



KU Leuven

Departement Computerwetenschappen

P&O: COMPUTERWETENSCHAPPEN

Eindverslag

Team:
Zilver

BRAM VANDENDRIESSCHE (Coördinator)

ARNE VLIETINCK (Secretaris)

MATTHIAS VAN DER HEYDEN

JEF VERSYCK

VINCENT VLIEN

LAURA VRANKEN

Academiejaar 2016-2017

Inhoudsopgave

1	Ontwerp	2
1.1	Drone Autopilot	2
1.1.1	Beeldverwerking	2
1.1.2	Vliegstrategie	3
1.1.3	Scannen wereld	4
1.1.4	PI Controllers	5
1.1.5	Obstakels	5
1.2	Virtual Testbed	6
2	Algoritmes	6
2.1	Drone Autopilot	6
2.1.1	Kleinste kwadraten circle fit	6
2.1.2	Kortste pad	7
2.2	Virtual Testbed	7
2.2.1	Assenstelsels	7
2.2.2	Krachten	8
2.2.3	Collision detection	9
3	Software	9
3.1	Drone Autopilot	9
3.2	Virtual Testbed	9
4	GUI	10
4.1	Drone Autopilot	10
4.2	Virtual Testbed	10
5	Testen	11
5.1	Drone Autopilot	11
5.2	Virtual Testbed	11

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. Amazon¹) 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, en is een vervolg op het tussentijds verslag [6]. 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 software-systeem 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 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 lichte invloed van wind in willekeurige richtingen. Bovendien moet ook voor beiden een grafische user interface (*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.

1 Ontwerp

Auteur: ; redactie: Arne Vlietinck

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 gegenereerd door de dronecamera's. Bovendien zorgt de Autopilot ervoor dat de drone juist 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 ook rekening houden met een mogelijke invloed van wind die de drone van zijn koers doet afwijken en ook met obstakels waarrond hij moet vliegen.

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, indien er al een doel beslist is. Anders zullen de pixels gegroepeerd worden per kleur dat voorkomt in een HashMap. De gekleurde pixels worden bijgehouden door hun positie ten opzichte van het beeld, uitgedrukt in rij en kolom, op te slaan. We baseren onze

¹Amazon Prime Air

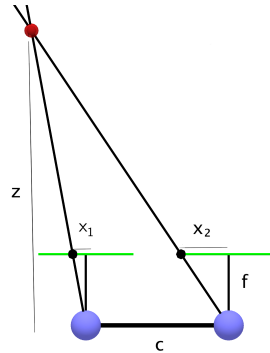
berekeningen op het midden van de bol. Dit kan benaderd worden op twee manieren: via het zwaartepunt of de kleinste-kwadratenmethode op de randpunten van de cirkel. Het zwaartepunt van een bepaalde kleur pixels is te berekenen via het gemiddelde van de opgeslagen coördinaten. De kleinste-kwadratenmethode zoekt daarentegen eerst de randpunten uit van de cirkel. Deze worden vervolgens gebruikt in een algoritme (zie 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 geleidelijkaan de wereld afscannen. Hierover meer info in subsectie 1.1.3.

1.1.2 Vliegstrategie

Wanneer de Autopilot iets in beeld gekregen heeft, zal hij starten met zijn positie relatief tegenover zijn doel te bepalen. Dit gebeurt in volgende stappen.

Ten eerst wordt de diepte bepaald. Dit kan met behulp van de formule van stereo vision [5] uitgewerkt 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] voor tussen het middelpunt van het beeld en het middelpunt van de bol op het beeld voor.

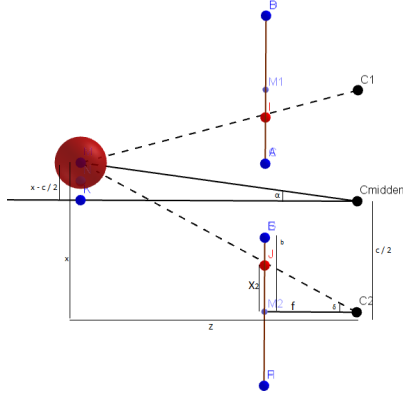


Figuur 1: Diepteberekening tussen drone en doel. In formulevorm: $Z = \frac{c * f}{x_1 - x_2}$.

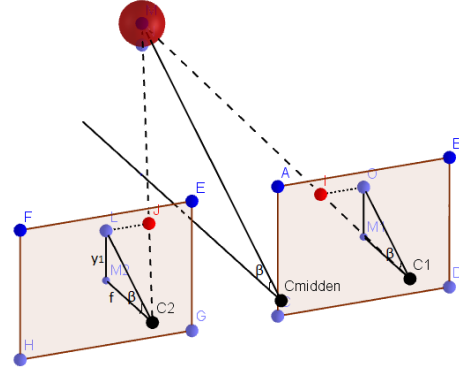
Vervolgens bepalen we de fout op de horizontale hoek, α . Hiermee wordt de afwijking tussen de horizontale positie van de bol en het midden van de drone bedoeld. Deze formule wordt afgeleid via de goniometrische regels. Zie Figuur 2 voor grafische ondersteuning. De hoek δ stelt hier de helft van de horizontale hoek voor die het beeld overspant en b stelt de helft van de breedte van het beeld voor. Onder de figuur, kan eveneens ook de uitwerking van de formule gevonden worden. Tenslotte bepalen we ook de fout op de verticale hoek, β . 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 het middelpunt van het beeld en het middelpunt van de bol.

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 subsectie 1.1.4. Deze bijsturing gebeurt tijdens het vliegen naar het doel.

Nu rest er ons enkel nog de pitch te bepalen zodat we onder een ingestelde thrust voorwaarts richting de bol kunnen bewegen. Het eerste idee was om dit op basis van een snelheidscontroller te doen, aangezien de snelheid afhankelijk is van de pitch en we hierdoor ook de snelheid zouden kunnen controlleren en laag houden. De snelheid is echter niet te bepalen door de onbekende windkrachten of door de numeriek wiskundige beperkingen op afgeleiden berekenen. Bijgevolg zijn we overgegaan op het tweede plan. Dit houdt in dat de drone pitcht met een rate op basis van een afstandscontroller. Deze legt altijd een doel afstand vast op een kleine afstand voor de drone zodat



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 1 tot 4.

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:

$$\tan(\delta) = \frac{b}{f} \quad (1)$$

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

Berekening afstand x:

$$\frac{x_2}{f} = \frac{x}{Z} \quad (3)$$

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

de drone beperkt wordt tot telkens kleine afstanden te overbruggen. Door de combinatie van de kleine afstanden en 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 afwijkt van zijn doel afstand, zal de afstandscontroller de drone toelaten om verder te pitchen om toch zijn doel te bereiken.

Tenslotte 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.

De bijkomstige moeilijkheid van obstakels ontwijken zal verder worden uitgewerkt in subsectie 1.1.5.

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 af te scannen, voeren we volgende stappen uit.

Ten eerste begint de drone rond zijn as te draaien d.m.v. een yaw beweging. Hij krijgt de opdracht dit te doen gedurende 540° . We nemen een overschot van 180° zodat hij zeker volledig rondgedraaid heeft, mits invloed van eventuele rotationele wind.

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 drijven.

Indien we na dit proces nog steeds niks waargenomen hebben, herstarten we het van het begin.

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. Volgens [4] fungeert de controller als een controlelus-feedback mechanisme. Dit systeem gaat proberen de fout op de meetwaarde proberen te corrigeren. Het berekent namelijk continu de foutwaarde $e(t)$ als het verschil tussen de verlangde 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 wordt gebruikt. 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 verlangde 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 verlangde 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 (MSS FOTO?) en hun gedrag te optimaliseren door deze coëfficiënten aan te passen.

Tenslotte wordt de correctie dan op basis van de volgende formule berekend.

$$u(t) = K_p * e(t) + K_i * \int e(t)dt \quad (5)$$

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

Nu gaan we deze theorie toepassen op ons eigen project.

Ten eerste bekijken we de roll controller. Hij heeft als doel de roll gelijk aan nul te houden. Dit is dan ook ineens de verlangde waarde. De controller begint slechts bij te sturen wanneer de gemeten roll waarde groter is dan $0,2^\circ$ in positieve of negatieve zin. Bijsturen gebeurt door de correctiewaarde als input te nemen voor de roll rate.

Vervolgens hebben we de yaw controller. Deze werkt enkel wanneer de doel bol in beeld is, aangezien hij een oriëntatiepunt nodig heeft om de yaw te kunnen bepalen. De verlangde 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 hoogte controller stelt de thrust waarde 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 brengen.

De pitch controller wordt gebruikt om zo snel mogelijk te kunnen gaan hoveren en dus een pitch van nul te krijgen. Wederom op dezelfde manier.

Tenslotte is er ook nog een afstandscontroller. Deze gaat proberen de afstand naar nul te reduceren met stappen van 0.1m en helpt bij het voorwaarts vliegen om de pitchrate in te stellen, zoals eerder uitgelegd.

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. Tenslotte kiezen we de dichtste eruit, aangezien we die eerst zullen moeten ontwijken.

Het ontwijken zelf gebeurt a.d.h.v. de hoogte controller. Indien de bol in de bovenste helft van het beeld staat, zullen we onder de bol doorvliegen. Indien de bol in de onderste helft van het beeld staat, zullen we erover vliegen. We hebben gekozen om er onder en boven te vliegen aangezien de reactie op de thrust vrij snel is. De andere optie is er naast vliegen. Dit zou veel moeilijker zijn omdat de drone nog voorwaarts zal verder drijven, ook al is de yaw aangepast. De roll aanpassen om dat probleem op te lossen, zou veel moeilijker zijn. Vandaar 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. Blender², JMonkeyEngine³ en OpenGL⁴ haalden als meest geschikte kandidaten de laatste ronde. Andere keuzes, zoals Unity⁵, 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 te 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.

2 Algoritmes

Auteurs: ; redactie: Arne Vlietinck

2.1 Drone Autopilot

Auteurs: Vincent Vliegen

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* terug geeft. 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 zwaartepuntsberekeningen en een kleinste kwadraten circle fit algoritme. Aangezien het circle fit algoritme beter werkt bij niet volledig zichtbare bollen, zal de *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 [2]. 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 kwadraten algoritme berekent dan de (u, v) -coördinaten van het middelpunt waarvoor de som van de gekwadrateerde fouten minimaal is. Tot slot wordt er bij deze

²<https://www.blender.org/>

³<http://jmonkeyengine.org/>

⁴<https://www.opengl.org/>

⁵<https://unity3d.com/>

coördinaten de eerder berekende gemiddelden opgeteld wat resulteert in het middelpunt van de best passende cirkel in het (x, y) -assenstelsel.

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 het beste 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 afscannen.

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 zicht 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 aan het vliegen is, richting de eerste bol, 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 dichterbij ligt bij de eerste bol dan de huidige tweede, zal die ook de huidige vervangen.

Op voorwaarde dat 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

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 ontstaat 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.

⁶X-as van het relatieve assenstelsel na yaw beweging

⁷Z-as van het relatieve assenstelsel na yaw en roll beweging

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

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

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.

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

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

$$\vec{D} = \begin{pmatrix} -v_x \cdot d \\ -v_y \cdot d \\ -v_z \cdot d \end{pmatrix}$$

De windkracht W is een kracht met een veranderlijke richting en snelheid. Op voorbepaalde 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 aërodynamische coëfficiënt d , geeft ons de kracht in Newton.

$$\vec{W} = \begin{pmatrix} W_x \cdot d \\ W_y \cdot d \\ W_z \cdot d \end{pmatrix}$$

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:

$$\sum_1^n \vec{F} = m \cdot \vec{a}$$

Ten slotte 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.

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

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

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:

$$\sqrt{(x_A - x_B)^2 + (y_A - y_B)^2 + (z_A - z_B)^2} \leq r_A + r_B \Rightarrow \text{Botsing}$$

3 Software

Auteurs: ; Redactie: Arne Vlietinck

3.1 Drone Autopilot

Auteur: Laura Vranken

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 geparseerde 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.

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 er voor 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.

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 de mogelijkheid om de camera's te kiezen, worden er ook *JSlders* 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. Ook wordt de snelheid en de positie van de drone weergegeven. Deze wordt continu opnieuw opgevraagd en geüpdatet in de *GUI*.

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

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

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: ; redactie: Arne Vlietinck

5.1 Drone Autopilot

Auteur: Jef Versyck

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. VANDENRIESCHE, A. VLIETINCK, M. VAN DER HEYDEN, J. VERSYCK, V. VLIENEN, AND L. VRANKEN, *P&O: Computerwetenschappen Tussentijds verslag*, 9 november 2016.