

UNIVERSITEIT ANTWERPEN
Faculteit Toegepaste Ingenieurswetenschappen
2014-2015

Multicopters 'standalone navigatie' met behulp van OpenCV

Verhandeling voorgedragen
tot het bekomen van de graad

Yannis De Cleene

**Industrieel Ingenieur
Elektromechanica**

Promotor:
prof. dr.ir. Luc Mertens

Opdrachtomschrijving

Multicopters: ‘standalone navigatie’ met behulp van OpenCV

Probleemstelling en motivatie

Dit eindwerk zal handelen over standalone navigatie van multicopters, beter gekend als drones. Hierbij wordt gebruik gemaakt van Open Source platformen zoals Raspberry Pi, Minnowboard en toegankelijke programmeertalen als Python. Door het gebruik van de Open Source visiebibliotheek OpenCV zorgen we voor een kleinere instapdrempel omdat bedrijven zo geen licenties hoeven te betalen. De beeldverwerking en de bijhorende stuurprogramma’s zullen getest worden aan de hand van metingen om zo het gehele proces van industriële automatisering zo goed mogelijk te benaderen.

Doelstellingen

- Kennis opbouwen en delen over OpenCV
- Leren werken met de verschillende professionele camera’s
- Toepassingen vinden voor verschillende camera’s in verband met multicopters
- Multicopters aansturen op basis van beeldtechnologie

Deliverables

- een literatuurstudie over OpenCV op standalone systemen
- programma's voor beeldverwerking van de gebruikte camera's
- een proof of concept voor een standalone gestuurde multicopter
- een set meetresultaten van de uitgevoerde proeven

Risicoanalyse

De haalbaarheid van het geheel wordt steeds bedreigd door het niet 100% compatibel zijn van systemen waarvan je dacht dat ze verenigbaar zijn. Problemen in dat verband schuilen steeds in een klein hoekje. Er blijken echter wel andere ‘development boards’ te bestaan die dan wel compatibel lijken, maar dat moet dan uiteindelijk in elk detail bevestigd worden. Mogelijk is de software van de camera's compatibel met Python. Mocht dit problemen geven dan zal er contact worden opgenomen met de producenten om te vragen of zij een oplossing hebben.

Promotor:

prof. dr.ir. Luc Mertens

Student:

Yannis De Cleene

Voorwoord

Door mijn studiejaren heen is mijn fascinatie voor technologie enkel gegroeid. Ik leerde complexe systemen begrijpen en terugbrengen tot simpele modellen. In het Multicopter project zag ik dan ook een grote uitdaging die ik met veel motivatie en zin heb aangegaan. Tijdens dit jaar heb ik geleerd wat het inhoudt om je eigen onderzoek te voeren. Wat eerst een hoop puzzelstukken waren, past nu op een logische wijze in elkaar als één grote puzzel.

De onderdelen vinden en ze correct koppelen was niet altijd gemakkelijk en daarvoor wil ik in de eerste plaats graag mijn promotor emphprof. dr.ir. Luc Mertens bedanken. Hij liet me enkele keren door de bomen het bos weer zien en daar ben ik hem zeer dankbaar voor. Ook wil ik mijn naaste collega's, Andy Slock, Inge Coudron en Laurent Keersmaekers graag bedanken voor de reflecterende gesprekken en hun raad bij tal van vraagstukken.

Tot slot wil ik ook mijn familie, mijn vrienden en mijn partner bedanken voor hun geduld en begrip wanneer ik druk aan het werk was, hun luisterend oor als ik iets knap had gemaakt of er weer een puzzelstuk paste. En voor de steun als het even wat minder ging.

Het was een behoorlijke klus maar ik ben fier op het resultaat, bedankt allemaal!

Yannis De Cleene

15 mei 2015

Inhoudsopgave

Opdrachtomschrijving	ii
Voorwoord	iv
Inhoudsopgave	v
Lijst van Figuren	ix
Woordenlijst	xi
Inleiding	1
I Thesistekst	2
1 Literatuurstudie & Marktstudie	3
1.1 Marktpotentieel	3
1.2 Processoren	4
1.2.1 ARM Processor	4
1.2.2 x86 Processor	4
1.3 Camera's	5
1.4 Embedded systemen	6

1.4.1	Vereisten	6
1.4.2	Raspberry Pi	7
1.4.3	Intel Galileo	8
1.4.4	Pandaboard	8
1.4.5	Minnowboard Max	9
1.4.6	Besluit	10
1.5	Python	11
1.5.1	Matlab versus Python	12
1.6	Camera parameters en 3D naar 2D projectie	13
1.6.1	3D naar 3D	13
1.6.2	3D naar 2D	15
1.7	OpenCV	17
1.7.1	Over OpenCV	17
1.7.2	Camera Kalibratie	18
1.7.3	Filteren op kleur	20
1.8	Verdere aanpak	22
2	Praktische uitvoering	23
2.1	Proefopstelling en materiaal	23
2.2	APM Mission Planner en Python	26
2.2.1	MAVProxy	27
2.2.2	Opstartprocedure	28
2.3	Pose Estimation	28
2.4	SLAM	34
2.4.1	Wat is SLAM?	34

2.4.2	Verschillende SLAM algoritmes	35
2.5	Het framework	38
2.6	Meetresultaten	40
2.7	Alternatieve methoden	45
2.7.1	GPS	45
2.7.2	Optical Flow	45
2.7.3	Motion capture	45
2.7.4	Motion from structure	46
2.8	Mogelijke toepassingen	46
2.8.1	Landbouw	46
2.8.2	Mijnbouw	46
2.8.3	Bouw en wegenwerken	47
2.8.4	GIS	47
2.8.5	Security en bewaking van terreinen	47
2.8.6	Politiewerk	47
2.8.7	Zoek en reddingsacties	48
2.8.8	Onbemand cargo transport	48
2.8.9	Entertainment industrie	48
3	Besluit	49
Bibliografie		52
II	Bijlagen	53
A	Formularium	54

B Python en OpenCV installeren op Windows	56
C IDS uEye USB 2.0 camera op Raspberry pi	58
D Broncode Programma's	60

Lijst van figuren

1.1	Raspberry Pi 2 model B	7
1.2	Intel Galileo 2014	8
1.3	Pandaboard	9
1.4	Minnowboard Max dual core 1,33 GHz	9
1.5	Vergelijkingstabel	10
1.6	Coördinaten transformatie tussen twee assenstelsels	14
1.7	Het beeldvlak van een camera	16
1.8	De projectie van een 3D punt op het 2D beeldvlak	16
1.9	Interne cameramatrix	20
2.1	Testframe multicopter	23
2.2	DJI Flame Wheel hexacopter	24
2.3	DX7 7-kanaals controller	24
2.4	Verbinding tussen APM Flight controller en Raspberry Pi	25
2.5	APM Mission Planner	26
2.6	Bayesiaanse en Kalman filter	34
2.7	Flowchart SLAM algoritme	36
2.8	Schema van het huidige programma	38
2.9	Schema van het mogelijke toekomstige programma	39

2.10 Singlethreading vs Multithreading	40
2.11 Roll- en Pitchniveau tijdens meetproef 1	41
2.12 Throttleniveau tijdens meetproef 1	42
2.13 Roll- (en Pitch-) niveau tijdens meetproef 2	42
2.14 Pitch- (en Roll-) niveau tijdens meetproef 3	43
2.15 Throttleniveau tijdens meetproef 2 en 3	43
2.16 Roll- en Pitchniveau tijdens meetproef 4	44
2.17 Roll- en Pitchniveau tijdens meetproef 5	44

Afkortingen

SBC	Single Board Computer
CPU	Central Processing Unit
RGB	Red Green Blue / Rood Groen Blauw - format
HSV	Hue Saturated Value / Helderheid Saturatie Kleurwaarde - format
ToF	Time of Flight (camera)
IR	Infra Red (camera)
HDMI	High-Definition Multimedia Interface
GPIO	General Purpose Input Output
RCA	Tulpstekker of cinchstekker bekend van de rood, wit, geel kabels
UAV	Unmanned Aerial Vehicle
RPAS	Remotely Piloted Aircraft Systems
Drone	Synoniem voor UAV of RPAS

Inleiding

Deze thesis zal handelt over de grondbeginselen van het autonoom laten vliegen van een multicopter met als specifieke onderzoeks vraag: "Welke elementaire bouwstenen zijn nodig tot het bekomen van een autonome vlucht?". Het vraagstuk achter deze masterproef, navigatie met behulp van embedded systemen, zet voet aan de grond in verschillende domeinen. Het is uiterst belangrijk dat we tijdens het oplossen van zulk vraagstuk overzicht houden over alle belangrijke parameters en we er zo voor zorgen dat ten allen tijde aan de vereisten is voldaan. Zo moeten de hardware zoals camera's en processoren en de software waarmee deze worden aangestuurd compatibel zijn. Op software niveau moeten alle onderdelen programmeerbaar zijn zodat de camera-inputs kunnen worden geïmporteerd en de output na de berekeningen kan worden doorgestuurd naar de motoren. Doorheen de scriptie zullen de gevuldte redeneringen worden toegelicht met uitleg, formules en voorbeelden.

Om het prestatievermogen van het navigatiesysteem te optimaliseren zal onderzocht worden welke software zowel voordelig, krachtig en praktisch is. Hierbij zullen we gebruik maken van functionaliteiten via externe bibliotheken om zo snel betrouwbare code te kunnen schrijven.

Voor technische aanvulling op deze thesis wordt er verwezen naar de parallel lopende thesis door Andy Slock.

DEEL I

Thesistekst

Hoofdstuk 1

Literatuurstudie & Marktstudie

1.1 Marktpotentieel

Momenteel wordt de dronemarkt gevalideerd op zo'n 6,5 miljard dollar, waarvan een overgroot deel toe te schrijven is aan het gebruik voor militaire doeleinden. De B2B en B2C markt is echter ook al zo'n 700 miljoen waard en stijgt met 20% per jaar. Met het Koninklijk Besluit in aantocht maken de Belgische en de Europese markt zich klaar voor commercieel gebruik van drones. Aangezien de ontwikkeling van UAV technologie sneller gaat dan het wetgevend kader, is het voor vele bedrijven interessant zich klaar te maken voor een stroomversnelling op het gebied van implementatie eens de wetgeving commercieel gebruik toestaat. Zo hoopt Europa rond midden 2016 zijn wetgeving klaar te hebben.

Als we bovendien rekening houden met de groeiende markten van *Internet of Things* en *Big Data* kan men zien dat multicopters en andere UAV's hier mooi op aansluiten. Door het gebruik van drones te analyseren kunnen verschillende processen in kaart gebracht worden om deze te analyseren en verder te optimaliseren. Waar multicopters voor sommigen nog als toekomstmuziek klinken, bieden ze nu in vele applicaties al een grote meerwaarde en zullen ze over enkele jaren niet meer uit het dagelijkse leven weg te denken zijn.

1.2 Processoren

1.2.1 ARM Processor

De ARM processor is een processor die voornamelijk bekend is door zijn lage verbruik. Dit is te danken aan het lagere aantal transistoren in vergelijking met andere processoren zoals de x86 reeks van Intel. Hierdoor is de CPU ook opmerkelijk goedkoper dan de meeste andere CPU's. Momenteel wordt hij veel gebruikt voor toestellen zoals smartphones, tablets en camera's. Bij deze draagbare toestellen zijn batterijlevensduur en kostprijs de belangrijkste vereisten en dus leent de ARM processor zich hier uitermate toe. De toestellen hebben niet de rekenkracht nodig die men van een desktop computer verwacht en dit vormt dus ook geen nadeel. Ze zijn door hun licht gewicht uitermate bruikbaar voor embedded systemen en kunnen gecombineerd worden tot multicores waardoor men de rekenkracht verhoogt.

1.2.2 x86 Processor

De naam x86 wordt gegeven aan Intel processoren gebaseerd op Intel's 8086 processor. Deze reeks bevat de 80286, 80386 ,80486 en 80586 processoren waarbij de 'x' staat voor de verschillende typen. Bovendien wordt de eerste '80' meestal weggelaten om herhaling te vermijden. Elke processor uit de x86 reeks is gebaseerd op de x86 instructie set architectuur en kan geprogrammeerd worden met zowel Assembly als AT&T programmatuur. Een kenmerk uit de x86 reeks is dat de processoren achterwaarts compatibel zijn en dus programma's van oudere processoren kunnen uitvoeren maar niet van nieuwere types. De x86 processoren kunnen echter geen programma's uitvoeren die voor ARM processoren zijn geschreven en omgekeerd. Dit zorgt ervoor dat x86 drivers enkel kunnen worden geïnstalleerd op x86 processoren en dit dus een beperking oplegt aan de toestellen waarvoor ze zijn geschreven, zoals dit bij de Mesa en de Flir camera's het geval is.

Met de Intel Atom processor brengt Intel een oplossing voor lichte toestellen en embedded systemen die energieuwig moeten zijn en toch over de rekenkracht en x86 architectuur moeten beschikken. De Atom is reeds in enkele mobiele toestellen te vinden zoals zakelijke tablets, smartphones en kleine low-performance laptops. [1] [2]

1.3 Camera's

In het vraagstuk van Smart Data Clouds is het belangrijk de samenwerking van meerdere camera's als mogelijkheid te behouden. De camera's die men zou kunnen gebruiken, zijn RGB kleurencamera's, Time of Flight dieptecamera's en Infrarood warmtecamera's. Verschillende industriële camera's communiceren echter niet altijd via dezelfde protocollen. Zo werkt de uEye RGB via USB en de MESA ToF en de FLIR IR camera via Ethernet wat al voor een eerste hindernis zorgt. Men kan de diepte- en warmtecamera vrij eenvoudig tegelijk laten werken en data laten communiceren met de computer door een switch tussen te schakelen die het dataverkeer regelt in het kleine netwerk van apparaten.

Waar men ook rekening mee dient te houden zijn de drivers van de gebruikte camera's. MESA en FLIR hebben enkel x86 gebaseerde drivers ter beschikking voor hun apparaten. Dit zorgt ervoor dat deze camera's niet compatibel zijn met toestellen die niet met een x86 processor maar met bijvoorbeeld een ARM Cortex processor zijn uitgerust. Men kan zien dat door deze beperking al een behoorlijk percentage van de Single Board Computers op de markt afvallen. Tot slot moet men ook rekening houden met het feit dat in een scenario met 3 camera's de processing unit niet 1 maar 3 beelden moet doorrekenen rekening houdend met de framerates van de afzonderlijke camera's. In realtime toepassingen is het nodig dat de dataververing hoog genoeg ligt zodat het geregelde systeem voldoende snel kan ingrijpen op afwijkingen van buitenaf. Rekenkracht dient dus voldoende aanwezig te zijn indien men alle beelden wil kunnen verwerken en een redelijke performantie wil bekomen. Deze vereisten noteren we voor het selecteren van een gepaste SBC.

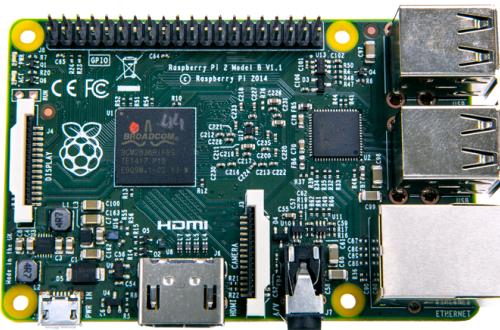
1.4 Embedded systemen

De markt van Single Board Computer is de laatste jaren sterk gegroeid. Door het grote aanbod van verschillende boards met elk hun verschillende specificaties is het belangrijk eerst de benodige specificaties op te lijsten voor de verschillende scenario's zodat men snel de ongeschikte boards uit het selectieproces kan filteren. Vervolgens worden de overgebleven boards vergeleken en kiest men hieruit het board dat het beste bij het beschreven scenario past. [3]

1.4.1 Vereisten

- Beeld output voor makkelijk programmeren, debuggen en een simpele interactie (HDMI)
- Ondersteuning voor al de gebruikte camera's en hun drivers
- Licht in gewicht
- Voldoende rekenkracht
- Beperkt stroomverbruik
- Lage kostprijs

1.4.2 Raspberry Pi



Figuur 1.1: Raspberry Pi 2 model B

Raspberry Pi is een goedkope, kleine Single Board Computer ontwikkeld in het Computer Lab aan de Universiteit van Cambridge. Het board is ontwikkeld met als hoofddoelen: jongeren meer voeling laten krijgen met computers en programmeren en dit op een goedkope manier. Het board is gebaseerd op de goedkope ARM processor en beschikt over de standaard interfaces zijnde Ethernet, USB, GPIO, Full HD HDMI en zelfs RCA zodat men het board op oude televisietoestellen kan aansluiten.

Raspbian, de lichtgewicht versie van Debian Linux, kan men makkelijk installeren op een SD-kaart en komt standaard met Python geïnstalleerd. Door de GPIO pins kan men knoppen en ledjes op het board aansluiten als analoge en digitale in- en uitgangen. Dit maakt dat men zowel softwarematig kan leren programmeren als hardwarematig met in- en outputs kan leren werken en experimenteren. De Raspberry Pi heeft een grote community die ervoor zorgt dat men makkelijk tutorials kan vinden en mogelijke problemen snel kunnen worden opgelost via diverse fora. Enkele voorbeelden hiervan kan men vinden op de website van Raspberry Pi zelf onder de tabs community en forums. Het board heeft een prijs in de grootteorde van 30-50 euro. Omwille van de hoge interactie en het snel kunnen programmeren en aansturen met Python is de Raspberry Pi een goede keuze voor onderzoekslabo's en een plezier om mee te werken. [4]

1.4.3 Intel Galileo

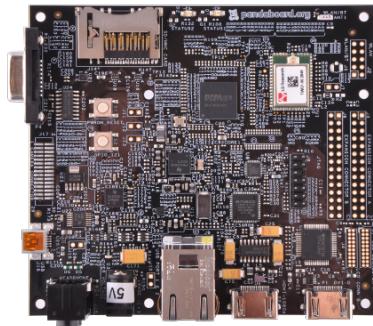


Figuur 1.2: Intel Galileo 2014

Intel Galileo is een developmentboard dat is voortgekomen uit de samenwerking tussen de twee grote spelers in de sector van hobby-engineering namelijk Intel en Arduino. Waar Intel bekend staat om zijn computer expertise staat Arduino bekend om zijn microprocessor boards met een hoge interactiefactor. Arduino maakt het makkelijk voor iedereen om motortjes aan te sturen en te meten met verschillende sensoren. Dankzij de vele uitbereidingsboardjes is er voor ieders wat wils binnen het platform. Galileo combineert het beste van beide, krachtige Intel processoren en makkelijke interactie via de Arduino layout. Het board beschikt dankzij de Intel CPU over een x86 architectuur maar door het ontbreken van een video output voldoet het board niet aan de vereisten. Het board heeft een prijs in de grootteorde van 75-100 euro. [5]

1.4.4 Pandaboard

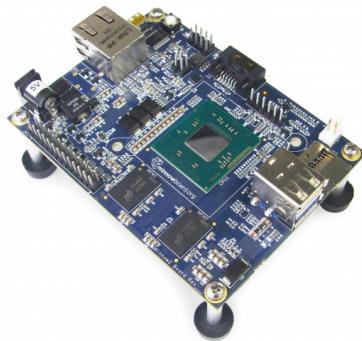
Pandaboard is een SBC die beschikt over de ARM Cortex-A9 architectuur dankzij zijn OMAP4430 chip van Texas Instruments. Pandaboard heeft 1Gb RAM geheugen en een CPU met een kloksnelheid van 1GHz daar de CPU van de Pandaboard ES een kloksnelheid van 1,2GHz heeft door zijn dualcore. Het board heeft een SD-kaartslot voor externe geheugenkaartjes tot 32Gb, een HDMI output voor makkelijke interactie, een Ethernet poort en twee USB poorten. Door deze goede specificaties wordt dit board vaak gebruikt voor het aansturen van UAV's. De beperking is echter wel de ARM processor die ervoor zorgt dat men geen gebruik kan maken van



Figuur 1.3: Pandaboard

camera's die enkel over x86 drivers beschikken. De prijs van het Pandaboard is in de grootteorde van 150 tot 200 euro. [6]

1.4.5 Minnowboard Max



Figuur 1.4: Minnowboard Max dual core 1,33 GHz

Minnowboard is een high performance development board op basis van de 64-bit Intel Atom processor en daardoor volledig compatibel met de x86 architectuur. Het board beschikt over een HDMI uitgang met Full HD resolutie waardoor het makkelijk aan te sluiten is op externe schermen. Wat dit board geschikt maakt voor professionele toepassingen zijn de 2Gb DDR3 Ram en de E3825 dual-core 1.33GHz processor. Als besturingssysteem kan men gebruik maken van zowel Linux als een light versie van Windows. Door de goede specificaties komt het board op een prijs in de grootteorde van 100-200 euro. [7]

1.4.6 Besluit

	USB	Ethernet	CPU-Type	CPU-Speed	RAM	HDMI	GPIO
Raspberry Pi 2	4	1	ARM	900MHz	1Gb		40
Intel Galileo	1	1	x86	400MHz	256Mb		12
Pandaboard ES	3	1	ARM	1,2GHz	1Gb		1
Minnowboard Max	2	1	X86	1,33GHz	2Gb		8

Figuur 1.5: Vergelijkingstabel

We kunnen uit de tabel en de vereisten besluiten dat al deze boards over voldoende capaciteiten beschikken en elk gemaakt zijn voor hun eigen doeleinden. Wanneer we enkel een RGB camera gebruiken en we slechts een minimale rekensnelheid nodig hebben komen we tot het besluit dat de Raspberry Pi de beste optie is als goedkoop instapmodel.

Wanneer we echter enkel een RGB camera gebruiken maar meer rekensnelheid nodig hebben komen de andere boards met hogere kloksnelheden met een oplossing. De hogere rekenkracht gaat echter gepaard met een grotere kost, dit zowel in aankoopsprijs als verbruik en gewicht.

Indien we, zoals binnen het project van Smart Data Clouds, met meerdere camera's willen werken moet er rekening gehouden worden met de drivers van de individuele camera's. Aangezien de ToF en IR camera x86 drivers hebben kunnen we enkel een x86 systeem gebruiken. Intel's Minnowboard Max is als een van de weinige x86 SCB's een mogelijke optie. Het board heeft een hoge kloksnelheid, de benodigde architectuur en een aanvaardbare kostprijs en is daarom de beste optie in deze situatie.

1.5 Python

Python is een object georiënteerde programmeertaal ontworpen door Guido van Rossum aan het Centrum Wiskunde & Informatica in Nederland. De programmeertaal behoort tot de groep van programmeertalen waarvan de code niet gecompileerd moet worden, genaamd de scriptingtalen. Extra functionaliteiten kunnen gemakkelijk worden toegevoegd met behulp van modules en subprogramma's die kunnen worden geïmporteerd met het „import” commando. Dit kan zowel gebeuren met extern gecompileerde modules als met subprogramma's die in dezelfde map als het hoofdprogramma staan.

Binnen het kader van dit eindwerk is er gekozen voor Python omwille van het snel iteratief kunnen programmeren. Doordat de code niet dient gecompileerd te worden, kunnen er zo gaandeweg eenvoudig aanpassingen worden gemaakt en getest. Men kan op deze manier concepten uitwerken en de focus leggen op de functionaliteiten van de programma's en nadien waar nodig de prestatie nog verbeteren. Omdat de OpenCV bibliotheek C functies bevat en deze slechts worden opgeroepen vanuit Python heeft dit geen grote negatieve gevolgen voor de prestatie van de code. Het gebruik van functies binnen Python moet echter waar mogelijk worden vermeden omdat dit het prestatievermogen wel degelijk doet dalen.

Wanneer men enige kennis heeft verkregen over OpenCV en men die applicatie in de praktijk wil inzetten kan er voor optimalisatie over geschakeld worden naar C++ om de performantie te verhogen. Omdat de OpenCV bibliotheek geschreven is in C/C++ kan deze ook in C++ programma's worden geïmporteerd. In C++ kan men echter wel zonder al te veel snelheidsverliezen eigen functies toevoegen aan de hand van klassen, wat mogelijk is omdat C++ over het objectgeoriënteerde paradigma beschikt. De hele applicatie wordt dan gecompileerd en zo krijgt men een stabiel programma. Indien de lezer nog geen basiskennis programmeren heeft genoten is het echter aan te raden te starten met Python. Van alle programmeertalen behoort Python tot de gemakkelijkst leesbare en leunt deze het dichtste aan bij pseudo-code (het uitschrijven van programmalogica in gewone taal). Wanneer de lezer al enige ervaring heeft met objectgeoriënteerd programmeren is de stap naar C++ uiteraard een minder grote drempel.

1.5.1 Matlab versus Python

Volgens de huidige status werken nog heel wat onderzoekslabo's en bedrijven met het softwarepakket Matlab van Mathworks. Om de verschillen en gelijkenissen af te wegen zetten we ze hieronder op een rijtje.

Matlab	Python/Numpy/matplotlib
De indexering van arrays begint van 1	De indexering van arrays begint van 0
Matrixen zijn 2 dimensionale arrays	Numpy heeft n-dimensionale arrays. De rij-vector is 1D en kan niet worden getransponeerd.
Matlab is ontworpen als pakket voor lineaire algebra. De API voor GUI's is later pas bijgevoegd.	Python is ontworpen als een "general purpose" programmeertaal. Vele add-ons zoals Numpy zijn ontworpen voor zeer specifieke doeleinden.
Matlab heeft een redelijk grote actieve community met veel programma's die ter beschikking worden gesteld, maar is door de licentiekost in groei beperkt.	Python heeft een grote actieve community waar Numpy echter een beperkte community heeft maar is door zijn Open Source karakter niet in groei beperkt en gemakkelijk uitbreidbaar.
Matlab heeft veel domein-specifieke toolboxen zoals signal-processing, image-processing en Simulink die meegeleverd worden of die men apart kan aankopen.	Python heeft veel minder domein-specifieke toolboxen maar heeft wel enkele add-ons en bibliotheken zoals OpenCV die als module kunnen worden ingevoerd. Er is nog geen Open Source alternatief voor Simulink.
Licenties voor Matlab kosten duizenden euro's voor bedrijven. Bovendien is de broncode niet te verkrijgen waardoor de functies niet op juistheid kunnen worden gecontroleerd. En men heeft geen invloed op bugs, performance en toekomstige ontwikkeling.	Python, Numpy en Matplotlib zijn volledig gratis voor om het even wie en medewerking of aanpassing van deze pakketen is mogelijk.

Voor de ontwikkeling van visie algoritmes en andere programma's kunnen onderzoeksgroepen met een licentie van MATLAB sinds 2012 ook gebruik maken van MATLAB Runtime. Hiermee kunnen de ontwikkelde programma's gecompileerd worden en uitgevoerd op systemen waar MATLAB zelf niet op geïnstalleerd is. Het nadeel blijft wel dat de ontwikkelaar nog steeds een MATLAB licentie nodig heeft. Dit geeft onderzoeksgroepen het voordeel dat zij complexe systemen kunnen modelleren in Simulink en deze daarna aan bedrijven kunnen verkopen zonder dat deze bedrijven de licentiekosten moeten dragen.

1.6 Camera parameters en 3D naar 2D projectie

1.6.1 3D naar 3D

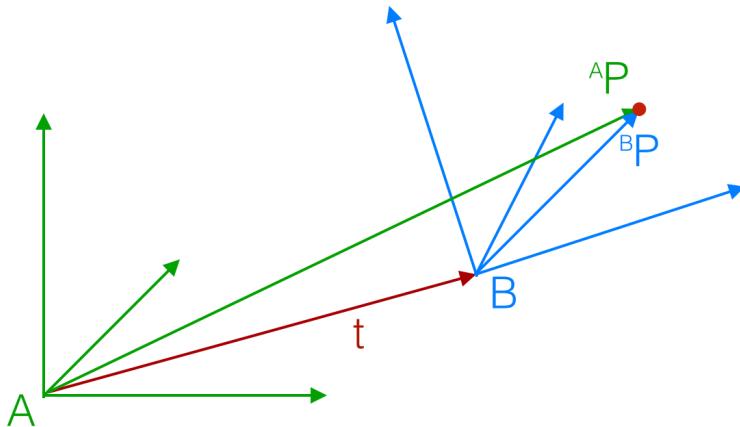
Wanneer men de locatie van de multicopter in de wereld wil weten, kan men deze voorstellen als het assenstelsel van de camera van de multicopter in referentie tot het wereldassenstelsel van de ruimte waarin de multicopter zich begeeft. Om slechts met één transformatie te rekenen, vereenvoudigen we de werkelijkheid en nemen we het assenstelsel van de multicopter gelijk aan dat van de camera. Wanneer de transformatie tussen het camera- en wereld-assenstelsel gevonden is, kan men met behulp van de transformatiematrix cameracoördinaten naar wereldcoördinaten omzetten en met behulp van de inverse transformatiematrix wereldcoördinaten terug naar cameracoördinaten omzetten. Dit zal later van toepassing zijn om de verticale verplaatsing in z en horizontale verplaatsing in x en y van de multicopter t.o.v. de wereld te bepalen. [8] [9]

$${}^A P = {}^A R * {}^B P + {}^B t_{Aorg} \quad (1.1)$$

$${}^A P = {}^A H * {}^B P \quad (1.2)$$

met ${}^A P$ en ${}^B P$ zijnde het punt P in homogene coördinaten tegenover de respectieve assenstelsel A en B en waar de totale transformatie kan worden geschreven als de homografie ${}^A_B H$ tussen beide assenstelsels:

$${}^A_B H = \begin{bmatrix} {}^A R & {}^A t_{Borg} \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (1.3)$$



Figuur 1.6: Coördinaten transformatie tussen twee assenstelsels

Voor alle punten

$$\begin{bmatrix} u_1 & u_2 & \dots & u_n \\ v_1 & v_2 & \dots & v_n \\ w_1 & w_2 & \dots & w_n \\ s_1 & s_2 & \dots & s_n \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \\ 0 & 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} x_1 & x_2 & \dots & x_n \\ y_1 & y_2 & \dots & y_n \\ z_1 & z_2 & \dots & z_n \\ 1 & 1 & \dots & 1 \end{bmatrix} \quad (1.4)$$

$$x_n = \frac{u_n}{s_n}, y_n = \frac{v_n}{s_n}, z_n = \frac{w_n}{s_n}, \text{gewicht} = \frac{s_n}{s_n} = 1 \quad (1.5)$$

Wanneer we vervolgens de gevonden punten delen door \$s_n\$ worden voor alle punten de gewichten weer 1 en staan alle punten weer in de genormeerde homogeenen coördinaten.

Indien men wil terugrekenen en een lichaam in wereldcoördinaten in cameracoördinaten wil voorstellen moet men de inverse transformatiematrix gebruiken.

$${}^A_B H = {}^B_A H^{-1} = \begin{bmatrix} {}^A_B R^T & {}^A_B R(-{}^B t_{Aorg}) \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (1.6)$$

Het belang hiervan wordt duidelijk wanneer we de transformatiematrix van camera naar wereld kennen want zo kan men aan de hand van de inverse transformatiematrix direct de locatie en rotatie van de camera t.o.v. de wereld te weten komen.

1.6.2 3D naar 2D

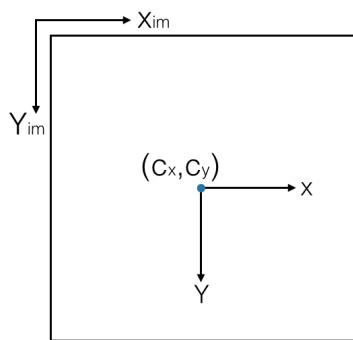
Telkens wanneer men een camera gebruikt voor beeldopnames moet men goed weten welke parameters men in beschouwing moet nemen. De focale lengte en het optisch centrum wijken altijd met een bepaalde hoeveelheid af van het ideale ontwerp. Dit kan echter gecompenseerd worden wanneer we alle cameraparameters kennen. De belangrijkste parameters eigen aan de camera zijn f_x , f_y , c_x , c_y en γ en worden de intrinsieke cameraparameters genoemd. De intrinsieke parameters kunnen voor berekeningen in matrixvorm worden geschreven.

$$K = \begin{bmatrix} f_x & \gamma & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \quad (1.7)$$

f_x en f_y stellen de focale lengte in de x en y richting voor met een eenheid van pixels/cm met betrekking tot de sensor. We bekomen s_x en s_y door de lengte van de sensor in x en y richting te delen door het aantal pixels in x en y richting. Dit geeft een maat in mm/pixel of mm aangezien pixel dimensieloos is. Formule 1.8 geeft de focale lengte in x en y richting. γ is de skew factor en wordt ter vereenvoudiging 0 genomen. Skew is een maat voor schuinheid voor wanneer men vroeger een foto van een foto maakte.

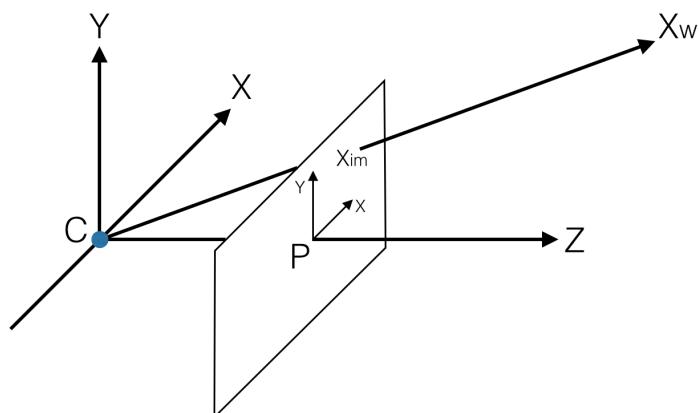
$$f_x = \frac{f}{s_x}, f_y = \frac{f}{s_y} \quad (1.8)$$

Vervolgens vinden we de pixelcoördinaten c_x en c_y van het optisch centrum door de oorsprong van het beeldvlak, genomen in de rechter bovenhoek, te verschuiven over een aantal pixels in de x en y richting (Figuur 1.7). Ideaal gezien zou het optisch centrum in het midden van het beeldvlak liggen maar dit is zelden waar. Een goede kalibratie helpt ons bij het vinden van de ware coördinaten van het optisch centrum tegenover de oorsprong van het beeldvlak evenals de best passende focale lengte van de lens.



Figuur 1.7: Het beeldvlak van een camera

Wanneer we met de camera een beeld nemen van de wereld zal elk punt in de ruimte geprojecteerd worden op het 2D beeldvlak. Elk punt met coördinaten x, y, z in de ruimte komt na deze projectie overeen met een punt i, j in het beeldvlak dat afgerond binnen de grenzen van een pixel valt (Figuur 1.8).



Figuur 1.8: De projectie van een 3D punt op het 2D beeldvlak

Als we de coördinaten van alle punten van een lichaam, zoals bijvoorbeeld de hoeken van een kubus, in matrixvorm noteren kunnen we deze met behulp van de intrinsieke matrix en de projectiematrix afbeelden op een 2D vlak.

$$\begin{bmatrix} u_1 & u_2 & \dots & u_n \\ v_1 & v_2 & \dots & v_n \\ s_1 & s_2 & \dots & s_n \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \end{bmatrix} * \begin{bmatrix} x_1 & x_2 & \dots & x_n \\ y_1 & y_2 & \dots & y_n \\ z_1 & z_2 & \dots & z_n \\ 1 & 1 & \dots & 1 \end{bmatrix} \quad (1.9)$$

of korter

$$B = K * \begin{bmatrix} R & T \end{bmatrix} * M \quad (1.10)$$

met de beeld i,j coördinaten voor elk punt

$$i_n = \frac{u_n}{s_n}, j_n = \frac{v_n}{s_n}, gewicht = \frac{s_n}{s_n} = 1 \quad (1.11)$$

Op deze manier kunnen we een model projecteren met rotatieparameters θ_x , θ_y , θ_z en translatieparameters ΔX , ΔY , ΔZ . Vervolgens kunnen we dan zoeken hoe hard deze projectie afwijkt van de werkelijkheid en de parameters aanpassen tot we de werkelijke transformatieparameters hebben waarmee we door terugrekenen de locatie van de multicopter kennen. Merk op dat de translatiematrix slecht 3 rijen heeft in vergelijking met 4 bij de 3D naar 3D transformatie. Dit is van belang om de matrices te kunnen vermenigvuldigen.

1.7 OpenCV

1.7.1 Over OpenCV

OpenCV is een Open Source bibliotheek boordevol functies gericht op computer visie. De functies zijn geschreven in C/C++ waardoor de bibliotheek een heel efficiënte code bekomt en ze toepasbaar is voor realtime toepassingen. Men kan OpenCV dus zowel voor academische als industriële toepassingen gebruiken. De bibliotheek is te gebruiken op alle belangrijke hedendaagse gebruikers platformen zijnde Windows, Linux, Mac, iOS en Android en heeft interfaces voor de programmeertalen C, C++, Python en Java. Dit laat zien hoe flexibel de functiebibliotheek is en voor hoeveel verschillende toepassingen ze kan worden ingezet. Van labo opstellingen tot standalone platformen en zelfs smartphones. [10]

1.7.2 Camera Kalibratie

Om betrouwbare meting uit beelden te verkrijgen is een goede kalibratie van de camera nodig. Dit kan men binnen OpenCV bekomen met behulp van een schaakbordpatroon. Het schaakbord patroon wordt in minstens 10 verschillende standen gefotografeerd, waarna men aan de hand van enkele functies de beeldkromming en de best passende intrinsieke matrix kan bepalen. Deze waarden moeten worden bijgehouden om in latere programma's nauwkeurige berekeningen te bekomen.

We laden de Numpy module, OpenCV en Glob, een aparte module om meerdere foto's tegelijk te laden. [11] [12]

```
import numpy as np
import cv2
import glob
```

Vervolgens bepalen we het aantal rijen en kolommen in het schaakbord patroon. We initiëren een objectmatrix die we telkens gebruiken om het schaakbord als model voor te stellen in een 3D coördinatenstelsel. Deze wordt gemaakt met nullen en overschreven met de best-fit waarden voor het model. Ook maken we arrays voor de 3D punten van het model en hun 2D projectie op de beeldsensor en de bijhorende pixelcoördinaten.

```
criteria = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER, 30, 0.001)

rows=7
columns=5
# prepare object points, like (0,0,0), (1,0,0), (2,0,0) ....,(6,5,0)
objp = np.zeros((columns*rows,3), np.float32)
objp[:, :2] = np.mgrid[0:rows,0:columns].T.reshape(-1,2)

# Arrays to store object points and image points from all the images.
objpoints = [] # 3d point in real world space
imgpoints = [] # 2d points in image plane.
```

Alle afbeeldingen die op voorhand zijn gemaakt worden geladen. Deze afbeeldingen worden één voor één bewerkt en geanalyseerd in de volgende loop.

Lees de afbeelding en converteer de BGR-lagen naar een grijswaarden beeld (OpenCV gebruikt BRG i.p.v. RGB). Zoek de hoekpunten van het schaakbord patroon met als parameters het grijswaarde beeld, het aantal rijen en het aantal kolommen. Als deze gevonden zijn, vul dan de modelmatrix aan met de modelcoördinaten en doe hetzelfde met de beeldcoördinaten nadat hun gevonden locatie verfijnd is. De hoekpunten worden op de afbeelding getekend en aan de gebruiker getoond. Tot slot worden aan de hand van de aangevulde model en projectiecoördinaten de intrinsieke matrix, de distortiefactoren, de rotatiematrix en de translatie vectoren berekend, keer op keer nauwkeuriger.

```
images = glob.glob('calib/*.jpg')

for fname in images:
    img = cv2.imread(fname)
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

    # Find the chess board corners
    ret, corners = cv2.findChessboardCorners(gray, (rows,columns),None)

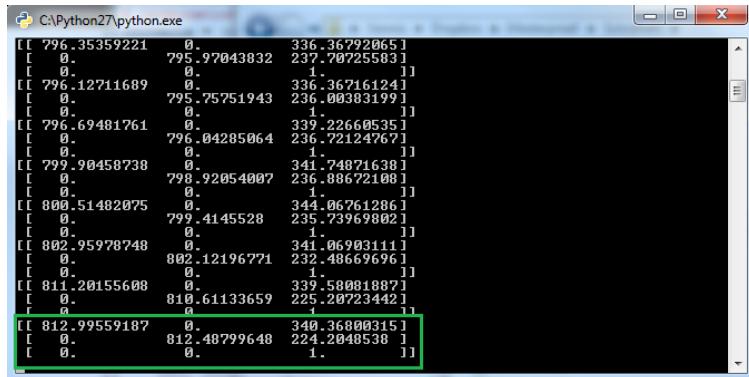
    # If found, add object points, image points (after refining them)
    if ret == True:
        objpoints.append(objp)

        cv2.cornerSubPix(gray,corners,(11,11),(-1,-1),criteria)
        imgpoints.append(corners)

    # Draw and display the corners
    cv2.drawChessboardCorners(img, (rows,columns), corners,ret)
    cv2.imshow('img',img)
    cv2.waitKey(500)

ret, mtx, dist, rvecs, tvecs = cv2.calibrateCamera(objpoints,
    imgpoints, gray.shape[::-1],None,None)
print mtx

cv2.destroyAllWindows()
```



```

[[ 796.35359221  0.          336.36792065]
 [ 0.           795.97043832  237.70725583]
 [[ 796.12711689  0.          336.36716124]
 [ 0.           795.75751943  236.00383199]
 [[ 0.           0.          1.          ]
 [ 0.           0.          1.          ]
 [[ 796.69481761  0.          339.22660535]
 [ 0.           796.04285064  236.72124767]
 [[ 0.           0.          1.          ]
 [ 0.           0.          1.          ]
 [[ 799.90458738  0.          341.74871638]
 [ 0.           798.92054007  236.88672188]
 [[ 0.           0.          1.          ]
 [ 0.           0.          1.          ]
 [[ 800.51482075  0.          344.06761286]
 [ 0.           799.4145528   235.73969802]
 [[ 0.           0.          1.          ]
 [ 0.           0.          1.          ]
 [[ 802.95978748  0.          341.06903111]
 [ 0.           802.12196277  232.48669696]
 [[ 0.           0.          1.          ]
 [ 0.           0.          1.          ]
 [[ 811.20155608  0.          339.58001887]
 [ 0.           810.61133659  225.20723442]
 [[ 0.           0.          1.          ]
 [ 0.           0.          1.          ]
 [[ 812.99559187  0.          340.36800315]
 [ 0.           812.48299648  224.2048538 ]
 [[ 0.           0.          1.          ]
 [ 0.           0.          1.          ]]

```

Figuur 1.9: Interne cameramatrix

1.7.3 Filteren op kleur

Zoals gewoonlijk worden eerst alle benodigde bibliotheken geïmporteerd en de camera geopend.

```

import numpy as np
import cv2

```

We benoemen de functie "nothing" die wordt opgeroepen wanneer de waarde van één van de schuivers wordt veranderd. Aangezien we de waarden aflezen telkens we de loop doorlopen, moet hier dus niets speciaal gebeuren en laten we ze gewoon passeren. Hierna worden de verschillende vensters aangemaakt om de belangrijkste informatie te laten zien en voegen we aan het sliders venster vier schuivers toe. De eerste twee bepalen de onderste waarde en de volgende twee de bovenste waarde.

```

cap = cv2.VideoCapture(0)

def nothing(x):
    pass

cv2.namedWindow('blob_tracker', cv2.WINDOW_NORMAL)
cv2.namedWindow('thresholded_image', cv2.WINDOW_NORMAL)
cv2.namedWindow('hsv', cv2.WINDOW_NORMAL)
cv2.namedWindow('sliders')
cv2.createTrackbar('A', 'sliders', 0, 255, nothing)
cv2.createTrackbar('B', 'sliders', 0, 255, nothing)

```

```
cv2.createTrackbar('C','sliders',0,255,nothing)
cv2.createTrackbar('D','sliders',0,255,nothing)
```

De loop wordt geopend en sluit pas als men op Q drukt. Het beeld van de camera wordt geladen en de waarden van de schuivers worden gelezen. Vervolgens wordt het beeld uitgemiddeld met een Gausiaanse blur, converteren we het BGR beeld naar een HSV (Hue Satruration Value) beeld en threshholden we het beeld met de waarden die men adhv de schuivers heeft ingegeven. Van dit zwart-wit beeld zoekt men alle witte vlakken en de contouren hiervan. We zoeken uit deze verzameling vlakken, het vlak met de grootste oppervlakte. Van dit grootste vlak zoeken we de momenten waaruit we het zwaartepunt van het vlak kunnen halen. Op het zwaartepunt tekenen we een kleine cirkel zodat men de juistheid visueel kan controleren. De gebruiker wordt er via de controller attent op gemaakt indien er geen vlakken zijn die groot genoeg zijn om te traceren.

```
while(cv2.waitKey(1) & 0xFF != ord('q')):
    ret,im = cap.read()

    a = cv2.getTrackbarPos('A','sliders')
    b = cv2.getTrackbarPos('B','sliders')
    c = cv2.getTrackbarPos('C','sliders')
    d = cv2.getTrackbarPos('D','sliders')

    # Enkel voor uEye camera op laptop
    #im2 = cv2.flip(im2,-1)

    im = cv2.blur(im,(5,5))
    hsv = cv2.cvtColor(im,cv2.COLOR_BGR2HSV)
    thresh = cv2.inRange(hsv,np.array((a, b, b)), np.array((c, d, d)))
    thresh2 = thresh.copy()

    contours,hierarchy =
        cv2.findContours(thresh, cv2.RETR_LIST, cv2.CHAIN_APPROX_SIMPLE)

    if len(contours)>0:
        max_area = 0
```

```
for contour in contours:
    area = cv2.contourArea(contour)
    if area > max_area:
        max_area = area
        best_blob = contour
    if max_area > 10:
        M = cv2.moments(best_blob)
        cx = int(M['m10']/M['m00'])
        cy = int(M['m01']/M['m00'])
        cv2.circle(im,(cx,cy),3,255,-1)
        (x,y),(MA,ma),angle = cv2.fitEllipse(best_blob)
        #print angle
    else:
        print "blobs too small"
else:
    print "no blobs found!"
```

Tot slot worden de verwerkte beelden getoond aan de gebruiker en begint de loop opnieuw. Wanneer de loop wordt gesloten worden ook de camera en alle vensters gesloten. (Voor het gebruiksgemak worden Saturation en Value gelijk genomen. Dit leverde aanvaardbare testresultaten op.)

```
cap.release()
cv2.destroyAllWindows()
```

1.8 Verdere aanpak

De vorige 2 voorbeelden zijn inleiding tot de OpenCV bibliotheek met betrekking tot deze masterproef en geven een inkijk in de mogelijkheden die deze module aanbied. De programma's zullen ontwikkeld worden met python omdat deze programmeertaal pragmatischere strategieën toelaat. Indien bijkomende functies moeten worden geschreven of hogere prestatie nodig blijkt te zijn kan er makkelijk vertaald worden naar C++ omdat de OpenCV functies gelijklopend zijn.

Hoofdstuk 2

Praktische uitvoering

2.1 Proefopstelling en materiaal

Om programma's te kunnen uitvoeren in een veilige omgeving en zonder de multicopter te beschadigen hebben we onderstaand frame gebouwd. Door de bouw van het frame kan de multicopter roteren rond al zijn primaire assen en transleren langs de z-as van het frame. Zo kan men met de multicopter vliegen zonder omstaanders of de multicopter te beschadigen bij mogelijke toestanden die anders crashes zouden opleveren.



Figuur 2.1: Testframe multicopter

Door het maken van verstelbare latjes kunnen zowel de populaire DJI Phantom quadcopter als de DJI hexacopter geïnstalleerd worden in het frame.

Omdat bij Op3Mech de focus ligt op het onderzoek naar optische meettechnieken is voor dit onderzoek gekozen voor een multicopter bestaande uit een voorgeselecteerde bouwkit. Dit versnelt het bouwen van een bestuurbaar voertuig en verhelpt voor het grootste deel de kinderziektes die gepaard gaan met het zelf samenstellen en laten functioneren van alle onderdelen. Het voordeel van deze kit is dat men gemakkelijk kan kiezen welke motoren men aansluit. Indien men later zou beslissen meerdere camera's aan het frame te bevestigen, vervangt men de motoren door krachtigere modellen en kan er op deze manier een zwaarder payload vervoerd worden.



Figuur 2.2: DJI Flame Wheel hexacopter

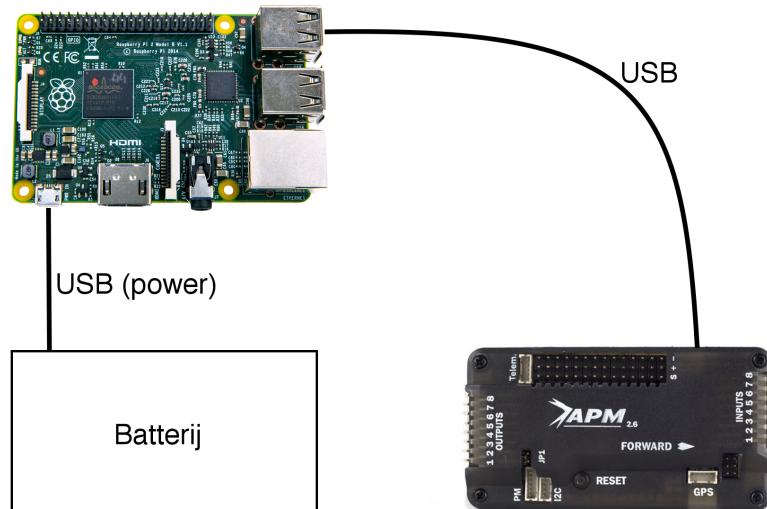
Als manuele controller gebruiken we de DX7 7-kanaals controller omdat deze de mogelijkheid geeft om buiten de commando signalen voor de motoren ook extra kanaalinputs te selecteren. Dit gebruiken we om aan de Raspberry Pi aan te geven wanneer we wensen een programma te laten runnen en te stoppen.



Figuur 2.3: DX7 7-kanaals controller

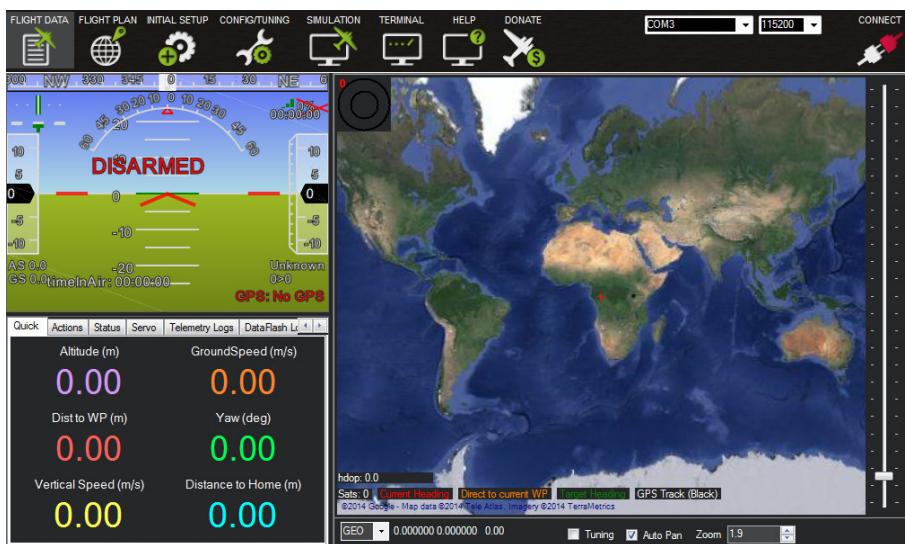
Verbinding van de APM flight controller met het embedded systeem.

Omwille van de goede eigenschappen van de Raspberry Pi is deze zeer geschikt voor gebruik met de APM 2.6 flight controller. Voor een snelle data uitwisseling kiezen we voor een USB-kabel tussen beiden. Een andere reden hiervoor is dat de USB-poorten op hetzelfde spanningsniveau werken en men geen logische omvormer moet solderen tussen de twee. Dit zou wel nodig geweest zijn moest men gebruik maken van de seriële pinnen van de GPIO headers van beide toestellen omdat deze van de Raspberry Pi werken op 3,3 Volt en die van de APM op 5 Volt. In de testfase kan men hierdoor ook met de APM werken en deze tegelijk voeden via de USB-poort. Een ander voordeel is dat de USB-poort automatisch gevonden wordt bij het openen van de link en ze niet expliciet bij het commando moet worden meegegeven.



Figuur 2.4: Verbinding tussen APM Flight controller en Raspberry Pi

2.2 APM Mission Planner en Python



Figuur 2.5: APM Mission Planner

Tijdens ons onderzoek hebben we gebruik gemaakt van de Ardupilot flight controller versie APM 2.6. Ardupilot is een stabiel platform waar men gemakkelijk een eigen voertuig kan samenstellen en dit aan de hand van vastgelegde commando's kan besturen. Door de vele inputs die de APM aan boord heeft zoals barometer, GPS, gyroscoop, enzovoort kunnen deze inputs stap voor stap worden uitgeschaakt zodat men geleidelijk aan naar meer autonomie toe werkt. Dit geeft ons de kans programma's uit te testen en toch controle te behouden over het toestel. Een bijkomende reden om voor het Ardupilot platform te kiezen is de compatibiliteit met Python. APM Mission Planner, het programma voor autonome vlucht planning, heeft de mogelijkheid Python scripts in te laden. Deze worden dan uitgevoerd binnen IronPython (een gecompileerde versie van Python). Het nadeel hiervan is dat modules en bibliotheken niet allemaal opgeroepen kunnen worden binnen het hoofdprogramma. Om dit te omzeilen werken we met subprocesses. Dit geeft ons de mogelijkheid aparte processen zoals beeldverwerking buiten het hoofdprogramma te brengen en deze simultaan/pseudo-simultaan met het hoofdprogramma te laten draaien. Door het subprocess op te roepen komen we in de omgeving van Python die op de computer zelf staat geïnstalleerd. Hier kunnen we dan de vooraf geïnstalleerde bibliotheken en modules zoals OpenCV en Numpy oproepen en beeldverwerking starten. Als men echter kiest om te werken binnen Mission Planner zelf moet men de

computer ofwel rechtstreeks moet een USB kabel aansluiten ofwel via een webserver werken. Dit kan problemen geven in 'the field' en daarom zullen we gebruik maken van MAVProxy en Raspberry Pi.

2.2.1 MAVProxy

Wanneer men een multicopter wil besturen heeft deze commandosignalen nodig. Deze signalen kunnen verstuurd worden door ofwel een manuele controller zoals de DX7 ofwel door een Ground Control Station (GCS). MAVProxy is een open source GCS softwarepakket voor UAV's die gebaseerd zijn op het MAVLink protocol. Wij zullen verder ook van deze software gebruik maken omdat deze compatibel is met Python, de APM flight controller en licht genoeg is om op de Raspberry Pi te draaien.

Het feit dat het MAVLink protocol zeer generiek is, maakt dit tot zowel een sterkte als een zwakte. Men kan met behulp van MAVLink verschillende voertuigen aansturen zoals bijvoorbeeld een modelauto, een rover, een modelvliegtuig, verschillende types multicooters en meer. Dit heeft als gevolg dat er weinig tot geen documentatie te vinden is over specifieke toepassingen zoals een hexacopter.

Verschillende teams van open source developers proberen voor zulke specifieke toepassingen modules of bibliotheken te schrijven met een extra laag tussen de eindprogrammeur en de broncode tot gevolg. De broncode van veel van deze modules wordt gehost op platformen zoals Github. Deze platformen zorgen ervoor dat gebruikers van deze modules bugs kunnen melden en eventueel zelfs bugs kunnen oplossen en zo bijdragen tot de specifieke module. Het belang van contributers is meermaals duidelijk geworden tijdens ons onderzoek. Zonder hen kan men weken tot maanden met een bug zitten die ervoor zorgt dat bepaalde onderdelen van de programma's niet werken.

Om de weg naar autonomie stapsgewijs te laten verlopen wijzen we een aparte kanaal input toe aan het runnen van een programma. Dit stelt ons in staat de multicopter te laten opstijgen, besturen en vervolgens de autonomie in te schakelen zonder gevaar de controle over het toestel te verliezen. Wanneer de vlucht niet meer stabiel zou zijn of men de controle weer over wenst te nemen schakelen we deze kanaal input gewoon weer uit.

2.2.2 Opstartprocedure

Wanneer men de hexacopter en de Raspberry Pi opstart moet dit in een correcte volgorde gebeuren aangezien de Raspberry Pi zal zoeken naar de flight controller van de hexacopter en een error geven als deze niet gevonden wordt.

1. Start de Raspberry Pi op ontkoppelt van de flight controller
2. Zet de hexacopter aan
3. Verbind de flight controller en de Raspberry Pi via USB
4. change directory naar waar het programma staat
 \$ cd droneapi/hexacopter
5. start MAVProxy
 \$ mavproxy.py –baudrate 115200 –aircraft Hexacopter
6. start het gewenste python programma met:
 \$ api start *programma.py*

2.3 Pose Estimation

Het bepalen van de pose van de camera en indirect dus ook het voertuig waarop deze gemonteerd is noemt men binnen de beeldverwerking ook wel ”pose estimation”. Wanneer men pose estimation uitvoert krijgt men als resultaat een rotatie- en een translatie vector. Het nadeel van vele optische pose estimation algoritmes is dat deze ongevoelig zijn voor schaling en hierdoor een relatieve pose weergeven. Wanneer men echter een model van de wereld zou kunnen vastleggen en men het model ijkt met een gewenste maateenheid dan krijgt men het resultaat als een absolute pose. Het nadeel hiervan is dat men het model moet kunnen presenteren aan de camera wat men ”marker tracking” noemt. In een ideaal scenario zou de marker het schaal van de wereld moeten initiëren en heeft men deze vervolgens niet meer nodig. Dit zou mogelijk zijn moesten er zich echter geen cumulatieve meetfouten voordoen. Dit zou men echter kunnen oplossen aan de hand van herkalibratie met de marker of een ongeschaalde meting (laser of Time of Flight camera’s binnen hun meetbereik).

Een voorgestelde methode is om de marker ABCDE te gebruiken omdat deze makkelijk voor te stellen is, weinig te berekenen punten heeft en geen verschillende poses met identieke projecties heeft.

$$ABCDE = \begin{bmatrix} 0 & 0 & 10 & 10 & 20 \\ 0 & 10 & 10 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix} \quad (2.1)$$

Dit is de matrix voor het marker model ABCDE, zo geijkt dat het mooi op een A4-blad past. Zo blijft de marker nog zichtbaar wanneer men er boven vliegt. Elk punt wordt met een apparte kleur aangeduid die een andere range binnen het HSV bereik beschrijft. Het pose estimation algortime dat we hier gebruiken gaat eerst het model projecteren met standaard projectieparameters. Hierna worden de punten van de marker gezocht en de geschatte beeldpunten vergeleken met de gemeten beeldpunten. Vervolgens construeren we aan de hand van een schatting van de partieel afgeleide de parameters x waarvoor de beeldpunten y de kleinste fout geven tegenover het model. Deze verandering tellen we op bij de vorige schatting en bekomen we een nieuwe schatting van de projectieparameters. Deze loop wordt doorlopen tot de afwijking binnen een toegelaten gebied valt. Wanneer men voor de daarop volgende beelden stelt dat de pose slechts beperkte veranderingen ondergaat kan een accurate pose gevonden worden binnen een 2 tot 3 tal iteraties van de loop. Een hogere samptijd van de beelden resulteert in een kleinere verplaatsing en dus zijn er minder iteraties nodig om een voldoende accurate pose te berekenen.

Na het importeren van de math, Numpy en OpenCV bibliotheken definiëren we de functie die het model projecteert met de opgegeven projectieparameters. De eerste drie zijn de rotaties rond de x-, y- en z-as en de laatste drie zijn de translaties langs deze assen. Hiermee worden de resulterende rotatiematrix en translatiematrix samengesteld die op hun beurt de externe matrix vormen (3D naar 3D). Door het combineren met de interne cameramatrix krijgt men de projectiematrix die na normering de beeldpunten van het geprojecteerd model terug geeft via kolommatrix a (deze 1D matrix werkt gemakkelijker).

```
def fProject(x,P_M,K):  
    ax, ay, az, tx, ty, tz = x[0], x[1], x[2], x[3], x[4], x[5]  
  
    Rx =np.array([[1,0,0],[0,cos(ax),-sin(ax)],[0,sin(ax),cos(ax)]],float)  
    Ry =np.array([[cos(ay),0,sin(ay)],[0,1,0],[-sin(ay),0,cos(ay)]],float)  
    Rz =np.array([[cos(az),-sin(az),0],[sin(az),cos(az),0],[0,0,1]],float)  
    R = np.dot(Rz,np.dot(Ry,Rx))  
  
    t = np.array([[tx],[ty],[tz]])  
    Mext = np.concatenate((R,t),axis=1)  
  
    ph = np.dot(np.dot(K,Mext),P_M)  
    ph1 = np.divide(ph[0],ph[2])  
    ph2 = np.divide(ph[1],ph[2])  
  
    a = np.array([ph1,ph2])  
    a = np.reshape(a, np.shape(P_M)[1]*2, order='F')  
  
    return a
```

We definieren vervolgens een functie die de grootste blob van een bepaalde kleur gaat zoeken en ons het zwaartepunt hiervan geeft.

```
def find_center(color,colortname):
```

De camera wordt geopend en de volgende parameters worden geïnitieerd: de beginschatting van de projectieparameters x , de modelmatrix in homogene coordinaten PM en de interne cameramatrix (Logitech C310).

```
# Start of program
# Initialiseer camera, APM en matrices
cap = cv2.VideoCapture(0)

x=np.array([0,0,0,0,0,20])
P_M = np.array([[0,0,10,10,25],[0,15,15,0,0],[0,0,0,0,0],[1,1,1,1,1]])
f=813
cx=340
cy=224
K = np.array([[f,0,cx],[0,f,cy],[0,0,1]])
```

We starten de loop. Het beeld wordt ingelezen en Gaussiaans vervaagd om beeldruis te verminderen. Daarna converteren we het beeld naar HSV om een beter onderscheid te kunnen maken tussen de kleuren en benoemen we de range van iedere getrackte kleur. Vervolgens wordt het beeld getreshold op deze verschillende kleuren en vinden we de centerpunten van de blobs die we ook terugvinden in array y0.

```
# Beeldacquisitie
ret, img = cap.read()

# Beeldverwerking
im = cv2.blur(img,(3,3))

# 1. Filter/threshold op HSV range
hsv = cv2.cvtColor(im,cv2.COLOR_BGR2HSV)
yellow = cv2.inRange(hsv,np.array((15, 85, 85)), np.array((30, 255, 255)))
orange = cv2.inRange(hsv,np.array((0, 85, 85)), np.array((15, 255, 255)))
blue = cv2.inRange(hsv,np.array((50, 85, 85)), np.array((110, 255, 255)))
pink = cv2.inRange(hsv,np.array((120, 85, 85)), np.array((170, 255, 255)))
green = cv2.inRange(hsv,np.array((40, 85, 85)), np.array((50, 255, 255)))

# 2. Vind zwaartepunt van de blobs
c_yellow=find_center(yellow,"yellow")
c_blue=find_center(blue,"blue")
c_orange=find_center(orange,"orange")
c_green=find_center(green,"green")
c_pink=find_center(pink,"pink")

# 3. Maak een array van de getrackte punten
y0 = [[c_blue[0]],[c_blue[1]],[c_orange[0]],[c_orange[1]],
[c_pink[0]],[c_pink[1]],[c_yellow[0]],[c_yellow[1]],[c_green[0]],[c_green[1]]]
```

Bij elke iteratie wordt er ook een projectie gemaakt van het model met in array x de projectieparameters. Het resultaat hiervan wordt geplot. Vervolgens wordt de jacobiaan berekend door een benadering te maken van de partieel afgeleide van de projectie die als resultaat beeldpunten y geeft. De afwijking van de beeldpunten wordt berekend en vervolgens ook de afwijking van de projectieparameters die, indien de afwijking te groot is, wordt geupdate als input voor de volgende iteratie.

```

# Projecteer model
y = fProject(x,P_M,K)

for i in xrange(0,len(y),2):
    cv2.rectangle(img,(int(y[i])-2,int(y[i+1])-2),
                  (int(y[i])+2,int(y[i+1])+2),[0,96,144],-1)

e = 0.00001
J1 = (( fProject(x+[e,0,0,0,0,0],P_M,K) - y )/e)[np.newaxis,:].T
J2 = (( fProject(x+[0,e,0,0,0,0],P_M,K) - y )/e)[np.newaxis,:].T
J3 = (( fProject(x+[0,0,e,0,0,0],P_M,K) - y )/e)[np.newaxis,:].T
J4 = (( fProject(x+[0,0,0,e,0,0],P_M,K) - y )/e)[np.newaxis,:].T
J5 = (( fProject(x+[0,0,0,0,e,0],P_M,K) - y )/e)[np.newaxis,:].T
J6 = (( fProject(x+[0,0,0,0,0,e],P_M,K) - y )/e)[np.newaxis,:].T
J=np.column_stack((J1,J2,J3,J4,J5,J6))

dy = y0-(y[np.newaxis,:].T)
print "The residual error="+str(np.linalg.norm(dy))
pinvj=np.linalg.pinv(J)
dx=np.dot(pinvj,dy)

# Accuraatheidstest
if abs(np.linalg.norm(dx)/np.linalg.norm(x))< 0.000001:
    # Accuraat
else:
    # Update state estimation
    dx=np.transpose(dx)
    dx=dx[0,:]
    x=x+dx

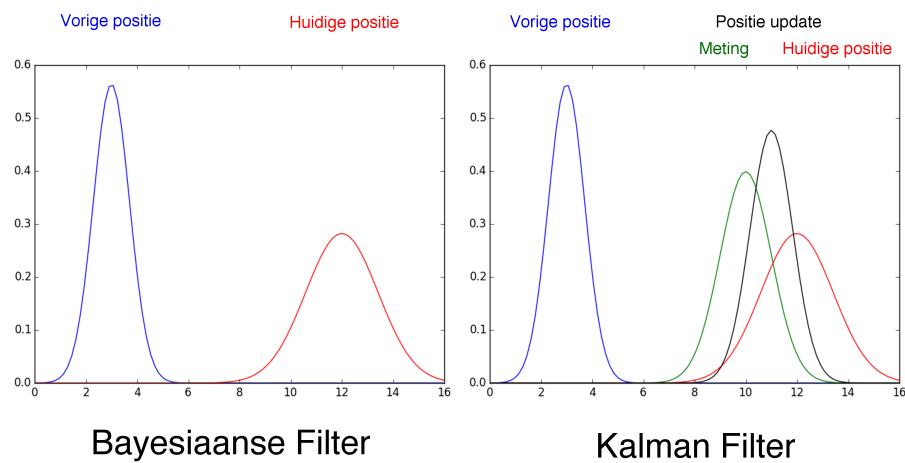
```

2.4 SLAM

2.4.1 Wat is SLAM?

Wanneer een robot zich door een onbekende omgeving verplaatst moeten enkele factoren gekend zijn om de verplaatsing nauwkeurig te laten verlopen. Het voertuig wordt meestal aangestuurd door verschillende motoren die samen een resulterende verplaatsing van het voertuig geven over een bepaald tijdsinterval. De motoren hebben echter een vaste onnauwkeurigheid die ervoor zorgt dat het aangestuurde commando en de werkelijke uitvoering ervan nooit perfect identiek zijn. We kunnen de plaats van het voertuig dus beschrijven als een waarschijnlijkheidsverdeling, net zoals de motoren een gemiddelde nauwkeurigheid hebben met een vaste standaardafwijking.

De robot kan zijn positie op het huidige tijdstip bepalen aan de hand van zijn vorige positie en de verplaatsing, met hun respectievelijke waarschijnlijkheden. Dit noemt men de Bayesiaanse filter. Indien er ook nog rekening wordt gehouden met een meting tegenover het assenstelsel waardoor men zich beweegt, bv. GPS, noemt men dit de Kalman Filter.



Figuur 2.6: Bayesiaanse en Kalman filter

Bij het gebruik van een Bayesiaanse of Kalman filter spreekt men van lokalisatie. Zolang de robot een waardevolle meting kan doen, kan hij een beeld schetsen van

waar hij zich denkt te bevinden. Het nadeel hiervan is dat wanneer de meting weg valt, bij elke positiestap de standaardafwijking groter wordt en men minder zeker wordt van de reële positie.

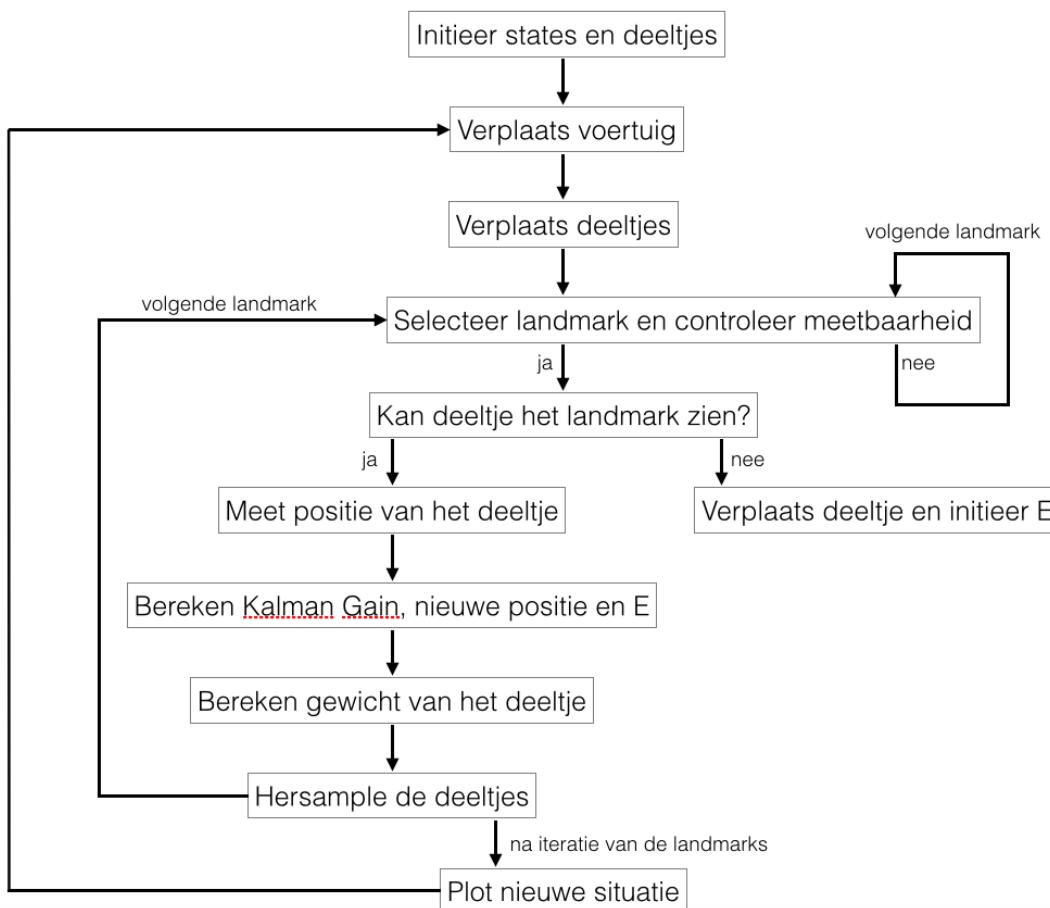
Wanneer men niet enkel aan lokalisatie doet maar tegelijk ook een kaart opbouwt spreekt men over Simultaneous Localization And Mapping (SLAM). De sterkte van SLAM algoritmen ligt er in dat de robot een kaart kan opbouwen van het onbekende gebied en zich aan de hand hiervan ook weer kan lokaliseren. Belangrijk bij een optische meting zijn de features die men waarneemt. Een buffer van enkele sequentiële beelden wordt voorzien om er zeker van te zijn dat alle features betrouwbaar genoeg zijn en we geen beeldruis meten. Dit zorgt voor een maximale correspondentie tussen de features gezien vanuit verschillende poses. [13]

2.4.2 Verschillende SLAM algoritmes

SLAM

Bij gewone SLAM algoritmes kan men gebruik maken van verschillende meettechnieken zoals sonar, lidar, feature tracking enz. Vervolgens wordt de nieuwe *state* geschat en kan er aan de hand van een filter zoals een Kalman filter of een Particle filter een state update gemaakt worden om de nauwkeurigheid van de huidige staat te verbeteren. We beschrijven kort de opbouw van een eenvoudig SLAM algoritme met voorgeprogrammeerde landmarks.

Bij het starten van het programma worden de benodigde matrices en een vast aantal deeltjes geïnitieerd. Hierna begint het voertuig met verplaatsen en geeft het zijn verplaatsings commando door. Dit wordt meegegeven aan de deeltjes zodat zij op hun beurt evenveel mogelijke hypotheses over de positie van het voertuig en de structuur van de wereld kunnen toetsen als er deeltjes zijn. Tegenover elke landmark of feature wordt de positie gemeten, waarbij men rekening houdt met de waarschijnlijkheid op een juiste meting. Hiervoor gebruikt men de Kalman Gain en het gewicht van het deeltje. [14]



Figuur 2.7: Flowchart SLAM algoritme

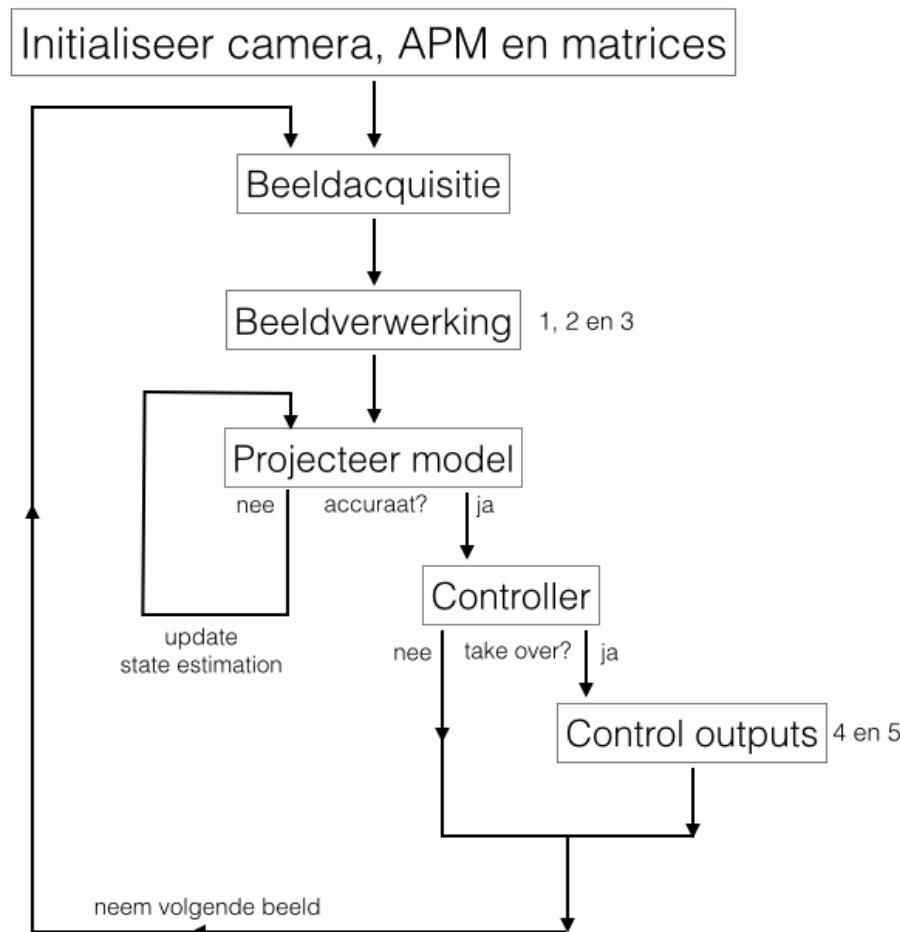
PTAM

Parallel Tracking And Mapping is een SLAM techniek waarbij men gebruik maakt van *visuele odometrie*. Het algoritme gaat op zoek naar betrouwbare kenmerkende punten en past feature tracking toe op de sequentiële beelden. Omdat het algoritme werkt met visuele odometrie kan het gebruikt worden als een SLAM algoritme toegepast op één enkele RGB camera. [15]

LSD-SLAM

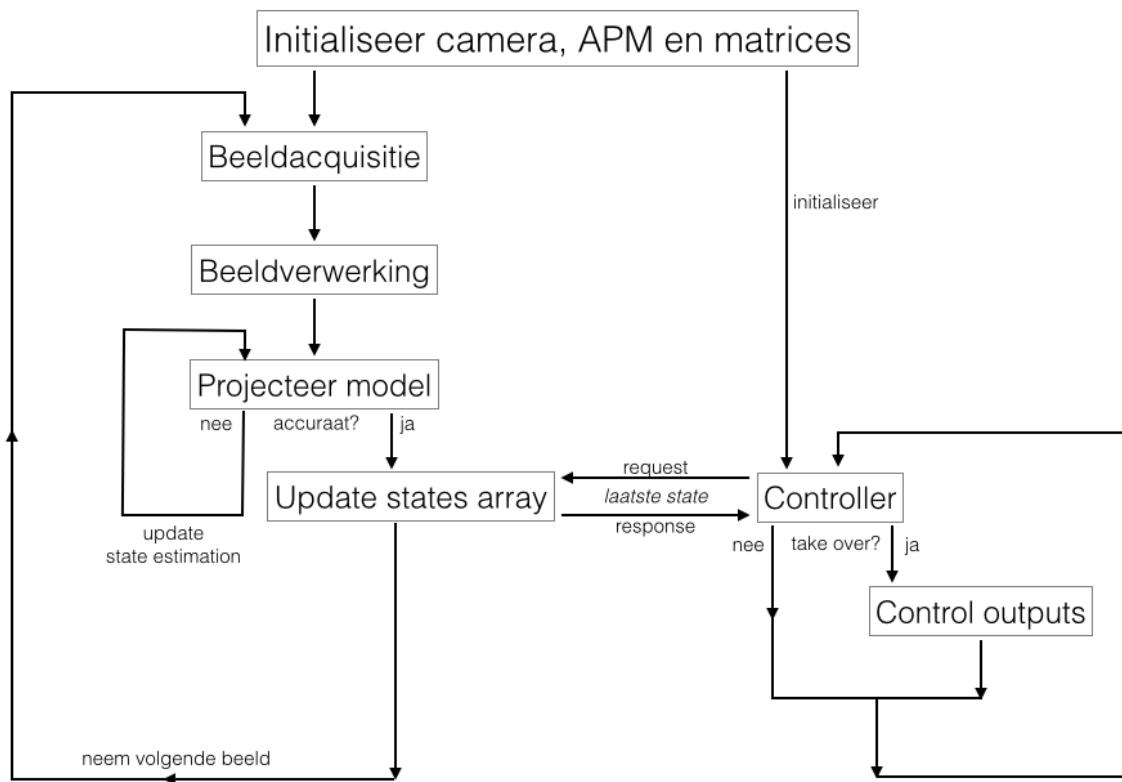
Large-Scale Direct Monocular-SLAM is een techniek die gepubliceerd is in 2014 aan de Technische Universiteit van München en uitermate veel toekomst heeft aangezien een volledige map kan worden opgebouwd met slechts één RGB camera. Het algoritme berust op featureless tracking en kent hierdoor veel voordelen tegenover Parallel Tracking And Mapping. De techniek van Parallel Tracking And Mapping is een doelgerichte toepassing voor het gebruik van RGB camera's bij SLAM problemen maar omdat PTAM echter berust op features gaat veel informatie van de beelden verloren die eigenlijk kan gebruikt worden om een point cloud van te reconstrueren. Waar andere algoritmen vaak problemen ervaren bij schalingen van de wereld is LSD-SLAM 'scale-aware'. Tot slot zijn de onderzoekers erin geslaagd het algoritme realtime te laten runnen op een moderne smartphone wat laat zien dat deze techniek slechts in beperkte maten reken-intensief is. Een nadeel is echter nog steeds dat hoewel het algoritme 'scale-aware' is men werkt met een relatieve maateenheid. [16]

2.5 Het framework



Figuur 2.8: Schema van het huidige programma

In het huidige programma worden alle commando's sequentieel uitgevoerd. Dit heeft als gevolg dat de snelheid van het programma afhangt van de snelheid van de processor en hoe snel elke *loop* hiermee kan doorlopen worden. De totale runtime van één hoofdloop telt cumulatief op. Alhoewel dit een goede beginsituatie is, indien men de looptijd kan beperken, is er ruimte voor verbetering.



Figuur 2.9: Schema van het mogelijke toekomstige programma

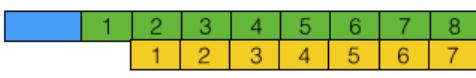
Een mogelijkheid tot verbetering dient zich aan in de vorm van multiprocessing. Aangezien de Raspberry Pi 2 B+ uitgerust is met 4 cores in plaats van slechts 1, kan men aparte taken toewijzen aan verschillende cores. Zo zou men bijvoorbeeld 1 van de cores kunnen toewijzen aan beeldacquisitie, 1 aan beeldverwerking, 1 aan de regeling van controller outputs en 1 aan het draaien van de command shell met het MAVLink communicatie protocol. Uiteraard kan men hiermee spelen voor moest men hier nog mogelijkheid tot SLAM aan willen toevoegen. In een eerste stap zou men de controller al van de rest van het programma kunnen scheiden. Dit zodat de controller sneller kan reageren en men aan het volgende beeld kan beginnen alvorens de controller alle outputs heeft aangestuurd. We spreken dan over 2 parallelle threads of processen.

IDLE, de interpreter van Python, is van zichzelf *thread blocking* wat betekend dat slechts 1 thread tegelijk kan worden uitgevoerd binnen de interpreter (Global Interpreter Lock). Indien men meerdere threads simultaan wil laten lopen kan men gebruik maken van de multiprocessing module die als truck meerdere IDLE inter-

preters tegelijk opent die elk hun eigen thread uitvoeren en data opslaan in globale variabelen.

Single threaded: 

VS

Multi threaded:
(multiprocessing) 



Figuur 2.10: Singlethreading vs Multithreading

Een voordeel van multithreading wordt duidelijk wanneer men de beeldacquisitie en beeldverwerking opdeelt in 2 aparte threads. Alle opgenomen beelden worden in een globaal toegankelijke array ingeladen waarnaar het beeldverwerkingsprocess een request stuurt en als response het laatste beeld ontvangt. Hierdoor kan men het proces van beeldaquisitie + beeldverwerking versnellen tot 200% van de oorspronkelijke snelheid.

2.6 Meetresultaten

Het rollniveau wordt aangeduid met de rode lijn en het pitchniveau met de groene lijn. De hexacopter hangt bij aanvang van de proef in onbalans in het frame. Wanneer het programma start zal de hexacopter een bepaalde houding aannemen maar deze wordt niet volledig behaald door restricties van het frame.

Bij meetproef 1 verhogen we gedurende 13 seconden en volgens een lineaire verloop de kanaal output van beide kanalen van -37 graden tot +31 graden.

De grafieken stoppen af omdat het frame in rotatiehoeken beperkt is.

Als we het throttleniveau vergelijken met het roll- en pitchniveau zien we als eerste dat dit beeld iets vroeger is genomen maar ook over 13 seconden de throttle hoog aanstuurt. De dip in het throttleniveau die overeen komt met een lage roll en

pitchniveau duidt op een beperkte hoogte correctie die wordt toegepast naarmate de roll of pitch hoek vergroot.

Tijdens meetproef 2 laten we de roll hoek variëren van -25 tot +20 graden terwijl we de pitch hoek aansturen met een constante waarde.

Tijdens meetproef 3 laten we de pitch hoek variëren van -25 tot +20 graden terwijl we de roll hoek aansturen met een constante waarde. Uit de proeven kunnen we besluiten dat hoewel we een kanaal een constante waarde geven deze kan worden beïnvloed door de rotatie om een andere as.

In tegenstelling tot het throttleniveau bij meetproef 1 blijft het throttleniveau gedurende meetproef 2 en 3 constant. De lichte stijging in het begin is te wijten aan het opstarten van de hexacopter.

Bij meetproef 4 laten we de hexacopter 4 setpunten bereiken en meten we de afwijking. De setpunten voor beide rotatieassen zijn 0 en 10 graden en zo kan men zien dat de aansturing van roll en pitch overeenkomende resultaten geeft. Het niet volledig behalen van de theoretische setpunten heeft te hoogstwaarschijnlijk maken met de onvolmaaktheid van het testframe.

Bij meetproef 5 doorlopen we de setpunten 2 maal om te besluiten dat we een goede herhaalbaarheid bekomen.



Figuur 2.11: Roll- en Pitchniveau tijdens meetproef 1



Figuur 2.12: Throttleniveau tijdens meetproef 1



Figuur 2.13: Roll- (en Pitch-) niveau tijdens meetproef 2



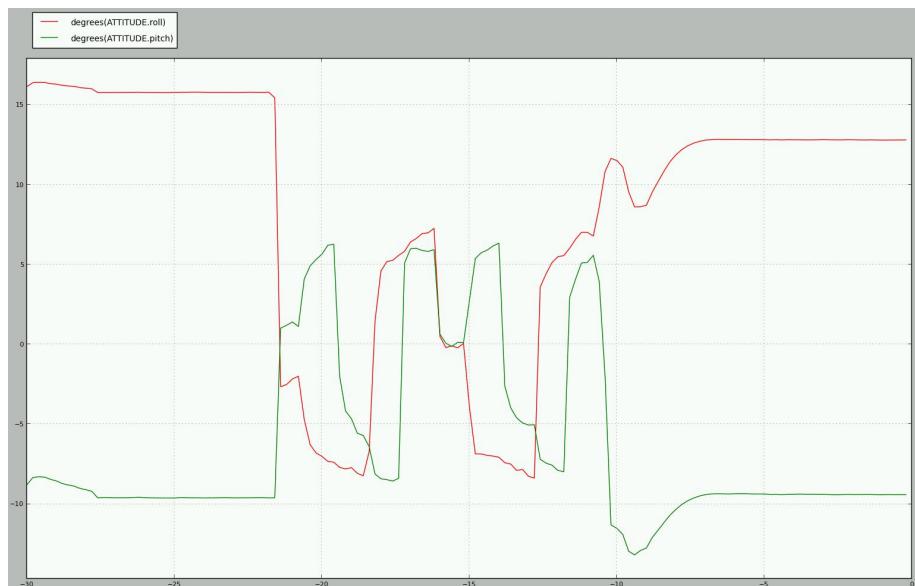
Figuur 2.14: Pitch- (en Roll-) niveau tijdens meetproef 3



Figuur 2.15: Throttleniveau tijdens meetproef 2 en 3



Figuur 2.16: Roll- en Pitchniveau tijdens meetproef 4



Figuur 2.17: Roll- en Pitchniveau tijdens meetproef 5

2.7 Alternatieve methoden

2.7.1 GPS

Wanneer men UAV's wil gebruiken bij het controlleren of bewaken van gekende terreinen en onveranderend gebieden, kan men kiezen om op voorhand een vluchtplan samen te stellen aan de hand van waypoints (GPS locaties). Mission Planner beschikt ook over deze methode. Op deze manier kan men op voorhand zelf de *region of interest* gaan bepalen.

2.7.2 Optical Flow

Een gekende tak binnen visietechnologie is deze van *optical flow*. Hierbij worden bepaalde features van de sequentiële beelden van de camera vergeleken om de relatieve translaties en rotaties van de camera te achterhalen. Deze techniek heeft als voordeel dat wanneer men betrouwbare features weet te selecteren enkel met een RGB camera gewerkt kan worden. Optical flow faalt echter wanneer de omgeving te dynamisch is en men tegenover een fout ijkpunt begin te meten (denk hierbij aan een drukke straat met veel voetgangers en auto's).

2.7.3 Motion capture

Tal van onderzoekslabo's werken met *motion capture* systemen waarbij enkele camera's verspreid over de vliegzaal de positie van de multicopter meten. Met behulp van reflecterende markers op het voertuig kunnen de beelden van de verschillende camera's vergeleken worden om de pose van het voertuig te berekenen. Een voordeel van deze techniek is dat de beeldverwerking op een krachtige centrale computer gebeurt en dit de snelheid van de beeldverwerking en de hieraan gekoppelde nauwkeurigheid van het voertuig verhoogt. De verhoogde snelheid en nauwkeurigheid zorgen ook voor de mogelijkheid tot acrobatische manoeuvres. Bovendien kunnen meerdere voertuigen tegelijk getracked worden wat taken mogelijk maakt waarbij samengewerkt moet worden. Een groot nadeel is echter dat motion capture systemen

duur zijn, in de grootteorde van 5 000-25 000 euro afhankelijk van de gewenste specificaties. Ook kunnen deze systemen enkel indoor gebruikt worden wat het aantal toepassingen aanzienlijk verminderd. [17]

2.7.4 Motion from structure

Binnen het labo van Op3Mech werkt men aan een techniek genaamd *Motion from structure*. Dit houdt in dat men datapunten uit sequentiële beelden van een ToF camera met elkaar vergelijkt en hieruit de meest waarschijnlijke translatie en rotatie extraheert. Motion from structure heeft als voordeel dat men, zolang men binnen het meetbereik blijft, met absolute meetwaarden rekent en er dus geen schaalfactor meespeelt binnen de berekeningen. Een ander voordeel is dat deze techniek een oplossing biedt waar RGB camera's falen. Zo kan men vliegen bij condities waar het zicht beperkt is bijvoorbeeld 's nachts of bij mist. [18]

2.8 Mogelijke toepassingen

2.8.1 Landbouw

Waar technologie niet meer uit de landbouwsector weg te denken is, zullen multicopters weldra ook een grootschalige intrede maken. Door het gebruik van verschillende camera's worden vele tijdrovende toepassingen nu overgenomen door controlerondes van de multicopters. Zo kunnen met een RGB camera visuele inspecties worden gedaan van de site. Een ToF camera kan de oogst voorspellen, tellen en uitvoeren. Een multispectraal of red edge camera kan de gezondheid en reipingsfase van de gewassen controleren en een infrarood camera kan aan irrigatie planning doen.

2.8.2 Mijnbouw

Veiligheid speelt een grote rol wanneer men mijn werken doet. Een multicopter kan een prospectie doen van een mogelijke mijn en deze in kaart brengen zonder dat hier mijnwerkers bij in gevaar moeten komen. Nadien kunnen er updates gemaakt worden van de mijn-map om te controleren op erosie. Dankzij deze updates kan men

een vergelijking doen van voor en na situaties en het volumetrisch verschil berekenen dat gedurende de tussenperiode gemijnd is.

2.8.3 Bouw en wegenwerken

Multicopters zijn uitermate inzetbaar voor technische controle op moeilijk bereikbare plaatsen zoals onder bruggen, op hoogspanningsmasten en bij windturbines. Met behulp van UAV's kan een pointcloud gemaakt worden van de constructie, kan er gecontroleerd worden op defecten of ongewenste situaties. Projectmanagement wordt ook mogelijk aan de hand van visuele inspectie op basis van pointcloud mapping. Indien men hoge resolutie beelden kan maken is dit een belangrijke bron aan informatie voor landmeters en project managers. [19] [20]

2.8.4 GIS

Geographic Information Systems heeft aan UAV's een enorme aanwinst. GIS researchers hoeven niet langer te berusten op satellietbeelden maar kunnen zelf kiezen welke gebieden ze in kaart brengen en monitorren. Zo kunnen ook het verkeer of het vervoer van grondstoffen getraceerd, gerapporteerd en geanalyseerd worden.

2.8.5 Security en bewaking van terreinen

Wanneer men 's nachts een industrieterrein moet bewaken kan dit een grote klus zijn. Multicopters kunnen hierbij helpen door ingeplande routes te vliegen en met behulp van IR of ToF camera's eventuele onbevoegde personen op de terreinen te detecteren. [21]

2.8.6 Politiewerk

Als na een ongeval de politie ter plaatse komt moet er een situatieschets gemaakt worden. Dit kan nu veel sneller dankzij de mapping mogelijkheden van UAV's. Bovendien kan men deze toestellen inzetten tijdens patrouille werk of tijdens een achtervolging waar men vroeger steun van een helikopter nodig had.

2.8.7 Zoek en reddingsacties

Getroffen gebieden kunnen snel in kaart gebracht worden om de schade op te meten en herstellingswerken te plannen. Zulke gebieden zijn niet altijd goed bereikbaar en met behulp van IR of RGB camera's kan men zo snel mogelijk ter plaatsen zijn en in kaart brengen waar mogelijke slachtoffers zich bevinden nog voor een reddingsteam gearriveerd is.

2.8.8 Onbemand cargo transport

Indien men hulp wil bieden aan getroffen of moeilijk bereikbare gebieden kan men de hulp van UAV's inschakelen. Door bijna permanente toegang via het luchtruim kunnen EHBO-sets en levensmiddelen snel voorzien worden. Ook voor dagdagelijkse levering zouden deze toestellen kunnen worden ingezet en hier doen grote retailbedrijven zoals Amazon en DHL ook al onderzoek naar. [22] [23]

2.8.9 Entertainment industrie

Waar multicopters kunnen gebruikt worden om controlebeelden mee te maken kunnen ze ook gebruikt worden om cinematografisch knappe beelden mee te maken. De reden dat sommige multicopters een goede adoptie behaalden bij particuliere consumenten is omdat ze uitstekende luchtbeelden kunnen maken. Binnen de entertainment sector is dit een goedkope oplossing om filmen met een helicopter te vervangen maar ook om camerahoeken te bereiken die voordien onmogelijk waren. Zo kan een multicopter bijvoorbeeld boven een peloton wielrenners zweven en een overzicht hebben van de grote ervan, of een skier gedurende zijn afvalding in een vloeiende lijn in beeld brengen. Dankzij slimmere autonome controllers en een fijner afgesteld vlucht pad zijn de mogelijkheden beperkt tot de verbeelding van de regisseur. [24]

Hoofdstuk 3

Besluit

Tijdens het schrijven van deze thesis ben ik op zoek gegaan naar antwoorden op de onderzoeksfrage: "Welke elementaire bouwstenen zijn nodig tot het bekomen van een autonome vlucht?". Na het probleem op te delen in opeenvolgende blokken ben ik deze stap voor stap beginnen onderzoeken. Het proces begint bij beeldcaptatie, gevuld door beeldverwerking en tot slot het aansturen van het voertuig.

Bij het onderdeel van beeldcaptatie is er gekozen om toe te spitsen op een RGB camera. Deze camera vereiste de minste drivers en is het makkelijkst voor handen. Hierdoor konden we een point based tracking algoritme gebruiken om de pose van het voertuig te bepalen. Dit algoritme samen met Python bleek te werken maar kent nog vele beperkingen. Zo was het ondermeer niet snel genoeg, onbetrouwbaar zodra er één of meerdere punten uit het scherm verdwenen en slechts met de aanwezigheid van de marker te gebruiken. Om deze reden wordt er een framework of combinatie van de volgende technologieën voorgesteld om in de toekomst meerdere visie algoritmen mee te testen die specifiek bedoeld zijn voor multicopters. Een voorstel voor de framework naam is MARO naar:

- MAVProxy (Python)
- Ardupilot
- Raspberry Pi
- OpenCV (Python)

Zo hoeft men niet, telkens men een nieuw visie algoritme wil testen, van nul af aan te beginnen. De code voor de sturing kan hergebruikt worden en ook kan hier nog SLAM aan toegevoegd worden. Op deze manier kunnen vakmensen in hun gebied binnen het visiedomein experimenteren en hun kennis bijdragen aan autonome besturing van multicopters met behulp van visietechnologie.

Vanaf het moment dat de visie algoritmen zijn gevalideerd, kan men de code naar C/C++ vertalen om een hogere performantie te bekomen. Er moet dan ook gebruik gemaakt worden van een MAVLink bibliotheek voor C/C++ om de multicopter te kunnen aanspreken.

In het MARO framework zijn alle elementaire bouwstenen voor autonome vlucht aanwezig en kan men verder onderzoek doen naar de alternatieve visie algoritmen. De bouwstenen zijn:

- Initialisering van alle parameters
- Beeldcaptatie
- Beeldverwerking en lokalisatie
- Aansturing van het voertuig
- State feedback door de flight controller

De code kan worden teruggevonden op de CD-ROM achteraan de cursus en op <https://github.com/YannisDC/AutonomousMulticopters>.

Bibliografie

- [1] x86 Assembly Guide. <http://www.cs.virginia.edu/~evans/cs216/guides/x86.html>, 2014. [Online; bezocht 27-September-2014].
- [2] Arm vs x86 processors: What's the difference? <http://www.brighthub.com/computing/hardware/articles/107133.aspx>, 2014. [Online; accessed 27-September-2014].
- [3] Comparison of single-board computers. http://en.wikipedia.org/wiki/Comparison_of_single-board_computers/, 2014. [Online; accessed 20-October-2014].
- [4] Raspberry pi website. <http://www.raspberrypi.org/>, 2014. [Online; accessed 22-September-2014].
- [5] Intel galileo datasheet. <http://www.intel.com/content/dam/www/public/us/en/documents/datasheets/galileo-g2-datasheet.pdf>, 2014. [Online; accessed 20-October-2014].
- [6] Pandaboard datasheet. http://pandaboard.org/sites/default/files/board_reference/ES/Panda_Board_Spec_DOC-21054_REV0_1.pdf, 2014. [Online; accessed 20-October-2014].
- [7] The yocto project. <https://www.yoctoproject.org/>, 2014. [Online; accessed 11-November-2014].
- [8] Richard Szeliski. *Computer Vision: Algorithms and Applications*. Springer.
- [9] Eeng 512 / csci 512 - computer vision. <http://inside.mines.edu/~whoff/courses/EENG512/>, 2014. [Online; accessed 27-October-2014].

- [10] Opencv website. <http://opencv.org/>, 2014. [Online; accessed 24-September-2014].
- [11] M. Scott Shell. *An introduction to Numpy and Scipy*.
- [12] Camera calibration opencv. http://opencv-python-tutroals.readthedocs.org/en/latest/py_tutorials/py_calib3d/py_calibration/py_calibration.html, 2014. [Online; accessed 27-October-2014].
- [13] Søren Riisgaard and Morten Rufus Blas. Slam for dummies: A tutorial approach to simultaneous localization and mapping. Technical report, 2005.
- [14] Simple slam matlab programma. <https://github.com/randvoorhies/SimpleSLAM>, 2015. [Online; accessed 11-March-2015].
- [15] Parallel tracking and mapping. <http://www.robots.ox.ac.uk/~gk/publications.html#2007ISMAR>, 2014. [Online; accessed 11-November-2014].
- [16] Computer vision group technical university of munich. <http://vision.in.tum.de/publications>, 2014. [Online; accessed 11-November-2014].
- [17] Optitrack motion capture systems. <https://www.optitrack.com/systems/>, 2015. [Online; accessed 11-April-2015].
- [18] Op3mech university of antwerp. <https://www.uantwerpen.be/en/rg/op3mech/>, 2014. [Online; accessed 11-November-2014].
- [19] Service drone. <http://www.service-drone.com/en/production/surveying>, 2015. [Online; accessed 16-April-2015].
- [20] Bouwkroniek. Drones op komst in de bouwsector. *Bouwkroniek*, (16):5–8, 2015.
- [21] Micro drones. <http://www.microdrones.com/en/applications/>, 2015. [Online; accessed 19-April-2015].
- [22] Matternet. <http://mttr.net>, 2015. [Online; accessed 1-May-2015].
- [23] VOKA. De mogelijkheden van drones. *Voka Tribune*, 9:16–17, 2015.
- [24] Argus vision. <http://argusvision.be/av/toepassingen>, 2015. [Online; accessed 9-May-2015].

DEEL II

Bijlagen

Bijlage A

Formularium

3D naar 3D

$${}^A P = {}^A_B R * {}^B P + {}^B t_{Aorg} \quad (\text{A.1})$$

$${}^A P = {}^A_B H * {}^B P \quad (\text{A.2})$$

$${}^A_B H = \begin{bmatrix} {}^A_B R & {}^A t_{Borg} \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (\text{A.3})$$

$$\begin{bmatrix} u_1 & u_2 & \dots & u_n \\ v_1 & v_2 & \dots & v_n \\ w_1 & w_2 & \dots & w_n \\ s_1 & s_2 & \dots & s_n \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \\ 0 & 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} x_1 & x_2 & \dots & x_n \\ y_1 & y_2 & \dots & y_n \\ z_1 & z_2 & \dots & z_n \\ 1 & 1 & \dots & 1 \end{bmatrix} \quad (\text{A.4})$$

$$x_n = \frac{u_n}{s_n}, y_n = \frac{v_n}{s_n}, z_n = \frac{w_n}{s_n}, gewicht = \frac{s_n}{s_n} = 1 \quad (\text{A.5})$$

3D naar 2D projectie

$$B = K * \begin{bmatrix} R & T \end{bmatrix} * M \quad (\text{A.6})$$

$$\begin{bmatrix} u_1 & u_2 & \dots & u_n \\ v_1 & v_2 & \dots & v_n \\ s_1 & s_2 & \dots & s_n \end{bmatrix} = \begin{bmatrix} f_x & \gamma & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \end{bmatrix} * \begin{bmatrix} x_1 & x_2 & \dots & x_n \\ y_1 & y_2 & \dots & y_n \\ z_1 & z_2 & \dots & z_n \\ 1 & 1 & \dots & 1 \end{bmatrix} \quad (\text{A.7})$$

$$i_n = \frac{u_n}{s_n}, j_n = \frac{v_n}{s_n}, gewicht = \frac{s_n}{s_n} = 1 \quad (\text{A.8})$$

Bijlage B

Python en OpenCV installeren op Windows

Python op pc via internet

- Ga naar <https://www.python.org/> en onder download klik op 2.7.6
- Dubbelklik op de gedownloaden file

Python op pc via bestandenmap

- Dubbelklik op python-2.7.8 en installeer

Portable Python met enkele vooraf geïnstalleerde bibliotheken

- Ga naar <http://portablepython.com/> en download de versie naar keuze of selecteer uit map met bestand
- Dubbelklik op de file PortablePython_2.7.6.1 of PortablePython_3.2.5.1 en kies USB als doellocatie

Installeren van OpenCV voor Python

- Download de zipfile unzip en zoek cv2.pyd in opencv/build/python/2.7/x86/cv2.pyd
- Kopieer cv2.pyd en plak deze in Python/Lib/site-packages

Installeren van SciPy voor Python

- Download via <http://sourceforge.net/projects/numpy/files/latest/download?source=files>
- Dubbelklik en installer

Installeren van andere bibliotheken voor Python

- Surf naar <https://pypi.python.org/>
- Zoek de gewenste bibliotheek
- Download en unzip de zipfile
- Kopieer de map en plak deze in de Python home directory
- Open de command line en cd naar de deze map tot je in de map met de setup.py file zit.
- Run: python setup.py install

Bijlage C

IDS uEye USB 2.0 camera op Raspberry pi

Via IDS website

Installeren van de Raspbian Image via de IDS website: <http://store-en.ids-imaging.com/xs-raspberrypi.html>

Installeren van alle benodigde onderdelen via de commandline

- sudo apt-get update
- sudo apt-get install python-dev python-pip python-numpy python-matplotlib

Installeren van Cython

- wget <https://pypi.python.org/packages/source/C/Cython/Cython-0.21.tar.gz>
- tar xvzf Cython-0.21.tar.gz
- cd Cython-0.21
- sudo python setup.py install

Installeren van de IDS wrapper voor Python

- surf naar <https://github.com/ncsuarc/ids>
- Downloaden en unzippen
- Map kopiëren naar root van RPi /home/pi
- Cd in de directory waar de setup file staat
- sudo python setup.py install

Vervolgens kan men python scripts schrijven waaronder een voorbeeld in bijlage. Het programma kan worden gerund via de comandline met: sudo python CaptureIDS.py Een foto wordt gemaakt en deze wordt opgeslagen in de zelfde map als het programma (root).

```
import ids
import cv2

cam = ids.Camera()
cam.color_mode = ids.ids_core.COLOR_RGB8 # Gets image in RGB format
cam.exposure = 5                         # Set initial exposure to 5ms
cam.auto_exposure = True
cam.continuous_capture = True            # Start image capture
img, meta = cam.next()                  # Get image as Numpy array
bgr_img = cv2.cvtColor(img, cv2.COLOR_RGB2BGR)
cv2.imwrite('cv2.jpg', bgr_img)
```

Bijlage D

Broncode Programma's

LogitechC310 backprojection navigation using Jacobian estimation

```
from math import *
import numpy as np
import cv2
import time
from droneapi.lib import VehicleMode, Location

def fProject(x,P_M,K):
    ax, ay, az, tx, ty, tz = x[0], x[1], x[2], x[3], x[4], x[5]

    Rx =np.array([[1,0,0],[0,cos(ax),-sin(ax)],[0,sin(ax),cos(ax)]],float)
    Ry =np.array([[cos(ay),0,sin(ay)],[0,1,0],[-sin(ay),0,cos(ay)]],float)
    Rz =np.array([[cos(az),-sin(az),0],[sin(az),cos(az),0],[0,0,1]],float)
    R = np.dot(Rz,np.dot(Ry,Rx))

    t = np.array([[tx],[ty],[tz]])
    Mext = np.concatenate((R,t),axis=1)

    ph = np.dot(np.dot(K,Mext),P_M)
    ph1=np.divide(ph[0],ph[2])
```

```
ph2=np.divide(ph[1],ph[2])

a=np.array([ph1,ph2])
a=np.reshape(a, np.shape(P_M)[1]*2, order='F')

return a

def find_center(color,colordname):
    contours,hierarchy =
        cv2.findContours(color, cv2.RETR_LIST, cv2.CHAIN_APPROX_SIMPLE)

    if len(contours)>0:
        # find biggest blob
        max_area = 0
        for contour in contours:
            area = cv2.contourArea(contour)
            if area > max_area:
                max_area = area
                best_blob = contour
        if max_area > 10:
            # find centroid
            M = cv2.moments(best_blob)
            cx = int(M['m10']/M['m00'])
            cy = int(M['m01']/M['m00'])
            cv2.circle(im,(cx,cy),3,255,-1)
            (xcenter,ycenter),(MA,ma),angle = cv2.fitEllipse(best_blob)
        else:
            print colordname+" too small"
            return(int(xcenter),int(ycenter))
    else:
        print colordname+" not found!"
        return(0,0)

def nothing(xx):
    pass

class Controller():
```

```
def __init__(self, pGain):
    self.leanAngleMax = 45.0 # degrees
    self.yawRate = 200.0 # degrees per second
    self.rollValueMin = 1132
    self.rollValueMid = 1525
    self.rollValueMax = 1927
    self.pitchValueMin = 1126
    self.pitchValueMid = 1521
    self.pitchValueMax = 1921
    self.yawValueMin = 1128
    self.yawValueMid = 1525
    self.yawValueMax = 1923
    self.throttleValueMin = 1123
    self.throttleValueMax = 1921
    self.Kp = pGain
    self.Ki = 0.5
    self.Kd = 0.5

    self.rollP = float(self.rollValueMax - self.rollValueMin) / (2 *
        self.leanAngleMax) # roll parameter, PWM change per degree
    self.pitchP = float(self.pitchValueMax - self.pitchValueMin) / (2
        * self.leanAngleMax) # pitch parameter, PWM change per degree
    self.yawP = float(self.yawValueMax - self.yawValueMin) / (2 *
        self.yawRate) # parameter, PWM change per degree/second

def control_angles(self, roll, pitch, yaw, x, y, z):
    rollDesired = int(round( self.rollValueMid - (y/100 * self.Kp) *
        self.rollP ))
    pitchDesired = int(round( self.pitchValueMid - (x/100 * self.Kp) *
        self.pitchP ))
    yawDesired = self.yawValueMid
    throttleDesired = 1300
    return rollDesired, pitchDesired, yawDesired, throttleDesired
```

```
# Start of program
# Initialiseer camera, APM en matrices
cap = cv2.VideoCapture(0)

x=np.array([0,0,0,0,0,20])
P_M = np.array([[0,0,10,10,25],[0,15,15,0,0],[0,0,0,0,0],[1,1,1,1,1]])
f=813
cx=340
cy=224
K = np.array([[f,0,cx],[0,f,cy],[0,0,1]])

api = local_connect()
print "connected"

v = api.get_vehicles()[0]
print "vehicle found"

if v.mode.name == "INITIALISING":
    print "Vehicle still booting, try again later"
    return

# Initialiseer controller
c = Controller(2)

while(True):
    # Beeldacquisitie
    ret, img = cap.read()

    # Beeldverwerking
    im = cv2.blur(img,(3,3))

    # 1. Filter/threshold op HSV range
    hsv = cv2.cvtColor(im,cv2.COLOR_BGR2HSV)
    yellow = cv2.inRange(hsv,np.array((15, 85, 85)), np.array((30, 255,
    255)))
    orange = cv2.inRange(hsv,np.array((0, 85, 85)), np.array((15, 255,
    255)))
```

```
blue = cv2.inRange(hsv,np.array((50, 85, 85)), np.array((110, 255,
255)))
pink = cv2.inRange(hsv,np.array((120, 85, 85)), np.array((170, 255,
255)))
green = cv2.inRange(hsv,np.array((40, 85, 85)), np.array((50, 255,
255)))

# 2. Vind zwaartepunt van de blobs
c_yellow = find_center(yellow,"yellow")
c_blue = find_center(blue,"blue")
c_orange = find_center(orange,"orange")
c_green = find_center(green,"green")
c_pink = find_center(pink,"pink")

# 3. Maak een array van de getrackte punten
y0 = [[c_blue[0]], [c_blue[1]], [c_orange[0]], [c_orange[1]], [c_pink[0]]
,[c_pink[1]], [c_yellow[0]], [c_yellow[1]], [c_green[0]], [c_green[1]]]

# Projecteer model
y = fProject(x,P_M,K)

for i in xrange(0,len(y),2):
    cv2.rectangle(img,(int(y[i])-2,int(y[i+1])-2)
,(int(y[i])+2,int(y[i+1])+2),[0,96,144],-1)

e = 0.00001
J1 = (( fProject(x+[e,0,0,0,0,0],P_M,K) - y )/e)[np.newaxis,:,:].T
J2 = (( fProject(x+[0,e,0,0,0,0],P_M,K) - y )/e)[np.newaxis,:,:].T
J3 = (( fProject(x+[0,0,e,0,0,0],P_M,K) - y )/e)[np.newaxis,:,:].T
J4 = (( fProject(x+[0,0,0,e,0,0],P_M,K) - y )/e)[np.newaxis,:,:].T
J5 = (( fProject(x+[0,0,0,0,e,0],P_M,K) - y )/e)[np.newaxis,:,:].T
J6 = (( fProject(x+[0,0,0,0,0,e],P_M,K) - y )/e)[np.newaxis,:,:].T
J=np.column_stack((J1,J2,J3,J4,J5,J6))

dy = y0-(y[np.newaxis,:,:].T)
print "The residual error="+str(np.linalg.norm(dy))
pinvj=np.linalg.pinv(J)
```

```
dx=np.dot(pinvj,dy)

# Accuraatheidstest
if abs(np.linalg.norm(dx)/np.linalg.norm(x))< 0.000001:
    # Accuraat
    rollPWM, pitchPWM, yawPWM, throttlePWM = c.control_angles(y[0],
        y[1], y[2], y[3], y[4], y[5])

    # Check of autonome navigatie wordt toegestaan door de piloot
    if (v.mode.name == "STABILIZE") and (v.channel_readback['6'] >
        1600) :
        print "Overriding a RC channel"
        v.channel_override = { "1" : rollPWM, "2" : pitchPWM, "3" :
            throttlePWM, "4" : yawPWM }
        v.flush()
        print "Current overrides are:", v.channel_override
        print "RC readback:", v.channel_readback
        time.sleep(1)

else:
    # Update state estimation
    dx=np.transpose(dx)
    dx=dx[0,:]
    x=x+dx

# Sluit camera's en eventuele openstaande windows
cap.release()
cv2.destroyAllWindows()
```

Basic SLAM algoritme

```
import numpy as np
from numpy.linalg import inv
from numpy import transpose
from numpy import dot
from math import *
import matplotlib.pyplot as plt

def moveParticle(xt, m0, movementCovariance):
    v = speed = np.random.normal(m0[0], movementCovariance[0][0], 1)
    theta = rotation = np.random.normal(m0[1], movementCovariance[1][1],
                                         1)

    delta = [cos(xt[2]+theta)*v,sin(xt[2]+theta)*v,theta]
    return xt+delta

def getMeasurement(vehiclePosition, landmarkPosition,
                   measurementCovariance):
    deltaX = landmarkPosition.position[0] - vehiclePosition[0]
    deltaY = landmarkPosition.position[1] - vehiclePosition[1]
    deltaXY = vectorToLandmark = [deltaX, deltaY]

    normXY = np.linalg.norm(deltaXY)
    normXY += np.random.normal(0,Ez[0][0],1)

    theta = np.arctan(deltaXY[1], deltaXY[0])
    theta += np.random.normal(0,Ez[1][1],1)

    q = normXY*normXY
    deltaX = deltaX[0]
    deltaY = deltaY[0]

    H = np.array([[ -deltaX/sqrt(q), -deltaY/sqrt(q), 0.0],[deltaX/q[0],
                                                               deltaY/q[0], -1.0],[0.0, 0.0, -1.0]])
    z = np.array([normXY[0],theta[0],0.0])
```

```
    return z,H

def resample(oldParticles):
    weightSum = 0.0001

    for oldParticle in oldParticles:
        weightSum += oldParticle.weight

    for oldParticle in oldParticles:
        oldParticle.weight = oldParticle.weight / weightSum

    M = len(oldParticles)
    newParticles = []

    r = np.random.rand() / M
    c = oldParticles[0].weight
    i = 0

    for m in range(0,M):
        U = ( r + (m - 1) ) / M

        while U > c:
            i += 1
            c += oldParticles[i].weight

        newParticles.append( oldParticles[i] )

    for particle in newParticles:
        particle.weight = 1 / len(newParticles)

    return newParticles

timesteps = 10
maxReadDistance = 2.5
```

```
landmarks = np.array([[1.0, 3.0, 0.0], [2.0, 2.5, 0.0], [0.0, 3.4,
    0.0], [0.0, 1.5, 0.0], [1.0, 3.5, 0.0]])
landmarkList = []
realPosition = np.array([[0.0], [-1.0], [pi/3]])

movementCommand = np.array([[0.05], [0.01]]) # = m0

Ex = np.array([[0.1, 0.0], [0.0, 0.05]]) # = movementCovariance
Ez = np.array([[0.1, 0.0, 0.0], [0.0, 0.01, 0.0], [0.0, 0.0, 0.0001]]) # =
    measurementCovariance

zeroVariance = np.array([[0.0, 0.0], [0.0, 0.0]])

numParticles = 5
particles = []

class Particle:
    def __init__(self, amount, position, landmarks):
        self.number = amount
        self.x = position[0]
        self.y = position[1]
        self.r = position[2]
        self.p = position
        self.weight = 1.0/amount
        self.landmarks = landmarks

class Landmark:
    def __init__(self, xPos, yPos):
        self.x = xPos
        self.y = yPos
        self.theta = 0
        self.position = [xPos, yPos, 0]
        self.seen = True
        self.E = np.array([[0.0, 0.0, 0.0], [0.0, 0.0, 0.0], [0.0, 0.0, 0.0]]) #
            Position covariance

    landmarkList.append(Landmark(1.0, 3.0))
```

```
landmarkList.append(Landmark(2.0,2.5))
landmarkList.append(Landmark(0.0,3.4))
landmarkList.append(Landmark(0.0,1.5))
landmarkList.append(Landmark(1.0,3.5))

for particle in range(0,numParticles):
    particles.append(Particle(numParticles, realPosition, landmarkList))

positionHistory = []

for step in range(0,timesteps):
    # Move robot to current position
    realPosition = moveParticle(realPosition, movementCommand, Ex)
    positionHistory.append(realPosition)

    # Move particle to see many possible current positions
    for particle in particles:
        particle.p = moveParticle(particle.p, movementCommand, Ex)
        particle.p[2] = realPosition[2] # Since the vehicle only got one
                                      # possible heading they should stay the same

    doResample = False

    for index, landmark in enumerate(landmarkList):
        z_real, G = getMeasurement(realPosition, landmark, Ez)
        deltaL = read_distance_landmark = z_real[0]
        thetaL = read_angle_landmark = z_real[1]

        if deltaL < maxReadDistance:
            doResample = True

        for particle in particles:
            if particle.landmarks[index].seen == False:
```

```
particle.landmarks[index].position[0] +=
    cos(thetaL)*deltaL
particle.landmarks[index].position[1] +=
    sin(thetaL)*deltaL

particle.landmarks[index].E = dot( dot(inv(G),Ez) ,
    transpose(inv(G)) )
particle.landmarks[index].seen == True

else:
    z_p, G_p = getMeasurement(particle.p,
        particle.landmarks[index], zeroVariance)
    deltaZ = z_real - z_p

    Q = dot( dot(transpose(G),particle.landmarks[index].E)
        , G) + Ez
    K = dot(dot(particle.landmarks[index].E,G) , inv(Q))

    particle.landmarks[index].position += dot(K,deltaZ)
    particle.landmarks[index].E = dot((np.identity(3) -
        dot(K,transpose(G))), particle.landmarks[index].E)

    particle.weight = particle.weight *
        np.power(np.linalg.norm(2*pi*Q),(-0.5*np.exp(-0.5*dot(
            dot(transpose(deltaZ),inv(Q)),deltaZ) )))

if doResample:
    particles = resample(particles)
    print "resample"

for landmark in landmarkList:
    plt.plot([landmark.x], [landmark.y], 'bs')

for particle in particles:
    plt.plot(particle.p[0], particle.p[1], 'ro')
```

```
print positionHistory
xHistory = []
yHistory = []

for position in positionHistory:
    xHistory.append(position[0][0])
    yHistory.append(position[1][0])

plt.plot(xHistory, yHistory)

plt.axis([-0.5, 2.5, -2, 6])
plt.ylabel('some numbers')
plt.show()
```
