KU Leuven
Departement Computerwetenschappen

P&O: Computerwetenschappen

Eindverslag

Team:
Zilver

Bram Vandendriessche (Coördinator)
Arne Vlietinck (Secretaris)
Matthias Van der Heyden
Jef Versyck
Vincent Vliegen
Laura Vranken

Academiejaar 2016-2017

Samenvatting

Auteur: Arne Vlietinck

Dit verslag behandelt het ontwerp en de implementatie van een Autopilot en Virtual Testbed voor een drone. In de simulator is voor de eerste mijlpaal een rode bol in een witte ruimte te zien. Bij de tweede mijlpaal wordt deze wereld uitgebreid met een windkracht in een willekeurige richting. Voor de derde mijlpaal zijn er verschillende bollen zichtbaar die allemaal op een zo efficiënt mogelijke manier moeten worden doorprikt. Bij de laatste mijlpaal zijn niet alle bollen direct zichtbaar. Een uitbreiding is het ontwijken van grijze obstakels.
De Autopilot zorgt voor een correcte aansturing van de drone. Hij bepaalt de vliegroute op basis van informatie verkregen van twee camera's die zich op de drone bevinden. Hierbij is de relatieve plaatsbepaling van de bol ten opzichte van de drone van belang. De Autopilot leidt daaruit de beweging van de drone af en laat de simulator deze uitvoeren.
Het programma wordt interactief gemaakt door het gebruik van *GUI's*. Deze geven de mogelijkheid het camerastandpunt te kiezen, extra bollen toe te voegen, ook de voltooiingsgraad, snelheid en positie van de drone worden weergegeven. Daarnaast kunnen externe factoren (wind en gravitatieconstante) aangepast worden.

Inhoudsopgave

Inleiding

Auteurs: Laura Vranken & Arne Vlietinck

Drones zijn de laatste jaren enorm in populariteit toegenomen en blijven hierdoor ook in positieve zin evolueren. Ze worden tegenwoordig gebruikt voor talloze toepassingen. De bekendste toepassing bevindt zich binnen Defensie, die drones gebruiken om informatie te verkrijgen over vijandelijk gebied zonder mensenlevens te moeten riskeren. Daarnaast hebben ook grote bedrijven (o.a. Amazon[1]) de weg naar deze technologie gevonden. De toekomst brengt echter nog veel meer voordelen. Enkele voorbeelden [3] zijn veiligheidsinspectie van windturbines of elektriciteitslijnen, luchtsteun bij zoek- en reddingsoperaties, bewaking en luchtfotografie.
Wanneer een drone autonoom functioneert, is een betrouwbare aansturing door de Autopilot van levensbelang. Hij moet namelijk bestand zijn tegen allerlei externe factoren (bv. wind, obstakels...).

Dit verslag behandelt de autonome aansturing van een drone, meer bepaald een quadcopter. Er wordt uitgegaan van een drone waarop twee voorwaarts gerichte camera's bevestigd zijn. Op basis van deze beelden moeten afstand en positie tegenover het doel ingeschat worden en nieuwe bewegingsopdrachten voor de drone gegenereerd worden. Deze bewegingen worden weergegeven in een Virtual Testbed [1]. Dit is een softwaresysteem dat een fysieke opstelling van een drone en camera's simuleert. De simulator genereert beelden van de drone uit verschillende standpunten a.d.h.v. de verkregen bewegingsopdrachten van de Autopilot.

De Autopilot en het Virtual Testbed moeten zo ontworpen worden dat de drone in staat is om zijn doel, een niet-grijze bol, te lokaliseren en ernaar toe te vliegen. Dit eventueel onder invloed van lichte wind in willekeurige richtingen. Bovendien moet ook voor beiden een *graphical user interface* of kortweg *GUI* ontworpen worden. De *GUI* toont de vooruitgang en laat de gebruiker toe allerlei informatie (snelheid, positie en verschillende camerastandpunten) op te vragen. Daarnaast kan de gebruiker nieuwe bollen toevoegen en de wind manueel aanpassen in de verschillende richtingen.

De tekst is als volgt opgebouwd. Eerst wordt het ontwerp van de Autopilot en Virtual Testbed verder uitgediept (sectie 1). Vervolgens wordt er ingegaan op de gebruikte algoritmes (sectie 2) en wordt de opbouw van onze software verduidelijkt (sectie 3). Ook is er extra informatie te vinden over de *GUI* (sectie 4). Afsluiten wordt er gedaan met de uitgevoerde testen te bespreken (sectie 5) en een kort besluit.

1 Ontwerp

Auteurs: Arne Vlietinck; redactie: Arne Vlietinck

De concepten die gebruikt worden om het softwaresysteem te implementeren zijn van cruciaal belang. Hieronder wordt het stappenplan dat de Autopilot volgt, verduidelijkt en wiskundig toegelicht. Ook wordt er dieper ingegaan op de gemaakte keuzes tijdens de ontwerpfase.

1.1 Drone Autopilot

Auteur: Laura Vranken

De Drone Autopilot bepaalt de positie van de drone relatief ten opzichte van zijn doel a.d.h.v. twee beelden gegeneerd door de dronecamera's. Bovendien zorgt de Autopilot ervoor dat de drone op een correcte manier naar zijn doel toe vliegt. De drone bereikt zijn doel wanneer hij door een gekleurde bol gevlogen is of door ze allemaal in geval van meerdere bollen. Daarnaast moet de Autopilot rekening houden met een mogelijke invloed van wind die de drone van zijn koers doet afwijken en obstakels waarrond hij moet vliegen.

[1] Amazon Prime Air

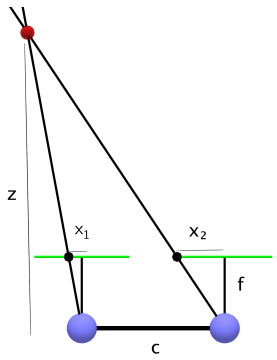
1.1.1 Beeldverwerking

Ten eerste moeten de beelden die de Autopilot van de Virtual Testbed binnenkrijgt, geanalyseerd worden. Dit gebeurt door iteratief de kleurwaarden van elke pixel te vergelijken met de waarde van de opgegeven kleur. Deze methode wordt logischerwijs enkel gebruikt indien er al een doelkleur beslist is. Anders zullen de pixels gegroepeerd worden per kleur die voorkomt in een *HashMap*. De gekleurde pixels worden bijgehouden door hun positie ten opzichte van het beeld, uitgedrukt in rij en kolom. De berekeningen worden gebaseerd op het midden van de bol. Dit kan benaderd worden op twee manieren: via het zwaartepunt of de kleinste-kwadratenmethode op de randpunten van de cirkel. Het zwaartepunt van een bepaalde kleur pixels is te berekenen via het gemiddelde van de opgeslagen coördinaten. De kleinste-kwadratenmethode zoekt daarentegen eerst de randpunten uit van de cirkel. Deze worden vervolgens gebruikt in het zoekalgoritme (zie sectie 2.1.1) dat de cirkel bepaalt die het beste past in de gegeven randpunten. Hieruit kan dan de positie van het centrum van de bol bepaald worden. De Autopilot zal eerst gebruik maken van de kleinste-kwadratenmethode en overschakelen op de zwaartepuntberekening wanneer er onvoldoende randpunten zijn, aangezien deze minder nauwkeurig is wanneer het middelpunt buiten beeld ligt.
Indien de Autopilot geen gekleurde pixels detecteert, zal de drone systematisch de wereld scannen. Hierover meer info in sectie 1.1.3.

1.1.2 Vliegstrategie

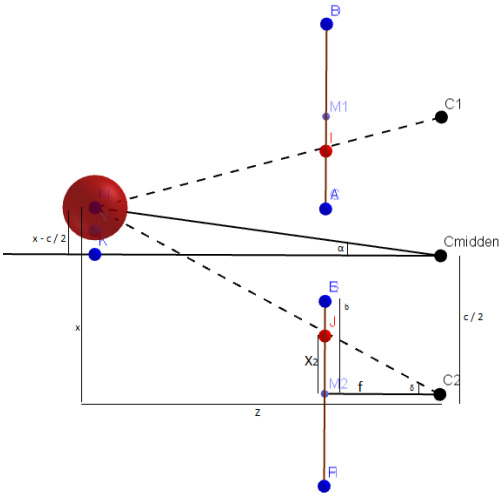
Wanneer de Autopilot een bol in beeld gekregen heeft, zal hij starten met zijn positie relatief tegenover zijn doel te bepalen. De werkwijze wordt in onderstaande beschrijving verduidelijkt.
Ten eerst wordt de diepte bepaald. Dit kan met behulp van de formule van stereo vision [5] berekend worden. Zie Figuur 1 voor een grafische weergave van de berekening. Z stelt de diepte [m] voor, c de afstand [m] tussen de camera's, f de focale afstand [pixel] en x_1 en x_2 stellen de afstand [pixel] tussen het middelpunt van het beeld en het middelpunt van de bol op het beeld voor. Het is eenvoudig in te zien dat de diepte berekend kan worden door formule 1.

$$Z = \frac{c \cdot f}{x_1 - x_2} \tag{1}$$

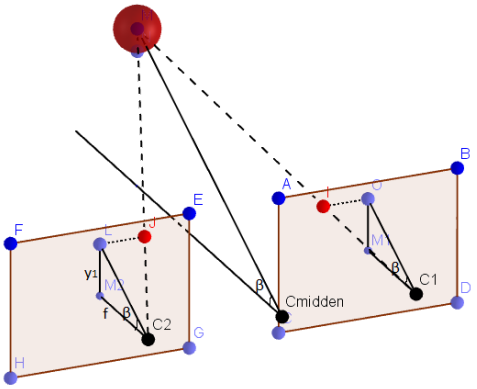


Figuur 1: Diepteberekening tussen drone en doel. In formulevorm: $Z = \frac{c \cdot f}{x_1 - x_2}$.

Vervolgens wordt de fout op de horizontale hoek, α bepaald. Hiermee wordt de afwijking tussen de horizontale positie van de bol en het midden van de drone bedoeld. Deze formule wordt afgeleid via de goniometrische regels. Zie Figuur 2 voor grafische ondersteuning. De hoek δ stelt hier de helft van de horizontale hoek voor die het beeld overspant en b stelt de helft van de breedte van het beeld voor. Onder Figuur 2 kan eveneens de uitwerking van de formule gevonden worden.
Ten slotte wordt ook de fout op de verticale hoek, β bepaald. Dit is de afwijking van de hoogte van de bol ten opzichte van de hoogte van de drone. Wederom afgeleid via de goniometrie en weergegeven in Figuur 3, met y_2 de verticale afstand [pixel] tussen de het middelpunt van het beeld en het middelpunt van de bol.



Figuur 2: Relatieve horizontale hoek.



Figuur 3: Relatieve verticale hoek.

In Figuur 2 wordt de relatieve horizontale hoek tussen drone en doel weergegeven door $\tan(\alpha) = \frac{x-\frac{c}{2}}{Z}$, voor de volledige uitwerking zie formule 2 tot 5.
In Figuur 3 wordt de relatieve verticale hoek weergegeven tussen drone en doel. De relatie wordt gegeven door volgende formule: $\tan(\beta) = \frac{y_1}{f}$.

Berekening brandpuntsafstand f:

$$\tan(\delta) = \frac{b}{f} \qquad (2)$$

$$f = \frac{b}{\tan(\delta)} \qquad (3)$$

Berekening afstand x:

$$\frac{x_2}{f} = \frac{x}{Z} \qquad (4)$$

$$x = Z \cdot \frac{x_2}{f} \qquad (5)$$

Nadat de drone zijn positie bepaald heeft, kan hij zijn fouten (horizontale en verticale hoek en eventuele roll) bijsturen a.d.h.v. PI-controllers die beslissen over respectievelijk yaw, thrust en roll bewegingen. Meer info over de werking van de controllers wordt gegeven in sectie 1.1.4.

Nu rest enkel nog het bepalen van de pitch zodat er onder een ingestelde thrust voorwaarts richting de bol kan bewogen worden. Het eerste idee was om dit op basis van een snelheidscontroller te doen, aangezien de snelheid afhankelijk is van de pitch en hierdoor het controleerbaar zou zijn. De snelheid is echter niet te bepalen door de onbekende windkrachten en door de numeriek wiskundige beperkingen op het berekenen van afgeleiden. Hierdoor zou immers de fout te groot worden. Bijgevolg werd er overgegaan op een alternatief plan. Dit houdt in dat de drone pitcht met een rate op basis van een afstandscontroller. Deze legt altijd een doelafstand vast op een kleine afstand voor de drone zodat hij beperkt wordt tot telkens kleine afstanden overbruggen. Door de combinatie van de kleine afstanden en het feit dat de drone telkens terug horizontaal moet komen per overbrugde afstand, kan de snelheid laag gehouden worden. Wanneer er tegenwind is en de drone verder zou afwijken van zijn doelafstand, zal de afstandscontroller de drone toelaten om verder te pitchen om zo toch tegen de windkracht in zijn doel te bereiken.

Ten slotte moet dit proces herhaaldelijk worden uitgevoerd ten gevolge van de invloed van wind. De wind kan de drone namelijk uit koers brengen. Hierdoor zal de drone telkens zijn positie moeten herberekenen, zich opnieuw oriënteren en de controllers laten bijsturen.

1.1.3 Scannen wereld

Wanneer de drone geen bollen meer in beeld heeft, moet hij rond zich beginnen zoeken totdat hij een nieuwe bol waarneemt. Om de hele wereld te scannen, wordt volgende strategie toegepast.

Ten eerste begint de drone rond zijn as te draaien d.m.v. een yaw beweging. Hij krijgt de opdracht dit te doen gedurende 630°. Een overschot van 270° wordt in rekening gebracht, zodat hij zeker volledig rondgedraaid heeft. Dit om eventuele rotationele invloeden van de wind op te heffen.
Vervolgens vliegt de drone achteruit over een bepaalde afstand zodat mogelijke bollen die zich boven of onder de drone bevinden ook zichtbaar worden. Dit gebeurt door eerst omhoog te pitchen en achterwaartse snelheid op te bouwen, om vervolgens weer horizontaal te komen zodat ook de bollen onderaan zichtbaar blijven. We maken van de opgebouwde snelheid in het begin gebruik om ver genoeg achteruit te vliegen.

Indien er na dit proces nog steeds niks waargenomen is, wordt het geheel opnieuw aangeroepen.

1.1.4 PI Controllers

Om een PI-controller te kunnen gebruiken, is het belangrijk om te weten wat het is en wat het doet. Zoals in [4] wordt aangegeven, fungeert de controller als een controlelus-feedback mechanisme. Dit systeem gaat proberen de fout op de meetwaarde te corrigeren. Het berekent namelijk continu de foutwaarde $e(t)$ als het verschil tussen de correcte waarde en de gemeten waarde. Op basis van deze gegevens berekent de controller de nodige correctie.

Er is gekozen voor een PI-controller i.p.v. een PID-controller, aangezien die ook in de meeste reële systemen gebruikt wordt. Bovendien is de term D (Derivative) heel gevoelig aan ruis en zou die een bijkomende moeilijkheid vormen om af te stellen.
PI staat voor Proportional-Integral. Het P-deel zorgt ervoor dat het verschil in gewenste waarde en de gemeten waarde met een factor K_p wordt versterkt. I zorgt voor een constante sommatie van de fout en vergroot de correctiewaarde afhankelijk van hoelang er een fout bestaat tussen gemeten en gewenste waarde. Dit wordt met een factor K_i versterkt. Deze constanten worden manueel bepaald door de reactie van de controller uit te plotten in grafieken en hun gedrag te optimaliseren door deze coëfficiënten aan te passen.

De correctie kan berekend worden op basis van formule 6.

$$u(t) = K_p * e(t) + K_i * \int e(t)dt \tag{6}$$

Hier is $u(t)$ de correctie, K_p de proportional constante, K_i de integral constante en $e(t)$ de foutwaarde op tijdstip t.

Voorgaande theorie wordt toegepast op dit project.
Ten eerste wordt de roll-controller bekeken. Hij heeft als doel de roll gelijk aan nul te houden. Dit is dan ook ineens de gewenste waarde. De controller begint slechts bij te sturen wanneer de gemeten roll waarde groter is dan 0.2° in positieve of negatieve zin. Bijsturen gebeurt door de correctiewaarde als input te nemen voor de rollrate.
Vervolgens is er een yaw-controller. Deze werkt enkel wanneer het doel in beeld is, aangezien de drone een oriëntatiepunt nodig heeft om de yaw te kunnen bepalen. De gewenste waarde is gelijk aan een horizontale afwijking α van nul graden. De controller gaat dit bijsturen op dezelfde manier als bij de roll-controller.
De height-controller stelt de thrustwaarde in en verschilt licht van de anderen doordat hij de correctiewaarde optelt bij de standaard waarde om de zwaartekracht tegen te gaan. Verder probeert hij de verticale afwijking naar nul te herleiden.
De pitch-controller wordt gebruikt om zo snel mogelijk te kunnen hoveren en dus een pitch van nul te krijgen. Dit gebeurt op dezelfde manier als voorgaande controllers.
Ten slotte is er ook nog een distance-controller. Deze gaat proberen de afstand naar nul te reduce-

ren met stappen van 0.1m en helpt bij het voorwaarts vliegen om de pitchrate in te stellen, zoals uitgelegd in sectie 1.1.2.

1.1.5 Obstakels

De laatste hindernis is het ontwijken van obstakels. De eerste taak is het bepalen van de positie van de obstakels. Dit gebeurt door eerst de verschillende tinten grijs uit het beeld te filteren en vervolgens de obstakels te verwijderen die niet in de buurt van onze vliegrichting liggen. Ten slotte wordt de dichtste eruit gekozen, aangezien die als eerste moet worden ontweken.

Het ontwijken zelf gebeurt a.d.h.v. de height-controller. Indien de bol in de bovenste helft van het beeld staat, zal de drone onder de bol door vliegen. Indien de bol in de onderste helft van het beeld staat, zal hij erover vliegen. Er wordt geopteerd voor deze strategie aangezien de reactie op de thrust vrij snel en accuraat is. De andere optie is ernaast vliegen. Dit zou veel moeilijker en onvoorspelbaarder zijn daar de drone nog voorwaarts zou verder vliegen, ook al is de yaw reeds aangepast. De roll aanpassen, biedt een oplossing, maar is een moeilijke opdracht. Dit verantwoordt de keuze om de hoogte aan te passen.

1.2 Virtual Testbed

Auteur: Bram Vandendriessche

Het bouwen van het Testbed is begonnen met de belangrijke en moeilijke keuze voor de te gebruiken 3D-omgeving. Deze omgeving moest vlot onder de knie te krijgen zijn vanwege de snelle deadlines en ook flexibiliteit was een vereiste. Blender[2], JMonkeyEngine[3] en OpenGL[4] haalden als meest geschikte kandidaten de laatste ronde. Andere keuzes, zoals Unity[5], vielen af door bijvoorbeeld het gebrek aan kennis van `C#` en aan tijd om hier verandering in te brengen.

Blender is een erg uitgebreid programma met tal van mogelijkheden om 3D-objecten en -werelden te maken en manipuleren. Het maakt gebruik van Python, een programmeertaal met een vrij eenvoudige leercurve voor wie al programmeerervaring heeft. Daarentegen is Blender uitdagend en tijdrovend om te beheersen, net door alle mogelijke functies. Aangezien we bovendien gepland hadden in Java te werken, moest gezocht worden naar een manier om Java en Python te verbinden. Blender zou dan vanuit Java gestart moeten worden, wat een omslachtig proces leek. Vooral omwille van de extra leertijd en omweg, werd niet voor Blender gekozen.
De tweede optie, JMonkeyEngine, leek erg gebruiksvriendelijk, had een goede tutorial en enkele handige ingebouwde functies, zoals het vastpinnen van een camera op een object (wat erg nuttig is voor het project). Anderzijds leek de community-ondersteuning (op fora als StackOverflow) bij specifieke zoektermen die bij het testen geprobeerd werden, erg beperkt. Hierdoor viel ook deze optie weg ten opzichte van OpenGL.

De keuze viel uiteindelijk op OpenGL. Ondanks een erg moeizaam begin, is deze toch de goede gebleken. OpenGL kan rechtstreeks in Java worden gebruikt, heeft een brede community met heel wat tutorials en werkt erg flexibel.

In deze fase van het project wordt de drone op een eerder simplistische manier voorgesteld door een balk. Later kan deze vervangen worden door een 3D-vorm van een echte quadcopter. Dit werd tot op heden niet gedaan wegens tijdsgebrek en focus op de inhoudelijke deadlines.

[2] `https://www.blender.org/`
[3] `http://jmonkeyengine.org/`
[4] `https://www.opengl.org/`
[5] `https://unity3d.com/`

2 Algoritmes

Auteurs: Jef Versyck; redactie: Arne Vlietinck

Om een realistisch en correct beeld van de situatie te schetsen, is het belangrijk om op een juiste manier gebruik te maken van algoritmes. Ze beïnvloeden de correctheid en snelheid van het programma. Een verkeerde berekening kan immers leiden tot een waterval van andere fouten. Hieronder volgt een opsomming van alle gebruikte algoritmes in zowel de Autopilot als de simulator.

2.1 Drone Autopilot

Auteurs: Vincent Vliegen

De Autopilot maakt gebruik van algoritmes in verschillende instanties. Om de positie van de bollen te bepalen wordt er gebruik gemaakt van een kleinste kwadraten circle fit. Zo kan de Autopilot bepalen waar de bollen zich bevinden rondom de drone. Ook gebruikt hij een kortstepadalgoritme om de drone voort te laten bewegen tussen meerdere bollen. Dit is een must have om zo een uiterst efficiënt vliegtraject te bepalen.

2.1.1 Kleinste kwadraten circle fit

De nauwkeurigheid van de acties van de Autopilot hangt af van de precisie van de informatie die *ImageCalculations* teruggeeft. De afstand en positionering van alle objecten rond de drone zijn afhankelijk van hun centrum, waarvan de coördinaten op regelmatige basis worden herberekend. Om de meest exacte posities van deze objecten te verkrijgen, maakt *ImageCalculations* gebruik van zwaartepuntberekening en een kleinste kwadraten circle fit algoritme [2]. Aangezien het circle fit algoritme beter werkt bij niet volledig zichtbare bollen, zal *ImageCalculations* altijd hiervoor kiezen, tenzij er niet voldoende pixels zitten op de cirkelomtrek.
Het kleinste kwadraten circle fit algoritme probeert een cirkel te vinden die zo goed mogelijk overeenkomt met de afbeelding van de gegeven bol. Het gebruikt enkel de pixels op de omtrek van de cirkel. Het kleinste kwadraten algoritme berekent voor elk van deze pixels de horizontale en verticale afwijking tegenover de gemiddelde positie van deze pixels in een (x, y)-assenstelsel. Deze afwijkingen worden dan omgezet naar coördinaten in een (u, v)-assenstelsel, waar de fout bepaald kan worden. Het kleinste-kwadratenalgoritme berekent dan de (u, v)-coördinaten van het middelpunt waarvoor de som van de gekwadrateerde fouten minimaal is. Tot slot worden bij deze coördinaten de eerder berekende gemiddelden opgeteld wat resulteert in het middelpunt van de best passende cirkel in het (x, y)-assenstelsel. Voor meer duidelijkheid omtrent de gebruikte assenstelsels en formules is de literatuur [2] aangewezen.

2.1.2 Kortste pad

Wanneer de drone een traject moet afleggen tussen meerdere bollen, bepaalt een algoritme het kortste pad, zodat het afleggen ervan efficiënt kan gebeuren. Om een zo goed mogelijke benadering te maken van het kortste pad, is het best om al vanaf het begin zoveel mogelijk bollen in het algoritme op te nemen. Echter, bij een grote groep bollen zal dit voor extra vertraging zorgen door de vele berekeningen. Daarom gaat het algoritme altijd maar maximaal de twee dichtste bollen berekenen.

Het algoritme bestaat uit twee grote onderdelen: het berekenen van de eerste twee bollen en het herberekenen van deze bollen.
Om het algoritme te kunnen starten, moeten er gekleurde bollen in beeld zijn. Zolang dit niet het geval is, zal de drone de wereld scannen.
Wanneer er voor de eerste keer een bol in beeld komt, stelt de Autopilot een lijst op van alle pixels per kleur. Hiervan bewaart het algoritme de n grootste groepen pixels en berekent het voor elk van deze geselecteerde bollen de afstand. De dichtstbijzijnde bol wordt hierbij ingesteld als eerste doel. Vervolgens bepaalt de Autopilot de afstanden tussen deze eerste bol en de overige $n - 1$ bollen. Hieruit kan de tweede bol worden afgeleid. Indien er slechts één bol in beeld is, gaat de Autopilot

niet op zoek naar een tweede. Hij vliegt eerst naar de gevonden bol, vooraleer hij andere bollen gaat zoeken.
Wanneer de drone naar de eerste bol vliegt, herberekent de Autopilot op regelmatige tijdstippen de eerste en tweede bol. Indien er een bol dichterbij ligt dan de huidige eerste bol, zal deze de nieuwe eerste bol worden en wordt de overeenkomstige tweede bol ook opnieuw berekend. Indien er een bol dichter ligt bij de eerste bol dan de huidige tweede, zal die ook de huidige vervangen.
Indien de drone de eerste bol bereikt, wordt de huidige tweede bol ingesteld als nieuw doel en wordt er een nieuwe tweede bol berekend. Indien er geen tweede bol was ingesteld, zal de drone opnieuw beginnen scannen.

2.2 Virtual Testbed

Auteur: Jef Versyck

Om alle fysische elementen op een zo waarheidsgetrouw mogelijke manier weer te geven, wordt er een fysische *library* gecreëerd. Hiermee is het mogelijk om bewegingen in het assenstelsel juist weer te geven. ook moeten de krachten op een correcte manier behandeld worden. Voor de laatste mijlpaal wordt er collision detection toegevoegd.

2.2.1 Assenstelsels

Het relatief assenstelsel heeft het centrum van de drone als oorsprong en is zo georiënteerd dat de drone kijkt in de richting van de positieve X-as, met de Y-as naar boven en de Z-as naar rechts gericht. Het assenstelsel ondergaat dezelfde translaties en rotaties als de drone.

De drone kan drie bewegingen uitvoeren: yaw Y, roll R en pitch P. De yaw is een negatieve rotatie rond de Y-as, de roll is een positieve rotatie rond de X'-as [6] en de pitch is een negatieve rotatie rond de Z"-as [7]. Merk op dat er een verschil bestaat tussen de huidige roll en pitch die de drone heeft op een gegeven moment en de yaw, roll en pitch die hij moet uitvoeren om in die positie te geraken.

Door de volgorde van de drie rotaties onstaat er een specifieke rotatiematrix. Met behulp van deze rotatiematrix kunnen de posities van de camera's van de drone bepaald worden. Ook kunnen we hiermee de veranderingen van yaw, roll en pitch berekenen. De verandering van de pitch is immers afhankelijk van de roll: hoe groter de roll, hoe trager de pitch zal veranderen.
De bekomen rotatiematrix:

$$M = \begin{bmatrix} \cos(Y)\cos(P) - \sin(Y)\sin(R)\sin(P) & \sin(P)\cos(R) & -\sin(Y)\cos(P) - \cos(Y)\sin(P)\sin(R) \\ -\cos(Y)\sin(P) - \cos(P)\sin(R)\sin(Y) & \cos(R)\cos(P) & \sin(Y)\sin(P) - \cos(P)\cos(Y)\sin(R) \\ \sin(Y)\cos(R) & \sin(R) & \cos(R)\cos(Y) \end{bmatrix}$$

De veranderingen van yaw, roll en pitch steunen op het volgende principe:

$$M^{-1} \cdot M \cdot x = x$$

met M de transformatiematrix, M^{-1} de inverse van de transformatiematrix en x de algemene beweging. Beschouw $M \cdot x$ als een relatieve beweging, zoals de verandering van de pitch. De relatieve beweging vermenigvuldigd met de inverse van de transformatiematrix geeft de algemene beweging die de drone moet uitvoeren voor zijn pitchverandering.

2.2.2 Krachten

Op elke drone werken er minstens twee krachten in: de zwaartekracht en de thrust van de drone. De zwaartekracht G is gedefinieerd als een kracht volgens de negatieve Y-richting gelijk aan de massa

[6] X-as van het relatieve assenstelsel na yaw beweging
[7] Z-as van het relatieve assenstelsel na yaw en roll beweging

van de drone maal de gravitatieconstante. De gravitatieconstante wordt als negatief beschouwd.

$$\vec{G} = \begin{Bmatrix} 0 \\ m \cdot g \\ 0 \end{Bmatrix}$$

De thrust T daarentegen is een kracht afhankelijk van de huidige rotaties van de drone. Daarom zal deze vermenigvuldigd worden met de rotatiematrix. Hieronder staat het resultaat van deze vermenigvuldiging. Oorspronkelijk staat deze kracht enkel volgens de positieve Y-richting.

$$\vec{T} = \begin{Bmatrix} T\cos(R)\sin(P) \\ T\cos(R)\cos(P) \\ T\sin(R) \end{Bmatrix}$$

Een derde kracht die kan voorkomen, is de drag D. Deze kracht is afhankelijk van de snelheid v van de drone en de aerodynamische coëfficiënt d. De constante wordt als positief beschouwd.

$$\vec{D} = \begin{Bmatrix} -v_{\mathrm{x}} \cdot d \\ -v_{\mathrm{y}} \cdot d \\ -v_{\mathrm{z}} \cdot d \end{Bmatrix}$$

De windkracht W is een kracht met een veranderlijke richting en snelheid. Op bepaalde tijdstippen verandert de kracht van richting en van snelheid. De tijdstippen worden, net als de grootte van deze twee variabelen, bij het aanmaken van de simulator bepaald. De snelheden W_x, W_y en W_z, vermenigvuldigd met de aerodynamische coëfficiënt d, geeft ons de kracht in Newton.

$$\vec{W} = \begin{Bmatrix} W_{\mathrm{x}} \cdot d \\ W_{\mathrm{y}} \cdot d \\ W_{\mathrm{z}} \cdot d \end{Bmatrix}$$

Alle vectorkrachten die inwerken op de drone op een bepaald moment, worden bij elkaar opgeteld. De bekomen vector, gedeeld door de massa m van de drone, geeft de versnelling a van de drone op dat bepaald tijdstip, via de formule:

$$\sum_{1}^{n} \vec{F} = m \cdot \vec{a}$$

Tenslotte kunnen via de snelheids- en positievergelijkingen de huidige snelheid en positie berekend worden. Merk wel op dat er geen constante versnelling is en de vergelijkingen dus telkens opnieuw opgesteld moeten worden. De δ is hier de tijdsverandering ten opzichte van de vorige positie- en snelheidsverandering.

$$x_{\mathrm{n}} = (a_{\mathrm{n\text{-}1}} \cdot \delta^2)/2 + v_{\mathrm{n\text{-}1}} \cdot \delta + x_{\mathrm{n\text{-}1}}$$

$$v_{\mathrm{n}} = a_{\mathrm{n\text{-}1}} \cdot \delta + v_{\mathrm{n\text{-}1}}$$

2.2.3 Collision detection

Twee objecten A en B botsen met elkaar indien de afstand tussen hun zwaartepuntscentra kleiner is dan de som van hun twee radii. Dit wordt gecontroleerd met behulp van de volgende formule:

$$\sqrt{(x_{\mathrm{A}} - x_{\mathrm{B}})^2 + (y_{\mathrm{A}} - y_{\mathrm{B}})^2 + (z_{\mathrm{A}} - z_{\mathrm{B}})^2} < r_{\mathrm{A}} + r_{\mathrm{B}} \Rightarrow \mathit{Botsing}$$

3 Software

Auteurs: ; Redactie: Arne Vlietinck

3.1 Drone Autopilot

Auteur: Arne Vlietinck

De aanmaak van een Autopilot en bijhorende *GUI* gebeurt in de *DroneAutopilotFactory*-klasse, geïmplementeerd met de interface *AutopilotFactory*. Hierin worden ook de beginwaarden voor yaw, roll, pitch en thrust direct ingesteld.
Door *DroneAutopilotFactory* wordt een nieuw object *DroneAutopilot* aangemaakt die de interface *Autopilot* implementeert. De *DroneAutopilot*-klasse bestaat uit één functie, namelijk `timeHasPassed()` die continu door de simulator wordt uitgevoerd. Vanuit deze methode wordt verwezen naar een bepaalde *mission*-klasse die verder de opdracht voor zijn rekening neemt.

Er zijn twee missies die tot nu toe in de milestones nodig zijn namelijk *OneSphere* en *SeveralSpheres*. Hierdoor vliegt de drone respectievelijk naar een specifiek gekleurde bol of probeert hij alle bollen te bereiken. De keuze van de bepaalde missie wordt gemaakt in de *GUI* door de gewenste missie te selecteren in het dropdownmenu. Om dit onderdeel aanpasbaar te maken, wordt een abstracte klasse *Mission* voorzien, deze voorziet de basiselementen van de verschillende missies. Elke missie selecteert op zijn beurt *MoveToTarget* om de drone naar zijn beoogd doel te laten bewegen.

In *MoveToTarget* worden de aansturing en bewegingen van de drone bepaald. Ook zullen de verschillende rates doorgegeven worden aan de simulator. Bovendien wordt hierin ook telkens de *GUI* geüpdatet wanneer een nieuwe waarde voor de diepte bepaald is.

Om de bewegingen te kunnen berekenen wordt er gesteund op twee *calculation*-klassen namelijk *ImageCalculations* en *PhysicsCalculations*. Deze twee klassen staan los van de Autopilot, maar zijn van cruciaal belang om alles correct te laten verlopen. *ImageCalculations* is verantwoordelijk voor alles omtrent de beelden die de Autopilot binnenkrijgt, m.a.w. de rode pixels zoeken en het zwaartepunt bepalen. In de klasse *PhysicsCalculations*, worden alle fysische berekening van hoeken en afstanden uitgevoerd.

Om de fouten die gemaakt zijn in de calculation-klassen binnen de perken te houden wordt zoals eerder al vermeld in sectie 1.1.4, *PI-controllers* voorzien. De algemene uitwerking van een controller wordt geïmplementeerd in een abstracte klasse *PIController*. Voor elk te controleren waarde is er een specifieke controller.
Daarnaast wordt om de code van *MoveToTarget* te verlichten *Correctors* voorzien. In deze klassen wordt de methode gegeven hoe er moet omgegaan worden met de specifieke PI-controller. Terug is er een abstracte klasse voorzien om latere uitbreidingen gemakkelijker te maken.

3.2 Virtual Testbed

Auteur: Bram Vandendriessche

Voor de simulator is geprobeerd om zo modulair mogelijk te werken, zodat nieuwe opstellingen of constraints gemakkelijk kunnen worden toegevoegd zonder dat dit voor problemen zou zorgen in de bestaande structuur. De klasse *World* staat centraal in de simulator. Zij bevat alle basiselementen voor zowel het fysische als voor het 3D-gedeelte van de wereld. Het fysische aspect wordt behandeld door de klasse *Physics*, wat al eerder werd besproken in sectie 2.2.

World is geïmplementeerd als een subklasse van *GLCanvas* en legt de basis voor de 3D-weergave van de verschillende opstellingen. De klasse houdt bij welke *WorldObjects* ze bevat (*Spheres*, *SimulationDrones*, *ObstacleSpheres*) en implementeert enkele belangrijke functies voor de 3D-visualisatie ervan. De initialisatie van de wereld gebeurt door `init()`. De functie `draw()` roept de `draw()`-functie op van elk *WorldObject* dat de wereld bevat. De `display()`-functie maakt meerdere keren gebruik van `draw()` om de wereld zowel naar het venster van de *GUI* te renderen als naar de

framebuffer objects die dienen voor het offscreen renderen (voor `takeimage()`).
Bovendien bevat *World* ook een abstracte methode `setup()`. Deze kan naar wens geïmplementeerd worden door elke subklasse, zodat per mijlpaal een specifieke opstelling kan worden vastgelegd. Voor Milestone 1.1 is dit bijvoorbeeld enkel een rode bol, een drone en enkele camera's. Voor een wereld die vanuit een invoerbestand wordt opgebouwd met behulp van de parser, bestaat *WorldParser*, waarbij de `setup()`-functie gebruik maakt van de geparste waarden.

Uiteraard is er niets aan een 3D-wereld indien hij niet bekeken kan worden. De klasse *GeneralCamera* werd ingevoerd om te kunnen werken met het concept van een camera, die een positie en kijkrichting krijgt. Zulke camera's kunnen dan op verschillende punten geplaatst worden (vastgelegd in de `setup()` van de wereld), zodat de gebruiker de voortgang van de drone goed kan volgen. Een uitbreiding op dit concept is de *DroneCamera*, die zich voor zijn positie en oriëntatie op de drone (waartoe hij behoort) zal baseren. Dit laatste type kan gebruikt worden voor de "ogen" van de drone en deze klasse implementeert de *Camera*-interface van de *API* voor de verbinding van Autopilot en Testbed.
Deze rol vult *SimulationDrone* in voor de *Drone*-interface. Deze klasse tekent de drone als een blauwe balk.

4 GUI

Auteurs: Matthias Van der Heyden; redactie: Arne Vlietinck

De Drone Autopilot en het Virtual Testbed hebben beide een *graphical user interface* of kortweg een *GUI*. Deze zorgen voor de communicatie tussen de gebruiker en het programma. De gebruiker kan de opdracht, het camerastandpunt, de windsnelheid en gravitatieconstante instellen en het programma geeft de berekende beelden terug. Om eenvoudig de *GUI* te programmeren wordt Java's Abstract Window Toolkit (*java.awt*) gebruikt. Een overzicht van beide *GUI's* kan gevonden worden in appendix A.

4.1 Drone Autopilot

Auteur: Matthias Van der Heyden

De *GUI* van de Autopilot heeft twee functies: de gebruiker de mogelijk geven een opdracht voor de drone te selecteren en de voortgang van het bereiken van de volgende bol weergeven.

Voor het selecteren van een opdracht is er een dropdownmenu voorzien dat gebruik maakt van *JComboBox* uit de *Swing library*[8] van Java. Wanneer de gebruiker een optie aanduidt, verandert de boolean van deze optie naar *true*. Dit zorgt ervoor dat de juiste commando's voor deze opdracht uitgevoerd worden. Er zijn twee mogelijke opdrachten: *Fly shortest path*, die de drone de kortste weg door de verschillende bollen laat vliegen, en *Fly to red sphere*, die ervoor zorgt dat de drone een rode bol zoekt en er heen vliegt. Deze laatste optie kan eenvoudig uitgebreid worden naar bijvoorbeeld *Fly to colored sphere* waarbij de gebruiker in een pop-up de gewenste kleur invult of aanduidt op een kleurenpalet.

Een *JProgressBar* uit de *Swing library* van Java geeft in de *GUI* de voortgang van de drone tot de volgende bol weer. Ze krijgt de kleur van de bol waar de drone naartoe vliegt. Bij elke berekening van de afstand tot het doel wordt deze geüpdatet. Is de afstand groter dan de laatste grootste afstand tot het doel, dan stelt de progress bar deze nieuwe afstand in als het nieuwe maximum en is de voltooiingsgraad weer nul. Is de afstand kleiner, dan is de nieuwe voltooiingsgraad gelijk aan 100 procent verminderd met de verhouding van de grootste afstand en de huidige afstand.

[8] *Swing library*: https://docs.oracle.com/javase/7/docs/api/javax/swing/package-summary.html

4.2 Virtual Testbed

Auteurs: Arne Vlietinck

In het Virtual Testbed wordt een panel van vaste grootte voorzien waarin de *GUI* vervat zit. In tegenstelling tot de Autopilot wordt hier gebruik gemaakt van een iets uitgebreidere lay-outvorm namelijk de *GridBagLayout*[9]. Dit zorgt voor een op maat gemaakte lay-out, die noodzakelijk was voor het uitlijnen van de verschillende functies.

De centrale functie van deze *GUI* is het selecteren van verschillende camerastandpunten. Dit kan door middel van de *JButtons*. Naargelang de hoeveelheid camerastandpunten zullen er automatisch extra buttons gegenereerd worden. De basiscamera's bestaan uit de verschillende wereldcamera's, de linkerdronecamera en een third-personcamera, die de drone op de voet volgt. Naast deze mogelijke camera's kan er in het scherm van de wereldcamera gebruik gemaakt worden van de pijltjestoetsen (en P&M) [10] om de drone op de voet te volgen. Zo kan er steeds een gewenst zicht gecreëerd worden.

Naast de mogelijkheid om de camera's te kiezen, worden er ook *JSliders* voorzien die het mogelijk maken de wind (in x-, y- en z-richting) en gravitatieconstante aan te passen naar behoeven. Zo kan ten allen tijde de invloed van respectievelijk de wind en zwaartekracht onderzocht worden. Wanneer er gewerkt wordt met de WorldParser zijn er reeds windkarakteristieken meegegeven. Indien dit het geval is, zal er enkel een output zijn van de windsterkte in verschillende richtingen. Ook wordt de snelheid en de positie van de drone weergegeven. Deze wordt continu opnieuw opgevraagd en geüpdatet in de *GUI*.

Tot slot wordt er nog een laatste button voorzien waarmee het mogelijk is om een nieuwe bol toe te voegen. Door de button te gebruiken opent er een nieuw frame waarin de gewilde coördinaten ingegeven kunnen worden.

5 Testen

Auteurs: Arne Vlietinck; redactie: Arne Vlietinck

Voor beide programma's zullen er uitvoerig testklassen gegenereerd en uitgevoerd worden. Hiermee kan de correcte werking en de nauwkeurigheid van de gebruikte algoritmes gecontroleerd worden. Deze klassen vormen daarnaast ook de basis voor het opsporen van fouten en het debuggen ervan.

Indien er updates zijn in waarden van de testmethodes tussen het schrijven van het verslag en de demo zal er gezorgd worden voor een actualisatie van de waarden op de presentatie.

5.1 Drone Autopilot

Auteur: Vincent Vliegen

De Autopilot is voorzien van enkele *JUnit*-testklassen. Deze testen verschillende methodes op hun nauwkeurigheid en correctheid. Zo is het eenvoudig de juistheid van deze veelgebruikte methodes na te gaan.

De *ImageCalculationsTest* is uitgerust met testen voor elke methode in de *ImageCalculations*-klasse. Er is gebruik gemaakt van een anonieme klasse die de *Camera* interface implementeert, zodat er afbeeldingen naar keuze gegenereerd kunnen worden. Deze afbeeldingen zijn omwille van hun grote hoeveelheid informatie beperkt tot een minimale grootte van 9x9 en 10x10 pixels, dit vereenvoudigt immers het testen. Desalniettemin zijn er ook twee bitmap bestanden ter beschikking

[9] *GridBagLayout*: https://docs.oracle.com/javase/7/docs/api/java/awt/GridBagLayout.html

[10] Respectievelijk links, rechts, voorwaarts en achterwaarts met de pijltjestoetsen en opwaarts (P) of neerwaarts (M).

met elk een representatief beeld van een rode bol. Dit geeft een voldoende nauwkeurige, cirkelvormige afbeelding van een rode bol voor de berekening van desbetreffend middelpunt. De testen tonen aan dat wanneer het centrum van de bol zich buiten beeld bevindt, de benadering van het middelpunt beter is bij berekeningen via het least square circle fit algoritme dan bij berekeningen van het zwaartepunt van de zichtbare pixels. Om deze bevinden te staven worden testwaarden toegevoegd aan Appendix B. Daarnaast heeft least square circle fit algoritme nog als voordeel dat de straal ook berekend kan worden.

De *PhysicsCalculationsTest* verschaft testmethodes voor de *PhysicsCalculations*-klasse. Ook hier zijn anonieme klassen geïmplementeerd, namelijk voor de *Camera* en *Drone* interfaces. Deze testen geven weer dat de gebruikte formules in iedere situatie voldoende zijn om alle fysische data nauwkeurig te berekenen.

5.2 Virtual Testbed

Auteur: Jef Versyck

Het testen van bepaalde functies van de simulator gebeurt via een aangemaakte *dummy* wereld die enkel een *SimulationDrone*, een *Sphere* en een *ObstacleSphere* bevat. Om de collision detection te testen, worden beide *Spheres* eerst apart, vervolgens tesamen op de drone geplaatst en wordt telkens collision detection opgeroepen. Ten tweede wordt de bol net buiten en net binnen het verwachte bereik geplaatst en wordt wederom collision detection opgeroepen.

Besluit

Auteurs: ; redactie: Arne Vlietinck

Referenties

[1] H. Blockeel, B. Jacobs, and D. Nuyens, *Een automatische piloot en virtuele testomgeving voor drones*, 3 oktober 2016.

[2] R. Bullock, *Least-squares circle fit.* `http://www.dtcenter.org/met/users/docs/write_ups/circle_fit.pdf`, 2006. [Geraadpleegd op 10 oktober 2016].

[3] M. GMBH, *Microdrone-applications: aerial, photography, mapping, surveying, etc.* `https://www.microdrones.com/en/applications/`, 2016. [Geraadpleegd op 30 oktober 2016].

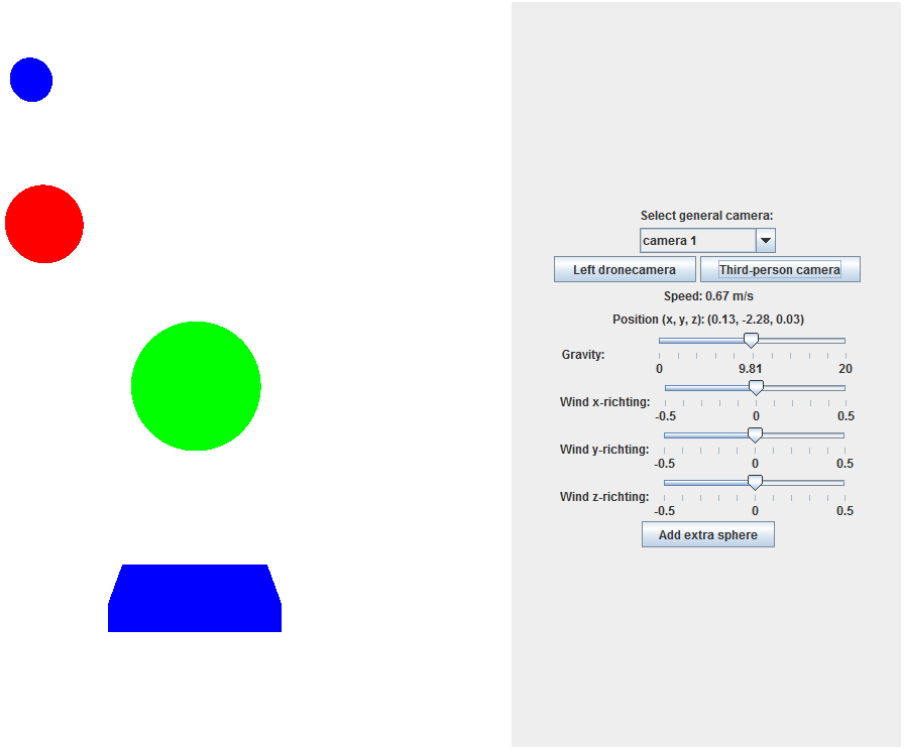
[4] N. Instruments, *Pid theory explained.* `http://www.ni.com/white-paper/3782/en/`, 2011. [Geraadpleegd op 21 november 2016].

[5] D. Nair, *A guide to stereovision and 3d imaging.* `http://www.techbriefs.com/component/content/article/14925`, 1 oktober 2012. [Geraadpleegd op 5 oktober 2016].

[6] B. Vandenriessche, A. Vlietinck, M. Van der Heyden, J. Versyck, V. Vliegen, and L. Vranken, *P&O: Computerwetenschappen Tussentijds verslag*, 9 november 2016.

A GUI

Figuur 4: De *GUI* van de Autopilot. Hierop is het dropdownmenu te zien waar de missie kan geselecteerd worden. Ook de vooruitgang naar de volgende (in dit geval groene bol) is waar te nemen.



Figuur 5: De *GUI* van het Testbed. De camera kan gekozen worden a.d.h.v. het dropdownmenu of de twee extra buttons. Daarnaast wordt de ogenblikkelijke snelheid en positie van de drone weergeven. Via de schuifbalken is het mogelijk de wind en gravitatieconstante aan te passen. Om uiteindelijk een extra bol (via coördinaten) toe te voegen wordt de laatste button gebruikt.

B Testen

correcte waardes (2048x2048)			berekend via circle fit				berekend via cog	
coördinaten centrum bol	straal	% zichtbaar	coördinaten centrum bol	uitwijking coödinaten	straal	uitwijking straal	coördinaten centrum bol	uitwijking coödinaten
{1024, 1024}	300	100.00	{1023.59, 1023.54}	{0.41, 0.46}	291.24	8.76	{1023.12, 1023.58}	{0.88, 0.42}
{25, 675}	100	65.75	{24.53, 774.50}	{0.47, 0.50}	98.99	1.01	{53.57, 774.50}	{-28.57, 0.50}
{1000, -50}	250	37.35	{999.50, -50.33}	{0.50, 0.33}	248.96	1.04	{999.12, 82.86}	{0.88, -132.86}
{1600, 2175}	400	30.13	{1599.50, 2174.30}	{0.50, 0.70}	398.96	1.04	{1599.04, 1935.18}	{0.96, 239.82}
{250, 2150}	350	29.85	{249.44, 2149.48}	{0.56, 0.52}	349.13	0.87	{266.67, 1941.59}	{-16.67, 208.41}
{2200, 200}	600	24.75	{2200.36, 199.39}	{-0.36, 0.61}	591.90	8.10	{1854.30, 326.76}	{345.70, -126.76}

Figuur 6: Deze testdata van de *ImageCalculation* geeft de verschillen weer tussen beide algoritmes die gebruikt worden om de afstand van de bol te berekenen. Het is duidelijk dat het circle-fit algoritme beduidend minder afwijkt van de correcte waardes dan het algoritme op basis van het zwaartepunt (cog). Hoe minder de bol zichtbaar is, hoe groter de afwijking.