KU Leuven
Departement Computerwetenschappen

P&O: Computerwetenschappen

Eindverslag

Team:
Geel

Coomans Arno
(Coördinator)
Saniz Bernardo
(Secretaris)
Geens Tomas
Peeters Florian
Theuns Flor
Willemot Toon

Academiejaar 2017-2018

Samenvatting

Bernardo

De interessantste toepassingen van robots zijn die waarin de robots zelf met elkaar coördineren om de taak te volbrengen. Op dat moment kan bijna volledig abstractie gemaakt worden van de mens, wat efficiënter en/of veiliger kan zijn. Dit verslag beschrijft een simulatie van een pakjesbezorgingssysteem waarin een centrale eenheid taken verdeelt onder verschillende vliegtuigdrones. De afzonderlijke drones lossen dan zelf die gegeven taak op, namelijk een pakje van de éne luchthaven naar de andere brengen. Hoe de taken verdeeld worden en hoe de drones hun vlucht controleren, wordt verder in detail besproken.

Inhoudsopgave

1 Inleiding

Arno, Flor

Het doel van dit deel van het project is om meerdere drones autonoom tussen verschillende luchthavens te laten vliegen. Deze drones dragen pakketjes met een vaste oorsprong en bestemming. Er komen steeds meer pakketten bij, die telkens een drone toegewezen krijgen. Een centraal algoritme, de AutopilotModule, zorgt voor deze takenverdeling. Aandachtspunten hierbij zijn een nauwkeurigere motion planning voor de drones, het verwerken van meerdere drone inputs en outputs, en het voorkomen van deadlocks en desynchs bij de takenverdeling. Dit wordt allemaal verder in meer detail uitgelegd.

2 Ontwerp

2.1 Droneparameters

Bernardo

De meeste parameters lagen al vast, maar niet die in verband met de wielen van de drone. Het landingsgestel is gebaseerd op onze berekeningen voor de MQ-1 Predator. dampSlope en tyreSlope zijn verhoogd om het botsen van de drone bij landing te verminderen.

Tabel 1: Gebruikte AutopilotConfig

Parameter	**Vastgelegde waarde**
gravity	9,81
wingX	3,35
tailSize	5,11
engineMass	190,24
wingMass	102,80
tailMass	116,15
maxThrust	4500
maxAOA	15
wingLiftSlope	12,5
horStabLiftSlope	6,25
verStabLiftSlope	4,69
horizontalAngleOfView	120
verticalAngleOfView	120
nbColumns	200
nbRows	200

Parameter	**Gekozen waarde**
wheelY	-1,37
frontWheelZ	-2,10
rearWheelZ	1,00
rearWheelX	1,39
tyreSlope	50000
dampSlope	5000
tyreRadius	0,2
rMax	2000
fcMax	0,7

3 Pathfinding

Toon, Tomas

Vooraleer een drone mag vertrekken moet er een pad dat een aantal doelen bevat aan toegekend worden. De bepaling van zo een pad wordt in deze sectie besproken.

3.1 Pad tussen kubussen

De volgorde waarin de drone naar de doelen vliegt is op een greedy manier bepaald. Na ieder behaald doel berekent het programma welk doel het dichtstbij ligt en stelt dit als volgende doel. Dit doel is niet noodzakelijk het snelst bereikbaar, indien de drone een grote bocht moet maken bijvoorbeeld, is de werkelijke afstand aanzienlijk groter dan de berekende afstand. Deze optimalisatie kan nog geïmplementeerd worden. Het vliegen tussen twee doellocaties is opgesplitst in drie fases. In iedere fase houdt de autopiloot een aantal locaties bij die een pad vormen dat de drone moet volgen. In de eerste fase stijgt/daalt de drone tot hij op dezelfde hoogte zit als zijn doel. In de tweede fase draait de drone tot zijn heading richting doel gericht is. In de derde fase vliegt de drone rechtdoor tot het doel bereikt is.

Het uitwerken van de eerste fase is vrij eenvoudig. Het hoogteverschil tussen drone en doellocatie bepaalt of de drone moet stijgen of dalen. De $maxInclination$ of $maxDeclination$ van de drone bepaalt de afstand die nodig is om op dezelfde hoogte als de doel-locatie te komen. Met deze afstand en de heading van de drone wordt een (tussen)positie bepaald en aan het pad toegevoegd. De coördinaten van het punt na stijging zijn in vergelijking 1 berekend, waarbij *current* de beginlocatie is en *goal* de doellocatie. Het dalen gebeurt analoog.

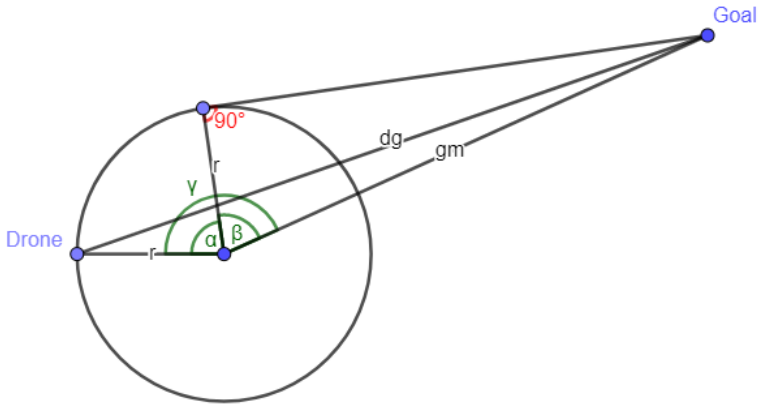
$$\begin{cases} x = current.x + \frac{sin(heading) \cdot (goal.y - current.y)}{maxInclination} \\ y = goal.y \\ z = current.z - \frac{cos(heading) \cdot (goal.y - current.y)}{maxInclination} \end{cases} \tag{1}$$

De tweede fase is minder eenvoudig. Eerst wordt gekeken of het doel binnen draaibereik van de drone ligt. Dit wordt bepaald a.d.h.v. de draaicirkel van de drone. Is dit niet het geval, laten we de drone rechtdoor vliegen tot de doellocatie wel binnen draaibereik ligt. Dan moet de drone bepalen hoelang hij de draaicirkel moet volgen opdat zijn heading naar het doel gericht is. In figuur 1 zoeken we α. De straal r is gekend. De afstand tussen het doel en het middelpunt gm en de afstand tussen de drone en het doel dg kunnen we berekenen met de formule voor afstand tussen twee punten. Nu passen we de cosinus-regel toe: $\gamma = acos(\frac{dg^2 - gm^2 - r^2)}{-2 \cdot r \cdot gm}$. Omdat de raaklijn van een cirkel een rechte hoek maakt met de straal geldt: $\beta = acos(\frac{r}{mg})$. En uiteindelijk $\alpha = \gamma - \beta$. Met deze hoek, de initiële heading en de straal van de draaicirkel weten we hoelang de drone moet draaien. De derde fase is simpelweg rechtdoor vliegen. Het laatste punt dat we toevoegen is het doel zelf.

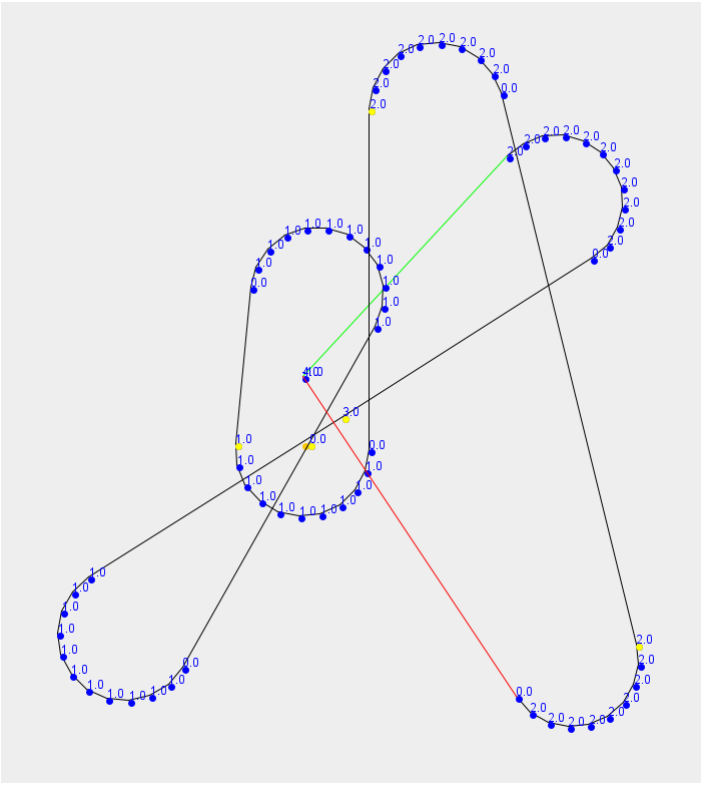
Het is belangrijk te beseffen dat het exact volgen van dit pad niet realistisch is, en ook niet de bedoeling is. Het eerste berekende pad is namelijk een best-case scenario, Waarin aangenomen is dat we precies weten hoe de drone vliegt, en de posities van de kubussen exact gekend is. Beide aannames zijn niet realistisch. De drone is onderhevig aan verschillende fysische wetten en bij de berekening van het pad zijn deze weinig in rekening gebracht. De gegeven posities van de kubussen zijn ook slechts een benadering met een nauwkeurigheid van 5m. Het eerste probleem lossen we op door het pad tijdens het vliegen te herrekenen. Zo worden afwijkingen van het initiële pad in rekening gebracht. Om het tweede probleem op te lossen schakelen we de beeldherkenning in.

3.1.1 Beeldherkenning

De beeldherkenning is in een vorige versie van het verslag reeds uitgebreid besproken. Het algoritme steunde op het feit dat de kubussen in de wereld eenheidskubussen waren. De nieuwe opgave vereiste dat de zijde van de kubussen 5m moet zijn. Dit heeft de nauwkeurigheid van het



Figuur 1: Afbeelding voor het berekenen van de draailengte.



Figuur 2: Een voorbeeld van een pad. Blauw zijn de berekende tussenlocaties. Geel zijn de doellocaties.

algoritme op een enorm negatieve manier beïnvloed, de nauwkeurigheid van het algoritme is zelfs zozeer achteruit gegaan dat het niet meer voldoet aan de vereiste van de opgave. De voornaamste reden hiervoor volgt uit de aannames die het algoritme maakt. Het algoritme stelt dat de twee uiterste punten op het beeld van de kubus een diagonaal vormen, en dat deze diagonaal loodrecht op de kijkrichting van de drone ligt. Vervolgens wordt aangenomen dat de uiterste punten van deze diagonaal samen met de positie van de camera een gelijkbenige driehoek vormen. Het algoritme kan dan op basis van driehoeksmeetkunde een afstandsschatting maken. Uit de resultaten van toen bleek dat het algoritme, met deze aannames, voldoende nauwkeurige resultaten opleverde maar wel traag naar de correcte positie convergeerde.

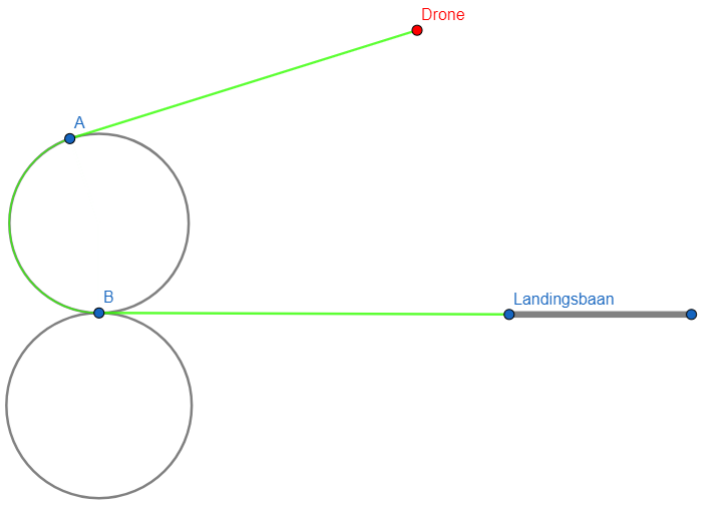
Het is deze trage convergentie die het voornaamste probleem geeft, en als gevolg van de aannames, inherent is aan het algoritme. Door de toegenomen onnauwkeurigheid van de afstandsschatting krijgt de drone pas een voldoende nauwkeurige positie wanneer hij ongeveer 13 meter van de kubus verwijderd is. Het is dan vaak al te laat voor de piloot om nog voldoende bij te sturen, waardoor het algoritme onbetrouwbaar is.

Om met beeldherkenning te kunnen sturen moet een beter algoritme geïmplementeerd worden. Tijdens de implementatie van het huidige algoritme bleek de afstandsschatting het grootste struikelblok. Op basis van één afbeelding is het enorm lastig om een afstand correct in te schatten. Een nieuw algoritme zou gebruik kunnen maken van twee onafhankelijke camerabeelden. Zo kunnen we dieptezicht simuleren om een nauwkeurigere afstandsschatting te krijgen. Het is wellicht ook mogelijk om met slechts één camerabeeld een betere schatting te verkrijgen, moest er meer rekening gehouden worden met perspectief en relatieve afmetingen van de kubus.

3.2 Pad naar landingsbaan

Tomas

Voor deel drie ligt de focus niet meer op het raken van kubussen, maar op het landen op de landingsbaan. Het vliegtuig kan niet zomaar in de richting van de landingsbaan vliegen. De autopiloot moet op voorhand plannen hoe het vliegtuig moet vliegen zodat het in de richting van de landingsbaan kan landen. Het vliegen gebeurt in twee delen die aan de hand van figuur 3 worden uitgelegd. Het eerste deel begint na het opstijgen en eindigt in punt A. Daarna volgt de drone een cirkel tot hij in de richting van de landingsbaan door punt B vliegt. Daarna kan hij overschakelen naar de landingspiloot. Hij kan deze cirkel ook blijven volgen als de landingsbaan bezet is (zie 5.5). Het bepalen van de draaicirkel hebben we experimenteel gedaan door de drone in een cirkel te laten vliegen. Punt B vonden we door de landing op meerdere afstanden en hoogten te testen. De exacte locatie van dit punt is minder belangrijk, zolang het een overschatting is van de werkelijk benodigde afstand tot de landingsbaan. Zo heeft de drone voldoende plek om te landen.



Figuur 3: De weg die het vliegtuig aflegt na het opstijgen.

Tijdens het eerste deel wordt in iedere tijdstap een doel-heading berekent naar punt A. Dit punt ligt op de draaicirkel waarvan de lijn naar de drone een raaklijn van de cirkel is. Naarmate de positie van de drone verandert kan ook de positie van punt A veranderen (vooral meteen na het opstijgen, dan komt de heading van de drone nog helemaal niet overeen met de doel-heading). Het is dus belangrijk dat we dit in iedere tijdsstap opnieuw berekenen. Er zijn kleine verschillen in de uitwerking als het vliegtuig links of rechts en voor of achter punt B ligt. In dit verslag werken

we het uit als het vliegtuig voor en links van punt B ligt. Op figuur 4 zoeken we naar α. De afstand tussen de drone en het middelpunt x en y, de afstand tussen de drone en punt B, kunnen we berekenen met de formule voor afstand tussen twee punten. Nu passen we de cosinus-regel toe:

$$\gamma = 2 \cdot \pi - \arccos\left(\frac{y^2 - x^2 - r^2)}{-2 \cdot r \cdot x}\right) \tag{2}$$

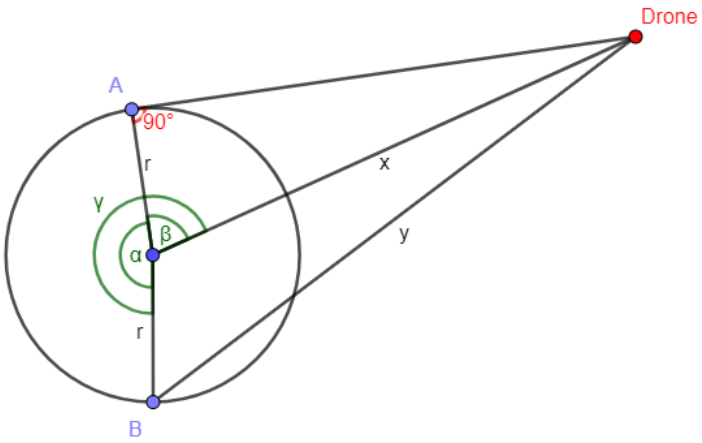
Omdat de raaklijn van een cirkel een rechte hoek maakt met de straal geldt het volgende:

$$\beta = \arccos\left(\frac{r}{x}\right) \tag{3}$$

En uiteindelijk:

$$\alpha = \gamma - \beta \tag{4}$$

Ten slotte combineren we α met de heading van de luchthaven om de doel-heading te berekenen.



Figuur 4: Afbeelding voor het berekenen van de draailengte.

Het tweede deel draait de drone naar links of rechts tot zijn heading overeenkomt met de heading van de luchthaven. Nadat gecheckt is of de luchthaven vrij is schakelt de drone over naar zijn landingspiloot, als de de luchthaven niet vrij is, vliegt hij een cirkel en probeert daarna opnieuw.

Als luchthavens dicht bij elkaar staan kan het zijn dat de drone niet op tijd op de draaicirkel kan vliegen en dan gaat hij in de cirkel. Op dit moment vliegt de drone dan rechtdoor tot hij terug uit de draaicirkel is. Dan begint hij opnieuw moet de doel-heading te berekenen. Theoretisch zal de drone maximaal twee keer een draaicirkel binnengaan (eenmaal iedere draaicirkel) voordat hij genoeg plaats heeft om op tijd te draaien.

4 Physics engine

Florian

De physics is ten opzichte van vorig semester weinig veranderd. Alle opgestelde vergelijkingen en oplossingsmethodes zijn onveranderd, en worden hier dus ook niet meer vermeld. De enige verandering bestaat uit de toevoeging van wielen aan het vliegtuig. Deze wielen zorgen voor een verticale kracht die het vliegtuig omhoog houdt als het op de grond staat, en eventueel een remkracht om te vertragen.

4.1 Wielen

De verticale kracht in elk wiel is gedefiniëerd met de volgende formule: $F_v = S_t \cdot D + S_d \cdot \frac{dD}{dt}$. Hier is S_t de *tyreSlope* en S_d de dampSlope, de waarde voor deze constanten wordt besproken in sectie 2.1. D is de radiale indrukking van de band. De remkracht op de wielen ligt in het grondvlak, en tegengesteld aan de snelheid van het wiel. Als het wiel geen snelheid heeft, zorgt de remkracht ervoor dat het vliegtuig niet beweegt. De grootte van de kracht wordt door de autopiloot ingesteld, met een waarde tussen 0 en de maximale remkracht. De twee achterste wielen ondervinden ook nog een zijwaartse wrijving, volgens de x-as van de drone. De grootte is gegeven door: $F_w = fcMax \cdot v \cdot N$. Hier is $fcMax$ de wrijvingsconstante van de wielen, v de snelheid van het wiel en N de grootte van de verticale kracht op het wiel.

In deel 3 worden luchthavens toegevoegd, en dit brengt met zich mee dat wielen enkel op een luchthaven de grond mogen raken. Om dit efficiënt na te gaan voor alle wielen, maken we gebruik van de veronderstelling dat er geen twee luchthavens tegen elkaar kunnen liggen. Dan weet de physics engine dat als een wiel van een drone op een bepaalde luchthaven staat, de andere wielen alleen op die luchthaven kunnen staan. Ook weet de physics engine dat als een drone in de vorige simulatiestap op een luchthaven was, de drone enkel op die luchthaven kan zijn, als hij niet opgestegen is.

Deze twee optimalisaties beperken het zoeken in de lijst van luchthavens tijdens een volledig bezoek (landen, taxiën en opstijgen) tot maximaal 3 keer, als hij de eerste keer de grond raakt.

4.2 Rotatieproblemen

Vorig semester waren er nog problemen met het draaien van de drone, en dit bleef moeilijk in het begin van dit semester. Toen we niet alleen tijdens het vliegen vreemd gedrag opmerkten, maar ook tijdens het taxiën, werd het duidelijk dat het probleem in de physics lag. Dit werd bevestigd door de drone bij het opstarten niet in de richting van de negatieve z-as te zetten, maar bijvoorbeeld de x-as. Dan begon de drone te draaien rond zijn z-as, wat niet de bedoeling is. Na nog een aantal testen werd het duidelijk dat de rotatie van de drone enkel correct was als hij volgens de negatieve z-as ging.

Na uitgebreid zoeken in de code, hebben we opgemerkt dat de rotatiesnelheid, die we intern berekenen in het drone-assenstelsel, niet getransformeerd werd om de rotatie van de drone te berekenen in het wereldassenstelsel. Dit levert inderdaad juist gedrag op als beide assenstelsels niet gedraaid staan ten opzichte van elkaar, maar vanaf dat ze gedraaid werden liep het mis. De oplossing was uiteindelijk zeer gemakkelijk, namelijk de rotatiesnelheid transformeren naar het wereldassenstelsel voordat we de rotatie ermee berekenen.

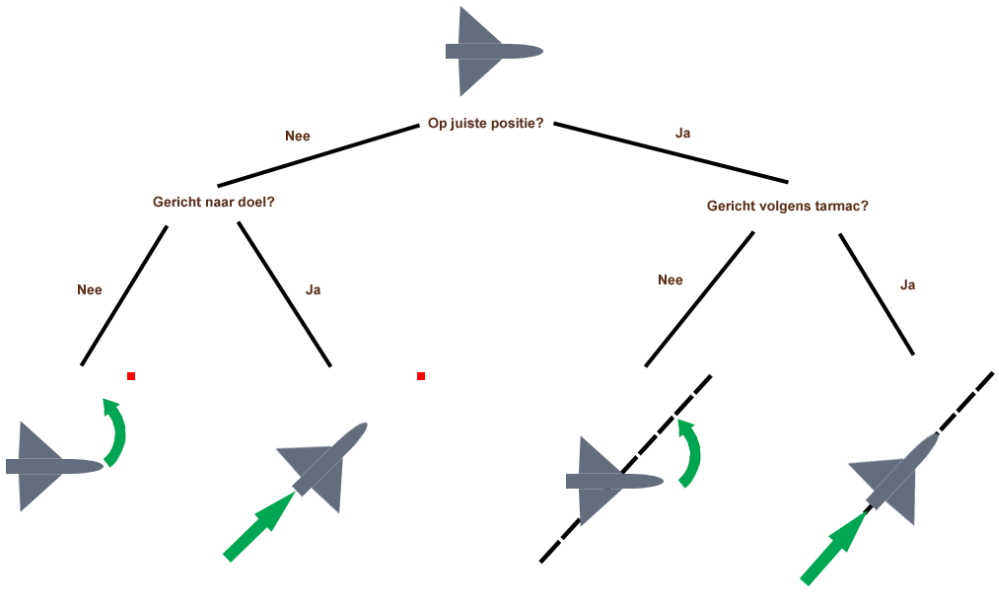
5 Motion Planning

De motion planning van de drone is in tegenstelling tot vorig semester niet meer gebaseerd op de image recognition maar op de gegeven coördinaten. De gegeven coördinaten worden met behulp van de pathfinding omgezet tot een reeks van commando's die in de motion planning zijn geïmplementeerd. Verder gaat de image recognition nog gebruikt worden om de locatie van de kubussen die niet helemaal nauwkeurig, is beter te bepalen en zo de vlucht lichtjes aan te passen. We splitsen de vlucht op in 4 delen: het taxiën, het opstijgen, het vliegen en het landen. Hieronder wordt elk onderdeel verder beschreven.

5.1 Taxiën

Bernardo

Taxiën is het over de grond rijden van een vliegtuig. Bij het taxiën kan de drone enkel manoeuvreren door gebruik te maken van zijn motor en remmen. Een eenvoudige en courante strategie hiervoor is differentiële sturing. Het principe hiervan gaat als volgt. Als het linker- en rechterwiel van een robot aan verschillende snelheden draaien, beschrijven ze allebei cirkels met verschillende straal. Op deze manier volgt de robot een cirkelboog tussen deze twee draaicirkels. De linker- en rechterrem asymmetrisch aanzetten zorgt voor dit snelheidsverschil.



Figuur 5: Het stappenplan gevolgd door de drone bij het taxiën.

Bij het taxiën zijn twee parameters van belang: de positie en de oriëntatie van de drone. De drone wordt dus aangedreven door twee checks. De eerste kijkt of het vliegtuig dichter dan $0.5W$ (in het xz-vlak) bij zijn doel staat. Als dit waar is ligt het massacentrum van de drone binnen de gate. Als uit de eerste check blijkt dat de drone nog niet binnen de juiste gate staat, moet hij eerst daar geraken. De tweede check verifieert of de drone naar het doel gericht is of niet. Zo niet, dan draait de drone zich tot dit wel het geval is. Zo ja, dan rijdt de drone recht vooruit tot hij op die positie komt. Eens de drone op zijn plaats staat draait hij zich op dezelfde manier tot het in de richting van de landingsbaan staat. Als de checks voor zowel positie als oriëntatie slagen, gaat de controle over naar de volgende state (opstijgen of stilstaan). Dit stappenplan wordt weergegeven in figuur 5.

Binnen een straal van $0.5W$ van zijn doel rijdt de drone aan de gevraagde 1m/s. Binnen een afstand W van het doel rijdt hij aan 3m/s en anders heeft de drone een snelheid van 10m/s. Dit

getrapt verloop houdt de drone controleerbaar. Om te vertragen worden alle remmen aangezet. Dit is niet meer problematisch aangezien de drone in principe zijn richting niet moet veranderen bij het rijden. De thrust wordt door een PID-controller aangedreven, maar de remmen dus niet meer.

Om volledig tot stilstand te komen worden de remmen eerst op maximale kracht aangezet. De snelheid wordt nog niet 0, maar wisselt op een bepaald ogenblik steeds tussen een klein negatief en een klein positief getal. Door de discrete aard van de simulatie wisselt de richting van de gesimuleerde remkracht dan steeds van teken. Eens dit wisselend gedrag gemeten wordt, wordt de remkracht gehalveerd. Dan daalt de absolute waarde van de wisselende snelheden. De autopiloot herhaalt dit tot de snelheid kleiner wordt dan een gevraagde waarde.

5.2 Opstijgen

Florian

Tijdens het opstijgen moet de drone uit stilstand vertrekken, om uiteindelijk een bepaalde hoogte te halen. Deze hoogte wordt bepaald door de pathfinding, en vanaf dan begint het volgende deel van de vlucht: het vliegen. Het volledige opstijgen verloopt in 1 richting, de autopiloot kan nog niet al draaiend klimmen. Dit lijkt ons wel perfect haalbaar, maar bleek voor de huidige taak niet nodig.

Het opstijgen wordt intern nog verder opgedeeld in twee eenvoudige onderdelen: versnellen en klimmen. Bij het versnellen wordt de motor op volle kracht ingesteld, en staan alle vleugels in neutrale stand om geen tegenkracht te veroorzaken. Vanaf het moment dat de drone genoeg snelheid heeft, wordt er omhoog gepitcht met de verticale stabilisator om te beginnen klimmen. De benodigde snelheid om te kunnen opstijgen hebben we experimenteel bepaald, en ligt momenteel op 120 km/u. De drone kan al vanaf een aanzienlijk lagere snelheid opstijgen, maar dan bleek het moeilijker om snel omhoog te pitchen. Een andere reden voor de ruime marge op de opstijgsnelheid is dat er momenteel geen limiet ligt op de lengte van de landingsbaan.

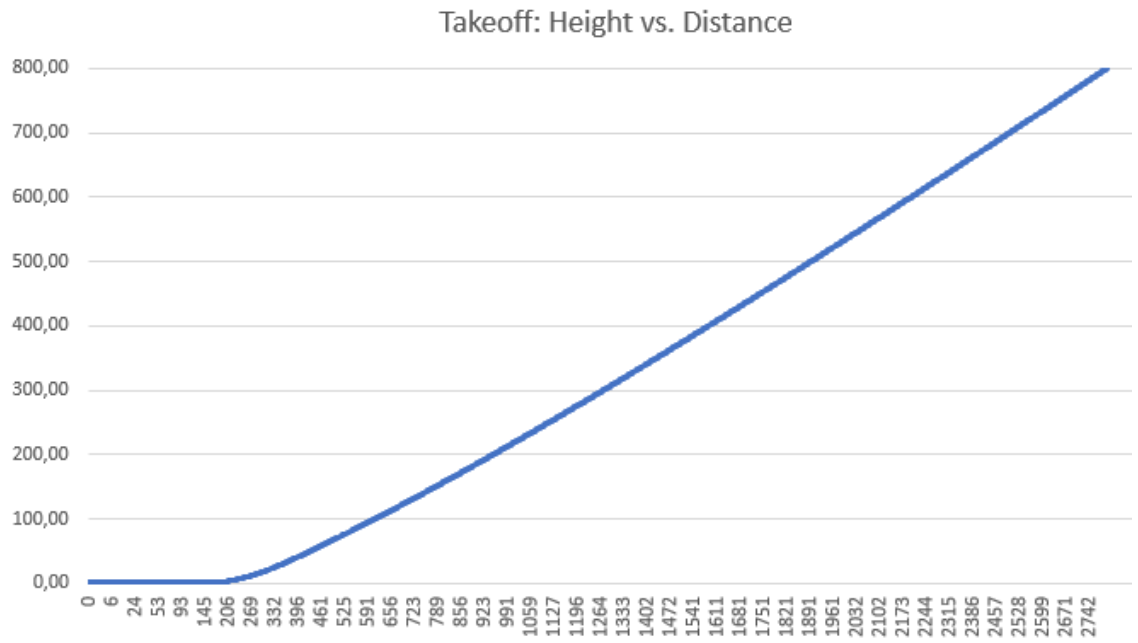
Men kan de benodigde afstand om een bepaalde snelheid te halen benaderen met formule 6. Deze formule komt uit het oplossen van het stelsel met de bewegingsvergelijkingen (zie vergelijking: 5). Hierin is m de massa, F de kracht van de motor, v de te behalen snelheid. Hiermee berekent men de versnelling a, de benodigde tijd t en de benodigde afstand x. Onze drone bijvoorbeeld zou 133.33 m nodig hebben. Experimenteel blijkt dat de vermelde formule eerder een absoluut minimum geeft, want om echt 120 km/u te halen is eigenlijk ongeveer 160 m nodig. Dit komt doordat de vleugels niet perfect horizontaal staan tijdens het versnellen, en dus weerstand opleveren.

$$\begin{cases} F = m \cdot a \\ v = a \cdot t \\ x = \frac{a \cdot t^2}{2} \end{cases} \tag{5}$$

$$x = \frac{v^2}{2 \cdot a} = \frac{m \cdot v^2}{2 \cdot F} \tag{6}$$

Vanaf het moment dat de drone omhoog begint te pitchen, wordt de instelling van de horizontale stabilisator geregeld door een PD-controller. Dit is een PID-controller waarbij er geen integrale component is. Deze wordt ingesteld op de gewenste klimhoek, en zorgt er dan voor dat de drone naar die klimhoek zal pitchen. Experimenteel bleek 20° een goede klimhoek te zijn, en waarden van respectievelijk 1.1 en 0.3 voor de P en D constanten in de controller gaven een goed klimgedrag. Om verder een constante lift te genereren met de vleugels, worden die vast ingesteld op een inclinatie van 6° tijdens het klimmen. Zo blijft de drone verder gaan totdat de gevraagde hoogte bereikt is.

Voor de pathfinding hadden we een schatting nodig van de afstand die de drone nodig heeft om tot een bepaalde hoogte te komen. Hiervoor is een grafiek (zie figuur 6) opgesteld van de bereikte hoogte in functie van de afstand. Uit die grafiek bleek dat het verband tussen beide praktisch



Figuur 6: Grafiek van het verloop van het opstijgen, met de bereikte hoogte op de y-as in functie van de afgelegde afstand.

lineair is als de hoogte hoog genoeg is. De functie bekomen met regressie levert dan de benodigde afstand in functie van de gevraagde hoogte op, te zien in vergelijking 7. Voor een hoogte onder 50m is geen van beide formules accuraat, maar dit zal normaal nooit voorkomen.

$$x = \begin{cases} 254.0966 + 3.5519 \cdot h & 50 < h <= 150 \\ 343.0420 + 3.0874 \cdot h & 150 < h \end{cases} \tag{7}$$

5.3 Vliegen

Flor

Het vliegen bevat het deel nadat het opstijgen het vliegtuig tot op een zekere hoogte heeft gebracht totdat het vliegtuig volledig wordt gestabiliseerd, een traject aflegt en dan een zo laag mogelijke snelheid heeft zodat het landen kan beginnen.

5.3.1 Raken van kubussen

Tussen opstijgen en landen moet het vliegtuig verschillende kubussen raken. De volgorde wordt op een "greedy", de dichstbijzijnde eerst, manier bepaald. Eens de kubus die het volgende doel is bepaald is gaat de drone zijn hoogte daaraan aanpassen door te stijgen of te dalen.

Deze bewegingen werden in 4 states opgedeeld: *StrongUp*, *Up*, *StrongDown* en *Down*. Nadat de drone zich op de juiste positie bevindt gaat hij ook zijn heading aanpassen zodat hij recht op de kubus afvliegt. Deze beweging bestaat eerst uit een controle of de kubus zich binnen de draaicirkel van de drone bevindt, als dit zo is volgt een bocht totdat de heading van het vliegtuig zich binnen een bereik van 10°van de *targetheading* bevindt. Dan gaat de drone zijn heading zo goed mogelijk bijwerken om het verschil zo klein mogelijk te maken. De states die hiervoor werden gebruikt zijn: *Left*, *SoftLeft*, *Right* en *SoftRight*. Om ervoor te zorgen dat er geen *AOAexceptions* optreden is er tussen de states een kleine stabilisatie-periode toe te voegen. Door de nieuwe parameters van de drone, die in het begin van het tweede semester werden opgelegd, moest de tuning van de PID's opnieuw gebeuren en moesten nieuwe waarden voor de snelheid en thrust worden bepaald.

5.3.2 Naar landingsbaan vliegen

De hierboven genoemde methode voor het vliegen is voor deel 2 geldig, hieronder gaan we de veranderingen die zijn gemaakt voor deel 3 toelichten. Hier gaat het niet meer over kubussen maar over de positie die moet worden gehaald voor een landingsbaan zodat de drone op de landingsbaan landt. In 3.2 wordt uitgelegd hoe deze positie wordt berekend en welke posities daarvoor moeten worden gehaald. Het behalen van die posities wordt op dezelfde manier gedaan als het behalen van de kubussen zoals hierboven besproken.

5.4 Landen

Tomas

Voor het vliegtuig begint met landen is het vliegtuig gestabiliseerd en heeft het geen roll en geen verticale snelheid meer. Dit wordt in het deel van het vliegen bekomen. Nu is een eerste stap om de maximale snelheid te bepalen waarmee het vliegtuig mag landen. We maken een opstelling waar de drone begint op de grond en een snelheid heeft volgens de negatieve y-as. Het vliegtuig crasht wanneer we deze snelheid boven 1.8 m/s kiezen, dit is dus de maximale landingssnelheid. Ten tweede moet de drone landen op de wielen die het dichtst bij het massacentrum liggen. Anders zou er een te groot moment veroorzaakt worden. In dit geval zijn dit de achterste wielen. Deze doelen kunnen we bereiken door het vliegtuig een positieve pitch te geven. Zo landt het vliegtuig op zijn achterwielen en als het vliegtuig te snel daalt kan het zijn thrust aanzetten en is er een acceleratie naar boven. Experimenteel vonden we dat een goede waarde voor deze pitch 5° is.

Eenmaal de drone de grond raakt, begint hij met remmen. Nu kunnen we berekenen hoe ver de drone zal bollen als alle remmen (3 in totaal) maximaal staan. De formule berekend in vergelijking 9 volgt uit stelsel 8. Als we de waarden van onze drone invullen met een startsnelheid van 40m/s, vinden we 192m.

$$\begin{cases} 3F = m \cdot a \\ 0 = a \cdot t + v_0 \\ x = \frac{a \cdot t^2}{2} + v_0 \cdot t \end{cases} \tag{8}$$

$$x = -\frac{v_0^2}{2 \cdot a} - \frac{v_0^2}{a} = -\frac{3 \cdot v_0^2}{2 \cdot a} = \frac{3 \cdot m \cdot v_0^2}{2 \cdot F} \tag{9}$$

Voor het laatste deel moet de drone ook kunnen landen op een landingsbaan. Het richten van de drone naar de luchthaven is niet altijd perfect. Daarom heeft de landingspiloot ook de mogelijkheid om een lichte roll te geven aan de drone met behulp van een PID. Verder is er ook een methode die controleert of het vliegtuig boven de landingsbaan vliegt zodat het niet te vroeg landt.

5.5 Collision prevention

Flor

We hebben twee maatregelen getroffen die botsingen tussen drones verhinderen. De eerste is relevant voor de drones die aan het vliegen zijn, we geven elke drone een verschillende hoogte in de lucht, een zogenaamde *airslice*. Zo gaan de vliegtuigen niet in de lucht tegen elkaar botsen. Om de drones voldoende ruimte te geven kozen we voor airslices van 10m verschil. De tweede maatregel heeft betrekking op vliegtuigen die willen landen. Voordat een vliegtuig zijn landing inzet vraagt het toestemming aan de airportmanager.

De airportmanager checkt 3 voorwaarden: ten eerste of er geen vliegtuigen al aan het landen zijn op dezelfde luchthaven, ten tweede of er vliegtuigen op de landingsbaan aan het taxiën zijn en ten derde of de gate vrij is waar hij moet landen. Er moet geen rekening worden gehouden met opstijgende drones aangezien het opstijgen en landen telkens op een andere landingsbaan gebeurt. Als er aan een van de drie niet voldaan is gaat de drone een volledige bocht maken totdat hij terug op de ideale positie voor een landing komt. Doordat de bocht van een vliegtuig heel stabiel is kunnen de drones dit zo lang als nodig volhouden.

Dit geeft geen 100% zekerheid, er kunnen nog steeds botsingen optreden als er toevallig een vliegtuig aan het opstijgen of landen is terwijl er een vliegtuig passeert. Deze kans is zo klein dat we deze hebben verwaarloosd, zeker als de luchthavens ver genoeg uit elkaar liggen en ze niet op een route tussen 2 andere luchthavens liggen.

6 Scheduling

Arno

De moeilijkheid in dit deel ligt in het feit dat de autopilot kant synchroon moet blijven met de testbed kant. Aangezien er in de interface tussen de twee geen *dronePickedUpPackage()* call is moeten we dit aan de autopilot kant zelf controleren.

Tijdens het ontwerpen waren er enkele problemen, randgevallen, die ervoor zorgden dat deze twee kanten toch soms asynchroon gedrag vertoonden. Om dit op te lossen hebben we enkele constraints opgelegd aan de package generator. Dit was de meest efficiënte manier om dit probleem te vermijden, opties zoals de individuele autopilots van actieve drones overschrijven wanneer er een desync plaats vond, verschoof alleen maar het probleem.

6.1 Synchronisatie Constraints

I Package Generator

(a) er mogen geen pakketjes van Airport A Gate 1/2 naar Airport A Gate 2/1 gaan.

(b) pakketjes spawnen niet op een positie waar al een drone staat. Behalve als alle drones stilstaan, om een mogelijke deadlock te vermijden in een situatie waar elke airport 2 drones heeft die stilstaan op de gates.

II Scheduler

(a) er mag maar één drone tegelijk onderweg zijn naar een airport A en gate G.

6.1.1 Verantwoording

I (a) deze hebben we toegevoegd omdat dit geen meerwaarde gaf aan het project, het is interessanter om drones te zien vliegen.

I (b) is toegevoegd zodat drones die geen pakketje aan het vervoeren zijn, niet ineens een pakketje opnemen terwijl we in onze scheduler deze drone al een andere taak hebben toegekend. Hierdoor is het genoeg om tijdens het afleveren te controleren of deze drone al op een nieuw pakketje staat en als dat het geval is die opnemen.

II (a) de redenering hierachter is dat drones op die manier nooit pakketjes van elkaar kunnen stelen. Anders kan je gevallen krijgen waar een drone in het midden van een vlucht zijn scheduling moet aanpassen omdat het pakketje naar waar hij onderweg was, al is opgepakt door een andere drone. We hebben het ons op deze manier ook niet té makkelijk gemaakt, er moet nog altijd gecontroleerd worden dat er maar één drone tegelijk kan landen.

6.1.2 Desync problemen

Momenteel is er nog één error die heel soms voorkomt waardoor de autopilot nog steeds asynchroon loopt naast het testbed. Dit is een geval waarbij de drone te dicht of op een gate landt terwijl deze drone op de andere gate moest zijn. Dit creëert een edge case waar een drone een pakketje van een andere drone kan stelen. Momenteel zijn we hier een hotfix voor aan het creëeren. Dit deel verdwijnt uit het finale verslag mocht het tegen dan opgelost zijn.

6.2 Naieve Scheduling Algorithm

Allereerst gaan we nakijken of de drone die momenteel klaar is niet al op de locatie van een pakketje staat dat hij zelf niet heeft opgepakt volgens de autopilot kant. Als dit het geval is dan moeten we niet schedulen en gewoon dat pakketje opnemen en dat afleveren.

Met deze constraints heeft de scheduler een makkelijke taak. We kijken voor elke Δ of er voor de pakketjes in de queue een nieuwe actieve drone is in deze volgorde:

1. staat er een ready drone op PakketjeFrom$_{Airport}$ en PakketjeFrom$_{Gate}$?
2. staat er een ready drone op PakketjeFrom$_{Airport}$?
3. is er ergens een ready drone?

We nemen dan de eerste optie die evalueert naar True. Deze drone wordt dan een pilot toegekend die weet hoe hij van zijn huidige locatie naar de andere locatie kan gaan en het scheduling algoritme is dus klaar.

7 Rendering

Arno

Naast het toevoegen van landingsbanen, wielen en een grond is er op het vlak van rendering niet veel veranderd. Deze onderdelen toevoegen heeft maar één probleem veroorzaakt, namelijk dat objecten die dicht bij elkaar staan alternerend op de voorgrond zouden gerenderd worden. Een oplossing hiervoor was de Z_{near} value van de view frustum verder weg zetten[1]. Zo verliest een camera detail van objecten die zich dichtbij bevinden. In OpenGL wordt het gebied dichtbij de camera gerendered met veel precisie, waardoor de Z_{buffer} vol staat met gedetailleerde informatie die in onze applicatie nooit gebruikt gaat worden. In onze huidige render engine betekende dat het volgende:

$$Z_{near} : 0.01 \rightarrow 0.7 \tag{10}$$

Aangezien de 70 centimeter voor de drone zo goed als verwaarloosbaar zijn voor image recognition is het geen probleem om het Z_{near} plane zo ver naar voor te duwen. Later kunnen we nu met gemak het Z_{far} plane tot $25km$ verhogen. Momenteel wordt die waarde wel op $10km$ gehouden om het renderen lightweight te houden.

7.1 Skybox en Grond

De top en right cam zijn heel onoverzichtelijk geworden door het landen en opstijgen, ook het verschil tussen vliegen en naar beneden vallen was niet helemaal duidelijk. Naast de grond die in de opgave gespecificeerd is hebben we ook, in de right ortho cam, een skybox toegevoegd met een texture waardoor we hoogteverschillen van de drone ook duidelijk kunnen maken. Deze texture in de lucht wordt niet getekend in de andere camera's voor efficiency en minder clutter in de grafische voorstelling.

7.2 Vleugel Rotatie

Al sinds het eerste semester was er geen rotatie van de drone vleugels. Dit lag aan het feit dat we een hoek probeerden te transformeren terwijl dit eigenlijk met de hoekpunten van de vleugels moest gebeuren. Dit is nu aangepast waar we de vleugel transleren naar het middelpunt van de wereld en daar de hoek gewoon optellen bij de $orientation_x$ van de vleugel

8 GUI

Florian

In deel 3 van het project werden er zowel voor het testbed als voor de autopilot nieuwe vereisten gesteld voor de GUI's. Al de aanpassingen die we maakten, worden in de volgende twee secties uiteengezet.

8.1 Testbed

Omdat er nu meerdere drones rondvliegen in de wereld, is een overzicht van alle drones een nodige uitbreiding van de GUI. Er is een tabel met alle drones bijgemaakt, die per drone zijn ID, naam, huidige locatie en eventuele ID van een opgepakt pakket laat zien. Een drone selecteren in de tabel zorgt ervoor dat het hoofd scherm die drone zal volgen. De pakketjes worden ook weergegeven in een tabel in de GUI, met per pakket de startlocatie, bestemming, en huidige status. Een pakket selecteren in de GUI verplaatst de freecam naar de huidige locatie van het pakket, als het nog niet afgeleverd is.

Om een extra overzicht te geven van de wereld, is er een minimap te zien onderaan in de GUI van het testbed. Hierop zijn alle drones en luchthavens als gekleurde vierkanten zichtbaar. De

luchthavens worden afgebeeld als een blauw vierkant met hun ID erboven, de drones door kleinere rode vierkanten met hun ID ernaast. De drone die momenteel gevolgd wordt in het hoofd scherm, wordt groen gekleurd op de minimap.

8.2 Autopilot

Om de synchronisatie tussen testbed en autopilot weer te geven, heeft de GUI van de autopilot ook een tabel met alle drones en een tabel met alle pakketten. Er wordt wel extra informatie getoond, om de huidige status van de planning weer te geven. Bij de drones worden de huidige taak van de piloot en het huidige doel extra weergegeven. De tabel van de pakketten laat ook hier zien wat de status van het pakket is in de scheduling, en aan welke drone het pakket is toegewezen.

De autopilotGUI van de vorige delen van het project is ook behouden, zodat de instelling van alle vleugels van een drone nog altijd gevolgd kunnen worden. Ook is het nog altijd mogelijk om een drone met de hand te besturen in de GUI.

9 Resultaten

Arno

Hier bespreken we kort enkele benchmarks die we hebben behaald dit semester.

9.1 3 Drones - 3 Airports

In een simulatie van één uur waarbij drie drones op drie airports vlogen behaalden we gemiddeld 40 geleverde pakketjes binnen het uur zonder enige desynchronisatie, botsing of package loss. Deze simulatie gebruikte een random package generator waar ongeveer elke minuut een nieuw pakketje verscheen in de wereld.

9.2 9 Drones - 9 Airports

Bij een run van 1u 45min waren er 130 pakketjes succesvol afgeleverd zonder enige de-synchronisatie problemen. Helemaal op het einde is een drone met zijn wiel op het gras geland, dit is te wijten aan het feit dat we onze simulatie op tien keer de snelheid aan het afspelen en zo door numerieke fouten op het gras geglitched zijn.

9.3 18 Drones - 9 Airports

Ook deze setup is uitvoerbaar voor het testbed. Toch begint het hier duidelijk te worden dat nog enkele optimalisaties nodig zijn om dit uit te voeren. Tegen de laatste demo gaan we nog proberen deze toe te voegen. Eentje daarvan is alle drones op hoogt 50 te laten vliegen en alleen drones naar een hogere airslice sturen wanneer ze te dicht bij elkaar vliegen. Momenteel gaan drones die té hoog vliegen, slechter landen waardoor de simulatie minder goed verloopt. Ook de scheduler loopt nu elke Δ over een lijst van packages wanneer er een drone ready is. Dit zorgt voor framedrops wanneer er veel drones tegelijk willen opstijgen.

9.4 Visuele test

De render engine kan zonder enig probleem met 50+ drones, zonder autopilot, rondvliegen. Dit gaf ons al een duidelijke upper bound waar we naartoe konden werken. Indien we dit getal nog hoger willen krijgen kunnen we met instanced rendering[2] veel extra drones de lucht in krijgen. Dit geld ook voor al de wereld objecten.

10 Besluit

Bernardo

Alle afzonderlijke componenten van de simulatie werken zoals verwacht. Pathfinding geeft een correct en haalbaar pad terug. Elke afzonderlijke stap van de motion planning werkt goed, en de drone gedraagt zich correct en stabiel als stappen gecombineerd worden. Textures voor de grond en landingsbaan maken de positie en snelheid van de drone duidelijk zichtbaar. Het samenbrengen van alle componenten is relatief vlot verlopen. Taken worden correct verdeeld en coördinatie is effectief, waardoor de simulatie gedurende lange tijd kan blijven draaien. In de toekomst zouden de pathfinding van de drones en efficiëntie van de takenverdeling kunnen verbeteren, maar de huidige versie voldoet aan alle vereisten uit de opgave. Het programma is bovendien voldoende robuust om een groot aantal drones te behandelen, of een klein aantal aan een hoge snelheid. In het algemeen is dit project zeker in zijn opzet geslaagd.

Referenties

[1] S. Baker, *Learning to love your z-buffer.* sjbaker.org, 2002.

[2] learnopengl.com, *Instancing*, 2018.