Dedicated to innovation in aerospace

SCRIPTIE

SIMULTANEOUS LOCALIZATION AND MAPPING IN 3D ENVIRONMENTS USING A MINI-UAS

Naam: Robbert Proost
Studentnr: 1563124
Datum: 27-05-2013

MANAGEMENTSAMENVATTING

Onbemande vliegende systemen van diverse formaten worden door een groeiend aantal gebruikers ingezet voor een groeiend aantal taken. Geleidelijk maar gestaag groeien deze systemen daarmee uit tot een belangrijk hulpmiddel in een steeds breder wordende verzameling vaak civiele toepassingsgebieden. Hierbij kan gedacht worden aan een rol bij de het 3D inscannen van crime scene. Door het gebrek aan de autonomie van onbemande vliegende systemen is het voor ongeoefende gebruikers lastig een dergelijk systeem te bedienen. Om onbemande vliegende systemen toegankelijker te maken wordt er binnen het Nationaal Lucht- en Ruimtevaart Laboratorium onderzoek gedaan naar het autonoom laten vliegen van onbemande vliegende systemen.

Een probleem wat het autonoom navigeren van Unmanned Aerial Systems (UAS) beperkt is de lokalisatie van het systeem. Om het systeem autonoom te kunnen laten navigeren in een ruimte/gebied moet het systeem kennis hebben van zijn locatie. Op dit moment wordt dit grotendeels gedaan met behulp van Global Positioning System (GPS). Deze manier van lokaliseren werkt goed voor open gebieden met weinig bebouwing, maar levert als snel problemen in druk bebouwde gebieden en werkt binnen in gebouwen meestal helemaal niet. Hierdoor kan een UAS niet of moeilijk zijn locatie bepalen in deze gebieden en kan het UAS niet autonoom navigeren.

Er is onderzoek gedaan naar een alternatieve lokalisering methode op basis van het Simultaneous Localization and Mapping (SLAM) principe. Dit is een methode waarbij een 3D model van de bezochte omgeving wordt gebouwd en waar vervolgens het systeem in gelokaliseerd kan worden. Hiervoor is een sensor nodig die de directe omgeving kan waarnemen. Als sensor voor de omgevingsdata is de Microsoft Kinect sensor gebruikt. Er is gekeken naar eerdere onderzoeken op dit gebied en naar de resultaten van deze onderzoeken. Hieruit kan de conclusie getrokken worden dat SLAM algoritmen zwaar rekenintensief zijn. Deze rekenkracht is vaak niet aanwezig op een UAS. Op een grondstation kan de benodigde rekenkracht eenvoudig beschikbaar gemaakt worden, maar het systeem kan zichzelf niet lokaliseren als de verbinding met het grondstation onderbroken wordt. Daarom is er gekozen voor een gedistribueerde opzet van de software. De lokale positie bepaling wordt op het vliegende systeem berekend en het mapping gedeelte en de correctie van de lokale positie bepaling, de zogenaamde loop closure, worden op het grondstation uitgevoerd. Het grondstation stuurt na correctie de gecorrigeerde positie terug naar het UAS, zodat deze verder gaat met de gecorrigeerde positie. Het resultaat is dat er op het grondstation een 3D model gecreëerd wordt van de omgeving waarin gevlogen is en daarnaast een lokale positie bepaling op het vliegende systeem zelf, waardoor als de verbinding tussen het UAS en het grondstation verbroken wordt, het systeem zichzelf nog steeds zijn locatie kan bepalen.

Naast het onderzoek naar lokalisatie aan de hand van SLAM is er ook gekeken naar hoe de Pelican UAS uitgerust kan worden met *ROS* ondersteuning. Op het processorboard, met Intel Atom processor, aan boord van de Pelican is Ubuntu geïnstalleerd en daar op is *ROS* geïnstalleerd. Naast de standaard *ROS* componenten is er gekeken naar hoe de motoren aangestuurd kunnen worden en hoe de sensoren uitgelezen kunnen worden vanuit *ROS*. Uit dit onderzoek is gebleken dat er door verschillende partijen interfaces zijn ontwikkelt. Er is gekozen om van een van deze interfaces gebruik te maken. Door de *ROS* installatie op het processingboard met Atom processor en de software voor de interfacing met de Pelican sensoren en actuatoren is het theorie mogelijk om de ontwikkelde algoritmen die werken op de AR.drone ook te kunnen gebruiken op de Pelican.

VOORWOORD

Voor u ligt mijn afstudeerscriptie dat het project beschrijft wat ik tijdens mijn afstuderen bij het UASLab van het Nationaal Lucht- en ruimtevaart laboratorium heb uitgevoerd. Dit is een zeer leerzame, maar ook leuke periode geweest. Graag wil ik iedereen die mij geholpen heeft tijdens mijn afstuderen bedanken. In het bijzonder wil ik de volgende mensen bedanken voor het mogelijk maken van deze prettige afstudeerperiode:

- Gerald Poppinga

  Voor het mogelijk maken van de opdracht, het bieden van de juiste faciliteiten en voor de goede begeleiding tijdens mijn afstudeerperiode. Ook bedankt voor de goede feedback op de documentatie.

- Mijn directe collega's binnen het UASLab

  Voor de hulp en voor de gezellige sfeer binnen het UASLab.

- Leo van Moergestel

  Voor de goede begeleiding vanuit school en voor de goede feedback op de documentatie.

Ik wens u als lezer veel plezier met het doorlezen van deze afstudeerscriptie.

Utrecht, 28 mei 2013

Robbert Proost

INHOUD

1 INLEIDING 5

2 PROJECT OMSCHRIJVING 6

2.1 PROBLEEMSTELLING 6
2.2 DOELSTELLING 6
2.3 OPDRACHTOMSCHRIJVING 7

3 OVERZICHT TOEGEPASTE SYSTEMEN EN TECHNOLOGIEËN 8

3.1 MINI-UAS 8
3.2 SIMULTANEOUS LOCALIZATION AND MAPPING (SLAM) 9
3.3 ROBOT OPERATING SYSTEM (ROS) 11
3.4 KINECT SENSOR 13

4 RELATED WORK 16

4.1 RGBDSLAM (UNIVERSITY OF FREIBURG, GERMANY) 16
4.2 CCNY_RGBD (CITY UNIVERSITY OF NEW YORK, USA) 18
4.3 FOVIS (MASSACHUSETTS INSTITUTE OF TECHNOLOGY, USA) 20

5 SLAM MET EEN UAV 22

5.1 GEDISTRIBUEERD SLAM 22
5.2 ROS OP DE PELICAN 27

6 RESULTAAT 29

6.1 SIMULTANEOUS LOCALIZATION AND MAPPING 29
6.2 ROS OP DE PELICAN 32

7 CONCLUSIE 33

8 AANBEVELINGEN / FOLLOW UPS 34

8.1 ONDERZOEK NAAR ANDERE SENSORS 34
8.2 EXPLORATION 34
8.3 EFFICIËNTER ALGORITME 34
8.4 MEER GEBRUIK MAKEN VAN DIEPTE INFORMATIE 34
8.5 SOFTWARE UITVOEREN OP GPU 34
8.6 MEER ONBOARD PROCESSING POWER 35
8.7 DATA FUSION 35

9 BRONVERMELDING 36

BIJLAGEN

Evaluatie
Plan van Aanpak

1 INLEIDING

Deze scriptie beschrijft het afstudeerproject bij het UASLab van het Nationaal Lucht- en Ruimtevaart Laboratorium. Het afstudeerproject bestaat uit het onderzoek doen naar de mogelijkheden tot de bruikbaarheid van een Microsoft Kinect sensor in combinatie met het Simultaneous Localization and Mapping (SLAM) principe als alternatieve wijze van lokalisatie van een mini-Unmanned Aerial System (mini-UAS) in gebieden waar niet vertrouwd kan worden op het Global Positioning System (GPS). Hierbij is gekeken naar vorige onderzoeken op dit gebied, de bruikbaarheid van de resultaten van deze onderzoeken en vervolgens is er een proof-of-concept gemaakt om te onderzoeken in hoeverre deze alternatieve lokalisatie methode bruikbaar is op een mini-UAS.

In hoofdstuk 2 wordt het project omschreven. Eerst wordt een korte omschrijving van de beginsituatie gegeven. Daarna komen de probleemstellingen, doelstelling en opdrachtomschrijving aan bod. In Hoofdstuk 3 worden verschillende gebruikte technieken en systemen beschreven. Vervolgens wordt in hoofdstuk 4 een aantal vorige onderzoeken en de resultaten hiervan beschreven over ditzelfde onderwerp. Er wordt in dat hoofdstuk aandacht besteedt aan de theorie van het resultaat van deze onderzoeken en de performance van deze resultaten. In hoofdstuk 5 wordt besproken wat SLAM op een mini-UAS betekent en welke keuzes zijn gemaakt om dit mogelijk te maken. Daarnaast wordt toegelicht hoe hier technisch invulling aan gegeven is. Hoofdstuk 6 bespreekt vervolgens de resultaten van dit project, waarna in hoofdstuk 7 de conclusie volgt. Aan het einde van dit document bevinden zich de aanbevelingen met uitbreidingsmogelijkheden en vervolgens de evaluatie..

2 PROJECT OMSCHRIJVING

Onbemande vliegende systemen van diverse formaten worden door een groeiend aantal gebruikers ingezet voor een groeiend aantal taken. Geleidelijk maar gestaag groeien deze systemen daarmee uit tot een belangrijk hulpmiddel in een steeds breder wordende verzameling vaak civiele toepassingsgebieden. Hierbij kan gedacht worden aan een rol bij de inspectie van hoge of moeilijk te bereiken objecten, het creëren van overzicht binnen de handhaving van openbare orde en veiligheid [1], bij de detectie van bos- en duinbranden [2], als middel voor precision farming doeleinden [3] en als instrument voor landmeetkundige toepassingen [4].

Het UAS lab beschikt over een vloot van mini-UASen, variërend van vastvleugelige tot (multi-)rotorcraft, zoals de Parrot AR.drone [5], Ascending Technologies Pelican [6] en de Oktokopter [7]. Daarnaast zijn er verschillende sensoren en vele moderne processor systemen aanwezig. Daarnaast wordt binnen het UAS Lab voor het ontwikkelen van nieuwe functionaliteit gebruikt gemaakt van het Robot Operating System (ROS) [8]. Door ROS wordt het relatief eenvoudig om allerlei ondersteunde sensoren (zoals laser scanners en structured light oplossingen) en platformen (zoals de AR.drone en de Pelican) met elkaar te integreren. ROS maakt het daarnaast eenvoudig om functionaliteit fysiek binnen het netwerk te verplaatsen; Initieel kan er bijvoorbeeld een beeldbewerkingsalgoritme op een laptop met de goedkope AR.drones ontwikkeld en getest worden, om het vervolgens met minimale inspanning naar de on-board processing power van de Pelican te verplaatsen.

2.1 PROBLEEMSTELLING

In het kader van deze opdracht zijn er twee probleemstellingen.

LOKALISATIE

Een belangrijk probleem wat de autonomie van UAS systemen beperkt is de lokalisatie van het systeem. Om het systeem autonoom te kunnen laten navigeren in een ruimte/gebied moet het systeem kennis hebben van zijn locatie. Op dit moment wordt dit grotendeels gedaan met behulp van Global Positioning System (GPS). Deze manier van lokaliseren werkt goed voor open gebieden met weinig bebouwing, maar levert als snel problemen in druk bebouwde gebieden en werkt binnen in gebouwen meestal helemaal niet. Op die plekken is er namelijk geen of een verstoort GPS signaal beschikbaar. Hierdoor kan een UAS niet of moeilijk zijn locatie bepalen in deze gebieden en zou het UAS niet autonoom kunnen navigeren.

ROBOT OPERATING SYSTEM (ROS)

Op dit moment werkt de Pelican in het UAS lab nog niet onder ROS, maar er is wel software beschikbaar voor de integratie van de Pelican binnen ROS [9]. Doordat de Pelican nog niet onder ROS werkt kunnen reeds ontwikkelde algoritmen, die getest zijn op de AR.Drone, nog niet gedraaid en getest worden op de Pelican.

2.2 DOELSTELLING

LOKALISATIE

Op dit moment is er buiten GPS geen enkele andere positie bepaling mogelijk op de UAS systemen van het UAS lab van het NLR. Het streven binnen het UAS lab is om alles zo low-cost mogelijk proberen te houden om zo de mogelijkheden van UAS zoveel mogelijk toegankelijk te maken. De doelstelling voor dit project is om een alternatieve lokalisatie methode te realiseren voor gebieden waar geen GPS beschikbaar is, met behulp van low-cost sensor(en). De positie bepaling van het UAS zal een relatieve positie bepaling worden waarbij het nul punt het punt is waar het UAS is opgestegen. Aan het einde van het project zal dus het UAS zijn positie kunnen bepalen ten opzichte van het punt waar het systeem is opgestegen.

Het zorgvuldig documenteren van de bevindingen is ook een belangrijke doelstelling. Het lokalisatie probleem is een probleem dat niet in één afstudeerperiode perfect opgelost kan worden, waardoor de kans groot is dat

er na mijn afstuderen iemand anders met dit probleem verder gaat. Het is daarom van belang dat gemaakte keuzes, problemen waar tegenaan gelopen is en andere relevante informatie duidelijk worden gedocumenteerd zodat een eventueel vervolg soepel van start kan gaan.

ROBOT OPERATING SYSTEM (ROS)

Het UAS lab wil in de toekomst alle UAS systemen met het Robot Operating System uitrusten. Omdat het Robot Operating System dient als abstractie laag kan al ontwikkelde software voor ROS relatief eenvoudig verplaatst worden naar andere platformen. De AR.Drone van Parrot is al uitgerust met deze software, maar de Pelican nog niet. Een doelstelling voor dit project is dat de Pelican uitgerust wordt met de ROS software zodat op deze platformen de ontwikkelde algoritmen ook gebruikt kunnen worden.

2.3 OPDRACHTOMSCHRIJVING

LOKALISATIE

Om het probleem met de lokalisatie aan te pakken zal er gekeken worden naar oplossing voor het lokaliseren op plekken waar geen GPS signaal is. Hierbij moet gedacht worden aan algoritmen waarmee een 3D model gemaakt kan worden van de omgeving van een systeem, in dit geval die van het UAS. Tegelijkertijd kan de positie van het UAS binnen het 3D model bepaald worden. Een voorbeeld van een dergelijke techniek is SLAM (Simultaneous Localization and Mapping). Er zijn in het verleden al verschillende onderzoeken gedaan naar deze manier van lokaliseren [10] [11] [12]. De input van een SLAM algoritme is omgevingsdata, welke verkregen kan worden van verschillende sensoren. Een voorbeeld van een dergelijk sensor is de Kinect sensor van Microsoft. Dit is een zogenaamde low-cost RGB-D camera (Red Green Blue – Depth) die een kleurencamera en een dieptecamera heeft. Deze twee beelden kunnen “over elkaar heen gelegd worden” zodat zowel de diepte informatie als de features in het kleurenbeeld gebruikt kunnen worden door het algoritme. Het UAS Lab beschikt reeds over een Microsoft Kinect sensor.

De opdracht omvat het onderzoek doen naar de bruikbaarheid van SLAM in combinatie met een Kinect sensor van Microsoft voor het lokaliseren van een UAV in gebieden waar geen GPS signalen beschikbaar zijn. Hierbij zal gekeken worden naar eerdere onderzoeken naar dit onderwerp en zal onderzocht worden in hoeverre reeds geïmplementeerde algoritmen bruikbaar zijn voor deze specifieke situatie. De verwachting is dat veel van deze algoritmen veel rekenkracht vergen, die niet aanwezig is op een UAS zelf. Er zal gekeken worden of dit het geval is en hoe dit opgelost kan worden door. Hierbij wordt gedacht aan het simplificeren van het algoritme of het distribueren van de rekentaken over verschillende machines.

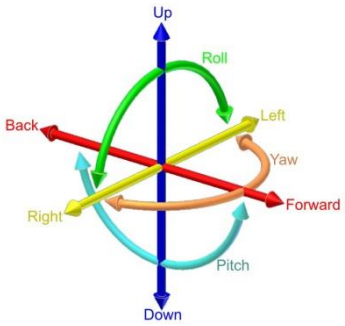
ROBOT OPERATING SYSTEM (ROS)

Naast het onderzoek naar een lokalisatie methode zal de on-board software voor de Pelican mini-UAS aangepast moeten worden, zodat de deze UASen onder ROS gebruikt kunnen worden in combinatie met reeds ontwikkelde software.

3 OVERZICHT TOEGEPASTE SYSTEMEN EN TECHNOLOGIEËN

3.1 MINI-UAS

De afkorting UAS staat voor Unmanned Aerial System. Alle onbemande luchtvaartuigen vallen onder deze categorie. In deze categorie bevinden zich verschillende subcategorien, zoals mini-UAS, micro-UAS en nano-UAS. Het gewicht en de afmetingen van het UAS bepalen in welke categorie het UAS valt. In tegenstelling tot grond voertuigen die 3 vrijheidsgraden hebben, hebben UASen 6 vrijheidsgraden. Het systeem kan namelijk transleren over de x-, y- en z-as en ook rond elk van deze assen roteren. Voor het roteren om de assen wordt voor vliegtuigen gerefereerd naar roll (roteren rond de x-as), pitch (roteren rond de y-as) en yaw (roteren rond de z-as). (zie Figuur 1)



FIGUUR 1 SIX DEGREES OF FREEDOM

3.1.1 ASCTEC PELICAN UAV

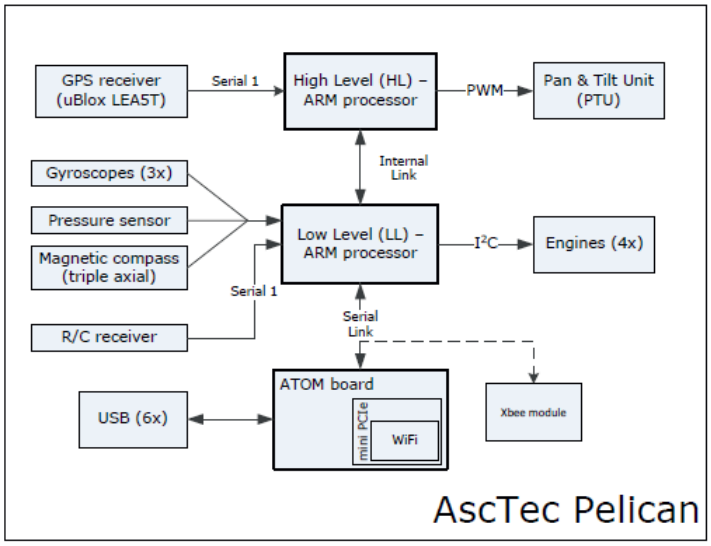
Eén van de ontwikkelplatformen die binnen het UASLab gebruikt wordt is de Pelican quadrocopter van Ascending Technologies. Dit is een mini-UAS die bedoeld is voor onderzoeksdoeleinden. De Pelican is heel modulair, waardoor deze eenvoudig voorzien kan worden van andere sensoren en actuatoren zodat de drone voor verschillende doeleinden inzetbaar is.

Enkele eigenschappen:

- Min. take-off gewicht: 630 g
- Max. payload: 650 g
- Max. vliegtijd met maximale payload: 15 min
- Programmeerbaar via de AscTec SDK en AscTec Simulink toolkit
- Modulair toren ontwerp met eenvoudige toegang tot de electronica
- Onboard rekenkracht doormiddel van een Atomboard.

FIGUUR 2 ASCTEC PELICAN QUADROCOPTER

De aansturing van de Pelican bestaat uit 3 processorboards: de High Level processor (HLP), de Low Level processor (LLP) en een Atom board. Daarnaast beschikt de Pelican over een GPS receiver, gyroscoop, een druk sensor, een magnetisch kompas en een R/C receiver. Een overzicht hiervan is te zien in Figuur 3.



FIGUUR 3 PELICAN SYSTEM OVERVIEW

Het Low Level processorboard is het hart van het UAS. Dit board is verantwoordelijk voor de aansturing van de motoren, het uitlezen van de sensoren en het omzetten van R/C commando's naar stuurcommando's. Dit board is niet door de gebruiker te programmeren. Aan deze LLP zit het High Level processor board. Zoals te zien is in de schematische weergave zijn er naast de verbinding met de LLP ook verbindingen naar de GPS reciever en de Pan & Tilt unit. Af fabriek is de enige functie van de HLP board het monitoren van het batterijvoltage en het verwerken en doorsturen van de GPS informatie. Met behulp van de SDK van AscTec kan voor dit board wel software geschreven worden. Hierbij kan bijvoorbeeld gedacht worden aan software om het UAS zichzelf te laten stabiliseren. De combinatie van beide processors (die beide op dezelfde PCB zitten) wordt verder in dit document AutoPilot genoemd.

De Pelican in de opstelling zoals die aanwezig in het UASLab beschikt ook over een AscTec Atomboard. Dit is een embedded pc met een Intel Atom processor die draait op 1.6Ghz. In dit document zal dit board het Atom board genoemd worden. Op dit systeem draait als besturingssysteem de Linux-variant Ubuntu. De autopilot van de Pelican is gekoppeld aan het Atom board met een seriële verbinding waardoor het Atom board kan communiceren met de autopilot. Met behulp van de SDK kan er software geschreven worden om commando's te sturen naar de autopilot om de Roll, Pitch, Yaw en thrust te regelen en om sensordata zoals GPS data, batterij status en IMU terug te lezen. Om veiligheidsredenen kan de R/C controller altijd de commando's van de seriële verbinding overriden.

In het verleden is er een stagiaire bezig geweest met het maken van een constructie voor onder de Pelican om het systeem nog flexibeler te maken. Zo is er toen ook een constructie gemaakt om de Kinect onder de Pelican te hangen. Daarnaast is er binnen het UASLab ook een gestripte versie van de Kinect aanwezig. Hierdoor weegt de Kinect een stuk minder en neemt deze minder ruimte in beslag.

3.2 SIMULTANEOUS LOCALIZATION AND MAPPING (SLAM)

Een belangrijk probleem in de robotica is de lokalisatie. Om een systeem autonoom te laten bewegen in een ruimte is het van belang dat het systeem weet waar deze zich bevind. Dit kan bijvoorbeeld gedaan worden met

behulp van GPS of met ander externe hulpmiddelen, zoals bakens. Het nadeel van het gebruik van deze extern hulpmiddelen is dat het systeem hier afhankelijk van is om zijn positie te bepalen. Als deze systemen (tijdelijk) niet beschikbaar zijn kan het systeem zich niet meer lokaliseren en kan het systeem zijn pad niet autonoom voortzetten. Een andere manier van lokaliseren die minder afhankelijk is van externe hulpmiddelen is lokaliseren op basis van het SLAM principe.

3.2.1 Wat is Simultaneous Localization and Mapping?

Het idee achter SLAM is dat een robot in onbekend gebied op een onbekende positie wordt neergezet en dat de robot vanaf daar stap voor stap een model gaat opbouwen van zijn omgeving en zich binnen deze map lokaliseert. De map die gegenereerd wordt dient dus als referentie. Om dit model te kunnen maken en zich hierin te lokaliseren moet het systeem kennis hebben van de directe omgeving. Dit kan met verschillende sensoren zoals camera's, laser rangefinders, Time-Of-Flight camera's en andere sensoren waar informatie van de omgeving verkregen kan worden. De meest gebruikte sensoren zijn sensoren die diepte kunnen waarnemen.

Zoals de naam al zegt combineert het algoritme twee taken, die van het in kaart brengen en het lokaliseren. Dit algoritme kent echter een vorm van het bekende kip-ei probleem. Om de positie te bepalen van het systeem is namelijk een referentie map nodig en om een map te creëren is de locatie van de sensor een vereiste. In gebieden waar GPS beschikbaar is kan deze locatie als initiële positie gebruikt worden, maar in totaal onbekende gebieden waar geen GPS beschikbaar is, is de start positie van het systeem 0,0,0 (x,y,z). Vanaf die positie begint het systeem maken van een model van de omgeving en het systeem zal zich positioneren ten opzichte van deze start positie.

Odometrie

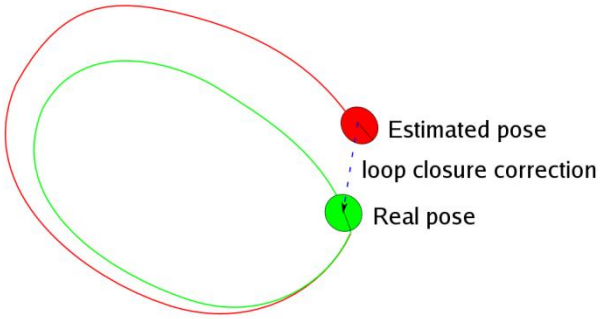
Als basis voor de lokalisatie dient de odometrie. Odometrie is een techniek waarbij data van bewegende sensoren gebruikt wordt om de wijzigingen in positie te bepalen over tijd. Bij rijdende robots kan deze informatie bijvoorbeeld verkregen worden van wielencoder sensoren om zo het afgelegde pad bij te houden. Vliegende robots of humanoid robots hebben geen wielenencoders. Een andere vorm van odometrie is visual odometrie, waarbij camerabeelden gebruikt worden om te verplaatsing te bepalen. Hierbij worden twee op één volgende frames met elkaar vergeleken en wordt bepaald hoe de pixels of features in de afbeeldingen zijn verschoven ten opzichte van elkaar. Aan de hand van deze verplaatsing kan bepaald worden in welke richting de robot bewogen heeft en kan zo zijn positie bijgehouden worden.

Mapping

Met de omgevingsdata afkomstig van de sensor kan een model gemaakt worden van de omgeving waarin het systeem zich bevindt. Afhankelijk van welke sensor de informatie komt kan er een 2D of een 3D model gemaakt worden van deze omgeving. Deze informatie wordt in het model geplot ten opzichte van de positie die berekend is door de odometrie.

Loop closure

Bij SLAM wordt het lokaliseren en het creëren van een model van de wereld gecombineerd. Doordat de odometrie (bijna) nooit perfect is zal naarmate de robot verder gereden/gevlogen heeft de fout in odometrie toenemen (zie Figuur 4). Dit resulteert in een model dat niet netjes op elkaar aansluit. Als de robot een rondje heeft gereden of gevlogen moet het model netjes op elkaar aansluiten. Doordat de odometrie niet perfect is, is dit vaak niet het geval. Dit probleem wordt loop closure genoemd. Door bij het detecteren van een loop een correctie uit voeren zodat het afgelegde pad aan elkaar sluit, komt de positie van de robot beter overeen met de werkelijkheid.



FIGUUR 4 LOOP CLOSURE PROBLEEM

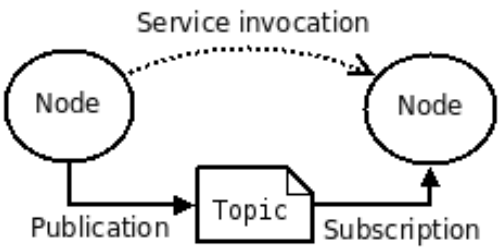
3.2.2 SLAM MET EEN RGBD-SENSOR

Traditioneel werd voor de input van een SLAM algoritme vaak gebruik gemaakt van laserscanners of een kleurencamera (monoculair SLAM). Sinds enkele jaren zijn er relatief goedkope RGBD-sensoren te koop die in eerste instantie bedoeld waren voor home-entertainment en gaming. Deze sensoren zijn echter ook zeer interessant voor robotica toepassingen. Zo is er in de afgelopen paar jaar onderzoek gedaan naar de mogelijkheden van goedkope RGBD-sensoren voor het gebruik als input voor SLAM. Met dit type sensor kan het algoritme zo gemaakt worden dat deze gebruik kan maken van zowel het kleurenbeeld als de diepte informatie.

Het SLAM algoritme is er in zowel een 2D- als 3D-variant. Ondanks dat het resultaat van het algoritme in beide gevallen vergelijkbaar zal zijn (beide leveren een positie en een kaart) is er een groot verschil in de manier waarop dit gedaan wordt. Een groot verschil is de hoeveelheid data die het algoritme moet verwerken. Zo is er bij 2D alleen informatie over de omgeving op één hoogte. Bij 2D SLAM wordt dan ook vaak bijvoorbeeld laser data vergeleken met vorige scans (scanmatching). Bij 3D SLAM zou dit veel te rekenintensief zijn omdat er veel meer data is die vergeleken moet worden. Daarom wordt bij 3D SLAM vaak gebruik gemaakt van zogenaamde landmarks. Deze landmarks zijn een soort van herkenningspunten die zichtbaar zijn vanuit een bepaalde positie. Als de landsmarks van verschillende frames met elkaar vergeleken worden en er overeenkomsten gevonden worden, kan bepaald worden hoe de frames ten opzichte van elkaar zijn gedraaid/verschoven. Deze landmarks kunnen bepaald worden op basis van camerabeelden maar ook op basis van bijvoorbeeld reliëf. Deze manier van oriënteren is vergelijkbaar met hoe mensen dat doen. Bij gebruik van data uit camerabeelden wordt vaak gerefereerd naar features in plaats van landmarks.

3.3 ROBOT OPERATING SYSTEM (ROS)

Het Robot Operating System *(ROS*) is een open source middleware framework dat op Linux draait. De middelware vereenvoudigt het om de software modulair op te bouwen [8]. *ROS* doet dit door verschillende delen van de software op te delen in zogenaamde Nodes. Een node is een op zich zelf staand proces dat via TCP/IP kan praten met andere nodes. Dit kunnen ze doen doormiddel van Topics en Services. Dit basisprincipe is weergegeven in Figuur 5. Een node schrijft data in een topic (publishen), welke vervolgens via TCP/IP verstuurd wordt naar andere nodes die zich aangemeld (subscribed) hebben op dit topic. Deze manier van communiceren (over TCP/IP) maakt het mogelijk om software gedistribueerd te draaien waarbij nodes op verschillende systemen kunnen draaien.



FIGUUR 5 BASIS CONCEPT ROS

Het is ook mogelijk om nodes met elkaar te laten communiceren met behulp van services. Het grote verschil tussen services en topics is dat services voor een specifieke node aangeroepen worden, terwijl bij een topic alle subscribers (kunnen verschillende nodes zijn) genotified worden. Daarnaast is communicatie via topics asynchroon en via services is synchroon. Dit laatste betekent dat bij een service aanroep, de node die aanroept pas verder gaat met het uitvoeren van zijn programma als de aangeroepen service volledig is uitgevoerd. Bij het publishen in een topic wordt niet gewacht op een confirmatie en zal de node direct verder gaan.

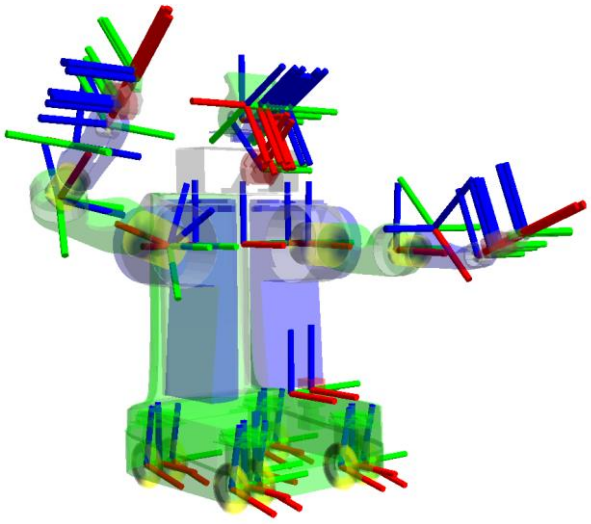
Naast dat ROS het vereenvoudigt om gedistribueerde software te schrijven biedt ROS standaard al veel ondersteuning voor verschillende soorten hardware zoals sensoren en actuatoren. Hierdoor is de kans groot dat aansturing voor de gewenste sensoren al beschikbaar is en deze niet zelf geschreven hoeft te worden. Zo is er bijvoorbeeld al een ROS driver voor de Kinect Sensor.

Op dit moment is de meest recente versie ROS groovy. De vorige stabiele versie van ROS was ROS fuerte. Tussen deze twee versies zijn grote verschillen, onder meer in het build systeem. Daarnaast zijn er grote wijzigingen doorgevoerd in de GUI packages en de API voor deze packages.

3.3.1 TRANSFORMATIES/TF

Een robot systeem heeft vaak meerdere 3D coordinate frames die veranderen over tijd. Deze frames zijn voornamelijk bedoeld voor onderdelen van een robot die kunnen bewegen of waar ten opzichte van bewogen kan worden. Voorbeelden hiervan zijn frames zoals het world frame, base frame, gripper frame, head frame, enz (zie Figuur 6). Binnen ROS is er een package om deze zaken te regelen en bij te houden [13]. Deze package, genaamd TF, bevind zich standaard in ROS. Zo wordt informatie over de verschillende frames bijgehouden door deze package en kunnen vragen gesteld worden als:

- Waar was het head frame ten opzichte van het world frame (origin van de wereld) 1 seconde geleden?
- Wat is de huidige positie van van het baseframe ten opzichte van het world frame?

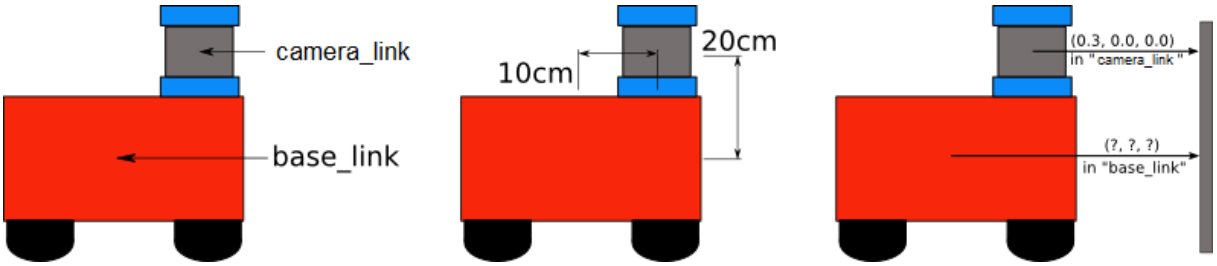


FIGUUR 6 VERSCHILLENDE COORDINATE FRAMES [44]

Meestal zijn er voor een robotsysteem een aantal standaard frames. In het geval van een robot met een Kinect zijn er de volgende frames:

- /base_link
- /camera_link

Het frame /base_link is het frame met als origin het midden van de robot en het frame /camera_link is het frame met als origin het midden van de camera. De transformatie tussen base_link en camera_link is in dit geval statisch doordat de Kinect op een vaste positie gemonteerd is op de robot. In Figuur 7 is een voorbeeld van deze opstelling te zien met een grond robot.



FIGUUR 7 VOORBEELD RELATIE TUSSEN FRAMES OP EEN ROBOT

De sensordata afkomstig van de Kinect is data ten opzichte van het frame /camera_link. Als de transformatie tussen /base_link en /camera_link 0 zou zijn (de kinect staat precies in het midden van de robot) dan kan deze informatie direct gebruikt worden. In de meeste gevallen is dat niet zo en dus is er een extra berekening nodig om de sensordata ten opzichte van /base_link te verkrijgen. Aangezien de transformatie tussen /camera_link en /base_link bekend en statisch is kan berekend worden hoe de omgeving er uit ziet ten opzichte van het /base_link frame. In het voorbeeld van Figuur 7 is de transformatie tussen de frames {0.1, 0.2, 0.0}. Als de door de Kinect gemeten afstand 0.3m is, dan is dit ten opzichte van het /base_link frame 0.4m (0.3m + 0.1m). Hierbij wordt er van uit gegaan dat afstand tot het object ter hoogte van het midden van de robot hetzelfde is als op de hoogte van de camera.

3.3.2 Point Cloud Library

De Point Cloud Library (PCL) is een framework gericht op het werken met n-dimensionale point clouds en 3D geometry processing [14]. De library bevat verschillende algoritmen, zoals algoritmes voor filtering en surface reconstruction, en data structuren voor 3D modellen. Een van deze datastructuren is de zogenaamde Point Cloud. Dit is een vector met daarin Points welke een X, Y en Z waarde hebben en daarnaast kleureninformatie bevat in de vorm RGB. De *PCL* is ontwikkeld door hetzelfde bedrijf dat *ROS* heeft ontwikkelt. *ROS* biedt daarom ook standaard ondersteuning voor *PCL*. Meer informatie over PCL kan gevonden worden op de website van PCL [15].

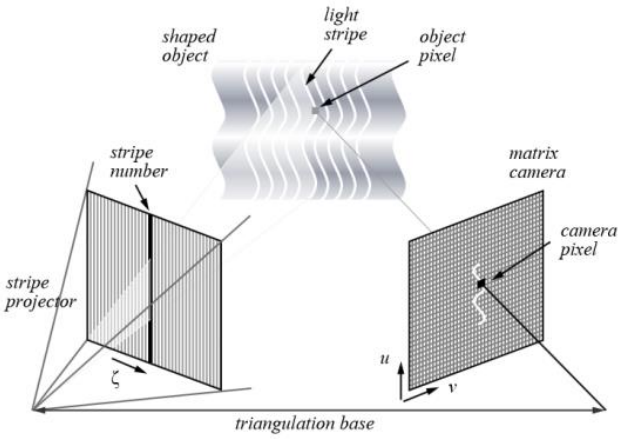
3.4 Kinect sensor

Een onderdeel van het project was het onderzoek doen naar de bruikbaarheid van de Kinect sensor onder een UAV voor het uitvoeren van SLAM. De Kinect sensor is een sensor die door Microsoft is ontwikkeld en welke bedoeld is als Human Interface Device voor de spelcomputer van Microsoft, de Xbox 360. Door voor de sensor te staan en bewegingen te maken kan een spel bestuurd worden. Dit is mogelijk doordat de sensor een dieptecamera heeft en daardoor de speler onderscheiden kan worden van de achtergrond (dichtbij-veraf). Al snel na de uitkomst van de sensor werd door onderzoekers gezien dat deze sensor ook interessant kan zijn voor andere toepassingen, mede door de relatief lage prijs (€149,99[1]) in combinatie met de dieptecamera.

[1] Advies prijs bij de release

3.4.1 De techniek achter de Kinect Sensor

De Kinect is een speciale camera. Naast dat deze sensor een reguliere kleurencamera heeft, beschikt de Kinect ook over een extra camera om diepte waar te nemen. In tegenstelling tot Laser rangefinders (die binnen de robotica veel gebruikt worden als sensor om diepte informatie te verkrijgen) werkt de diepte camera van de Kinect niet op basis van het Time-of-Flight principe, maar op basis van structured light. Bij structured light wordt er een patroon van IR puntjes geprojecteerd op de omgeving waarna er daarna met een IR camera gekeken wordt hoe dit patroon is vervormd. Aan de hand van deze vervorming kan bepaald worden wat de afstand is tot objecten in de directe omgeving. In Figuur 8 wordt dit geïllustreerd. De berekeningen die nodig zijn om de diepte informatie uit het vervormde patroon te halen worden uitgevoerd door de sensor zelf en niet door de software op de computer of Xbox. Dit betekent dat de diepte informatie voor het systeem die gebruik maakt van de data "gratis" is in tegenstelling tot stereo vision waarbij de diepte informatie berekend moet worden door de software op computer.



FIGUUR 8. STRUCTURED LIGHT PRINCIPE [43]

Naast de diepte informatie heeft de Kinect ook een kleurencamera, een microfoonarray en een IMU[2]. De beelden van de kleurencamera en de diepte camera kunnen over elkaar heen gelegd worden waardoor van elke kleurenpixel bekend is op welke diepte deze ligt. Dit wordt "registered depth" genoemd. Deze eigenschap maakt de sensor interessant voor andere doeleinden dan het besturen van een spel.

3.4.2 Drivers voor Linux

Doordat de Kinect in eerste instantie bedoeld was als sensor voor de spelcomputer, waren er geen drivers of Software Development Kit beschikbaar. Een maand nadat Microsoft de Kinect uitgebracht had, was deze al gehackt [16]. Nadat Microsoft hoogte had gekregen van de interesse voor het gebruik van de sensor voor andere toepassingen hebben ze later, min of meer noodgedwongen, een SDK uitgebracht voor Windows. Microsoft geeft geen officiële ondersteuning voor Linux. Om de Kinect toch te kunnen gebruiken onder Linux is een community genaamd OpenKinect verder gegaan met het reverse engineren van de Kinect en het maken van opensource drivers die zowel moeten werken onder Windows als Linux [17]. Deze driver is inmiddels gereleaset onder de naam libfreenect en hier wordt nog aan ontwikkeld.

De chip in de Kinect is van de fabrikant PrimeSense. Deze fabrikant heeft voor haar chips een opensource SDK uitgebracht onder de naam OpenNI die zowel voor Windows als voor Linux beschikbaar is [18]. Deze zou ook gebruikt kunnen worden voor de Kinect, zij het niet dat Microsoft niets heeft vrijgegeven over de communicatie tussen de PC en de chip. Door reverse enginering is achterhaald hoe deze communicatie verloopt. Om de PrimeSense driver bruikbaar te maken voor de Kinect is daarom een patch nodig [19].

3.4.3 Ondersteuning ROS

Doordat er inmiddels een opensource drivers beschikbaar zijn voor Linux kan de Kinect sensor ook gebruikt worden in ROS. Om de sensor binnen ROS te gebruiken is er een wrapper nodig die de output van de driver omzet in ROS specifieke formaten. Momenteel zijn er twee van deze wrappers beschikbaar die beide een andere driver gebruiken: libfreenect en OpenNI. Beide wrappers houden de zelfde naming conventions aan qua

[2] Inertial measurement unit

topics en services, waardoor de software die de Kinect output gebruikt niet hoeft aangepast te worden als er van driver gewisseld wordt.

Enkele verschillen tussen de drivers:

	Libfreenect	**OpenNI**
Ondersteuning verschillende resoluties (QVGA, VGA, SVGA)	Nee, alleen VGA (640x480)	Ja
Werkt onder VMware	Ja	Ja
Ondersteuning andere RGBD apparaten	Nee	Ja

TABEL 1 VERGELIJKING KINECT DRIVERS VOOR ROS

Naast deze verschillen op papier is er ook gekeken naar de performance van beide drivers. Dit is gedaan door voor beide drivers de volgende tests te doen op een zelfde systeem[3] :

- Weergeven van stream van de kleurencamera
- Weergeven van de stream van de diepte informatie

Daarnaast is er gekeken naar het dataverbruik van de Kinect als deze op 30Hz zijn beelden aanbiedt. Dit is getest door voor een periode van 30 seconden elke seconde de dataproductie te noteren en vervolgens hiervan het gemiddelde van te nemen. De CPU load is vergelijkbare wijze gemeten. Hierbij is voor een periode van 30 seconden elke seconde de CPU load van het proces genoteerd en vervolgens is hiervan het gemiddelde genomen. De tests resulteerden in de volgende resultaten :

	Libfreenect	**OpenNI**
Weergeven kleurencamera stream	11 % CPU load	16 % CPU load
Weergeven diepteinformatie	35 % CPU load	47 % CPU load
Data verbruik kleurenbeeld	27,7 MB/s (221,6 Mbit/s)	27,8 MB/s (222,4 Mbit/s)
Data verbruik diepteinformatie	37,3 MB/s (298,4 Mbit/s)	37,3 MB/s (298,4 Mbit/s)

TABEL 2 TEST RESULTATEN ROS DRIVERS

Uit deze resultaten is te concluderen de libfreenect iets minder rekenkracht vraagt als de driver van OpenNI. Het verschil is niet bijzonder groot, maar omdat we te maken hebben met een systeem met beperkte rekenkracht is de libfreenect een logische keuze.

De grootte van de datastream die beide drivers genereren verschilt niet significant van elkaar, maar de hoeveelheid data die verstuurd wordt in beide gevallen wel veel. Dit komt doordat voor beide streams geldt dat ze ruwe data bevatten die op geen enkele wijze gecomprimeerd is. Dit betekent voor het kleurenbeeld dat per frame 640x480x3 bytes (aantal pixels * aantal bytes per pixel) verstuurd worden. Wat overeen komt met ongeveer 0,92MB per frame. Bij de diepte informatie wordt per pixel een float gestuurd, wat neerkomt op 640x480x4 (aantal pixels * 4 bytes per float). Dat is ongeveer 1,23 MB per frame. Aangezien de Kinect op een frequentie van 30 Hz beelden genereert, verklaart dat de bij de tests gevonden data productie.

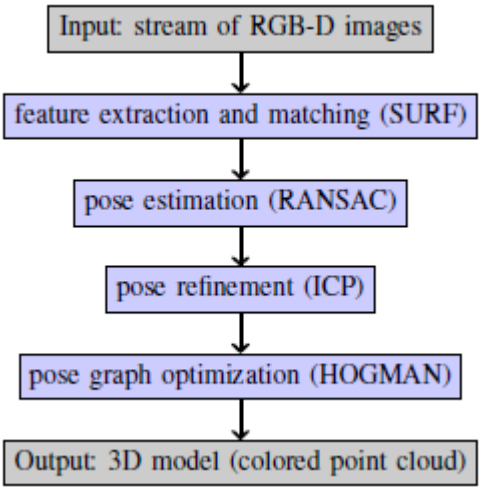
[3] Core i5 processor, 2GB memory, VMware met Ubuntu 12.04

4 RELATED WORK

Afgelopen jaren is er al veelvuldig onderzoek gedaan naar SLAM in combinatie met een RGBD sensor. In dit hoofdstuk wordt er gekeken naar de resultaten van deze onderzoeken en wat voor implementaties er reeds zijn ontwikkeld. Daarnaast wordt er kort aandacht besteed aan de performance van deze implementaties. Voor de performance tests is bij alle tests gebruik gemaakt van de zelfde testset en de tests zijn op de zelfde machine uitgevoerd.

4.1 RGBDSLAM (UNIVERSITY OF FREIBURG, GERMANY)

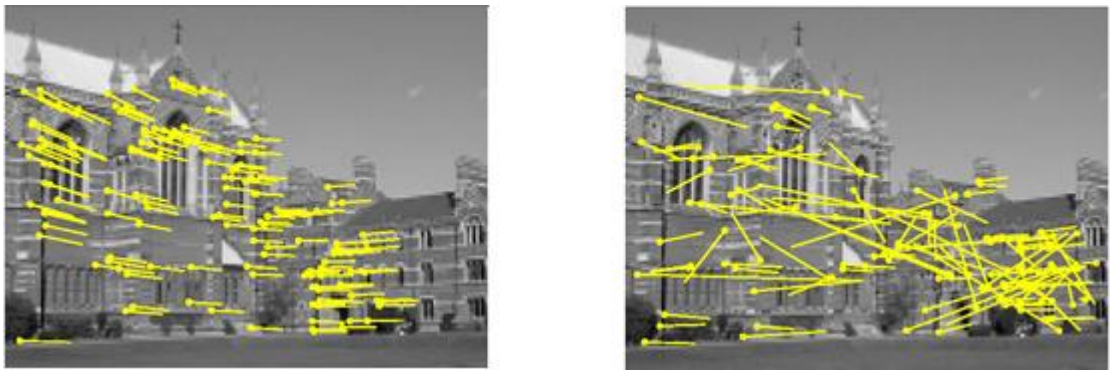
Door de universiteit van Freiburg in Duitsland is er een aantal jaar terug in het kader van de 3D challenge van *ROS* een opstart gemaakt met het schrijven van een SLAM algoritme dat gebruik maakt van de sensorinformatie van de Kinect [20] [21]. Het algoritme dat zij ontwikkeld hebben maakt gebruik van zowel de beelden afkomstig van de kleurencamera als de diepte informatie. De volgende stappen worden genomen.



FIGUUR 9 DE VIER PROCESSING STAPPEN VAN RGBDSLAM [21]

SURF EN RANSAC

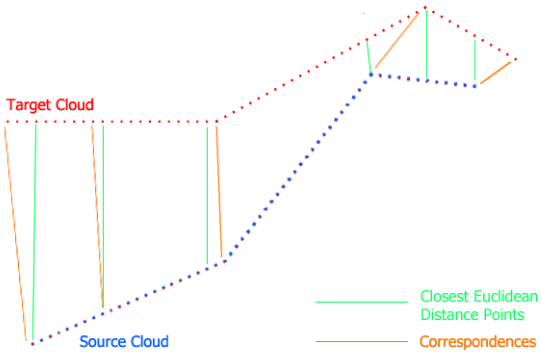
Het beeld van de kleurencamera wordt eerst omgezet naar grijswaarden en vervolgens worden uit dit beeld SURF-features (Speeded Up Robust Features) gehaald [22]. Deze features worden vervolgens vergeleken met eerdere binnengekregen beelden om de transformatie van de camera te kunnen bepalen. Doordat er ook gebruik gemaakt kan worden van de diepte informatie dient deze als metrische schaal. Hierdoor kan bepaald worden hoeveel werkelijke centimeters de camera verplaatst is. Omdat er bijna altijd sprake is van afwijkingen in het beeld of in de diepte informatie (ruis) kan het voorkomen dat bij het vergelijken van twee frames fouten worden gemaakt bij het matchen van features. Deze fouten resulteren in een foutieve transformatie. Door een aantal transformaties met elkaar te vergelijken, kan bepaald worden wat de trend is en welke transformaties waarschijnlijk foutief zijn doordat deze de trend niet volgen. Een voorbeeld hiervan is zichtbaar in Figuur 10. Links zijn de transformaties die binnen de trend van de meeste transformaties passen en rechts de transformaties die buiten deze trend vallen. Op deze manier worden de foutief berekende transformaties er uit gefilterd. Dit algoritme is het RANSAC algoritme [23].



FIGUUR 10 LINKS: INLIERS; RECHTS: OUTLIERS [24]

ICP

Bij deze stap is de transformatie bepaald met behulp van de verplaatsing van de features. Om een nauwkeurigere transformatie te bepalen wordt vervolgens met de diepte informatie een soortgelijke berekening gedaan. Dit wordt gedaan met het zogenaamde Iterative Closest Point (ICP) algoritme [25]. Dit algoritme heeft als input twee point clouds die getransformeerd zijn ten opzichte van elkaar. Om de transformatie te bepalen, moeten corresponderende punten tussen de twee clouds bepaald worden. Het ICP algoritme gaat er van uit dat de corresponderende punten de punten zijn die het dichtst bij elkaar liggen. Met andere woorden de punten waarbij de euclidische afstand het kleinst is. Een voorbeeld is te zien in Figuur 11 ICP Correspondences tussen twee clouds. In dit figuur zijn de groene lijnen de overeenkomsten die het algoritme vindt en de oranje lijnen de daadwerkelijke overeenkomsten.



FIGUUR 11 ICP CORRESPONDENCES TUSSEN TWEE CLOUDS

In verschillende iteraties wordt vervolgens geprobeerd om deze afstanden zo klein mogelijk te maken. Als de twee clouds (bijna) over elkaar heen vallen is bekend hoe cloud b getransformeerd is ten opzichte van cloud a.

HOGMAN

Voor het loop closure probleem wordt gebruik gemaakt van een variant van GraphSLAM. Bij GraphSLAM wordt het probleem (SLAM probleem) gerepresenteerd door een graph. In deze graph is elke node een positie van de robot tijdens het maken van het model en elke edge tussen de nodes staat voor de overeenkomsten tussen deze nodes. Een implementatie dit type graph optimization is HOGMAN [26].

4.1.1 PERFORMANCE

Deze software is geschreven voor *ROS* en er kan met behulp van parameters het een en ander getuned worden aan het algoritme. Een nadeel van deze implementatie is dat het odometrie gedeelte en het mapping gedeelte in elkaar verweven zijn. Beide taken draaien dan ook in het zelfde proces. Hierdoor is het lastig om de taken op

verschillende frequenties te laten draaien. Het is wenselijk als de lokale positiebepaling, de visuele odometrie, realtime wordt uitgevoerd. Het mapping proces daarin tegen zou op een lagere frequentie kunnen draaien.

Tijdens het testen bleek dat het algoritme niet verder gaat als deze eenmaal track verliest. Als er tussen twee frames te weinig overeenkomsten gevonden worden wordt die genegeerd. De kans is groot dat de daar op volgende frames ook geen tot weinig overeenkomsten hebben met het laatste opgeslagen frame waardoor het algoritme ook dan niet verder gaat en dus niet meer lokaliseert en mapt. Het algoritme zal pas weer verder gaan als de camera in een positie komt waarbij er features zichtbaar zijn die ook zichtbaar zijn in het laatste opgeslagen frame. Dit kan in sommige gevallen lang duren.

Naast het voorgenoemde probleem is een ander probleem dat er regelmatig segfaults optreden tijdens het proces. Het proces crasht dan en er kan weer opnieuw begonnen worden. De software is nog volop in ontwikkeling en de verwachting is dat deze problemen in de komende tijd opgelost zullen worden. Een resultaat van het algoritme met een dataset afkomstig van Technische Universiteit München [27] is te zien in Figuur 12.

FIGUUR 12 RESULTAAT VAN RGBDSLAM

4.2 CCNY_RGBD (City University of New York, USA)

Op de City University of New York wordt er ook onderzoek gedaan naar SLAM in combinatie met een RGBD-sensor. In februari 2013 hebben zij hun geschreven software opensource beschikbaar gemaakt [28]. Deze software bestaat uit een ROS node die visuele odometrie uitvoert en een node die offline SLAM kan uitvoeren. Offline houdt in dat de beelden en de informatie afkomstig van de visuele odometrie node opgeslagen worden tijdens opnemen van de beelden en pas achteraf verwerkt worden tot een map.

De paper die geschreven is over deze implementatie is gepubliceerd op de International Conference on Robotics and Automation (ICRA) die plaats vond van 6 – 10 mei 2013. De paper nog niet publiekelijk te downloaden. De stappen die hieronder zijn beschreven zijn gevonden door de sourcecode te bekijken.

De stappen van het visuele odometrie gedeelte zijn vergelijkbaar met die van RGDBSLAM van de Freiburg universiteit. Eerst worden eerst features uit het beeld van de kleurencamera gehaald. Vervolgens worden deze gecombineerd met de diepte informatie en wordt met behulp van ICP (zie H4.1) de transformatie gezocht ten

opzichte van het vorige frame. Het type features dat gedetecteerd wordt is instelbaar, maar de standaard instelling is GFT features (Good Features to Track). Als laatste stap wordt met behulp van een Kalmanfilter de ruis weg gefilterd.

Het tweede gedeelte van deze implementatie is het mapping gedeelte. Deze is zo opgezet dat elke keer als de Kinect sensor meer dan een bepaalde drempelwaarde verplaatst is (volgens de visuele odometrie), er een zogenaamd keyframe opgeslagen wordt. In deze keyframe worden de positie op basis van de visuele odometrie, het kleurenbeeld en de diepte informatie opgeslagen. Vervolgens kan de gebruiker de opdracht geven op deze data het SLAM algoritme uit te voeren. Daarbij worden de volgende stappen uitgevoerd:

1. SURF features in het kleuren beeld detecteren
2. Transformatie ten opzichte van het vorige keyframe op basis van de visuele odometrie verkrijgen
3. Met behulp van een FLANN matcher overeenkomsten tussen frames vinden
4. Met behulp van RANSAC outliners er uit filteren
5. Associaties toevoegen aan graph
6. Graph optimaliseren
7. Keyframe poses updaten

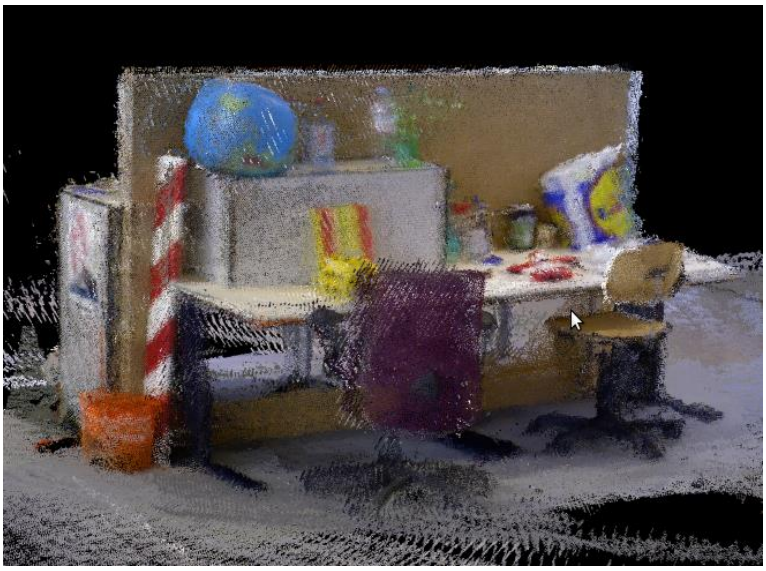
De eerste twee stappen zijn het verkrijgen van data over het huidige frame. Vervolgens wordt met behulp van een FLANN matcher overeenkomsten gezocht met andere frames. De FLANN matcher [29] is een nearest neighbors search methode waarbij gezocht wordt naar punten in twee frames die het dichtst bij elkaar liggen. Op deze manier kan gedetecteerd worden of frames overeenkomsten vertonen en kan berekend worden hoe deze gedraaid zijn ten opzichte van elkaar. Vervolgens wordt met behulp van RANSAC de uitschieters uit deze overeenkomsten weg gefilterd. Het resultaat van deze stappen is dat er nu een lijstje is met associaties van het nieuwe frame met verschillende andere frames. Voor het corrigeren van de posities van de keyframes en voor de loop closure wordt gebruik gemaakt van een graph optimalisatie methode, net zoals bij RGBDSLAM. Meer over de werking van graph optimalisatie kan gevonden worden in H4.1. De gevonden accociaties uit stap 4 worden aan het graph toegevoegd en vervolgens wordt het graph geoptimaliseerd. Aan de hand van het geoptimaliseerde graph worden vervolgens de posities in de keyframes geupdate.

4.2.1 PERFORMANCE

Net zoals RGBDSLAM is ook deze software geschreven voor *ROS*. Daardoor is het ook bij deze software mogelijk om relatief eenvoudig parameters te tunen. Een voordeel van de opzet van deze implementatie is dat het uit twee gedeeltes bestaat: visuele odometrie en mapping. Doordat er gebruik gemaakt wordt van *ROS* is het daarom mogelijk om beide processen op een verschillend systeem uit te voeren. Een nadeel van de implementatie is dat het mapping gedeelte nu alleen offline uitgevoerd kan worden.

Tijdens het testen van deze software op de zelfde dataset als die gebruikt is om RGBDSLAM te testen is gebleken dat de software van CCNY geen last heeft van het probleem dat als de visuele odometrie faalt het hele proces vast loopt. De software gaat verder en het proces vervolgt zich. Dit heeft ook een nadeel. Als de visuele odometrie faalt, zal op het moment dat de visuele odometrie weer back on track is deze verder gaan vanaf de laatst bekende positie. Mogelijk is de camera in de tussentijd al behoorlijk verplaatst waardoor de odometrie niet meer klopt. Doordat het mapping gedeelte niet online wordt uitgevoerd wordt dit niet gecorrigeerd en zal de fout in de odometrie op blijven lopen.

De software is begin dit jaar pas vrijgegeven en er wordt nog volop aan ontwikkeld. De verwachting is dat er in de loop van de tijd extra functies worden toegevoegd en dat de algoritmen worden geoptimaliseerd. Een resultaat van de huidige software en een dataset van de TU München is te zien in Figuur 13.



FIGUUR 13 RESULTAAT CCNY IMPLEMENTATIE

4.3 FOVIS (MASSACHUSETTS INSTITUTE OF TECHNOLOGY, USA)

Massachusetts Institute of Technology (MIT) heeft een aantal jaar terug in samenwerking met de University of Washington onderzoek gedaan naar SLAM met een Kinect op een UAS [30]. Het visuele odometrie gedeelte van resultaat van dit onderzoek is opensource vrijgegeven. Deze hebben zij gereleaset onder de naam Fast Odometry from VISion (FOVIS) library [31].

In tegenstelling tot de vorige twee besproken algoritmen wordt bij dit algoritme minder gebruik gemaakt van bestaande technieken. Er zijn zelf methoden ontwikkelt en deze zijn vergeleken met bestaande technieken. Hieruit kwam naar voren dat de nieuwe methoden in de meeste gevallen beter presteerden. Het algoritme bestaat uit de volgende stappen die uitgevoerd worden telkens als er een nieuw frame binnen komt:

1. Image Preprocessing: Het kleurenbeeld wordt omgezet in grijswaarden en het beeld wordt gesmoothed met behulp van een gaussian filter.
2. Feature extractie: Uit het beeld dat verkregen is in stap 1 worden FAST features [32] gehaald. Daarnaast wordt de diepte informatie opgeslagen bij deze features.
3. Initiele rotatie bepaling
4. Feature matching: De verkregen features worden vergeleken het vorige frame om de verplaatsing te kunnen bepalen.
5. Inlier detectie: Van de verkregen matches in stap 4 worden de uitschieters tussen de overeenkomsten weg gefilterd
6. Motion estimation: Als laatste stap wordt met de overgebleven inliers de beweging berekend. Dit wordt gedaan door middel van een nonlinear least-squares solver [33].

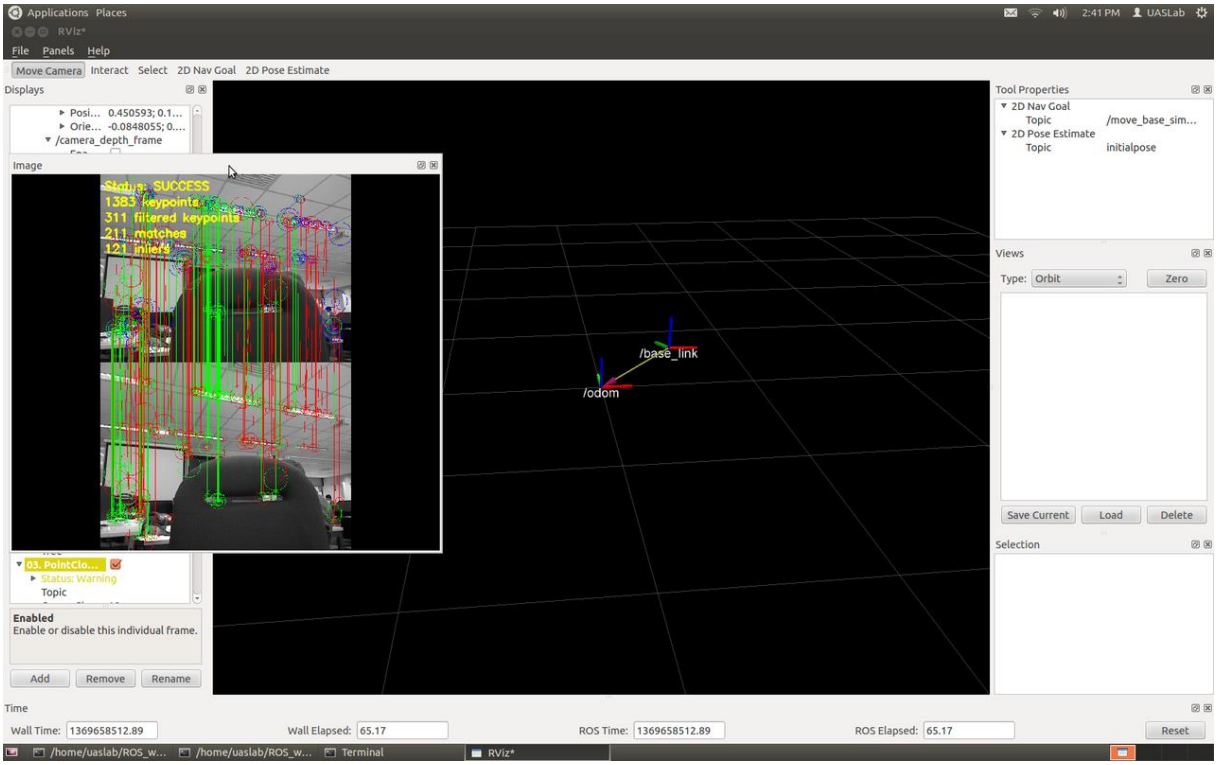
Het resultaat van de stappen is een 6D transformatie ten opzichte van het vorige frame.

4.3.1 PERFORMANCE

Deze software is niet geschreven voor *ROS*, maar is wel zo geschreven dat deze zo portable mogelijk is. De enigste vereisten zijn dat de Eigen 3 library geïnstalleerd is en dat de CPU Intel SSE2 ondersteund. De reden voor deze laatste is omdat er gebruik wordt gemaakt van SIMD (Single Instruction Multiple Data) instructies. Dit komt de performance ten goede. Ondanks dat de software niet geschreven is om met *ROS* te werken is er een duidelijke API beschikbaar op de website en is het relatief eenvoudig om een ROS wrapper te schrijven voor de

library. Inmiddels is er een ROS wrapper beschikbaar voor FOVIS. Deze is niet geschreven door de makers van FOVIS [34].

In tegenstelling tot RGBDSLAM en CCNY_RGBD bevat de FOVIS library alleen algoritmen voor visuele odometrie. Deze library creëert geen model en voert geen loop closure uit. Ondanks dat er geen loop closure wordt gedaan door de library is de positie bepaling zeer goed te noemen. Dit is getest door dezelfde dataset te gebruiken als bij RGBDSLAM en CCNY_RGBD en vervolgens met RViz[4] te kijken naar de transformaties die berekend worden. Op het oog kloppen de transformaties met de werkelijkheid en daarnaast raakt het algoritme niet één keer track kwijt. Ook is de software zeer stabiel en vertoont deze geen segfaults. De visualisatie in RViz van de transformaties berekend door FOVIS en de visualisatie van de feature matching zijn te zien in Figuur 14.



FIGUUR 14 TRANSFORMATIE VISUALISATIE EN FEATURE MATCHING IN RVIZ

De FOVIS library is al enige tijd beschikbaar [31] en hier wordt momenteel niet actief aan ontwikkeld. De verwachting is dat aan deze library niet veel meer zal veranderen omdat deze al zeer stabiel is.

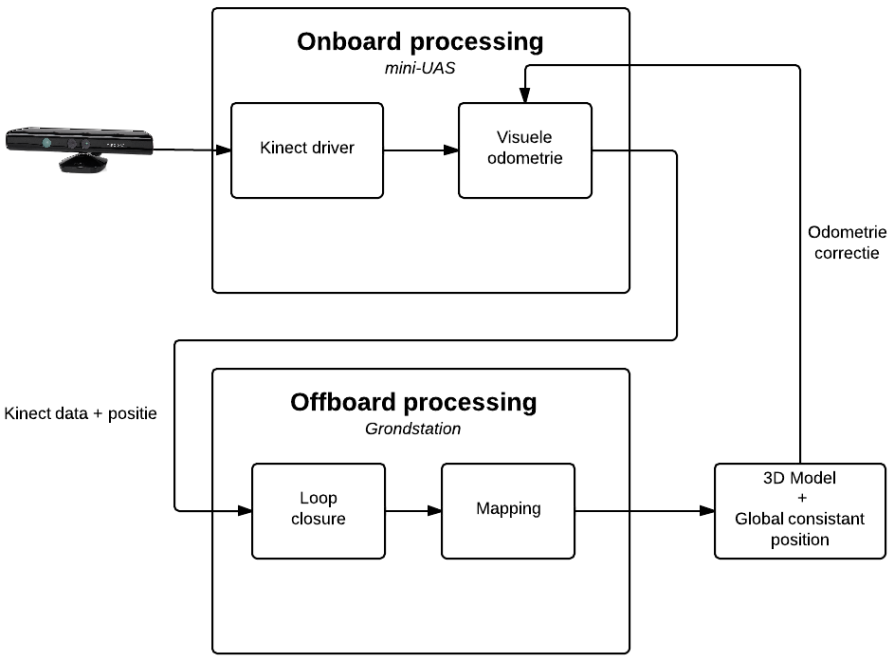
[4] Visualisatie software in ROS. Voor meer informatie: [46]

5 SLAM MET EEN UAV

Zoals beschreven in het vorige hoofdstuk is er al de nodige software beschikbaar voor SLAM met een RGBD sensor. Deze software is zeer rekenintensief. Om deze berekeningen allemaal of deels realtime uit te laten voeren is relatief veel rekenkracht nodig. Deze rekenkracht kan aan board van de UAS gebracht worden of extern gepositioneerd worden met een datalink naar de sensoren die zich op de UAS bevinden. Veel rekenkracht betekent veel energieverbruik. Dit betekent dat als deze rekenkracht zich aan board van de UAS bevind dat de UAS een flinke accu mee moet dragen om het systeem draaiende te kunnen houden. Meestal resulteert dit in een kortere vliegtijd doordat het vele rekenwerk de accu sneller laat leeglopen. Daarnaast mag het systeem dat de berekeningen uitvoert niet te veel wegen in verband met het beperkte lift vermogen van een UAV. En hoe meer het systeem weegt, des te meer energie er nodig is om het systeem in de lucht te houden.
Een andere oplossing waarbij het rekenwerk extern (offboard) wordt uitgevoerd kent ook een aantal nadelige eigenschappen. Zo moet de datalink tussen de drone en het grondstation continue een goede performance hebben om de sensor data van de drone te versturen naar het grondstation waar de berekeningen uitgevoerd kunnen worden. Als deze verbinding niet voldoende robust is en wegvalt, gaat er kostbare sensor data verloren en raakt het algoritme van slag. Ook krijgt de drone dan geen informatie meer terug over zijn huidige positie, waardoor autonoom vliegen niet meer mogelijk is. De bestaande software voor SLAM kan dus niet direct gebruikt worden in combinatie met een UAS.

5.1 GEDISTRIBUEERD SLAM

Het SLAM algoritme is op te delen in twee delen: positie bepaling en mapping. Omdat er niet heel veel rekenkracht beschikbaar is op de drone, maar er toch een bepaalde zekerheid gewenst is als de verbinding wegvalt met het grondstation is het interessant om deze twee delen gedistribueerd uit te voeren. Hierbij zal de lokale positie bepaling op de drone uitgevoerd worden terwijl het mappen en de loop closure uitgevoerd zullen worden op het grondstation. Hierdoor heeft de drone als de verbinding met het grondstation wegvalt toch nog een lokale positie bepaling waardoor deze autonoom kan verder vliegen.



5.1.1 ONBOARD

Omdat het niet wenselijk is dat het systeem afhankelijk is van een extern systeem om zijn lokale positie te bepalen moet de local pose estimation op het systeem aan boord van de UAV uitgevoerd worden. Omdat we te maken hebben met positiebepaling en een vliegend systeem is het een must dat dit realtime uitgevoerd kan worden. Als de data te laat verwerkt wordt kan dit voor problemen zorgen tijdens het autonoom vliegen. Een vliegend systeem heeft in tegenstelling tot een grond voertuig geen wielencoder informatie om transformatiedata van te verkrijgen. Er zal dus op een andere manier bepaald moeten worden met welke snelheid het systeem vliegt en hoever deze verplaatst is. Een van deze mogelijkheden is visuele odometrie. Hierbij worden twee op elkaar volgende frames met elkaar vergeleken en wordt bepaald hoe de pixels of features in de afbeeldingen zijn verschoven ten opzichte van elkaar. Om dit uit te drukken in een metrische schaal moet er echter wel extra informatie aanwezig zijn over de schaal. Met een RGBD sensor kan deze schaal afgeleid worden van de diepte informatie. Door de features te combineren met de diepte informatie kan bepaald worden welke afstand is afgelegd tussen de twee frames en in welke richting.

Na onderzoek naar bestaande implementaties van visuele odometrie met een RGBD-sensor is ervoor gekozen om gebruik te maken van de FOVIS library van MIT (zie H4.3.). Er is gekozen voor deze library om een aantal redenen:

- Goed gedocumenteerde en gestructureerde code
- Snel genoeg voor onboard processing (zie 4.3)
- Werkt ook nog goed met relatief weinig features
- Maakt gebruik van diepte- en kleureninformatie

Voor de FOVIS library is er ook al een ROS wrapper beschikbaar. Ook deze is goed gedocumenteerd en goed gestructureerd. Wel ontbraken er een aantal functies zoals de mogelijkheid tot het runtime aanpassen van parameters en het gebruik maken van een gecomprimeerde stream vanaf de Kinect. De eerste functie is handig voor het fine tunen van het algoritme voor een specifiek systeem. Bijvoorbeeld de parameters zo aanpassen dat het algoritme niet te zwaar is voor het systeem waar het op draait, maar wel nog steeds een goed resultaat geeft. De functie voor het gebruik maken van een gecomprimeerde stream is handig als toch besloten wordt om ook visuele odometrie off board uit te laten voeren. Hierbij kan gedacht worden aan het plaatsen van een Kinect onder een drone die bijna geen rekenkracht aan boord heeft. Deze kan dan een gecomprimeerde stream van de Kinect doorsturen naar het grondstation, waar vervolgens visuele odometrie uitgevoerd wordt. Als de stream niet gecomprimeerd is, wordt er ongeveer 40 MB/s verstuurd. Dit is voor een draadloze verbinding veel te veel. (Zie H3.4). Om beide functies toch te kunnen gebruiken is de ROS wrapper aangepast. Zo kunnen de parameters nu runtime aangepast worden met behulp van Dynamic Reconfigure en kan de FOVIS library nu ook de gecomprimeerde stream van de Kinect gebruiken als input door een parameter te wijzigen.

Zoals beschreven in H3.3.1 heeft ROS een systeem om transformaties tussen frames bij te houden. In dit geval met een UAS is er spraken van de volgende frames:

- /base_link : middelpunt van de UAS
- /camera_link: middelpunt van de camera
- /odom: referentiepunt van odometrie
- /map : referentiepunt van het gecreëerde model

Hierbij is de transformatie tussen /base_link en /camera_link statisch aangezien de camera vast gemonteerd zit op het UAS. De transformatie tussen /camera_link en /odom wordt berekend en gepubliceerd door de onboard software en representeert de transformaties die berekend worden aan de hand van visuele odometrie. Dit houdt in de dat de transformatie staat voor de positie van het UAS ten opzichte van het start punt op basis van

odometrie. Dus zonder enige correctie van bijvoorbeeld het loop closure gedeelte. Dit gedeelte is dus niet afhankelijk van het grondstation en dus kan het UAS op deze manier toch nog deze transformatie informatie gebruiken om zijn positie te bepalen.

5.1.2 OFF BOARD

De minder tijd kritische taken kunnen uitgevoerd worden op het zogenaamde grondstation. Onder deze taken verstaan we het daadwerkelijk opbouwen van het model van de omgeving en het toepassen van de loop closure.

DATA COMMUNICATIE

Voor deze taken moet ook het grondstation sensor data ontvangen van het vliegende systeem. Aangezien de communicatie verloopt via WiFi moet de hoeveelheid data die verstuurd moet worden zo klein mogelijk zijn. Omdat WiFi sterk afhankelijk is van externe factoren kan de performance van WiFi soms sterk afnemen, waardoor het niet wenselijk is om grote hoeveelheden data te versturen. Daarnaast is het ook het meest efficiënt als het offboard gedeelte gebruik maakt van informatie die al is berekend op het vliegende systeem zelf: de positie aan de hand van odometrie. Naast deze informatie van het vliegende systeem heeft het grondstation ook de camerabeelden en de dieptebeelden nodig om een model op te kunnen bouwen. Deze data “ruw” oversturen is geen optie aangezien dit bijna 40 MB/s is (Zie H3.4). Daarom is er voor gekozen om de data van de Kinect te comprimeren en dan over te sturen. Dit kost een klein beetje meer rekenkracht van het vliegende systeem, maar hierdoor is de data stream nog maar 3.5 MB/s. Zodra het aankomt op het grondstation wordt het dan weer gedecomprimeerd en kan het gebruikt worden. Het kleurenbeeld wordt gecomprimeerd als JPG en het dieptebeeld als PNG.

PROCESSING

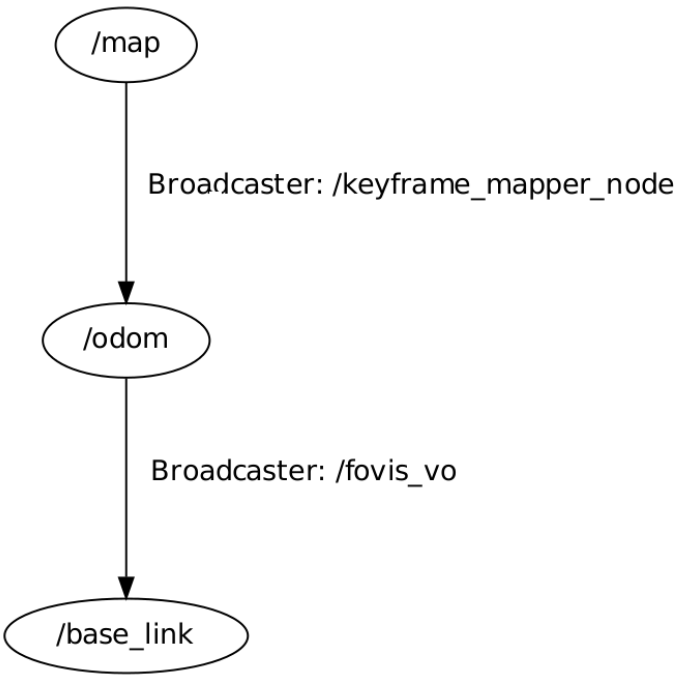
Er is gekeken naar hoe de bestaande oplossingen die genoemd zijn in H4 te gebruiken zijn. Hieruit kwam naar voren dat de implementatie van CCNY het meest geschikt was om de volgende redenen:

- Aparte node voor het creëren van het model
- Goed gedocumenteerd
- Goed gestructureerde code
- Actieve ontwikkeling
- Kan gebruik maken van visuele odometrie informatie die onboard is berekend

Deze library maakt gebruik van zogenaamde keyframes. Als de positie van het vliegende systeem meer als een gedefinieerde drempelwaarde veranderd is, wordt er een keyframe opgeslagen. Dit keyframe bevat het kleurenbeeld, de diepte informatie en de positie van het systeem op dat moment. Deze positie wordt verkregen van de odometrie data die berekend is op het vliegende systeem zelf. Vervolgens wordt dit keyframe vergeleken met een aantal random andere keyframes om loop closure te detecteren. Als deze frames overeenkomsten hebben en het blijkt dat er een loop geclosed is, wordt het graph geoptimaliseerd wat er voor zorgt dat de posities die opgeslagen zijn bij de andere keyframes gecorrigeerd worden. Meer over de technische werking van deze implementatie is te lezen in H4.1.1.

De implementatie van CCNY is zo geschreven dat het verwerken van de keyframes, en dus het maken van de map en het corrigeren van de loop closure, gestart moet worden door de gebruiker. Het proces wordt dus offline uitgevoerd in plaats van online. Hierdoor wordt er tijdens het vliegen geen model gemaakt en dus ook geen correctie uitgevoerd. Dit is niet ideaal omdat het systeem dan alleen de relatief onnauwkeurige odometrie data heeft tijdens het vliegen en daarnaast heeft de operator geen inzicht in hoe de omgeving er uit ziet waar het systeem zich bevind. Daarom is de code aangepast om ook online de keyframes te kunnen verwerken tot een model en correcties van de odometrie kunnen worden toegepast.

Om de optimalisatie stap online te kunnen uitvoeren worden de stappen die in de huidige implementatie op batch basis werden uitgevoerd voor alle keyframes, nu uitgevoerd elke keer als er een keyframe toegevoegd wordt. Elke keer als er een keyframe wordt gegenereerd worden de stappen uitgevoerd die beschreven zijn in H4.2.
Een extra stap die ondernomen moet worden is het creëren van een transformatie om de odometrie te corrigeren. Deze transformatie is de transformatie tussen het frame /map en het frame /odom. Als de odometrie perfect zou zijn, dan zou de transformatie tussen /map en /odom nul zijn. Het model dat opgebouwd wordt tijdens het vliegen heeft als origin het /map frame. Als tijdens het optimalisatie proces de poses van de keyframes veranderen, dan zijn dit de posities ten opzichte van het /map frame. Door deze te vergelijken met de positie zoals deze eerst was ten opzichte van /odom (bij de start is de transformatie tussen /map en /odom nul) kan bepaald worden wat het verschil is in positie en kan dus de transformatie tussen /map en /odom berekend worden. Deze wordt vervolgens gepubliceerd in het /tf topic, het topic in *ROS* waar alle transformatie worden gepubliceerd. De relaties tussen de verschillende frames is te zien in Figuur 15.



FIGUUR 15 DE RELATIES TUSSEN DE VERSCHILLENDE TF FRAMES

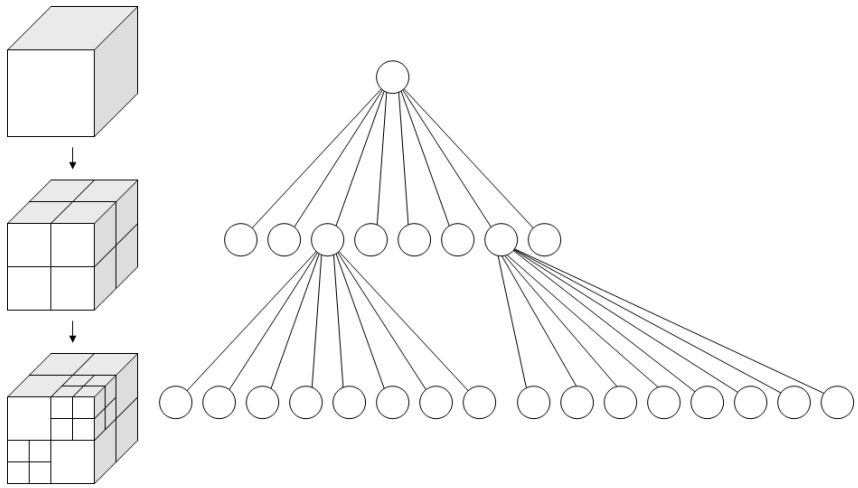
HET CREËREN VAN DE POINT CLOUD

Door het bovenstaande proces is er een verzameling van keyframes met hun corrigeerde positie ontstaan. Dit zijn nu nog geen pointclouds, maar alleen data structures met daarin het kleurenbeeld, het dieptebeeld, de camera positie en enkele andere meta data. In de implementatie van CCNY werd de point cloud pas gecreëerd als de gebruiker hier opdracht toe gaf. Bij dit proces worden eerst alle keyframes omgezet naar kleine point clouds (één point cloud per keyframe). Vervolgens worden deze point clouds getransformeerd aan de hand van de camerapositie die is opgeslagen in het keyframe. Als dit gedaan is worden alle zogenaamde keypointclouds samengevoegd tot één grote point cloud waardoor er een model van de omgeving ontstaat. Dit model kan vervolgens opgeslagen worden of worden getoond aan de operator. Doordat dit proces pas geactiveerd wordt als de gebruiker daar opdracht toe geeft, heeft de operator pas inzicht in de omgeving van de mini-UAS als deze vraagt om het model op te bouwen. Dit is niet ideaal en daarom is er voor gekozen om telkens als de positie in een keyframe wijzigt door de graph optimalisatie stap, de point cloud van dat keyframe wordt

geüpdatet. Vervolgens wordt op een instelbare frequentie al deze keypointclouds samengevoegd tot één point cloud. De frequentie van deze stap is instelbaar met behulp van Dynamic Reconfigure in *ROS* [35].

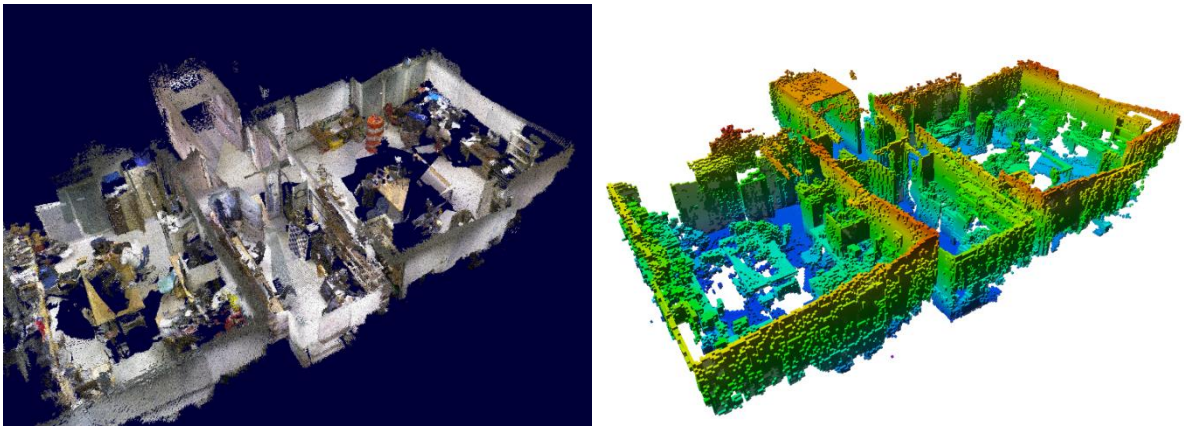
GEHEUGEN EFFICIËNT MODEL

Naarmate het model groeit, groeit ook het aantal punten in de point cloud en dus ook het geheugen gebruik. Daarnaast is het, met het oog op autonome navigatie, niet wenselijk als een algoritme alle punten in het model moet gaan controleren op bijvoorbeeld eventuele collisions. Daarom is er gekeken naar een manier van datareductie waardoor er minder geheugen nodig is en waardoor het voor een (later te ontwikkelen) planningsalgoritme eenvoudiger wordt om een collision vrij pad te vinden. Uit dit onderzoek is OctoMap naar voren gekomen [36]. OctoMap is een library die algoritmes en datastructuren gebaseerd op octrees biedt. Een octree is een representatie van een 3-dimensionale ruimte waarbij elke node géén of acht kindernodes heeft. De root-nodes stelt de volledige 3D ruimte voor, welke vervolgens in acht blokken verdeeld wordt. De kinderen van de rootnode kunnen vervolgens ook weer opgesplitst worden in acht kindernodes per node. Dit principe is weergegeven in Figuur 16. De kindernodes samen omvatten de zelfde informatie als de oudernode. Daarom kan geconcludeerd worden dat als de oudernode niet interessant is, de kindernodes ook niet interessant zullen zijn.



FIGUUR 16 OCTREE PRINCIPE [37]

Voor het creëren van de octomap geldt het zelfde als voor het creëren van de Point Cloud: dit hoeft niet real-time te gebeuren. Er is daarom ervoor gekozen om de octomap met een instelbare frequentie te laten updaten. Met behulp van Dynamic Reconfigure [35] kan ingesteld worden met welke frequentie de octomap geüpdatet moet worden. Hierbij moet rekening gehouden worden met de rekenkracht van het grondstation. Als de updatefrequentie te hoog ligt voor het specifieke grondstation, zullen de overige taken van het grondstation in geding komen. In Figuur 17 is een voorbeeld van het verschil tussen een Point Cloud en een octomap.



FIGUUR 17 POINTCLOUD VS OCTOMAP [38]

5.2 ROS OP DE PELICAN

Als ontwikkelplatform voor dit project is de Asctec Pelican gebruikt. De Pelican die in het UASLab aanwezig is, was nog niet uitgerust met ROS software. Er was wel al een Ubuntu installatie aanwezig op het systeem. Deze installatie was verouderd en niet meer stabiel. Er is daarom voor gekozen om met een schone installatie te beginnen. Er is gekozen voor de Ubuntu 12.04 LTS minimal installatie. De reden voor Ubuntu 12.04 is omdat dit een stabiel besturingssysteem is en welke aanbevolen wordt door de makers van *ROS*. De standaard installatie van Ubuntu is heel uitgebreid en heeft veel software geïnstalleerd die niet interessant is voor aan boord van het UAS. Een voorbeeld hiervan is de grafische user interface. Daarom is er voor gekozen de minimal installatie van Ubuntu te installeren. Deze bevat alleen de basis en benodigde software die ontbreekt, kan als nog gedownload en geïnstalleerd worden. Hierdoor worden er zo weinig mogelijk resources van het systeem verbruikt aan het besturingssysteem.

Van *ROS* zijn er ook verschillende installatie pakketten beschikbaar. Deze variëren van een installatie met alleen de basis componenten tot een installatie met ook zaken als simulators en dergelijken. Net zoals bij de keuze voor Ubuntu minimal is er ook hier gekozen voor de minimale installatie met het daarnaast installeren van de vereiste extra *ROS* packages. Daarnaast is er voor gekozen om de meest recente versie van ROS, ROS groovy, te installeren. Deze versie deels backwards compatible met de vorige versie van ROS, waardoor software die geschreven is voor ROS fuerte bijna altijd ook werkt op ROS groovy. Ook is de verwachting dat software die in de toekomst geschreven zal worden gebaseerd is op de API en het build systeem van ROS groovy.

Zoals aangegeven in H3.1.1 heeft de Pelican, buiten de GPS waypoint navigatie om, geen enkele vorm van autonomie. Dit betekent dat in tegenstelling tot de AR.Drone van Parrot het systeem zichzelf niet stabiel in de lucht kan laten hangen. Daarom kunnen de algoritmen die zijn ontwikkelt voor de AR.Drone nog niet direct gebruikt worden op de Pelican (zie 2.1). De software om het systeem stabiel te laten hangen in de lucht zal dus op het Atom board of op het HPL board moeten komen draaien. Daarnaast zal er een interface moeten zijn naar *ROS* waardoor de UAS stuurcommando's van de algoritmen kan ontvangen. Twee universiteiten, CCNY en ETH Zurich, hebben beiden een interface geschreven voor de Pelican [39] [40].

Een overzicht van de mogelijkheden van beide drivers is te vinden in Tabel 3.

	asctec_drivers	**asctec_mav_framework**
Firmware upgrade HPL vereist	Nee	Ja
Maximale communicatie baudrate	57600 baud	921600 baud
Maximale update frequentie	20 Hz	1kHz
IMU data uitlezen	Ja	Ja
Motoren aansturen	Ja	Ja
State estimation op HPL	Nee	Ja
Position Control	Nee	Ja

TABEL 3 MOGELIJKHEDEN ROS DRIVER PELICAN

Uit de tabel is duidelijk af te lezen dat de software die geschreven is door de ETH Zurich meer mogelijkheden heeft. Zeker de mogelijkheid tot het uitvoeren van state estimation op de HPL is interessant omdat deze er voor zorgt dat het systeem zichzelf stabiliseert. De position control functie is ook een belangrijke feature. Deze kan gebruik maken van de visuele odometrie om de snelheid van de UAS te regelen en op een positie te blijven hangen en om de snelheid waarmee het systeem vliegt te kunnen controleren. Ook wordt gebruik gemaakt van de HPL waardoor er zoveel mogelijk rekenkracht overblijft op het Atom board voor andere rekentaken. Er is daarom voor gekozen om deze software te gebruiken als interface naar Pelican.

6 RESULTAAT

In het vorige hoofdstuk is beschreven dat ervoor gekozen is om gedistribueerd SLAM te realiseren. Hierbij is er gebruik gemaakt van bestaande libraries en zijn deze libraries uitgebreid. In dit hoofdstuk wordt er aandacht besteed aan het resultaat van deze software. Daarnaast wordt er ook aandacht besteed aan de performance van de software. De testresultaten in dit hoofdstuk zijn behaalt met een systeem met een embedded Core i5 processor met twee fysieke cores, 8GB intern geheugen en een SSD. Er is voor de tests gebruik gemaakt van testsets van de TU Munchen, omdat zij groundtruth informatie bij de data leveren. Uiteindelijk is deze groundtruth informatie helaas niet gebruikt.

6.1 SIMULTANEOUS LOCALIZATION AND MAPPING

Met de software die ontwikkeld is kan met behulp van een Kinect sensor een 3D model gemaakt worden van de directe omgeving. Binnen deze omgeving kan de sensor gelokaliseerd worden, en dus ook het systeem waarop de sensor gemonteerd zit. De software bestaat uit twee gedeelten: visuele odometrie en mapping. Deze twee gedeelten kunnen beide op een verschillend systeem draaien. Hierdoor kan er één gedeelte op de mini-UAS uitgevoerd worden en een ander gedeelte op het grondstation. Dit heeft als gevolg dat als de verbinding tussen het grondstation en de mini-UAS onderbroken wordt voor enige tijd, het vliegende systeem nog steeds een vorm van lokalisatie heeft door middel van de lokale visuele odometrie. Voor de communicatie tussen de twee gedeelten wordt gebruik gemaakt van het framework *ROS*.

6.1.1 KINECT EN VISUELE ODOMETRIE OP DE MINI-UAS

De Kinect driver en de visuele odometrie worden uitgevoerd op het vliegende systeem. In H3.4.3 is beschreven dat de datastream van de Kinect erg groot is. Omdat de beelden naar het grondstation verstuurt moeten worden is er gekeken naar een manier om de beelden te comprimeren zoals beschreven in H5.1.1. Het resultaat van de comprimeerstap is dat de datastream van de Kinect significant kleiner is geworden. In Tabel 4 wordt een overzicht gegeven van de grootte van de datastreams, gecomprimeerd en ongecomprimeerd.

	Ruwe data	**Gecomprimeerde data (Max)**
Kleurenbeeld (640x480)	27,7 MB/s (221,6 Mbit/s)	1,5 MB/s (12 Mbit/s)
Diepte informatie (640x480)	37,3 MB/s (298,4 Mbit/s)	2,9 MB/s (23,2 Mbit/s)

TABEL 4 BANDBREEDTE VERBUIK VAN DE DATASTREAM VAN DE KINECT

Bij de ruwe data is het bandbreedteverbruik altijd gelijk. Per pixel worden er dan drie bytes overgestuurd voor het kleurenbeeld en vier bytes voor de diepte informatie. Bij de gecomprimeerde stream is het afhankelijk van de verschillen tussen de pixels. Bij een zwart beeld zal de stream veel kleiner zijn als bij een plaatje met veel variatie. Dit geldt ook voor de diepte informatie. Hoe meer variatie in diepte, des te groter is de hoeveelheid data. Voor het testen van de gecomprimeerde stream is er met de Kinect rondgelopen door het lab en is het hoogst gemeten bandbreedteverbruik genomen. Het comprimeren van de informatie kost wel extra rekenkracht. In tabel x is een overzicht gegeven van de verschillen tussen het doorsturen van een ongecomprimeerde stream en het versturen van een gecomprimeerde stream.

	Ruwe data	**Gecomprimeerde data**
Kleurenbeeld (640x480)	11 % CPU load	18 % CPU load[5]
Diepte informatie (640x480)	35 % CPU load	91 % CPU load[4]

TABEL 5CPU LOAD BIJ HET VERSTUREN VAN KINECT DATA

Voor de visuele odometrie wordt gebruik gemaakt van zowel de diepte informatie als het kleurenbeeld. In dit kleurenbeeld worden features gedetecteerd en deze worden in combinatie met de diepte informatie

[5] Verdeelt over 2 cores

opgeslagen. Het idee hierachter is dat de visuele odometrie ook werkt in gebieden die minder feature rijk zijn, maar nog wel voldoende reliëf bevatten om iets te kunnen met de diepte informatie. De diepte informatie wordt alleen niet op deze manier gebruikt. In de library FOVIS, die gebruikt is voor de visuele odometrie, wordt de diepte informatie gebruikt om de features op die positie te plotten en de diepte informatie te gebruiken als extra informatie van de feature. Als er geen features zijn, zal dus ook de diepte informatie niet gebruikt worden. Nu wordt de diepte informatie gebruikt om de metrische schaal te bepalen. Het resultaat van de visuele odometrie is ronduit goed te noemen in gebieden waar voldoende features beschikbaar zijn. Dit resultaat is gebaseerd op visuele waarnemingen. Door de berekende transformaties te visualiseren en deze te vergelijken met de daadwerkelijk bewegingen is geconcludeerd dat het algoritme goed werkt.

Aan de library zijn een aantal functies toegevoegd zoals beschreven in H5.1.1. Het resultaat van deze toevoegingen is dat de parameters van het algoritme runtime kunnen worden aangepast. Op deze manier kan het algoritme eenvoudig gefinetuned worden zonder telkens de software opnieuw te compileren. Daarnaast kan de software nu ook omgaan met de gecomprimeerde datastream van de Kinect. Hierdoor kan het de software voor de visuele odometrie ook extern draaien als de Kinect bijvoorbeeld onder een systeem hangt dat bijna geen rekenkracht aan boord heeft.

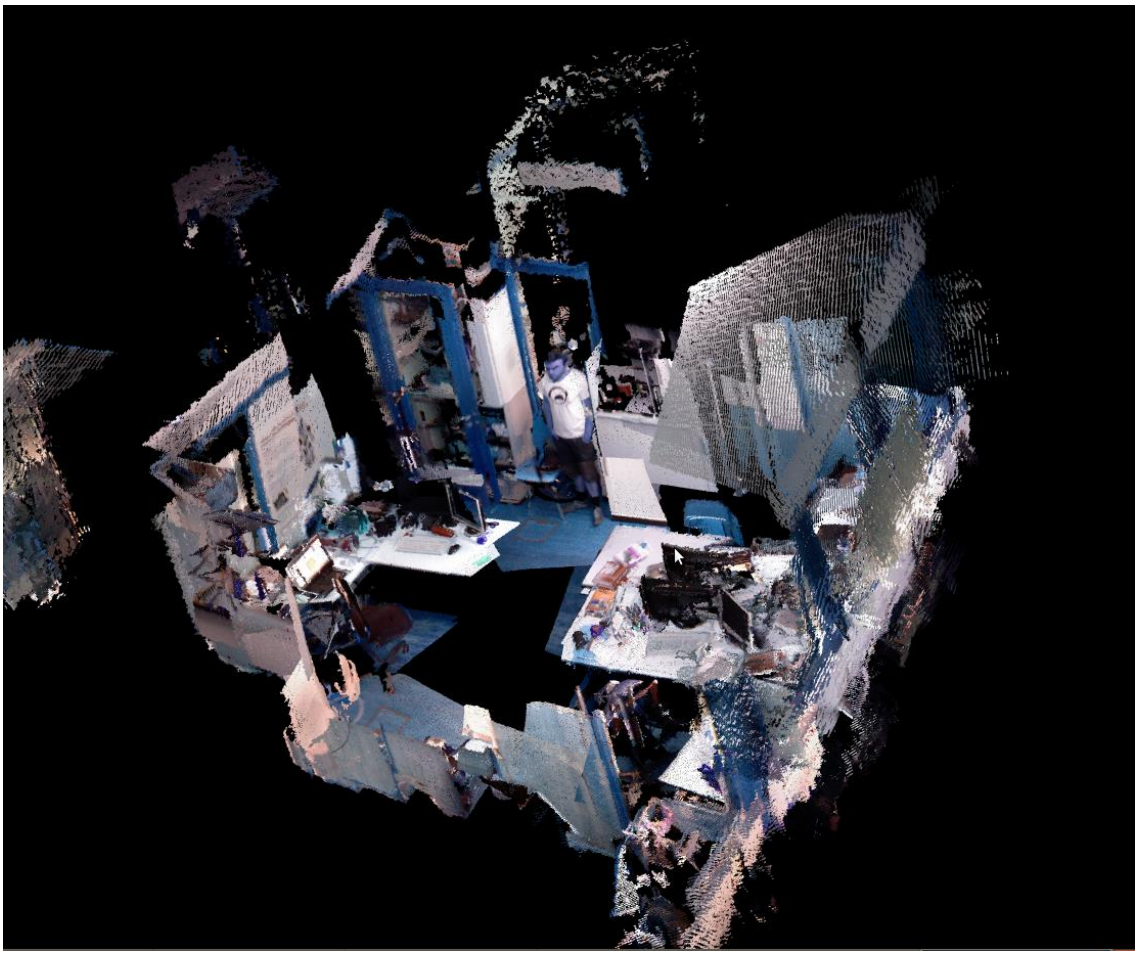
Omdat de visuele odometrie op het vliegende systeem berekend moet worden is getest hoeveel systeem resources de software gebruikt. Hieruit is het volgende resultaat gekomen:

	CPU Usage	**Memory Usage**
Visuele odometrie (fovis)	38 %	34 MB

In combinatie met de Kinect driver is dit te rekenintensief voor het processorboard met de Atom dat op de Pelican zit gemonteerd. De CPU wordt dan voor 100% belast en op deze manier kan niet gegarandeerd worden dat het algoritme op tijd de frames verwerkt. Het is daarom ook niet mogelijk om een reele performance test te doen op de Pelican omdat de CPU continue de maximale rekenkracht gebruikt.

6.1.2 Mapping en loop closure op het grondstation

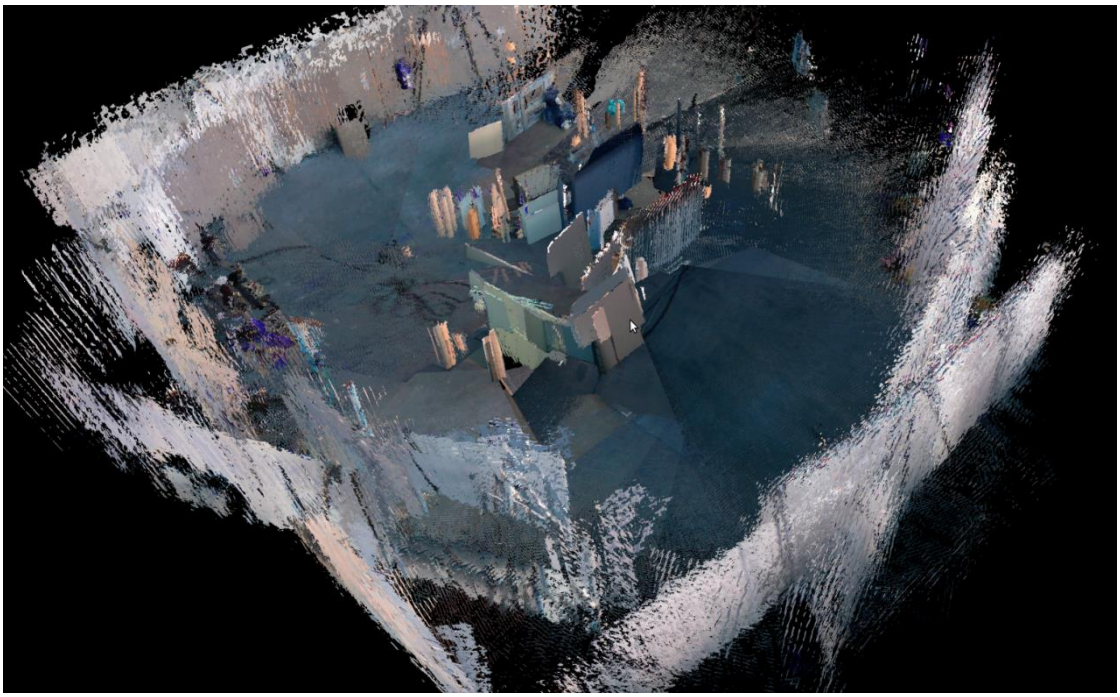
Het mapping algoritme en de loop closure worden uitgevoerd op het grondstation. Door middel van een WiFi verbinding ontvangt het grondstation de Kinect data en de positie op basis van de visuele odometrie. Als basis voor het algoritme is gebruik gemaakt van de library van CCNY welke beschreven staat in H4.2. Dit algoritme miste de optie om het algoritme online uit te voeren. Hierdoor werden de graph optimalisatie en het bouwen van het model alleen gedaan op batch basis. De software is zo aangepast dat het algoritme wel online uitgevoerd wordt, waardoor de operator tijdens het vliegen inzicht heeft in omgeving waar het systeem vliegt. Het resultaat van deze aanpassingen is dat het nu mogelijk is om online een model te creëren. Daarnaast wordt de graph optimalisatie, die onder meer de loop closure verzorgt, online uitgevoerd. Hierdoor wordt de fout in de visuele odometrie ook online gecorrigeerd. Dit laatste heeft als voordeel dat als het systeem in een later stadium autonoom gaat vliegen, deze uit kan gaan van de gecorrigeerde positie. Het resultaat van een gecreëerde map is te zien in Figuur 18.



FIGUUR 18 RESULTAAT VAN HET MAPPING ALGORITME

Het bovenstaande resultaat is gemaakt met een dataset die vrij beschikbaar is op de website van TU Munchen [27]. Deze set bevat de ruwe camera beelden van de Kinect. De gebruikte set, Sequence 'freiburg1_room', is opgenomen in een kantoor welke veel features bevat en daarnaast is tijdens het opnemen een rondje door de kamer gelopen waardoor ook de loop closure getest kan worden. Zoals te zien in Figuur 18 werkt het mapping algoritme naar behoren en werkt ook de loop closure (het model sluit netjes op elkaar aan).

De zelfde test is ook gedaan met een testset van een grotere ruimte. Hierbij stond de Kinect regelmatig gericht in richtingen waar de afstand tot het dichtstbijzijnde obstakel verder was dan het dieptebereik van de Kinect. Daarnaast bevat de ruimte waar de beelden zijn opgenomen weinig features. Aan het begin van de set weet de visuele odometrie het pad goed bij te houden. Maar zodra de Kinect wordt gericht in de richting waar de diepte meer dan 3,5m bedraagt, verliest de visuele odometrie de track en vervolgens krijgt het mapping algoritme ook de verkeerde positie binnen. Deze posities zijn dus danig uit de richting dat deze ook niet meer gecorrigeerd kunnen worden tijdens de graph optimalisatie. Het resultaat dat de software genereert met deze data set is te zien in Figuur 19.



FIGUUR 19 RESULTAAT VAN DE SOFTWARE MET EEN MINDER FEATURE RIJKE DATASET

6.2 ROS OP DE PELICAN

Naast het onderzoek naar lokalisatie aan de hand van SLAM is er ook gekeken naar hoe de Pelican UAS uitgerust kan worden met ROS ondersteuning. Op het Atom board aan boord van de Pelican is Ubuntu geïnstalleerd en daar op is ROS geïnstalleerd. Naast de standaard *ROS* componenten is er gekeken naar hoe de motoren aangestuurd kunnen worden en hoe de sensoren uitgelezen kunnen worden vanuit *ROS*. Door twee universiteiten is er software geschreven voor ROS welke een interface bieden naar de motoren en de sensoren aan boord van de Pelican. Er is gekozen voor de software van de universiteit ETH in Zurich omdat deze meer mogelijkheden biedt, waaronder stabilisatie software voor op het High level processorboard van de Pelican. Door de ROS installatie op het processingboard en de software voor de interfacing met de Pelican sensoren en actuatoren is het in theorie mogelijk om de ontwikkelde algoritmen die werken op de AR.drone ook te kunnen gebruiken op de Pelican. Maar omdat de position control software van de ETH afhankelijk is van visuele odometrie, is het niet getest in de praktijk. De visuele odometrie software in combinatie met de Kinect driver is te zwaar voor het Atom processingboard, waardoor niet vertrouwd kan worden op deze positiebepaling en het niet veilig is om het systeem autonoom te laten vliegen.

7 CONCLUSIE

Het doel van dit project was het onderzoek doen naar de mogelijkheden tot het gebruiken van een Kinect in combinatie met SLAM om een alternatieve manier van lokalisatie te ontwikkelen voor in gebieden waar geen GPS beschikbaar is. Hierbij is eerst gekeken naar bestaande oplossingen en is onderzocht in hoeverre deze bruikbaar zijn voor deze situatie. Omdat het gaat om de lokalisatie van een vliegend systeem is de beschikte rekenkracht beperkt. Er is gekozen voor een gedistribueerde opzet waarbij er een gedeelte van de berekeningen onboard en een gedeelte offboard berekend worden. Op deze manier is het vliegende systeem niet volledig afhankelijk van het grondstation en kan deze nog steeds zijn eigen positie bepalen als de verbinding met het grondstation verloren is.

In de praktijk is gebleken dat deze manier van lokalisatie op een mini-UAS potentie heeft maar nog geen volwaardige vervanger is voor GPS. Het algoritme is te sterk afhankelijk van features in het beeld en het bereik van de Kinect is te klein, waardoor de visuele odometrie track kwijt raakt en hierdoor heeft ook het mapping algoritme moeite om een mooi model te creëren dat op alle plekken netjes op elkaar aansluit. Vervolgens wijkt de bepaalde positie steeds meer af van de werkelijkheid. Daarnaast is door de eigenschappen van de Kinect deze manier van lokalisatie niet geschikt voor outdoor.

De uiteindelijke conclusie is dat deze manier van lokaliseren en mappen zeer veel potentie heeft, maar dat er nog veel verbeterd kan worden. Een aantal suggesties voor verbeterpunten zijn beschreven in H8.

Een ander onderdeel van het project was het uitrusten van de Pelican met ROS ondersteuning. Hiervoor is Ubuntu met het ROS framework geïnstalleerd op het Atom board. Daarnaast is er software geïnstalleerd van de universiteit ETH Zurich op de motoren en de sensoren aan boord van de Pelican te interfacen. Door deze softwarepakketen is het in mogelijk dat de ontwikkelde algoritmen voor de AR.Drone ook uitgevoerd kunnen worden op de Pelican, mits er voldoende rekenkracht aanwezig is op de Pelican.

8 AANBEVELINGEN / FOLLOW UPS

Naar aanleiding van dit project zijn er nog een aantal zaken die verbeterd en uitgebreid kunnen worden. Deze worden in dit hoofdstuk aangehaald en hiervan wordt een korte beschrijving geven.

8.1 ONDERZOEK NAAR ANDERE SENSORS

Tijdens dit project is de Kinect sensor gebruikt. De prijs-kwaliteit verhouding van deze sensor is prima te noemen. Maar de sensor heeft een aantal nadelen. Zo bevat de diepte informatie veel ruis en is deze zeer gevoelig voor zonlicht. Voor indoor omgevingen levert dit niet veel problemen op, maar voor outdoor is deze sensor niet geschikt. Daarnaast heeft de Kinect qua diepte waarneming maar een beperkt bereik: 0.8 – 3.5m. In zeer grote ruimtes, bijvoorbeeld loodsen, is dit te beperkt. Daarnaast kunnen objecten die op een afstand van minder dan 80cm van de Kinect staan niet worden waargenomen, waardoor de Kinect niet de meest geschikte sensor is voor obstacle avoidance. Het is daarom aan te raden onderzoek te doen naar andere sensoren om de tekortkomingen van de Kinect te compenseren.

8.2 EXPLORATION

Met de ontwikkelde software is het nu mogelijk om een 3D model te creëren van de omgeving. Op dit moment moet de mini-UAS hiervoor nog gevlogen worden door een human operator. Een vervolg op dit project zou een onderzoek naar een exploration strategie kunnen zijn. Door een algoritme te ontwikkelen dat kan bepalen welke gebieden van de omgeving nog niet in kaart gebracht zijn, kan de mini-UAS zelf het gebied gaan verkennen. Dit heeft als voordeel dat de bediener van het apparaat dan bezig kan zijn met andere zaken, zoals het analyseren van de live video verbinding.

8.3 EFFICIËNTER ALGORITME

Het algoritme dat nu gebruikt wordt is nog steeds zeer rekenintensief. Omdat de zwaarste taken offboard uitgevoerd worden, levert dat niet direct problemen op. Maar het zou wenselijk zijn als het algoritme minder rekenintensief zou zijn, zodat er zoveel mogelijk onboard berekend kan worden en zodat er nog rekenkracht overblijft voor eventuele andere taken.

8.4 MEER GEBRUIK MAKEN VAN DIEPTE INFORMATIE

Op dit moment wordt de diepte informatie gebruikt voor het maken van het model. Daarnaast wordt de diepte informatie ook gebruikt door de visuele odometrie, maar in beperkte mate. De diepte informatie wordt alleen gebruikt als er in het beeld van de kleurencamera op die positie een feature is gedetecteerd. Dit betekent als er wel betrouwbare diepte informatie beschikbaar is maar het kleurenbeeld weinig features bevat, dat de visuele odometrie geen gebruik maakt van deze informatie. Een uitbreiding zou zijn om de visuele odometrie uit te breiden door deze ook gebruik te laten maken van de diepte informatie als er weinig features aanwezig zijn.

8.5 SOFTWARE UITVOEREN OP GPU

Om de performance van de software te verbeteren is het mogelijk interessant om te kijken naar de mogelijkheden van het gebruik maken van de GPU. De huidige software is nu zo geschreven om alleen gebruik te maken van de CPU. Door bepaalde taken parallel uit te voeren op de GPU kan er een performance winst geboekt worden en mogelijk kan er dan ook een beter resultaat behaald worden.

8.6 MEER ONBOARD PROCESSING POWER

Op dit moment beschikt de Pelican over een processorboard met een Intel Atom processor. Uit de tests is gebleken dat dit processorboard te zwak is om de Kinect driver en de visuele odometrie software te draaien (zie H6.1.1). Mogelijk is het interessant om meer onboard processing power naar de Pelican te brengen. Hierdoor is het systeem beter in staat om de visuele odometrie uit te voeren en is er meer rekenkracht over voor eventuele andere onboard taken.

8.7 DATA FUSION

Een vervelende eigenschap van visuele odometrie is dat deze sterk afhankelijk is van de hoeveelheid features in het beeld. Als de camera gericht wordt op een muur met één effen kleur, dan is de kans groot dat hier geen of weinig features uitgehaald kunnen worden. Hierdoor kan niet bepaald worden of het systeem verplaatst is ja of nee. Om de gevolgen van dit probleem in te perken kan er gekeken naar de mogelijkheid om ook gebruik te maken van de onboard IMU-informatie. Door ook te kijken naar de informatie van de gyro- en accelerometer en het magnetisch kompas kan ook bepaald worden of de stand van het systeem veranderd is. Naast dat deze informatie beschikbaar is als visuele odometrie faalt, kan deze informatie ook vaker worden opgevraagd. De IMU data van de Pelican wordt bijvoorbeeld met een frequentie van 1kHz geüpdatet [41]. Voor *ROS* is er al software geschreven die gebruik maakt van een Extended Kalman filter[6] met een 6D model (6 vrijheidsgraden) om informatie van verschillende odometrie sources (visuele odometrie, wielencoders, IMU, enz) te combineren om op die manier een zo nauwkeurig mogelijk de positie van een robot te bepalen [42].

[6] Een filteralgoritme om ruis uit data te verwijderen [45]

9 BRONVERMELDING

[1] G. ten Buuren, „Small UAS for Law Enforcement," in *2011-2012 UAS Yearbook - UAS: The Global Perspective*, 2011, pp. 134-137.

[2] „FireSwarm | Onbemande vliegtuigjes voor bosbranden," Almende B.V., [Online]. Available: http://fireswarm.nl/. [Geopend 02 2013].

[3] R. Paul, „UAVs / Drones and Precision Agriculture," [Online]. Available: http://aerialfarmer.blogspot.nl/. [Geopend 02 2013].

[4] ORBIT GeoSpatial Technologies NV, „Efficiënte updates van je kaart met UAV'S," in *Orbit - Magazine for Geospatial Technologies*, September 2012, pp. 10-15.

[5] „AR.Drone 2.0. Parrot new wi-fi quadricopter," Parrot, [Online]. Available: http://ardrone2.parrot.com. [Geopend 02 2013].

[6] „AscTec Pelican," Ascending Technologies, [Online]. Available: http://www.asctec.de/uav-applications/research/products/asctec-pelican/. [Geopend 02 2013].

[7] „OktoKopter XL ARF," [Online]. Available: http://www.mikrokopter.de/ucwiki/ArfOktoXL. [Geopend 03 2013].

[8] [Online]. Available: http://www.ros.org/wiki/.

[9] „Robots/Asctec - ROS Wiki," [Online]. Available: http://www.ros.org/wiki/Robots/AscTec. [Geopend 02 2013].

[10] A. J. Davison, Y. G. Cid en N. Kita, „Real-Time 3D SLAM with Wide-Angle Vision," in *Proc IFAC Symposium on Intelligent Autonomous Vehicles*, 2004.

[11] S. Kohlbrecher, O. von Stryk, J. Meyer en U. Klingauf, „A Flexible and Scalable SLAM System with Full 3D Motion Estimation," in *Proc. IEEE International Symposium on Safety, Security and Rescue Robotics (SSRR)*, 2011.

[12] F. Endres, J. Hess, N. Engelhard, J. Sturm en W. Burgard, „An Evaluation of the RGB-D SLAM System," in *Proc. of the IEEE Int. Conf. on Robotics and Automation (ICRA)*, 2012.

[13] T. Foote, E. Marder-Eppstein en W. Meeussen, „tf - ROS Wiki," [Online]. Available: http://www.ros.org/wiki/tf. [Geopend 01 05 2013].

[14] R. Rusu en S. Cousins, „3D is here: Point Cloud Library (PCL)," in *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, Shanghai, 2011.

[15] „PCL - Point Cloud Library," Willow Garage, [Online]. Available: http://pointclouds.org/. [Geopend 05 2013].

[16] „Hacker wins contest for open-source Kinect driver," [Online]. Available: http://news.cnet.com/8301-13772_3-20022467-52.html. [Geopend 05 2013].

[17] „OpenKinect," [Online]. Available: http://openkinect.org/wiki/Main_Page. [Geopend 05 2013].

[18] „OpenNI | The standard framework for 3D sensing," [Online]. Available: http://www.openni.org/. [Geopend 05 2013].

[19] „avin2/SensorKinect - Github," [Online]. Available: https://github.com/avin2/SensorKinect. [Geopend 05 2013].

[20] „Openni/Contests/ROS 3D/RGBD-6D-SLAM - ROS Wiki," 2011. [Online]. Available: http://www.ros.org/wiki/openni/Contests/ROS%203D/RGBD-6D-SLAM. [Geopend 03 2013].

[21] N. Engelhard, F. Endres, J. Hess, J. Sturm en W. Burgard, „Real-time 3D visual SLAM with a hand-held camera," in *In Proc. of the RGB-D Workshop on 3D Perception in Robotics at the European Robotics Forum*, 2011.

[22] H. Bay, T. Tuytelaars en L. van Gool, „SURF: Speeded Up Robust Features," in *European Conference on Computer Vision*, Graz, 2006.

[23] M. A. Fischler en R. C. Bolles, „Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography," *Communications of the ACM,* vol. 24, nr. 6, pp. 381-395, 1981.

[24] R. Hartley en A. Zisserman, Multiple View Geometry in Computer Vision, Cambridge University Press, 2004.

[25] P. J. Besl en N. D. McKay, „A Method for Registration of 3-D Shapes," *IEEE Transactions on Pattern Analysis and Machine Intelligence,* vol. 14, nr. 2, pp. 239-256, 1992.

[26] G. Grisetti, R. Kuemmerle, C. Stachniss, U. Frese en C. Hertzberg, „Hierarchical Optimization on Manifolds for Online 2D and 3D Mapping," in *International Conference on Robotics and Automation (ICRA)*, Anchorage, 2010.

[27] Computer Vision Group - Datasets - RGB-D SLAM Dataset and Benchmark, [Online]. Available: http://vision.in.tum.de/data/datasets/rgbd-dataset. [Geopend 04 2013].

[28] I. Dryanovski, R. G. Valenti en J. Xiao, „Fast Visual Odometry and Mapping from RGB-D Data," in *International Conference on Robotics and Automation (ICRA)*, Karlsruhe, 2013.

[29] M. Muja en D. G. Lowe, „Fast Approximate Nearest Neighbors with Automatic Algorithm Configuration," in *International Conference on Computer Vision Theory and Application*, 2009.

[30] A. S. Huang, A. Bachrach, P. Henry, M. Krainin, D. Maturana, D. Fox en N. Roy, „Visual Odometry and Mapping for Autonomous Flight Using an RGB-D Camera," in *Int. Symposium on Robotics Research (ISRR)*, Flagstaff, Arizona, USA, 2011.

[31] „fovis - Fast Odometry from VISion," [Online]. Available: http://code.google.com/p/fovis/.

[32] E. Rosten en T. Drummond, „Machine learning for high-speed corner detection," in *In European Conference on Computer Vision*, 2006.

[33] C.-M. Kuan, „Nonlinear Least Squares Theory," 2004. [Online]. Available: http://homepage.ntu.edu.tw/~ckuan/pdf/et01/ch8.pdf.

[34] „fovis_ros - ROS Wikie," [Online]. Available: http://www.ros.org/wiki/fovis_ros. [Geopend 05 2013].

[35] „dynamic_reconfigure - ROS Wiki," [Online]. Available: http://ros.org/wiki/dynamic_reconfigure.

[36] A. Hornung, K. M. Wurm, M. Bennewitz, C. Stachniss en W. Burgard, „OctoMap: An Efficient Probabilistic 3D Mapping Framework Based on Octrees," *Autonomous Robots,* 2013.

[37] „Schematic drawing of an octree, a data structure of computer science.," [Online]. Available: http://en.wikipedia.org/wiki/File:Octree2.svg.

[38] „ccny_rgbd / keyframe_mapper - ROS Wiki," [Online]. Available: http://www.ros.org/wiki/ccny_rgbd/keyframe_mapper.

[39] I. Dryanovski, „asctec_drivers - ROS Wiki," [Online]. Available: http://www.ros.org/wiki/asctec_drivers.

[40] M. Achtelik, „asctec_mav_framework," [Online]. Available: http://www.ros.org/wiki/asctec_mav_framework. [Geopend 05 2013].

[41] „Flight Dynamics & Flight Control - Ascending Technologies," Ascending Technologies, [Online]. Available: http://www.asctec.de/uav-applications/research/applications/applications-2/flight-dynamics/. [Geopend 05 2013].

[42] W. Meeussen, „robot_pose_ekf - ROS Wiki," [Online]. Available: http://www.ros.org/wiki/robot_pose_ekf. [Geopend 05 2013].

[43] „Structured Light - Maxence," [Online]. Available: http://maxenceparache.blogspot.nl/2011/03/structured-light.html.

[44] „tf - ROS Wiki," [Online]. Available: http://ros.org/wiki/tf.

[45] M. I. Ribeiro, „Kalman and Extended Kalman Filters," 2004. [Online]. Available: http://users.isr.ist.utl.pt/~mir/pub/kalman.pdf. [Geopend 05 2013].

[46] „RViz - ROS Wiki," [Online]. Available: http://www.ros.org/wiki/rviz.

EVALUATIE

DE VOLGENDE TAKEN VERLIEPEN GOED:

- **Samenwerking**

Ondanks dat ik dit project zelfstandig heb uitgevoerd heb ik ook veel samengewerkt met mijn directe collega's. Deze samenwerking verliep goed. Als er iemand vast zat in probleem hielp ik deze persoon en als ik vast liep stonden mijn collega's ook voor mij klaar.

- **Aanpak**

Het probleem dat centraal stond tijdens dit project is een belangrijk probleem binnen de robotica. Het is dan ook onmogelijk om dit probleem in een afstudeerperiode volledig op te lossen. Ik heb daarom aan het begin van de periode goed gekeken naar eerdere onderzoeken en heb hier veelvuldig gebruik van gemaakt om toch een zo goed mogelijk resultaat neer te zetten.

- **Ontwikkeling van software**

De realisatie van de software ging goed. Ik heb gebruik gemaakt van verschillende libraries en heb de zelf geschreven code zo goed mogelijk van commentaar voorzien. Daarnaast heb ik gebruik gemaakt van *ROS*, wat in het UASLab ook gebruik wordt voor andere software en algoritmen.

- **Communicatie met begeleiders**

Wekelijks had ik met mijn begeleider van het NLR, Gerald Poppinga, een overlegmoment. We spraken dan door wat de stand van zaken was en of alles goed verliep. Daarnaast kon ik Gerald altijd aanspreken als hij aanwezig was of anders een mailtje sturen. De communicatie verliep goed en heb ik als zeer fijn ervaren.

WAT KON ER VERBETERD WORDEN:

- **Planning volgen**

Ondanks dat de geplande taken uitgevoerd zijn, zou het volgen van de planning beter kunnen. De planning zat goed in elkaar en zou ook haalbaar zijn. Doordat ik documenteren het minst leuke werk vind, ben ik te laat begonnen met het schrijven van de scriptie waardoor het de laatste paar weken erg druk was. Als ik me beter aan de planning had gehouden, had ik het beter kunnen spreiden en was het minder druk geweest.

Dedicated to innovation in aerospace

PLAN VAN AANPAK

SIMULTANEOUS LOCALIZATION AND MAPPING IN 3D ENVIRONMENTS USING A MINI-UAS

Naam: Robbert Proost
Studentnr: 1563124
Datum: 14-03-2013

Goedkeuring

Dit document is goedgekeurd door

Naam	Handtekening	Datum goedkeuring

INHOUD

Plan van Aanpak 1
Inleiding 4
Context 4
Probleemstellingen 4
Doelstelling 5
Opdracht 5
Randvoorwaarden 6
Producten 6
Planning 6
Bedrijfs-/Persoonsgegevens 7
Projectleden 7
Begeleiders/Opdrachtgever 7
Bedrijfsgegevens 7
Bibliografie 8

INLEIDING

Onbemande vliegende systemen van diverse formaten worden door een groeiend aantal gebruikers ingezet voor een groeiend aantal taken. Geleidelijk maar gestaag groeien deze systemen daarmee uit tot een belangrijk hulpmiddel in een steeds breder wordende verzameling vaak civiele toepassingsgebieden. Hierbij kan gedacht worden aan een rol bij de inspectie van hoge of moeilijk te bereiken objecten, het creëren van overzicht binnen de handhaving van openbare orde en veiligheid [1], bij de detectie van (bos- en duin-)branden, als middel voor precision farming doeleinden, als instrument voor landmeetkundige toepassingen en nog veel meer.

Momenteel bestaan er nog de nodige belemmeringen bij het inzetten van vliegende onbemande systemen. Zo hebben (potentiële) eindgebruikers vaak slechts een informatiebehoefte, maar is er veel kennis van het systeem en de omgeving benodigd voordat het ingezet kan worden. Ook vergt het vliegen met (meerdere) systemen vaak complexe vaardigheden van bedienaars. Het Nationaal Lucht- en Ruimtevaartlaboratorium onderzoekt daarom onder andere methoden om onbemande systemen beter inzetbaar te maken waardoor het voor eindgebruikers makkelijker wordt om met een systeem te vliegen. Belangrijke aandachtsgebieden hierbij zijn Computer Vision, Machine Learning, Multi-Agent Technology, Data Fusion en Robustly Bounded Autonomy.

CONTEXT

Het UAS lab beschikt over een vloot van mini-UAS, variërend van vastvleugelige tot (multi-)rotorcraft (zoals de Parror AR.drone [2], Ascending technologies Pelican [3] en de Oktokopter [4]), verschillende sensoren en vele moderne processor systemen. Daarnaast wordt binnen het UAS Lab voor het ontwikkelen van nieuwe functionaliteit gebruikt gemaakt van het Robot Operating System (ROS) [5]. Door ROS wordt het relatief eenvoudig om allerlei ondersteunde sensoren (zoals laser scanners en structured light oplossingen) en platformen (zoals de Ar.drone en de Pelican) met elkaar te integreren. ROS maakt het daarnaast eenvoudig om functionaliteit fysiek binnen het netwerk te verplaatsen; Initieel kan er bijvoorbeeld een beeldbewerkingsalgoritme op een laptop met de goedkope Ar.drones ontwikkeld en getest worden, om het vervolgens met minimale inspanning naar de on-board processing power van de Pelican te verplaatsen.

PROBLEEMSTELLINGEN

In het kader van deze opdracht zijn er twee probleemstellingen. Deze zullen hier kort worden toegelicht.

LOKALISATIE

Een groot probleem wat de autonomie van UAS systemen beperkt is de lokalisatie van het systeem binnen een bepaalde omgeving. Om het systeem autonoom te kunnen laten navigeren in een ruimte/gebied moet het systeem kennis hebben van zijn locatie. Op dit moment wordt dit grotendeels gedaan met behulp van Global Positioning System (GPS). Deze manier van lokaliseren werkt goed voor open gebieden met weinig bebouwing, maar levert als snel problemen in druk bebouwde gebieden en werkt binnen in gebouwen meestal helemaal niet. Op die plekken is er namelijk geen GPS signaal beschikbaar. Hierdoor kan een UAS systeem zich niet of moeilijk zijn locatie bepalen in deze gebieden en zou het UAS niet autonoom kunnen navigeren.

ROBOT OPERATING SYSTEM (ROS)

Op dit moment werkt de Pelican in het UAS lab nog niet onder ROS, maar er is wel software beschikbaar voor de integratie van de Pelican binnen ROS [6]. De al geschreven on-board software voor de bediening van de Pelican met behulp van een tablet draait nog niet onder ROS. Doordat de Pelican nog niet onder ROS werkt kunnen reeds ontwikkelde algoritmen, die getest zijn op de AR.Drone, nog niet gedraaid en getest worden op de Pelican.

DOELSTELLING

LOKALISATIE

Op dit moment is er buiten GPS geen enkele andere positie bepaling mogelijk op de UAS systemen van het UAS lab van het NLR. Het streven binnen het UAS lab is om alles zo low-cost mogelijk proberen te houden om zo de mogelijkheden van UAS zoveel mogelijk toegankelijk te maken. De doelstelling voor dit project is om een alternatieve lokalisatie methode te realiseren voor gebieden waar geen GPS beschikbaar is, met behulp van low-cost sensor(en). De positie bepaling van het UAS zal een relatieve positie bepaling worden waarbij het nul punt het punt is waar het UAS is opgestegen. Aan het einde van het project zal dus het UAS zijn positie kunnen bepalen ten opzichte van het punt waar het systeem is opgestegen.

Daarnaast is het doel om dit alles zorgvuldig te documenteren. Het lokalisatie probleem is een probleem dat niet in één afstudeerperiode perfect opgelost kan worden, waardoor de kans groot is dat er na mijn afstuderen iemand anders met dit probleem verder gaat. Het is daarom van belang dat gemaakte keuzes, problemen waar tegenaan gelopen is en andere relevante informatie duidelijk worden gedocumenteerd zodat een eventueel vervolg soepel van start kan gaan.

ROBOT OPERATING SYSTEM (ROS)

Het UAS lab wil in de toekomst alle UAS systemen met het Robot Operating System uitrusten. Omdat het Robot Operating System dienst als abstractie laag kan reeds ontwikkelde software voor ROS relatief eenvoudig verplaatst worden naar andere platformen. De AR.Drone van Parrot is reeds uitgerust met deze software, maar de Pelican nog niet. Een doelstelling voor dit project is dat de Pelican uitgerust worden met de ROS software zodat op deze platformen de ontwikkelde algoritmen ook gebruikt kunnen worden.

OPDRACHT

LOKALISATIE

Om het probleem met de lokalisatie aan te pakken zal er gekeken worden naar oplossing voor het lokaliseren op plekken waar geen GPS signaal is. Hierbij moet gedacht worden aan algoritmen waarmee een 3D model gemaakt kan worden van de omgeving, in dit geval die van het UAS. Tegelijkertijd kan de positie van het UAS binnen het 3D model bepaald worden. Een voorbeeld van een dergelijke techniek is SLAM (Simultanious Localization and Mapping). Er zijn in het verleden al verschillende onderzoeken gedaan deze manier van lokaliseren [7] [8] [9]. De input van een SLAM algoritme is omgevingsdata, welke verkregen kan worden van verschillende sensoren. Een voorbeeld van een dergelijk sensor is de Kinect sensor van Microsoft. Dit is een zogenaamde low-cost RGB-D camera (Red Green Blue – Depth) die een kleuren camera en een diepte camera heeft. Deze twee beelden kunnen “over elkaar heen gelegd worden” zodat zowel de diepte informatie als de features in het kleurenbeeld gebruikt kunnen worden door het algoritme. Het UAS Lab beschikt reeds over een Microsoft Kinect sensor.

De opdracht omvat het onderzoek doen naar de bruikbaarheid van SLAM in combinatie met een Kinect sensor van Microsoft voor het lokaliseren van een UAV in gebieden waar geen GPS signalen beschikbaar zijn. Hierbij zal gekeken worden naar eerdere onderzoeken naar dit onderwerp en zal onderzocht worden in hoeverre reeds geïmplementeerde algoritmen bruikbaar zijn voor deze specifieke situatie. De verwachting is dat veel van deze algoritmen veel rekenkracht vergen, die niet aanwezig is op een UAS zelf. Er zal gekeken worden of dit het geval is en hoe dit opgelost kan worden door. Hierbij wordt gedacht aan het simplificeren van het algoritme of het distribueren van de rekentaken over verschillende machines.

ROBOT OPERATING SYSTEM (ROS)

Naast het onderzoek naar een lokalisatie methode zal de on-board software voor de Pelican mini-UAS aangepast moeten worden, zodat de deze UASen onder ROS gebruikt kunnen worden in combinatie met reeds ontwikkelde software.

RANDVOORWAARDEN

Binnen dit project zijn de volgende randvoorwaarden opgesteld.

- Er wordt gebruik gemaakt van het framework ROS
- Er wordt gebruik gemaakt van Linux
- Voor het verkrijgen van omgevingsdata wordt de Microsft Kinect sensor gebruikt
- De UASen die gebruikt zullen worden voor dit project zijn de Asctec Pelican en de Octocopter.

PRODUCTEN

Aan het einde van het project zullen de volgende producten opgeleverd worden.

- ROS aansturing voor de Pelican quadrocopter
- Geïmplementeerd SLAM algoritme die gebruik maakt van de Microsoft Kinect sensor
- Documentatie met beschrijving van de werking van de geschreven en gebruikte software
- Scriptie/verslag met beschrijving van het behaalde resultaat en de uitkomsten van het onderzoek

PLANNING

Taak\Week nr	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
Opstarten/Inwerken																				
Plan van aanpak																				
Onderzoek werking Kinect sensor																				
Onderzoek naar bestaande SLAM algoritmen (pros/cons)																				
Bestaande SLAM algoritmen testen op bruikbaarheid																				
Realisatie local estimation (lokale lokalisatie zonder loop closure, cloudstiching, etc.)																				
Realisatie SLAM (loop closure, cloudstiching, drift correction)																				
Combineren local estimation en SLAM																				
ROS aansturing realiseren Pelican																				
Presentatie																				
Afstudeerscriptie																				
(Technische) documentatie																				

BEDRIJFS-/PERSOONSGEGEVENS

PROJECTLEDEN

Naam	Robbert Proost
Email	robbert.proost@student.hu.nl
Student #	1563124

BEGELEIDERS/OPDRACHTGEVER

Naam	Gerald Poppinga
Email	gerald.poppinga@nlr.nl
Telefoon	+31 (0)88-511 3213
Rol	Bedrijfsbegeleider

Naam	Christian Muller
Email	christian.muller@nlr.nl
Telefoon	+31 (0)88-511 3381
Rol	2e Bedrijfsbegeleider

Naam	Leo van Moergestel
Email	leo.vanmoergestel@hu.nl
Rol	(Afstudeer)begeleider

BEDRIJFSGEGEVENS

National Aerospace Laboratory (NLR)
Anthony Fokkerweg 2
1059 CM Amsterdam

www.nlr.nl

BIBLIOGRAFIE

[1] G. ten Buuren, "Small UAS for Law Enforcement," in *2011-2012 UAS Yearbook - UAS: The Global Perspective*, 2011, pp. 134-137.

[2] "AR.Drone 2.0. Parrot new wi-fi quadricopter," Parrot, [Online]. Available: http://ardrone2.parrot.com. [Accessed 02 2013].

[3] "AscTec Pelican," Ascending Technologies, [Online]. Available: http://www.asctec.de/uav-applications/research/products/asctec-pelican/. [Accessed 02 2013].

[4] "OktoKopter XL ARF," [Online]. Available: http://www.mikrokopter.de/ucwiki/ArfOktoXL. [Accessed 03 2013].

[5] [Online]. Available: http://www.ros.org/wiki/.

[6] "Robots/Asctec - ROS Wiki," [Online]. Available: http://www.ros.org/wiki/Robots/AscTec. [Accessed 02 2013].

[7] A. J. Davison, Y. G. Cid and N. Kita, "Real-Time 3D SLAM with Wide-Angle Vision," in *Proc IFAC Symposium on Intelligent Autonomous Vehicles*, 2004.

[8] S. Kohlbrecher, O. von Stryk, J. Meyer and U. Klingauf, "A Flexible and Scalable SLAM System with Full 3D Motion Estimation," in *Proc. IEEE International Symposium on Safety, Security and Rescue Robotics (SSRR)*, 2011.

[9] F. Endres, J. Hess, N. Engelhard, J. Sturm and W. Burgard, "An Evaluation of the RGB-D SLAM System," in *Proc. of the IEEE Int. Conf. on Robotics and Automation (ICRA)*, 2012.