Camera-stabilisatie-systeem

Afstudeerverslag over het optimalisatieproces van een bestaand camera-stabilisatie-systeem voor de Teensy 3.0.

Jeroen de Meij

Studentnummer: 11100877

Afstudeerverslag

De Haagse Hogeschool, opleiding Technische Informatica, afdeling Delft

Delft Dynamics B.V.

Bedrijfsmentor: Dhr. G. M. Voorsluijs

Examinatoren: Dhr. Tony Andrioli en mw. Cobie van der Hoek

Datum: 2 juni 2014

Versie beheer

Versie	Datum	Geschreven door	Beschrijving
0.1	20 feb 2014	Jeroen de Meij	Eerste opzet van het document, waarbij de structuur is opgezet.
0.2	21 feb 2014	Jeroen de Meij	Structuur is iets gewijzigd en de hoofdstukken 2, 3 en 4 zijn geschreven.
0.3	28 feb 2014	Jeroen de Meij	Paragraafnummering toegepast, typefouten verbeterd, begrippenlijst toegevoegd en hoofdstuk 2.2 en 5.1 tot en met 5.3 geschreven.
0.4	7 mrt 2014	Jeroen de Meij	Hoofdstuk 5.3 uitgebreid en hoofdstukken 5.4 en 6 geschreven.
0.5	14 mrt 2014	Jeroen de Meij	Hoofdstuk 7 tot en met 7.1.4 geschreven.
0.6	4 apr 2014	Jeroen de Meij	Hoofdstuk 7.1.5 tot en met 7.2.2 geschreven
0.7	25 apr 2014	Jeroen de Meij	Inhoudsopgave aangepast, hoofdstuktitels verbeterd en tekst verbeterd naar aanleiding van de eerste concept bespreking.
0.8	6 mei 2014	Jeroen de Meij	De rest van de tekst verbeterd naar aanleiding van de eerste concept bespreking en hoofdstukken 7.2.3 tot en met 8 en hoofdstuk 13 geschreven.
1.0	2 juni 2014	Jeroen de Meij	Aanpassingen gedaan aan de hand van het tussentijds assessment: kopteksten veranderd, hoofdstuk nummering aangepast, vormgeving inhoudsopgave aangepast, afbeeldingen toegevoegd en de laatste hoofdstukken geschreven.

Referaat

Jeroen de Meij, “Camera-stabilisatie-systeem”. Afstudeerverslag opleiding Technische Informatica, Haagse Hogeschool, 2014.

Voor het bedrijf Delft Dynamics is in de periode van 10 februari 2014 tot en met 6 juni 2014 gewerkt aan het optimaliseren van een bestaand camera-stabilisatie-systeem voor de Teensy 3.0. Dit systeem moet vervolgens onder een door Delft Dynamics gebouwde onbemande robothelikopter komen te hangen, waardoor de camera tijdens de vlucht een stabiel beeld kan opnemen. Dit systeem is ontwikkeld aan de hand van de watervalmethode. Tijdens het optimaliseren is gekeken naar de structuur van de code, die in het oude systeem erg rommelig was, en de documentatie, die in het oude systeem zeer gering was. Vervolgens zijn er een viertal uitbreidingen doorgevoerd: het bedienen van de hoek pitch hoek van de camera; wisselen tussen verschillende camera’s; de infrarood camera kalibreren en gemakkelijk een aantal systeem variabelen kunnen aanpassen. Ook is er een poging gedaan met het implementeren van een logging mogelijkheid die helaas is mislukt. Het systeem is uiteindelijk werkend onder een multicopter getest.

Descriptoren:

- afstudeerverslag
- camera-stabilisatie-systeem
- onbemande multicopter
- embedded systems
- Teensy

Voorwoord

Voor u ligt dan mijn afstudeerverslag. Het verslag wat mijn afstudeer periode beschrijft en daarmee vrijwel het einde van mijn opleiding Technische Informatica is. Ik heb het tijdens het uitvoeren van mijn afstudeer periode erg naar mijn zin gehad bij Delft Dynamics, ondanks dat ik tijdens het uitvoeren van mijn opdracht wel eens letterlijk met mijn handen in het haar heb gezeten. Desalniettemin ben ik tevreden met het opgeleverde product.

Wat ik ook erg leuk vind, is dat ik tijdens mijn verblijf bij Delft Dynamics ook het een en ander heb mogen leren over de luchtvaart, aangezien het bedrijf zich specialiseert in het ontwikkelen van onbemande robothelikopters. Dit was voor mij een hele nieuwe en onbekende wereld, maar ik vond het interessant om er tijdens met afstudeerperiode meer over te leren.

Daarnaast zijn er een aantal mensen die ik wil bedanken die mij tijdens mijn afstudeerperiode hebben geholpen bij de uitvoering van mijn project.

Als eerste wil ik de werknemers van Delft Dynamics en in het bijzonder Gerwin Voorsluijs bedanken, die tijdens deze periode als begeleider en opdrachtgever heeft gefunctioneerd. Ik bedank ze voor de open en vriendelijke sfeer waarbinnen ik mijn afstudeer opdracht heb mogen uitvoeren. Dit heeft ervoor gezorgd dat ik me snel thuis heb gevoeld tijdens het afstuderen. Ook bedank ik ze voor de begeleiding die ze hebben gegeven. In momenten waarop ik vast liep kon ik altijd bij ze terecht voor hulp.

Daarnaast wil ik ook mijn begeleiders Cobie van der Hoek en Tony Andrioli bedanken. Ik bedank ze voor de begeleiding en kritiek die ze hebben gegeven tijdens het afstuderen en dan vooral op de terugkoppel momenten. Dit heeft er mede voor gezorgd dat er voor u nu een mooi verslag op tafel ligt.

En tot slot bedank ik mijn vader voor het nalezen van mijn gehele verslag en documentatie om er voor te zorgen dat er zo min mogelijk taalfouten in zitten.

Jeroen de Meij, Ter Aar, 1 juni 2014

Samenvatting

Gedurende een periode van zeventien weken is er bij het bedrijf Delft Dynamics gewerkt aan het optimaliseren van een camera-stabilisatie-systeem voor de Teensy 3.0. Delft Dynamics is een bedrijf dat zich heeft gespecialiseerd in het ontwikkelen van onbemande robot helikopters die door het slim toevoegen van meet apparatuur metingen kunnen uitvoeren op plekken die voor mensen ontoegankelijk zijn of te gevaarlijk. Daarnaast is het ook mogelijk om een camera aan de multicopter te bevestigen, waardoor er van uit de lucht observaties kunnen worden gemaakt bij bijvoorbeeld files of branden.

Het systeem wat Delft Dynamics op dit moment gebruikt is niet uitbreidbaar, doordat de Arduino mini waar het systeem op draait aan het maximum van zijn kunnen zit. Na een analyse blijkt dat ook de code ervoor zorgt dat het systeem moeilijk uitbreidbaar is. De code zit warrig in elkaar en is slecht gedocumenteerd. Vervolgens is in deze fase een requirements document opgesteld naar aanleiding van deze probleem analyse.

Aan de hand van de watervalmethode wordt gewerkt om het systeem over te zetten naar en te optimaliseren voor de Teensy 3.0. Hiervoor is een basis ontwerp gemaakt, waarin een globaal klassendiagram is opgesteld voor de nieuwe structuur en een testplan is gemaakt, zodat het product uiteindelijk goed getest is.

Van dit basis ontwerp is een technisch ontwerp gemaakt, waarin alle functies uit het oude systeem in de nieuwe structuur zijn ingedeeld. Daarnaast is er ook onderzoek gedaan naar de real-time aspecten van het systeem, zodat gewaarborgd kan worden dat het systeem snel genoeg reageert op de bewegingen van de multicopter.

In een volgende fase is het systeem daadwerkelijk overgezet naar de nieuwe situatie. Tijdens dit proces gaf de opdrachtgever aan dat hij tijdens dit project ook graag uitbreidingen wil toevoegen aan het systeem, omdat de opdracht sneller ging dan verwacht. Hierdoor was het prettiger om een iteratieve methode te gebruiken in plaats van de watervalmethode. Daarom is de waterval methode iets aangepast om dit mogelijk te maken. Vervolgens zijn de uitbreidingen toegevoegd, waardoor het mogelijk is dat de gebruikers tijdens de vlucht de pitch hoek van de camera kunnen aanpassen, kunnen switchen tussen verschillende videostromen en de infrarood camera kunnen kalibreren. Ook is het mogelijk om van tevoren een aantal systeem variabelen aan te passen, zodat het programma per camera systeem kan worden geoptimaliseerd. Tot slot is in deze fase een poging gewaagd om een manier van logging mogelijk te maken, waarbij gegevens naar een MicroSD kaart worden gelogd. Dit is helaas niet gelukt, omdat er problemen waren met de gebruikte library.

Tot slot is het systeem grondig getest. Tijdens deze testfase was de opdrachtgever ook klaar met een nieuwe versie van de hardware van het systeem. Deze versie loste een aantal problemen uit de eerste versie op, waardoor aan het einde een mooi product tot stand is gekomen. Dit product is tijdens een kleine testvlucht getest en blijkt naar wens te functioneren.

DEEL 1: INLEIDENDE HOOFDSTUKKEN

DEEL 2: VERSLAG VAN HET ONTWIKKELPROCES

DEEL 3: AFRONDENDE HOOFDSTUKKEN

DEEL 1: Inleidende hoofdstukken

1. Inleiding

We komen het steeds meer tegen: elektrische apparaten die mensen helpen om bepaalde taken sneller en soms ook beter uit te voeren. Hierbij is te denken aan simpele apparaten als afwasmachines, waardoor de bezitters van deze machine niet zelf iedere dag met een afwasborstel in de hand hoeven te staan. Maar ook de wat grote apparaten in ziekenhuizen, die ervoor zorgen dat doctoren worden geholpen bij het behandelen van hun patiënten.

Afbeelding 1.1: Een multicopter van Delft Dynamics

Dit project staat echter in het teken van robothelikopters: kleine onbemande helikopters die door het slim samenvoegen van computer- en sensorapparatuur gebruikt kunnen worden als stabiel en gemakkelijk te besturen sensorplatform. Het bedrijf Delft Dynamics heeft zich gespecialiseerd in het ontwikkelen van dergelijke robothelikopters. In afbeelding 1.1 is een dergelijke multicopter te zien. Deze multicopters kunnen vervolgens worden ingezet om metingen uit te voeren op locaties die voor mensen te gevaarlijk zijn of moeilijk te bereiken, maar ze kunnen ook worden ingezet om filmbeelden te maken in dergelijke situaties, zoals bij branden. Op deze manier hoeven er geen mensen in gevaar te worden gebracht om te kijken, maar kan deze multicopter worden ingezet. Dit laatste is een optie die bij Delft Dynamics veel gebruikt wordt. Hiervoor is een stabilisatie-systeem ontwikkeld die ervoor zorgt dat de camera tijdens het vliegen de bewegingen van de multicopter tegen gaat, waardoor de camera een stabiel beeld kan opnemen.

Er is echter een probleem, het systeem wat op dit moment gebruikt wordt door Delft Dynamics is niet meer uitbreidbaar. Het systeem draait op een Arduino Mini en die zit aan het maximum van zijn kunnen. Door de ontwikkelingen in embedded processing en miniatuur sensoren wordt het nu rendabel om een extra ARM-processor met eigen sensoren toe te voegen aan het camera-stabilisatie-systeem zelf. Tijdens dit project zal het huidige systeem van de Arduino worden overgezet naar en geoptimaliseerd voor de Teensy 3.0. Dit gebeurt aan de hand van de waterval methode. In deel twee van dit verslag is aan de hand van de verschillende fasen van de waterval methode te lezen op welke manier dit project is aangepakt.

Tijdens de analyse fase bleek echter dat het probleem vooral zat in de warrige code van het huidige systeem. Dit systeem in namelijk gebaseerd op een gratis framework dat op internet wordt aangeboden, maar zeer onoverzichtelijk in elkaar zit. Op deze manier is het ook moeilijk uitbreidingen toe te voegen aan het systeem.

Naast het optimaliseren zijn er ook een aantal uitbreidingen doorgevoerd, waardoor het systeem nog prettiger wordt om te gebruiken. Hierbij is te denken aan het aansturen van de hoek waarin de camera filmt, het switchen tussen een daglichtcamera en infrarood camera, het kalibreren van deze ingebouwde infrarood camera en de mogelijkheid om systeem variabelen aan te passen, zodat het systeem per uitvoering geoptimaliseerd kan worden. Daarnaast is er een poging gedaan om een loggings mogelijkheid aan het systeem toe te voegen.

Afbeelding 1.2: Helikopter met het nieuwe systeem

In afbeelding 1.2 is het resultaat te zien van dit 17 weken durende project. Het systeem draait uiteindelijk op een Teensy 3.1 en door de extra uitbreidingen is het een systeem wat prettig is om te gebruiken en in veel verschillende situaties kan worden ingezet.

2. Beschrijving organisatie Delft Dynamics

In dit hoofdstuk wordt een beschrijving gegeven van het bedrijf Delft Dynamics. Dit is de organisatie waarvoor de afstudeeropdracht is uitgevoerd. In deze beschrijving wordt bekeken wat de organisatiedoelen zijn, wat de structuur is van het bedrijf, met de daarbij heersende bedrijfscultuur, en tot slot in welke mate de afstudeeropdracht samenhangt met de organisatiedoelen. De informatie die bij de organisatiedoelen wordt gegeven is voornamelijk afkomstig van de persoonlijke ervaringen en gesprekken tijdens de afstudeer periode, maar daarnaast is voor paragraaf 2.1 'Organisatie doelen' ook de website van Delft Dynamics geraadpleegd: www.delftdynamics.com

2.1 Organisatie doelen

Delft Dynamics B.V. is in februari 2006 opgericht en heeft zich gespecialiseerd in het bouwen van robothelikopters: kleine onbemande helikopters die door het slim samenvoegen van computer- en sensorapparatuur gebruikt kunnen worden als stabiel en gemakkelijk te besturen sensorplatform. Deze robot helikopters kunnen in verschillende situaties worden ingezet. Zo kan een overzicht vanuit de lucht worden gemaakt bij gevaarlijke situaties, zoals:

- Overzicht bij verkeersongelukken
- Het vinden van 'hot spots' bij branden
- Metingen van concentraties van gevaarlijke stoffen
- Inspectie van dijken
- Kustwacht en grensbewaking

Daarnaast is het ook mogelijk om routine matige taken, zoals: het observeren van files of inspectie van het spoor, hoogspanningsmasten of olie- en gasleidingen, te laten uitvoeren door deze helikopters. Op deze manier kunnen deze inspecties, die normaal door mensen worden gedaan, voor veel lagere kosten worden uitgevoerd.

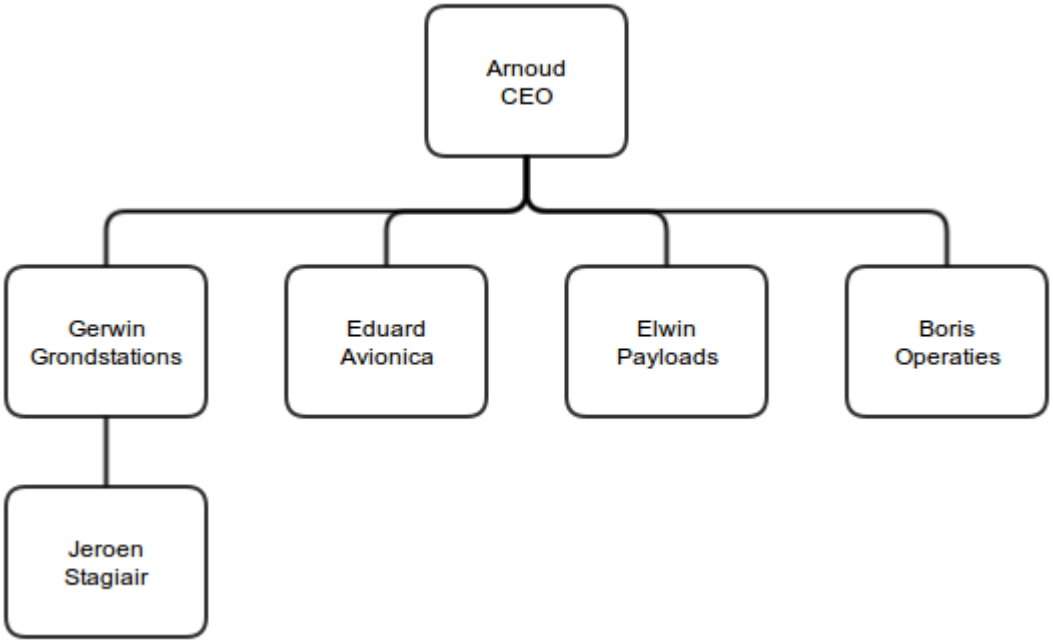
Afbeelding 2.1: Afbeelding van hoogspanningsmast, gemaakt door een helikopter van Delft Dynamics

Een voorbeeld hiervan is te zien in afbeelding 2.1. In opdracht van netbeheerder Tennet is, in samenwerking met Liandon, een inspectie van een hoogspanningsmast uitgevoerd. Hierbij zijn foto's en video's gemaakt met zowel daglicht- als infrarood-camera's. De foto van de top van de hoogspanningsmast toont de kwaliteit van de gemaakte beelden.

Klanten van Delft Dynamics kunnen deze helikopters, inclusief piloot, huren om metingen uit te voeren of foto's te maken, maar het is ook mogelijk om helikopters te kopen. Hierbij is de piloot de persoon die vanaf de grond de helikopter bestuurd. Zo is een aantal robothelikopters geleverd aan de Korps Landelijke Politiediensten (KLPD) en Defensie voor operationele evaluatie. Dit is ook meteen de reden waarom sommige informatie,die in dit afstudeerverslag staat, vertrouwelijk is. Vanwege veiligheidsredenen willen de KLPD en Defensie niet dat bekend wordt welke technieken in de helikopter gebruikt worden. Om deze reden is bij aanvang van de afstudeerperiode een geheimhoudingsverklaring ondertekend.

2.2 Organisatie structuur

In afbeelding 2.2 is een organogram te zien van Delft Dynamics. Onderaan deze afbeelding wordt het diagram verder toegelicht.



Afbeelding 2.2: Organogram Delft Dynamics

De bedrijfscultuur die binnen de organisatie heerst is informeel en heeft een platte hiërarchie. In het organogram is te zien dat het bedrijf wel een CEO heeft, maar dit is meer een formele rol, omdat iemand het moest zijn. Binnen het bedrijf heeft iedereen in theorie zijn eigen taak waar hij mee bezig is, maar in de praktijk komt het veel voor dat er samen wordt gewerkt en mensen delen van de taken van een collega overnemen als deze het wat drukker heeft. Iedereen spreekt elkaar aan met je en als je even met een andere collega wil overleggen loop je gemakkelijk bij hem naar binnen. Dit zorgt voor een zeer open en vriendelijke sfeer.

2.3 De relatie met de afstudeeropdracht

Afbeelding 2.3: De RH4 Spyder

In 2013 bracht Delft Dynamics haar nieuwste product op de markt: de RH4 'Spyder', te zien in afbeelding 2.3. Deze compacte en lichtgewicht multicopter kan verschillende soorten camera's dragen en is zeer gemakkelijk te bedienen. Op deze wijze is het mogelijk om op een eenvoudige manier vanuit de lucht foto- en videobeelden te maken van de omgeving.

Het probleem is echter dat het systeem wat de camera stabiliseert, niet toegankelijk is voor uitbreidingen. Bij het systeem wordt gebruik gemaakt van een opensource framework dat op internet wordt aangeboden en speciaal is ontwikkeld voor dit stabilisatie-systeem. Tijdens het afstuderen ligt de focus op het optimaliseren van dit systeem voor de wensen van Delft Dynamics, zodat het mogelijk is later extra functionaliteit aan het systeem toe te voegen. Een verdere uitwerking van deze opdracht is te vinden in hoofdstuk drie: 'Opdrachtomschrijving'.

3. Omschrijving camera-stabilisatie-systeem optimalisatie

In dit hoofdstuk wordt dieper ingegaan op de opdracht die tijdens het afstuderen is uitgevoerd. Hierbij wordt gekeken naar de aanleiding, probleemstelling, doelstelling en resultaat van de opdracht.

3.1 Aanleiding

De huidige avionica van de RH4 Spyder stabiliseert de multicopter en verzorgt ook de compensatie voor de bewegingen van de multicopter voor de camera. Het huidige camera-stabilisatie-systeem maakt gebruik van een Arduino mini. Door de ontwikkelingen in embedded processing en miniatuur sensoren wordt het nu rendabel om een extra (ARM-)processor met eigen sensoren toe te voegen aan het camera-stabilisatie-systeem zelf. Hierna is het mogelijk om eventueel extra functionaliteit aan de ARM-processor toe te kennen.

3.2 Probleemstelling

De huidige avionica zit aan het maximum van zijn kunnen, waardoor er geen mogelijkheid is om uitbreidingen te doen aan het camera-stabilisatie-systeem.

3.3 Doelstelling

Voor het nieuwe systeem is een aantal hardwarecomponenten door de opdrachtgever uitgekozen. Dit zijn een Teensy 3.0, MPU6000 sensor, Gimbal en GoPro camera. Met deze componenten moet een nieuwe versie van het camera-stabilisatie-systeem worden gemaakt. De Teensy 3.0 is een USB development board met een ARM-prosessor, deze moet via SPI communiceren met de MPU6000. Dit is een IMU die bestaat uit een 3-assige gyroscoop en acceleratiesensor en een temperatuur sensor. Aan de hand van de gegevens van de sensor stuurt de Teensy via PWM de Gimbal aan. De Gimbal is een systeem met twee motoren die de pitch en roll bewegingen van de helikopter kan neutraliseren.

3.4 Resultaat

Het uiteindelijke resultaat is een geoptimaliseerde versie van het huidige camera-stabilisatie-systeem met de doorgevoerde uitbreidingen. Hierbij wordt vooral gefocust op de optimalisatie van de begrijpelijkheid en uitbreidbaarheid van de code. Daarnaast zal de code ook worden aangepast om goed te kunnen werken op de ARM-processor.

3.5 Veranderingen in de opdracht tijdens het project

Bij de probleem analyse, die in paragraaf 5.2 verder wordt beschreven, bleek echter dat er meer aan de hand was dan de vooraf veronderstelde problemen. Het huidige systeem maakt gebruik van een open source framework. Het probleem met dit framework is dat het heel rommelig is opgezet, waardoor het zeer onoverzichtelijk is wat de verschillende functies precies doen en hoe het systeem werkt. Hierdoor is het moeilijk om extra functionaliteit aan het systeem toe te voegen. Daarnaast bleek tijdens het uitvoeren van de opdracht, dat de opdrachtgever ook nog een viertal uitbreidingen wilde doorvoeren, omdat het project sneller verliep dan gedacht: de mogelijkheid om systeem variabelen aan te passen, de pitch hoek van de camera te kunnen aansturen, een aantal functies om met de camera te communiceren en de mogelijkheid om gegevens op te slaan in een logfile. Dit werd echter pas bekend tijdens de constructiefase. De uitbreidingen worden verder toegelicht in hoofdstuk 9.

Het afstudeerplan wat ten grondslag lag aan de afstudeer opdracht, is te vinden in bijlage A.

4. Project aanpak

In dit hoofdstuk wordt beschreven op welke manier het project is ingepland. Hierbij wordt onder andere gekeken naar de keuze van de ontwikkel- en testmethode. Bij het bepalen van de juiste aanpak werd nog verondersteld dat het project alleen zou gaan om de optimalisatie van de huidige code. Op dit moment was nog niet bekend dat: het probleem vooral lag bij het rommelig opgezette framework en dat de opdrachtgever ook uitbreidingen wilde doorvoeren.

Voordat het project gestart werd, wilde de opdrachtgever zeker weten, dat de begrippen die tijdens het project kunnen voorkomen, goed duidelijk zijn. Deze begrippen heeft de opdrachtgever in een begrippenlijst geplaatst, met als opdracht de juiste beschrijving te geven bij deze begrippen. Dit is een standaard procedure van het bedrijf bij het aannemen van stagiaires. De meeste begrippen waren al bekend. Voornamelijk de begrippen die specifiek te maken hebben met luchtvaart moesten worden onderzocht, net als de specifieke eigenschappen van een aantal communicatie protocollen die gebruikt worden. Door zelf de beschrijvingen van de begrippen te maken of op te zoeken, wordt ervoor gezorgd dat er al een goede basis kennis wordt opgedaan over de projectomgeving. Deze begrippenlijst is aan het einde van dit verslag toegevoegd als hoofdstuk “Verklarende woordenlijst”. In deze lijst staan alle begrippen uit de begrippenlijst, met daarnaast een aantal zelf toegevoegde begrippen die naar eigen inzicht ook uitleg nodig hadden.

Nadat de begrippenlijst was opgesteld en deze door de opdrachtgever was goedgekeurd, kon worden begonnen aan de inrichting van het project. Hiervoor is het plan van aanpak opgesteld. De informatie die nodig was om dit plan van aanpak te kunnen schrijven, is voornamelijk opgedaan uit gesprekken met de opdrachtgever. In deze gesprekken is nogmaals besproken wat de opdracht is en wat de daarbij behorende projectgrenzen zijn. Op die manier is duidelijk vastgesteld dat de focus alleen ligt op het stabiliseren van de camera. Het verwerken van de camerabeelden en de communicatie tussen het camera-stabilisatie-systeem en de rest van het helikopter systeem, vallen op dit moment buiten de opdracht. Dit zijn eventuele uitbreidingen die in de toekomst gedaan kunnen worden.

4.1 Bepaling ontwikkelmethode

Tijdens het opstellen van het plan van aanpak, is ook de te volgen ontwikkelmethode bepaald. Als eerste is met de opdrachtgever overlegd of er binnen het bedrijf een bepaalde ontwikkelmethode wordt gevolgd die moest worden aangehouden. Dit was niet het geval en de opdrachtgever gaf de vrije keus om zelf een geschikte ontwikkelmethode te bepalen.

Vervolgens zijn de verschillende aspecten binnen het project, die van invloed zijn op de keuze van de ontwikkelmethode, op een rijtje gezet. Bij het opstellen van deze aspecten was nog niet bekend dat er nog uitbreidingen aan het project toegevoegd zouden worden, zoals in hoofdstuk 9 wordt beschreven. Deze aspecten zijn opgenomen in tabel 4.1, met de toelichting op welke manier en in welke mate deze aspecten zich in het project voordoen.

Omvang van het project	De omvang van het project is niet heel groot. De focus ligt op één systeem, waarbij nog niet gelet hoeft te worden op de communicatie van dit systeem met andere systemen. Daardoor is het een vrij afgebakend project.
Inhoud van het project	Inhoudelijk bestaat het project grotendeels uit onderzoeken en ontwerpen. Het daadwerkelijk ontwikkelen van de software zal minder tijd in beslag nemen, omdat het huidige systeem als basis wordt genomen, wat vervolgens wordt geoptimaliseerd.
Duidelijkheid einddoel	Het einddoel is goed gedefinieerd en loopt zeer weinig kans om te veranderen. Mocht het onverhoopt toch veranderen, dan zal dit minimaal zijn.
Inhoudelijke kennis	Inhoudelijk ontbreekt er kennis over bepaalde onderdelen van het project, waaronder de onbekendheid met een ARM processor, waardoor de onderzoeks- en ontwerpfasen van het project wat uitgebreider zullen zijn om vertrouwd te raken met dit type processor.

Tabel 4.1: Aspecten van het project

Aan de hand van deze aspecten is bepaald welke strategie gebruikt zou worden. De incrementele en iteratieve aanpak vielen af, omdat deze vooral geschikt zijn voor grote projecten waar het einddoel nogal eens kan wijzigen tijdens het ontwikkel traject. Via de incrementen of iteraties kan dan worden geverifieerd of dat wat ontwikkeld wordt, nog wel is gewenst. Daarnaast is de omvang van het ontwikkelgedeelte van het project veel minder dan wanneer een systeem van de grond af aan moet worden opgebouwd. De nadruk zal meer liggen op het analyseren en ontwerpen van het framework. Er hoeft op dit moment geen extra functionaliteit aan het systeem worden toegevoegd en de code die ontwikkeld wordt, is voor een groot gedeelte afkomstig uit het al bestaande framework. Kijkend naar deze redenen was de verwachting dat het werken in iteraties eerder zou afremmen bij de uitvoering van het project, dan dat het de veiligheid biedt die bij grotere projecten van belang is. Dat zijn de redenen waarom er is gekozen voor de watervalmethode.

Vervolgens moest worden onderzocht welke fasen de waterval methode doorloopt. Als eerste indicatie is op Wikipedia informatie opgezocht. De informatie van de Wikipedia pagina is gebaseerd op het document “NAGING THE DEVELOPMENT OF LARGE SOFTWARE SYSTEMS” van Dr. Winston W. Rovce . De fasering op de Wikipedia pagina wijkt iets af van het document Dr. Winston W. Rovce , maar sloot beter aan bij de eisen van het project.

In tabel 4.2 staan de fasen van de waterval methode vermeld, met daarbij aangegeven: wanneer deze fasen worden uitgevoerd, wat globaal de inhoud is van de fasen en wat de mijlpalen zijn die aan het einde van de verschillende fasen worden opgeleverd.

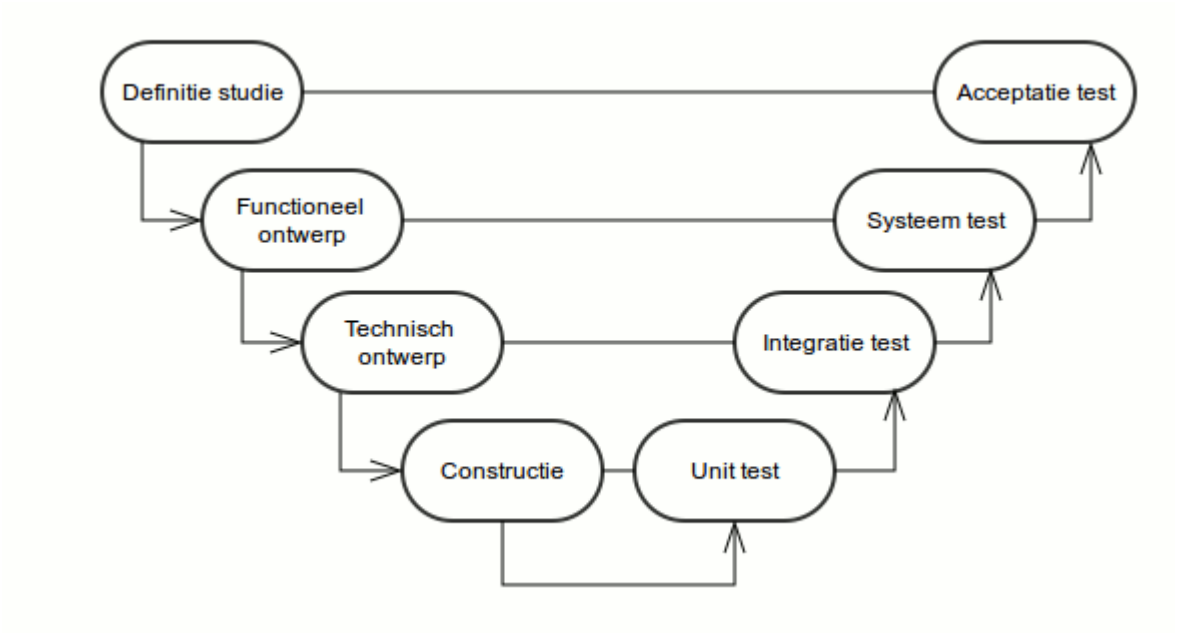
Definitiestudie/ analyse	Maandag 10 – dinsdag 18 februari 2014
	In deze fase wordt er een beter beeld gekregen van het project. Zo wordt er een plan van aanpak geschreven die als basis dient voor de rest van het project, wordt er gekeken hoe het huidige systeem in elkaar zit en wordt er hierover een probleemanalyse uitgevoerd.
	Mijlpalen: Begrippenlijst, plan van aanpak, oriëntatie document
Basisontwerp	Woensdag 19 februari – zondag 9 maart 2014
	In deze fase wordt gekeken naar het toekomstige systeem. Er wordt gekeken wat het systeem moet kunnen en de informatie over de verschillende hardware onderdelen wordt opgezocht. Vervolgens wordt met deze informatie een globaal ontwerp van het systeem gemaakt.
	Mijlpalen: Presentatie, Requirements document, onderzoek naar gcc compiler

Technisch ontwerp/ detailontwerp	Maandag 10 – zondag 23 maart 2014
	In deze fase wordt dieper ingegaan op het globale ontwerp van de vorige fase. Er wordt bijvoorbeeld beschreven op welke manier de sensoren worden uitgelezen en op welke manier de motoren worden aangestuurd. Hierbij moet worden gedacht aan een beschrijving van communicatie protocollen die worden gebruikt en de manier waarop en op welke pinnen de verschillende componenten worden aangesloten. Ook wordt de laatste hand gelegd aan het testplan.
	Mijlpalen: Gesoldeerde Teeny 3.0, Technisch ontwerp, testplan met testcases
Bouw/ implementatie	Maandag 31 maart – zondag 27 april 2014
	In deze fase wordt de software ontwikkeld. Aan de hand van de eerst gemaakte ontwerpen wordt de code gemaakt voor dit systeem en worden tijdens het ontwikkelen unit tests uitgevoerd.
	Mijlpalen: code
Testen	Maandag 28 april – zondag 11 mei 2014
	In deze fase wordt het systeem uitvoerig getest aan de hand van het eerder opgestelde testplan. Hierbij worden de integratie tests en systeem tests uitgevoerd. Bij de integratietest wordt getest of de losse software onderdelen samen werken. Daarnaast wordt in de systeem tests gekeken of het systeem helemaal naar behoren functioneert. Tijdens het testen kan het voorkomen dat stukken van het systeem moeten worden aangepast.
	Mijlpalen: Testresultaten
Integratie	Maandag 12 mei – vrijdag 6 juni 2014
	In deze fase wordt het camera-stabilisatie-systeem aan de 'Spyder' bevestigd en wordt via een systeem en acceptatie test gekeken of het systeem ook in de lucht nog steeds goed functioneert en of de opdrachtgever tevreden is met het resultaat. Tijdens het testen kan het voorkomen dat stukken van het systeem moeten worden aangepast.
	Mijlpalen: Testresultaten, geïmplementeerd systeem, bijbehorende documentatie, presentatie
Beheer en onderhoud	Vanaf vrijdag 6 juni 2014
	Deze fase valt in principe buiten de scope van dit project. Maar tijdens dit project wordt documentatie geschreven over de werking van het systeem, waardoor het mogelijk is om het systeem in de toekomst te kunnen beheren en onderhouden.
	Mijlpalen: -

Tabel 4.2: Fasering van het project met tijdsplanning, globale doelen en mijlpalen

4.2 Bepaling testmethode

Om een goede project aanpak te kunnen maken, is naast de ontwikkelmethode ook bepaald welke testmethode wordt gebruikt. Aangezien het ontwikkelproject via de watervalmethode verloopt, is een testmethode gekozen die aansluit bij het waterval model. Dit is het V-model geworden, te zien in afbeelding 4.1, in combinatie met de SmarTEST methode. De overige invulling van het testproces wordt gedaan in de basisontwerpfase, wat is te lezen in hoofdstuk 6 van dit verslag.



Afbeelding 4.1: Diagram van het V-model

Deze informatie is allemaal opgenomen in het plan van aanpak. Het plan van aanpak bevat verdere informatie over de vormgeving van het project en een gedetailleerde strokenplanning. Het plan van aanpak is samen met de strokenplanning terug te vinden in bijlage B.

DEEL 2: Verslag van het ontwikkelproces

5. Definitiestudie/analyse

In dit hoofdstuk worden de activiteiten beschreven die in de eerste definitiestudie/analyse fase zijn uitgevoerd. In dit hoofdstuk wordt de probleemanalyse beschreven die over de huidige situatie is uitgevoerd, met de hiervoor gekozen oplossing.

Nadat de structuur van het te volgen traject was bepaald, is de huidige situatie nog beter bestudeerd. Om te kijken welke problemen er zijn aan de huidige situatie, is een probleem analyse uitgevoerd. Om dit goed te analyseren is aan de opdrachtgever gevraagd of er een huidige, werkende versie van het systeem beschikbaar was om te onderzoeken. Dat was helaas niet mogelijk, aangezien deze systemen nog in gebruik waren. Het enige dat beschikbaar was, was de code die op het systeem draait en een lijst met de huidige hardware van het systeem. Daarnaast zijn er ook gesprekken met de opdrachtgever gevoerd om te achterhalen welke problemen zij op dit moment ondervonden.

5.1 De gebruikte componenten

Op dit moment bestaat het camera-stabilisatie-systeem uit drie verschillende hardware componenten:

- Arduino Mini als controller
- MPU6050 sensor (3 assige gyroscoop en acceleratie sensor)
- Gimbal met twee motoren

De software die op dit systeem draait is een gratis open-source framework, wat speciaal is ontwikkeld voor de besturing van dit systeem.

5.2 Het probleem

Van tevoren was bekend dat het systeem niet verder uitbreidbaar is, omdat de Arduino aan het maximum zit van wat hij aankan. Maar tijdens het analyseren van de code en in gesprekken met de opdrachtgever, bleek er meer aan de hand te zijn.

De code van het framework is op dit moment moeilijk uitbreidbaar. Aan het begin van de analyse is geprobeerd met het programma Doxygen een klassendiagram te genereren van het framework. Dit lukte helemaal niet, omdat de code een warboel is van allemaal functies. In de code wordt ook geen gebruik gemaakt van klassen, maar zijn alle functies die horen bij één aspect van het systeem, los in dezelfde file gezet. Ook zijn er twee files waarin vrijwel alle defines of variabelen staan, die in de rest van het systeem gebruikt worden, zonder dat duidelijk is bij welk onderdeel van het systeem ze thuis horen. Daarnaast is het framework zeer slecht gedocumenteerd. Hier en daar staat wat commentaar, maar dit is lang niet toereikend. Dit alles bij elkaar zorgt ervoor dat de functie van alle code niet meteen duidelijk is. De wens om in de toekomst meer functionaliteit aan dit systeem toe te voegen is mede dankzij de rommelige code zeer moeilijk.

Tijdens het onderzoek is van iedere file vastgelegd welke functies de file bevat en wat deze functies doen. Ook voor de file met variabelen is gekeken wat de verschillende variabelen inhielden. Hierbij zijn direct al overtollige variabelen en functies weg gehaald, die nergens meer gebruikt werden, maar nog niet waren weggehaald. Op deze manier is stapje voor stapje duidelijk geworden hoe het systeem in elkaar zit.

Naast de warrige code komt het af en toe voor dat het systeem vastloopt. Dit gebeurt op het moment dat het systeem meer moet stabiliseren dan fysiek mogelijk is. Op deze momenten tikt de camera tegen de onderkant van de helikopter aan en kan zichzelf dus niet verder stabiliseren. Uit testen, uitgevoerd door de opdrachtgever voor de aanvang van het project, is gebleken dat het systeem vast loopt op het moment voordat er informatie vanaf de sensor aan de Arduino wordt teruggegeven. Dit blijkt te maken te hebben met de I^2C communicatie, waarbij op het moment van vastlopen de master (Arduino Mini) tevergeefs wacht op informatie van de slave (MPU6050).

5.3 De gekozen oplossingen

Om in de toekomst meer functionaliteit aan het systeem te kunnen toevoegen, heeft de opdrachtgever ervoor gekozen om een heel ander bordje te gebruiken als controller, de Teensy 3.0. Dit is een bordje gebaseerd op een ARM processor met veel meer capaciteit dan de Arduino, zowel als processor als geheugen. Daarnaast heeft de opdrachtgever er ook voor gekozen om een andere versie van de sensor te gebruiken. Dit wordt de MPU6000. Dit is vrijwel dezelfde versie als de 6050, maar met de extra mogelijkheid om via SPI te communiceren in plaats van alleen via I^2C. De hoop is dat het systeem met gebruik van deze nieuwe communicatie niet meer vastloopt.

Om het probleem van de warrige code op te lossen, wordt als eerste, op basis van de kennis die is opgedaan over de werking van het huidige systeem, een nieuwe en globale opzet van het systeem gemaakt. Daarbij moet worden gedacht aan het opstellen van de eisen van het systeem, globaal kijken hoe de verschillende hardware onderdelen met elkaar gaan communiceren en op welke manier de code wordt gestructureerd.

Als deze globale opzet bekend is wordt dieper ingegaan op het technische ontwerp van het systeem. Hierbij moet worden bepaald hoe de hardware gaat worden aangestuurd, op welke pinnen de verschillende hardware componenten moeten worden aangesloten en moet er een ontwerp komen van het framework, waarbij er een aantal verschillende klassen worden ontworpen om de structuur van het systeem overzichtelijk te maken.

Daarna kan daadwerkelijk worden begonnen met de ontwikkeling van het framework. Hierbij moet worden gekeken welke code van het oude framework hergebruikt kan worden voor het nieuwe framework.

Tegelijkertijd wordt er grondig aandacht besteed aan het testen van het systeem, om er zeker van te zijn dat na de herstructurering de werking en kwaliteit van het systeem niet achteruit gaat en hopelijk zelfs verbeterd zijn.

Door deze bevindingen is het project er iets anders uit gaan zien dan bij aanvang werd gedacht. Er komt iets meer kijken bij het uiteindelijk ontwikkelen van de code. Het kan zijn dat het wenselijk is om in de bouw/implementatie fase een iteratie strategie toe te passen. Deze beslissing wordt genomen aan het einde van de functionele ontwerp fase, omdat op dat moment definitief duidelijk is hoe de software in elkaar moet zitten en in welke mate er gebruik kan worden gemaakt van het huidige systeem.

Deze bovenstaande probleemanalyse is opgenomen in het oriëntatie document. Naast deze analyse is in het oriëntatie document ook de tabel opgenomen, waarin van iedere file in het huidige framework is uitgelegd wat de functie is van de code die erin staat en op welke manier deze samenhangt met de rest van het framework. Dit oriëntatie document is terug te vinden in bijlage C.

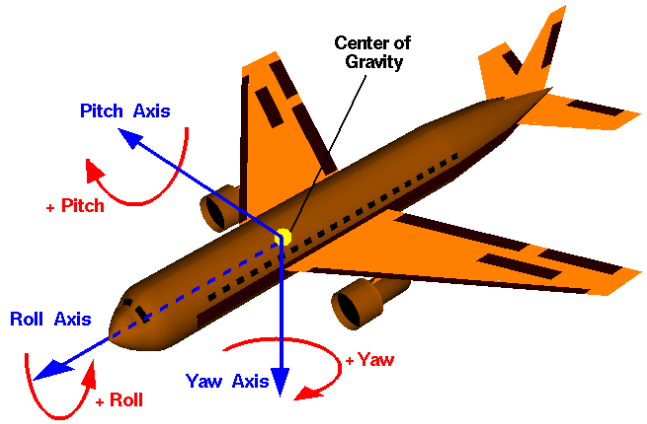
6. Basisontwerp

In dit hoofdstuk worden de activiteiten beschreven die zijn uitgevoerd tijdens de basisontwerp fase. Zo is in deze fase een requirements document opgesteld, een basisontwerp van het systeem gemaakt, een testplan gemaakt en onderzoek gedaan naar de gcc-compiler voor ARM processen.

6.1 Opstellen requirements document

Om van tevoren goed vast te leggen aan welke eisen het systeem moet voldoen is er een requirements document samengesteld. Aan de hand van de gesprekken met de opdrachtgever en de probleem analyse, was een goed beeld gekregen waaraan het systeem moet voldoen. Eigenlijk kent het systeem slechts één functionele eis:

- Het systeem moet aan de hand van de MPU6000 kunnen uitlezen wat de draaiing is van de roll en pitch as van de multicopter en met behulp van een gimbal en twee brushless motoren deze draaiing bij de camera tegengaan. Op deze manier moet de camera stabiel kunnen filmen. Om dit voor elkaar te krijgen moet de MPU6000 met een frequentie van 1kHz worden uitgelezen, waarna de stand van de motoren moet worden aangepast.



Afbeelding 6.1: Roll, Pitch en Yaw beschrijving

Ter verduidelijking van deze eis is in afbeelding 6.1 een vliegtuig te zien, met daarbij aangegeven wat de Roll, Pitch en Yaw assen van een luchtvoertuig zijn.

Deze functionele eis is afgeleid aan de hand van de huidige werking van het systeem. Tijdens het stabiliseren van de camera wordt de sensor 1000 keer per seconde uitgelezen. Vervolgens worden deze waarden gebruikt om te bepalen in welke stand de motoren moeten komen te staan om deze draaiing tegen te gaan. Dit is de basis functie van het systeem waar de opdrachtgever op dit moment de aandacht op wil vestigen.

Maar het grootste gedeelte van het werk zit voornamelijk in de niet-functionele eisen en dan vooral op het punt van maintainability (onderhoudbaarheid). Zoals in het vorige hoofdstuk is beschreven, is de huidige code een chaos. Een zeer belangrijke eis aan het nieuwe systeem is dat het overzichtelijk is, zodat later extra functionaliteit aan het systeem is toe te voegen.

Dit zijn de belangrijkste eisen aan het systeem. Daarnaast zijn nog andere eisen aan het systeem vastgelegd, zoals de communicatie protocollen die tussen de verschillende hardware componenten gebruikt gaan worden en dergelijke.

Nadat de concept versie van het requirements document was gemaakt, is deze voorgelegd aan de opdrachtgever. Aangezien de opdrachtgever zelf ook een software ontwikkelaar is, was hij in staat het document te begrijpen zonder verdere toelichting. Hij is akkoord gegaan met de inhoud en op deze manier zijn de eisen die aan het systeem worden gesteld officieel vastgelegd. Het volledige requirements document is te vinden in bijlage D.

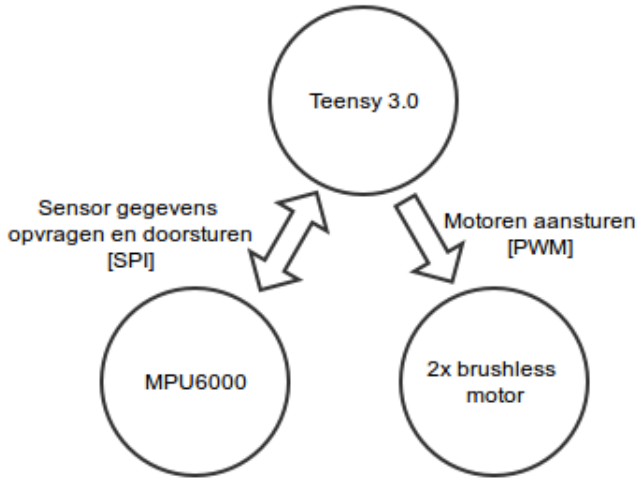
6.2 Maken van een globaal ontwerp

Nadat het requirements document was opgesteld, moest er een globaal ontwerp van het nieuwe systeem worden gemaakt. Dit globale ontwerp wordt in de technische ontwerp fase verder ingevuld. Voor het basis ontwerp is op de officiële site van de Teensy (www.pjrc.com/teensy) informatie opgezocht over de besturing van de Teensy. Zo zijn de algemene hardware specificaties van de Teensy opgezocht, om als naslag te kunnen gebruiken. Daarnaast zijn de timer, PWM en SPI mogelijkheden van de Teensy onderzocht. Voor alle drie de functies zijn op de officiële Teensy site libraries beschikbaar.

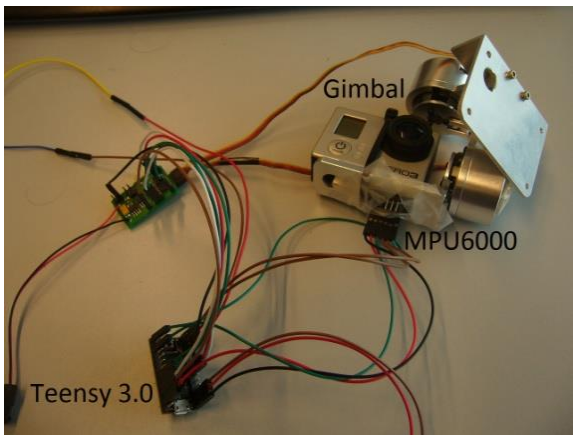
Nadat dit onderzoek was gedaan voor de Teensy, is hetzelfde gedaan voor de MPU6000 sensor. Voor deze sensor zijn ook de specificaties opgezocht en dan voornamelijk welke functionaliteit de verschillende pinnen hebben. Op de officiële site van de MPU6000 (http://www.invensense.com/mems/gyro/mpu6050.html) is een aantal documenten te vinden over het gebruik van de MPU. Via deze documenten is meer kennis opgedaan over het gebruik en functioneren van de sensor.

Tot slot is onderzoek gedaan naar de motoren die gebruikt worden. Dit zijn brushless motoren die door drie verschillende PWM signalen worden bestuurd. In de technisch/detail ontwerp fase wordt verder onderzocht hoe de verschillende hardware componenten op elkaar worden aangesloten en op welke manier de informatie tussen de componenten wordt verstuurd. In die fase wordt duidelijk welke PWM signalen naar de motoren moeten worden gestuurd en wordt de SPI communicatie vormgegeven.

In afbeelding 6.2 is een zeer globaal overzicht gegeven van de verschillende hardware componenten, welke informatie er tussen de componenten wordt verzonden en via welk protocol dit gebeurt. Daarnaast is in afbeelding 6.3 te zien hoe het systeem in de realiteit is aangesloten.

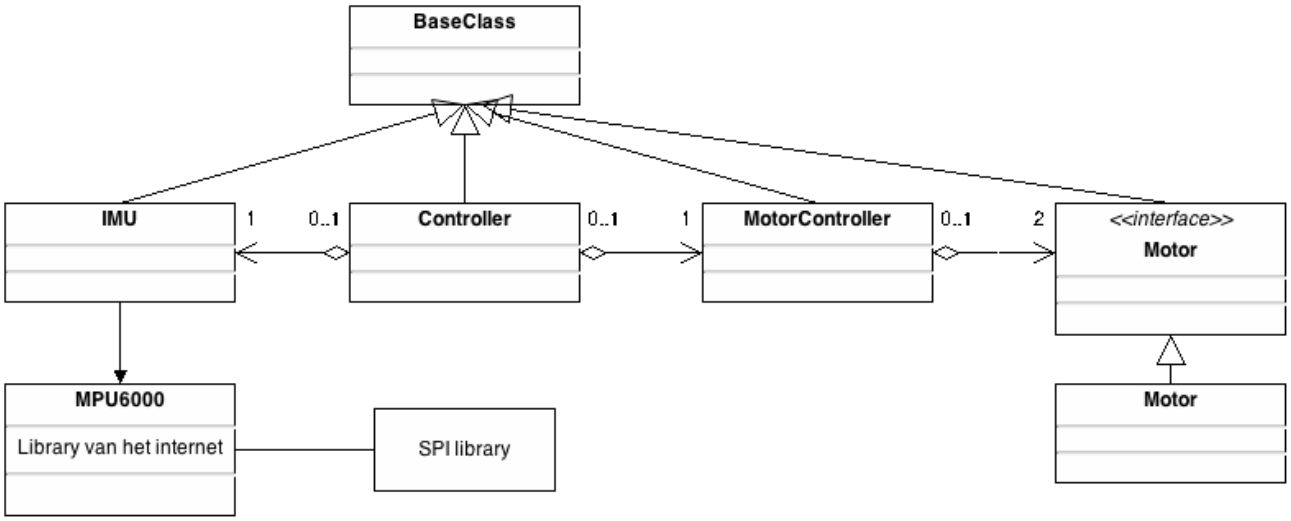


Afbeelding 6.2: Globaal aansluitdiagram componenten



Afbeelding 6.3: De aangesloten componenten

Daarnaast is van dit basis ontwerp een globaal klassendiagram ontworpen. Dit diagram is te zien in afbeelding 6.4 en wordt na de afbeelding verder toegelicht.



Afbeelding 6.4: Globaal klassendiagram

Om tijdens het ontwikkelen een gemakkelijke manier van debuggen mogelijk te maken is de BaseClass ontworpen. Deze klasse is helemaal bovenaan in het diagram te vinden. Deze BaseClass moet een variabele gaan bevatten om bij te houden in welke debug modus het object zich bevindt, zodat ieder object in het framework in een andere modus van debuggen is te zetten. Daarnaast is het in de toekomst mogelijk om eventuele uitbreidingen die alle klassen aangaan, aan de BaseClass toe te voegen.

Onder deze BaseClass is de Controller klasse te zien. Dit wordt de 'hoofdklasse' die al het regelwerk doet in het systeem en de koppeling vormt tussen de gegevens uit de IMU en de aansturing van de motoren. In het oude framework bevinden zich een setup() en een loop() functie, die het centrale regelwerk van het systeem op zich namen. Er is voor gekozen om deze twee functies, samen met een aantal extra functies, op te nemen in een centrale klasse.

Iedere Controller bevat vervolgens een MotorController. Deze MotorController zorgt ervoor dat de juiste gegevens naar de Motor objecten worden gestuurd om deze in de goede stand te zetten. Iedere MotorController bevat twee Motoren. Een roll en een pitch motor. Op deze manier zijn deze twee bewegingen van de multicopter te corrigeren. Het Motor object berekent de juiste PWM configuratie om naar de fysieke motor te zenden. Er is gebruik gemaakt van een Motor interface, om op deze manier de mogelijkheid te bieden in een later stadium gemakkelijk andere motoren op het systeem te kunnen aansluiten.

Daarnaast bevat ook iedere Controller een IMU object. Dit is de klasse die verantwoordelijk is voor de communicatie met de sensor. Deze klasse kan de sensor uitlezen en configureren. Er moet nog worden gekeken of het wenselijk is om net als de Motor gebruik te maken van een interface. Dit wordt verder bekeken in de technische ontwerp fase. In deze fase worden de functies en variabelen aan de klassen toegevoegd en wordt bij het maken van de klassen rond de sensor bekeken hoe de structuur het beste kan worden vormgeven: het laten zoals het nu is; werken met een IMU interface of werken met een interface voor de sensor.

6.3 Maken van het testplan en testscenario's

Nadat het basis onderzoek was afgerond, zijn de plannen gemaakt om het systeem te kunnen testen. Deze plannen zijn gemaakt naar aanleiding van de SmarTEST methode. Via deze methode is een testplan gemaakt, samen met verschillende testscenario's. Als basis voor het testplan is gebruik gemaakt van gesprekken met de opdrachtgever, het opgestelde requirements document en het basis ontwerp.

6.3.1 Het testplan

Om goed te kunnen bepalen op welke aspecten er moet worden gelet tijdens het testen van het systeem, is onderzocht welke kwaliteitsattributen uit de ISO 9126 norm op het systeem van toepassing zijn. Door alle attributen van de ISO norm te naast de opgestelde eisen in het requirements document te leggen, zijn een elftal attributen naar voren gekomen die bij het testen van het systeem naar voren moeten komen. Vervolgens is aan de hand van de wensen van de opdrachtgever gekeken welke attributen belangrijker waren en welke wat minder. Hierbij is een relatief belang van alle elf de attributen opgesteld. In tabel 6.1 zijn deze attributen opgenomen, met daarbij per attribuut aangegeven wat het relatief belang is voor het systeem.

Kwaliteitsattributen volgens ISO 9126	Relatief belang (%)
Functionaliteit	
Geschiktheid *Mate waarin de gewenste functies aanwezig zijn en geschikt om gespecificeerde taken uit te voeren.*	10
Juistheid *Juistheid van de uitvoer van het systeem, overeenkomstig de invoer en de gespecificeerde bewerkingen.*	20
Betrouwbaarheid	
Volwassenheid *Mate waarin fouten en kinderziektes verholpen zijn en het systeem vrij blijft van storingen.*	10
Beschikbaarheid *Mate waarin het systeem op de gewenste tijden beschikbaar is voor de gebruiker.*	5
Efficiëntie	
Tijdbeslag *Responstijd, transactiesnelheid, snelheid batchverwerking.*	10
Middelenbeslag *Hoeveelheid benodigde resources (voornamelijk geheugen).*	5
Overzetbaarheid	
Aanpasbaarheid *Gemak waarmee het systeem overgezet kan worden naar een ander hardware/software-platform of naar een nieuwe versie daarvan (zoals bij toevoeging van extra functionaliteit).*	20
Inpasbaarheid *Het gemak waarmee het systeem een bestaand systeem kan vervangen. Mate waarin het systeem aansluit bij de bedrijfsprocessen en (handmatige) procedures.*	5

Onderhoudbaarheid	
Analyseerbaarheid *Gemak waarmee de oorzaak van fouten opgespoord kan worden waarmee te wijzigen onderdelen kunnen worden gevonden.*	5
Testbaarheid *Gemak waarmee de juiste werking getest en gevalideerd kan worden.*	5
Beheerbaarheid *Gemak waarmee het systeem in operationele staat gebracht en gehouden kan worden.*	5

Tabel 6.1: De kwaliteitsaspecten uit de ISO 9126 norm die aan bod komen in het systeem met het bijbehorende relatief belang.

Deze attributen zijn in een Product Risico Matrix (PRIMA) uitgezet tegen de verschillende functionaliteiten van het systeem, te zien in tabel 6.2. Het systeem is op dit moment onder te verdelen in slechts twee functionaliteiten: als eerste het stabiliseren van de camera en als tweede het genereren van debug output. Hierbij is het genereren van debug output van ondergeschikt belang en wordt daarom op slechts drie kwaliteitsaspecten getest: geschiktheid, juistheid en tijdsbeslag. Op het stabiliseren van de camera zijn alle kwaliteitsaspecten uit tabel 6.1 van toepassing. De kruisjes in de PRIMA geven aan hoe belangrijk de verschillende kwaliteit eisen zijn per functionaliteit. Hierbij is rekening gehouden met het relatief belang uit tabel 6.1

	Onderdelen ->	Stabiliseren	Output
Kwaliteitsaspecten	100	90	10
1. Geschiktheid	10	xx	x
2. Juistheid	20	xxx	x
3. Volwassenheid	10	xx	
4. Beschikbaarheid	5	x	
5. Tijdbeslag	10	xx	x
6. Middelenbeslag	5	x	
7. Aanpasbaarheid	20	xxx	
8. Inpasbaarheid	5	x	
9. Analyseerbaarheid	5	x	
10. Testbaarheid	5	x	
11. Beheerbaarheid	5	x	

Tabel 6.2: Product Risico Matrix (PRIMA)

Hierna zijn de verschillende eisen en kwaliteitsaspecten vertaald naar acceptatiecriteria waar het systeem aan moet voldoen bij oplevering en is bepaald welke verschillende testmethoden gebruikt gaan worden om deze criteria te testen. Deze testmethoden zijn gekozen aan de hand van het eerder genoemde V-model (te zien in afbeelding 4.1). Dit V-model beschrijft vier verschillende fasen van testen. Als eerste komen de unittests. Voor deze testen worden geen testscenario's ontwikkeld. Dit zijn de testen die tijden het ontwikkelen worden uitgevoerd om te controleren of de gemaakte stukjes naar wens functioneren. Voor de overige drie fasen worden wel testscenario's ontwikkeld. Bij het maken van deze testen wordt de PRIMA gebruikt om te bepalen waar extra op gelet moet worden en om te controleren of alles wel voldoende is getest. Hierna volgt een overzicht hoe deze testfasen eruit komen te zien:

- Integratie test fase – In deze testfase wordt gekeken of de verschillende componenten samen goed werken. Hoe is de koppeling tussen de Controller en de IMU/MPU6000 en hoe communiceert deze weer met de MotorController/Motoren?
- Systeem test fase – In deze test wordt gekeken of voldaan is aan de functionele eisen. Is het systeem in staat om de sensor 1000 keer per seconden uit te lezen en daarmee de camera te stabiliseren? Bij deze testen is het camera systeem nog niet aangesloten op de multicopter.
- Acceptatie test fase – Deze test is onderverdeeld in een viertal sub tests. Deze testen worden uitgevoerd op het moment dat het systeem klaar is om aan de helikopter te bevestigen.
    - FAT (Functionele acceptatie test) – In deze test wordt het systeem aan de multicopter bevestigd en beoordelen de gebruikers (werknemers van Delft Dynamics) of het systeem voldoet aan de eisen.
    - GAT (Gebruikers acceptatie test) – In deze test wordt gekeken of de gebruikers het systeem werkbaar vinden, dus gemakkelijk in gebruik te nemen is.
    - PAT (Productie acceptatie test) – In deze test wordt gekeken of het systeem voor de beheerders gemakkelijk is te onderhouden.
    - Continuïteit test – Hierbij wordt gekeken of het punt waar het vorige systeem vastliep is verholpen en niet op andere punten vastloopt.

Naast dit alles zijn in het testplan ook overige formaliteiten opgenomen over de wijze van terugkoppeling aan de opdrachtgever en wijze van documenteren. Het testplan is aan de opdrachtgever getoond en na een aantal kleine aanpassingen ging hij hiermee akkoord. Het volledige testplan is te vinden in bijlage E.

6.3.2 De testscenario's

Na het opzetten van het testplan is er begonnen aan het schrijven van de testscenario's voor de acceptatie testen en systeem testen. De acceptatie testen konden al helemaal worden afgerond, omdat op dit moment al duidelijk was aan welke eisen het systeem moest voldoen. Dit geldt ook voor het grootste gedeelte van de systeem testen, maar hier missen nog een aantal kleine details in de beschrijving van de testgevallen. Deze details worden ingevuld aan het einde van de Technisch ontwerp/detailontwerp fase. Aan het einde van deze fase worden ook de scenario's voor de integratie testen geschreven. Bij het maken van de scenario's is nagedacht over de scenario-ID's. Zo moet het mogelijk zijn om ieder scenario terug te vinden aan de hand van deze ID's en deze ID's worden ook gebruikt om de status van de scenario's bij te houden. Zo staat de eerste letter voor de testsoort, de eventuele tweede letter voor de sub-test soort en het cijfer voor de hoeveelste test dit is in die categorie. Vervolgens geeft een streepje met daarachter een cijfer aan om welk testgeval uit het scenario het gaat. Zo staat het scenario-ID AF1-2 voor Acceptatie test, sub test: FAT, nummer 1, geval 2.

In tabel 6.3 is een uitgewerkt testscenario van een systeem test te zien. Dit scenario is opgenomen om een voorbeeld te geven van de structuur van een scenario. De overige scenario's voldoen aan dezelfde structuur. Dit specifieke scenario is geschreven om te testen of de koppeling tussen de Controller en de sensor werkt.

Administratieve gegevens

Scenario-ID **S2**

Omschrijving **De controller moet sensor waarden kunnen opvragen**

Versie	Datum	Auteur	Opmerkingen
0.1	25 feb 2014	Jeroen de Meij	Eerste versie van het scenario
0.2	26 feb 2014	Jeroen de Meij	Kwaliteitseisen toegevoegd aan de testbasis

Testobject en achtergrond

In deze test wordt gekeken of de koppeling tussen de controller en de sensor naar behoren werkt, waarbij de sensor 1000 keer per seconde gegevens moet uitlezen.

Testmissie

Testen of via de controller de sensor uitgelezen kan worden.

Testbasis

1. Als basis voor deze test is de eis om de camera te stabiliseren door onder andere gegevens uit de sensor uit te lezen. Hierbij is als eis gesteld dat de sensor 1000 keer per seconde moet worden uitgelezen en de gegevens worden verwerkt.
2. Daarnaast is deze test gebaseerd op de kwaliteitseis juistheid van het systeem. Te vinden in de PRIMA, opgenomen in het testplan.

Uitgangssituatie en randvoorwaarden

De Teensy moet via een USB-kabel aan de pc zijn verbonden. En de sensor moet zijn aangesloten aan de Teensy.

Testdoelen met testaanwijzingen

- DE SENSOR MOET VOLDOENDE WORDEN UITGELEZEN

In het systeem moet worden bijgehouden hoe vaak de sensor per seconde wordt uitgelezen. Dit wordt vervolgens op het scherm getoond.

Niet testen

Er hoeft niet te worden getest of de gegevens van de sensor juist worden verwerkt om de motoren aan te sturen.

Testgevallen

- Start de timer die de sensor moet uitlezen.
- Houd een teller bij met het aantal keer dat de sensor wordt uitgelezen.
- Toon na een minuut de inhoud van de teller op het scherm.

Verwacht resultaat: Op het scherm wordt getoond dat de sensor in totaal 1000 x 60 = 60.000 keer is aangeroepen.

Tabel 6.3: Testscenario AF1 – Gebruikers testen de beeld kwaliteit.

6.4 Onderzoek naar de gcc-compiler

Tot slot is in deze fase het onderzoek naar de gcc-compiler afgerond. Tijdens de analyse was bij het inwerken van de opdracht al een klein begin gemaakt aan het onderzoek. Uiteindelijk bleek dat op de officiële website van de Teensy (http://www.pjrc.com/teensy/) precies staat uitgelegd hoe de compiler moet worden geïnstalleerd.

Als eerste moet de Arduino IDE worden gedownload en geïnstalleerd. De Teensy kan namelijk gebruik maken van een aantal Arduino libraries. In de volgende stap moet Teensyduino worden geïnstalleerd. Deze Teensyduino gaat er van uit dat de Arduino IDE al is geïnstalleerd en plaatst zichzelf in de map structuur van de Arduino IDE. Op deze manier zijn de libraries voor de Arduino ook grotendeel beschikbaar voor de Teensy. Voor het installeren van Teensyduino is een installatie handleiding gemaakt die terug is te vinden in bijlage F.

Daarnaast kan met een bijgeleverde Makefile het programma worden gecompileerd en op de Teensy worden gezet. Op deze manier wordt er geen gebruik meer gemaakt van de Arduino structuur, die gebruik maakt van een setup() en loop() functie in plaats van een main() functie.

Als programmeertaal kon worden gekozen tussen C en C++, aangezien dit de talen zijn die standaard door de Teensy worden ondersteund. Omdat maintainability een belangrijke eis was voor het systeem is gekozen voor C++, zodat op deze manier met klassen kan worden gewerkt en het systeem al meteen meer structuur krijgt en daardoor ook overzichtelijker wordt.

Aan het einde van deze fase bleek dat het project anderhalve week voor lag op de planning. Op dit moment is ervoor gekozen dit even te laten voor wat het is en door te gaan naar de technisch ontwerp/ detailontwerp fase. Aan het einde van die fase is er een nog beter beeld van het project gekregen en kan met die kennis worden bekeken of de planning aangepast moet worden. Op die manier wordt voorkomen dat de planning iedere keer opnieuw moet worden gemaakt, waardoor er minder tijd besteed kan worden voor het uitvoeren van het project.

7. Technisch ontwerp/detailontwerp

In dit hoofdstuk worden de activiteiten beschreven die zijn uitgevoerd tijdens de technisch ontwerp/ detailontwerp fase. In deze fase zijn de Teensy en MPU6000 van header pins voorzien en is het technisch ontwerp van het systeem gemaakt, waarbij duidelijkheid is verkregen over de invulling, werking en aansluitingen van het systeem.

7.1 Teensy solderen

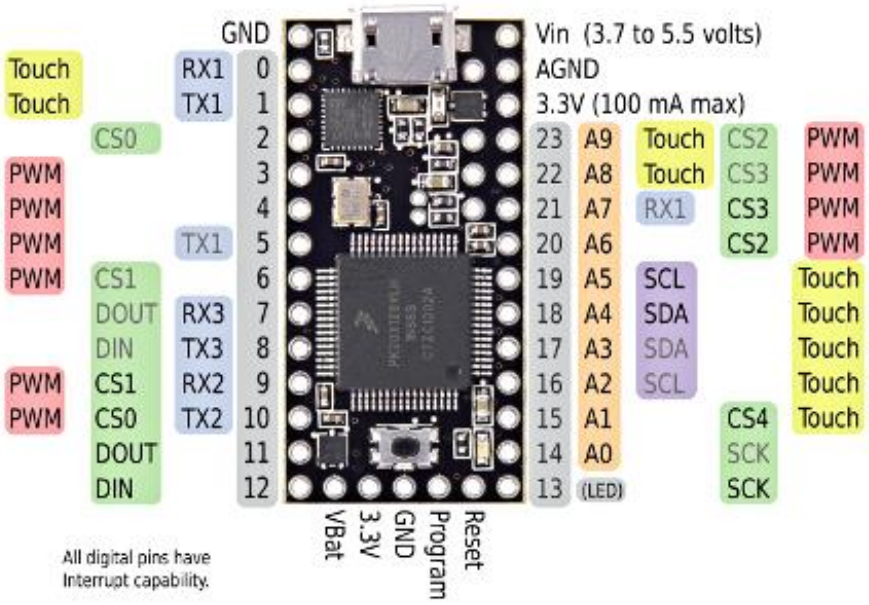
Aan het begin van deze fase zijn de Teensy en de MPU6000 voorzien van header pins. Hiervoor moest eerst worden geleerd te solderen, aangezien dit nog niet was geleerd. Op deze manier kon de sensor op de Teensy worden aangesloten.

7.2 Maken technisch ontwerp

Om in de volgende fase te kunnen beginnen met de ontwikkeling van het systeem, is er nagedacht op welke pinnen de verschillende hardware componenten moeten worden aangesloten. Daarnaast is het basis diagram uit de basisontwerp fase verder uitgebreid, door de functies en variabelen uit het huidige systeem erin te plaatsen. Aan het einde van deze fase is het dus mogelijk om de globale werking van het systeem uit te leggen.

7.2.1 Onderzoek aansluiting hardware componenten

Aan de hand van afbeelding 7.1 is onderzocht op welke pinnen de verschillende componenten moeten worden aangesloten. Na deze afbeelding wordt uitgelegd waar de keuzen op zijn gebaseerd.



Afbeelding 7.1: Pinout van de voorkant van de Teensy

Allereerst is gekeken naar de pinnen waar de sensor op moet worden aangesloten. Dat zijn om te beginnen de 3.3 V pin en de GND pin. Deze zijn uiteraard nodig om de sensor van elektriciteit te voorzien. Daarnaast is het de bedoeling dat de sensor via SPI met de Teensy gaat communiceren. Nu voorziet de Teensy al in een SPI library die standaard een aantal poorten in gebruik heeft, in de afbeelding in het groen aangegeven (pin 9, 10, 11, 12, et cetera). Waarschijnlijk wordt er voor de SPI communicatie niet gebruik gemaakt van deze standaard library. Op het internet is een library gevonden die de MPU6000 kan bedienen en zijn eigen SPI gedeelte erin heeft zitten, maar om uitbreiding/aanpassing gemakkelijk te maken is ervoor gekozen om de sensor toch op de SPI pinnen van de Teensy aan te sluiten. In tabel 7.1 staat aangegeven welke pinnen van

de Teensy en de MPU6000 met elkaar zijn verbonden.

Teensy pin		MPU6000 pin
10	CS0	/CS
11	DOUT	SDI
12	DIN	SDO
13	SCK	SCLK

Tabel 7.1: Verbinding tussen de Teensy en de MPU6000

Hierna is gekeken op welke pinnen de motoren moesten worden aangesloten. Een motor heeft drie verschillende pwm signalen nodig voor de aansturing, dus in totaal zijn er zes poorten nodig die pwm signalen ondersteunen. In afbeelding 7.1 zijn deze pinnen in het rood/roze aangegeven (pin 3, 4, 5, 6, et cetera). In het basis ontwerp was al gekeken naar de mogelijkheden om de pwm signalen aan te passen op frequentie en resolutie. Bij dat onderzoek kwam naar voren dat de Teensy twee timers gebruikt voor de pin aansturing. Als op een pin de frequentie of resolutie wordt aangepast, geldt deze verandering voor alle pinnen die op dezelfde timer zijn aangesloten. In tabel 6.2 is een overzicht te zien welke pinnen er op een timer zitten aangesloten.

Timer	pwm pinnen	Standaard frequentie
0	5, 6, 9, 10, 20, 21, 22, 23	488.28 Hz
1	3, 4	488.28 Hz

Tabel 7.2: De pwm pinnen van de Teensy 3.0

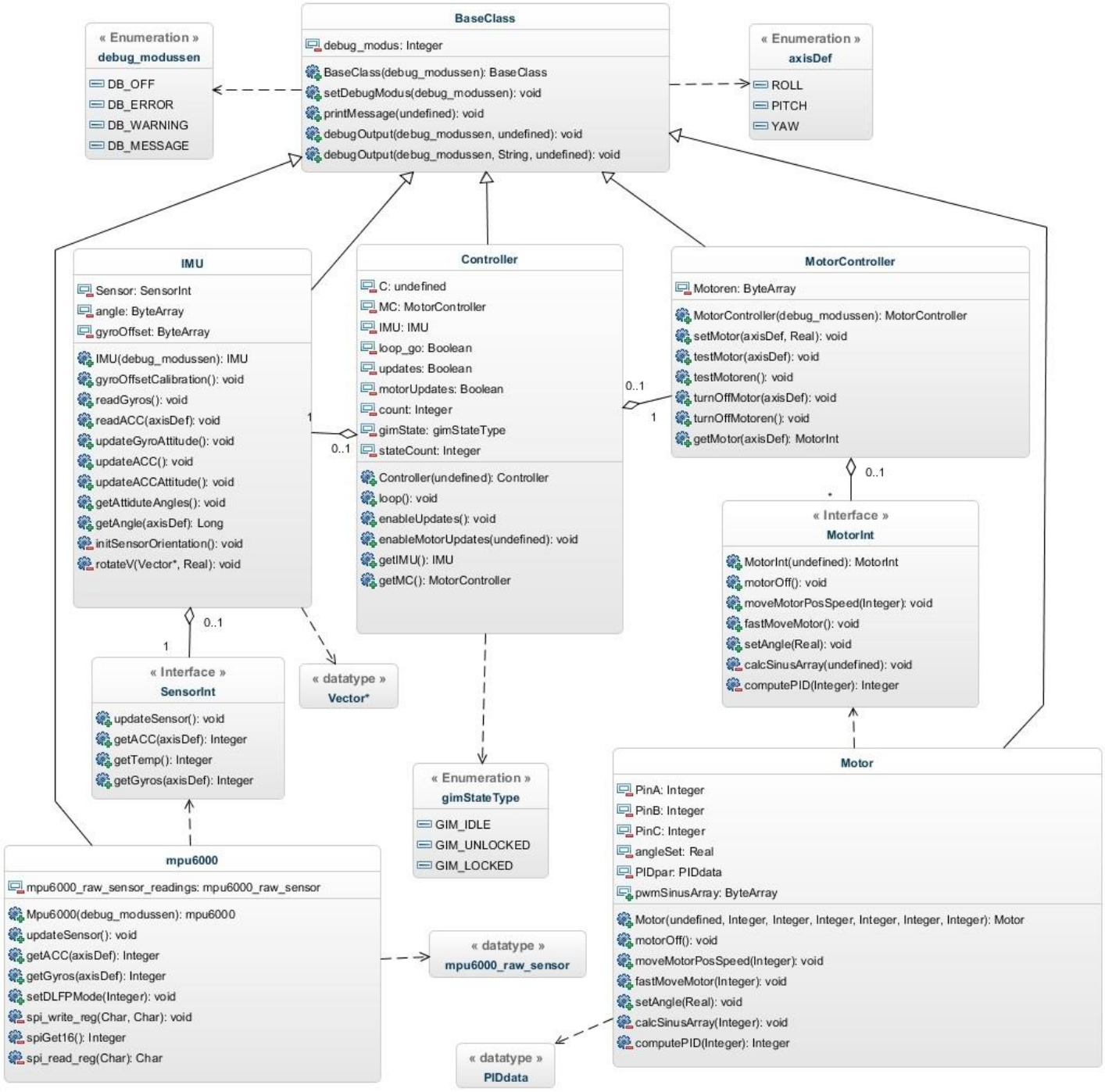
Er is gekeken of alle motoren op timer 0 aangesloten konden worden, omdat op deze manier timer 1 vrij blijft voor een eventuele andere pwm configuratie. Aangezien er nog zeven pinnen van timer 0 vrij waren en er maar zes worden gebruikt voor de motoren, ging dit ook gemakkelijk lukken. Daarom is besloten om de pinnen 5, 6 en 20 te gebruiken voor de roll motor en de pinnen 21, 22 en 23 voor de pitch motor. Op deze manier worden ook alle seriële poorten vrij gehouden. Dit was de wens van de opdrachtgever, zodat later meer functionaliteit toegevoegd kan worden die gebruik maakt van seriële communicatie.

7.2.2 Maken uitgebreid klassendiagram

Bij het maken van het klassendiagram, is het huidige framework onderverdeeld in een aantal klassen. Dit is een vrijwel letterlijke overname van het huidige systeem, waarbij nog niet is gekeken naar het anders moduleren van bepaalde functies. Een voorbeeld hiervan is het letterlijk overnemen van een enumeratie voor states, zonder gebruik te maken van bijvoorbeeld het States design pattern. Nadat het systeem op deze manier is omgezet kan in een volgende stap worden bekeken op welke punten het systeem verbeterd kan gaan worden, dan wel door de code te optimaliseren, dan wel door de structuur anders te maken.

In de analyse fase is al bekeken welke functies moeten worden overgezet om te voldoen aan de functionele eis. Al deze functies hebben een plaats gekregen in het klassendiagram. Als de functies in de constructiefase van het oude systeem naar het nieuwe worden overgezet zullen sommige functies moeten worden aangepast om te werken op de Teensy. Zo worden in de oude code bijvoorbeeld nog een aantal registers gezet voor pwm, die niet zullen werken op de Teensy.

In afbeelding 7.2 is het klassendiagram te zien van het camera-stabilisatie-systeem. Dit is niet het volledige diagram, maar een eenvoudigere versie. In dit diagram zijn wel alle klassen opgenomen, maar een aantal functies is niet getoond, om op deze manier een iets overzichtelijker diagram te kunnen tonen. Op de volgende pagina wordt het diagram verder toegelicht.



Afbeelding 7.2: Klassendiagram camera-stabilisatie-systeem

Zoals al beschreven is in het basis ontwerp, is de Controller het regelcentrum van het systeem. De Controller zorgt ervoor dat alle hardware componenten worden geïnitialiseerd en bevat daarnaast een loop functie die met regelmaat (standaard iedere milliseconden) wordt aangeroepen en de hardware componenten aanstuurt. Tot slot beschikt de Controller over een enumeratie die een aantal States bevat, waar de Controller zich in kan bevinden. Deze States worden bij het initialiseren van het systeem gebruikt om onder andere de PIDcontroller de mogelijkheid te geven zichzelf te configureren en stabiliseren.

De Controller is afgeleid van de BaseClass. Deze klasse is de 'hoofdklasse' van het gehele systeem. Alle overige klassen zijn van deze BaseClass afgeleid, omdat de BaseClass voor een aantal functies zorgt die alle overige klassen in het systeem moeten hebben. Op dit moment zijn dat functies om debug berichten op het scherm te kunnen tonen, maar in de toekomst kunnen eventuele uitbreidingen die alle klassen moeten bezitten in de BaseClass worden geplaatst. Naast de mogelijkheid om debug berichten te kunnen tonen, bevat de BaseClass een enumeratie met de drie verschillende draaiingen die de helikopter kan maken (roll, pitch en yaw). Deze enumeratie wordt in de rest van het systeem ook gebruikt.

De Controller bevat altijd één instantie van een IMU. Deze IMU klasse zorgt voor de communicatie met de daadwerkelijke sensor. De IMU bevat een functie om de sensor te initialiseren en gegevens van de sensor op te vragen. Daarnaast beschikt de IMU over een aantal functies om met deze gegevens de draaiing van het systeem te bepalen. Hierbij wordt gebruik gemaakt van een aantal structs om de gegevens te ordenen, waaronder een struct die een vector representatie geeft van de draaiing van de sensor. Deze strucs zijn niet als dusdanig weer te geven in het diagram. Het diagram is gemaakt via de website app.genmymodel.com en door beperkingen in deze ontwerp software zijn de structs weergegeven als << *datatype* >>.

De IMU heeft op zijn beurt weer altijd één verwijzing naar een daadwerkelijke sensor. Deze verwijzing loopt via een sensor interface, om op deze manier de mogelijkheid te bieden om in de toekomst de sensor gemakkelijk met een andere sensor te kunnen verwisselen of een extra sensor type toe te voegen. Deze sensor klasse zorgt voor de daadwerkelijke communicatie met de sensor. In dit geval is alleen de klasse gemaakt voor de MPU6000, aangezien dit de sensor is die in het systeem gebruikt gaat worden. Deze klasse biedt de mogelijkheid om via SPI achter elkaar de gegevens uit de acceleratiesensor, temperatuursensor en gyroscoop uit te lezen en deze in het object, in een struct, op te slaan. Dit is sneller dan wanneer de sensor iedere keer opnieuw een variabele uit de sensor moet lezen. De MPU6000 leest zelf met een snelheid van 1 kHz de sensor waarden uit en plaatst ze in zijn interne registers, waarna de MPU6000 klasse alleen nog maar de juiste registers hoeft uit te lezen. Voor deze klasse is gebruik gemaakt van een MPU6000 library die op het internet is gevonden en is omgebouwd tot gebruik voor dit systeem. Deze library is te vinden op: http://code.google.com/p/gluonpilot/source/browse/trunk/Firmware/lib/mpu6000/?r=146 (laatst geraadpleegd: 5 maart 2014)

Naast een instantie van een IMU, bevat de Controller ook een instantie van een MotorController. Deze MotorController bevat op zijn beurt weer altijd twee instanties van Motoren: een roll motor en een pitch motor. In het diagram kon niet worden weergegeven dat de MotorController twee MotorInt objecten bevat, dus is met een asterisk weergegeven dat het er meer dan één is. Via de MotorController is het mogelijk om de motoren aan te maken en te configureren. Daarnaast kunnen de twee motoren worden bediend. Hierbij is te denken aan het uitzetten van de motoren of de motoren naar een bepaalde positie bewegen. Voor de berekening waarin de motor moet komen te staan, wordt gebruik gemaakt van een PIDcontroller. Tot slot kunnen de motoren worden getest.

De koppeling tussen de MotorController en de twee Motoren verloopt, net als bij de sensoren, via een Motor interface. Op deze manier kunnen ook andere type motoren gemakkelijk aan het systeem worden toegevoegd.

Op deze manier is alle functionaliteit uit het oude systeem overgezet wat te maken heeft met het stabiliseren van de camera. Tijdens de constructiefase worden alle functies overgezet en kan meteen worden gecontroleerd of alle benodigde functies daadwerkelijk in het diagram zijn opgenomen.

7.2.3 Globale werking nieuwe systeem

Globaal is het nieuwe systeem op te delen in twee verschillende momenten: het eerste moment is het moment waarop het systeem zichzelf opstart en gereed maakt voor gebruik; het tweede moment is wanneer het systeem actief is en door middel van een timer de sensor uitleest en de motoren aanstuurt. Hieronder staat per moment uitgelegd wat het systeem voor handelingen gaat uitvoeren.

Bij het opstarten van het systeem wordt eerst de verbinding aangelegd met de IMU en de sensor. Van deze objecten worden alle gegevens goed gezet, zoals het bepalen van de gyroscoopoffset, low pass filter et cetera. Wanneer dit is gedaan wordt hetzelfde gedaan met de motoren, waarbij onder andere de juiste frequentie en resolutie worden gezet. Daarnaast wordt per motor een array aangemaakt met sinus berekeningen. Deze array wordt later weer gebruikt om de motor in een bepaalde positie te zetten. Op deze manier hoeft het systeem niet meer tijdens het stabiliseren iedere keer een sinus berekening uit te voeren.

In de main wordt door middel van een timer de loop() functie van de Controller iedere milliseconde aangeroepen. In het begin bevindt het systeem zich in de IDLE state. In deze staat zorgt de loop() functie ervoor dat de PID controller wordt geconfigureerd. Daarna komst het systeem in de UNLOCKED state en krijgen de motoren de tijd om zich te positioneren. Tot slot komt het systeem in de LOCKED state en houdt het zich bezig met het stabiliseren van de camera. Op dat moment bevat de loop() functie eigenlijk twee stappen. Als eerste stap worden via de IMU alle gegevens uit de sensor uitgelezen, waarna de IMU deze gegevens omzet naar hoekberekeningen en dergelijke. In de tweede stap worden deze hoekberekeningen doorgestuurd naar de motoren die daardoor in de juiste stand komen te staan.

7.2.4 Het real-time aspect

Aangezien het bij het ontwikkelen van dit systeem gaat om een real-time systeem, is ook gekeken naar bepaalde tijdgaranties die het systeem moet nakomen en de vormgeving van het systeem, lettend op deze real-time aspecten.

Op dit moment kent het systeem maar één hoofdtaak: het stabiliseren van een camera. Daarom is het niet nodig om een geheel embedded operating system op de Teensy te installeren. De processen die door de Teensy worden uitgevoerd volgen elkaar in chronologische volgorde op en herhalen zichzelf daarna weer. Hierdoor komt het niet voor dat verschillende processen gescheduled hoeven te worden en kan het systeem zonder de hulp van een real-time operating system worden ontwikkeld.

Wel moet er gelet worden op de duur van bepaalde processen. Het moet namelijk mogelijk blijven om met een snelheid van 1 kH de sensor uit te kunnen lezen en deze gegevens te bewerken voor de motor aansturing. Dit houdt in dat deze loop in totaal maar 1 milliseconde mag duren. (1 seconde / 1000 keer uitlezen = 1 milliseconde) Om dit voor elkaar te krijgen moet gekeken worden naar een aantal bottlenecks.

De eerste bottleneck is het verkrijgen van de informatie uit de sensor. Om deze tijd zo klein mogelijk te maken, wordt alle informatie uit de sensor in een keer uitgelezen en in het sensor object opgeslagen. Dit is sneller dan de gegevens één voor één uit te lezen. Wanneer er namelijk één gegeven uit de MPU moet worden gelezen, moet eerst dit register adres naar de MPU worden geschreven. Vervolgens moet een byte met de waarde 0 naar de MPU worden geschreven, om aan te geven dat dit register moet worden uitgelezen. De MPU biedt daarnaast de mogelijkheid, dat wanneer er vervolgens nog een 0 wordt weggeschreven, hij automatisch de waarde in het volgende register mee terug geeft. Aangezien alle registers die sensor waarden bevatten na elkaar liggen, is het sneller om alle registers in een keer uit te lezen en de gegevens in het sensor object op te slaan.

Een tweede bottleneck is het uitvoeren van complexe of veel voorkomende berekeningen. Hier en daar is in het framework geprobeerd om deze berekeningen al van tevoren uit te voeren. Zoals eerder al is beschreven wordt bij het aanmaken van een motor object een array gemaakt en gevuld met arcsinus berekeningen. Op deze manier hoeft er tijdens het uitvoeren van het programma niet telkens de arcsinus worden uitgerekend van een bepaalde waarde, maar kan de juiste waarde uit deze array worden gehaald.

Daarnaast wordt in het framewrok voor kleine en veelvoorkomende functies gebruik gemaakt van inline functies. Dit bespaart weer de tijd die anders verloren zou gaan om naar die plek in het geheugen te springen. Al met al moet tijdens het omzetten worden gekeken in hoeverre deze strategieën toe te passen zijn, om ervoor te zorgen dat er zo min mogelijk tijd verloren gaat.

De verwachting is dat dankzij deze maatregelen de functionele eis moet worden behaald. Deze eis stelt dat de sensor met een snelheid van 1kH wordt uitgelezen. Bij de Arduino ging dit al goed en aangezien de Teensy nog krachtiger en sneller is wordt ervan uitgegaan dat alle berekeningen binnen de milliseconde worden uitgevoerd. Tijdens de constructiefase kan dit pas echt worden getest. Wanneer het op dat moment blijkt dat de berekeningen toch meer dan een milliseconde duren, moet worden gekeken waar er nog extra tijd gewonnen kan worden.

Al deze informatie is opgenomen in het Technische ontwerp dat terug is te vinden in bijlage G.

Aan het einde van deze fase was er drie weken ingelopen op de planning. Omdat de opdrachtgever er mee kwam dat hij tijdens dit project al een aantal extra uitbreidingen aan het systeem wilde toevoegen, is ervoor gekozen de bouw/ontwikkel fase naar voren te schuiven en op te rekken. Op deze manier kunnen de uitbreidingen in die fase worden gemaakt. Welke uitbreidingen er precies worden doorgevoerd en in welke volgorde, moet tegen die tijd met de opdrachtgever worden overlegd. Op deze manier zal de ontwikkel fase meer weg krijgen van een RUP ontwikkelmethode, waarbij een soort van iteraties worden gemaakt. In deze korte iteraties worden wel de fasen van het waterval model aangehouden. Zoals de definitiestudie/analyse fase, Basis en technische ontwerp fase en bouw/ontwikkel fase. Deze fasen zullen wat korter zijn, omdat het telkens gaat om een kleine aanvulling, maar waarvan van tevoren al wel goed duidelijk is wat deze moeten doen.

8. Bouw/implementatie

In dit hoofdstuk worden de activiteiten beschreven die zijn uitgevoerd tijdens de bouw/implementatie fase. De eerste week van deze fase is gewerkt aan het overzetten van het oude systeem naar de Teensy. Hierbij zijn allereerst alle functies uit het oude systeem letterlijk overgezet in de bijbehorende klassen, om het systeem op die manier werkend te krijgen. In de volgende stap is het systeem vervolgens geoptimaliseerd. Hierbij is vooral rekening gehouden dat de code begrijpelijk moet zijn.

8.1 Overzetten van het huidige systeem

De eerste stap in deze fase was het overzetten van het oude systeem naar de Teensy. Hiervoor is de oude code vaak letterlijk overgezet naar de nieuwe situatie met het klassendiagram als basis.

8.1.1 De BaseClass

Allereerst is de BaseClass voor het systeem ontwikkeld. Dit is een functionaliteit dat nog niet in het oude systeem zat. In het oude systeem werd er wel informatie in de terminal geprint, maar dit moest wat meer gestructureerd worden. Op deze manier wordt het gemakkelijker om bepaalde debug informatie wel of niet te printen. Tijdens de ontwikkeling is ervoor gekozen om gebruik te maken van template functies. Op deze manier kan dezelfde debug functie worden gebruikt, om verschillende datatypen via de terminal te laten printen. Deze constructie werd bedacht tijdens de constructie fase, waardoor deze functies nog niet in het klassendiagram waren opgenomen. Voor de rest is zoals geplant een debug_modus variabele toegevoegd om aan te geven in welke debug modus een bepaald object zich bevind. Hierbij is gebruik gemaakt van een enumeratie die op dit moment vier verschillende waarden kent: DB_OFF, DB_ERROR, DB_WARNING en DB_MESSAGE. In tabel 7.1 is te zien welke type berichten er worden geprint bij de verschillende standen.

Debug modus	Berichten die worden geprint
DB_OFF	Er worden geen debug berichten geprint
DB_ERROR	Alleen berichten met de DB_ERROR waarde worden geprint.
DB_WARNING	Berichten met de DB_ERROR of DB_WARNING waarde worden geprint.
DB_MESSAGE	Berichten met de DB_ERROR, DB_WARNING of DB_MESSAGE waarde worden geprint.

Tabel 8.1: De verschillende debug standen

De template functie is te zien in code fragment 8.1.

```
template<typename T> void debugOut(debug_modussen modus, T output)
{
	if (debug_modus >= modus)
	{
		//Wacht totdat er een verbinding is met de seriële port
		while(!Serial){}
		switch (modus)
		{
			case DB_MESSAGE:
				Serial.print("DEBUG MESSAGE: ");
			break;
			case DB_WARNING:
				Serial.print("DEBUG WARNING: ");
			break;
			case DB_ERROR:
				Serial.print("DEBUG ERROR:   ");
			break;
			default:
				Serial.print("DEBUG:         ");
			break;
		}
		printMessage(output);
	}
}
```

Code fragment 8.1: De template functie debugOut()

Om meer structuur te hebben tijdens het overzetten van het systeem, is toen de BaseClass was gemaakt eerst de structuur van het gehele framework aangemaakt. Hierbij zijn alle klassen aangemaakt en op de juiste manier met elkaar verbonden. Op deze manier was het gemakkelijker om in de volgende stappen de verschillende functies uit het oude systeem over te zetten.

8.1.2 Het sensor gedeelte

Vervolgens is begonnen met het invullen van het sensor gedeelte van het systeem. Allereerst is de klasse van de MPU6000 ingevuld. Deze is gebaseerd op een klasse die op internet is gevonden. Om deze library te gebruiken zijn een paar kleine veranderingen aangebracht in de pin aansturing. Daarnaast zijn is een paar kleine functies toegevoegd aan de klasse, omdat die wel door het stabilisatie systeem gebruikt worden, maar nog niet in de klassen verwerkt waren. Hierbij moet worden gedacht aan functies om de sensor waarden uit de sensor klasse op te vragen en een functie om de Low Pass Filter van de sensor te kunnen zetten. Tot slot zijn de public functies ook toegevoegd aan de SensorInt (interface) klasse.

Vervolgens is de IMU naar het nieuwe systeem overgezet. Het grootste probleem hierbij was om alle kleine bewerkingen die in het oude systeem werden gedaan op de juiste manier onder te brengen in de klasse. Omdat er in het oude systeem geen sprake was van een klassen indeling, konden alle verschillende onderdelen binnen het systeem gebruik maken van alle bestaande functies en variabelen. Daardoor was het een heel gepuzzel om uit te zoeken welke functies alleen gebruikt worden door de IMU. Daarnaast zijn er wederom een paar kleine functies bijgeschreven om private data members te kunnen uitlezen en wijzigen. Ook zijn op een paar plaatsen data members veranderd van een integer type in een float, bijvoorbeeld bij de hoekberekening van de roll en ptich as. Deze waren in het oude systeem namelijk met 1000 vermenigvuldigd en opgeslagen in een integer. Dit was gedaan omdat de Arduino waar het systeem oorspronkelijk op draaide minder gemakkelijk om kon gaan met floating point berekeningen. Met de Teensy is dit geen probleem meer.

8.1.3 De Controller

Nadat de IMU en de sensor klasse af waren, is verder gegaan met de Controller. De setup() en de loop() functie van de oorspronkelijke Arduino code zijn globaal overgezet naar de Controller. Een aantal bewerkingen die in de setup() functie van de Arduino code werden gedaan zijn bijvoorbeeld onder gebracht in de constructor van de IMU of de Sensor objecten. Door middel van het Controller object, werd het mogelijk om de eerste testen uit te voeren, waarbij is gekeken of de sensor kon worden uitgelezen en of vervolgens de hoek berekeningen goed gingen. Hierbij moest ervoor worden gezorgd dat de Controller over een loop functie beschikt, die 1000 keer per seconden kan worden aangeroepen. Dit komt voort uit de functionele eis die stelt dat de sensor met een frequentie van 1kH moet worden uitgelezen, waarna deze gegevens worden gebruikt om de camera te stabiliseren. De loop functie uit de Controller zorgt ervoor dat de overige objecten de juiste taken uitvoeren om dit voor elkaar te krijgen.

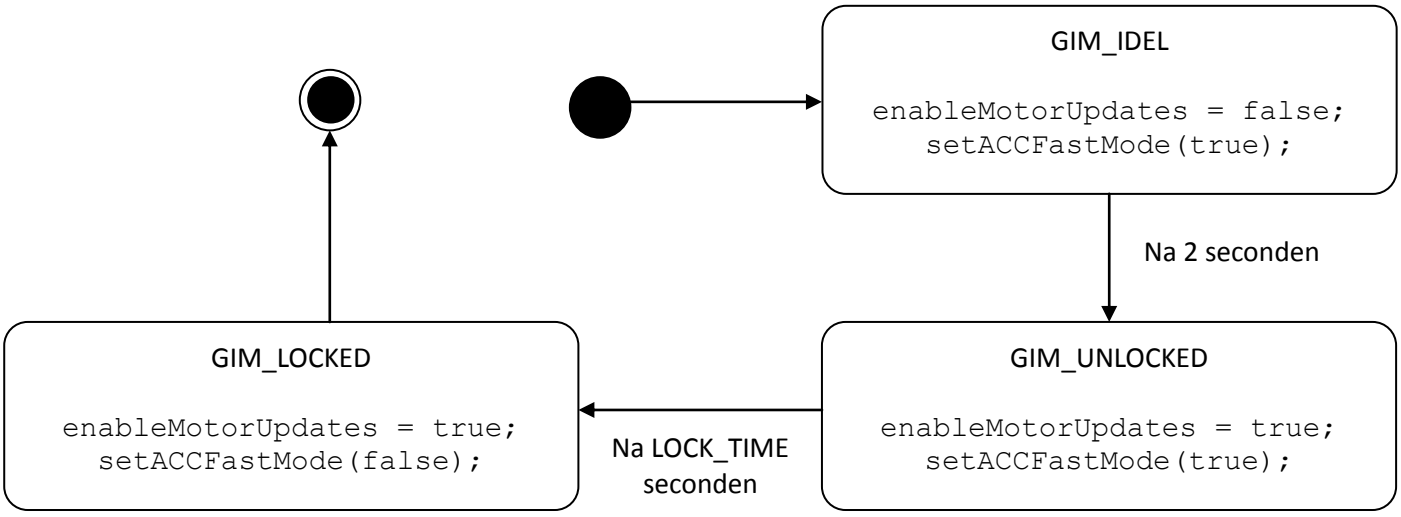
In eerste instantie was er een functie gevonden die dit mogelijk maakte. De IntervalTimer die standaard in de Arduino IDE zit. Hierbij kan worden opgegeven om de hoeveel microseconden een bepaalde functie moet worden aangeroepen, maar deze functie kan niet overweg met member functies van objecten. Het is dus niet mogelijk om rechtstreeks de loop functie van de Controller aan te roepen. Om dit op te lossen bevat de loop functie van de Controller een grote while() loop. Deze loop checkt één variabele, waardoor het mogelijk is om de controller loop ook weer te stoppen. Binnen deze loop wordt vervolgens via een if-statement een andere variabele gecheckt. Deze variabele wordt 1000 keer per seconde op true gezet, waardoor alle code binnen het if-statement 1000 keer per seconde wordt uitgevoerd. Het setten van deze variabele gebeurt nog steeds door de IntervalTimer, maar in de main.cpp file. Dit is grofweg dezelfde methode zoals deze ook in het oude systeem werd gebruikt. Bij de optimalisatie moet worden gekeken of hier een betere methode voor te vinden is. In codefragment 8.2 is een gedeelte de code met de IntervalTimer opgenomen.

```
IntervalTimer loopTimer;
Controller *c;

extern "C" int main(void)
{
	c = new Controller(DB_OFF);
	startLoop(LOOP_TIMER);
}

void startLoop(int interval)
{
	loopTimer.begin(loop, interval);
}

void stopLoop()
{
	loopTimer.end();
}

void loop()
{
	c->enableLoop();
}
```

Codefragment 8.2: De IntervalTimer

De loop functie bevat een groot switch statement met 10 cases. Deze checkt op een variabele die iedere loop wordt opgehoogd, waardoor de code binnen een loop effectief maar 100 keer per seconde wordt uitgevoerd. In deze loop worden in de eerste drie cases de drie acceleratie assen uitgelezen om in de vierde case de hoeken van de roll en pitch as via deze waarden weer gelijk te zetten. In de overige gevallen berekent het systeem de hoek van de assen door middel van de gyroscoop waarden. Hierbij worden 1000 keer per seconde deze waarden uitgelezen, waarna via een vector berekening wordt gekeken hoeveel de hoek moet zijn veranderd ten opzichte van de vorige meting. Deze meting is niet heel precies en wordt daarom 100 keer per seconde gelijk getrokken. Alleen de acceleratie sensor gebruiken om de camera te stabiliseren kan echter ook niet, want wanneer het systeem in die situatie een snelle beweging vooruit maakt, maakt het systeem vanwege de snel veranderende waarde van de acceleratie sensor een verkeerde hoekberekening.

In dit switchstatement zit ook een vorm van State controller opgenomen, zoals eerder in paragaraaf 7.2.3 is behandeld. In het begin bevindt het systeem zich in de IDLE state. In deze staat zorgt de loop() functie ervoor dat de PID controller wordt geconfigureerd. Daarna komst het systeem in de UNLOCKED state en krijgen de motoren de tijd om zich te positioneren. Tot slot komt het systeem in de LOCKED state en houdt het zich bezig met het stabiliseren van de camera. Dit is op dit moment letterlijk uit het oude systeem overgenomen. In een later stadium kan nog worden bekeken of hier een nettere manier voor moet worden gevonden. In afbeelding 8.1 is ter verduidelijking een statediagram te zien van deze statewisseling.



Afbeelding 8.1: Gimbal state diagram

In codefragment 8.3 is vervolgens te zien hoe het switch statement dat zorgt voor het uitlezen van de acceleratie sensor en het wisselen tussen de states.

```
switch (count)
{
	case 1:
		I->readACC(ROLL);
	break;

	case 2:
		I->readACC(PITCH);
	break;

	case 3:
		I->readACC(YAW);
	break;

	case 4:
		I->updateACC();
	break;

	case 5:
		//niets
	break;

	case 6:

		// gimbal state transitions
		switch (gimState)
		{
			case GIM_IDLE :
				stateCount++;
				if (stateCount >= LOOPUPDATE_FREQ/10*2)
				{
					gimState = GIM_UNLOCKED;
					stateCount = 0;
				}
			break;

			case GIM_UNLOCKED :
				stateCount++;
				if (stateCount >= LOOPUPDATE_FREQ/10*LOCK_TIME_SEC)
				{
					gimState = GIM_LOCKED;
					stateCount = 0;
				}
				break;

			case GIM_LOCKED :
			break;
		}

		// gimbal state actions
		switch (gimState)
		{

			case GIM_IDLE :
				enableMotorUpdates = false;
				setACCFastMode(true);
			break;
```

```
                  case GIM_UNLOCKED :
                        enableMotorUpdates = true;
                        setACCFastMode(true);
                  break;

                  case GIM_LOCKED :
                        enableMotorUpdates = true;
                        setACCFastMode(false);
                  break;
            }
      break;

      case 7:
            //niets
      break;

      case 8:
            //niets
      break;
      case 9:
            //niets
      break;
      case 10:
            count=0;
      break;
      default:
      break;
}
count++;
```

Codefragment 8.3: Het switch statement uit de loop() functie van de Controller

8.1.4 Het motor gedeelte

Tot slot is het motor gedeelte van het systeem overgezet. Het meeste werk zat in het feit dat er in het oude systeem voor alle motor variabelen twee varianten bestonden, aangezien er twee motoren worden gebruikt. In het nieuwe systeem wordt alles onderverdeeld in een motor klasse, waardoor ieder motor object zijn eigen variabelen beheert. Daarnaast waren er een aantal functies die heel veel parameters meekregen, waaronder de PIDController. In het nieuwe systeem is dit niet meer het geval, ook vanwege het feit dat iedere motor zijn eigen variabelen beheert. Hierdoor is het systeem direct een stuk overzichtelijker.

Als tussenliggende laag is de MotorController klasse ontwikkeld. Deze klasse bestond nog niet in het oude systeem. Met behulp van de MotorController zijn de motoren veel simpeler te gebruiken in de Controller klasse en zitten alle ingewikkelde berekeningen en methoden weggestopt in de MotorController en Motor klassen. Dit draagt op zijn beurt ook weer bij aan het gemakkelijker gebruiken van het systeem.

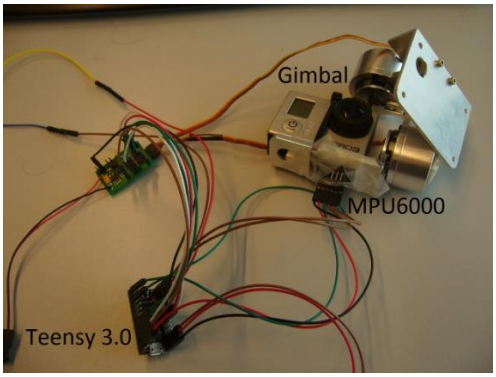
8.2 Optimaliseren

Tijdens het overzetten van het systeem is steeds in de gaten gehouden hoe het zat met de uitvoeringstijd van de code, zodat de sensor met een snelheid van 1 kHz uit kan worden gelezen en deze gegevens verwerken. Dit is gedaan door middel van de elapsedMicroseconds klasse die standaard bij de Arduino IDE wordt geboden. Deze klasse houdt bij hoeveel microseconden er voorbij zijn sinds de start van het programma. Hiermee is te berekenen hoelang tijd een bepaald stuk code duurt, door de tijd aan het begin van de code af te trekken van de tijd aan het eind van de code. Hieruit bleek dat de code op dit moment ruimschoots binnen de maximale tijd valt die het mag duren. De maximale uitvoeringstijd van de loop() functie kwam neer op ongeveer 450 us. In codefragment 8.4 is de code te zien waarmee de tijdsduur van de functies zijn opgemeten.

```
elapsedMicros teller;
int tijd;

tijd = teller;

// De te meten functies

tijd = teller - tijd;

Serial.print("Deze functie duurde: ");
Serial.print(tijd);
Serial.println(" us");
```

Codefragment 8.4: Tijdsduur meten

Voor de rest is nog niet heel specifiek gekeken naar de optimalisatie van de werking van de code. Hiervoor is nog niet de juiste hardware beschikbaar. Op dit moment is de opdrachtgever bezig met de ontwikkeling van een printplaatje die op de Teensy aansluit. Op dit printplaatje worden vervolgens de motoren en sensoren aangesloten. Deze zitten nu nog met stugge prototype draadjes aan de Teensy verbonden, waardoor ze soms de stabilisatie tegenwerken. Dit is te zien in afbeelding 8.2. De MPU is aan de onderkant van de gimbal bevestigd en met stugge prototype draadjes aan de Teensy verbonden. Wanneer alle hardware binnen is, kan hier de focus op worden gelegd en kan vervolgens een uitgebreid onderzoek worden gedaan naar de uitvoeringstijd van de verschillende functies binnen het systeem.



Afbeelding 8.2: Aangesloten componenten

Wel is bij de optimalisatie veel aandacht besteed aan de duidelijkheid van het systeem Hierbij is te denken aan de documentatie van systeem, zoals klassendiagrammen en beschrijvingen. Deze documentatie is eerder in dit verslag al aan de orde gekomen, tijdens de ontwerp fasen. Maar daarnaast is iedere functie gedocumenteerd, waardoor duidelijk is welke paramaters er moeten worden meegegeven, welke waarde de functie retourneert, wat voor de rest de werking is van de functie en hoe lang deze duurt. Dit is gedaan in de stijl van het programma Doxygen, zodat met dit programma uiteindelijk een hele wiki kan worden gebouwd. In code fragment 8.5 is te zien op welke manier de private functie spi_write_reg(uint8_t addr, uint8_t data) uit de MPU6000 klasse is gedocumenteerd en in afbeelding 8.3 is te zien welke opmaakt doxygen hieraan geeft.

```
/*************************************************************************//*!
      Duur: 5 us

      Deze functie schrijft een bepaalde waarde naar een aangegeven register van
      de MPU6000.
      @param addr Het adres van het register waar naartoe geschreven moet worden
      @param data De waarde die naar het register geschreven moet worden
*************************************************************************/
void MPU6000::spi_write_reg(uint8_t addr, uint8_t data)
{
    spi_cs_disable();
    spi_cs_enable();

    spi_comm_bitbang(addr);
    spi_comm_bitbang(data);

    spi_cs_disable();
}
```

Codefragment 8.5: documentatie van de functie functie spi_write_reg(uint8_t addr, uint8_t data) uit de MPU6000 klasse

void MPU6000::spi_write_reg (uint8_t addr,
uint8_t data
) [private]

Duur: 5 us

Deze functie schrijft een bepaalde waarde naar een aangegeven register van de **MPU6000**.

Parameters:

addr Het adress van het register waar naartoe geschreven moet worden

data De waarde die naar het register geschreven moet worden

Afbeelding 8.3: Doxygen voorbeeld van de functie functie spi_write_reg(uint8_t addr, uint8_t data) uit de MPU6000 klasse

9 Toevoegen van extra functionaliteit

Aan het begin van het project was de verwachting dat de opdrachtgever zoveel mogelijk wilde doen aan de optimalisatie van het systeem. Tijdens het project bleek echter dat de opdracht sneller klaar was dan geplant en heeft de opdrachtgever besloten om een aantal uitbreidingen op het systeem te maken die hij anders later zelf zou maken. De opdrachtgever heeft ervoor gekozen om dit zelf in fasen bekend te maken en uit te leggen, om zo te zorgen voor wat minder werkdruk. Dit was aan het begin van het project dus nog niet duidelijk, waardoor deze uitbreidingen niet zijn meegenomen in de keuze van de ontwikkelmethode. Het project was al dermate ver gevorderd dat het ook niet meer mogelijk was om nog compleet van ontwikkelmethode te veranderen. Om toch binnen de ontwikkelmethode ruimte te maken om deze uitbreidingen door te voeren is ervoor gekozen een aantal kleine iteraties te maken, waarin per uitbreiding alle fasen van de waterval methode worden doorlopen. Bij iedere uitbreiding is eerst een ontwerp van het systeem gemaakt, waarbij is gekeken naar de requirements van de uitbreiding, de libraries die daarvoor konden worden gebruikt en hoe het klasse diagram moest worden uitgebreid/aangepast. Tot slot is gekeken hoe deze uitbreiding uiteindelijk moet worden getest. Deze ontwerpen zijn apart per uitbreiding in een document gezet en vormen daarbij een aanvulling op de al bestaande requirements document en overige ontwerp documenten. Op deze manier sluiten de uitbreidingen toch aan bij het werk wat al was gedaan. Bij het doorvoeren van de uitbreidingen is altijd besproken met de opdrachtgever welke uitbreiding de meeste prioriteit had en wat de wensen/eisen waren bij deze uitbreiding. Wanneer deze voorbereidingen klaar waren, zijn de uitbreiding daadwerkelijk gebouwd.

9.1 Variabelen aanpassen

In de eerste uitbreiding moet het mogelijk zijn om extern een aantal variabelen van het systeem aan te passen, die de werking van het systeem bepalen. Hierbij moet worden gedacht aan het aanpassen van de waarden die bij de PID Controller worden gebruikt, de pinnen waarop de motoren worden aangesloten, et cetera. Op deze manier is het systeem voor alle afzonderlijke camera-stabilisatie-systemen te optimaliseren.

In het oorspronkelijke systeem zat al een dergelijke functionaliteit. Deze is als basis gebruikt voor de uitbreiding. Hierbij worden de variabelen die aanpasbaar moeten zijn, gegroepeerd in een struct. Deze struct wordt vervolgens naar de EEPROM weggeschreven. De struct bevat een speciale variabele die bijhoudt welke versie van de struct er wordt gebruikt. Hierdoor is het mogelijk de structuur van de struct te wijzigen en naar de EEPROM te schrijven, zonder dat een systeem wat nog niet is aangepast op deze verandering, helemaal de mist in loopt. Op het moment dat de struct uit de EEPROM is gelezen, wordt gecontroleerd of de versie van de EEPROM struct overeenkomt met de versie van de struct uit het systeem. Wanneer deze waarden niet overeenkomen, wordt er een functie aangeroepen die standaard waarden in de struct plaatst. Hierdoor zal het programma toch werken, alleen niet optimaal. Om dit te kunnen garanderen moet de eerste waarde uit de struct altijd de struct versie bevatten, ook wanneer de struct wordt uitgebreid/aangepast. Wanneer dit niet gebeurt kan het systeem op dit punt fouten maken.

In eerste instantie was het voorstel om een command-line user interface te maken die het gemakkelijk maakte om de variabelen aan te passen. De opdrachtgever vond dat niet perse noodzakelijk. Voor hem was het voldoende om een los programmaatje te schrijven, waar de variabelen in de code kunnen worden veranderd, met de mogelijkheid ze vervolgens in de EEPROM weg te schrijven. Dit programma moet vervolgens worden gecompileerd en op de Teensy worden gezet, waarna de struct in de EEPROM wordt weg geschreven. De gebruikers van het systeem zijn zelf ook technisch onderlegd, waardoor dit voor hun geen probleem is. Op deze manier hoeft niet heel veel tijd worden besteed aan het maken van de user interface, maar kan de focus liggen op verdere uitbreidingen.

Voor het wegschrijven van de gegevens in de EEPROM wordt gebruik gemaakt van de EEPROM library die standaard wordt meegeleverd bij de Arduino IDE die ook geschikt is voor de Teensy. Daarnaast is het systeem uitgebreid met een Config klasse. Deze klasse is een Singleton en zorgt voor het beheer van de struct met variabelen. Op deze manier kunnen alle klassen in het framework een bepaalde variabele opvragen uit de struct.

Voordat dit plan is uitgevoerd, is een ontwerp document gemaakt voor het systeem. Hierin zijn de requirements, ontwerp en aanvulling op het testplan opgenomen. Daarnaast zijn er voor deze uitbreiding een aantal testscenario's gemaakt. Het ontwerp document over het aanpassen van de variabelen is terug te vinden in bijlage H.

9.2 Pitch hoek aanpassen

De volgende uitbreiding op het systeem was om de gebruiker de mogelijkheid te geven om de pitch hoek van de camera aan te passen. Op deze manier kan de gebruiker zelf de camera omhoog of omlaag bewegen, waarna de camera in die hoek wordt gestabiliseerd. De eis van de opdrachtgever was om een hoek tussen de +45° en -135° te mogelijk te maken. Deze gegevens worden opgeslagen in een 12 bits getal en via een seriële verbinding vanaf de avionica van de helikopter doorgestuurd naar de Teensy. Hierbij staat 0 voor de hoek -135° en 4095 (2^{12-1}) voor de hoek +45°.

Om deze gegevens te kunnen ontvangen is de loop van de Controller uitgebreid, door daar te checken of er een waarde is binnengekomen op de seriële poort. Wanneer dit zo is, wordt er een functie aangeroepen die deze waarde uitleest en omzet naar data. De structuur van de byte is als volgt gedefinieerd:

- De eerste 2 bits geven aan of het om de MS6B of de LS6B gaat
    - 11 – Most significant 6 bits
    - 00 – Least significant 6 bits
- De laatste 6 bits zijn respectievelijk de MS6B of LS6B van de 12 bits waarde.

Wanneer de LS6B worden ontvangen, worden deze in een variabele opgeslagen. Als de volgende keer de MS6B worden ontvangen, wordt samen met de LS6B de 12 bits waarde geconstrueerd. Hierdoor moet altijd eerst de LS6B en daarna de MS6B worden gestuurd.

Om deze verandering door te voeren, is in de Motor een extra variabele opgenomen die de ingestelde hoek van de gebruiker bevat. Deze waarde wordt vervolgens meegenomen als de hoek waar de motor in moet komen te staan wordt berekend. Deze 12 bits codering van de hoekaanpassing is hetzelfde als het oude camera-stabilisatie-systeem. Hierdoor hoeft er op dit moment nog niets te worden aangepast in de software van de helikopter. Vervolgens is aan het einde van de loop() functie van de Controller een while loop die iedere keer kijkt of er informatie is doorgestuurd via de seriele poort. Wanneer dit het geval is wordt de functie readAvionicaInput() van de Controller aangeroepen, die de seriële poort uitleest en vervolgens de juiste handelingen uitvoert. In codefragment 9.1 is heel globaal de code opgenomen die zorgt draagt voor het uitlezen en verwerken van de seriële verbinding.

```
// In de loop() van de Controller
while(AVIONICA_SERIAL.available())
{
	readAvionicaInput();
}


void Controller::readAvionicaInput()
{
	static unsigned char ls6b_in = 0, ms6b_in, ls6b = 0, ms6b;
	int t12b = 0;
	unsigned char c = AVIONICA_SERIAL.read();

	// Byte met waarde 00------
	// LS6B van de pitch input
	if ((c & 0xC0) == 0x00)
	{
		ls6b_in = c;
		ls6b = (c & 0x3F);
	}

	// Byte met waarde 11------
	// MS6B van de pitch input
	if ((c & 0xC0) == 0xC0)
	{
		ms6b_in = c;
		ms6b = (c & 0x3F);
		t12b = ((ms6b << 6) + ls6b);

		//Bereken de hoek en stuur deze door naar de PITCH motor
	}
}
```

Codefragement 9.1: Het uitlezen en verwerken van de seriële verbinding

Voor deze uitbreiding zijn wederom een aantal testscenario's geschreven. Tijdens de ontwikkeling is al globaal getest of de uitbreiding werkt. Hierbij is gebruik gemaakt van de Arduino Mini. Voor deze Arduino is een klein programmaatje geschreven die om de 5 seconden een vooraf ingestelde hoek via de seriële verbinding naar de Teensy stuurt. Na het doen van een paar kleine aanpassingen reageerde het systeem als gewenst op de input vanaf de Arduino.

Net als bij de uitbreiding van de variabelen aanpassen, is er een ontwerp document gemaakt voor het systeem. Hierin zijn de requirements, ontwerp en aanvulling op het testplan opgenomen. Daarnaast zijn er voor deze uitbreiding een aantal testscenario's gemaakt. Het ontwerp document over het aanpassen van de pitch hoek is terug te vinden in bijlage I.

9.3 Basis camera aansturing

Een uitbreiding die nog niet in het oude systeem bestond, was de mogelijkheid om de camera, die aan de gimbal is bevestigd, aan te sturen. Dit is de volgende uitbreiding waarbij de opdrachtgever de gebruiker twee mogelijkheden wil bieden.

Als eerste moet de gebruiker de mogelijkheid krijgen om te schakelen tussen de verschillende videostromen. De camera die aan de gimbal wordt bevestigd bevat namelijk twee losse camera's in één, een infrarood camera en een daglicht camera. Deze twee camera's zijn samen in een behuizing gezet en aangesloten op een switch. Deze switch kan door middel van verschillende pwm signalen schakelen tussen de twee camera's.

Als tweede moest de gebruiker de mogelijkheid krijgen om de infrarood camera te kunnen kalibreren. Tijdens dit kalibreren moet de camera naar de achterkant van het systeem kijken, op de plek waar de motor zit, waardoor deze een oppervlakte filmt waar alles even warm is. Wanneer de camera juist staat gepositioneerd moet via een serieel protocol de DO_FFC (Flat Field Correction) modus van de camera worden geactiveerd.

Deze opdracht wordt aan de Teensy doorgegeven via de avionica, net als de opdracht tot het aanpassen van de pitch hoek van de camera. Omdat het wenselijk is niets aan de pitch aansturing in de avionica te hoeven aanpassen, is ervoor gekozen om de LS6B en MS6B van de pitch aanpassing nog steeds aan te duiden met 00 en 11 als eerste twee bits. De commando's voor de camera worden vervolgens aangeduid met 10 als eerste twee bits. Wanneer de Controller deze code doorkrijgt uit de seriële port, wordt het commando doorgestuurd naar de command() functie van het nieuw aangemaakte Camera object. Deze functie bevat een switch statement die bepaalt welke handeling er vervolgens verricht moet worden. Op dit moment kent de camera vier commando's:

- 0 – Selecteer de eerste video stroom
- 1 – Selecteer de tweede video stroom
- 2 – Selecteer de derde video stroom
- 3 – Kalibreer de infrarood camera.

Om vervolgens de video stroom te kunnen activeren moet er altijd een pwm signaal met een frequentie van 50 hz, of terwijl 20 ms, naar de switch worden gestuurd. Vervolgens bepaalt de duty cicle van het signaal welke video stroom wordt geselecteerd. Door een duty cicle te genereren van 1.1 ms, 1.5 ms of 1.9 ms, kan worden geswitcht tussen drie videostromen. Op dit moment worden er nog maar twee gebruikt, maar de opdrachtgever wil in de toekomst waarschijnlijk drie camera’s ondersteunen.

Toen deze functionaliteit was ingebouwd, kon de focus worden gelegd op het kalibreren van de infrarood camera. Hiervoor moest worden uitgezocht welk pwm signaal naar de motoren moest worden gestuurd om ze in de juiste stand vast te zetten. Hierbij wordt gebruik gemaakt van de waarden uit de sinus array van de motoren. Omdat er wordt gewerkt met brushless motoren, die gebruik maken van spoelen en magnetisme om de motor te positioneren, kunnen de motoren met de hand in een andere stand worden gezet wanneer ze stilstaan. Als eerste moest worden bepaald in hoeveel standen de motor kan staan bij het doorsturen van een pwm waarde. Hiervoor is een willekeurige waarde naar de motor gestuurd en is met de hand de motor in alle mogelijke standen gezet. Hieruit bleek dat de motor zeven verschillende standen heeft. Wanneer de motor in een volgende stand wordt geduwd, houdt dat in dat de motor een sprong maakt van 360°/7 ≈ 51,4°. In een volgende test bleek dat de waarden uit de sinus array bepaalden op welke positie binnen deze stap de motor kwam te staan. Wanneer de motor bij entry 0 uit de array op 0° zou staan, zou deze bij entry 126 op ongeveer 25,7° staan en bij entry 255 bijna op 51,4°.

Om er vervolgens voor te zorgen dat de camera naar de achterkent van het systeem komt te kijken tijdens het kalibreren, draait de camera eerst 180° achteruit. Hierbij wordt gelet op de waarde die de sensor meegeeft. Wanneer de sensor registreert dat de camera 180° achteruit kijkt, wordt berekend welke waarde er naar de motor moet worden gestuurd, om hem in een hoek van 180° ten opzichte van de helikopter vast te zetten. Hierbij zal de motor in de dichtstbijzijnde stand springen. Hetzelfde wordt gedaan bij de roll motor, die in een hoek van 0° ten opzichte van de helikopter moet komen te staan. Dit houdt in dat de helikopter zelf in een maximale hoek van 51,4°/2 = 25,7° kan vliegen tijdens het kalibreren. Wanneer dit meer is kan het zijn dat de motor net in de volgende stabiele stand springt en niet goed is uitgelijnd om te kalibreren. Volgens de opdrachtgever blijkt in de praktijk dat de helikopter een maximale hoek van 30° bereikt wanneer deze vliegt. Dit is eigenlijk net iets te veel. In overleg met de opdrachtgever is besloten dat dit niet erg is, aangezien de helikopter tijdens het kalibreren vrijwel altijd stil hangt. Wanneer het toch mocht mislukken is het niet erg om de kalibratie voor een tweede keer uit te voeren.

Toen het mogelijk was de stand van de motoren vast te zetten, moest worden gekeken naar de seriële communicatie met de infrarood camera. Deze camera beschikt zelf over een communicatie protocol wat is uitgelegd in tabel 9.1:

Byte nummer	Betekenis
1	Process code. Deze byte moet altijd de waarde 0x6E hebben.
2	Status byte. Wordt door de camera zelf gebruikt om de status terug te sturen en wordt genegeerd bij alle commando's die naar de camera worden toegestuurd.
3	Gereserveerd.
4	Code van de functie die moet worden uitgevoerd.
5	MSB van het aantal argument bytes die aan de functie worden doorgegeven.
6	LSB van het aantal argument bytes die aan de functie worden doorgegeven.
7	MSB van de CRC van de eerste 6 bytes. Alle CRC berekeningen zijn gebaseerd op de CCITT-16 (xmodus) CRC.
8	LSB van de CRC van de eerste 6 bytes.
9 t/m N-2	Alle argumenten die aan de functie moeten worden meegegeven.
N-1	MSB van de CRC van byte 0 tot en met N-2.
N	LSB van de CRC van byte 0 tot en met N-2.

Tabel 9.1: Het communicatie protocol van de infrarood camera

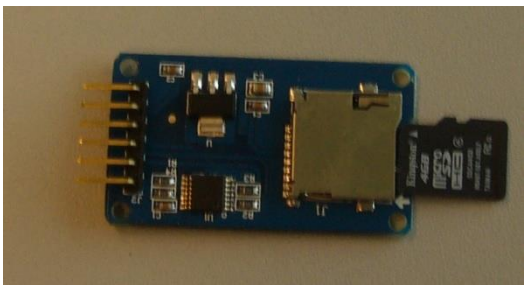
(Bron voor de protocol uitleg en functie beschrijvingen: “TauTM 2 / QuarkTM Software Interface Description Document (IDD) , May 20 2013 , Document Number: 102-PS242-43 , Version 120”)

Op basis van dit protocol is een functie ontwikkeld die drie parameters meekrijgt: een array die de eerste 4 bytes van het commando bevat, een byte die alle argumenten bevat die de camera moet meekrijgen en een integer die aangeeft hoeveel entries de laatste array bevat, aangezien deze array kan verschillen in grootte. In de functie wordt vervolgens weer gebruik gemaakt van een functie die op internet is gevonden om de CRC te berekenen. (http://www.nongnu.org/avr-libc/user-manual/group__util__crc.html laatst geraadpleegd: 7 mei 2014) Op deze manier kan de functie in de toekomst worden gebruikt om alle mogelijke functionaliteit van de camera aan te roepen.

Voorafgaand aan de constructie van deze uitbreiding is er een ontwerp document gemaakt, dat overeenkomt met de ontwerpdocumenten van de voorgaande uitbreidingen. Dit ontwerp document is terug te vinden in bijlage J. Daarnaast zijn er een aantal testscenario's gemaakt voor deze uitbreiding.

9.4 Logging mogelijkheid

De laatste uitbreiding die de opdrachtgever wilde doorvoeren, was de mogelijkheid om de hoeken van roll en pitch as te kunnen loggen op een SD kaart. Op deze manier wordt het gemakkelijker om in de toekomst het systeem te kunnen fine tunen, door de variabelen van de PID controller te kunnen setten en vervolgens te bekijken hoe goed het systeem erop reageert.



Afbeelding 9.1: De CATALEX Micro SD Card en 4GB klasse 4 Micro SD HC kaart

Op de officiële site van de Teensy staat een referentie naar een forum post die uitleg gaf over de mogelijkheid om te kunnen loggen op een SD kaart. (http://forum.pjrc.com/threads/16758-Teensy-3-MicroSD-guide laatst geraadpleegd: 6 mei 2014) Hierbij wordt er gebruikt gemaakt van een Micro SD Card Adapter die via SPI met de Teensy kan communiceren. Op de site van de Teensy stond er een aanbevolen, maar uiteindelijk

heeft de begeleider ervoor gekozen te kiezen voor de Micro SD Card Adapter van het bedrijf CATALEX, omdat hiervan de totale prijs wat goedkoper was. Daarnaast is er een 4GB klasse 4 Micro SD HC kaart van Kingston aangeschaft, omdat in de blog werd vermeld dat dit type kaart zou werken met de Adapter. Deze componenten zijn te zien in afbeelding 9.1.

Vervolgens kan via de SDFAT library (http://code.google.com/p/sdfatlib/downloads/detail?name=sdfatlib20131225.zip&can=2&q laatst geraadpleegd: 5 mei 2014) een file worden aangemaakt, worden bewerkt en vervolgens worden weggeschreven naar de Micro SD kaart.

Als eerste is geprobeerd de communicatie met de SD kaart aan de praat te krijgen, zonder nog te letten op de rest van het systeem. Hiervoor moest de SD kaart eerst worden geformatteerd en voorzien van een FAT16 of FAT 32 partitie tabel. Dit is geprobeerd met een aantal op Linux gebaseerde programma's, maar zonder resultaat. Uiteindelijk is het gelukt met het Windows programma SDFormatter.

Vervolgens kon er via SPI met de SD kaart worden gecommuniceerd. Daarvoor moest iedere keer de file worden geopend, iets worden weggeschreven en vervolgens de file weer worden gesloten voordat de informatie naar de SD kaart werd weggeschreven. Dit gehele proces duurde 4 ms, waardoor de maximale uitvoeringstijd van 1 ms die de loop mocht duren ruim werd overschreden. Vervolgens is met de opdrachtgever overlegd wat wenselijk was: of bij het opstarten van het systeem de file openen en aan het einde weer te sluiten, met het risico dat de gegevens verloren gaan op het moment dat het systeem vastloopt; of de loop functie van de Controller in een interrupt plaatsen en daarnaast de logging uit te voeren, waardoor de loop wel met een frequentie van 1000 Hz kan worden uitgevoerd, maar dat er minder vaak wordt gelogd. De opdrachtgever koos voor optie twee. Zodoende is de loop functie van de Controller in een interrupt geplaatst door de IntervalTimer, genoemd in paragraaf 8.1.3, in de main een functie te laten aanroepen, die op zijn beurt weer de loop functie van de Controller aanroept. Waarna de IntervalTimer is gestart, wordt in de main een while loop binnen gegaan die continu de gegevens van de roll en pitch as opvraagt en ze naar de SD kaart wegschrijft.

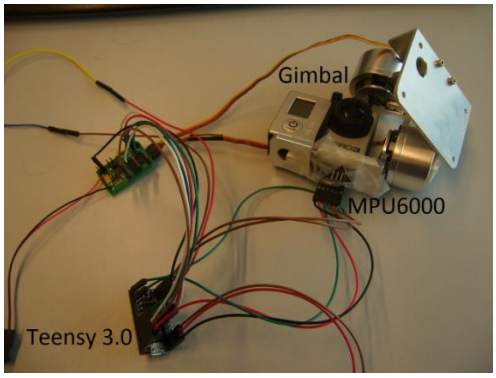
Toen vervolgens de mogelijkheid om te loggen werd toegevoegd aan het systeem, begon het systeem allemaal rare bewegingen te maken. Na veel gepuzzel bleek na onderzoek op internet dat de SDFAT library intern een aantal SPI registers aanpast, om met de SD kaart te kunnen communiceren, waardoor er naast de Micro SD Card Adapter geen extra SPI device kan worden gebruikt. In overleg met de opdrachtgever is besloten de logging mogelijkheid even links te laten liggen en goed te documenteren dat alles los werkt en hoe het werkt. Op deze manier kan in de toekomst worden gekeken of er misschien een extra embed aan de Teensy kan worden gekoppeld die bijvoorbeeld via een seriële verbinding gegevens naar de embed toestuurt, waarna de embed via SPI de gegevens naar de SD kaart wegschrijft.

Voorafgaand aan de constructie van deze uitbreiding is er een ontwerp document gemaakt, dat overeenkomt met de ontwerpdocumenten van de voorgaande uitbreidingen. Dit ontwerp document is terug te vinden in bijlage K. Daarnaast zijn er een aantal testscenario's gemaakt voor deze uitbreiding.

9.5 Afronding Bouw/Implementatie fase

Tot slot is in deze fase nog gewerkt aan de documentatie bij de code. Het was de bedoeling om het gehele systeem al te fine-tunen, maar daar wordt nog mee gewacht, omdat de juiste hardware er nog niet is. Deze hardware komt hopelijk in de Integratie fasen, waarna er nog extra aandacht kan worden besteed aan de fine-tuning.

10. Testfase en integratiefase



Afbeelding 10.1: De huidige gimbal

De test- en integratiefase liepen een beetje door elkaar. Dit kwam omdat de opdrachtgever nog bezig was met een nieuwe gimbal. De huidige versie is te zien in afbeelding 10.1. In deze afbeelding is te zien dat de MPU6000 hier nog aan de onderkant van de camera is geplakt. Uiteindelijk zou de Teensy bovenop worden geplakt, maar met het probleem dat de draadjes tussen de MPU en de Teensy buitenom zouden lopen. De opdrachtgever is ook niet meer bezig om een printplaatje voor op de Teensy te maken, maar is hij bezig aan een gimbal met andere motoren, waarbij de draden die de Teensy en de MPU verbinden, door de motor assen lopen. Daardoor leveren de draadjes minder weerstand tijdens het stabiliseren. Dit nieuwe systeem gaat draaien op een Teensy 3.1 in plaats van de Teensy 3.0 die nu wordt gebruikt. Deze nieuwe versie van de Teensy is uitgekomen tijdens de doorlooptijd van dit project. De begeleider heeft de keuze gemaakt om de Teensy 3.1 te gaan gebruiken, omdat deze iets meer capaciteit heeft dan de Teensy 3.0. De Teensy 3.1 heeft een hogere kloksnelheid, meer RAM en meer EEPROM. Daarnaast wordt gebruik gemaakt van een ander materiaal als printplaat, waarvan de ontwikkelaars van de Teensy zeggen dat het beter werkt. Tot slot is de Teensy 3.0 bijna niet meer verkrijgbaar, waardoor de keuze om over te stappen op de Teensy 3.1 ook gemakkelijker was.

Wanneer het systeem af is, moet worden gekeken of de omzetting van de Teensy 3.0 naar de Teensy 3.1 erg veel aanpassingen vergt. De voorspelling is dat dit weinig tot geen inspanning zal vergen, omdat de Teensy 3.1 alleen meer capaciteit zal hebben, maar voor de rest op dezelfde manier werkt. Wanneer de nieuwe gimbal af is en de code is overgezet naar de Teensy 3.1, kan verder worden gegaan met het uitvoeren van de Systeem tests en integratie tests.

In de tussentijd zijn wel de integratie testen uitgevoerd. Deze testen zijn bedoeld om te kijken of de verschillende objecten binnen het systeem goed met elkaar communiceren en op de juiste manier de sensor uitlezen en de motoren aansturen. Aangezien in het nieuwe systeem de aansturing tussen de verschillende componenten gelijk blijf,t konden deze testen gewoon worden uitgevoerd.

Afbeelding 10.2 de nieuwe gimbal

In de loop van deze fase kwam de opdrachtgever met de nieuwe versie van de gimbal, te zien in afbeelding 10.2. Het blok bovenaan is bedoeld om de gimbal aan de multicopter te bevestigen. Vervolgens bevinden zich in de armen van het systeem de Teensy en de aansluit draden naar de motoren en de twee camera's. Toen het nieuwe systeem klaar was, is eerst aandacht besteed aan het fine tunen van het systeem, zoals het goed zetten van de PID waarden en overige variabelen. Tijdens dit fine-tunen bleek dat in de PID controller niet werkt als een officiële PID controller. Bij een echte PID controller wordt er een vast staande variabele opgehoogd of verlaagd aan de hand van de meegegeven fout. Vervolgens zou in dit systeem de te veranderen variabele moeten worden doorgegeven aan de motor. Dit lijkt bijvoorbeeld op de volgende code:

```
int x =+ ComputePID(angle);
SetMotor(x);
```

Echter, in de code gebeurt globaal het volgende:

```
SetMotor(ComputePID(angle));
```

Dat de werking van de huidige code verschilt met de officiële werking van een PID controller werd opgemerkt door de opdrachtgever. Zelf heb ik dit nooit gezien, aangezien ik nog nooit met een PID controller heb gewerkt en dacht dat de PID controller goed was geïmplementeerd. Vervolgens is er aandacht aan besteed om de PID controller aan te passen, zodat deze op de nette manier functioneert. Dit nam erg veel tijd in beslag en het systeem ging allemaal rare bewegingen maken dankzij de veranderingen. Na een tijdje is in overleg met de opdrachtgever besloten om de huidige PID controller te behouden en in de aanbevelingen een stukje te schrijven dat de PID controller in de toekomst misschien kan worden verbeterd.

Afbeelding 10.3: De RH4 Spyder met gimbal

Nadat het systeem gefine-tuned was, kon het volledige systeem worden getest. Hiervoor zijn alle systeem testen uitgevoerd en is een begin gemaakt aan de acceptatie testen. Tijdens deze testen is het systeem onder de multicopter bevestigd. In afbeelding 10.3 is de RH4 Spyder te zien, waaraan de gimbal is bevestigd. Op deze manier is getest of de gimbal kon worden bestuurd door het grondstation, waar ook de multicopter mee wordt bestuurd. Helaas bleek dat op de multicopter nog een verouderde software versie draaide, waardoor niet alle functionaliteit kon worden getest. Via de terminal gtkterm zijn, met de zelfde instellingen als die van de seriële verbinding tussen de avionica en de gimbal, een aantal hexadecimale codes gestuurd die overeenkomen met de codes die de avionica zou moeten sturen. Op deze manier kon wel worden getest of het systeem op de juiste codes reageert, maar is dit helaas niet tijdens een vlucht getest.

Vervolgens heeft de Spyder een kleine testvlucht gemaakt met de camera eronder om te testen of de stabiliteit van het beeld voldoende is, zoals te zien in afbeelding 10.4. Bij deze test bleek dat wanneer de multicopter een beweging maakt, de camera nog een klein beetje mee gaat in deze beweging, maar dit was zeer minimaal. Dit kan worden verholpen door de PID waarden nog iets strakker af te stellen. De opdrachtgever was tevreden over het resultaat en gaf aan zelf in de toekomst nog even te kijken of de PID waarden nog strakker kunnen worden gezet, omdat de afstudeertijd er bijna op zat.

Afbeelding 10.4: Testvlucht

Op het moment van schijven heeft de opdrachtgever nog geen tijd gehad om de source-code en documentatie van het systeem grondig door te nemen, om op die manier te testen of de code onderhoudbaar is. Tijdens het ontwikkelen zijn er wel een aantal momenten geweest waarop de opdrachtgever even globaal hiernaar heeft gekeken, waarbij hij telkens tevreden was over de manier waarop de code was gestructureerd en gedocumenteerd.

Het document met testscenario's is opgenomen in Bijlage L en de testresultaten in bijlage M. In deze documenten zijn ook de testen over de logging nog opgenomen, zodat deze eventueel in de toekomst nog gebruikt kunnen worden. Hierdoor is in de resultaten te zien dat een aantal testen nog niet zijn uitgevoerd.

11. Beheer en onderhoud

Dit hoofdstuk gaan over de laatste fase van de waterval methode. Dit is de fase waarin het systeem in gebruik wordt genomen en onderhouden. Deze fase valt buiten de doorlooptijd van dit project, maar in het project is er wel alles aan gedaan om deze fase zo goed mogelijk te maken. In het project is, zoals eerder al is vermeld, veel aandacht besteed aan de documentatie en leesbaarheid van de source-code. Daarnaast zijn ter verduidelijking nog een drietal sequentie diagrammen gemaakt die beschrijven wat er gebeurt tijdens: het opstarten van het systeem, de loop() functie van de Controller en het kalibreren van de infrarood camera. Deze sequentie diagrammen zijn opgenomen in bijlage N.

DEEL 3: Afrondende hoofdstukken

12. Conclusies

De doelstelling aan het begin van het project was om het huidige camera-stabilisatie-systeem te optimaliseren voor de Teensy 3.0. Tijdens de uitvoering van het project bleek dat dit vooral ging om de structuur en begrijpbaarheid van de code en daarnaast moeten er nog een viertal uitbreidingen worden toegevoegd. Aan het einde van dit project kan ik zeggen dat deze doelstellingen voor het grootste gedeelte behaald zijn.

Het stabilisatie programma is aangepast om te draaien op de Teensy 3.0, wat uiteindelijk zelfs een Teensy 3.1 is geworden. In de testen die aan het einde zijn uitgevoerd blijkt dat het systeem in staat is om onder een multicopter te worden bevestigd en de bewegingen voor het grootste gedeelte te neutraliseren. De kleine afwijking die het systeem nu nog heeft is niet storend en het kan daarom gewoon worden ingezet.

Daarnaast ben ik ook tevreden over het eindresultaat van de doorgevoerde uitbreidingen. Het is mogelijk om systeem variabelen aan te passen en tijdens het fine-tunen van het systeem bleken ze erg handig te zijn en goed te werken.

Het aanpassen van de pitch hoek werkt ook naar wens. De piloot is in staat om de camera hoek aan te passen, waarna de camera met een vloeiende beweging naar de aangegeven hoek toe beweegt. Helaas draait de camera net iets te ver door, waardoor deze zich vervolgens weer moet corrigeren. Dit heeft ook te maken met het feit dat de PID waarden nog strakker afgesteld moeten worden.

Ook de communicatie met de camera werkt naar behoren. Het systeem reageert snel als het de opdracht krijgt om van videostroom te wisselen en het kalibreren van de infrarood camera werkt ook. Helaas konden deze laatste twee functies niet vliegend worden getest, vanwege een verouderde versie van de multicopter software, maar door middel van de gtkterm terminal is wel bevestigd dat het systeem goed reageert op de doorgegeven code’s.

Het is wel spijtig dat de logging mogelijkheid niet gelukt is. Doordat de SDFat library, die zorgt voor de communicatie met de SDkaart, zelf een aantal SPI registers aanpast, kon naast deze SD kaart adapter geen ander SPI device meer worden aangestuurd. Hierdoor konden de sensor waarden niet meer goed worden uitgelezen en begon het systeem allemaal rare bewegingen te maken. De logging uitbreiding is goed gedocumenteerd, waardoor het mogelijk is om deze uitbreiding in de toekomst door te voeren, door bijvoorbeeld een extra embed toe te voegen die zich puur bezighoud met de logging.

Tot slot is het systeem goed gedocumenteerd, waardoor de beheerders het systeem goed kunnen onderhouden. Dit zorgt ervoor dat het project, naar mijn mening, met succes is afgerond. Het huidige stabilisatie systeem is met succes naar de nieuwe situatie overgezet en er zijn een aantal uitbreidingen aan toegevoegd, waardoor het systeem zelfs verbeterd is.

13. Aanbevelingen

Aan het einde van het project zijn er een paar punten waar de opdrachtgever nog naar zou kunnen kijken.

Het eerste punt is de PID controller. Deze kan met behulp van het programma om de variabelen aan te passen wellicht nog iets strakker worden afgesteld. Op deze manier kan de kleine beweging die het systeem tijdens het stabiliseren nog maakt worden verholpen. Net als de momenten waarbij de pitch hoek wordt aangepast en de camera net even te ver door beweegt.

Ook kan de opdrachtgever nog kijken of het mogelijk is de PID controller als echte PID controller te gebruiken. Zoals in hoofdstuk 10 al is uitgelegd, wordt de uitkomst van de PID controller direct naar de motor doorgestuurd. Hierbij kan nog worden gekeken of het mogelijk is om daar een extra variabele tussen te zetten die wordt aangepast door de PID controller en vervolgens pas naar de motor wordt gestuurd. Dit kan helpen bij het fine-tunen van de waarden van de PID controller. Hier zijn allemaal methodes voor, die in dit geval niet lekker werken, omdat de PID controller niet als officiële PID controller is geïmplementeerd.

Tot slot moet de opdrachtgever nog kijken of het wenselijk is om de logging mogelijkheid nog in te bouwen. In dat geval kan er een extra embed worden aangeschaft die zich specifiek richt op het loggen. Deze kan via een seriële verbinding met de Teensy worden verbonden, waarna de Teensy alles doorstuurt naar de deze extra embed. Doordat de code voor het loggen nog is bewaard, kost het doorvoeren van deze uitbreiding niet heel veel tijd.

14. Evaluatie

In de evaluatie blik ik terug op het afstudeerproject, waarbij ik allereerst een verantwoording geef van de uitgevoerde beroepstaken, vervolgens het gemaakte product evalueer en tot slot nog het proces.

14.1 Beroepstaken verantwoording

Tijdens het afstudeer project heb ik de focus gelegd op het uitvoeren van een vijftal beroepstaken. Hieronder staan de beroepstaken vermeld, met daarbij per beroepstaak de bewijsvoering dat de beroepstaak op niveau is uitgevoerd.

A1 – Analyseren probleemdomein – niveau 3

Aan het begin van de stage moest worden bekeken waar het probleem lag in de huidige situatie. Dit heb ik gedaan in de analyse fase, beschreven in hoofdstuk 5. Tijdens deze analyse heb ik ontdekt dat er meer aan de hand was dan de opdrachtgever oorspronkelijk duidelijk maakte. Vooraf dacht ik dat het grootste probleem was dat de Arduino te weinig capaciteit had om uit te breiden, maar uiteindelijk bleek het grootste probleem te liggen bij de structuur van het huidige framework, wat erg rommelig in elkaar zat. De gehele probleem analyse met de bijbehorende oplossing is opgenomen in het Oriëntatie document in bijlage C.

A4 – Kiezen van een ontwikkelstrategie – niveau 2

Aan het begin van het project heb ik aan de hand van een aantal verschillende criteria bepaald dat de watervalmethode een geschikte ontwikkelmethode is voor het project. Hierbij heb ik ook gekeken naar incrementiële en iteratieve methoden, maar vanwege de geringe omvang en geringe onzekerheid van het project zijn deze destijds afgevallen. Aan het begin van het project werd namelijk verondersteld dat onderzoek een groot aandeel zou hebben in het project. Toen uiteindelijk duidelijk werd dat er ook nog uitbreidingen werden doorgevoerd tijdens de afstudeer periode, was de watervalmethode minder geschikt. Helaas was ik al te ver gevorderd met het project om nog van ontwikkelmethode te veranderen. Daarom heb ik door middel van het zelf toevoegen van iteraties, de methode dusdanig aangepast dat deze toch geschikt was om te gebruiken bij dit project, zodat alles aansloot op de al gemaakte documentatie. De keuzeverantwoording voor het kiezen van de juiste ontwikkelmethode is opgenomen in hoofdstuk 4: project aanpak. Daarnaast is deze keuzeverantwoording ook opgenomen in het plan van aanpak, te vinden in bijlage B.

C13 – Het betrekken van Real-time aspecten bij een ontwerp – niveau 3

In de technisch ontwerp fase is beschreven op welke manier er aandacht wordt besteed aan het real-time aspect van het systeem. Er is voor gekozen om geen real-time operating system te installeren op de Teensy, omdat de processen elkaar in chronologische volgorde opvolgen. Bij het ontwerp zijn ook geen extra optimalisaties toegepast, aangezien het systeem al functioneerde op een Arduino en de Teensy veel sneller is. Tijdens het ontwikkelen is wel steeds in de gaten gehouden dat het systeem de loop() functie uit de Controller binnen de milliseconde uitvoert. In het uiteindelijke systeem is dit zeker het geval. Van alle functies is getest hoe lang het duurt om ze uit te voeren. Het ontwerp van het real-time aspect is opgenomen in het Technisch ontwerp in bijlage G en in paragraaf 8.2 is beschreven op welke manier de tijdsduur van de verschillende functies is getest.

D17 – Testen van softwaresystemen – niveau 3

Tijdens het project is grondig aandacht besteed aan het testen van het project. Zo is er in de basisontwerp fase een testplan opgesteld, wat staat vermeld in hoofdstuk 6. Bij het opstellen van dit testplan is goed nagedacht over de manier waarop de verschillende eisen moeten worden getest en de verschillende kwaliteitsaspecten die daarbij komen kijken. Vervolgens zijn tijdens deze fase ook de eerste paar testscenario's ontwikkeld. Deze scenario's zijn in de loop van het project aangevuld en soms iets aangepast,

zodat ze beter aansloten bij de veranderende situatie. In de testfase zijn de meeste van deze testen uitgevoerd, waarover meer is te lezen in hoofdstuk 11. Sommige testen konden niet worden uitgevoerd, vanwege een verouderde versie van de software op de multicopter of omdat ze te maken hebben met de logging uitbreiding die uiteindelijk niet in het systeem is ingebouwd. Het testplan is te vinden in bijlage E, de testscenario's in bijlage L en de testresultaten in bijlage M.

H7 – Professioneel werken: Methodisch werken – niveau 3

Deze beroepstaak valt veel samen met beroepstaak A4 – Kiezen van een ontwikkelstrategie. In taak A4 is een methode gekozen en in taak H7 is deze gevolgd. Tijdens de uitvoering van het project is een aantal keer van de methode afgeweken, zoals beschreven is in hoofdstuk 9. De rapporten die zijn gemaakt en opgeleverd aan de opdrachtgever komen overeen met deze ontwikkelmethode. Per fase is er een rapport gemaakt, die allemaal zijn opgenomen in de bijlagen.

14.2 Product evaluatie

Ik ben zeer tevreden over het opgeleverde product. Toen het systeem uiteindelijk onder de multicopter werd bevestigd en ik zag dat het beeld stabiel was en de verschillende uitbreidingen werkten, was ik best trots. Tijdens het ontwikkelen heb ik me mogen verdiepen in PID controllers en dergelijke functies, waardoor er een product is opgeleverd wat goed functioneert. Daarnaast is ook alles goed gedocumenteerd en vind ik zelf dat veel duidelijker is wat de functies zijn van verschillende stukken code. Naar mijn idee was ook de opdrachtgever tevreden over het eindresultaat. Jammer dat de mogelijkheid om te loggen niet is gelukt en dat de PID controller nog niet helemaal netjes is geïmplementeerd. Dat had ik nog graag aan het product toegevoegd, maar ondanks dat ben ik zeer tevreden over het eindresultaat.

14.3 Proces evaluatie

Al met al ben ik tevreden over het verloop van het project. Ik heb veel geleerd over het functioneren binnen een bedrijf, maar heb ook weer veel ervaring opgedaan binnen het ontwikkelen van een technisch systeem. Een van de grootste uitdagingen was om de werking van het oorspronkelijke systeem te begrijpen en te bedenken hoe hier een betere structuur in kon worden aangebracht. Maar ik ben tevreden over de manier waarop het is gelukt en ergens ook wel trots dat het me is gelukt. Een andere grote uitdaging was het begrijpen van de werking van de PID controller. Hier had ik nog nooit mee gewerkt en heeft me veel stress opgeleverd voordat deze helemaal goed was ingesteld. Maar ook die uitdaging heb ik overwonnen.

Het enige punt in het proces waarvan ik het jammer vind hoe het is gelopen, is dat pas later duidelijk werd dat er uitbreidingen aan het systeem moesten worden toegevoegd. Wanneer ik dit van tevoren had geweten, had ik waarschijnlijk een andere keuze gemaakt voor de ontwikkelmethode. Op dit moment ben ik in de constructie fase als het ware afgeweken van de waterval methode, door toch een soort van iteraties in te plannen, waarin ik de nieuwe uitbreidingen maakte. Dit maakte het ontwerp proces ook wat onoverzichtelijker, omdat ik iedere keer een nieuw document heb aangemaakt, wat een aanvulling vormde op de al bestaande ontwerpdocumenten. Hierdoor kostte het uiteindelijk ook meer tijd om alle documenten aan het einde te vangen in een groot document wat aan het bedrijf moest worden opgeleverd.

Maar daarnaast ben ik wel heel dankbaar voor de begeleiding die de medewerkers van Delft Dynamics mij hebben gegeven en de goede sfeer waarbinnen ik heb mogen afstuderen. Wanneer ik zat met een probleem kon ik altijd bij ze terecht. Uiteraard heb ik wel zoveel mogelijk geprobeerd in eerste instantie zelf de problemen waar ik tegenaan liep op te lossen, maar ik voelde mij wel vrij om ze om hulp te vragen op het moment dat ik er echt niet uit kwam.

Literatuurlijst

- http://www.delftdynamics.nl/index.php
  - Laatst geraadpleegd: 21 februari 2014
  - Website van Delft Dynamics B.V.

- Grit, R. (2008) *Project management*. (5e druk). Groningen/Houten: Noordhoff Uitgevers
  - Boek over het managen van projecten in het algemeen.

- http://nl.wikipedia.org/wiki/Watervalmethode
  - Laatst geraadpleegd: 21 februari 2014
  - Wikipedia artikel over de waterval methode
  - Gebaseerd op: NAGING THE DEVELOPMENT OF LARGE SOFTWARE SYSTEMS , Dr. Winston W. Rovce
  - Pdf artikel te vinden op: http://www.cs.umd.edu/class/spring2003/cmsc838p/Process/waterfall.pdf
  - Laatst geraadpleegd op 25 april 2014

- Bouwman, E. (2008) *SmarTEST, Slim testen van informatiesystemen*. (2e geheel herziene druk). Den Haag: Sdu uitgevers bv.
  - Boek over SmarTEST, een methode voor het testen van software

- http://www.pjrc.com/teensy/
  - Laatst geraadpleegd: 28 februari 2014
  - Officiële website van de Teensy 3.0

- http://www.invensense.com/mems/gyro/mpu6050.html
  - Laatst geraadpleegd: 28 februari 2014
  - Officiële website van de MPU6000, met onder andere de volgende documenten:
  - MPU-6000/MPU-6050 9-Axis Evaluation Board User Guide
  - MPU-6000 and MPU-6050 Register Map en Descriptions Revision 4.2
  - MPU-6000 and MPU-6050 Product Specification Revision 3.4

- http://code.google.com/p/gluonpilot/source/browse/trunk/Firmware/lib/mpu6000/?r=146
  - Laatst geraadpleegd: 7 maart 2014
  - Website met een MPU6000 library

- PDF document: "*TauTM 2 / QuarkTM Software Interface Description Document (IDD)* , May 20 2013 , Document Number: 102-PS242-43 , Version 120"
  - Uitleg over het gebruik van het seriële protocol van de infrarood camera
  - Te vinden op: http://flir.com/cvs/cores/view/?id=51266&collectionid=549&col=51276
  - Laatst geraadpleegd: 7 mei 2014

- http://www.nongnu.org/avr-libc/user-manual/group__util__crc.html
  - Laatst geraadpleegd: 7 mei 2014
  - Website met verschillende C++ code fragmenten om verschillende CRC berekeningen te kunnen uitvoeren.

- http://forum.pjrc.com/threads/16758-Teensy-3-MicroSD-guide
  - Laatst geraadpleegd: 6 mei 2014
  - Post op een forum met een beschrijving hoe de MicroSD Card Adapter kan worden gebruikt voor de Teensy 3.0

- http://code.google.com/p/sdfatlib/downloads/detail?name=sdfatlib20131225.zip&can=2&q
    - Laatst geraadpleegd: 5 mei 2014
    - Site waar de SDFAT library te vinden is

Afbeelding verantwoording

- **Afbeelding 1.1: Een multicopter van Delft Dynamics**
  - http://www.delftdynamics.nl/images/stories/dd/20130523_spyder.jpg
  - Laatst geraadpleegd: 29 mei 2014

- **Afbeelding 2.1 – Afbeelding van hoogspanningsmast, gemaakt door een helikopter van Delft Dynamics**
  - http://www.delftdynamics.nl/images/stories/dd/20110923_liandon2.jpg
  - Laatst geraadpleegd: 28 februari 2014

- **Afbeelding 2.2 – Organogram Delft Dynamics**
  - Gemaakt via: http://www.gliffy.com/
  - Laatst geraadpleegd: 28 februari 2014

- **Afbeelding 2.3: De RH4 Spyder**
  - http://www.delftdynamics.nl/images/stories/dd/20130523_spyder.jpg
  - Laatst geraadpleegd: 29 mei 2014

- **Afbeelding 4.1 – Diagram van het V-model**
  - Gemaakt via: http://www.gliffy.com/
  - Laatst geraadpleegd: 28 februari 2014

- **Afbeelding 6.1: Roll, Pitch en Yaw beschrijving**
  - http://upload.wikimedia.org/wikipedia/commons/7/7e/Rollpitchyawplain.png
  - Laatst geraadpleegd: 30 mei 2014

- **Afbeelding 6.2 – Globaal aansluitdiagram hardware componenten**
  - Gemaakt via: http://www.gliffy.com/
  - Laatst geraadpleegd: 28 februari 2014

- **Afbeelding 6.4 – Globaal klassendiagram**
  - Gemaakt via: https://www.draw.io/
  - Laatst geraadpleegd: 28 februari 2014

- **Afbeelding 7.1 – Pinout van de voorkant van de Teensy**
  - http://www.pjrc.com/teensy/teensy30_front_pinout.png
  - Laatst geraadpleegd: 7 maart 2014

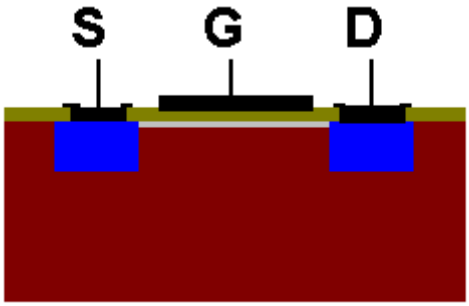
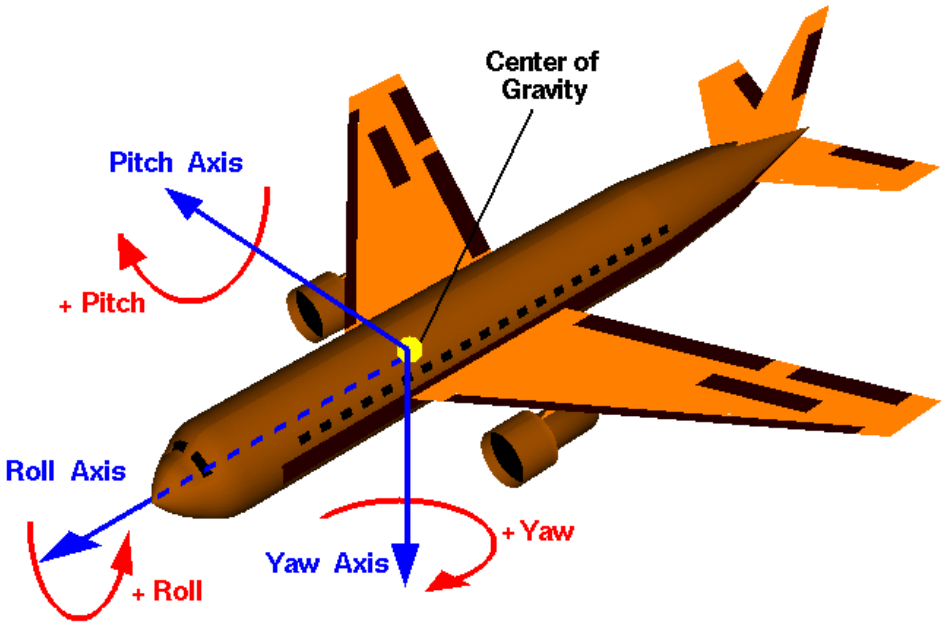
- **Afbeelding 7.2 – Klassendiagram camera-stabilisatie-systeem**
  - Gemaakt via: https://app.genmymodel.com
  - Laatst geraadpleegd: 30 mei 2014

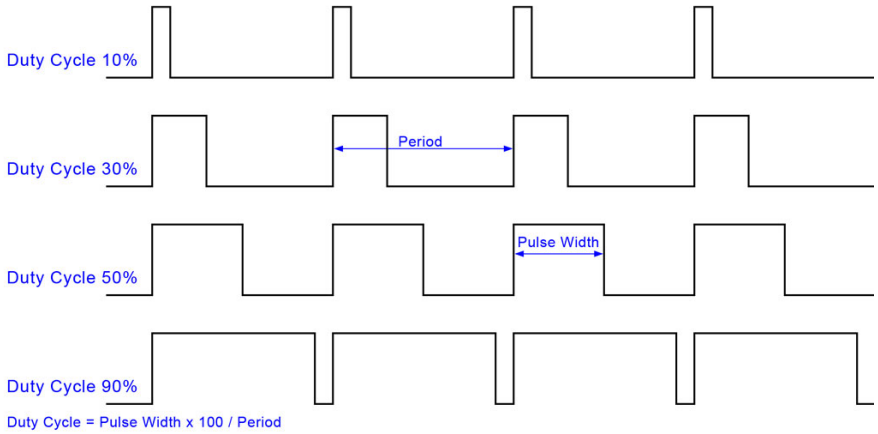
Verklarende woordenlijst

Accuracy	De accuracy van een sensor is de maximale waarde, die de gemeten waarde van een sensor kan verschillen met de werkelijke waarde.
ASCII	ASCII (American Standard Code for Information Interchange) is het systeem waarbij 128 standaard karakters vertaald zijn naar een 7 bits representatie. In een zogeheten ASCII tabel kan worden opgezocht welk karakter hoort bij welke 7 bit representatie.
Avionica	Avionica is een subonderdeel van de elektrotechniek, gespecialiseerd op elektronica in de luchtvaart.
Bandwidth *(sensoren)*	De Bandwidth van een sensor geeft aan hoe vaak een sensor metingen kan uitvoeren binnen een bepaalde tijd. Een sensor met de bandwidth van 25Hz, kan 25 metingen per seconde verrichten.
Baudrate	De baudrate geeft aan hoeveel bits (of pulsen) er per seconde over een communicatie kanaal worden verzonden.
Bias *(sensoren)*	Wanneer het output signaal van een sensor niet 0 is wanneer de sensor wel 0 meet heeft de sensor een bias (of offset). De bias is de output die de sensor geeft op het moment dat de sensor 0 meet.
Binary (Binair)	Decimal, Hexadecimal en Binary zijn verschillende getalstelsels met ieder een ander grondtal. Bij Decimal is het grondtal 10, bij Hexadecimal is dit 16 en bij Binary is dit 2. Zo wordt het getal twintig in Decimal: 20, Hexadecimal: 14 en Binary: 10100
Brushed motor	Het grote verschil tussen een brushless en brushed motor is dat een brushed motor koolborstels bevat en een brushless motor niet. Brushed motoren zijn tweefasige motoren (met twee aansluitdraden) die werken op gelijkstroom waarvan de snelheid geregeld kan worden met zowel een mechanische als een elektronische regelaar (deze hebben twee inkomende draden +- en twee uitgaande draden +-). Brushless motoren zijn driefasige motoren (met drie aansluitdraden) die werken op wisselstroom waarvan de snelheid van de pulsen (frequentie) wordt geregeld door een elektronische snelheidsregelaar welke van gelijkspanning een driefasige wisselspanning maakt (deze hebben twee inkomende draden+- en drie uitgaande draden wisselspanning). Door de pulsen dichter of verder uit elkaar te leggen (versnellen of vertragen) zal de motor sneller of trager draaien.

Brushless motor	*Zie: Brushed motor*
Checksum	Een checksum is een manier om te controleren of gegevens tijdens het verzenden niet corrupt zijn geraakt. Voordat een bericht wordt verzonden, wordt via een formule een getal berekend wat hoort bij dat bericht. Nadat het bericht is verzonden, wordt van het bericht, via dezelfde formule, nogmaals de checksum berekend. Als de checksum van voor het verzenden verschilt met die van na het verzenden is het bericht niet juist verzonden.
Decimal (Decimaal)	*Zie: Binary*
CRC	CRC (cyclic redundancy check) is een methode om fouten op te sporen in verzonden berichten. De decimale waarde van een bepaald bericht wordt voor het verzenden gedeeld door een vaststaand getal. De restwaarde van deze deling wordt mee verzonden. Nadat het bericht is verzonden wordt de deling nogmaals uitgevoerd, waarna te zien is of er een fout is opgetreden tijdens het verzenden.
Embedded operating system	Een operating system is de software laag die de verbinding vormt tussen de hardware van het systeem en de software programma's die een gebruiker gebruikt. Een embedded operating system is specifiek voor embedded computer systemen (computer systemen gemaakt voor één speciaal doel).
FET	Een FET (field-effect-transistor) is een unipolaire transistor met gewoonlijk drie aansluitingen: de source (S), de drain (D) en de gate (G) en bestaat uit een geleidingskanaal tussen de aansluitingen source (S) en drain (D), waarvan de geleiding beïnvloed kan worden door het elektrische veld van de spanning op de gate (G). De transistor heet unipolair omdat slechts één soort ladingsdrager (gaten of elektronen) deelneemt aan de stroom tussen source en drain.
Frequency	Frequency geeft aan hoe vaak een bepaalde gebeurtenis voorkomt binnen een bepaalde tijd. Vaak aangeduid in Hz (Hertz). Een gebeurtenis met een frequency van 25 Hz, komt 25 keer per seconde voor.
Hexadecimal (Hexadecimaal)	*Zie: Binary*

I^2C	I^2C (Inter-Integrated Circuit) is een bus ontwikkeld voor datacommunicatie tussen microprossesoren en andere IC's die gebruikt maakt van twee communicatie signalen: • SDA (serial data) • SCL (serial clock) Zowel de master als de slave communiceren via de SDA. Daarom mag een slave alleen communiceren nadat de master hem daarom vraagt. Daarnaast is I^2C een half-duplex protocol. Dit betekent dat óf de master óf de slave informatie kan verzenden, maar nooit tegelijk.
Interrupt	Een interrupt (letterlijk onderbreking) is een verzoek om aandacht van een hardwarecomponent aan een andere.
Interrupt handler	Een interrupt handler, of Interrupt Service Routine (ISR) is de subroutine die wordt uitgevoerd nadat er een bepaalde interrupt is ontvangen.
Interrupt Service Routine	*Zie: Interrupt handler*
IMU	Een IMU (inertial measurement unit) is een elektronisch apparaat dat de snelheid en oriëntatie van een voorwerp bepaalt, door gebruik te maken van een acceleratiesensor en een gyroscoop.
Latency	Latency geeft aan hoeveel vertraging er is tussen een bepaalde oorzaak en het gevolg. Als je bijvoorbeeld een knopje indrukt, waarbij een lampje moet gaan branden en het duurt, na het indrukken van het knopje, 1 seconde voordat het lampje gaat branden, heeft dat systeem een latency van 1 seconde.
LSB	LSB (Least Significant Bit) en MSB (Most Significant Bit) geven aan welk bit de hoogste (MSB) en de laagste (LSB) waarde heeft in een byte. Zo is in de byte 00010101 (decimale getal 21) de meest linker 0 de MSB en de meest rechter 1 de LSB.
MSB	*Zie: LSB*

MOSFET	Een MOFSET (Metal-Oxide-Semiconductor FET) is een speciaal soort FET die is opgebouwd uit de lagen: metaal, oxide en semiconductor. Door het aanbrengen van een spanning op de gate terminal verandert de ladingsdragerconcentratie in de halfgeleider, hierdoor verandert de weerstand tussen de source en drain terminal. Voorbeeld afbeelding van een MOSFET:  *Afbeelding: http://upload.wikimedia.org/wikipedia/commons/8/89/MOSFET.PNG*
Noise	Noise (letterlijk ruis) zijn resultaten die afwijken van de werkelijkheid. Bij geluid kan dit komen door teveel achtergrondgeluid. Een bekend voorbeeld bij foto's zijn spikkels of vlekken (bijvoorbeeld vanwege een te lage resolutie).
Operating system	*Zie: Embedded operating system*
Pitch	Roll, Pitch en Yaw geven in de luchtvaart de drie draaiingen aan die een luchtvoertuig kan maken. In onderstaande afbeelding is aangegeven om welke draaiingen dit gaat:  *Afbeelding: http://upload.wikimedia.org/wikipedia/commons/7/7e/Rollpitchyawplain.png*

PWM	PWM (pulse-width modulation) is een modulatie techniek die ervoor zorgt dat over een vaststaande periode, bijvoorbeeld 1 ms, de pulsbreedte (de tijd dat binnen die periode de puls hoog is) varieert. Op deze manier kan de gemiddelde waarde van een signaal worden beïnvloed. In onderstaande afbeelding is dit begrip visueel gemaakt.  *Afbeelding: http://d32zx1or0t1x0y.cloudfront.net/2011/06/atmega168a_pwm_02_lrg.jpg*
Resolution	De resolution van een sensor is de kleinst mogelijke verandering die een sensor kan meten.
Roll	*Zie: Pitch*

RS232

Dit is een communicatie standaard die gebruikt wordt om informatie tussen twee systemen te verzenden. RS232 maakt gebruik van minimaal 3 signalen (GND, TD en RD) en maximaal 9 verschillende signalen en kan worden gebruikt bij verschillende connectoren.

Signaal	DB-25	DB-9	EIA/TIA 561	Yost
Massa (Common Ground, GND)	7	5	4	4,5
Verzonden data (Transmitted Data, TD)	2	3	6	3
Ontvangen data (Received Data, RD)	3	2	5	6
Data gereed (Data Terminal Ready, DTR)	20	4	3	2
Data ontvangen (Data Set Ready, DSR)	6	6	1	7
Verzoek tot zenden (Request To Send, RTS)	4	7	8	1
Gereed voor zenden (Clear To Send, CTS)	5	8	7	8
Verbinding gedetecteerd (Carrier Detect, DCD)	8	1	2	7
Oproep indicator (Ring Indicator, RI)	22	9	1	-

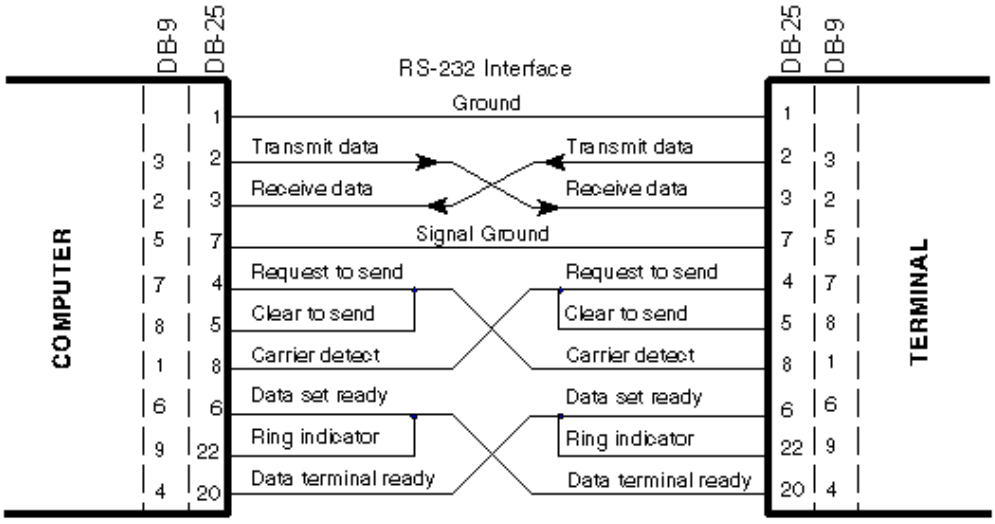


Figure 1. Direct-to-computer RS-232 Interface

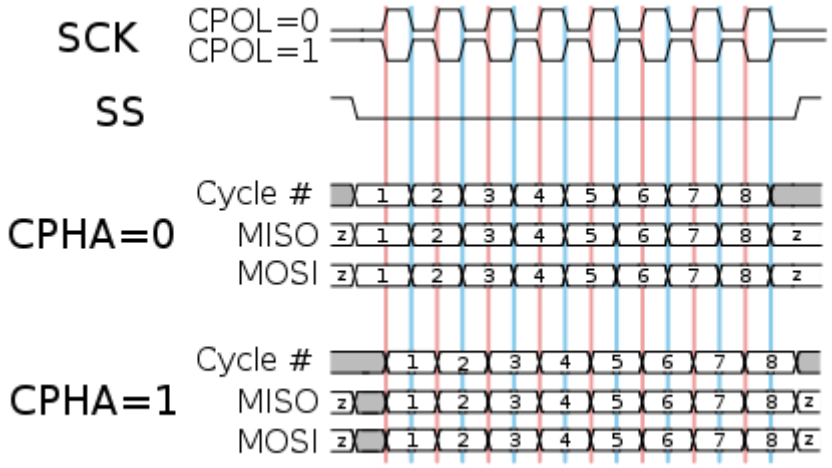
Afbeelding: http://huntvac.com/documents/rs-232-c.gif

SPI

SPI (Serial Peripheral Interface) is een communicatie protocol tussen een master en een slave apparaat. SPI maakt gebruik van vier signalen:

- SCLK: Serial Clock (output van master).
- MOSI: Master Output, Slave Input (output van master).
- MISO: Master Input, Slave Output (output van slave).
- SS: Slave Select (output van master).

Daarnaast is SPI een full-duplex protocol. Dit betekend dat zowel de master als de slave tegelijkertijd informatie kunnen verzenden.



Afbeelding:
http://upload.wikimedia.org/wikipedia/commons/b/b6/SPI_timing_diagram.svg

Voltage levels

Voltage levels geven aan welk voltage gebied overeenkomt met een binare 0 of 1. Deze voltage levels verschillen per protocol, namelijk:

TTL – 0	0V – 0.8V
TTL – 1	2V – 5V
CMOS – 0	0V – 1/3 aangeleverde voltage
CMOS – 1	2/3 aangeleverde voltage – aangeleverde voltage
RS232 – 0	+3v – +25V
RS232 – 1	-3V – -25V

Yaw

Zie: Pitch