur 28) bruikbaar als speler data voor de beweegvloer; is dit niet het geval. Zodra je de data van deze rechthoeken overzet op losse objecten, wat gedaan is in Unity, zie je hoe groot een probleem de ruis is. Als we dit vertalen naar een toepassing binnen de beweegvloer dan ontdekken we hoe groot dit probleem is. Stel dat een speler voor de beweegvloer punten kan scoren door op een bepaalde plek in het beeld te gaan staan. Een kind leest bijvoorbeeld het woord schaap en moet hierbij op een geprojecteerd schaap gaan staan. Met de implementatie die hierboven in 2.4.3 gemaakt is kan het zijn dat door storing in de tracking data de speler binnen de game op een andere plek uitkomt dan waar hij daadwerkelijk staat. In dit geval rekent het spel de actie van de speler als fout (hij is immers volgens het spel niet op de goede plek gaan staan) ook al staat de speler daar.
Dit is een scenario dat we niet willen hebben. Daarom moet de tracking data gefilterd worden. Als er ruis optreedt, grote veranderingen in de tracking data die niet kloppen, dan moet dit eruit gefilterd worden. Hier zijn algoritmes voor welke je toe kunt passen op de tracking data. De volgende stap is dan ook het onderzoeken welke algoritmes nodig zijn om de tracking data te filteren tot bruikbare data.

2.5 Object Tracking Algoritmes

2.5.1 Oriëntatie

Zoals uit de meest recente implementatie gebleken is, is puur het gebruik maken van een detectiemethode niet genoeg om tot daadwerkelijke tracking te komen. Wat het gebruik van een detectiemethode inhoudt is dat je per frame in je video op zoek gaat naar objecten, deze detecteert en vervolgens voor dat frame de positie overzet op je applicatie. Het resultaat is dat wanneer er ruis optreedt dit 1 op 1 overgezet wordt op je applicatie. Dit is

goed te zien in Figuur 28. De oplossing is het gebruik maken van algoritmes welke aan de hand van de detectiedata een logisch pad maken welke te gebruiken is in je applicatie.
Ik ben online op zoek gegaan naar oplossing onder de term "*multiple object tracking algorithm".* Al snel kwam de Kalman filter langs in verschillende artikelen; zoals bij deze tutorial van Matlab: (MathWorks, 2012) en in een paper waar het genoemd wordt als bestaande techniek (Jerome Berclaz, 2011). Ik ben gaan bekijken wat de Kalman filter doet en of deze bruikbaar is voor mijn probleem (het afhandelen van ruis in de detectie).

2.5.2 Kalman Filter

De Kalman filter is toe te passen op systemen waar je als het ware informatie mist, maar wel een goede gok kan doen (op basis van voorafgaande data) waar het systeem zich in de volgende stap gaat bevinden. Een Kalman filter is ideaal voor een systeem wat constant verandert. Het filter is niet zwaar op het geheugen zien er maar één staat behouden hoeft te worden, de vorige staat van het systeem. Ook werkt de Kalman filter snel wat hem toepasbaar maakt voor real time systemen (Babb, 2015).
Als we kijken naar het tot nu toe gebouwde tracking systeem dan hebben we tracking data vanuit de FindContours methode. Elke keer wanneer een contour gedetecteerd wordt dan krijgen we een positie van de gevonden contouren. Dit is onze bekende data. Stel dat een contour uit het scherm verdwijnt of bedekt wordt dan willen we alsnog wel de positie van onze objecten binnen de applicatie bij blijven werken. De Kalman filter kan aan de hand van eerdere data een nieuwe positie voorspellen. Het filter maakt niet alleen gebruik van de vorige staat van het systeem voor zijn voorspelling, er wordt ook rekening gehouden met externe invloeden; bijvoorbeeld dat ik het blad met de roze blokken ineens 2* zo snel verschuif voor een seconde. Daar naast houdt het filter ook rekening met externe factoren welke onzeker zijn, bijvoorbeeld wind welke het blaadje even weg laat waaien, ook dit wordt gemodelleerd in de Kalman filter (Babb, 2015).
Het mooie is, is dat de Kalman filter als functie binnen OpenCV zit. Je hoeft hem dus niet zelf uit te schrijven en kan hem direct implementeren. Om deze reden schrijf ik ook niet de hele werking van het algoritme uit. Het gaat er wiskundig diep op in en gezien je het toe kan passen als functie binnen OpenCV is het gebruiken er van genoeg. Wat het Kalman filter op moet lossen zijn de schokkende bewegingen in de tracking oplossing. Door middel van de voorspelling van het filter moet dit veranderen in een soepele beweging. Voor de beweegvloer betekent dit dat je als speler niet ineens op het speelveld van positie verandert terwijl je zelf misschien maar een kleine stap zette.

2.5.3 Hungarian Algoritme

Naast de schokkende bewegingen is er nog een ander probleem binnen de tracking oplossing. Dit is dat de game objecten meerdere malen in de tests van positie wisselden. Dit komt, doordat de FindContours methode geen enkele rekening houdt met eerder gedetecteerde objecten. Als de methode even voor een frame al zijn contouren kwijt is en ze daarna weer vindt dan is het goed mogelijk dat de posities van de gevonden contouren omgewisseld zijn. De contour die eerst op plek 1 in de lijst zat kan op die manier na een nieuwe detectie op plek 2 in de lijst zitten. Dit terwijl de posities van de gedetecteerde objecten in het veld niet veel verandert hoeven zijn. Ook hier is een oplossing voor te vinden in de vorm van een algoritme. Bij het zoeken naar een *"multiple object tracking algorithm"* kwam ik een oplossing tegen welke gebruikt maakt van beide een Kalman filter

en het *Hungarian* algoritme (Son, 2015). Het Kalman filter zijn we al bekend mee, maar het Hungarian algoritme nog niet.

Uitleg Hungarian algoritme

Stel dat je een aantal werknemers hebt en een aantal opdrachten. Voor elke opdracht heeft elke werknemer een eigen prijs wat het kost om de opdracht te doen. Het doel is om alle opdrachten uit te voeren op de goedkoopst mogelijke manier. Hiervoor kun je het Hungarian Algoritme toepassen, deze gaat op zoek naar de goedkoopste oplossing. De reden dat dit handig is voor de tracking oplossing is dat de werknemers uit het voorbeeld de spelers van de beweegvloer zijn, de te tracken objecten. De opdrachten in het voorbeeld zijn de gevonden posities door de FindContours methode. Wanneer elke speler gedetecteerd is dan is het aan het Hungarian algoritme om ervoor te zorgen dat elke speler op zijn plek blijft in de lijst (behoudt van identiteit) en niet wisselt van positie met een andere speler. Het algoritme moet ervoor zorgen dat de koppeling tussen de speler op de vloer en de detectie van die speler behouden blijft wanneer er meerdere spelers zijn.

Het algoritme is relatief simpel in zijn werking. Stel je hebt 4 werknemers en 4 opdrachten met voor elke opdracht een prijs per werknemer. Dit zorgt voor de volgende matrix:

	1	**2**	**3**	**4**
1	90	75	75	80
2	35	85	55	65
3	125	95	90	105
4	45	110	95	115

Figuur 29 Kosten matrix, (Suriyakula, 2013) slide 3

Stap 1. Vind voor elke rij het laagste getal van die rij en trek dit van alle getallen in die rij af.
Resultaat na stap 1:

15	**0**	**0**	**5**
0	50	20	30
35	5	0	15
0	65	50	70

Figuur 30 Matrix na stap 1, (Suriyakula, 2013) slide 5

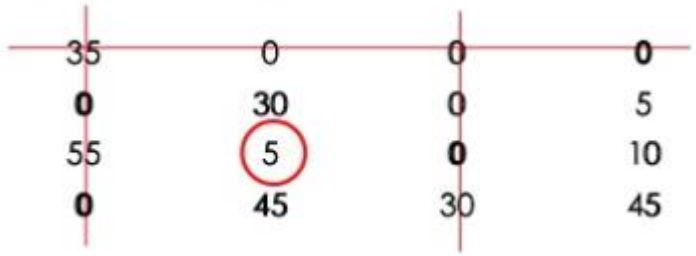
Stap 2. Herhaal stap 1, maar doe het ditmaal voor alle kolommen
Stap 3. Teken een minimaal aantal lijnen over de rijen en kolommen welke alle nullen bedekken.
Het resultaat na stap 3 is:

15	**0**	**0**	**0**
0	50	20	25
35	5	**0**	10
0	65	50	65

Figuur 31 De matrix na stap 3, alle nullen zijn bedekt met een minimaal aantal lijnen. (Suriyakula, 2013) slide 6

Stap 4. Wanneer het aantal lijnen gelijk is aan het aantal rijen in de matrix dan is er een optimale oplossing gevonden. In dit geval is het aantal lijnen 3 en het aantal rijen 4, dus de optimale oplossing is nog niet gevonden (Figuur 31).

Mits er nog geen oplossing gevonden is dan voeren we Stap 5 uit.
Stap 5. Vind het laagste getal wat niet doorstreept is. In dit geval 20. Dit getal halen we af van alle niet doorstreepte getallen en tellen we op bij alle dubbel doorstreepte (gekruiste) getallen. Het resultaat hiervan is; inclusief Stap 3 weer:

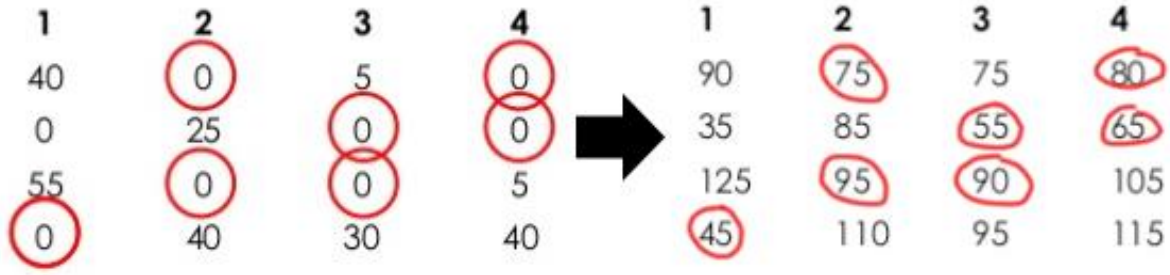


Figuur 32 Resultaat na Stap 5, het aantal lijnen is weer niet 4 met als laagste getal 5. (Suriyakula, 2013) slide 8

Het aantal lijnen is weer 3 en het laagste getal is ditmaal 5 (Figuur 32).
We herhalen stap 5 met het getal 5 met als resultaat de volgende matrix:

Figuur 33 Matrix na herhaling stap 5. (Suriyakula, 2013) slide 9

Nu hebben we 4 lijnen, het aantal dat we zoeken (Figuur 33).



Figuur 34 Mogelijke oplossingen.

De plek van de nullen bepalen de mogelijke oplossingen (Figuur 34).
Kies voor elke rij of kolom maar een enkel getal om tot een uiteindelijke oplossing te komen. Dit geeft ons de oplossingen:
75+65+90+45=275
80+55+95+45=275

Voor de uitleg van het Hungarian algoritme heb ik gebruikt gemaakt van de slides van Charith Suriyakula, (Suriyakula, 2013).

2.5.4 Implementatie Algoritmes in Unity3D & OpenCV

Zoals eerder vernoemd ben ik tijdens het onderzoeken van de algoritmes een oplossing tegengekomen welke beide Kalman en Hungarian toepaste. Dit was een project wat op GitHub stond en in Java geschreven was (Son, 2015). Het is een complete tracking oplossing geschreven als los programma. Gezien er niet heel veel speelruimte meer was om verschillende losse dingen te implementeren heb ik mij ingezet om deze tracking code te vertalen naar C# en in een Unity project te implementeren. Uiteraard gebruikt het project ook OpenCV, anders had het het niet zo makkelijk gebruikt kunnen worden.
Een andere reden dat ik gekozen heb voor het gebruiken van deze bestaande code is dat de twee gevonden algoritmes nog relatief complex zijn om uit te pluizen. Om te zorgen dat we tot een conclusie kunnen komen voor dit onderzoek heb ik daarom gekozen deze bestaande code te gebruiken. Dit in de hoop dat ik ondanks niet volledig de achterliggende wiskundige werking te weten toch bruikbare data voor mijn onderzoek vergaren kon.
Het nadeel van het gebruiken van andermans code is dat het niet zelf geschreven is en dus ook fouten lastiger op te sporen zijn; gezien het een flinke berg code is waar doorheen gegaan moet worden en niet alle functies de werking volledig duidelijk zijn.

Implementatie algoritmes

Uiteindelijk heb ik het werkend gekregen binnen Unity met als grootste kanttekening dat er nog een fout zit in de code van het Hungarian algoritme. De fout zit in het opstellen en doorrekenen met de kosten matrix, maar het produceert al wel bruikbare resultaten met een duidelijke verbetering tegenover de vorige oplossing uit hoofdstuk 2.4.3

```
1.
//Find minAreaRects with rotation and shit (we need these).
for (int i = 0; i < contours.Count; i++)
{
    //Only assign a rotatedRect if the contour is big enough (exceeds minimum size).
    if (contours[i].size().area() > 25)
    {
        array.Add(Imgproc.boundingRect(contours[i]));
    }
}

2.
foreach (OpenCVForUnity.Rect item in array)
{
    int ObjectCenterX = (int)((item.tl().x + item.br().x) / 2);
    int ObjectCenterY = (int)((item.tl().y + item.br().y) / 2);

    Point pt = new Point(ObjectCenterX, ObjectCenterY);
    detections.Add(pt);
}

3.
if (array.Count > 0)
{
    print("<color=blue>Detections length: " + detections.Count + "  --  Array length: " + 
    tracker.update(array, detections, imag);
}

4.
else if (array.Count == 0)
{
    tracker.updateKalman(imag, detections);
}
```

Figuur 35 Code waar alle functies van de tracker ((Son, 2015)) aangeroepen worden.

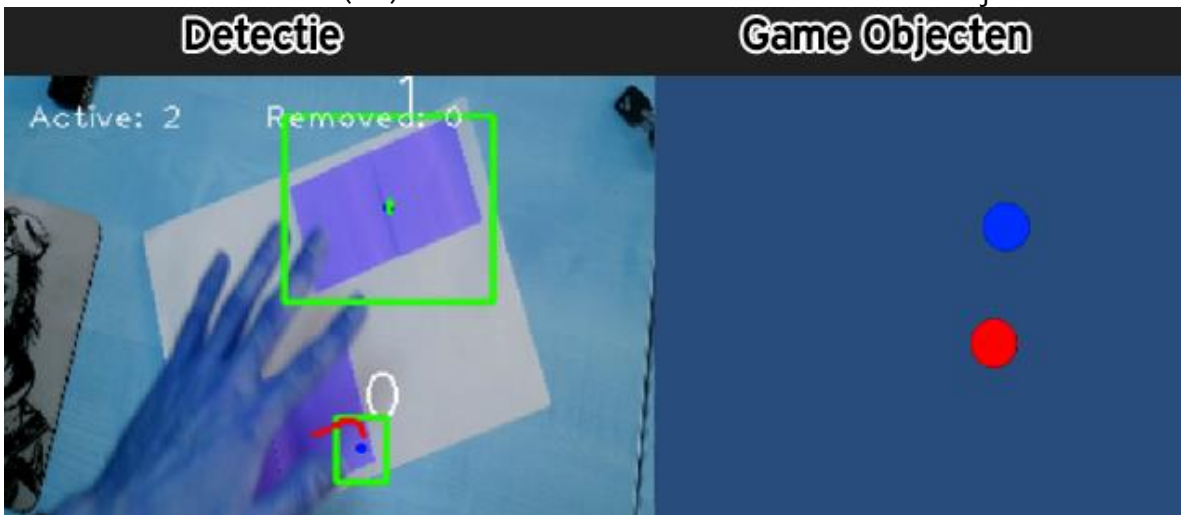
Globaal werkt de tracking code als volgt:

- **1 & 2.** Zodra de FindContours methode een contour vindt meldt hij deze aan bij de tracker.
- **3.** De positie van elke track wordt bijgehouden aan de hand van een Kalman filter en de bekende data uit de detectie.
- De tracker zorgt met behulp van het Hungarian algoritme dat elke *track* zijn eigen ID behoudt.
- **4.** Wanneer een track geen data binnenkrijgt wordt de positie alleen aan de hand van het Kalman filter bijgewerkt en wanneer deze track te lang uit beeld is wordt hij verwijdert.

Resultaten implementatie algoritmes

Het resultaat is een soepelere tracking dan bij mijn oplossing in 2.4.3 De game objecten verschieten niet meer schokkerig van positie. De reden dat de rotatie niet meegenomen is zoals bij het vorige voorbeeld is dat dit extra ruis gaf en rotatie voor nu nog niet relevant is bij de beweegvloer. Rotatie detecteren op mensen van boven gaat sowieso erg lastig zonder enig hulpmiddel dus om die reden valt dat voor nu af.

Zoals te zien is op Figuur 36 blijft de positie van het rode game object gelijk aan waar de tracker het onderste blok (#0) detecteert ondanks dat dit bedekt is met mijn hand.

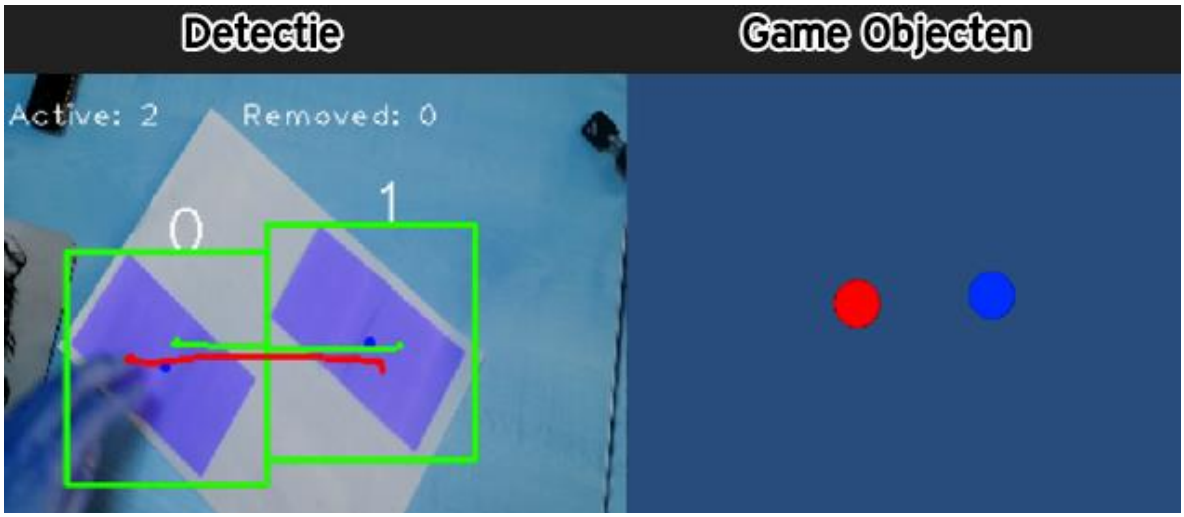


Figuur 36 Eén object wordt bedekt, tracking gaat door.

Op Figuur 37 valt te zien dat wanneer de gedetecteerde objecten uit het scherm verdwijnen de game objecten op hun laatst bekende positie blijven. Gezien de objecten maar kort uit het beeld zijn verdwijnen de tracks niet en gaat het tracken door zodra de objecten terug in beeld komen.

Figuur 37 De te tracken objecten zijn buiten het scherm, de game objecten blijven in positie.

De huidige implementatie in Unity van de tracker van (Son, 2015) is nog niet zonder fouten. De grootste fout die ik tegengekomen ben is dat wanneer de rechthoeken horizontaal op één lijn liggen ze van ID wisselen (Figuur 38). Dit komt hoogstwaarschijnlijk door een fout in het berekenen van de kosten matrix binnen het Hungarian algoritme.



Figuur 38 Bug: Wanneer beide objecten op één horizontale lijn liggen wisselen ze van positie.

Conclusie algoritmes

De resultaten van de eerste implementatie van een Kalman filter samen met het Hungarian algoritme zien er goed uit. De tracking data lijkt soepeler op de game objecten dankzij de Kalman filter en ook blijft de volgorde bewaard afgezien van de bug te zien in Figuur 38.
De code werkt nog niet foutloos. Er vinden een aantal fouten plaats tijdens het draaien van de applicatie te zien in Figuur 36 & Figuur 38. Gezien ik tegen het einde van mijn stageperiode aanliep was er geen tijd meer om deze fouten helemaal uit te pluizen en op te lossen. Het nadeel hier was ook dat ik de code van een ander gebruikt heb om tot deze oplossing te komen, dit was ook mede door de korte tijd die ik nog had.
Wel is duidelijk aan de hand van deze oplossing dat het gebruiken van de behandelde algoritmes meerwaarde heeft voor de beweegvloer. De fouten die in hoofdstuk 2.4 optraden kunnen we ons niet veroorloven bij de beweegvloer en de algoritmes behandelt in dit hoofdstuk, 2.5, zijn een mogelijke oplossing hiervoor.

Conclusie

Mijn onderzoeksvraag luidde als volgt:
Welke mogelijke techniek voor tracking gaan we gebruiken voor de beweegvloer en hoe implementeren we deze technieken in Unity3D?

De toegepaste technieken vallen binnen computer vision. Computer vision is de automatische analysering van afbeeldingen en video door computers om hier informatie uit te halen (Dawson-Howe, 2014, p. 1). Voor het toepassen van computer vision is gebruik gemaakt van de software library OpenCV. OpenCV is een veel gebruikte computer vision library en is breed inzetbaar zoals geconcludeerd in hoofdstuk 2.2.
Gezien tracking algoritmes het beste werken met een binair beeld, bestaande uit zwarte en witte pixels, is een gefilterd bronbeeld van groot belang om tracking uit te voeren zoals gebruikt door (Solderspot, 2014). Om deze reden valt een reguliere camera met kleurenbeeld af, omdat de variabele hier te groot zijn. Variabele zoals lichtintensiteit, kleur etc. verschillen allemaal per mogelijke locaties waar de beweegvloer moet komen in de toekomst. Nodig is één oplossing die op alle locaties werkt. Het beeld van een dieptesensor zoals die in de Microsoft Kinect is een solide oplossing wat consistentie betreft, omdat hier een neutraal beeld uit komt (Davison, 2012). Hierbij heb je minder last van ruis, het licht in de ruimte maakt niet uit en je hoeft de spelers van de Beweegvloer niet te voorzien van speciale tracking accessoires. De bruikbaarheid van Kinect is bewezen in project Tikkertje 2.0 (Poppe, 2017), (Universiteit Twente, 2015). Je gebruikt het beeld dat uit de dieptesensor komt om de detectie en tracking algoritmes op los te laten. Dat de spelers van de beweegvloer geen speciale acties hoeven te ondernemen om mee te kunnen doen is een zeer belangrijk voor Springlab punt om de beweegvloer op de markt te brengen.

Spelers worden gedetecteerd door middel van de FindContours methode binnen OpenCV. Deze methode geeft de meeste vrijheid in het oppakken van te detecteren vormen zoals geconcludeerd in paragraaf 2.3.4. De positie informatie die vloeit uit de gevonden contouren (positie van de gevonden objecten) wordt versoepeld met een Kalman filter (Babb, 2015). Dit zorgt ervoor dat de spelers vloeiend bewegen in de game, gezien de detectie data nog ruis kan bevatten. Om ervoor te zorgen dat de spelers in de tracking data niet van positie wisselen wordt het Hungarian algoritme toegepast (HungarianAlgorithm.com, 2013). Mijn implementatie van het Hungarian Algoritme is nog niet getest met daadwerkelijke spelers, dit is een belangrijke stap die gemaakt moet worden bij het bouwen van een beweegvloer prototype.
De detectie en tracking algoritmes worden gebruikt in Unity3D i.c.m. de OpenCV For Unity Asset. Er is voor de OpenCV For Unity Asset gekozen gezien dit pakket de snelste oplossing is om OpenCV in Unity3D te gebruiken zoals geconcludeerd in paragraaf 2.4.2. Het voordeel van deze asset is ook dat alles op één computer kan draaien (de game in Unity en de detectie/tracking in OpenCV binnen Unity) wat de kosten laag houdt voor de beweegvloer.

Aanbevelingen

Om tot een werkend prototype voor de beweegvloer te komen zou ik gebruik maken van mijn implementatie van FindContours samen met de Kinect sensor als bronbeeld. Op gebied van tracking algoritmes is nog speelruimte. Definitief gaan voor het Hungarian algoritme om de identiteit van elk gedetecteerd object te waarborgen is niet per sé nodig gezien het nog niet uitvoerig getest is met echte spelers, maar het kan al genoeg zijn voor een speelbaar prototype. Het Kalman filter is nuttig om de beweging soepeler te maken, maar het belangrijkste is een tracking algoritme testen op de detectie van echte spelers. Door middel van het gebruik van OpenCV For Unity is het daarna heel makkelijk om snel een werkend spel te ontwikkelen rondom de tracking data.

Begrippenlijst

Blob (in computer vision) - Een regio binnen een plaatje/beeld met samenhangende kenmerken.

Captcha – Een computerprogramma of systeem bedoeld om mensen door te laten en computer input tegen te houden. Vaak gebruikt tegen spam en/of om data te vergaren.

C# - Programmeertaal, hoger *level* dan C++ wat inhoudt dat er meer taken afgevangen worden door de computer.

C++ - Programmeertaal, lager *level* dan C# wat inhoudt dat je meer controle hebt over wat de code doet; bijvoorbeeld geheugen beheer.

Dilate methode - Methode binnen OpenCV welke felle pixels in een plaatje uitbreidt/laat groeien.

Erode methode - Methode binnen OpenCV welke felle pixels in een plaatje afbrokkelt/kleiner maakt.

FindContours methode – Methode binnen OpenCV welke op zoek gaat naar contouren in een plaatje.

Frame/Frames – Losse beelden uit een videobeeld. Eén video bestaat uit meerdere frames.

Gaussian Blur – Methode om een plaatje waziger te maken. Beschikbaar binnen OpenCV.

Hungarian algoritme – Algoritme om **N** werkers, welke ieder hun eigen kosten per opdracht hebben, **N** opdrachten zo goedkoop mogelijk uit te laten voeren.

HSV – Kleurenmodel; HSV staat voor *Hue, Saturation, Value*.

inRange methode – Methode binnen OpenCV om een plaatje naar binair te converteren. De functie zet pixels die binnen de opgegeven waardes vallen op wit, de pixels die erbuiten vallen gaan op zwart.

Java – Programmeertaal, vergelijkbaar met C# qua syntax.

Kalman filter – Algoritme om de toekomstige staat van een systeem te voorspellen aan de hand van reeds bekende data. Beschikbaar als functie binnen OpenCV.

Library – Binnen de informatica is een library (Engels voor bibliotheek) een verzameling functies en methoden welke de programmeur kan toepassen. Het voordeel van een library is dat je voorgeprogrammeerde code kunt toepassen.

Microsoft Kinect – Camera met verscheidene sensoren om beweging te detecteren. Origineel een accessoire voor de Xbox 360 van Microsoft. Ook bruikbaar op de computer.

OpenCV - Open Source Computer Vision library.

OptiTrack – Bestaand systeem voor *motion capture*; Het vastleggen van bewegingen met behulp van camera's en markers.

PlayStation Eye – Camera van Sony welke beweging kan detecteren.

PlayStation Move – Accessoire voor de Sony PlayStation om bewegingsinteractie te hebben in games.

Plugin – Een plugin is een aanvulling op een bestaand stuk software. Een plugin voeg je als het ware toe aan een bestaand programma met als reden om de functionaliteit uit te breiden.

Port – Translatie/overzetting van een stuk software van bijvoorbeeld de ene programmeertaal naar de andere of van de ene ontwikkelomgeving naar een andere.
Tracking/tracken – Het volgen van objecten; volgen houdt in het bijhouden van de locatie van gevolgde object.

Tutorial – (Vaak) online stap voor stap uitleg om iets specifieks te maken/ontwikkelen. Kan in geschreven vorm zijn of in videoformaat.

Unity(3D) – Software pakket voor het ontwikkelen van games & applicaties.

Unity Asset Store – Webwinkel van Unity Technologies waar je zogeheten *assets* kan kopen voor Unity. Dit zijn pakketten of *plugins* welke je toe kan voegen aan je Unity project.

Visual Studio – Software ontwikkelpakket van Microsoft.

Bronnenlijst

AForge.NET. (2013, July 17). *AForge.NET Framework 2.2.5 is now available.* Retrieved from AForge.NET: http://www.aforgenet.com/news/2013.07.17.releasing_framework_2.2.5.html

AForge.Net. (2013). *AForge.NET Framework Documentation.* Retrieved from AForge.Net: http://www.aforgenet.com/framework/docs/

Babb, T. (2015, August 11). *How a Kalman filter works, in pictures*. Retrieved from Bzarg: http://www.bzarg.com/p/how-a-kalman-filter-works-in-pictures/

Berge, T. v. (2017, March 5). *Unity and OpenCV - Part Three: Passing detection data to Unity.* Retrieved from Thomas Mountainborn: Creative development & Unity assets: http://thomasmountainborn.com/2017/03/05/unity-and-opencv-part-three-passing-detection-data-to-unity/

Bergevin, O. L. (2010, June 2). *Detection and Identification of Animals Using Stereo Vision.* Retrieved from University of Edinburgh School of Informatics: http://homepages.inf.ed.ac.uk/rbf/VAIB10PAPERS/levesquevaib.pdf

Davison, D. A. (2012, September 27). *Chapter 2.1. Charting the Depth Map*. Retrieved from fivedots.coe.psu.ac.th: http://fivedots.coe.psu.ac.th/~ad/kinect/ch021/

Dawson-Howe, K. (2014). *A Practical Introduction to Computer Vision with OpenCV.* Dublin: Wiley.

Emgu Corporation. (2017, May 11). *Emgu CV v3.x.* Retrieved from Unity3D Assetstore: https://www.assetstore.unity3d.com/en/#!/content/24681

Emgu Corporation. (2017, June 5). *EmguCV Wiki Main Page.* Retrieved from EmguCV Wiki: http://www.emgu.com/wiki/index.php/Main_Page

Enox Software. (2016, February 25). *OpenCVForUnity.Imgproc Class Reference*. Retrieved from OpenCV for Unity based on OpenCV2.4.11: http://enoxsoftware.github.io/OpenCVForUnity/doc/html/class_open_c_v_for_unity_1_1_imgproc.html#acf8b3a13358fd6cdbc17059281afca9f

Enox Software. (2017, July 11). *OpenCV for Unity*. Retrieved from Unity Asset Store: https://www.assetstore.unity3d.com/en/#!/content/21088

Faragher, R. (2012, September). *Understanding the Basis of the Kalman Filter.* Retrieved from University of Cambridge: http://www.cl.cam.ac.uk/~rmf25/papers/Understanding%20the%20Basis%20of%20the%20Kalman%20Filter.pdf

Google. (2017). *Recaptcha FAQ*. Retrieved from Support.Google.Com: https://support.google.com/recaptcha/?hl=en

Google. (2017). *The Recaptcha Advantage*. Retrieved from Google.com: https://www.google.com/recaptcha/intro/invisible.html#the-recaptcha-advantage

HungarianAlgorithm.com. (2013). *The Hungarian Algorithm: An example.* Retrieved from Hungarian Algorithm: http://www.hungarianalgorithm.com/examplehungarianalgorithm.php

Jerome Berclaz, F. F. (2011). *Paper: Multiple Object Tracking using K-Shortest Paths Optimization.* Retrieved from Idiap Research Institute: http://www.idiap.ch/~fleuret/papers/berclaz-et-al-tpami2011.pdf

Kaehler, G. B. (2008). *Learning OpenCV.* Boston: O'Reilly Media, Inc.

Leone, G. (2017, May 12). *What is Computer Vision and Why Is It Important?* Retrieved from Kairos Human Analytics Blogs: https://www.kairos.com/blog/what-is-computer-vision-and-why-is-it-important

Lowe, S. (2010, September 17). *The Tech Behind PlayStation Move*. Retrieved from IGN.com: http://www.ign.com/articles/2010/09/18/the-tech-behind-playstation-move

Mallick, S. (2015, February 17). *Blob Detection Using OpenCV*. Retrieved from LearnOpenCV.com: https://www.learnopencv.com/blob-detection-using-opencv-python-c/

Mallick, S. (2017, February 2013). *Object Tracking using OpenCV*. Retrieved from Learn OpenCV: https://www.learnopencv.com/object-tracking-using-opencv-cpp-python/

MathWorks. (2012). *Motion-Based Multiple Object Tracking*. Retrieved from Mathworks.com: https://nl.mathworks.com/help/vision/examples/motion-based-multiple-object-tracking.html?requestedDomain=www.mathworks.com

OpenCV Development Team. (2017). *Creating Bounding rotated boxes and ellipses for contours*. Retrieved from OpenCV Tutorials: http://docs.opencv.org/2.4/doc/tutorials/imgproc/shapedescriptors/bounding_rotated_ellipses/bounding_rotated_ellipses.html

OpenCV Development Team. (2017). *Eroding and Dilating.* Retrieved from OpenCV Documentation: http://docs.opencv.org/2.4/doc/tutorials/imgproc/erosion_dilatation/erosion_dilatation.html

OpenCV Development Team. (2017). *SimpleBlobDetector Class Reference*. Retrieved from OpenCV Documentation: http://docs.opencv.org/trunk/do/d7a/classcv_1_1SimpleBlobDetector.html

OpenCV Development Team. (2017). *Structural Analysis and Shape Descriptors*. Retrieved from OpenCV Documentation: http://docs.opencv.org/2.4/modules/imgproc/doc/structural_analysis_and_shape_descriptors.html

OpenCV team. (2017). *OpenCV About Page*. Retrieved from Opencv.org: http://opencv.org/about

Poppe, R. (2017, 5 4). Interview Ronald Poppe over Tikkertje 2.0. (T. Nap, Interviewer)

Shuman, S. (2010, September 7). *PlayStation Move: The Ultimate FAQ*. Retrieved from blog.us.playstation.com: https://blog.us.playstation.com/2010/09/07/playstation-move-the-ultimate-faq/

SMALLab (Director). (2011). *SMALLab Embodied Learning Environment* [Motion Picture].

SMALLab Learning. (2017). *SMALLab Embodied Learning Environment*. Retrieved from SMALLab Learning: http://smallablearning.com/product/smallab-environment/

Solderspot. (2014, October 18). *Using OpenCV for simple object detection*. Retrieved from Solderspot.wordpress.com: https://solderspot.wordpress.com/2014/10/18/using-opencv-for-simple-object-detection/

Son, K. D. (2015, June 1). *Moving-Target-Tracking-with-OpenCV*. Retrieved from Github.com: https://github.com/Franciscodesign/Moving-Target-Tracking-with-OpenCV

Sony Computer Entertainment Inc. (2010, February 22). *flickr.com.* Retrieved from Flickr: https://www.flickr.com/photos/playstationblog/4930864078/

Suriyakula, C. (2013, August 21). *Hungarian algorithm.* Retrieved from www.slideshare.net: https://www.slideshare.net/charith4/hungarian-algorithm

Suzuki, S. (1985). *COMPUTER VISION, GRAPHICS, AND IMAGE PROCESSING.* Retrieved from xuebalib.com: http://download.xuebalib.com/xuebalib.com.17233.pdf

Szeliski, R. (2010). *Computer Vision: Algorithms and Applications.* Springer.

Thakkar, K. (2012, November 21). *OpenCV vs. MATLAB — An insight*. Retrieved from Karan Jitendra Thakkar: https://karanjthakkar.wordpress.com/2012/11/21/what-is-opencv-opencv-vs-matlab/

Universiteit Twente. (2015, 04 14). *Children with "Shields"*. Retrieved from University of Twente: hmi.ewi.utwente.nl/IUALL/demos/interactive-tag-playground/img_5018/

Universiteit Twente. (2015). *Interactive Tag Playground*. Retrieved from UTwente.nl: http://hmi.ewi.utwente.nl/IUALL/demos/interactive-tag-playground/

Wikimedia. (n.d.). *Wikimedia Hue Scale.* Retrieved from Wikimedia.org: https://upload.wikimedia.org/wikipedia/commons/thumb/a/ad/HueScale.svg/2000px-HueScale.svg.png

Willow Garage. (2015). *Willow Garage Software - OpenCV.* Retrieved from Willow Garage: http://www.willowgarage.com/pages/software/opencv