Tutorial UAVREG
Baanregeling Quadcopter

Rufus Fraanje

17/04/2017

1 Inleiding

Autopilot systemen zijn inmiddels zeer gangbaar. In vliegtuigen worden autopilots al geruimte tijd gebruikt. Ook de meeste drones zijn uitgerust met een autopilot systeem. In sommige gevallen dient de autopilot er slechts voor om de drone (veilig) terug te laten vliegen naar zijn thuisbasis, in veel gevallen kan met een autopilot een traject langs van tevoren bepaalde punten worden gevlogen.

In deze tutorial ga je zelf een autopilot besturing realiseren waarmee een quadcopter een traject kan afleggen. Dit zal worden gedaan met behulp van een aantal PID (proportionele, integrerende en differentiërende) regelaars en nog een aantal rekenkundige bewerkingen.

De leerdoelen van deze tutorial zijn:

- begrijpen hoe de verschillende regelaars samenwerken om de quadcopter te besturen en hoe dit geïmplementeerd kan worden in een Python simulatie;
- een PID regelaar kunnen implementeren;
- de parameters K_p, K_i en K_d van PID regelaars kunnen instellen door middel van 'tunen'.

De autopilot besturing zullen we hier ontwerpen in een simulatie in Python. Zo kunnen we ervaring opdoen, zonder dat de quadcopter door instabiel gedrag kapot gaat en kunnen we de situatie iets vereenvoudigen. In de praktijk is het namelijk vaak lastig de precieze positie en oriëntatie van de quadcopter exact te kennen, in deze simulatie kennen we de positie en oriëntatie exact.

Voor de simulatie maken we gebruik van twee Python modules:

- `pybullet`: een module waarmee de beweging van de quadcopter berekend kan worden (quadcopter dynamica), `pybullet` is een interface naar de krachtige (C++) bibliotheek `bullet` (`http://bulletphysics.org`) die ook wordt gebruikt als physics-engine in programma's als Blender en diverse computer games, zie bijvoorbeeld `https://en.wikipedia.org/wiki/Bullet_(software)`.
- `pyqtgraph`: een module voor (extreem) snelle visualisatie van 2D en 3D graphics, deze module maakt gebruik van OpenGL waarmee direct de grafische kaart kan worden aangestuurd (`http://www.pyqtgraph.org`);

De installatie van deze modules wordt uitgelegd in paragraaf 2. Het principe van de autopilot wordt aan de hand van een blokdiagram uitgelegd in paragraaf 3. Met behulp van de modules `pybullet` en `pyqtgraph` is een deel van de simulatie al geprogrammeerd, hoe dit is gedaan wordt uitgelegd in paragraaf **??**. De opdracht voor de tutorial vind je in paragraaf **??**.

2 Benodigde Python modules

2.1 `pybullet`

The module `pybullet` kan worden aangemaakt door de `bullet` bibliotheek te compileren. Dit is mogelijk zowel onder Windows als Linux en waarschijnlijk ook voor Mac OSX en iOS. Dit is niet heel ingewikkeld, maar kan toch tijdrovend zijn. Daarom is de module `pybullet` voor Windows (64-bit) en Linux (64-bit) al beschikbaar gemaakt in Blackboard. Voor Windows is dat de file `pybullet.pyd` en voor Linux `pybullet.so`. Plaats deze file in de python folder `site-packages` of in de folder waar je op dit moment werkt. Dit laatste is het makkelijkst, maar als je de module vaker gaat gebruiken is het beter deze in `site-packages` te zetten. Test of de module goed wordt ingelezen met `import`:

```
>>> import pybullet
>>> dir(pybullet)
['CONTACT_RECOMPUTE_CLOSEST', 'CONTACT_REPORT_EXISTING', 'DIRECT', 'GUI', 'JOINT_FIXED', 'JOINT_PLAN
```

Werp eventueel een blik op de help met `help(pybullet)`. Uitgebreide documentatie vind je in `https://docs.google.com/document/d/10sXEhzFRSnvFcl3XxNGhnD4N2SedqwdAvK3dsihxVUA/edit?usp=sharing` maar voor deze tutorial is het niet nodig deze door te werken.

Voor de quadcopter simulatie hebben we ook de volgende bestanden, beschikbaar via Blackboard, nodig:

- `plane.urdf` en `plane.obj`
- `quadrotor.urdf` en `quadrotor.obj`

De `urdf` bestanden zijn Universal Robot Description Format bestanden die objecten in een 3D wereld specificeren, zoals vorm en afmetingen, massa en traagheidsmoment. Het `urdf`-formaat is oorspronkelijk ontwikkeld voor het Robot Operating System (ROS) [**?**], maar wordt nu ook breder gebruikt. The `obj`-bestanden bevatten een mesh-beschrijving afkomstig van een 3D CAD programma, die wordt gebruikt in de `urdf`-bestanden.

De `bullet` bibliotheek heeft ook een visualisatie-mogelijkheid. Deze zullen we in het vervolg niet gebruiken, maar je kunt hiermee wel alvast controleren of het grondvlak (`plane.urdf`) en de quadcopter (`quadrotor.urdf`) goed weergegeven worden:

```
>>> import pybullet as p
>>> #choose connection method: GUI, DIRECT, SHARED_MEMORY
>>> p.connect(p.GUI)
0
>>> p.loadURDF("plane.urdf",0,0,-1)
0
>>> #load URDF, given a relative or absolute file+path
>>> quadcopterId = p.loadURDF("quadrotor.urdf")
```

Om een idee te geven wat je met pybullet kunt doen, laten we gedurende een fractie van een seconde een verticale kracht van 100 N (vector [0,0,100]) uitoefenen in het centrum van de quadcopter (positie [0,0,0]), en vervolgens 100 simulatiestappen uitvoeren. Dat kun je doen met de volgende code in Python (voer deze commando's uit op de Python command-line):

```
>>> #give a vertical impulsive force of 100 N at center of quadcopter
>>> p.applyExternalForce(quadcopterId,-1,[0,0,100],[0,0,0],p.LINK_FRAME)
>>> #do a 100 time-steps in the simulation to see the effect of the force impulse
>>> for _ in range(100): p.stepSimulation()
... #note we use _ here rather then a letter such as i, this is a habit in python
>>> #to represent a dummy parameter, that's not being used anymore
```

Experimenteer een beetje, bijvoorbeeld door te kijken wat er gebeurd als je nog meer `p.stepSimulation()` commando's uitvoert. Ook kun je eens proberen een kracht in een andere richting of op een ander punt op de quadcopter uit te oefenen of door gravitatie toe te voegen met `p.setGravity(0,0,-9.8)`. Voor sommige simulatie experimenten is het handig om de gravity vector gelijk aan 0 te maken, dat kan dus eenvoudig met `p.setGravity(0,0,0)`. De optie `p.LINK_FRAME` geeft aan dat de kracht wordt uitgeoefend op de positie in het assenstelsel gekoppeld aan het quadcopter object. Een alternatief is de optie `p.WORLD_FRAME`, waarbij de kracht wordt uitgeoefend op de positie in het assenstelsel van de vaste wereld (dit zullen we bij de quadcopter echter niet doen).

Maar zoals gezegd, zullen we niet gebruikmaken van de ingebouwde visualisatie in `bullet` en `pybullet`, aangezien deze nogal wat CPU-power vraagt en minder snel is dan `pyqtgraph`. Sluit dus het venster met het grondvlak en de quadcopter en ga verder met het installeren van `pyqtgraph`.

2.2 pyqtgraph

Voor het plotten van 2D en 3D grafieken heb je wellicht al gebruik gemaakt van de module `matplotlib`. Deze module is uitstekend voor het maken van allerlei grafieken (zie de `matplotlib`-gallery) maar minder geschikt voor 'real-time' visualisatie, omdat `matplotlib` hiervoor niet snel genoeg is. De module `pyqtgraph` is hiervoor beter geschikt, aangezien deze direct de grafische kaart kan aansturen met OpenGL en daardoor extreem snel is. Het installeren van `pyqtgraph` gaat het makkelijkst via het commando `pip install <python_module_naam>`. We hebben in totaal vier extra modules nodig, die niet standaard in de Anaconda distributie van Python geïnstalleerd zijn, dat zijn de modules `OpenGL`, `OpenGL_accelerate`, `pyqtgraph` en `trimesh`[1] De module `trimesh` hebben we nodig om de CAD beschrijvingen in de `obj`-bestanden om te zetten naar een mesh-structuur van driehoekjes die door `pyqtgraph` gevisualiseerd kan worden. Het installeren gaat alsvolgt. Open onder Windows een Windows Command Prompt Window en geef daarin het commando:

```
C:\ pip install PyOpenGL PyOpenGL_accelerate pyqtgraph trimesh
```

Onder Linux of Mac OSX open je een terminal en geef je voor user-niveau installatie het commando

[1] We hebben daarnaast ook de `math` module nodig die behoord tot de standaard library van Python en de `numpy` module, waarvan we aannemen dat deze ook reeds geïnstalleerd is. In de Anaconda distributie komt `numpy` standaard mee. Als `numpy` niet geïnstalleerd is, kan dat eenvoudig door middel van bijvoorbeeld het commando `pip install numpy`.

```
$ pip install PyOpenGL PyOpenGL_accelerate pyqtgraph trimesh
```

en voor installatie op system-wide niveau het commando

```
$ sudo -H pip install PyOpenGL PyOpenGL_accelerate pyqtgraph trimesh
```

Test of `pyqtgraph` goed is geïnstalleerd door enkele voorbeelden die met `pyqtgraph` meekomen uit te proberen, door middel van:

```
>>> from pyqtgraph import examples
>>> examples.run()
```

Om één of andere reden werkt `examples.run()` soms niet. In dat geval kun je de `pyqtgraph` voorbeelden ook starten vanuit de windows command line met het commando:

```
C:\ python -m pyqtgraph.examples
```

Als het goed is zie je nu een menu met verschillende voorbeelden. Probeer in elk geval het voorbeeld `Basic Plotting` en onder 3D Graphics het voorbeeld `Mesh`. Bij het `Mesh` voorbeeld moet er een venster met een aantal 3D objecten verschijnen.

Het is handig om even na te gaan hoe je met de muis het beeld kan verschuiven, roteren en in- en uitzoomen. Uit de documentatie van `pyqtgraph`, zie `http://www.pyqtgraph.org/documentation/mouse_interaction.html`, halen we de volgende regels:

```
3D visualizations use the following mouse interaction:
  • Left button drag: Rotates the scene around a central point
  • Middle button drag: Pan the scene by moving the central “look-at” point
    within the x-y plane
  • Middle button drag + CTRL: Pan the scene by moving the central “look-at”
    point along the z axis
  • Wheel spin: zoom in/out
  • Wheel + CTRL: change field-of-view angle
And keyboard controls:
  • Arrow keys rotate around central point, just like dragging the left mouse button
```

Om met de muis het beeld te kunnen aanpassen en het beeld voortdurend te verversen is het nodig de GUI event loop voor `pyqtgraph` te starten. In de voorbeelden wordt dit gedaan door middel van een paar commando's. Wij zullen dat doen door de GUI event loop van tevoren te starten. Als je gebruik maakt van Spyder kan dat eenvoudig op de volgende manier (hoef je maar één keer te doen):

> Ga naar het `Tools` menu, kies `Preferences`, kies vervolgens `IPython console`, selecteer de `Graphics tab`, en kies dan vervolgens als `Graphics backend` de keuze `Qt` backend.

Zorg er vervolgens voor dat je de `ipython` console gebruikt als Python interpreter (dit is standaard in Spyder, maar gebruik dus niet de python console). Als je de `ipython` console gebruikt, ziet de command prompt er als volgt uit (of vergelijkbaar):

```
Python 3.5.2 (default, Nov 17 2016, 17:05:23)
Type "copyright", "credits" or "license" for more information.

IPython 5.1.0 -- An enhanced Interactive Python.
?         -> Introduction and overview of IPython's features.
%quickref -> Quick reference.
help      -> Python's own help system.
object?   -> Details about 'object', use 'object??' for extra details.

In [1]:
```

De `ipython` console heeft een groot aantal handige opties. Probeer bijvoorbeeld eens de volgende commando's om de huidige folder en de inhoud daarvan te bekijken:

```
In [1]: pwd
In [2]: ls
```

Heel soms werkt de `<Enter>` toets in `ipython` niet naar behoren. In dat geval maak gebruik van `<Shift><Enter>` in plaats van `<Enter>`. Ook handig is het commando `cd`, waarmee je naar een andere folder kan gaan (vergelijkbaar met het Linux terminal-commando `cd`).

De GUI event loop kun je ook starten vanuit de `ipython` console, dat gaat met de `gui` magic[2], geef daarvoor het volgende commando (raadpleeg eventueel ook de help met `%gui?`):

```
In [3]: %gui qt5      # comment: sommige oudere python versies gebruiken
                      # qt4 ipv qt5, in dat geval geef de optie qt of qt4
In [4]: %gui?
```

Echter op deze manier blijft de instelling niet behouden, dus je kunt beter de instelling via de `Preferences` onder het `Tools` menu maken.

Je zult soms de Python kernel van de `ipython` console opnieuw willen starten. Dat kun je eenvoudig doen met de toets-combinatie `<Ctrl>.` (dus de control-toets en de punt-toets tegelijk indrukken).

3 Quadcopter regeling

3.1 Vrijheidsgraden van de quadcopter

Voordat we daadwerkelijk beginnen met de quadcopter simulatie, laten we eerst naar de principes van de quadcopter autopilot besturing kijken. Er zijn verschillende mogelijkheden, maar wij zullen gebruik maken van een schema gebaseerd op een aantal PID (proportionele, integrerende en differentiërende) regelaars, voor elke vrijheidsgraad één. Dus in totaal hebben we zes PID regelaars, één voor de x-richting, één voor de y-richting, één voor de z-richting, één voor de roll, één voor de pitch, en één voor de yaw. De roll, pitch en yaw vrijheidsgraden zijn rotaties om respectievelijk de x, de y en de z as van het assenstelsel dat vast gekoppeld is aan de quadcopter (locaal assenstelsel). De roll, pitch en yaw worden ook wel de XYZ Euler hoeken genoemd.

[2]Magics zijn uitbreidingen van de python commando's in `ipython`. Een andere handige magic is de `run` magic, waarmee python-scripts uitgevoerd kunnen worden. Voor meer informatie over magics raadpleeg de `quickref` magic.

De quadcopter heeft echter maar vier motoren, die onafhankelijk kunnen worden aangestuurd. Dat betekent dat we op elk moment maar vier vrijheidsgraden kunnen besturen, dat zijn:

- **thrust**, som van de krachten van de motoren, levert versnelling loodrecht op het vlak van de quadcopter;
- **roll**, moment om de x-as van de quadcopter, veroorzaakt door het koppel van de twee motoren langs de y-as, levert een hoek-versnelling om de x-as van de quadcopter;
- **pitch**, moment om de y-as van de quadcopter, veroorzaakt door het koppel van de twee motoren langs de x-as, levert een hoek-versnelling om de y-as van de quadcopter;
- **yaw**, moment om de z-as van de quadcopter, veroorzaakt door som van de momenten van de vier motoren, levert een hoek-versnelling om de z-as van de quadcopter.

Als de quadcopter niet gekanteld is, ofwel geen roll en pitch maakt, en dus de verticale z-as die vast zit aan de quadcopter parallel staat met de z-as van de vaste-wereld, dan kunnen we met de **thrust** de hoogte regelen. Dit wordt behandeld in de volgende paragraaf.

Daarna zullen we de algemene situatie behandelen, waarin we eerst een kracht-vector berekenen met de PID regelaars voor positie in de vaste-wereld, en vervolgens de quadcopter kantelen zodat de quadcopter loodrecht op deze kracht-vector komt te staan. Deze krachtvector kan dan dus gerealiseerd worden door de juiste hoeveelheid thrust uit te oefenen.

Om de autopilot niet te complex te maken, zullen we de yaw-regeling buiten beschouwing laten, ofwel de yaw zal constant op 0 rad geregeld worden[3].

Vanaf nu, zullen we ook aannemen dat de vier motoren alleen een kracht uitoefenen. We laten dus airodynamische aspecten en de toerentalregeling buiten beschouwing. Dit is een benadering, waardoor de autopilot niet één op één naar de praktijk overgezet kan worden. De principes achter de PID regelingen zijn echter ook in de praktijk zeer goed bruikbaar.

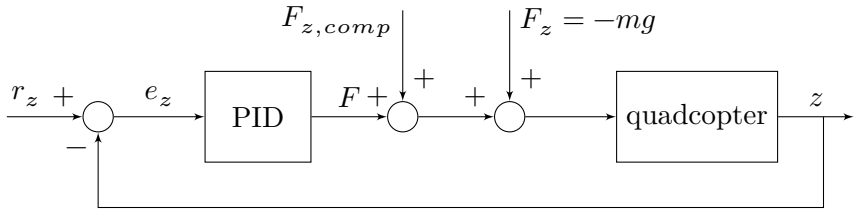
3.2 Hoogte regeling

Bij de regeling van de hoogte laten we de overige vijf vrijheidsgraden buiten beschouwing en nemen we aan dat de quadcopter horizontaal staat. Figuur 3.2 geeft het blokschema van de hoogte regeling, waarbij de referentiewaarde r_z de gewenste hoogte is, z de daadwerkelijke hoogte van de quadcopter, e_z de regelfout, F de regelkracht berekend door de PID regelaar, $F_{z,comp}$ is een compensatie term in de besturing waarmee de zwaartekracht gecompenseerd kan worden en tot slot $F_z = -mg$ is de zwaartekracht. Merk op, dat we voor zowel de krachten als de positie de vertikale richting omhoog de positieve richting is. Merk ook op, dat alle signalen, met uitzondering van F_z, afhangen van de tijd. Het is dus eigenlijk $r_z(t)$ in plaats van r_z, beide notaties zullen we gebruiken.

Het model voor de hoogte z van de quadcopter wordt gegeven door:

$$m\ddot{z} \;=\; F + F_{z,comp} - mg$$

[3]De yaw moet wel gestabiliseerd worden, omdat de quadcopter uit zichzelf instabiel is. Ofwel, een zeer gering moment in de yaw-richting leidt al tot het weglopen van de yaw, waardoor deze niet meer constant op 0 rad staat.



Figuur 1: Blokschema voor de hoogte regeling van de quadcopter (met zwaartekracht $Fz = -mg$ en zwaartekrachtcompensatie $F_{z,comp}$).

Kiezen we voor de zwaartekrachtcompensatie precies $F_{z,comp} = mg$, dan valt de zwaartekracht volledig weg uit de vergelijking en houden we over

$$m\ddot{z}(t) \;=\; F(t) \tag{1}$$

Omzetten naar het s-domein levert voor de hoogte (merk op: $F(s)$ is de s-getransformeerde van $F(t)$):

$$Z(s) \;=\; \frac{1}{ms^2}F(s) \tag{2}$$

De PID regeling wordt gegeven door de vergelijking:

$$F(t) \;=\; K_p e_z(t) + K_i \int_0^t e_z(\tau)\mathrm{d}\tau + K_d \dot{e}_z(t) \tag{3}$$

waarbij duidelijk de proportionele, integrerende en differentiërende term zijn te onderscheiden. In de simulatie moeten de integraal en de afgeleide van het foutsignaal $e_z(t)$ benaderd worden, omdat er met discrete tijdstappen gewerkt wordt. Dit is één van de opdrachten in de tutorial, zie paragraaf 5.1.2.

Het doel van het ontwerpen van een PID regelaar, is het bepalen van de waarden van de parameters K_p, K_i en K_d. In het college Regeltechniek zullen de waarden van K_p, K_i en K_d op een systematische manier ontworpen worden. In de tutorial UAVREG zullen de waarden bepaald worden door middel van tunen en tuningsregels. Om een beeld te krijgen van de invloed van de regelparameters, laten we eerst de invloed van K_p en K_d nagaan, waarbij $K_i = 0$. In dat geval wordt de PID regelaar een PD regelaar:

$$F(t) \;=\; K_p e_z(t) + K_d \dot{e}_z(t) \tag{4}$$

Zetten we deze vergelijking om naar het s-domein, met $F(s)$ de s-getransformeerde van $F(t)$ dan krijgen we:

$$F(s) \;=\; (K_p + K_d s)E_z(s) \tag{5}$$

Combineren we nu vergelijking (2) met (5) en gebruiken we het blokschema dan vinden we voor de gesloten-lus overdracht van $R_z(s)$ naar $Z(s)$ (Ga dit na!):

$$Z(s) \;=\; \frac{K_p + K_d s}{ms^2 + K_d s + K_p}R_z(s) \tag{6}$$

en voor $R_z(s)$ naar de regelfout $E_z(s)$ vinden we (Ga ook dit na!):

$$E_z(s) = \frac{1}{ms^2 + K_d s + K_p} R_z(s) \tag{7}$$

Vergelijken we dit met de overbrengingsfunctie van een massa-veer-demper systeem met massa m, dempingsconstante c_w en veerconstante c_v

$$\frac{1}{ms^2 + c_w s + c_v}$$

dan zien we dat invloed van K_d vergelijkbaar is met die van de dempingsconstante c_w, en de invloed van K_p vergelijkbaar met die van de veerconstante c_v. De proportionele regelactie maakt de regelkring dus 'stijver' en de differentieërende regelactie zorgt voor meer demping (minder oscillatie).

Merk ook op dat als de waarden K_p en K_d erg groot worden ten opzichte van m, de invloed van de massa kleiner wordt en de overbrengingsfunctie steeds meer naar 1 nadert, ofwel $Z(s) \approx R_z(s)$. In het vak regeltechniek leer je dat deze benadering alleen geldt voor lage frequenties, $s = j\omega$ met $\omega \ll \sqrt{K_p/m}$.

In de praktijk, vanwege vertraging en andere niet-idealiteiten, kunnen K_p en K_d echter niet onbeperkt vergroot worden, zodat de prestatie altijd beperkt blijft. Wel kan de bovenstaande analyse gebruikt worden om een eerste schatting te maken van de waarde van K_p en K_d. Bijvoorbeeld, als de massa m bekend is, en de gewenste trillingsfrequentie $\omega_n = \sqrt{K_p/m}$ en de gewenste demping $2\zeta\omega_n = K_d/m$, met ζ de dempingsverhouding, dan kunnen hiermee K_p en K_d berekend worden. Deze manier van regelaarontwerp wordt ook wel pole-placement genoemd, omdat hiermee de polen van het geregelde systeem op de gewenste plek (bepaald door ω_n en ζ) in het complexe vlak worden gelegd.

Tot nu toe hebben we het nog niet gehad over de integrerende actie, en de waarde van K_i. Integrerende actie wordt doorgaans gebruikt om de statische regelfout te verminderen of helemaal naar 0 te laten gaan. De keerzijde is dat dit vaak resulteert in een sterker oscillerend gedrag en soms zelfs in instabiliteit.

Als er een beginschatting is van K_p, K_d en eventueel ook K_i, dan kunnen deze waarden aangepast worden al naar gelang welk prestatiecriterium verbeterd dient te worden. Tabel 1 geeft weer wat het effect is van het vergroten van de regelparameters K_p, K_i en K_d op verschillende prestatiecriteria. Gebruik deze richtlijnen bij het tunen van de PID regelaars in de quadcopter. De tabel is afkomstig van een overigens zeer nuttige Wikipedia website over PID regelaars.

Er is nog meer vuistregels voor het instellen van PID regelaars. Een veelgebruikte vuistregel is de Ziegler-Nichols methode. Bij deze methode wordt eerst de proportionele versterkingsfactor langzaam verhoogd tot een bepaalde ultieme versterkingsfactor K_u, waarbij het systeem begint te oscilleren of oscilleert op een maximale frequentie. Nog verder verhogen van de versterking zou leiden tot schadelijke oscillaties of zelfs instabiel gedrag. De oscilatie periode die bij de ultieme versterkingsfactor K_u hoort wordt gemeten en aangeduid met de letter T_u. Op basis van K_u en T_u kunnen nu de regelparameters van P, PI en/of PID regelaars worden bepaald, zoals aangegeven in Tabel 2. Hoewel de Ziegler-Nichols tuningsregels erg nuttig kunnen zijn in de praktijk, leiden ze niet altijd meteen tot het gewenste regelgedrag en zullen de regelparameters naderhand bijgesteld moeten worden.

Tabel 1: Effect van onafhankelijk een parameter verhogen (Bron: Wikipedia `https://en.wikipedia.org/wiki/PID_controller`).

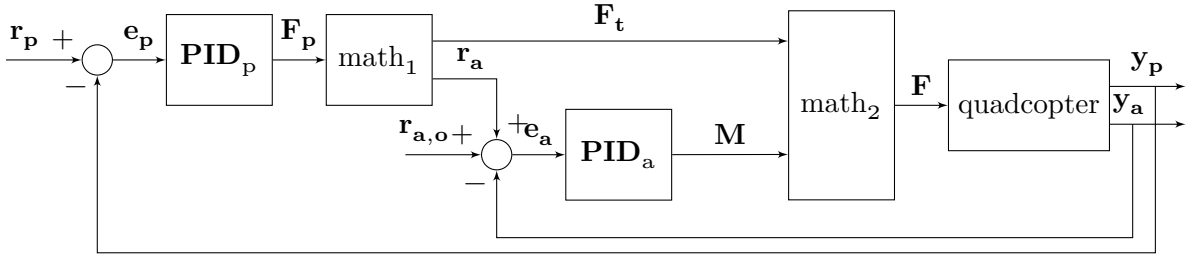
Parameter	Reistijd	Doorschot	Insteltijd	Statische fout	Stabiliteit
K_p	Afname	Toename	Kleine verandering	Afname	Vermindert
K_i	Afname	Toename	Toename	Verdwijnt	Vermindert
K_d	Nauwelijks	Afname	Afname	Geen invloed	Verbetert, mits K_d klein

Tabel 2: Ziegler-Nichols tuning methode voor P-, PI- en PID-regelaars (Bron: Wikipedia `https://en.wikipedia.org/wiki/PID_controller`).

Control Type	K_p	K_i	K_d
P	$0.50K_u$	-	-
PI	$0.45K_u$	$0.54K_u/T_u$	-
PID	$0.60K_u$	$1.2K_u/T_u$	$3K_uT_u/40$

3.3 Volledige autopilot regeling

Het blokschema voor de volledige autopilot, ofwel de regeling van de quadcopter, wordt weergegeven in Figuur 2. Voor het gemak zijn de zwaartekracht en de zwaartekrachtcompensatie weggelaten. Op het eerste gezicht ziet het blokschema er nogal complex uit, maar de globale werking is toch vrij goed te doorgronden als we er even wat langer bij stilstaan. Wat allereerst opvalt, zijn twee terugkoppelkringen met PID regelaars. Verder zien we blokken met math_1 en math_2, daar komen we dadelijk nog op terug. En natuurlijk weer de quadcopter. De signalen zijn allemaal vetgedrukt, omdat we hier te maken hebben met vectorsignalen, ofwel we hebben te maken met een multichannel systeem, ook wel MIMO (multiple-input-multiple-output) genoemd. De referentie-ingang $\mathbf{r_p}$ bestaat uit de referentie-waardes voor de positie (vandaar de underscore p), dus voor de x-richting, de y-richting en de z-richting, allemaal in



Figuur 2: Blokschema voor de regeling van de quadcopter (zonder zwaartekracht en zwaartekrachtcompensatie).

het assenstelsel van de vaste wereld, ofwel

$$\mathbf{r_p} = \begin{bmatrix} r_x \\ r_y \\ r_z \end{bmatrix}$$

Zo bestaat ook de uitgang $\mathbf{y_p}$ uit drie waardes, en wel de actuele positie in de x-, de y- en de z-richting in het assenstelsel van de vaste wereld, ofwel

$$\mathbf{y_p} = \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

De fout in de positie $\mathbf{e_p}$ is op vergelijkbare manier gedefinieerd.

Het blok $\mathbf{PID}_p$ bestaat dus uit drie PID regelaars, voor de x-, de y- en de z-richting. De uitgang is dan de vector $\mathbf{F_p}$ in x, y en z-richting

$$\mathbf{F_p} = \begin{bmatrix} F_{p,x} \\ F_{p,y} \\ F_{p,z} \end{bmatrix}$$

met de kracht die we willen uitoefenen op de quadcopter. Als deze kracht $\mathbf{F_p}$ loodrecht op het vlak van de quadcopter staat, dan is deze kracht de thrust $\mathbf{F}_t$ die we met de vier motoren kunnen uitoefenen. Maar heel vaak zal dat niet het geval zijn, en moeten we de quadcopter eerst kantelen voordat deze kracht loodrecht staat op het vlak van de quadcopter.

De gewenste kantelhoek, zowel in de roll als in de pitch richting kunnen we echter berekenen. Deze berekening wordt gedaan in het blokje math_1 en is al geprogrammeerd in de voorbeeldcode. De wiskunde die hiervoor nodig is maakt gebruik van Euler-hoeken en rotatiematrices, voor nu gaat dat echter te ver (deze wordt behandeld in de minor Robotics and Vision Design). Wel moet je natuurlijk met het resultaat kunnen werken.

Het eerste resultaat van math_1 is het deel van $\mathbf{F}_p$ wat al wel kan worden uitgevoerd door thrust van de quadcopter, ofwel $\mathbf{F_t}$ (om precies te zijn: de projectie van $\mathbf{F}_p$ op de lijn loodrecht op de quadcopter). Het tweede resultaat is de gewenste hoek $\mathbf{r_a}$ (de underscore a komt van het angle, het engelse woordt voor hoek). De vector $\mathbf{r_a}$ bestaat uit de gewenste *roll*, *pitch* en *yaw*. De *yaw* zullen we echter constant op 0 houden.

Deze referentie hoek $\mathbf{r_a}$ is de referentie van een tweede set PID controllers, maar nu niet voor de positie, maar voor de orientatie van de quadcopter.

Misschien is het je opgevallen, dat we nog een extra referentie-ingang hebben, namelijk $\mathbf{r_{a,o}}$. Deze extra ingang is niet nodig voor de autopilot, maar een handigheidje bij het tunen van de PID regelaar voor de roll en de pitch. Met behulp van $\mathbf{r_{a,o}}$ kunnen we zelf een setpoint waarde opgeven zodat we het effect van de roll en de pitch regelaars kunnen zien. Bijvoorbeeld als we de stapresponsie van de roll-controller willen zien, kiezen we

$$\mathbf{r_{a,o}} = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$$

En vervolgens kun je $\mathbf{r_{a,o}}$ bijvoorbeeld weer gelijk aan de 0-vector maken.

De uitgang van de roll, pitch en yaw-controllers is het moment $\mathbf{M}$ in de roll, pitch en yaw-richting. Het moment in de roll richting zorgt voor een hoekversnelling van de roll, ofwel rond de x-as van de quadcopter. Het moment in de roll richting kunnen we uitoefenen door middel van een koppel dat gegenereerd wordt door de 2 motoren op de y-as van de quadcopter. Op dezelfde manier kan het moment in de pitch richting worden uitgeoefend door middel van een koppel dat gegenereerd wordt door de 2 motoren op de x-as van de quadcopter. Het moment in de yaw-richting zal niet met behulp van de 4 motoren worden uitgeoefend, aangezien we uitgaan van de versimpelde situatie dat de motoren krachtactuatoren zijn. De yaw zal direct worden doorgegeven aan de **pybullet** module om een moment uit te oefenen op de quadcopter in de yaw-richting, ofwel langs de z-as van de quadcopter (NB Dit wordt niet weergegeven in het blokschema van Figuur 2!).

Het omrekenen van de thrust $\mathbf{F_t}$ en de momenten in de roll en de pitch richting (eerste twee elementen in $\mathbf{M}$) wordt gedaan in het blokje math_2. Dit omrekenen gaat vrij eenvoudig, en komt in wezen neer op het omrekenen van de momenten in koppels en het sommeren van de krachten, en is ook reeds geïmplementeerd in de voorbeeldcode. Het resultaat is de vector $\mathbf{F}$ die bestaat uit de 4 krachten voor elke actuator één.

De actuatorkrachten $\mathbf{F}$ worden doorgegeven aan de `pybullet` module die de simulatie van de quadcopter uitvoert. Het resultaat is een positie-vector $\mathbf{y_p}$ en een oriëntatie-vector $\mathbf{y_a}$, met daarin de roll, pitch en yaw.

De `pybullet` module geeft echter niet direct de oriëntatie in termen van roll, pitch en yaw, maar in termen van een *quaternion*. Quaternions zijn een uitbreiding op complexe getallen, en door de Ierse wiskunde William Rowan Hamilton geïntroduceerd, zie bijvoorbeeld de Wiki `https://nl.wikipedia.org/wiki/Quaternion`. Lange tijd waren quaternions alleen interessant voor wiskundigen, maar tegenwoordig worden quaternions veel gebruikt in robotica en computergraphics, zelfs de meeste grafische kaarten kunnen tegenwoordig rekenen met quaternions. Voor nu is het voldoende te bedenken dat een quaternion bestaat uit vier reële getallen die samen een oriëntatie definiëren en eenvoudig omgerekend kunnen worden naar drie Euler hoeken, ofwel roll, pitch en yaw. Dat laatste wordt gedaan in de `pybullet`-functie `getEulerFromQuaternion`, daar hoeven we ons dus niet meer druk over te maken.

4 Uitleg van `quadcopter_sim.py`

(Todo) Zie de documentatie in het voorbeeld `quadcopter_sim.py` en het overzicht van de variabelen in Tabel 3. Verder bedenk dat de quadcopter controller een object is, genaamd `qcc`. Objecten bevatten doorgaans data, dat kunnen ook andere objecten zijn, en methodes (of functies). Data kunnen we zowel lezen als schrijven. In het volgende voorbeeld bekijken we eerst de waarde van de referentie-positie `ref_pos`, daarna veranderen we deze in de vector $[0, 0, 2]$, bekijken deze opnieuw en bekijken vervolgens ook de waarde van `error_pos`

```
>>> qcc.ref_pos
>>> qcc.ref_pos = np.array([0,0,2])
>>> qcc.ref_pos
>>> qcc.error_pos
```

Ook de gains van de PID controllers kunnen we op deze manier opvragen en aanpassen. Bijvoorbeeld voor de z-controller:

Tabel 3: Overzicht van de symbolen uit het blokschema van Figuur 2 met de bijbehorende variabele in de Python simulatie en hun omschrijving.

Symbool	Variabele	Omschrijving
$\mathbf{r_p}$	`ref_pos`	Referentie positie (x/y/z) op te geven door gebruiker (3D vector)
$\mathbf{r_a}$	`ref_rpy`	Referentie hoek (roll/pitch/yaw) berekend op basis van $\mathbf{F_p}$ (3D vector)
$\mathbf{r_{a,o}}$	`ref_rpy_offset`	Referentie hoek (roll/pitch/yaw) op te geven door gebruiker (3D vector) (NB: wordt alleen gebruikt ten behoeve van instellen $\mathbf{PID}_a$ (roll/pitch/yaw) regelaars!)
$\mathbf{y_p}$	`pos_meas`	Positie (x/y/z) van de quadcopter (3D vector)
$\mathbf{y_a}$	`rpy_meas`	Hoek (roll/pitch/yaw) van de quadcopter (3D vector)
$\mathbf{e_p}$	`error_pos`	Resterende fout in positie (x/y/z) (3D vector)
$\mathbf{e_a}$	`error_rpy`	Resterende fout in hoek (roll/pitch/yaw) (3D vector)
$\mathbf{PID}_p$	`x_ctrl`	3 PID controllers voor positie regeling (x/y/z)
	`y_ctrl`	
	`z_ctrl`	
$\mathbf{PID}_a$	`roll_ctrl`	3 PID controllers voor hoek (roll/pitch/yaw) regeling
	`pitch_ctrl`	
	`yaw_ctrl`	
$\mathbf{F_p}$	`force_pos`	Kracht om naar gewenste positie $\mathbf{r_p}$ te sturen (3D vector)
$\mathbf{F_t}$	`thrust`	Hoeveelheid kracht loodrecht op vlak van de quadcopter, is gelijk aan som van de 4 motorkrachten (3D vector)
$\mathbf{F}$	`force_act1..4`	Vector met de 4 actuatorkrachten (4D vector)
$\mathbf{M}$	`moments`	Momenten om naar gewenste hoek $\mathbf{r_a} + \mathbf{r_{a,o}}$ te sturen (3D vector)
math_1		formules om de thrust $\mathbf{F_t}$ en de benodigde hoek $\mathbf{r_a}$ te berekenen
math_2		formules om op basis van de thrust $\mathbf{F_t}$ en de momenten $\mathbf{M}$ de 4 actuatorkrachten $\mathbf{F}$ te berekenen
quadcopter		simulatie van de quadcopter met behulp van de `pybullet` module

```
>>> qcc.z_ctrl.Kp
>>> qcc.z_ctrl.Kp = 5
>>> qcc.z_ctrl.Kp
>>> qcc.z_ctrl.Ki
>>> qcc.z_ctrl.Ki = 0
>>> qcc.z_ctrl.Kd
>>> qcc.z_ctrl.Kd = 1
```

Voor de andere controllers gaat dit op dezelfde manier.

Als je onbekend bent met object geörienteerd programmeren in Python, dan is het aan te raden een korte introductie zoals `https://nl.wikibooks.org/wiki/Programmeren_in_Python/Object-georiënteerd_programmeren` door te nemen. Ook op youtube staan nuttige filmpjes met uitleg over object georienteerd programmeren in Python, bijv. die van Derek Banas `https://www.youtube.com/watch?v=1AGyBuVCTeE`.

5 Opdracht tutorial UAVREG

5.1 Stappen

5.1.1 Stap 0: Bestudeer deze handout en voorbeeld code

5.1.2 Stap 1: PID-regelaar implementatie

Ga na hoe je een PID controller kunt implementeren en pas de code in `quadcopter_sim.py` aan.

5.1.3 Stap 2: Hoogte (z) regeling

Ga na hoe je de zwaartekracht kan compenseren en tune de PID controller voor de z-richting. Tip: begin met K_p, vervolgens K_d en evt. K_i.

5.1.4 Stap 3: Roll/pitch regeling (yaw is al gedaan)

Tune de Roll en de Pitch controllers. Vanwege symmetrie in de quadcopter kun je voor de pitch regeling dezelfde controller parameters gebruiken als de roll-richting. Hint: Gebruik de offset op de referentie `qcc.ref_rpy_offset` en zet gravity volledig uit (`p.setGravity(0,0,0)`) tijdens het tunen van deze regelaars zodat de quadcopter niet 'wegloopt' als deze wordt geroteerd. Zorg ervoor dat de roll en de pitch regelingen 'voldoende snel' zijn ten opzichte van de positie regeling. De positieregeling gaat er namelijk vanuit dat de stand van de quadcopter namelijk correct wordt aangenomen.

5.1.5 Stap 4: Horizontale (x/y) regeling

Ook hier geldt weer, begin met K_p, dan K_d en evt. nog K_i. Vanwege symmetrie kun je voor de y-regelaar dezelfde waarden kiezen als in de x-richting.

5.2 Afteken opdracht: Baan-regeling

Bedenk een baan bestaande uit ten minste 10 punten in de 3D ruimte (mag ook meer, ook veel meer). Leg deze baan af met de quadcopter autopilot die je zojuist hebt geprogrammeerd.

Denk na over de snelheid waarmee de baan wordt doorlopen. Demonstreer je baanregeling en zorg dat je je code en de keuze van de PID parameters kunt uitleggen.