KU Leuven
Departement Computerwetenschappen

P&O: Computerwetenschappen

Eindverslag

Versie 2: Locomotor Zeppelin

Team:
Indigo

Wander Bavin
Vince Goossens
Dimitri Jonckers
Sunil Tandan
Wout Vekemans

Academiejaar 2013 – 2014

Samenvatting

TODO!!!!!! uitbreiden
Dit document bespreekt de autonome zeppelin van Team Indigo, gemaakt in het kader van P&O Computerwetenschappen. Deze zeppelin, gebouwd met 2 ballonnen en 3 motoren en aangestuurd door een Raspberry Pi, kan commando's uitvoeren die op QR-cedes staan.
Dit verslag documenteert onze bevindingen en vooruitgang. Meer concreet beschrijven we de opbouw van onze zeppelin en de structuur van de software.

Inhoudsopgave

1 Inleiding **2**

2 Beschrijving materiaal en bouw zeppelin **2**

3 Testen **3**
3.1 Afstandssensor 3
3.2 Camera 5
3.3 Motoren 5

4 Algoritmes **7**
4.1 Verticale bewegingen 7
4.2 Andere bewegingen 7

5 Software **7**

6 GUI **9**

7 Besluit **9**

A Beschrijving van het proces **11**

B Beschrijving van de werkverdeling **12**

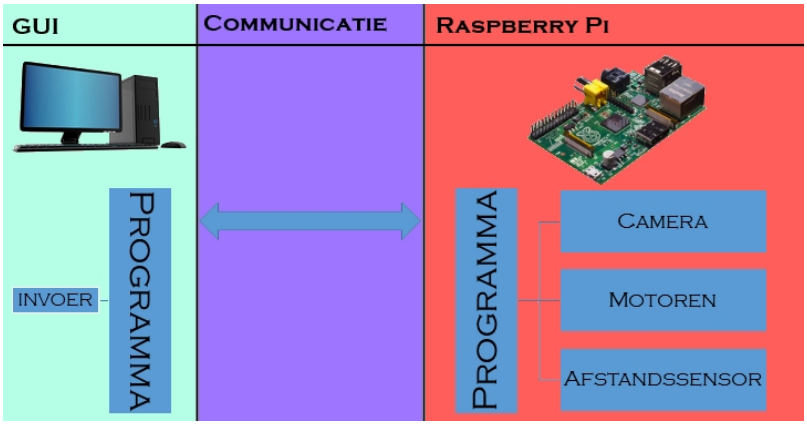
C Kritische analyse **12**

1 Inleiding

Het doel van deze opdracht is een autonome zeppelin te ontwikkelen. De zeppelin wordt aangestuurd door een Raspberry Pi. Via QR-codes kunnen instructies gegeven worden, zoals "stijg 50 cm" of "draai 45 graden". De gebruiker kan via een GUI op een client-pc zelf de pijltjestoetsen gebruiken om de zeppelin aan te sturen of gegevens bekijken over de toestand van de zeppelin (bijvoorbeeld de huidige hoogte).

Fysisch ontwerp
De zeppelin bestaat uit een houten frame waaraan 2 heliumballonnen (⌀90 cm) vastgemaakt zijn. Aan het frame is een afstandssensor gekoppeld die naar onderen is gericht. Verder is er een camera naar beneden gericht. Zowel de camera als de afstandssensor zijn verbonden met de Raspberry Pi die in het frame zit ingebed. Het geheel bevat drie propellers: twee voor horizontale bewegingen en één voor verticale bewegingen.

Software ontwerp
Het grootste deel van de opdracht is het schrijven van de software in een taal naar keuze. Wij hebben gekozen voor Java. De software bestaat uit 3 grote delen: de GUI, de communicatie tussen GUI en Raspberry Pi en de interne programmatie op de Pi (Zie figuur 1).
De communicatie tussen GUI en de Raspberry Pi gebeurt via sockets. Hierbij worden objecten uitgewisseld tussen de client en de server (de Pi), die commando's doorgeven aan de zeppelin of de status doorgeven aan de gebruiker. De interne programmatie op de Pi moet op basis van gegevens van zijn sensoren en instructies van QR-codes (en commando's van de gebruiker) de motoren aansturen.



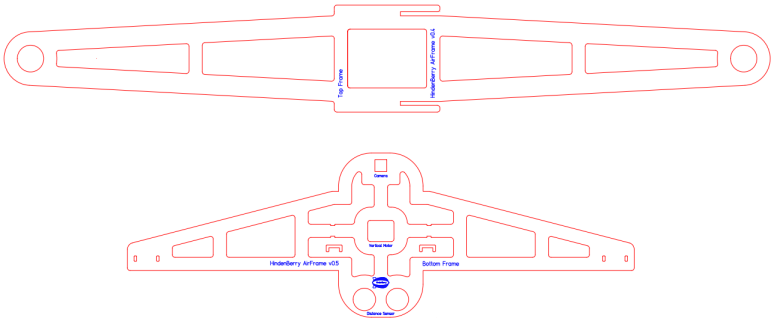
Figuur 1: Architectuur

2 Beschrijving materiaal en bouw zeppelin

De zeppelin bestaat uit een frame waaraan alle onderdelen zijn vastgemaakt. Hierop worden onder andere 3 propellers bevestigd. Twee hiervan dienen om naar links en rechts te draaien. Om vooruit te bewegen worden deze samen geactiveerd met dezelfde kracht. Deze propellers bevinden zich aan de uiteindes van de vleugels. De derde propeller, om de zeppelin te laten stijgen, is naar beneden gericht. De propellers kunnen op volle kracht worden aangestuurd (in 2 richtingen) of door middel van pwm[1]. Met deze techniek is het mogelijk om naast de richting ook de kracht van de motor in te stellen.

TODO figuren: even groot? (evt met height), caption bij upper?

[1] en.wikipedia.org/wiki/Pulse-width_modulation



Figuur 2: Blueprint van het frame

Om het geheel in de lucht te houden, zitten er 2 heliumballonnen (⌀90 cm) vast aan de bovenkant van het frame.

De zeppelin wordt aangestuurd door een Raspberry Pi model A. Deze heeft volgende specificaties:

- *Processor:* 700MHz ARM
- *Geheugen:* 256MB
- *Poorten:* 1 USB 2.0, HDMI, audio out, RCA video
- *Voeding:* Micro USB
- GPIO-pinnen om de hardware aan te sturen

In de Raspberry Pi zit een SD-kaart van 4 GB. Hierop staat Raspbian, het standaard besturingssysteem van de Raspberry Pi.

Verder zijn er nog 2 devices waarvan de zeppelin gebruik maakt:

- De camera laat toe foto's te nemen met een maximum resolutie van 5 MP. Hiermee kunnen we onder andere beelden maken van QR-codes. Daarnaast kan de camera video's maken met resoluties tot 1080p.
- De afstandssensor kan worden gebruikt om met ultrasone trillingen de afstand te meten tussen de zeppelin en de grond of muur. Het bereik gaat van 2-400 cm.

TODO zeppelin figuur
Om het geheel te monteren, hebben we gebruik gemaakt van plakband en zipties (bundelbandjes).

3 Testen

Om er zeker van te kunnen zijn dat het aansturen van de zeppelin correct gebeurt, is het nodig dat de componenten getest worden. In de volgende sectie volgt hierover meer informatie.

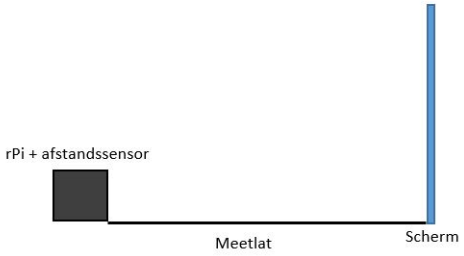
3.1 Afstandssensor

In deze sectie gaan we dieper in op de testen die we gedaan hebben naar de nauwkeurigheid en snelheid van de afstandssensor.

Opstelling
Benodigdheden:

- RaspberryPi met afstandssensor
- houten scherm
- meetlat

Deze componenten worden in deze opstelling geplaatst : zie figuur 3



Figuur 3: testopsteling

Verloop
De test bestaat uit het meten van 11 verschillende afstanden (10, 20, 30, 60, 90, 100, 110, 120, 130, 140 en 150 cm). Deze zijn zo gekozen om informatie te hebben over voldoende punten binnen het vlieghoogtebereik. Deze afstanden worden elk 500 keer gemeten, om de 60 ms. Deze waarden zijn zo gekozen, om toch voldoende nauwkeurigheid te hebben, veronderstellende dat er uitschieters zouden zijn.

Gegevens en verwerking
Het uitvoeren van de testen nam per afstand gemiddeld 32 seconden in beslag. Omdat dit tijdens het opereren van de zeppelin veel te lang is, hebben we besloten om de tijd tussen de metingen te verkorten. Zo kunnen de metingen sneller gebeuren. Hiervoor hebben we een extra test uitgevoerd op een hoogte van 170cm. Het blijkt dat we het interval tussen metingen kunnen terugbrengen naar 20 ms. Bij lagere waardes gaan signalen door elkaar lopen en zo de meting verstoren.

We hebben d.m.v. rolling median de mediaan berekend van 10 en 20 opeenvolgende gegevens, uit de set van 500 waardes per hoogte. Minder dan 10 is niet nauwkeurig genoeg, en voor meer dan 20 samples is de registratietijd te hoog. Uit deze gegevens hebben we om een aantal redenen besloten om een window van 10 samples te nemen. Ten eerste is er de nauwkeurigheid als de zeppelin stijgt/daalt. Het hoogteverschil tussen sample 1 en 10 is immers kleiner dan dat tussen 1 en 20. Ten tweede is er de snelheid. Het spreekt voor zich dat het registreren van 10 samples minder lang duurt dan 20 samples. Tenslotte is er ook het minieme verschil tussen de gemiddelde mediaan bij 10 en 20 metingen. Minder metingen zorgen dus voor een even grote nauwkeurigheid.
Een nadeel is wel dat de standaardafwijking van de medianen bij een breedte van 10 samples gemiddeld 0.2 cm groter is dan bij 20 samples. Dit weegt echter niet op tegen de winst aan snelheid en nauwkeurigheid.

Voor de meetgegevens: zie onderstaande tabel.

Afstand	10	20	30	60	90	100	110	120	130	140	150
μ (med.) (10)	10.8	20.5	30.8	58.0	88.3	98.7	108.5	117.3	127.74	137.0	147.10
μ (med.) (20)	10.8	20.7	30.7	58.0	88.3	98.7	108.5	117.3	127.7	137.0	147.1
σ (med.) (10)	0.21	0.52	0.82	0.55	0.70	0.72	0.68	0.94	0.58	0.60	0.69
σ (med.) (20)	0.09	0.33	0.71	0.41	0.59	0.56	0.47	0.54	0.43	0.37	0.49

Conclusie
De afstandsmeter meet het meest nauwkeurig bij kleine afstanden, bij grotere afstanden (meer dan 60 cm) liggen de meetresultaten gemiddeld iets onder de werkelijke afstand. Relatief gezien blijft de fout nog klein en kunnen we deze afwijking in het vervolg incalculeren bij het bepalen van de meetwaarden. De afstandssensor geeft telkens de mediaan van 10 gegevens door volgens het principe van de "rolling median"-techniek. Tien gegevens zorgen voor voldoende nauwkeurigheid, maar hebben ook het aspect dat de mediaanberekeningen snel genoeg gebeuren. Tevens is dit bij een hoogteverandering accurater. Deze methode boet wel in op de standaardafwijking. De gegevens wijken dus meer uit, maar dit weegt niet op tegen de voorgenoemde pluspunten. Om de afstandsregistratie nog sneller te laten verlopen, meet de afstandsmeter om de 20 ms.

3.2 Camera

Deze test is nog niet uitgevoerd.
TODO Het testen van de camera lijkt op het eerste zicht eenvoudig, maar er komt veel meer bij kijken dan enkel foto's nemen. Er moet rekening gehouden worden met lichtintensiteit, responstijd van de camera, positie van de QR-code op de foto, ...

De lichtintensiteit wordt getest in een kamer met controleerbare lichtbron (d.m.v. lichtschakelaar, verduistering). Er wordt getest hoe licht het moet zijn in de kamer opdat de QR-code in de foto nog herkend wordt door het programma.

We hebben al kleine testen uitgevoerd waarin de QR-code lichtjes gedraaid was (invalshoek), en er was geen probleem om ze te lezen. Omdat de QR-code altijd recht onder de zeppelin zal staan, gaan we dit niet verder testen.

De responstijd (tijd tot herkennen QR-code) wordt getest door in verschillende omstandigheden een code te fotograferen. Een simpel printcommando in Java vertelt ons hoe lang het verwerken duurt. Hiervoor wordt de interne klok van de computer gebruikt. Om een globaal beeld te krijgen van de responstijd wordt een gemiddelde berekend van deze gegevens. We verwachten dat op grotere hoogtes (grotere afstand tot de code) we een afbeelding moeten nemen in hogere resolutie, waardoor de verwerkingstijd langer zal zijn.

Om te bepalen tot op welke afstand de camera foto's kan maken die herkend kunnen worden, is er een eenvoudige test: de camera fotografeert een QR-code vanop verschillende afstanden. De positie van de QR-code in de foto is een cruciaal element. De codes bevatten commando's die relatief ten opzichte van de huidige positie moeten worden uitgevoerd. Kleine afwijkingen van de positie van de code tot het midden van de foto zijn onvermijdelijk en zullen zich voortplanten tijdens het verderzetten van het parcours. Dit heeft echter meer betrekking tot het testen en calibreren van de motoren (zie verder).

3.3 Motoren

Deze test is nog niet uitgevoerd.

We zijn bij onze eerste experimenten al enkele zaken te weten gekomen. Zo blijkt dat een minimale pwm-waarde van ongeveer 740 nodig is (ter info: maximumwaarde is 1024) om de propeller in beweging te krijgen. Daarnaast hebben we al gezien dat wanneer je de motoren voor enkele seconden laat draaien, de zeppelin nog een vrij lange tijd gaat uitbollen.

Hieronder is meer informatie te vinden over hoe we concreter en grondiger gaan testen voor bepaalde bewegingen.

Zijwaartse bewegingen
Om de zijwaartse bewegingen te testen, gaan we kijken hoe lang we de motor moeten aanzetten om een hoek van 10° , 20°, 30°, ... uit te voeren. We zullen deze testen uitvoeren op hoeken naar links. Hiervoor zetten we de linkermotor op volle kracht achteruit en de rechtermotor op volle kracht vooruit.

Om dit te testen, gaan we de zeppelin moeten positioneren boven een soort 'roos': een cirkel waarop de graden staan aangeduid. Aan de hand van foto's van de naar onder gerichte camera, kunnen we dan manueel aflezen hoe ver de zeppelin gedraaid is.

Indien we deze gegevens voor hoeken van 10 tot 180 graden hebben, kunnen we deze plotten. Hier verwachten we een lineair patroon te zien vanaf een bepaalde hoek, namelijk bij de hoek waarbij de maximale kracht van de motor bereikt is. Het lineair patroon gaat doorbroken worden wanneer de gevraagde hoek bijna is bereikt.

Wanneer we deze functie gevonden hebben, zijn enkel de volgende gegevens nodig:

- data voor voldoende hoeken kleiner dan de kritieke hoek waarbij het maximumvermogen van de motor wordt bereikt
- de lineaire functie
- duur van uitbollen

Voor kleine hoeken kunnen we dan een soort van tabel raadplegen, voor hoeken groter dan de kritieke hoek gebruiken we de lineaire functie.

Als laatste moeten we controleren of deze gegevens hetzelfde zijn voor bewegingen naar de rechterkant. Dit doen we door dezelfde gegevens voor een aantal hoeken toe te passen.

Voorwaartse bewegingen
Om voorwaartse bewegingen uit te voeren, zetten we zowel linker als rechter motor aan. Uit de testen die we gedaan hebben bij zijwaartse bewegingen, weten we of de 2 motoren hetzelfde afgesteld zijn. Indien ze niet perfect afgesteld zijn, zullen we moeten testen hoe vaak we tijdelijk de krachtigste motor moeten uitschakelen om een zo perfect mogelijke rechte baan te bekomen.

Het testen van de voorwaartse beweging gaat heel gelijkaardig zijn aan het testen van de zijwaartse bewegingen. De zeppelin beweegt boven een liniaal. Aan de hand van foto's van de camera kunnen we zien hoever hij van de rechte lijn afgeweken is en hoever hij geraakt is. We gaan opnieuw kunnen afleiden na hoeveel tijd de zeppelin zijn maximum vermogen bereikt. Zo gaan we opnieuw een formule opstellen die we dan gebruiken om te bepalen hoe lang we de motoren moeten laten draaien om een bepaalde afstand af te leggen.

Verticale bewegingen
We gaan allereerst op zoek naar de juiste pwm waarde waarbij de zeppelin op dezelfde hoogte blijft zweven ('zweef-pwm'). Dit gebeurt wanneer de stuwkracht gelijk is aan de gravitatiekracht. Vervolgens gaan we uitzoeken hoeveel centimeter de zeppelin nog extra stijgt wanneer we overgaan van maximale pwm naar zweef-pwm.

4 Algoritmes

4.1 Verticale bewegingen

TODO Om naar een bepaalde hoogte te stijgen, maken we gebruik van een PID-algoritme[2]. Hierbij gaan we op basis van de huidige fout in hoogte, bepalen of de zeppelin moet stijgen of dalen en met welke kracht. Daarnaast wordt rekening gehouden met de afgeleide, om toekomstige veranderingen te voorspellen. Tenslotte is er de integraal, die fouten uit het verleden voorstelt. Op basis hiervan wordt een pwm-waarde voor de motor gegeven. Dit algoritme moet nog verder getuned worden. De output wordt berekend op basis van deze formule:

```
output = Kp*error + Ki*integral + Kd*derivative
```

Hierin zijn Kp, Ki en Kd constanten die we hebben moeten bepalen. Eerst hadden we enkel rekening gehouden met de huidige error (Ki = Kd = 0), maar dit bleek er voor te zorgen dat de zeppelin veel te snel naar een bepaalde hoogte gaat en er dan ver boven of onder gaat. We hebben dit opgelost door Kd te verhogen. Door deze groot te maken, gaat de zeppelin veel rustiger naar de opgegeven hoogte en gaat hij er niet voorbij.

We hebben een HeightController die dit algoritme implementeert, en die er voor gaat zorgen dat de zeppelin zijn hoogte behoudt of naar een gevraagde hoogte gaat.

4.2 Andere bewegingen

TODO!! Hiervoor kunnen we geen gebruik maken van data van een sensor, dus gaan we een functie moeten opstellen om een verband te krijgen tussen afstand en tijd dat een motor moet aanstaan. Dit hadden we reeds vermeld in het gedeelte over het testen van de motoren.

5 Software

Zoals reeds vermeld, is de software volledig in Java geschreven. Deze keuze hebben we gemaakt omdat we al veel ervaring hebben met het schrijven van Java-programma's. Python was een andere mogelijke keuze. Er is veel voorbeeldcode in Python te vinden voor de Raspberry Pi, maar we hebben voor de functies die wij nodig hebben voldoende kennis en referenties in Java gevonden. Het leren van een extra taal zou erg tijdrovend zijn in verhouding met de voordelen die het brengt.

Om de software te schrijven, maken we gebruik van de Eclipse IDE. Daarnaast gebruiken we de Netbeans IDE. Deze is door de visuele interface veel gebruiksvriendelijker om de GUI te ontwerpen. Hierdoor moeten we geen tijd besteden aan het handmatig schrijven van alle code voor de lay-out. Voor het aansturen van de GPIO-pinnen gebruiken we Pi4J[3].

Op een client-pc kan de GUI worden gestart. Hier kan de gebruiker informatie aflezen over de toestand van de zeppelin en commando's geven met de pijltjestoetsen. Meer informatie hierover is terug te vinden in de volgende sectie.

De communicatie tussen de GUI (client) en de Raspberry Pi (server) gebeurt via sockets. Sockets zijn een eenvoudige en universele manier om data over een netwerk te sturen. Ze zijn te vergelijken met een deur waarlangs objecten van klassen uitgewisseld worden. Deze stellen bijvoorbeeld commando's van de gebruiker voor of de status van de zeppelin die aan de GUI wordt doorgegeven. Concreet opent de server een socket op poort 6789 (de eerste 1024 poorten zijn gereserveerd voor services zoals http en ftp, we hebben een willekeurige poort hierachter gekozen), waarop de client zal verbinden. We hebben zelf een klasse geïmplementeerd die Transfers voorstelt die worden doorgestuurd, en we regelen de coördinatie zelf via klasses voor communicatie

[2] en.wikipedia.org/wiki/PID_controller
[3] www.pi4j.com

op zowel client als server. Andere klasses maken hiervan gebruik om Transfers aan te maken met een bepaalde functie (bijvoorbeeld doorsturen van de status van een propeller) en deze te versturen.

Voor de communicatie maken we gebruik van een virtueel netwerk dat wordt opgezet vanaf een laptop. Het bleek niet eenvoudig om gebruik te maken van Eduroam voor de verbinding, omdat dat een groot deel van het netwerkverkeer blokkeert. Als alternatief hadden we gebruik kunnen maken van een router om een netwerk op te zetten, maar dan zouden we ofwel zelf voor een router moeten zorgen, ofwel afhankelijk zijn van een andere groep in dezelfde klas die de router in hun bezit hebben.

Om meerdere taken tegelijk te kunnen uitvoeren, maken we gebruik van threads. Bepaalde functies die continu moeten gebeuren (met telkens een wachttijd tussen elke uitvoering) worden zo tegelijk uitgevoerd (technisch gezien niet tegelijk, maar er wordt gewisseld tussen deze threads). Threads worden onder andere gebruikt voor de HeightController (constant bijsturen om te zorgen dat de juiste hoogte bereikt wordt of behouden blijft) en de ReceivedHandler (die binnenkomende Transfers van de socket opvangt en verwerkt).

Behalve de GUI draait alle software op de Raspberry Pi, omdat die autonoom moet kunnen werken. Hier worden beslissingen genomen op basis van QR-codes, die uit afbeeldingen komen die de camera op geregelde tijdstippen neemt. Het verwerken van de beelden gebeurt op de Raspberry Pi. We kunnen kiezen om de afbeeldingen volledig door te sturen naar de GUI, of enkel de geïnterpreteerde tekst. Logisch gezien is het weergeven van de status van de zeppelin geen taak van de Pi zelf. Daarom draait de GUI op de client. Daarnaast wordt door deze keuze de CPU van de Pi minder zwaar belast.

De hoogte wordt om de 20 ms opgemeten. Om een zo accuraat mogelijke waarde te krijgen, wordt de mediaan genomen van 10 opeenvolgende meetresultaten. De hoogte wordt elke seconde doorgestuurd naar de GUI.

Op de Raspberry Pi worden objecten aangemaakt die commando's voorstellen. Deze commando's zitten in de package 'command', en zijn allemaal subklasses van de klasse Command. De zeppelin heeft klasses die verantwoordelijk zijn voor de aanmaak van deze objecten op basis van QR-codes (QRParser) en het bijhouden van een rij van uit te voeren commando's (CommandController). De klasse CommandExecutioner zorgt voor de praktische implementatie van algoritmes van de commando's.

Om de afbeelding te doorzoeken op QR-codes, maken we gebruiken van ZXing[4] (Zebra Crossing). Dit is een bekende library voor het inlezen van QR-codes die oorspronkelijk in Java geschreven is.

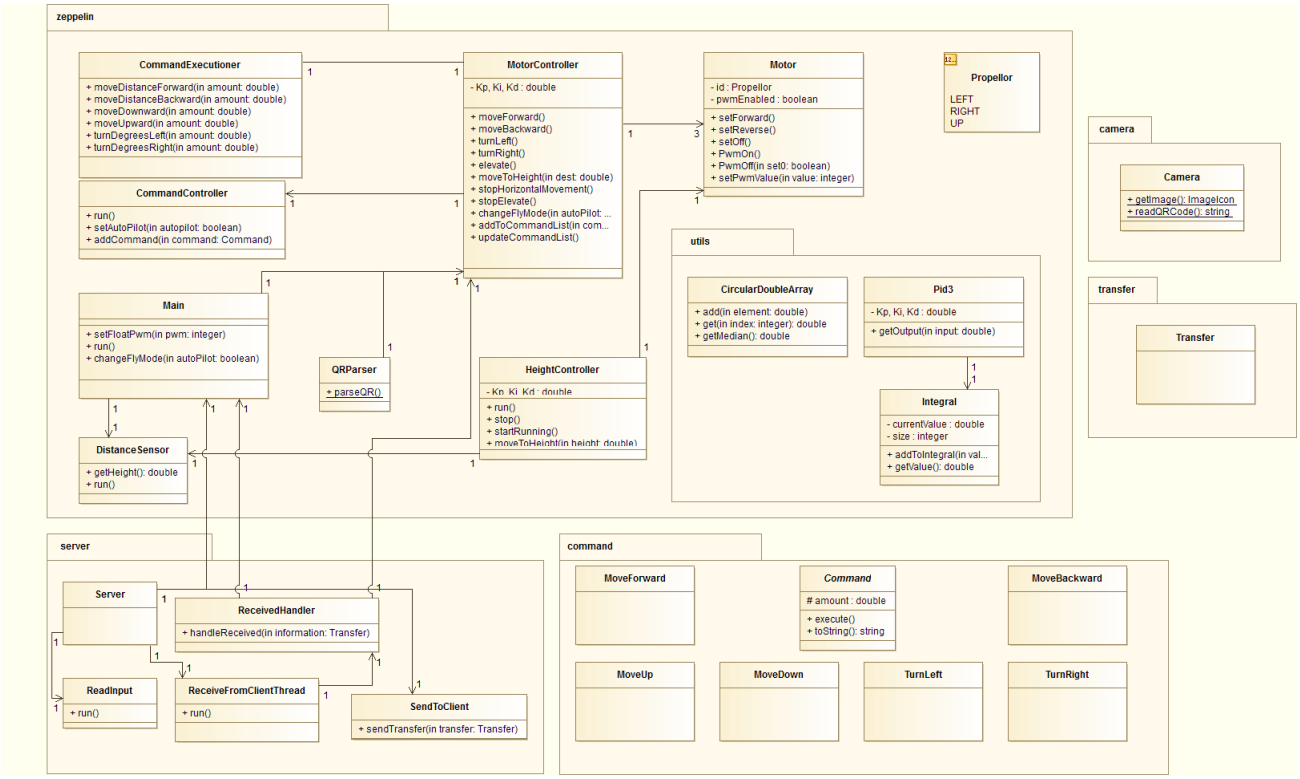
6 GUI

De User Interface stelt de gebruiker in staat om vanaf een client-pc verbinding te maken met de zeppelin. De GUI geeft informatie weer over de status, zoals de huidige hoogte en commando's die worden uitgevoerd. Daarnaast kan de gebruiker de zeppelin aansturen via de pijltjestoetsen.

De eerste tab ('main') toont de hoogte en toestand van de propellers. Er is ruimte voorzien om foto's (van de camera) te tonen en om de commando's weer te geven die in de wachtrij van de zeppelin zitten. Verder zijn hier toetsen om de motoren aan te sturen en om de zeppelin naar een bepaalde hoogte te laten stijgen.

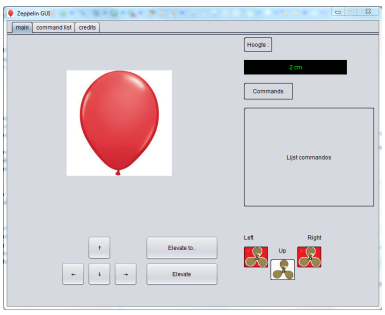
De tweede tab geeft een uitgebreider overzicht van de informatie die wordt uitgewisseld tussen de GUI en zeppelin, met een indicatie van de tijd.

[4] http://code.google.com/p/zxing/



Figuur 4: Klassediagram van de interne programmatie

De zeppelin (server) staat slechts toe dat er één client verbindt, dus dat er één GUI wordt getoond. Het zou mogelijk zijn meerdere clients toegang te geven, maar om conflicten (met pijltjestoetsen) te vermijden en om bandbreedte te besparen (bij het doorsturen van afbeeldingen), ondersteunt de server maar 1 client.



Figuur 5: GUI

7 Besluit

TODO uitbreiden
We zijn er in geslaagd een zeppelin in elkaar te steken en de motoren aan te sturen. Via sockets wisselen we informatie uit tussen de zeppelin en de GUI en kunnen we de zeppelin via de pijltjestoetsen aansturen. Verder toont de zeppelin op de GUI zijn status en afbeeldingen aan de gebruiker.

A Beschrijving van het proces

- **Welke moeilijkheden heb je ondervonden tijdens de uitwerking?**
  Kleine startproblemen bestonden uit het niet direct aanwezig zijn van alle hardware, waardoor we tijdens de beginfase veronderstellingen moesten maken over hoe deze in de praktijk zou werken. Hierdoor moesten we soms op enkele beslissingen terugkomen. De praktijk voldeed dus (zoals we wel hadden verwacht) niet volledig aan de theorie. Dit was goed duidelijk bij het naar links of rechts draaien van de zeppelin. In theorie zou deze op eenzelfde plaats moeten blijven hangen boven een bepaald punt, maar dit was niet het geval.
  Een ander probleem dat we vooral in het begin hadden, was de communicatie tussen de zeppelin en de GUI. Dit is echter een probleem waarmee alle groepen te maken hadden. In feite waren we er vrij snel in geslaagd de communicatie te laten werken, maar in latere weken hebben we soms nog aanpassingen moeten maken omdat er problemen opdoken van zodra meer informatie werd verstuurd.
  Een probleem waar we verschillende uren mee bezig zijn geweest, is de interferentie tussen de neerwaartse motor en de afstandssensor. Als de motor wordt ingeschakeld, vertoonden de meetwaarden van de afstandssensor deviant gedrag. Om dit op te lossen, gaan we het frame aanpassen.
  Later hadden we een nieuw probleem waarbij de sensor een veel grotere hoogte gaf dan de werkelijke hoogte. Dit bleek veroorzaakt door de vele threads die op dat moment geïntroduceerd waren.

- **Welke lessen heb je getrokken uit de manier waarop je het project hebt aangepakt?**
  Dat veel testen in de praktijk noodzakelijk zijn. Dit is uiteraard te verwachten, maar is op bepaalde momenten overduidelijk. Goede voorbeelden hiervan zijn de problemen met de distancesensor (hierboven beschreven) en het regelen van de hoogte van de zeppelin. Aannames die je maakt zijn vaak niet waar (bijvoorbeeld naar boven vliegen brengt vaak een horizontale beweging met zich mee), en dit wordt pas duidelijk wanneer je de zeppelin echt aan het werk ziet met de code.
  Op het gebied van groepswerking was het naar de tussentijdse demo toe duidelijk dat er een vrij groot verschil zat op het aantal gepresteerde uren. We beseffen nu dat we van in het begin een goede taakverdeling moeten voorzien, en dat het belangrijk is deze elke week op te volgen en desnoods bij te sturen.
  Op het vlak van rapporteren hebben we vooral geleerd dat we veel moeten motiveren. We mogen ook niet verwachten dat de lezer even goed als wijzelf op de hoogte is van de details van het project.

- **Hoe verliep het werken in team? Op welke manier werd de teamcoördinatie en planning aangepakt?**
  Naar de tussentijdse demo toe was het duidelijk dat onze taakverdeling niet perfect was, waardoor de gewerkte uren vrij sterk uit elkaar lagen. Sommige taken waren snel afgewerkt, en er was soms niet genoeg werk voor iedereen waardoor er af en toe met 2-3 personen aan 1 computer gewerkt moest worden.
  De weken nadien hebben we ons best gedaan om ervoor te zorgen dat degenen die achter liepen in aantal uren, wat meer taken kregen.
  Wanneer we met de volledige groep samenkomen, is het voordeel dat iedereen zijn mening kan geven en kan bijdragen aan het project. Het nadeel is echter dat we nooit alle vijf gedurende de hele sessie aan maximale productiviteit kunnen werken.
  De communicatie tussen de verschillende groepsleden verliep vlot en bij problemen konden we terecht op de Facebook-groep die we voor dit project hebben aangemaakt.
  De eerste les hebben we een teamcoördinator en secretaris gekozen, maar deze keuzes waren eerder gemaakt omdat we toen nog geen idee hadden hoe het project ging verlopen. Uiteindelijk hebben we allemaal gedeeltelijk de rol van secretaris ingevuld: wie even geen taak had, kon aan de verslagen werken, die dan nadien door alle groepsleden nagelezen en aangepast werden.

Tijdens de eerste 2 lessen werd het al duidelijk welke grote delen er geprogrammeerd moesten worden. Hiervoor hebben we dan een aantal personen per deel gezet. Meer hierover is in het volgende deel terug te vinden.

B Beschrijving van de werkverdeling

De eerste lessen hebben we besloten dat er 3 grote delen waren die moesten afgewerkt worden: de GUI, de communicatie tussen GUI en zeppelin en de zeppelin zelf. Ook waren er de tussentijdse verslagen en het testen van de onderdelen.

- Wander Bavin: Heeft meegewerkt aan het onderdeel GUI. Dit deel vroeg vooral in het begin wat werk. Nadat de GUI afgewerkt was heeft hij verslagen gemaakt/nagelezen en uiteindelijk meegewerkt aan het motor-gedeelte. Indien er extra informatie op de GUI moest verschijnen heeft hij hiervoor gezorgd. Vervulde de rol van coördinator.
- Dimitri Jonckers: Heeft meegewerkt aan het onderdeel GUI. Heeft net zoals Wander de verslagen gemaakt en nagelezen, alsook meegewerkt aan de interne programmatie van de Raspberry Pi. Ook hij heeft de GUI aangepast wanneer nodig, en het UML-diagram opgesteld. Heeft thuis veel opzoekwerk gedaan/geprogrammeerd. Heeft de meeste weekly progress reports geschreven.
- Sunil Tandan: Heeft gewerkt aan het deel communicatie tussen GUI en zeppelin. Dit was een groot en belangrijk deel dat hij op zichzelf heeft gemaakt. Dit deel vroeg redelijk wat werk en moest elke keer wel een beetje aangepast worden (bijvoorbeeld voor het versturen van images). Heeft elk verslag nagelezen en delen over de communicatie toegevoegd. Nadien heeft hij het werken met de camera op zich genomen.
- Wout Vekemans: Heeft meegewerkt aan het deel motoren/zeppelin. Tijdens de beginfase voornamelijk verslagen gemaakt/nagelezen. Heeft samen met Vince de afstandssensor getest/geprogrammeerd en is daarna aan de motoren beginnen werken. Hij vervulde de rol van secretaris en stuurde de verslagen dus telkens door.
- Vince Goossens: Heeft meegewerkt aan het deel motoren/zeppelin. Tijdens de beginfase voornamelijk verslagen nagelezen. Heeft samen met Wout de afstandssensor getest/geprogrammeerd en is daarna aan de motoren beginnen werken/programmeren.

Hieronder is een tabel te vinden met de gewerkte uren binnen en buiten de sessies:

TODO aanvullen en updaten

Overzicht:	Dimitri Jonckers	Wander Bavin	Wout Vekemans	Sunil Tandan	Vince Goossens
30/09 - 06/10	5	5	5	7	5
07/10 - 13/10	15.5	12	6	11	5
14/10 - 20/10	11	10.5	6.75	9	6.25
21/10 - 27/10	9	5.5	6	9	5.5
28/11 - 03/11	12	5	5	5	5
04/11 - 10/11	15	16	10	16.5	10
11/11 - 17/11	17	14.5	8	18.5	10
18/11 - 24/11	9.5	15	9	9	9
18/11 - 24/11	11.5	5	11	13	11
Totaal					

C Kritische analyse

De groepssfeer was goed. De werkverdeling kon echter wel beter. In de periode voor de tussentijdse demo wou iedereen wel ergens aan werken, maar vaak was iemand anders hiermee al begonnen. We zouden beter (en vanaf het begin) moeten zorgen voor een goede taakverdeling en deze ook

opvolgen en aanpassen aan de hand van eventuele problemen en van verschillen in werkuren.
Een tweede punt van kritiek is dat in de loop van het project een gedeelte van de code van de zeppelin sterk uitgebreid is. Hierdoor is de code niet altijd even duidelijk meer (bijvoorbeeld de introductie van een aantal boolean-variabelen om de flow te regelen). Dit betekent dat er extra aandacht nodig is bij het aanpassen van de code. In principe is het mogelijk deze code op te kuisen, maar we hebben ons in code laatste weken voor de demo vooral moeten bezighouden met het toevoegen van nieuwe functionaliteit.
Over de kwaliteit van de code zelf zijn we wel tevreden. Omwille van problemen met het aantal threads zijn we verplicht geweest deze een stuk efficiënter te maken.
Verder is het fysiek ontwerp van onze zeppelin niet perfect. Idealiter hadden we een nieuw frame moeten laten maken, waarbij een deftige plaats voor de distancesensor voorzien is en dat gemaakt is uit een stevig materiaal zoals plexiglas. Omwille van het vele werk met de programmatie zijn we hier niet toe gekomen. We zijn tevreden over onze keuze voor Java en hebben eigenlijk op geen enkel moment tijdens het project gedacht dat we beter af zouden zijn door Python te gebruiken. Alle functionaliteit die we moesten implementeren, was zonder problemen mogelijk in Java.