KU Leuven
Departement Computerwetenschappen

P&O: Computerwetenschappen

Tussentijds verslag

Team:
Zilver

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

Academiejaar 2016-2017

Samenvatting

Auteur: Laura Vranken

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. Voor de tweede mijlpaal wordt deze wereld uitgebreid met een windkracht in een willekeurige richting.
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 een *GUI*. Deze geeft de mogelijkheid het camerastandpunt te kiezen, ook de voltooiingsgraad, snelheid en positie van de drone worden weergegeven.

Inhoudsopgave

Inleiding

Auteur: 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 zijn veiligheidsinspectie van windturbines of elektriciteitslijnen, luchtsteun bij zoek- en reddingsoperaties, bewaking en luchtfotografie. [3]
Wanneer een drone autonoom functioneert, is een betrouwbare aansturing door de Autopilot van levensbelang. Hij moet namelijk bestand zijn tegen externe factoren.

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 van het doel ingeschat worden en nieuwe bewegingsopdrachten voor de drone gegenereerd worden. Deze bewegingen worden weergegeven in een Virtual Testbed. Dit is een softwaresysteem dat een fysieke opstelling van een drone en camera's simuleert. [1] De simulator genereert beelden van de drone in 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 rode 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.

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 algoritmen (sectie 2) en wordt de opbouw van onze software verduidelijkt (sectie 3). Ook is er extra informatie te vinden over de *GUI* (sectie 4). Er wordt afgesloten met de uitgevoerde testen te bespreken (sectie 5) en een kort besluit (sectie 6).

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 die de Autopilot volgt, verduidelijkt. Ook wordt er dieper ingegaan op gemaakte keuzes tijdens de ontwerpfase.

1.1 Drone Autopilot

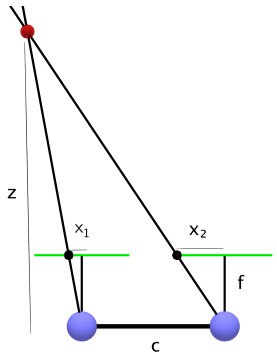
Auteur: Laura Vranken & Vincent Vliegen

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. Vervolgens zorgt de Autopilot ervoor dat de drone juist georiënteerd staat en naar zijn doel toe vliegt. Wanneer de drone zijn doel (in deze eerste fase is dit een rode bol) bereikt, moet hij daarin blijven zweven. Daarnaast moet de Autopilot ook rekening houden met een mogelijke invloed van wind die de drone van zijn koers doet afwijken.

Ten eerste moeten de beelden die de Autopilot van de Virtual Testbed binnenkrijgt, geanalyseerd

[1] Amazon Prime Air

worden. Dit gebeurt door iteratief de kleurwaarden van elke pixel te vergelijken met de waarde van de kleur rood. Alle rode pixels worden bijgehouden door hun positie ten opzichte van het beeld, uitgedrukt in rij en kolom, op te slaan. We baseren onze 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 de rode 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, dat de cirkel bepaalt die het beste past in de gegeven randpunten en bijgevolg de positie van het centrum van de bol. [2] 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 rode pixels detecteert, zal de drone geleidelijk 360 graden ronddraaien of m.a.w. een yaw beweging uitvoeren, totdat in beide beelden rode pixels verschijnen. Dan zal de Autopilot stoppen met draaien en zijn positie tegenover het doel berekenen.
Om dit te berekenen, wordt eerst de diepte bepaald. Dit kan met behulp van de formule van stereo vision [4] uitgewerkt worden. Zie Figuur 1 voor een grafische weergave van de berekening.



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

Vervolgens bepalen we de hoek waaronder de drone een yaw beweging moet uitvoeren om recht naar het doel gericht te zijn. Deze formule wordt afgeleid via de goniometrische regels. Zie Figuur 2 voor grafische ondersteuning.
Berekening brandpuntsafstand f:

$$\tan(\frac{\delta}{2}) = \frac{\frac{b}{2}}{f} \tag{1}$$

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

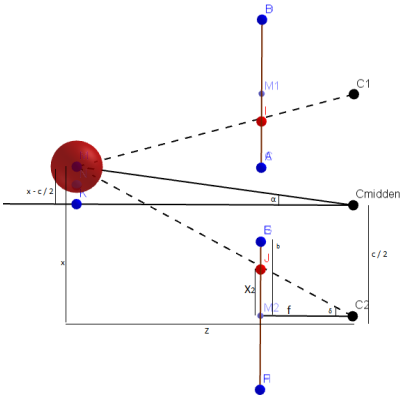
Berekening afstand x:

$$\frac{x_2}{f} = \frac{x}{z} \tag{3}$$

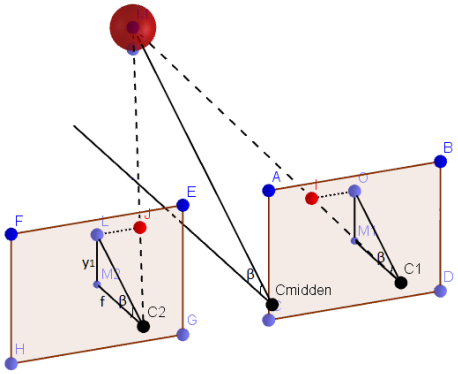
$$x = z * \frac{x_2}{f} \tag{4}$$

Om tenslotte naar het doel te kunnen vliegen, moet een evenwicht gevonden worden tussen pitch en thrust. De pitch wordt gekozen zodat het middelpunt van het doel nog juist in beeld blijft. Deze hoek is gelijk aan het verschil tussen de helft van de verticale hoek die het beeld overspant en de verticale hoek waaronder de bol zich tegenover de drone bevindt, zie Figuur 3.

Wanneer de pitch vastligt, kan de hoeveelheid thrust berekend worden zodat de drone in rechte lijn naar het doel kan vliegen. Voor een gedetaileerde uitwerking zie sectie 2.1.



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: $\tan(\beta) = \frac{y_1}{f}$.

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 en zich opnieuw oriënteren. Ook kan de wind ervoor zorgen dat de drone een roll uitvoert. Deze moet eerst gecompenseerd worden, vooraleer we verder onze berekeningen kunnen uitvoeren.
De drone bereikt zijn doel wanneer de Autopilot niets anders dan rode pixels opvangt. De drone zal dan de opdracht krijgen om zijn pitch te compenseren en vervolgens enkel via thrust de zwaartekracht tegen te werken.
Het effectief laten vliegen van de drone gebeurt in de simulator. De Autopilot zendt enkel de verhouding in graden per seconden en de thrust in Newton door. Hiermee worden de pitch, yaw en roll berekend en uitgevoerd. Het is slechts door herhaaldelijk te controleren hoe ver nog gedraaid moet worden, dat kan besloten worden wanneer de beweging volledig uitgevoerd is en wanneer er gestopt mag worden.

1.2 Virtual Testbed

Auteur: Bram Vandendriessche

Voor de simulator is de belangrijkste keuze uiteraard welke library het meest geschikt is voor het bouwen, weergeven en aanpassen van 3D-werelden. Hiervoor werden Blender[2], JMonkeyEngine[3] en OpenGL[4] onder de loep genomen.
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 zelf toch uitdagend en tijdrovend om onder de knie te krijgen. Aangezien we echter gepland hadden in Java te werken, moest gezocht worden naar een manier om Java en Python te verbinden. Bovendien zou Blender dan vanuit Java gestart moeten worden, wat een omslachtig proces bleek. Vooral omwille van de extra leertijd en deze laatste eigenschap 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. Helaas heeft deze *API* een erg beperkte community waardoor problemen vaak zonder hulp van het internet moeten worden opgelost.

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

Ondanks de moeilijke beginfase bij het leren van OpenGL werd toch hiervoor geopteerd. OpenGL kan rechtstreeks in Java worden gebruikt, heeft een tal van mogelijkheden, een brede community en er zijn heel wat tutorials beschikbaar.

2 Algoritmes

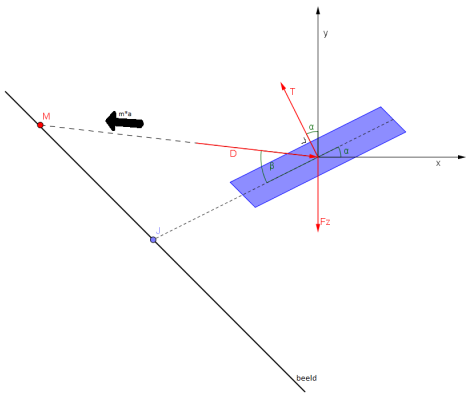
Auteurs: Jef Versyck; redactie: Arne Vlietinck

Om een realistisch 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 simulator als de Autopilot.

2.1 Drone Autopilot

Auteurs: Laura Vranken

De Autopilot gebruikt een algoritme om de hoeveelheid thrust te berekenen. De thrust wordt bepaald zodat de drone in een rechte lijn naar zijn doel vliegt. Zo vermijdt het onnodige bewegingen. Bovendien kan hierdoor ook de rode bol heel de tijd in beeld blijven. Een grafische voorstelling kan gevonden worden in figuur 4. Hierin is een drone onder een pitch α voorgesteld, het zwaartepunt M van de rode bol staat relatief ten opzichte van de drone onder hoek β en zijn ook de drag D, zwaartekracht F_z en thrust T voorgesteld.



Figuur 4: Grafische uitwerking van hoeveelheid thrust onder bepaalde pitch.

$$\begin{cases} T \cdot \sin(\alpha) - D \cdot \cos(\beta - \alpha) = m \cdot a \cdot \cos(\beta - \alpha) \\ T \cdot \cos(\alpha) - F_z - D \cdot \sin(\beta - \alpha) = m \cdot a \cdot \sin(\beta - \alpha) \end{cases} \tag{5}$$

$$\frac{T \cdot \sin(\alpha) - D \cdot \cos(\beta - \alpha)}{\cos(\beta - \alpha)} = \frac{T \cdot \cos(\alpha) - F_z - D \cdot \sin(\beta - \alpha)}{\sin(\beta - \alpha)} \tag{6}$$

$$T \cdot \sin(\alpha) \sin(\beta - \alpha) = T \cdot \cos(\alpha) \cos(\beta - \alpha) - F_z \cdot \cos(\beta - \alpha) \tag{7}$$

Algemeen geldt er:

$$\cos(x + y) = \cos(x) \cos(y) - \sin(x) \sin(y) \tag{8}$$

Door 8 toe te passen op vergelijking 7 wordt volgende relatie afgeleid:

$$T = \frac{F_z \cdot \cos(\beta - \alpha)}{\cos(\beta)} \tag{9}$$

2.2 Virtual Testbed

Auteurs: Jef Versyck

Het relatief assenstelsel hangt vast aan het centrum van de drone, zodat de linkercamera zich op de negatieve X-as bevindt en de rechtercamera op de positieve. De positieve Y-as staat loodrecht op de drone naar boven en de negatieve Z-as ligt in de diepte van het computerscherm. Het assenstelsel ondergaat identiek dezelfde bewegingen als de drone.

De drone kan drie bewegingen uitvoeren: roll, pitch en yaw. Het voert eerst zijn yaw uit, dan zijn roll en ten slotte zijn pitch. De yaw roteert rond de Y-as, de roll rond de Z'-as[5] en de pitch rond de X"-as[6]. Merk op dat er een verschil bestaat tussen de roll en pitch die de drone op een gegeven ogenblik heeft en de yaw, roll en pitch die de drone moet uitvoeren om in zijn huidige positie te geraken. Daarnaast zijn door de oriëntatie van het assenstelsel zowel de roll, pitch als yaw negatieve waarden.

Door de volgorde van de drie rotaties onstaat er een specifieke transformatiematrix die, vermenigvuldigd met de coördinaten van een punt (bv. het startpunt van de linkercamera), de coördinaten van het punt na de drie rotaties bepaalt. De verandering van de yaw/roll/pitch is ook afhankelijk van deze transformatiematrix. De verandering van de pitch is immers afhankelijk van de roll: hoe groter de roll, hoe trager de pitch verandert.

De bekomen rotatiematrix:

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

De berekening van de veranderingen van yaw, roll en pitch steunt op het volgend principe:

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

met R de transformatiematrix, R^{-1} de inverse van de transformatiematrix en x de algemene beweging. $R \cdot x$ is een relatieve beweging, zoals de verandering van de pitch. Deze relatieve beweging vermenigvuldigd met de inverse geeft de algemene beweging die de drone uitvoert voor die bepaalde verandering.

Elke drone heeft minstens twee krachten die erop uitgeoefend worden: de zwaartekracht en de thrust. De zwaartekracht is gelijk aan de massa maal de gravitatieconstante van de drone en zal altijd volgens de globale Y-as staan:

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

De thrust T daarentegen is afhankelijk van de huidige pitch en roll van de drone. Daarom zal deze vermenigvuldigd moeten worden met de transformatiematrix, om de huidige thrust te bekomen. Het resultaat van deze vermenigvuldiging is:

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

De windkracht, formule 10, is een kracht die een voorafbepaalde grootte en richting heeft. Deze kracht is optioneel in de implementatie.

[5] Z-as van het relatieve assenstelsel na yaw beweging.
[6] X-as van het relatieve assenstelsel na alle vorige bewegingen.

$$\vec{W} = \begin{Bmatrix} W_x \\ W_y \\ W_z \end{Bmatrix} \quad (10) \qquad \vec{F_d} = \begin{Bmatrix} v_x \cdot d \\ v_y \cdot d \\ v_z \cdot d \end{Bmatrix} \quad (11)$$

De wrijvingskracht, formule 11, is een kracht die evenredig is met de snelheid v en een constante drag d. Deze kracht werkt in tegen de snelheid, dus de constante is negatief.

Alle vectorkrachten die inwerken op een bepaalde drone worden vervolgens bij elkaar opgeteld. Deze vector, gedeeld door de massa van de drone, zal gelijk zijn aan de versnelling van de drone, volgens de formule 12

Ten slotte kunnen via de snelheids- en positievergelijkingen de huidige snelheid en positie berekend worden. Respectievelijk formule 13 en 14.

$$\sum_{1}^{n} \vec{F_i} = m \cdot \vec{a} \quad (12) \qquad v = a \cdot t + v_0 \quad (13)$$

$$x = a \cdot \frac{t^2}{2} + v_0 \cdot t + x_0 \quad (14)$$

3 Software

Auteurs: Arne Vlietinck; Redactie: Arne Vlietinck

Tijdens het programmeren, is het belangrijk om een duidelijke structuur voor ogen te houden. Dit leidt immers tot goede leesbaarheid voor buitenstaanders en bewaart overzichtelijkheid bij lange code. Ook werd er geprogrammeerd met het doel om later gemakkelijk extra functionaliteiten te kunnen toevoegen.
Om deze sectie beter te kunnen volgen, staan de klassendiagramma's onderaan deze sectie (zie figuur 5 en 6).

3.1 Drone Autopilot

Auteurs: Laura Vranken

Het deel van de Autopilot begint bij het creëren van een Autopilot in de *DroneAutopilotFactory*-klasse, geïmplementeerd met de interface *AutopilotFactory*. Hierin wordt zowel de Autopilot als *GUI* aangemaakt en worden de beginwaarden voor yaw, roll, pitch en thrust direct ingesteld.

Dit gebeurt d.m.v. een nieuw object *DroneAutopilot* die de interface *Autopilot* implementeert, aan te roepen. De *DroneAutopilot*-klasse bestaat uit één functie, namelijk *timeHasPassed* die continu door de simulator wordt uitgevoerd. Vanuit deze methode wordt de klasse *MoveToTarget* aangeroepen met de uit te voeren opdracht.

Deze klasse bepaalt de bewegingen en aansturing van de drone. Ook worden hier de verschillende rates doorgegeven aan de simulator. *MoveToTarget* steunt zowel op de informatie van de klasse *ImageCalculations* als van *PhysicsCalculations* om zijn bewegingen te bepalen. Bovendien wordt hierin ook telkens de *GUI* geüpdatet wanneer een nieuwe waarde voor de diepte bepaald is. *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 tenslotte alle fysische berekening van hoeken en afstanden uitgevoerd.

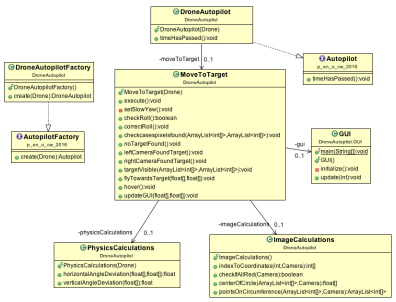
3.2 Virtual Testbed

Auteur: Bram Vandendriessche

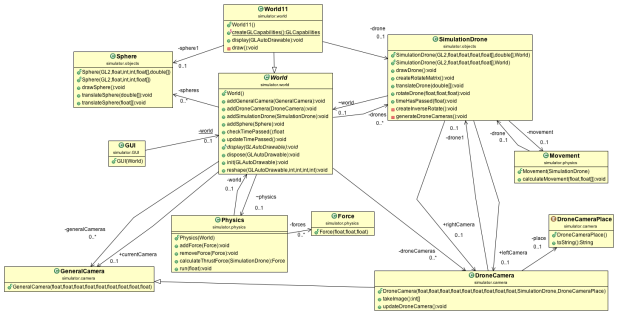
Voor de simulator kan de klasse *World* het hart van de structuur genoemd worden. Ze is geïmplementeerd als een subklasse van GLCanvas en vormt zo de basis voor het visuele gedeelte van de 3D-wereld die opgebouwd wordt met OpenGL. *World* houdt bij welke objecten (Camera, Drone, Sphere) er in de wereld bestaan en zal zorgen dat al deze objecten getekend worden door OpenGL.

Om te kunnen voldoen aan de specifieke opstelling voor elke mijlpaal wordt de setting telkens in een subklasse van *World* vastgelegd door implementatie van de OpenGL-functie *display*. Zo is de 3D-opstelling voor de eerste twee milestones wel gelijk, maar is er in *World12* een extra *Force* die in een willekeurige richting op de drone werkt om zo wind te simuleren. Deze *Force* maakt deel uit van de fysische omstandigheden in de wereld (gedefinieerd in de klasse *Physics*) waaronder bijvoorbeeld ook de zwaartekracht valt.

Uiteraard moet de gebruiker van de simulator op een eenvoudige manier een beeld krijgen van de huidige opstelling en situatie. Daarom zorgen camera's vanuit verschillende standpunten voor beelden die eenvoudig opgevraagd kunnen worden in de grafische interface (*GUI*).
De klasse *GeneralCamera* definieert de positie en de kijkrichting van de verschillende camera's en objecten van dit type. Zij geven een globaal beeld van de simulator. Een uitbreiding hiervan is de subklasse *DroneCamera* die twee beelden weergeeft vanuit het standpunt van de drone. Twee instanties worden immers aangemaakt bij initialisatie van elke *SimulationDrone*. De methodes van de interface *Camera* van de API voor de verbinding tussen Autopilot en simulator worden door *CameraDrone* geïmplementeerd. Hetzelfde gebeurt door de *SimulationDrone* voor de Drone-interface van de API.



Figuur 5: Klassendiagramma Autopilot.



Figuur 6: Klassendiagramma Virtual Testbed.

4 GUI

Auteurs: Arne Vlietinck; redactie: Arne Vlietinck

Om de interactie en visualisatie gebruiksvriendelijk te maken, worden beide programma's voorzien van een *graphical user interface* (*GUI*). Er wordt gebruik gemaakt van Java's Abstract Window Toolkit (*java.awt*) om gemakkelijk de lay-out te programmeren. In de opgave van het project [1] staan er reeds richtlijnen waaraan beide *GUI's* moeten voldoen.

4.1 Drone Autopilot

Auteur: Matthias Van der Heyden

De *GUI* van de Autopilot heeft tot nu toe twee functies: de gebruiker de mogelijk geven een

opdracht voor de drone te selecteren en de voortgang van de voltooiing van deze opdracht weergeven.

Voor het selecteren van een opdracht is er een dropdownmenu voorzien dat gebruik maakt van *JComboBox* uit de *Swing library*[7] 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. Op dit moment is de enige optie in het menu de opdracht om naar de rode bol te vliegen, maar een uitbreiding van mogelijke opdrachten voor volgende milestones is relatief eenvoudig.

Een *JProgressBar* uit de *Swing library* van Java geeft in de *GUI* de voltooiing van de geselecteerde opdracht weer. 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

Ook in het Virtual Testbed wordt een panel voorzien waarin de *GUI* vervat zit. In tegenstelling tot de Autopilot wordt hier gebruik gemaakt van een iets uitgebreidere lay-outvorm namelijk de *GridBagLayout*[8]. 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.
Daarnaast kunnen ook de dronecamera's apart geselecteerd worden. Zo kan de gebruiker van het programma gemakkelijk de bewegingen van de drone vanuit verschillende standpunten beleven.
Tot slot wordt ook de snelheid en de positie van de drone weergegeven. Deze wordt continu opnieuw opgevraagd en geüpdatet in de *GUI*.

5 Testen

Auteurs: Arne Vlietinck; redactie: Arne Vlietinck

Voor beide programma's zullen er uitvoerig testklasses gegenereerd en uitgevoerd worden. Hiermee kan de correcte werking en de nauwkeurigheid van de gebruikte algoritmes gecontroleerd worden. Ook eventuele programmeerfouten komen aan het licht en kunnen op deze manier aangepast worden.
Naar gelang de milestones uitdagender worden, zullen de testklassen striktere en hogere eisen stellen aan de geïmplementeerde testen. Voor deze eerste milestone werden er enkel testklasses voor de Autopilot gegenereerd.

5.1 Drone Autopilot

Auteurs: Vincent Vliegen

De Autopilot is voorzien van enkele *JUnit* testklassen. Deze testen de verschillende gebruikte methodes op hun nauwkeurigheid en correctheid. Zo is het eenvoudiger de capaciteiten van de Autopilot aan te tonen en de beperkingen duidelijk af te bakenen.

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

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 .jpg bestanden ter beschikking 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 algoritme dan bij berekeningen van het zwaartepunt van de zichtbare pixels. [2]
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

Auteurs: Jef Versyck

Vanwege het voortdurend veranderen van de simulator en alle bijhorende functies, is er nog geen tijd gevonden om alles op een fatsoenlijke manier te testen. Bijgevolg zijn alle testen (zoals positieverandering, berekening van de huidige rotaties) allemaal gebeurd in het programma zelf, zonder een aparte testklasse te gebruiken.

6 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] 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].