2013•2014
FACULTEIT WETENSCHAPPEN
master in de informatica

Masterproef

Investigating the use of UAVs in combination with low-cost sensors for human-UAV interaction

Promotor :
Prof. dr. Johannes SCHOENING

Ben Jorissen

Proefschrift ingediend tot het behalen van de graad van master in de informatica

2013•2014
FACULTEIT WETENSCHAPPEN
master in de informatica

Masterproef

Investigating the use of UAVs in combination with low-cost sensors for human-UAV interaction

Promotor :
Prof. dr. Johannes SCHOENING

Ben Jorissen
Proefschrift ingediend tot het behalen van de graad van master in de informatica

universiteit hasselt | Maastricht University

Abstract

In recent years, Unmanned Aerial Vehicles (UAV) have turned from military and industrial products to ordinary consumer products. Hobbyists and even children can use UAVs for various reasons, such as scouting for prey, photography or for entertainment. This evolution led us to believe that in the next decades, UAVs will evolve to a point where they can be present in our daily life.

In this thesis, we present a self-contained UAV system for autonomous flight. This system contains a low-cost UAV in combination with a low-cost RGBD camera. This combination allows the UAV to localize itself using a RGBD SLAM approach. Using the localization, the UAV is capable of mapping and navigating in an unknown environment. Due to this, the UAV is able fly autonomously without further user input. The accuracy of the localization is evaluated. Furthermore, we introduce concepts in which human-UAV interaction provides useful.

Summary

This summary is intended for a large Dutch audience.

Motivatie

Door de recente evolutie in de technologie zijn Unmanned Aerial Vehicles (UAVs) beschikbaar voor de particulier tegen een steeds lagere prijzen. De meest gekochte modellen zijn de Parrot AR Drone 2.0 en de DJI Phantom. Tegenwoordig zijn er verschillende onderzoeken verricht die de mogelijkheden van UAVs testen. Enkele voorbeelden hiervan zijn het werpen en vangen van een bal met behulp van UAVs, het bouwen van constructies of zelfs het fungeren als een joggingsgezel. Deze laatste richt zich op de interactie tussen de UAV en de gebruiker (human-UAV interactie), waar de focus van ons onderzoek ligt.

Werking

De evolutie van UAVs laat ons vermoeden dat ze binnen enkele jaren geintegreerd zullen zijn in ons dagelijkse leven. Om de mogelijke interacties met UAVs te onderzoeken, hebben we nood aan een autonoom UAV-systeem met lage kost, dat zich kan lokaliseren in een omgeving en erin kan navigeren.

Om het UAV-systeem inzetbaar te maken voor verschillende soorten omgevingen, is ervoor gekozen om de nodige sensoren met de UAV te combineren. De hardware setup bestaat uit een Microsoft Kinect en de Parrot AR Drone 2.0. Aanvankelijk was de AR Drone niet krachtig genoeg om te vliegen met de Kinect als draaglast. Hierdoor was het noodzakelijk om overbodig gewicht van de Kinect en de AR Drone te strippen.

De volgende stap is de lokalisatie van het UAV-systeem, met behulp van de Kinect. Daartoe moest een map opgebouwd worden, met daarin de kenmerken van de omgeving. Eens deze map met kenmerken is opgebouwd, kan die gebruikt worden om de positie van het UAV-systeem te bepalen.

Eens de positie van het UAV-systeem gekend is, is de navigatie aan de beurt.

Onder navigatie valt het plannen van een pad door de ruimte, alsook het uitvoeren van dit pad. Bij het plannen van een pad is een map van de omgeving nodig. De map opgebouwd voor lokalisatie is hiervoor niet geschikt, daarom moet er een apart model van de omgeving opgebouwd worden. Met behulp van dit model kunnen we een pad plannen en statische obstakels ontwijken. Het geplande pad wordt dan vertaald naar commando's voor de AR Drone.

Concepten

Om het gebruik van ons UAV-systeem te motiveren, hebben we drie voorbeeld scenario's uitgewerkt.

In het eerste scenario assisteert de UAV bij het inspecteren van een gebied dat voor de gebruiker moeilijk te bereiken is. De beelden van de Kinect worden doorgestuurd naar een smartphone. Op deze beelden kan de gebruiker een gebied aanduiden dat geïnspecteerd dient te worden. Via controls op de smartphone kan de gebruiker de positie van de UAV aanpassen om zo het gewenste beeld te bekomen.

Het tweede scenario gebruikt de UAV om bijvoorbeeld vergeten producten in een supermarkt op te halen. Hierbij kan de gebruiker specifiëren welk om product het gaat, waarna de UAV zelfstandig dat product kan halen. Dit laat toe om ondertussen verder te winkelen, wat tijd bespaart.

Het is vaak moeilijk om de weg te vinden in een onbekende omgeving. In het laatste scenario wordt de UAV gebruikt om de gebruiker te begeleiden in een voor de gebruiker onbekend gebouw. Bij het binnenkomen kan de UAV de gebruiker begroeten en vragen wie hij wil bezoeken. Nadat deze persoon gespecifieerd is, kan de UAV de gebruiker begeleiden naar het betreffende kantoor.

We merken op dat niet alle scenario's haalbaar zijn met de huidige stand van de technologie.

Evaluatie

In de ontwikkeling van het systeem zijn er twee verschillende lokalisatie algoritmen geïntegreerd. Deze algoritmen zijn beide geëvalueerd op basis van nauwkeurigheid. De

test setup bestond uit 144 gemeten punten waaruit de positie berekend moest worden. Het eerste algoritme slaagde erin om voor 119 punten (82.64%) een positie te bepalen, maar deze positie bevatte een gemiddelde fout van 1.42 meter ten opzichte van de echte locatie. Op basis van de standaardafwijking konden we bepalen dat 99.7% van de geschatte posities binnen een foutmarge van 4.81 meter lagen.

Het tweede algoritme kon slecht voor 72 (50%) punten een positie berekenen. De nauwkeurigheid was dan weer hoger, met een gemiddelde afwijking van 0.93 meter. Ook de foutmarge is lager, 99,7% van de geschatte posities ligt binnen een foutmarge van 2.96 meter.

Conclusie

Het ontwikkelen van een UAV-systeem dat autonoom kan navigeren, gebruik makend van de on-board sensoren is niet mogelijk zonder problemen. Bovendien waren er vrij ingrijpende aanpassingen nodig aan de hardware setup om het UAV-systeem stabiel te laten vliegen.

De software architectuur laat toe om verschillende algoritmen en frameworks voor lokalisatie en navigatie uit te wisselen. Op dit moment laat de nauwkeurigheid van de lokalisatie niet toe om de voorgestelde scenario's te implementeren zonder gevaar voor de gebruiker. Om dit op te lossen stellen we verschillende technologieën voor om dit te verbeteren.

Acknowledgments

I would like to thank prof. dr. Johannes Schöning for his guidance and inspiration during this thesis. As well as my supervisor, Raf Ramakers, for his indulgence during the testing phases and providing advice for writing. A special thanks to Tom De Weyer for providing assistance with the realization of the hardware setup.

And finally, my appreciation to my friends and family for the moral support.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Unmanned Aerial Vehicles (UAVs) have moved from the military and industrial domain to the consumer market. This was accelerated by the advancements in technology, which have led to a price drop. Today, hobbyists and even children are able to access and play with different kinds of UAVs. Figure 1.1 shows three noticeable milestones in the evolution of UAVs from military use to the consumer market. In figure 1.1a, the U.S. global hawk is shown, which is used for surveillance, reconnaissance and warfare. It first took flight in 1998 and its size is similar to that of a commercial flight airplane. In contrast to large scale military UAVs, the Parrot AR Drone 2.0 is shown in figure 1.1b. The Parrot AR Drone 2.0 is one of the bestselling consumer UAVs since it was released in 2010. It is controlled using a smartphone or tablet over Wifi. Figure 1.1b is another step in the evolution, showing the PowerUp 3.0 that is currently under development. The PowerUp 3.0 is a add-on for self-made paper gliders and transforms them into a flying paper plane that can be controlled using a smartphone or tablet via Bluetooth.

Companies like Amazon and DHL show interest in using UAVs as a delivery system, as covered in recent articles and blog posts. The Amazon delivery system is shown in figure 1.2b. These rapid developments in technology opened up a large design space for researchers as well to experiment with UAVs for different purposes. For example, D'Andrea et al. [17] developed a dedicated room to experiment with UAVs, called the Flying Machine Arena (FMA). It is used in combination with multiple

Figure 1.1: Different types of UAVs, showing the U.S. Global hawk (a), Parrot AR Drone 2.0 (b), PowerUp 3.0 (c).

small scale UAVs to solve tasks in a collaborative way, for example ball throwing or construction as shown in figure 1.2a and 1.2c respectively. In addition, Human-Computer interaction (HCI) researchers have explored the use of UAVs to interact with users. An example of human-UAV interaction is Joggobot by Meuller et al. [20], as shown in figure 1.2d. They investigate the use of a UAV as a jogging companion.

In line with recent research, we work towards the use of UAVs for human-UAV interaction. For this, a low-cost sensor, the Kinect, is used in combination with a commercial UAV, the Parrot AR Drone 2.0, to enable human-UAV interaction. By connecting the Kinect to the Parrot AR Drone 2.0, it is able to locate itself in an environment and to navigate through that environment autonomously. This UAV system was developed with the prospect of human-UAV interaction. To illustrate this interaction, a short scenario is scripted.

1.1 Scenario

Imagine having to go to a meeting at an unknown location. The user enters the building where the meeting takes place, then a UAV flies up to be his personal receptionist. The UAV asks the user which office he wants to go to and subsequently processes the input from the user. A path is then planned to the desired office and the UAV flies ahead of the user providing guidance, as shown in figure 1.3a. Once they arrive at the office, the UAV can display information about the office's occupancy, as seen in figure 1.3b. After the meeting, it is possible that the user has forgotten his keys, the

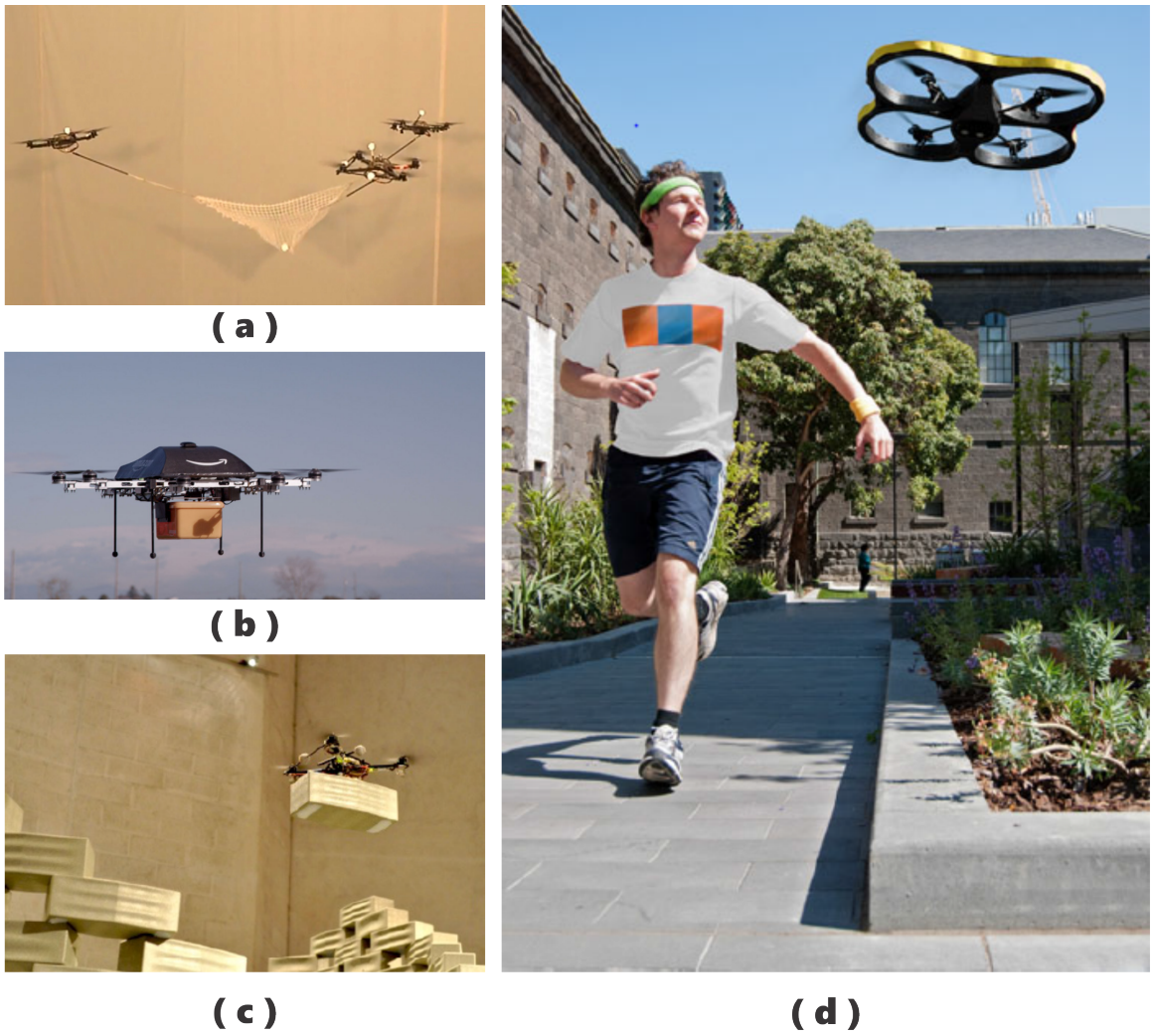

Figure 1.2: Several applications using UAVs.

UAV can retrieve the keys for the user, as shown in figure 1.3c.

To realize such a scenario, it is necessary to provide solutions for fundamental problems discussed in the next section.

1.2 Problems

In this thesis, we focus on the development of a self-contained UAV system that is able to navigate autonomously. This system is developed as a test bed for human-UAV interaction. To realize this system, localization and navigation have to be addressed.

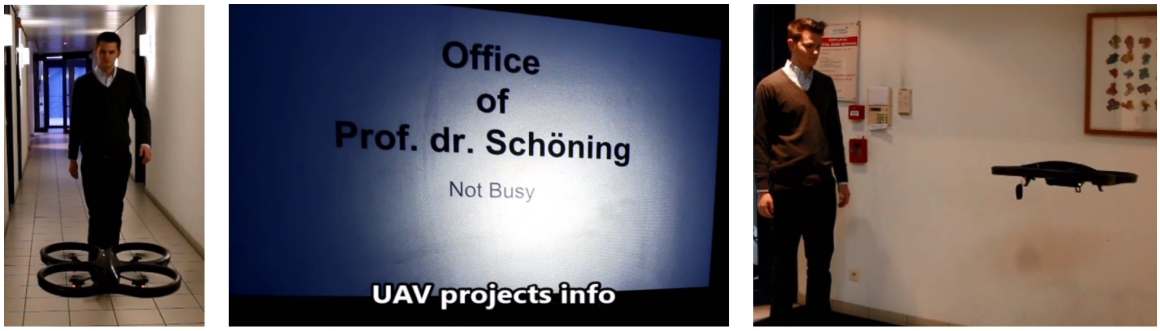

Figure 1.3: Example images from the meeting scenario: a UAV guiding user to the correct office (a), a UAV projecting information about the office (b), lost keys being retrieved by the UAV (c).

Localization In order to fly around autonomously, the location of the UAV needs to be known. There are a lot of existing solutions for example, marker tracking. These existing technologies and methods will be discussed in detail in section 2.2.1.

Navigation The UAV needs to independently navigate from location A to B, while avoiding obstacles. Section 2.2.2 elaborates on existing navigation techniques.

Interaction How the UAV should output information to the user, but also how it should interpret input from the user in a meaningful way. We elaborate on previous efforts of UAV interaction in section 2.3.2.

1.3 Contribution

This thesis makes a contribution in the following points:

1. A Parrot AR Drone 2.0 and Microsoft Kinect were modified to create a low-cost self-contained UAV system, capable of flight.

2. Different software packages were combined and tested, so the UAV system can localize itself and map the environment.

3. A navigation package was developed and tested in simulation.

1.4 Structure

Chapter two provides an overview of related work. This includes a discussion of different UAV models, localization systems, existing navigation solutions and several UAV-based systems that allow human-UAV interaction.

This is followed by chapter three that describes the concept of the implementation, what technologies are used and provides example scenarios in which a UAV system could provide assistance.

In the fourth chapter, the implementation is described more in detail. The specifics of path planning are discussed, and what techniques are used to store the environment.

In chapter five, we report on the evaluation of the localization techniques used.

Chapter six reflects on the work done and presents suggestions for future work, problems and open issues.

Chapter 2

Related work

This chapter describes related work. First, different existing UAVs and their properties are discussed. This is followed by a discussion on existing solutions for localization and navigation mostly used in robotics. Finally, a summary of related work in the area of human-UAV interaction is given, followed by a discussion about the differences from our approach.

2.1 Unmanned Aerial Vehicles

The evolution of UAVs has brought them to the consumer market. In this section, a comparison is given between the most commonly used UAVs and their properties.

The models listed in table 2.1 have different properties, making them suitable in different contexts. These contexts include military and warfare, research, human-UAV interaction, photography, remote sensing, entertainment, etc. UAVs are classified in two main designs, rotorcraft and fixed-wing aircraft. The discussion that follows is limited to rotorcraft designs, since fixed-wing aircraft designs are not capable of hovering. An example of fixed-wing aircraft design is the U.S. Global Hawk, mentioned in chapter 1.

Within rotorcraft design, there are many different setup possibilities depending on the number of rotors. Helicopter type UAVs have one or two rotors placed above each other. This is the most basic type of UAV. They have the disadvantage of low agility

and low resistance to outside forces. Rotorcraft designs with three rotors are called tricopters. In comparison to helicopter types, it provides a higher payload and agility. Commercial tricopters are not commonly sold, but are mostly used for homemade UAVs, because they are easier to build and program. The next type is a quadcopter, this design uses four rotors distributed symmetrically over the body. It allows for rapid changes in motion as with the tricopter, but the performed motion is smoother. The hexacopter and octocopter uses the same principles as the quadcopter, but with six and eight rotors respectively. The main advantage is the extra lift generated by the additional rotors, the octocopter more so than the hexacopter.

As the Parrot AR Drone 2.0 is used in chapter 4, it is discussed in detail in the next section.

2.1.1 Parrot AR Drone 2.0

The Parrot AR Drone 2.0 is a powerful, yet low-cost quadcopter. In this section, the flight dynamics and flight control of the AR Drone are discussed. Then, arguments are given to motivate why the AR Drone is chosen.

2.1.1.1 Flight dynamics

The AR Drone is a quadcopter, thus uses four rotors. By spinning the rotors at equal speed, the quadcopter generates upward force, called lift, allowing it to hover above the ground. In order to move, the AR Drone controls its pitch or roll by adjusting the speed of adjacent rotor pairs, as in figure 2.1a and b. To turn the quadcopter over the yaw-axis, it differentiates the speed of opposite pairs of rotors, as shown in figure 2.1c.

However, rotors are never completely in sync. This means that even when the quadcopter is hovering, it will always drift away from its position. To compensate for this, the AR Drone uses an on-board auto-pilot for stability.

Image	Model	Type	Indoor	Outdoor	Size	Payload	Autonomy	Control	Sensors	Cost
	AiBot-x6	Hexacopter	●	●	103 cm	2 kg	30 min	Radio-controller	Altimeter, gyroscope, magnetometer, GPS, universal mount	$$$
	AscTec Pelican	Quadcopter	●	●	65.1 cm	650 gr	16 min	Wifi commands	Configurable	$$$
	CrazyFlie Nano	Quadcopter	●		9 cm	10 gr	7 min	USB antenna	Altimeter, gyroscope, magnetometer	$
	Delfly Explorer	Flapping-wing	●		28 cm	0 gr	9 min	Radio-controller	Barometer, gyroscope, stereo camera	Unknown
	DJI Phantom	Quadcopter	●	●	35 cm	500 gr	12 min	Radio-controller	Gyroscope, GPS, universal mount	$$
	Parrot AR Drone 2.0	Quadcopter	●	●	72 cm	250 gr	15-18 min	Smartphone or tablet via Wifi	Gyroscope, accelerometer, magnetometer, pressure sensor, front camera, downward facing camera	$
	Parrot Bebop	Quadcopter	●	●	32 cm	Unknown	Unknown	Wifi controller	Gyroscope, accelerometer, magnetometer, pressure sensor, front camera, optical flow camera	Unknown
	Skybotix's CoaX	Helicopter	●	●	34 cm	Unknown	+20 min	Bluetooth, Wifi commands	IMU, pressure sensor, downward facing sonar, universal mount	$$$
	Turbo Ace infininity 9	Octocopter	●	●	169 cm	6.8 kg	12-20 min	Radio-controller	Gyroscope, GPS, universal mount	$$$

Table 2.1: Different types of UAVs and their properties.

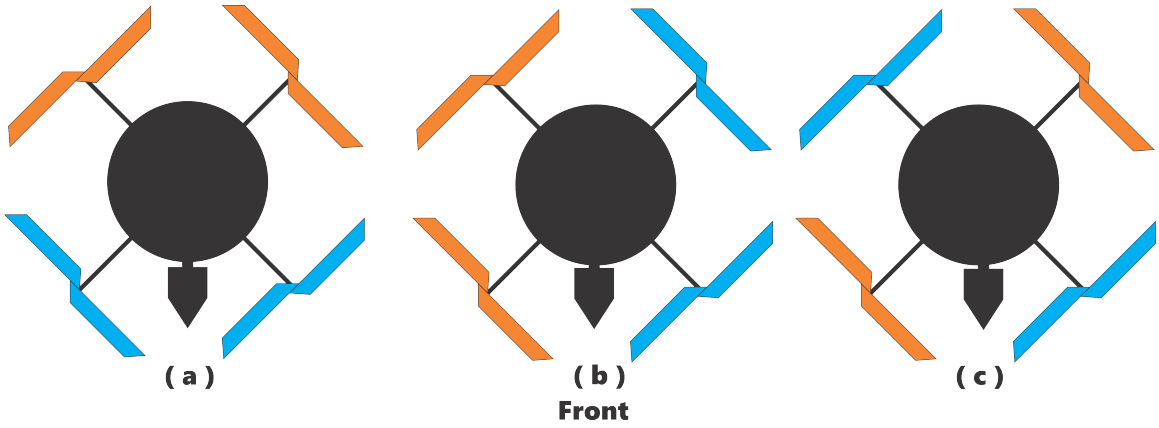

Figure 2.1: Different rotor combinations for a quadcopter movement. The blue rotors are slowed down for forward pitch, the orange rotors are slowed down for backward pitch (a). The blue rotors are slowed down for left roll, the orange rotors are slowed down for right roll (b). The blue pair is slowed down to rotate clockwise, the orange pair is slowed down to rotate counterclockwise (c).

2.1.1.2 Stability

The unique property of the AR Drone is that it can hover in place with high accuracy using its auto-pilot. This is made possible by analyzing and reacting to information from four sensors.

The first sensor the AR Drone uses is the accelerometer. This allows the auto-pilot to detect any pitch, roll or yaw movements the AR Drone makes. When it detects an external force like wind, it will analyze the data from the accelerometer and apply force in the opposite direction to maintain its position.

In order to correctly maintain height, the ultrasonic sensor is used. This sensor gives the auto-pilot updates about the height of flight so it can be maintained. This only works when the AR Drone is no higher than 3 meters, since the ultrasonic sensor can only detect distances up to 3 meters. Above this height, the AR Drone starts using the barometric pressure sensor, which is less accurate.

The final sensor used is the downward facing camera. It uses consecutive camera images to estimate drift with respect to the ground surface. The AR Drone compensates for the drift by flying small distances in the opposite direction.

The synergy of these separate stabilization methods allows the AR Drone to hover in place without external control being necessary.

The Parrot AR Drone 2.0 was chosen for its stability, low cost, available sensors (front and downward facing RGB-camera, accelerometer, ultrasound sensor), the high payload capacity in its price range and the open control interface through Wifi.

2.2 Localization and navigation

In this section, existing hardware and technologies are discussed that can be used for localization. Afterwards, navigation systems and properties a UAV navigation system should provide are described.

2.2.1 Localization

An in-depth discussion of localization hardware, that can be used for indoor tracking, is given. The focus is on indoor tracking, since the main goal is to test human-UAV interaction in an indoor environment. Thereafter, the technologies in which the hardware is incorporated, like KinectFusion and SLAM, are summarized. For an overview of localization technologies, including those that do not restrain to indoor localization, such as GPS, refer to Hightower et al. [8].

2.2.1.1 Localization hardware

Laser range scanner A laser range scanner, also known as Light Detection and Ranging (LiDAR), works by sending a laser beam to a target and analyzes the returned light to estimate the distance. Doing this over a horizontal or vertical range gives us a laser scan.

RGB-camera An RGB-camera combines three components (red, green and blue) to form a colored image.

Depth cameras Depth cameras return an image in which each pixel contains information about its distance to the sensor. Depth can be registered by an IR emitter and sensor like in the Microsoft Kinect. The other approach is using a stereo camera

system like with the Delfly Explorer [13]. With a stereo camera system, the images from the two cameras are analyzed for features and compared to estimate depth. Often a depth camera are combined with an RGB-camera. This combination is called an RGBD-camera.

Infra-red camera Infra-red (IR) cameras are equipped with an IR-pass filter. This allows the camera to only capture incoming IR-light.

2.2.1.2 Localization technologies

Feature tracking Feature tracking works by detecting markers or features from a camera image. Markers are designed to be easily recognized by one or more cameras. There are many types of markers, ranging from colors to geometric patterns. For RGB-cameras these can be colors, a printed pattern or even just recognizable features in the scene. IR-cameras on the other hand use markers that reflect IR-light and stand out from the rest of the scene. Geometric markers can be added to a scene that is observed by a depth-camera. A geometric marker is an object with unique dimensions, which stands out in a scene. When using markers there are two types of tracking that can be employed, absolute tracking and relative tracking. For absolute tracking, multiple cameras are placed in the room and track the position of the marker. Relative tracking places multiple markers around the room and the position of the camera can be determined relative to the markers. These systems do not scale well, since tracking a larger area requires more cameras or more markers. With absolute tracking, every change in camera setup requires a re-calibration.

In contrast to markers, there are also distinct features that can occur naturally in an environment. The general idea is that images are used to extract key-features, done by feature detectors like Speeded Up Robust Features (SURF) [1], Scale Invariant Feature Transform (SIFT) [16] or Oriented BRIEF(ORB) [28]. By comparing features from two consecutive frames, it is possible to find overlap and estimate the motion of the camera.

SLAM SLAM is the problem of tracking the position of a sensor while building a map of the environment. Different algorithms have been developed to solve this problem, a basic outline is provided to explain the working of SLAM algorithms. Most SLAM algorithms use consecutive frames of sensor data to estimate the motion of the sensor. This estimation can contain small errors. Over longer trajectories, these errors can build up and cause drift. SLAM algorithms try and bound this drift. They do this by saving a state, containing the sensor pose together with data about its received data, in a map. Every new state is compared against the map, when the state is not present in the map, it is added. But when the state is found in the map a loop is detected. The algorithm can then reduce the drift from the motion estimation using the reference state in the map. Three specific SLAM algorithms are now discussed in detail.

Canonical Scan Matcher (CSM) matches two scans from a laser range scanner. CSM is a fast variation of Iterative Closest point (ICP), using point-to-line metric by Censi et al. [2]. It has the advantage of being very fast, but has problems with large rotations between frames. It works by using the first laser scan as a starting point, then each consecutive scan is compared to previously obtained scans. CSM then calculates the laser range scanner's pose by calculating the transformation between the current laser scan and the previous frames.

Next is grid mapping, which uses a Rao- Blackwellized particle filter to estimate motion and build a map. Each particle in the filter contains an individual map of the environment. The movement of the sensor as well as the differences in sensor data are used to reduce the number of particles in the filter. When the particles are reduced, the uncertainty about the sensors position is reduced as well.

The last SLAM algorithm is RGBD-SLAM. It employs the tracking of key-features in an RGB-image to estimate the camera motion. Combining key-features and depth information enables the mapping of the obtained key-features in 3D space. When the camera revisits an area, a pose graph is built, which represents the transformation between the camera and previously detected features. This allows new features to be linked to older features and loop-closure can be done. When features are linked, the

algorithm has a known reference point for its current location which reduces the drift from the motion estimation.

KinectFusion KinectFusion by Newcombe et al. [22], extended by Whelan et al. [30] as Kintinuous, does surface reconstruction by utilizing the Microsoft Kinect. In order to do this, it solely uses depth information generated by the infrared sensor, allowing it to work in dark rooms, since it does not rely on visual features. The result is a more aesthetically surface reconstruction of the scene and a 6 DOF camera pose.

The first step of the algorithm is filtering the raw depth information received from the RGBD-camera with a bilateral filter. This filters out possible errors by smoothing the information. The filtered information is used to calculate a vertex- and normal map. Using ICP, the vertex and normal map are compared to a model constructed of previously processed frames, by calculating the camera transformation. The unfiltered depth information is added to the model according to the found camera transformation, using a Truncated Surface Distance Function (TSDF). The TSDF of the previously obtained model and the current TSDF are merged by using a weighted average. To generate a 3D surface model, a ray-casting algorithm is performed on the TSDF. This is looped at 30 frames-per-second (FPS) to generate real-time models.

6D monte carlo localization This method relies on a previously obtained map in an octree-structured manner [33]. The map can be obtained using RGBD-SLAM or KinectFusion. To determine and track the UAV's location, laser scans are put through a Monte Carlo simulation. It localizes in 3D space, but the main difference with other localization methods is that it does not generate a map by itself.

2.2.2 Navigation

After the location in the environment is known, navigation is the next step. Currently, the three navigation techniques discussed are often used in robotics.

2.2.2.1 MoveIT!

MoveIT! [3] is an open-source motion planning library, originating from the arm navigation and grasping pipeline. It contains several sampling-based motion planning algorithms like Probabilistic RoadMaps (PRM) and Rapidly-exploring Random Trees (RRT). The main purpose of MoveIT! is to plan a collision free path for robotic arms. In MoveIT! a robot is modeled as joints and links. Using MoveIT! with a UAV requires defining the UAV as a joint that can move freely.

2.2.2.2 4 DOF path planner

6 DOF path planners do not incorporate the flight dynamics of UAV flight in the path calculation. Since a UAV uses its pitch and roll to move from front to back or sideways, pitch and roll cannot be performed whilst staying in the same location. For this problem, Valenti et al. [29] argue the use of a 4 DOF path planner for UAV navigation. This reduces the complexity of path planning by 2 DOF and allows to the planned path to be directly executed by the UAV.

2.2.2.3 Polynomial trajectory planning

Richter et al. [26] developed a method for optimizing paths obtained from existing path planners for UAVs. To use the planned path, it needs to be defined as waypoints. To optimize the path, they connect the waypoints using a straight line. This straight line trajectory would be highly inefficient for UAVs to follow, because UAVs fly in a continuous motion and the straight line path is discontinuous. They do this by joining together the waypoints in a smooth minimum-snap trajectory, ensuring a continuous path. It allows the UAV to navigate a path without having to stop and turn.

2.3 UAV-based systems

In recent years, researchers have explored several usages for UAVs. In this section, different approaches are discussed and compared to our approach. Table 2.2 contains

an overview of the different approaches. For this discussion, the systems are split up into categories that are focused on UAV flight dynamics and those that focus on human-UAV interaction.

2.3.1 UAV flight dynamics

Systems that are purely concerned with the flight paths or with the localization of a UAV are described below.

Flying Machine Arena In the FMA, extensive research in the area of collaboration and flight control is done. All of the stated research done in the FMA use an external marker-based tracking system using IR-cameras and reflective markers.

Ritz et al. [27] developed a system that throws and catches a reflective ball, trackable by IR-cameras, by using three quadcopters. A net is carried by the quadcopters, by flying outward the quadcopters create tension on the net which launches the ball. Tilting the formation changes the direction in which the ball is thrown. The system works in real-time without pre-planning and uses an iterative learning algorithm that learns from experience, as discussed by Purwin et al. [25].

Willmann et al. [31] introduced a new approach towards architectural construction. A swarm of quadcopters work in parallel. They can lift, transport and interact with building blocks in order to erect a building.

Construction with quadcopters Lindsey et al. [15] developed an approach for UAV-enabled construction as well. Lightweight beams with magnetic connectors were used to erect tower-like structures. In this approach they investigated grasping, transport and assembly capabilities of UAVs. Tracking is again done by IR-cameras and reflective markers.

Musical drones Powers et al. [24] tested the influence of nearby objects on the aerodynamics of the quadcopter. This is how quadcopters react when the airflow from the rotors is bounced back by nearby objects. Their setup consisted of the same

tracking as FMA, a marker tracking system using IR-cameras and reflective markers. The test case consisted of quadcopters flying around and playing musical instruments.

CityFlyer Cityflyer is an Open-Source Navigation System for UAVs created by Morris et al. [19, 4, 18]. A UAV is equipped with a laser range scanner. The laser range scanner, with addition an Inertial Measurement Unit (IMU), are used for SLAM using CSM, explained in section 2.2.1. The position estimation from SLAM and data from the inertial measurement unit are fused in an extended Kalman filter to smooth the results. CityFlyer was extended by Valenti et al. [29] to use a RGBD-camera and RGBD-SLAM instead of a laser range scanner, but this had stability issues. They also suggest using a 4DOF path planner, since the UAVs pitch and roll are determined by its speed of movement, as explained in section 2.1.1.1.

2.3.2 Human-UAV interaction

In this category, the related work investigates the possibilities of human-UAV interaction. This means developing a different means of control or adding new elements to existing activities, such as jogging, soccer, driving, etc. in order to improve or enrich it.

Flying Head In the Rekimoto lab, Higuchi et al. [10] developed and tested a technique called Flying Head. This technique uses external tracking with IR-cameras and reflective markers. The main goal is to let a user control the UAV in a more intuitive way. Human motions like walking, crouching and looking around are used to control the UAV.

Mind over Machine LaFleur et al. [12] utilize brain-computer interfaces (BCI) to control a UAV. The controls work by coupling brain activity of physical movements to a command for the UAV. The user can then control the UAV simply by using their thoughts.

Flying Eyes Realized by Higuchi et al. [9], Flying Eyes follows the user and acts like a personal camera man, recording their actions. It uses a relative marker based tracking system, with a colored shirt as marker and an RGB-camera. A user interface is available to suggest the type of camera work. The system can also be used in sports training, so that the user can see and analyze their movements in real-time [11].

Joggobot Developed by Mueller et al. [20] to investigate how robots can assist people in exertion activities. Joggobot uses relative marker tracking with an RGB-camera to accompany the user. They found that users feel as if they are getting chased when the UAV flies behind them, that is why the drone flies in front of the user. With this system they hope to encourage the runner to run further, more often or even faster.

FollowMe Naseer et al. [21] developed FollowMe to autonomously follow a human and interpret hand gestures. This is done by using a RGBD camera. The depth information is used to track the human body and to do gesture recognition. The system allows the user to take pictures and land the UAV, by raising their right and left hand respectively.

Hoverball Nitta et al. [23] created a UAV surrounded by a spherical structure, called Hoverball. This allows a change in the physical dynamics of a ball, it can be slowed down, stay in the air, manipulated remotely, etc. This can be useful for people for whom the ball would move too fast, or are unable to pick up the ball from the ground, or even to completely change the dynamics of a sport.

Kwid flying companion[1] A UAV, called flying companion, is placed inside the roof of Renault's concept car called Kwid, when the user requests it, the UAV can fly off. It is controlled through an integrated tablet in the car's interior. The UAV can be used to scout for traffic, take pictures of the surroundings or even detect obstacles further up the road. The goal of the flying companion is to provide a more pleasant

[1] www.renault.com/en/innovation/l-univers-du-design/pages/kwid-concept-car.aspx

drive, without being confronted with unpleasant surprises like road-blocks or traffic jams.

2.4 Comparison to related work

In this chapter, different hardware sensors and technologies to allow localization were presented. Next, the focus is on standard navigation techniques that can be employed to plan UAV flight. Finally, there was a discussion of existing systems that focus on improving UAV flight dynamics or on human-UAV interaction. In figure 2.2, the discussed systems are positioned based on sensor cost and the type of tracking. The category of the system is represented by the color. Our focus is on systems in the categories of flight dynamics and user experience. Most of the systems that focus on flight dynamics use external tracking with expensive sensors. Our approach differentiates itself by developing a self-contained system that uses low-cost sensors. FollowMe is the only self-contained system that does tracking by using low-cost sensors. In contrast to our approach, FollowMe depends on the user's position in contrast to our system. In our approach, the system is environment aware and can navigate without having to track the user. By making the system independent of the user, extra freedom is allowed for the UAV, enabling more possibilities for interaction.

Name	Postioning	Indoor	Outdoor	SC	Control	Hardware	Method
CityFlyer [19]	Absolute	●	●	●	Autonomous	Laser scanner	SLAM
Construction with quadcopters [15]	Absolute	●			Autonomous	IR-camera and markers	Optical marker tracking
Flying eyes [9]	Relative	●	●	●	Autonomous	RGB-camera	Body tracking
Flying head [10]	Absolute	●	●		Human motion	IR-camera and markers	Optical marker tracking
Flying Machine Arena [17]	Absolute	●			Autonomous	IR-camera and markers	Optical marker tracking
FollowMe [21]	Relative	●	●	●	Gestures	RGBD-camera	Gesture recognition
Hoverball [23]	None	●	●	●	Physical interaction	Spherical frame	None
Joggobot [20]	Relative	●	●		Autonomous	RGB-camera and marker	Optical marker tracking
Kwid - flying companion	None		●	●	Tablet via Wifi	RGB-camera	Direct control
Mind over mechanics [12]	Absolute	●	●		Brain-computer interface	EEG-cap	Monitoring brain activity
Musical drones [24]	Absolute	●			Autonomous	IR-camera and markers	Optical marker tracking

Table 2.2: Summary of existing UAV-based systems. Self-contained (SC) means that the systems can be deployed without any adjustments to the environment.

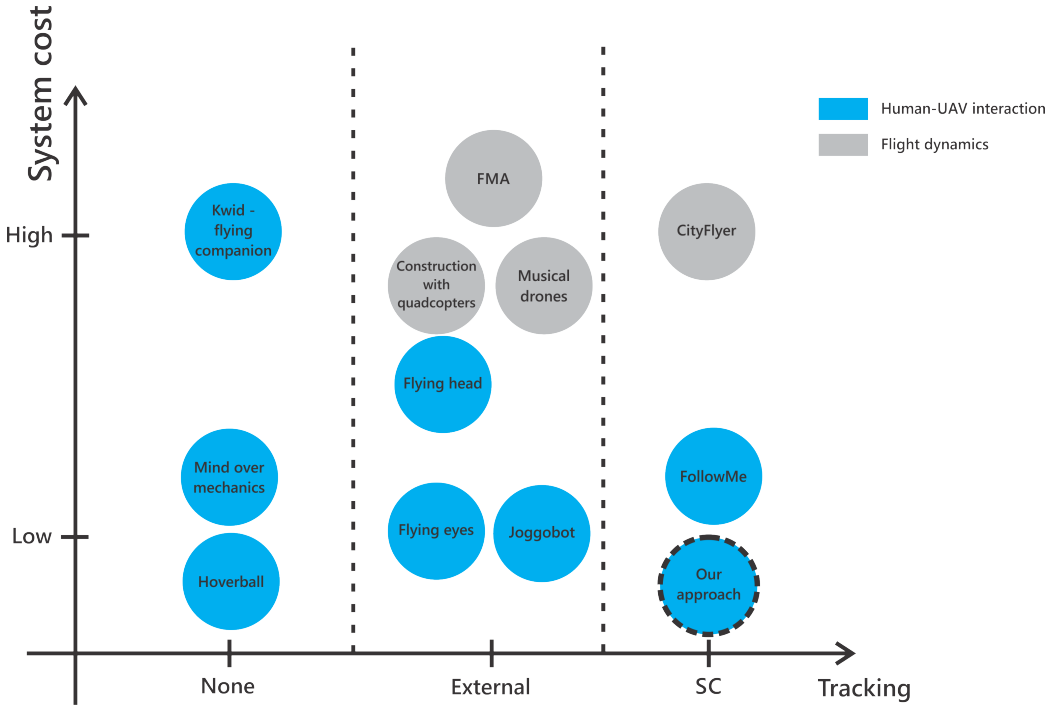

Figure 2.2: Graph situating related systems in comparison to our own approach. X-axis shows the type of tracking system and the y-axis depicts the cost of the system in a continuous space.

Chapter 3

Concept

In the previous chapter, existing technologies for localization and navigation were discussed extensively. In this chapter, we present and explain our self-contained UAV system, which is capable of navigation autonomously in an indoor environment. We start with a discussion of a hardware setup and software architecture. This hardware setup demonstrates how to combine a UAV and RGBD camera to obtain a self-contained UAV, with the prospect of self-localization, mapping and navigation. On top of that, a software architecture is used that enables a UAV to localize, map and navigate in an indoor environment. Afterwards, we demonstrate the potential of our system, by showing how it can be deployed in real world scenarios.

3.1 Hardware setup

In this section, we discuss the hardware setup that is needed to establish a self-contained UAV system, capable of navigating autonomously. To realize this, the Parrot AR Drone 2.0 and Kinect are used as low-cost hardware. To allow localization, mapping and navigation using only on-board sensors, the Kinect needs to be connected to the AR Drone, without restricting the AR Drone's flight abilities.

3.1.1 Weight-reduction

Simply attaching the Kinect to the AR Drone does not work, since the Kinect is to heavy for the AR Drone to carry. To overcome this weight issue, excessive hardware from the Kinect and the AR Drone were removed. From the Kinect, the following parts were removed[1]:

- All plastic casing.
- Motorized base and gears.
- Cooling fan.
- Microphone array and connector.

Next, the weight of the AR Drone is reduced by removing the battery from the hull. The cable of the battery is extended, so the battery can be placed outside of the UAV (on the ground, table, etc.). Finally, the Kinect and AR Drone have to be combined. The simplest solution is to hang it from the bottom, but this would interfere with the ultrasonic sensor. Blocking this sensor gives the AR Drone the illusion it is on the ground and makes it crash into the ceiling. To avoid interference with the AR Drone's sensors, the Kinect is attached on top of the hull of the AR Drone. To ensure stable flight, we measured out the balance point of the Kinect and AR Drone to locate a suitable mounting position for the Kinect. Attaching the cables from the Kinect and battery to hull ensures they do not interfere with the rotors or downward facing sensors. With this setup, the AR Drone only has a flight time of a couple of seconds. This is due to the added resistance of the cable, which extends the battery. To extend the flight time, an external power supply was attached, replacing the battery and providing constant power. The final setup of UAV and sensors is shown in figure 3.1, and we continue with an explanation of the software architecture.

[1]Instructions on disassembling the Kinect can be found on IFIXIT: http://www.ifixit.com/Teardown/Microsoft+Kinect+Teardown/4066

Figure 3.1: Stripped Kinect mounted on a Parrot AR Drone 2.0, capable of flight.

3.2 Software architecture

In this section, an explanation of the different phases in the system is given. An overview is shown in figure 3.2. For developing a system that is able to navigate independently from the user, the system is divided into two phases. In the first phase, the system learns about the environment and in the second phase, the system is able to plan paths and navigate across those paths.

3.2.1 Learning phase

The learning phase consists of three components, the RGBD sensor and UAV, RGBD SLAM and environment model. The first component is the combination of an RGBD sensor and UAV as a self-contained UAV system. The RGB and depth information of the RGBD sensor are communicated to the RGBD SLAM component. This component provides localization by building up training data and estimating the position of the UAV. RGBD SLAM works by extracting features from the RGB information. These features are mapped onto the depth-information to obtain a 3D point of the feature. The 3D points are matched in consecutive frames to estimate the transformation between the frames. From this transformation, the sensor pose is computed

and saved together with the features, resulting in a sparse feature map of the environment. This sparse map is saved as training data, but is not suitable for path planning and navigation. To overcome this problem, an environment model is build, which is a representation of the map, usable for path planning and navigation. Building the environment model is started when the localization stabilizes. The moment of stabilization is dependent of the incorporated RGBD SLAM algorithm.

Once the feature map and environment model are built, the learning phase is over and the operation phase is started.

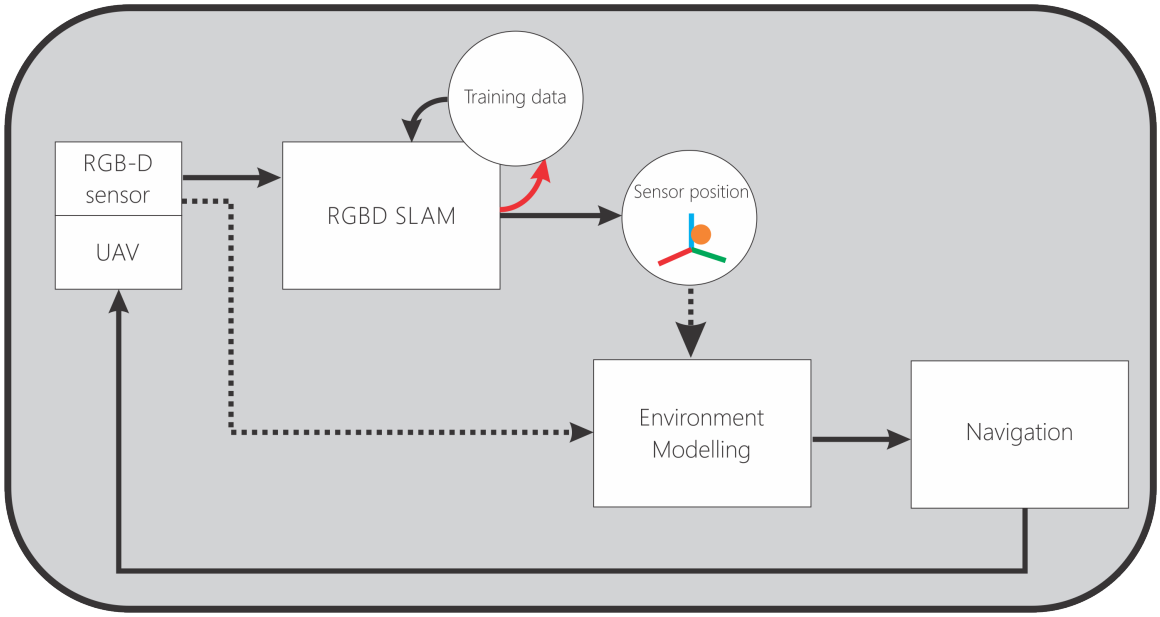

Figure 3.2: A conceptual representation of the different components. Dashed lines mean this connection is instantiated as soon as the RGBD SLAM stabilizes. Red lines mean this connection is broken after the learning phase.

3.2.2 Operation phase

In the operation phase, the newly received RGB and depth information are not added to the training data, but are merely used to estimate the position by comparing them to the training data. Next, the navigation component uses the estimated position and environment model to plan a collision-free path between two positions. This path is then communicated and translated into commands understandable by the UAV.

Now that the UAV is able to navigate rooms autonomously, example scenarios are provided in which the UAV can be deployed.

3.3 UAV interactions

In contrast to other robotic vehicles (humanoids, autonomous cars, etc.), UAVs are not restricted to a ground surface. This allows them to be deployed in wider range of scenarios. The following three examples motivate the usage of a self-contained UAV system for human-UAV interaction.

In these examples, the assumption is made that the training data has already been acquired. Furthermore, we mention that not all of these scenarios are technically achievable at this moment, but will be in the future as the technology advances.

3.3.1 Distant inspection

People often need to reach higher located places for close-up inspection. For example, to analyze structural integrity of a building or to confirm the location of an object. Most of the time this requires getting a ladder or lift to reach that area. Another approach is to manually fly the UAV for close-up inspection, but this requires training and experience that not everyone possesses. This is why we propose this concept and let an autonomous UAV handle the task.

For the UAV to know what area to inspect it must be defined by the user. Figure 3.3 shows a mobile application that can communicate this information to the UAV. The image seen on the screen is a video-stream from the UAV. It is perfectly possible that the area of interest is not in the field-of-view of the camera. For that reason, buttons at the side of the screen are provided to adjust the position of the camera. The user can highlight an area by touch. An area around the touch is used to extract features. The system then locates these features in the training data to translate the 2D features into 3D coordinates. Knowing the 3D coordinates, the system plans a path to that location and the user is able to inspect that area through the video-feed. The camera output could be replaced by other sensors. For example,

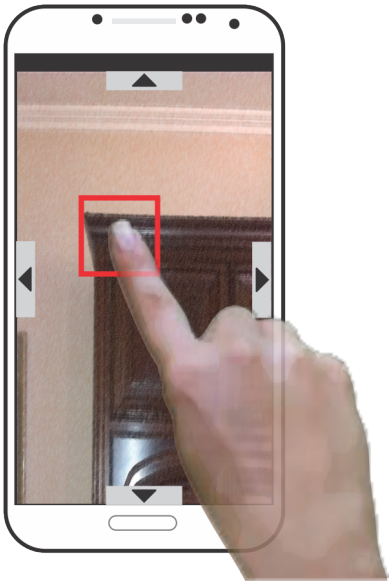

Figure 3.3: Mobile application concept to indicate what area the UAV should inspect.

a thermal imaging sensor to investigate the isolation of a building.

3.3.2 Item retriever

During grocery shopping, it is possible to forget a product that was on the shopping list. If this item is on the other side of the store it is necessary to walk all the way back to retrieve the product. Commanding a UAV to retrieve this product can save time.

To enable the UAV to retrieve the item it needs to know which item and where to find it. Specifying the item could be done using a shopping list app, that is able to communicate the unique product identifier, which is often the bar-code, to the UAV. Since grocery stores often do not contain a database linking products to a location, another approach is necessary. Using a bar-code scanner, the UAV can fly through the store in search of the product. Once the product is found, it can be delivered to the user. The time that would have been necessary to return to the forgotten item is then saved and can be used to finish the remainder of the shopping list.

3.3.3 Personal receptionist

Finding a place in a large unknown building can be difficult. A common method is to ask the receptionist for directions. These directions can be confusing or can be interpreted in a wrong way, leaving the visitor lost. Even worse, the receptionist cannot be found, because he or she is occupied somewhere else.

A UAV in the form of a personal receptionist can overcome these issues. When someone enters the building, the UAV is activated and flies towards the visitor. After being asked where the visitor needs to go, the UAV is able to plan a path from the entrance to the desired location. Determining the location can be done by linking office information and locations in the environment model. For instance, the visitor has an appointment with mr. Smith. By contacting the database containing personnel of the building, the location of mr. Smith is determined to be office A2. Then the UAV searches for the position of office A2 and finds it as pos(x,y,z). The UAV is now able to plan a path between its current location and pos(x,y,z) in order to guide the visitor to desired location.

3.4 Conclusion

First, the problems and solutions are described that came with creating a low-cost UAV system, where the UAV carries all necessary sensors on-board. According to us, future advancements in technology will result in smaller, lighter and more powerful sensors and UAVs, thus eliminating the hardware problems tackled in this chapter.

Thereafter, an explanation for the necessity for two phases in order to create a system that is independent of the user's location is given. In the first phase, the system learns about its environment. In the second phase, the system uses this training data to navigate and execute the path. After explaining the hardware and software components, the use of the system was motivated by discussing different use cases in different settings.

Chapter 4

Implementation

In this chapter, we elaborate on the implementation based on the software architecture discussed in section 3.2. The software architecture contains different components for localization, mapping and navigation. To allow these components to collaborate, a means of communication between them is necessary. For this, the Robotic Operating System (ROS) is used.

In advance, an explanation of ROS is given, in which concepts are discussed that are used throughout the implementation. After these core concepts are explained, an overview of the complete system is given, going into detail about its components in the sections that follow.

4.1 Robotic Operating System

In order to build a modular system, a protocol for communication between processes is necessary. There are well known software-libraries for inter-process communication, like Message Passing Interface (MPI). Using these systems would force custom generating and parsing of every type of message, increasing the workload considerably. To solve this problem, ROS was chosen as a foundation for our implementation. ROS is a framework for the development of robotic software. It contains tools, libraries and conventions for abstracting the development of robotic behavior in separate computational processes.

Before continuing, it is important to explain the core ROS concepts that are used throughout the implementation.

Node Processes that perform computation are called nodes. Nodes are the base for the modular design of ROS, they allow a large system to be broken down into smaller tasks.

Message Nodes communicate with each other by passing messages to topics. A message is a data structure, filled with fields of certain types. Standard types, for example, integer, float, vector, etc. can be part of a message. Messages can also contain other messages. For example, the transformation message contains a header message, translation message and quaternion message. The 'common_msgs' ontology contains precomposed messages for geometry, sensor data and navigation that are well established throughout the community.

Topic Topics are buses for a specific type of message. Different topics with the same message type are possible, but are separated by a unique name. Each node can publish or subscribe messages to a topic. The amount of nodes that can publish or subscribe to a topic is unlimited, meaning that multiple nodes can share the same information.

Wrapper Wrappers are nodes that wrap existing device drivers. They translate the input and output of the device drivers into ROS messages so they can communicate with ROS nodes through topics.

4.2 Overview

In this section, an overview is given of the developed system to understand how the nodes interact. First, the learning phase is explained, as shown in figure 4.1. The RGB and depth information of the Kinect is sent to the openNI wrapper, which publishes this information to the '/rgb' and '/depth' topics, respectively. Information from these

topics serve as input for the active RGBD SLAM algorithm, CCNY RGBD tools or Freiburg SLAM. These algorithms process and save the information as training data. Each algorithm checks newly received information against the training data. If the information is already present, it updates the matched set of training data using the new RGB and depth information to improve the quality of the training data. If no match is found, the new information is added to the training data. The result of processing this information is an estimated position of the UAV system. This estimated position is published to the '/tf' topic. Once the RGBD SLAM algorithms have stabilized (indicated by the user), Octomap is started. Octomap subscribes to the '/depth' and '/tf' topic to build a dense 3D occupancy grid of the environment. The learning phase is ended by disabling the updates to the learning data and occupancy grid. This is done by the user once the localization and mapping components have stored sufficient information about the environment.

After the learning phase, the operation phase is started. The outline of the operation phase is shown in figure 4.2. The first node that is encountered in the operation phase is MoveIT!. Using MoveIT!, the user can specify the start and end points of a path. MoveIT! then generates a collision-free path using these points based on the 3D occupancy grid. Finally, this path is published to the '/path' topic. The actionController transforms the waypoints from '/path' into velocities. These velocities are published to the '/cmd_vel' topic, to be interpreted by the AR Drone autonomy wrapper, which sends the commands to the Parrot AR Drone 2.0 over Wifi.

We have explained the general workflow of the software architecture. Now, the localization, mapping and navigation components are discussed in detail.

4.3 RGBD SLAM

Localization is the foundation for mapping and navigation. As mentioned in the previous section, both Octomap and actionController use the estimated position by subscribing to the '/tf' topic.

To perform localization, two RGBD SLAM algorithms, CCNY RGBD tools and

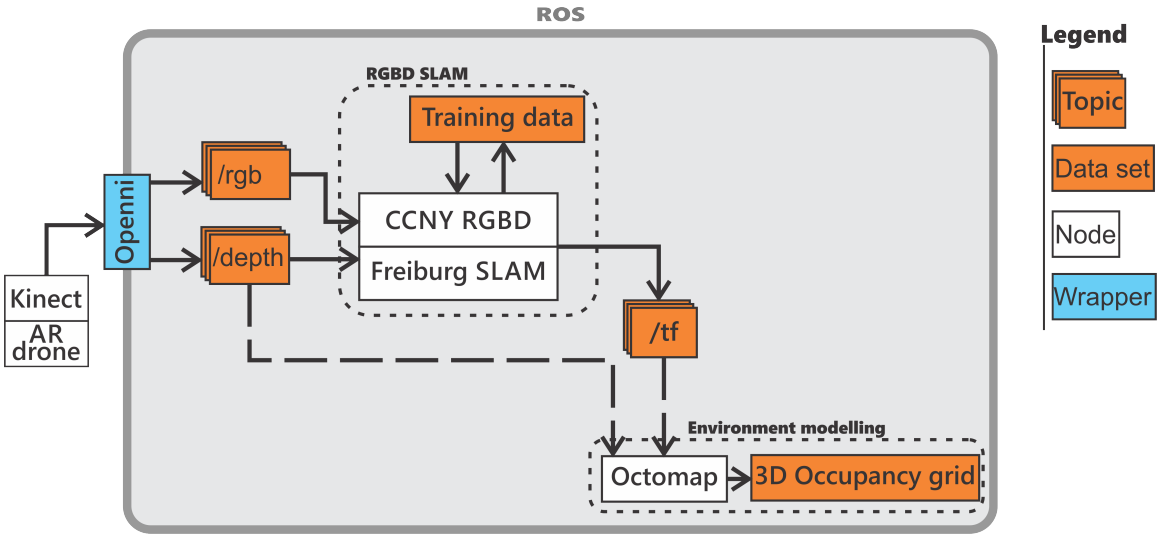

Figure 4.1: Visualization of the structure implemented in the learning phase.

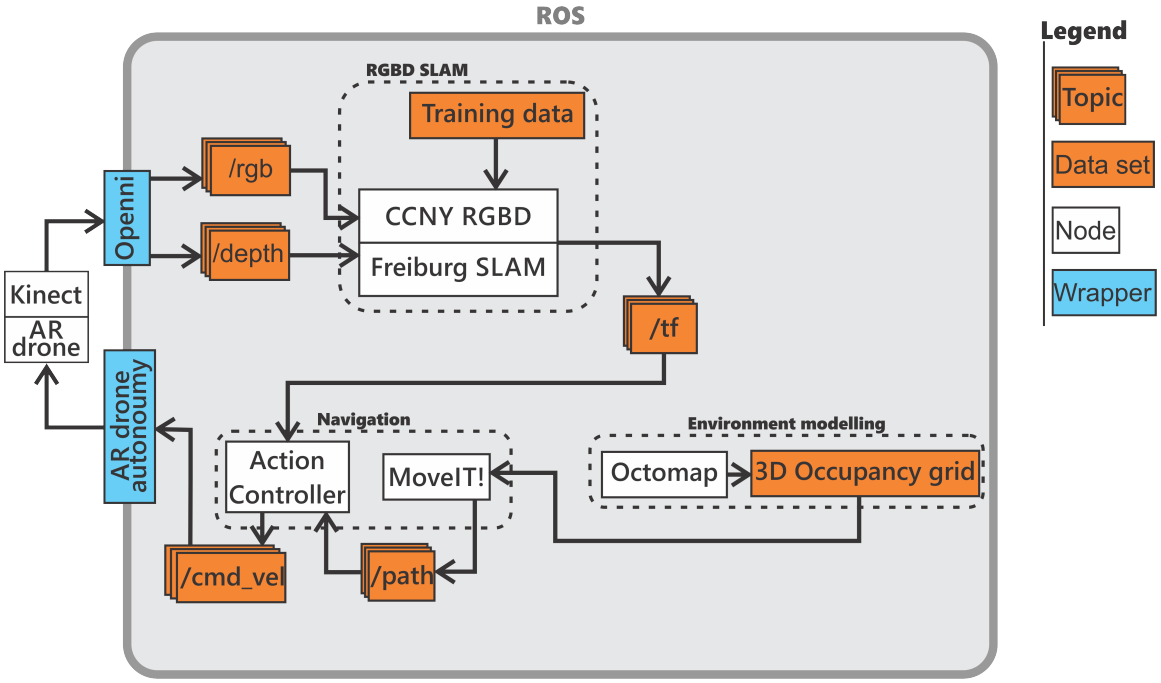

Figure 4.2: Visualization of the structure implemented in the operation phase.

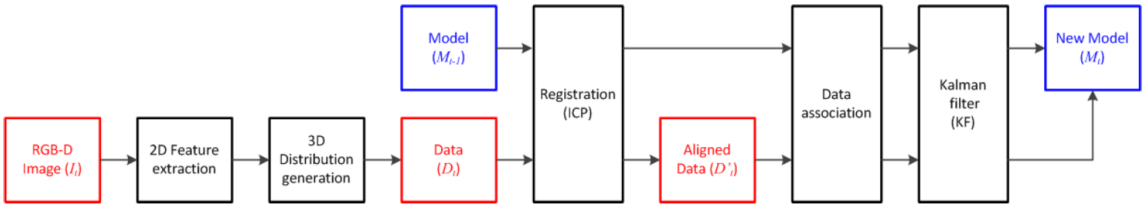

Figure 4.3: Pipeline for position estimation using CCNY RGBD tools [5].

Freiburg SLAM, are integrated. For each algorithm, a discussion about their implementation is given, followed with instructions on transferring them from learning to operation phase.

4.3.1 CCNY RGBD tools

CCNY RGBD tools is developed by Dryanovski et al. [5]. It is a performance focused visual odometry and mapping algorithm using RGBD data. It is able to run in a single thread without GPU acceleration and estimates the 6 DOF pose of the Kinect. It does not provide loop-closure in real time, this is done offline to create a final model of the room. Since our system runs in real-time, we do not use this loop-closure technique in our system.

The algorithm uses RGB and depth information from the Kinect. The RGB information is used to detect features with help of a feature detector, for example, SURF [1], ORB [28], FAST [14], etc. At the moment, these features are in 2D. Using an uncertainty model these features are mapped on the depth information, obtaining their 3D location with respect to the RGBD sensor. The 3D features are now stored as *Data*, as shown in figure 4.3. This *Data* is then compared to the *Model*, which is obtained from previous frames using ICP. ICP finds the transformation between the *Data* and the *Model*. Features that were matched in the *Model* are used to update the *Model*, using a Kalman Filter, while new features are added to the *Model*. This cycle continues until the learning phase ends. In the operation phase the *Model* is no longer updated, but is only used to align new *Data* to the *Model* using ICP.

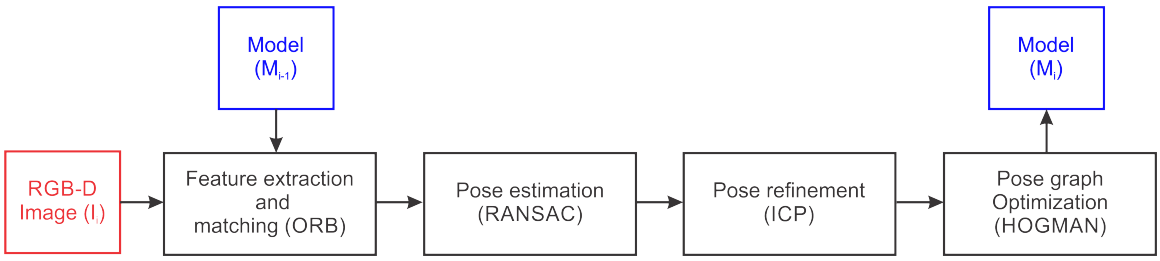

Figure 4.4: Pipeline for pose estimation using Freiburg SLAM.

4.3.2 Freiburg SLAM

The data acquisition of Freiburg SLAM, by Engelhard et al. [6], is the same as with CCNY RGBD tools. The steps of the algorithm are outlined in figure 4.4. After the data acquisition, features are extracted using ORB [28] from the incoming RGB information. These features are matched with features from previous frames. Using RANSAC, the transformation between the frames is estimated. The estimation from RANSAC is improved using a variant of the ICP algorithm. The final step is to make the pose estimation globally consistent. This is done by optimizing the pose graph, containing all previous frames, using a pose graph solver [7]. Next, an explanation on switching phases for both algorithms is given.

4.3.3 Phase switch

In section 3.2.1, we explained that the switch between learning and operation phase takes place once the training data and environment model have been build. Now, we explain how the phase switch can be done for the localization algorithms (section 4.4.1 explains the phase switch for mapping). This phase switch is done explicitly by the user. Since CCNY RGBD tools and Freiburg SLAM work in a different manner, the method for switching the phase depends on which algorithm is active. With CCNY RGBD tools, the graphical parameter server has to be started, this lets us adjust parameters inside a running node. The server is started by typing *rosrun rqt_reconfigure rqt_reconfigure* in a terminal. Thereafter, the visual_odometry node has to be selected and the option update_model has to be disabled. Hereafter, the training data is not updated anymore.

For Freiburg SLAM the phase switch is different. To stop the updating of the training data, the command *rosservice call /rgbdslam/ros_ui_b mapping false* has to be executed in a terminal. This pauses the mapping, but allows the localization to continue. The next steps are done by the actionController, explained in section 4.5.3.

4.4 Octomap

For maintaining the environment model, Octomap, developed by Wurm et al. [33], is used. Octomap has the advantage of being memory efficient and has the ability to do lower resolution queries of the map by adjusting the depth of the octree-structure, improving computational efficiency.

Octomap is a framework that implements a 3D occupancy grid. The grid is represented in an octree-structure. The octree contains voxels of equal size with a probabilistic representation of the occupancy of a region. Possible occupancy values for voxels are free, unknown or occupied. Octomap uses the estimated pose and depth information as input. The depth information is voxelized as shown in figure 4.5 and inserted in the octree according to the current position of the UAV system. Octomap can be set up separately, but for our system it is incorporated as plugin for MoveIT!.

4.4.1 Phase switch

Octomap is started as soon as the localization has stabilized. The moment Octomap is enabled is determined by the user, since the stabilization of the localization is heavily dependent on the type of algorithm used and the environment.

To switch from learning to operation phase, the Octomap node stops subscribing to the '/depth' and '/tf' topic, to stop updating the 3D occupancy grid. This is done by switching the topic names to empty. The switch is instantiated by the user when the working environment is sufficiently mapped. After the switch to the operation phase, the MoveIT! and actionController nodes are started.

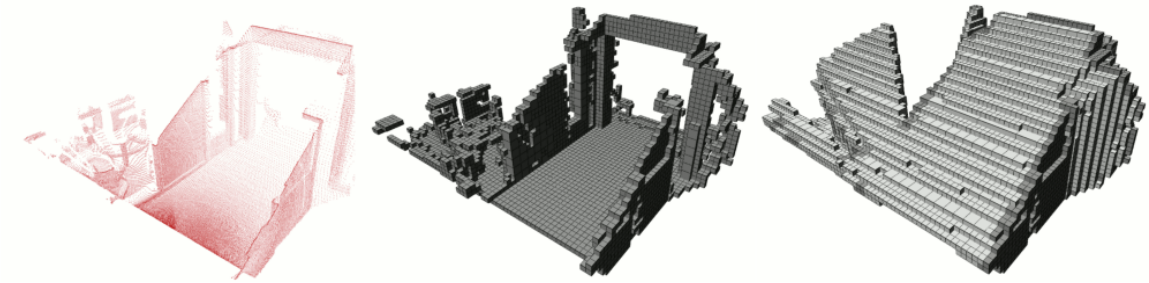

Figure 4.5: An octree map generated from example data. Left: Recorded point cloud. Center: Octree generated from the point cloud, showing occupied voxels only. Right: Octree generated from the point cloud, showing occupied voxels (dark) and free voxels (white) [33].

4.5 MoveIT!

MoveIT! has evolved from the arm navigation and grasping pipeline components of ROS into a state of the art motion planning software package. It is mainly used with ground-based robots. First, we explain the steps necessary to configure MoveIT! for UAV navigation. Thereafter, the settings for the graphical user interface, Rviz, are discussed. Finally, we discuss the actionController node, which translates the path calculated by MoveIT! to commands for the AR Drone.

4.5.1 Configuration

To plan trajectories for UAVs using MoveIT!, a package has to be configured using the MoveIT! setup assistant. First, MoveIT! needs to know the dimensions of the robot. For this, the model available in the 'hector_quadrotor' package [1] is modified. The modifications add a Kinect to the model that publishes its RGB and depth information to ROS topics. After the model is loaded into the setup assistant, a virtual joint is defined, as shown in figure 4.6a. The virtual joint is defined as a floating joint type to a fixed base, this allows the joint to move freely in 3D space. This floating joint type allows MoveIT! to be used for UAV path planning. The next step is to create a planning group with the name Quadrotor, as shown in figure 4.6b. In the final step of the setup assistant, files of the configured package are generated.

[1] http://wiki.ros.orghector_quadrotor

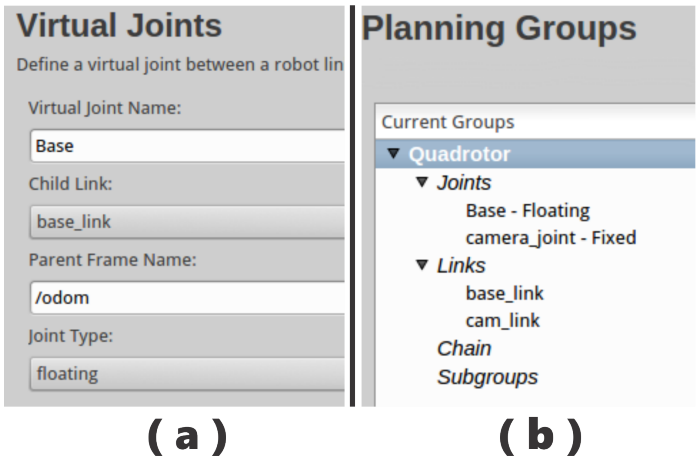

Figure 4.6: MoveIT! configuration. Settings to add a floating virtual joint (a). Settings for a planning group that contains all necessary joints and links for motion planning (b).

However, the configuration files provided by the setup assistant are not enough. The octomap node intergrated in MoveIT! does not know where to get its information. For this, the depth information and the estimated position need to be communicated. This is done by creating a configuration file that specifies the topic and type of message that is sent. An example file for depth images can be found in listing A.1 and one for point clouds in listing A.2.

The final step is to tell MoveIT! where the configuration files, containing sensor information, are located. This is done by filling in the files 'quadrotor_moveit_sensor_manager' and 'quadrotor_moveit_controller_manager' located in the launch directory of the configured MoveIT! package, these files are included in listing A.3 and A.4 respectively. Now, a fully functioning MoveIT! package is available for usage. The next step is to start MoveIT! with the configured package by running *roslaunch package_name move_group.launch*.

4.5.2 RViz-plugin

Once MoveIT! is running, it needs to know the start and end point of the path as well as which pathplanner to use. The easiest way to define these parameters is by using the MoveIT! plugin for RViz. RViz is a graphical user interface that

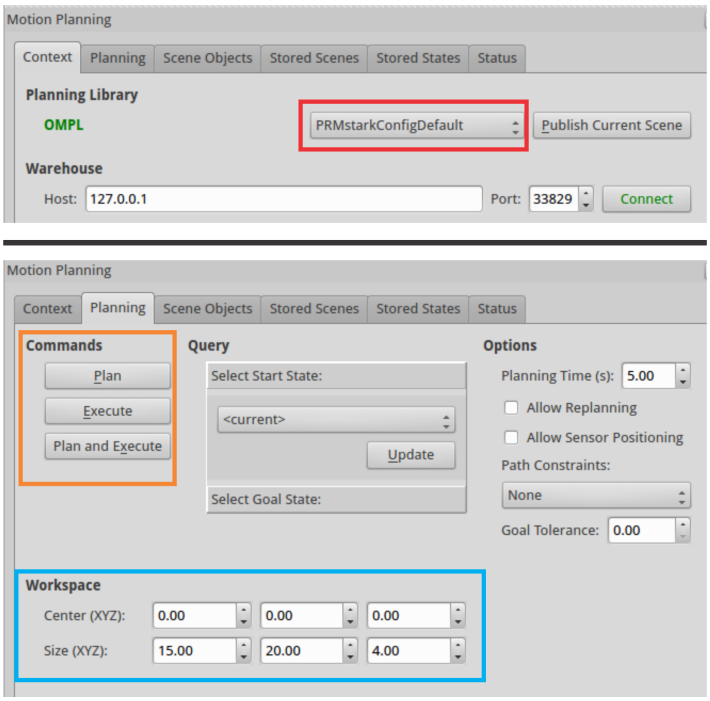

Figure 4.7: Settings panels for the MoveIT!-plugin for RViz. Red: option to select planning library. Orange: Buttons to plan a path and execute it. Blue: Dimensions of the workspace.

visualizes ROS messages in a 3D scene. The parameters that can be set are shown in figure 4.7. First, the type of planner to be used needs to be chosen, for instance, RRT or PRM. Thereafter, the workspace has to be determined. The workspace is the area in which the UAV is allowed to move. Limiting the height of the workspace is necessary, because the Octomap often does not contain the complete ceiling or floor of the environment. If the workspace is not limited, the planner is able to plan a path that goes through the floor or ceiling. After setting up the parameters, it is time to define the start and goal locations of the path. This is done by moving the 3D UAV models in RViz, as shown in figure 4.8a. Once these constraints are defined, the user tells MoveIT! to plan the path, the result is a series of waypoints, as shown in figure 4.8b. Finally, the command to execute the path is given, which sends the path to the actionController.

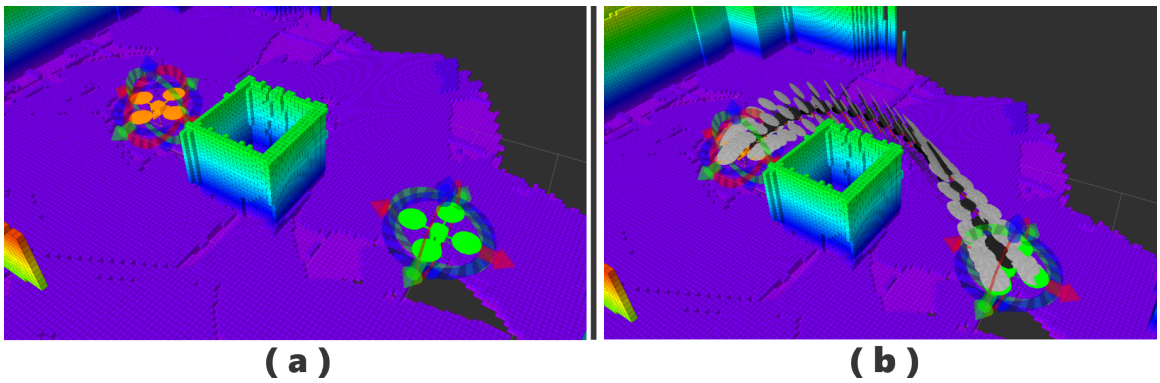

Figure 4.8: Screenshot of the MoveIT! scene in RViz, with the wayspoints of the path and the octomap visible. Green model: starting point of the path. Orange model: end point of the path.

4.5.3 ActionController

The actionController reduces the complexity of the path from 6 DOF to 4 DOF, based on the findings of Valenti et al. [29] discussed in section 2.2.2.2, and has two main tasks. The first task is to translate the 3D points from the planned path into velocities (meters per second). The second task is to ensure that the Kinect is always facing the same direction as the AR Drone is moving, to detect obstacles.

The planned path contains several waypoints, each having a 3D point and rotation. Since the path is limited to 4 DOF, the rotation is completely discarded. A rotation, only containing yaw, is calculated based on the direction the sensor is currently facing and the direction to the next waypoint. The pitch and roll are not calculated, since they are inherent to UAV movement. After the rotation is complete, the distance between the current position and the next waypoint is calculated. This distance is divided by the speed of movement, returning the seconds that are needed to execute the movement. The speed and rotation are published as 'geometry_msgs/Twist' messages to the '/cmd_vel' topic. These messages contain a vector for linear and angular velocity. The pseudo-code of actionController can be found in listing A.5.

The navigation and environment modeling heavily depend on the estimated UAV position. To test the navigation and environment modeling decoupled from the localization, a simulation environment is used.

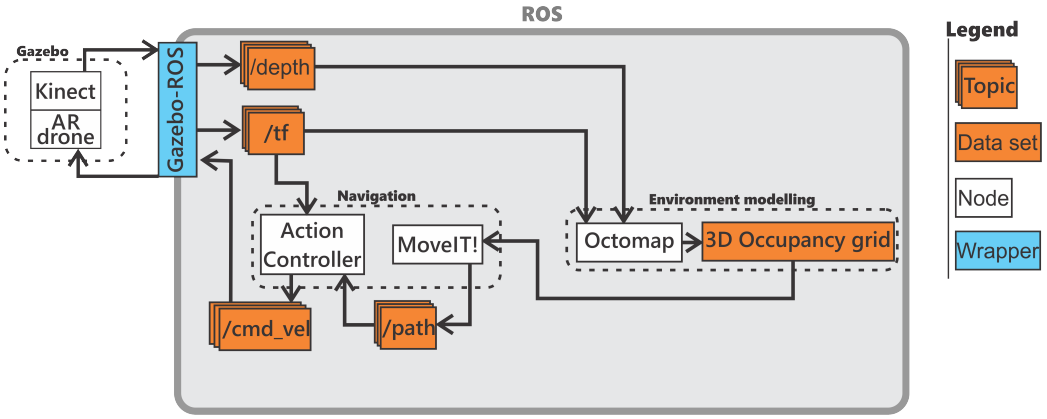

Figure 4.9: Visualization of the structure used in the simulation.

4.5.4 Simulation

Gazebo[2], a simulation environment for robotics, is used to test mapping and navigation separately. Figure 4.9 shows the structure behind the implemented simulation. As input, Gazebo expects a robot model, similar to that used in MoveIT!. The model used for MoveIT! is extended for Gazebo. This extension is done by adding Gazebo-specific entities in the XML-document of the model. These entities specify the type of messages and the topic to which the RGB and depth information need to be published. The UAV model with Kinect is shown in figure 4.10. To determine the UAV position, the ground truth provided by Gazebo is used to purely test the navigation. Gazebo publishes the UAV position and depth information to the '/tf' and '/depth' topics to allow integration with Octomap and actionController.

4.6 Conclusion

An overview of the implementation was given in this chapter, explaining the communication between the components. Next, each component was discussed in detail and the input they expect and the output that is generated. How to switch between learning and operation phase was explained as well, for the localization and mapping components. A system was developed that is modular, thus components can be replaced by similar ones.

[2]http://gazebosim.org/

Figure 4.10: Modified hector_quadrotor model with a Kinect attached for simulation with gazebo.

Chapter 5

Evaluation

In the previous chapter, we explained the implementation. In this implementation, the estimated position is fundamental for both mapping and navigation. This is the reason to extensively test and evaluate the localization, which estimates the position.

The first test in this chapter was used to determine if CCNY RGBD tools or Freiburg SLAM was applicable for UAV localization. Next, we tested the accuracy of both algorithms. Each section starts by explaining the test setup. Thereafter, the results from the test are disclosed. Finally, we analyze the usability of the localization algorithms, CCNY RGBD tools and Freiburg SLAM, for UAV localization.

5.1 Applicability

Before the AR drone and Kinect were integrated, we tested Freiburg SLAM and CCNY RGBD tools to determine if they are able to localize within an environment.

5.1.1 Test setup

During this test, we wanted to provide the localization algorithms with a best case scenario for estimating the position. A Kinect and laptop were mounted on a dolly, allowing the Kinect to be moved using a smooth and stable motion. The test environment was a hallway. First, the dolly was slowly driven around to build up training

data. Next, we moved the dolly between two marked locations, 1.5 meters apart. This movement was repeated five times.

5.1.2 Results

None of the measurements had an error of more than 20 centimeters. The 3D occupancy grid build up from the training data is displayed in figure 5.1. The resulting error and final model of the hallway led us to believe that the localization algorithms were suited for UAV localization.

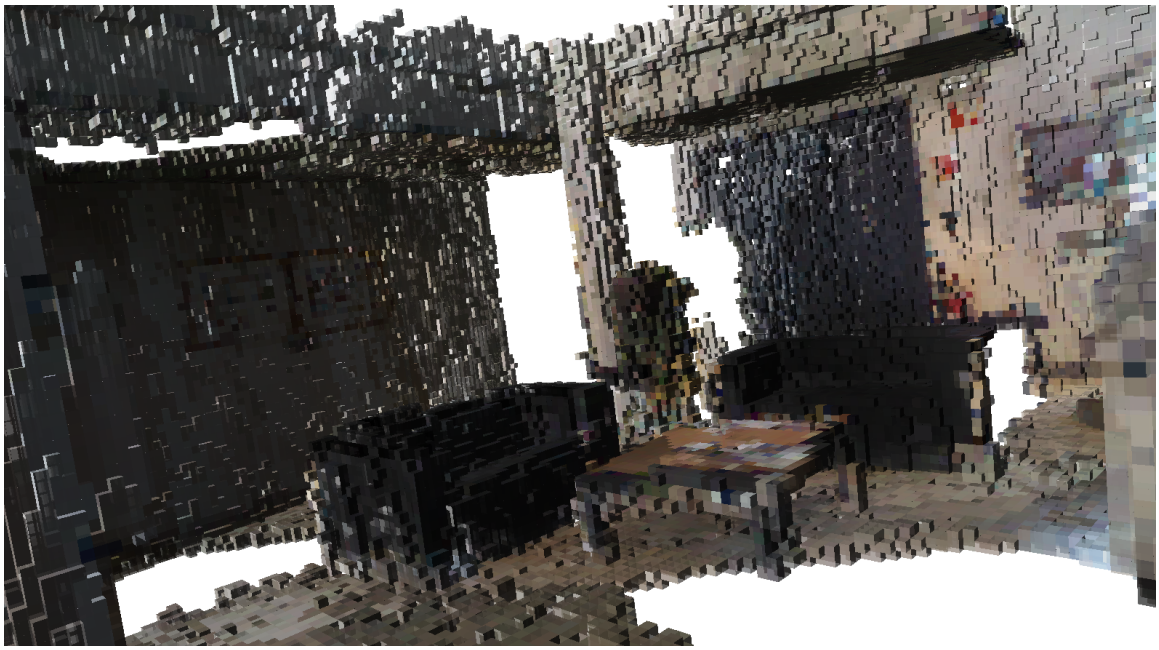

Figure 5.1: The resulting colored 3D occupancy grid the scan of the hallway in the EDM. Visible are the to black couches and the table in the center. In the background, pillars and walls are visible.

However, during the integration of the Kinect and the UAV, several issues arose with the accuracy of the localization. The free motion of scanning a room with a hand-held Kinect resulted in inaccurate localization of the UAV. The map that was built up had severe errors in it, as seen in figure 5.2. The error is highlighted by the yellow line, here it is clearly visible that in a scanned environment with one floor, two floors were detected and localization drifted heavily.

To investigate if the localization algorithms were able to estimate a position in

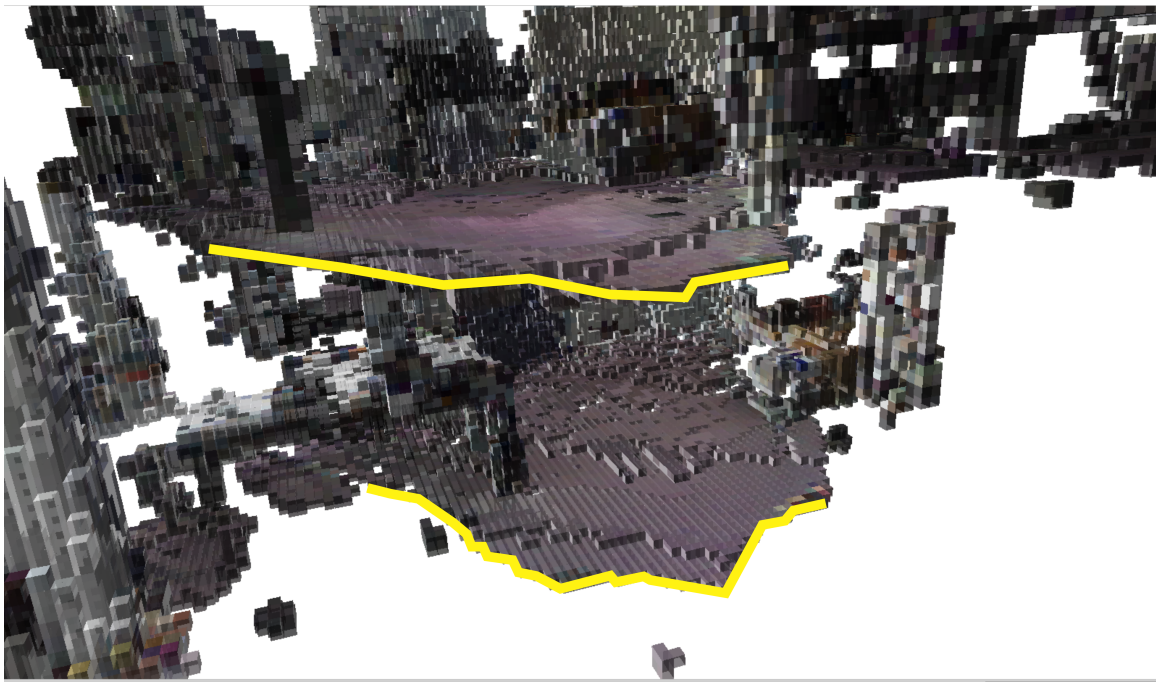

Figure 5.2: A colored 3D occupancy grid of the test room in the EDM. The scanned room was a consisted of a single floor without any difference in height. The yellow lines mark the edges of each detected floor, thus marking the error.

a previously captured environment, the software architecture was split up into two phases, as described in section 3.2. Using these phases, we tried to have the UAV system navigate in a real-world environment. Due to the fluctuations in the stability of the localization, the UAV system was unpredictable and unsafe to be used with the current issues.

5.2 Accuracy

The localization problems discovered during the integration of the hardware and software components in the UAV system motivated us to do a larger scale evaluation to determine the accuracy of both localization algorithms. The evaluation is focused on ability of the algorithms to localize the UAV system after training data is gathered.

5.2.1 Test setup

Camera motion and lighting differences in the output of the RGBD camera (in this case the Kinect) can vary, even when the camera follows a similar trajectory in the same room. These differences can influence the localization algorithms, which is undesirable for the comparison of these algorithms. To eliminate these influences, the RGBD output needs to be recorded and played back to each algorithm to allow unbiased comparison.

The RGBD data is recorded using a set of tools for recording and playing back ROS messages, called rosbag[1]. CCNY RGBD tools requires information about the transformation between the lenses of the Kinect, which is published to the '/tf' topic. Freiburg SLAM on the other hand, does not work when these transformations are being published, thus not allowing the recording of the '/tf' topic. To circumvent this problem, the static transform publisher[2] incorporated in the tf package is used. This publisher publishes messages to the '/tf' topic, allowing us to simulate the messages necessary for running CCNY RGBD tools while using the same RGBD input.

The first step of the test is recording the RGBD input needed for the learning phase. For this, the Kinect was moved through the environment slowly. The slow movement is necessary to assure that both algorithms have enough time to build up the training data. After the training data was recorded, a 3x3 meter grid was measured out in the environment, as shown in figure 5.3, creating 16 intersections. At each intersection, 15 second samples are recorded with the Kinect facing the front, right and back of the room. This was repeated for different heights (0.9, 0.6 and 1.2 meter), resulting in 48 samples per height and a total of 144 samples.

Before the data gathering can commence, it is necessary for the localization algorithms to build up training data. Each algorithms builds up the training data separately, using the RGBD data from the full scan of the environment, which is played back. Once the algorithm has build up training data, it is transferred to the operation phase. In this phase, each of the 144 recorded samples are replayed to the

[1]Information about rosbag: http://wiki.ros.org/rosbag

[2]Information about static transform publisher: http://wiki.ros.org/tf#static_transform_publisher

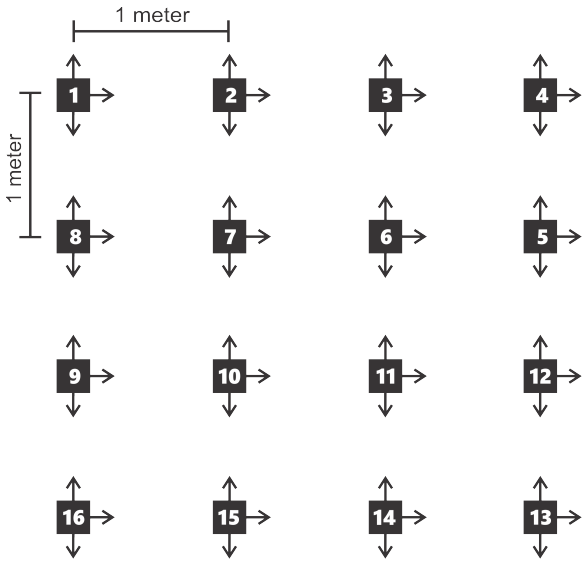

Figure 5.3: Schematic of the test setup. At each location, the three rotations are measured during 15 seconds. This is repeated for three different heights (0.6, 0.9 and 1.2 meter). The locations are numbered according to the sequence in which they were visited.

algorithm. After a sample has been played back, a variable is saved indicating if a position was estimated, together with the x-, y- and z-values of the last position that was found. The following section discloses the results of the test.

5.2.2 Results

In this section, the estimated positions, based on the recorded samples, by CCNY RGBD tools and Freiburg SLAM are compared. To compare the results from both algorithms, the coordinate systems must be aligned. However, the alignment is not possible, since each algorithms determines its coordinate system at runtime and the transformation between them is unknown. This problem is solved by transforming the first estimated position, for each algorithm, to the origin. This transformation is applied to all estimated positions, resulting in positions relative to the first estimated position (now the origin).

The results for CCNY RGBD tools and Freiburg SLAM are summarized in table 5.1. CCNY RGBD tools was not able to estimate a position for 25 samples, resulting in an availability of 82.64%. Freiburg SLAM failed to estimate a position

for 72 samples, resulting in a availability of 50%. To compare the accuracy of both algorithms, only the estimation error of available positions are considered. The estimation error is calculated using the distance of the estimated position to the origin. The difference between this distance and the distance that was traversed in reality (1 meter between sequential locations) is the estimation error in meters. Table 5.1 lists statistics about the estimation error. During the calculation of these statistics the first estimated position was not included, because this is the reference location and inherently is 100% accurate, thus influencing the statistics heavily. A first estimate about the accuracy can be derived from the mean. Freiburg SLAM is clearly more accurate, with a mean of 0.93 m, than CCNY RGBD tools, with a mean of 1.42 m. However, the mean does not measure the consistency of the position estimation. This consistency is reflected by the standard deviation, which shows the amount of variation from the mean. This indicates how close the estimated positions are to the mean. CCNY RGBD tools has a standard deviation of 1.13 m, which is almost double of Freiburg SLAM with 0.68 m. A lower standard deviation means that other estimation errors are closer to the means, thus the distribution of CCNY RGBD tools is spread over a larger range of possible values.

	CCNY RGBD Tools	Freiburg SLAM
# of samples	144	144
# of positions estimated	119	72
% of positions estimated	82.64%	50.00%
Estimation error (m)		
Minimum	0.006589	0.028129
Maximum	4.263358	2. 613484
Mean	1.420212	0.931467
Standard deviation	1.129887	0.676038
68th percentile	2.550099	1.607505
95th percentile	3.679986	2.283544
99th percentile	4.809872	2.959582

Table 5.1: Positional accuracy of localization comparing CCNY RGBD tools and Freiburg SLAM. The statistics for the estimation error are based on the number of positions estimated, not the number of samples

The larger standard deviation of CCNY RGBD tools also reflects in the minimum and maximum estimation error, which are smaller and larger than those of Freiburg

SLAM. These minimum and maximum are used to model the heat maps, shown in figure 5.4. The minimum and maximum error from CCNY RGBD tools are also used to model the heat maps of Freiburg SLAM. Building the heat maps of both algorithms with different minimum and maximum would not make them comparable, the color-code would not depict the same range of values. Each cell in the heat maps contains the average error of estimated position gotten from the three samples taken at that location. From the heat maps, the differences in the distribution of both algorithms are visible. CCNY RGBD tools clearly has a larger amount of bright colored cells, indicating the higher standard deviation. The availability is represented in the heat map as well, by the black colored cells. These cells indicate that no position could be estimated for that location. In figure 5.5, the estimation errors are distributed over 0.2 meter increments. Each increment contains the percentage of the estimated position with an error smaller or equal than the increment. From this distribution, we can see that for CCNY RGBD tools a lower percentage of estimation errors is contained within each increment, compared to Freiburg SLAM. This indicates the lower accuracy and variation of Freiburg SLAM as well.

A visualization of the standard deviation and the 68-95-99.7 rule is shown in figure 5.6. The 68th percentile means that 68.27% of the positions are smaller or equal to a certain value, 2.55 m for CCNY RGBD tools and 1.61 m for Freiburg SLAM. The 95th percentile applies this to 95.45% of the values, 3.68 m and 2.28 m for CCNY RGBD tools and Freiburg SLAM respectively. 99.73% of the positions estimated by CCNY RGBD tools or Freiburg SLAM are equal or smaller than 4.81 m and 2.96 m respectively. This representation is useful for the development of applications that incorporate the localization algorithms. The different percentile can be incorporated in the creation of a safety perimeter around the UAV. Using these results, we analyze if these localization algorithms are usable for estimating the position of a UAV system.

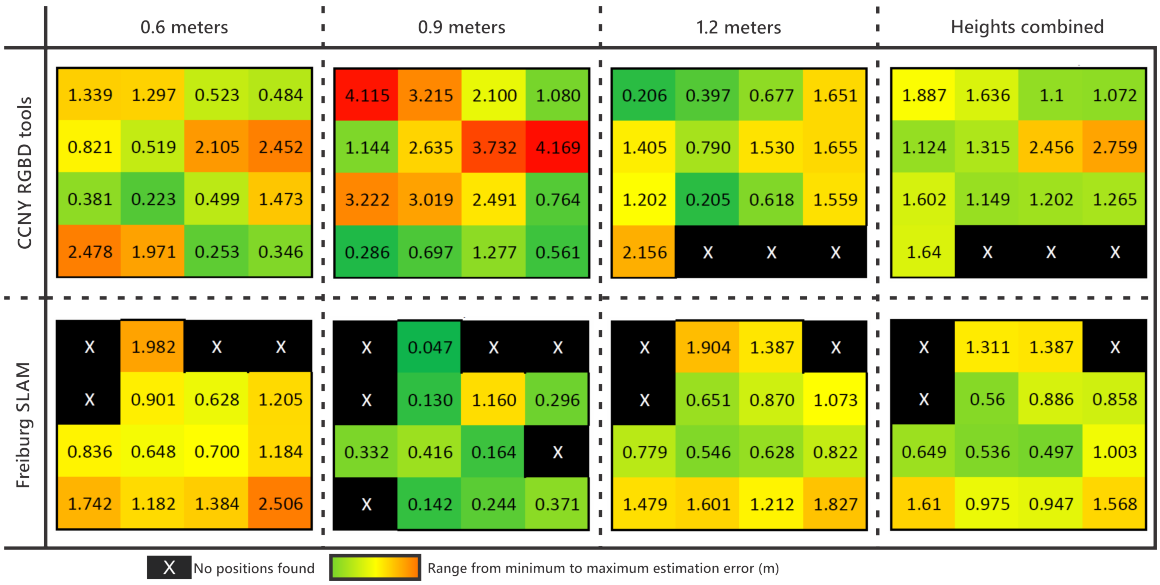

Figure 5.4: Heatmaps for CCNY RGBD tools and Freiburg SLAM per height (0.6, 0.9 and 1.2 meter) and the combination of the three heights, containing the average estimation error for all rotations per location. The value in a cell is the average estimation error of the estimated positions at that location in meters. Black squares indicate that no positions could be estimated for that location. The color code ranges from the minimum and maximum error, 0.01 m and 4.26 m respectively.

5.3 Conclusion

The results from both CCNY RGBD tools and Freiburg SLAM have been compared. Through observation the results from the applicability test were proven to be unreliable and thus motivated a deeper evaluation of the accuracy of the localization. CCNY RGBD tools has the advantage that it is able to estimate a position for more locations, although the estimated position is not accurate and is susceptible to large variation from the mean. On the other hand, Freiburg SLAM can only estimate a position half of the time, but has a higher accuracy and lower variance for these estimations. Which algorithm to use depends on the scenario for which it is used. If quick reaction time is required, but higher accuracy is not, then CCNY RGBD tools is the better choice. On the other hand, if reliable and lower accuracy is required, but the UAV system does not have to react instantly, Freiburg SLAM is the better choice.

The accuracy of CCNY RGBD tools is lower, because it does not perform loop-

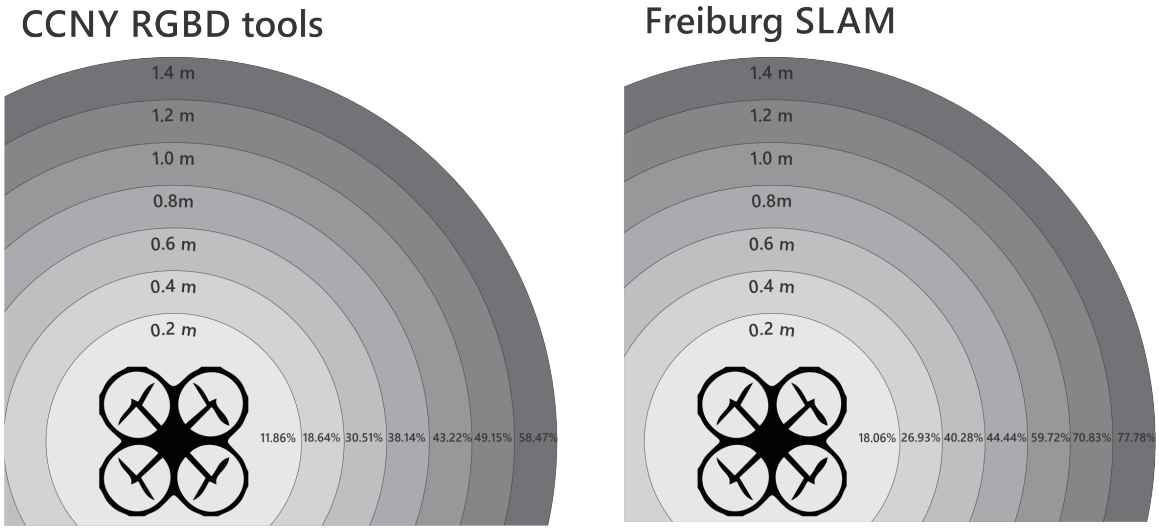

Figure 5.5: Distribution of the percentage of estimation errors in 0.2 meter increments. Each increment also contains the estimation errors of the lower increments.

closure (SLAM) in real-time. This can be resolved by adding this loop-closure in real-time. Freiburg SLAM's accuracy can be increased by using a high-end computer with CUDA enabled GPU. This allows Freiburg SLAM to incorporate SIFTGPU, which have a higher accuracy.

To use any of the algorithms discussed for UAV localization, a circular perimeter has to be set with a radius equal to the size of the 99th percentile. This assures that the estimation error does not cause the UAV to crash into the environment. In the best case, being Freiburg SLAM, this perimeter has a radius 2.96 meter. Rooms and hallways would need to have a height and width larger than the diameter of the perimeter to allow the UAV to move at all. This means, the currently integrated localization algorithms are not usable for the purpose of UAV localization in small indoor environments. However, it can still provide useful in larger environments (i.e. hangar or construction site), where high precision movements is not required.

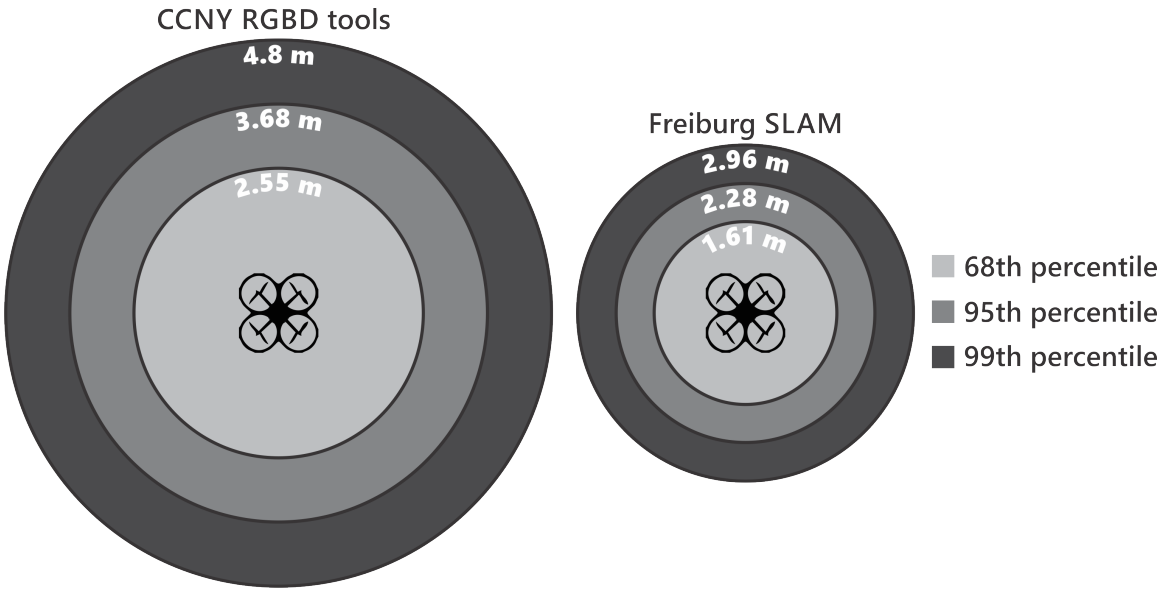

Figure 5.6: Accuracy of CCNY RGBD tools and Freiburg SLAM represented according to the 68-95-99.7 rule. CCNY RGBD tools: 68% of the estimated position have an estimation error smaller than 2.55 meter, 95% of the estimated position have an estimation error smaller than 3.68 meter and 99.7% of the estimated position have an estimation error smaller than 4.8 meter. Freiburg SLAM: 68% of the estimated position have an estimation error smaller than 1.61 meter, 95% of the estimated position have an estimation error smaller than 2.28 meter and 99.7% of the estimated position have an estimation error smaller than 2.96 meter.

Chapter 6

Conclusion

The goal of this thesis was to investigate the use of low-cost hardware to enable autonomous UAV flight for use in human-UAV interaction. The subject is motivated by the recent surge in popularity of UAVs as well as the price drop of sensors and UAVs. In this chapter, we reflect on the different aspects of the thesis and summarize our findings in the area of hardware, software and the usefulness of the UAV system for human-UAV interaction, as well as recommendations for future work.

6.1 Hardware setup

A major part of this thesis was constructing a low-cost hardware setup that allowed a UAV to maintain flight, while carrying the sensors necessary for localization, mapping and navigation on-board. The Parrot AR Drone 2.0 and Microsoft Kinect were used to create a UAV system for under 500 euro. During the development of the hardware setup, there were various challenges with payload capacity, flight stability and flight duration. The AR Drone and Kinect had to be modified to allow stable flight. We are aware that UAVs exist which can carry the Kinect as is, but often these are more expensive and not controllable through a computer without additional hardware, increasing the cost further. The same is true for sensors. Lighter, but more expensive sensors are becoming available, such as the sensors embedded in Project Tango[1] or

[1]Project page of Project Tango: https://www.google.com/atap/projecttango/#project

the Skybotix VISensor[2]. At this time, these sensors are still in development and not commercially available.

6.2 Software architecture

The software architecture running on top of the hardware setup consists out of three main components (localization, mapping and navigation). Localization was the most difficult to perform. The accuracy of the localization was vital for mapping and navigation. For this reason, the navigation and mapping were developed separately, and tested in simulation. It allows the UAV to plan a collision-free path and follow it with high precision. The precision with which the UAV is able to follow a path depends on the accuracy of the localization, since it uses the current position to execute the path.

6.3 UAV interactions

In section 3.3, example scenarios (distant inspection, item retriever and personal receptionist) were introduced, in which a self-contained UAV system could be deployed. Now that the accuracy and availability of the localization has been determined, we discuss the feasibility of these scenarios using the current localization.

For distant inspection, lower accuracy and availability can suffice when the UAV system is deployed in larger, open environments. The user is provided with controls through a smartphone, enabling small movement corrections, compensating for the low accuracy. Even the worst case availability of 50% is not an issue, since it is not a requirement that the UAV flies at high speed, the position can be estimated over more frames and thus a longer period of time.

With item retriever, the UAV and user move close to each other. Here, it is possible for the UAV to crash into a user, due to the low accuracy. Picking up items is also not possible, without exactly knowing the position of the UAV. Low availability

[2]Information about the VI Sensor: http://www.skybotix.com/vi-sensor-early-adopter-application/

of the localization is not desirable as well, if the UAV takes a longer time to retrieve an item than it would the user, the UAV system is not providing a useful service.

The final scenario was a personal receptionist. The low accuracy has the same issues as with the item retriever. User safety cannot be ensured, since the UAV flies in the vicinity of the user. The other issue is that when two doors are close together, the UAV might indicate the wrong door, or even a wall as the location of interest. The availability can provide problems as well, if the UAV system is not able to localize within a reasonable time span, it will slow down the user, which is undesirable. Since the current implementation is sub-optimal, both on speed, safety and accuracy, we present recommendations for future improvements in the next section.

6.4 Future work

This section contains recommendations for future work. These recommendations are divided into two categories. The first category focuses on the use of UAVs for human-UAV interaction, while the second contains recommendations for further improving the developed self-contained UAV system.

6.4.1 Human-UAV interaction

In section 2.3.2, FollowMe [21] and Joggobot [20] were explained. These systems allow a UAV to interact with a user, while maintaining a fixed distance relative to the user. This approach is suitable to test the capability of a UAV to track and follow a user or to recognize gestures during flight.

As mentioned, the UAV system was developed with the prospect of human-UAV interaction. An important step in this kind of interaction is to investigate to acceptance of the user towards the UAV system. We propose setting up a user test and let the user self-report on his experiences with the UAV system.

System Usability Scale (SUS)[3] is an industry standard for measuring the usability of a wide range of products and services. SUS can give an initial estimate of the

[3]http://www.usability.gov/how-to-and-tools/methods/system-usability-scale.html

usability, on a scale between 0 and 100. However, low scores on the SUS questionnaire do not yield any meaning why the system has a low score. To apprehend the reason for the low score, the user should be able to make remarks about the different systems based on their own opinion, after filling in the questionnaire.

Comparing the UAV system to a baseline, executing the same test scenario, can also provide useful. Even if the feedback on the interaction using the UAV system might be negative, it is still possible that the UAV system has some advantages over the baseline system.

6.4.2 Self-contained UAV system

From chapter 5, it is clear that the current localization algorithms are inaccurate. Therefor, we propose possible solutions that can improve the accuracy of localization. First, solutions that incorporate the low-cost RGBD camera are explained. Afterwards, we provide solutions to test the navigation by using a higher cost external tracking system.

6.4.2.1 RGBD localization

We present three possible solutions to improve the localization using an RGBD camera. The first two focus on optimizing the CCNY RGBD tools and Freiburg SLAM, while the last replaces the current localization algorithms.

Extended Kalman filter Adding an extended Kalman filter for robot poses[4] to CCNY RGBD tools allows the visual odometry to be combined with IMU data. By combining IMU data, the uncertainty of the visual odometry is reduced, since the IMU gives information about the roll, pitch and yaw of the UAV system. This optimizes the position estimation of CCNY RGBD tools with the correct rotation of the UAV system.

[4]http://wiki.ros.org/robot_pose_ekf

GPU accelerated Freiburg SLAM According to Engelhard et al. [6], it is possible to increase the accuracy of Freiburg SLAM by deploying it on a computer with a CUDA enabled GPU. Using this GPU, Freiburg is able to extract SIFTGPU [32] features, which allows higher accuracy, thus decreasing the position estimation error.

KinectFusion Exploring the usage of KinectFusion [22] for localization. To use KinectFusion for large areas, a high-end computer with CUDA enabled GPU is necessary. The KinectFusion implementation would have to be modified to publish and subscribe to ROS topics, to allow integration in the current software architecture.

6.4.2.2 External localization

In order to test the navigation and mapping in a real life setting, an external tracking system (i.e. Naturalmotion Optitrack) can be used to track the UAV system. By attaching a light weight marker to the UAV system, the UAV is able to fly without having to remove the RGBD camera, since no significant payload is added. Using an external tracking system, the UAV system is able to localize within a limited size environment. This is useful to test the mapping and navigation, while drift and external forces (i.e. wind) influence the movement of the UAV system. It can also be used to test human-UAV interaction, as described in section 6.4.1, in a confined environment.

Bibliography

[1] H. Bay, T. Tuytelaars, and L. Van Gool. Surf: Speeded up robust features. In *Computer Vision–ECCV 2006*, pages 404–417. Springer, 2006.

[2] A. Censi. An ICP variant using a point-to-line metric. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, Pasadena, CA, May 2008.

[3] S. Chitta, I. Sucan, and S. Cousins. Moveit! *IEEE Robotics Automation Magazine*, 19(1):18–19, 2012.

[4] I. Dryanovski, W. Morris, and J. Xiao. Multi-volume occupancy grids: An efficient probabilistic 3d mapping model for micro aerial vehicles. In *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on*, pages 1553–1559. IEEE, 2010.

[5] I. Dryanovski, R. Valenti, and J. Xiao. Fast visual odometry and mapping from rgb-d data. In *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, pages 2305–2310. IEEE, 2013.

[6] N. Engelhard, F. Endres, J. Hess, J. Sturm, and W. Burgard. Real-time 3d visual slam with a hand-held rgb-d camera. In *Proceedings of the RGB-D Workshop on 3D Perception in Robotics at the European Robotics Forum, Vasteras, Sweden*, volume 2011, 2011.

[7] G. Grisetti, R. Kummerle, C. Stachniss, U. Frese, and C. Hertzberg. Hierarchical optimization on manifolds for online 2d and 3d mapping. In *Robotics and Automation (ICRA), 2010 IEEE International Conference on*, pages 273–278. IEEE, 2010.

[8] J. Hightower and G. Borriello. A survey and taxonomy of location systems for ubiquitous computing. *IEEE computer*, 34(8):57–66, 2001.

[9] K. Higuchi, Y. Ishiguro, and J. Rekimoto. Flying eyes: Free-space content creation using autonomous aerial vehicles. In *CHI '11 Extended Abstracts on Human Factors in Computing Systems*, CHI EA '11, pages 561–570. ACM, 2011.

[10] K. Higuchi and J. Rekimoto. Flying head: a head motion synchronization mechanism for unmanned aerial vehicle control. In *CHI'13 Extended Abstracts on Human Factors in Computing Systems*, pages 2029–2038. ACM, 2013.

[11] K. Higuchi, T. Shimada, and J. Rekimoto. Flying sports assistant: external visual imagery representation for sports training. In *Proceedings of the 2nd Augmented Human International Conference*, page 7. ACM, 2011.

[12] K. LaFleur, K. Cassady, A. Doud, K. Shades, E. Rogin, and B. He. Quadcopter control in three-dimensional space using a noninvasive motor imagery-based brain–computer interface. *Journal of neural engineering*, 10(4):046003, 2013.

[13] D. Lentink, S. Jongerius, and N. Bradshaw. The scalable design of flapping micro-air vehicles inspired by insect flight. In *Flying Insects and Robots*, pages 185–205. Springer, 2010.

[14] T. Lindeberg. Feature detection with automatic scale selection. *International journal of computer vision*, 30(2):79–116, 1998.

[15] Q. Lindsey, D. Mellinger, and V. Kumar. Construction with quadrotor teams. *Autonomous Robots*, 33(3):323–336, 2012.

[16] D. Lowe. Object recognition from local scale-invariant features. In *Proceedings of the seventh IEEE international conference on Computer vision, 1999.*, volume 2, pages 1150–1157. Ieee, 1999.

[17] S. Lupashin, A. Schollig, M. Hehn, and R. D'Andrea. The flying machine arena as of 2010. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 2970–2971. IEEE, 2011.

[18] W. Morris, I. Dryanovski, and J. Xiao. 3d indoor mapping for micro-uavs using hybrid range finders and multi-volume occupancy grids. In *RSS 2010 workshop on RGB-D: Advanced Reasoning with Depth Cameras, Zaragoza, Spain*, 2010.

[19] W. Morris, I. Dryanovski, and J. Xiao. Cityflyer: Progress toward autonomous mav navigation and 3d mapping. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 2972–2973. IEEE, 2011.

[20] F. Mueller, E. Graether, and C. Toprak. Joggobot: Jogging with a flying robot. In *CHI '13 Extended Abstracts on Human Factors in Computing Systems*, CHI EA '13, pages 2845–2846, New York, NY, USA, 2013. ACM.

[21] T. Naseer, J. Sturm, and D. Cremers. Followme: Person following and gesture recognition with a quadrocopter. In *Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on*, pages 624–630. IEEE, 2013.

[22] R. Newcombe, A. Davison, S. Izadi, P. Kohli, O. Hilliges, J. Shotton, D. Molyneaux, S. Hodges, D. Kim, and A. Fitzgibbon. Kinectfusion: Real-time dense surface mapping and tracking. In *Mixed and augmented reality (ISMAR), 2011 10th IEEE international symposium on*, pages 127–136. IEEE, 2011.

[23] K. Nitta, K. Higuchi, and J. Rekimoto. Hoverball: augmented sports with a flying ball. In *Proceedings of the 5th Augmented Human International Conference*, page 13. ACM, 2014.

[24] C. Powers, D. Mellinger, A. Kushleyev, B. Kothmann, and V. Kumar. Influence of aerodynamics and proximity effects in quadrotor flight. In *Experimental Robotics*, pages 289–302. Springer, 2013.

[25] O. Purwin and R. D'Andrea. Performing aggressive maneuvers using iterative learning control. In *Robotics and Automation, 2009. ICRA'09. IEEE International Conference on*, pages 1731–1736. IEEE, 2009.

[26] C. Richter, A. Bry, and N. Roy. Polynomial trajectory planning for quadrotor flight. In *International Conference on Robotics and Automation*, 2013.

[27] R. Ritz, MW. Muller, M. Hehn, and R. D'Andrea. Cooperative quadrocopter ball throwing and catching. In *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, pages 4972–4978. IEEE, 2012.

[28] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski. Orb: an efficient alternative to sift or surf. In *Computer Vision (ICCV), 2011 IEEE International Conference on*, pages 2564–2571. IEEE, 2011.

[29] R. Valenti, I. Dryanovski, C. Jaramillo, D. Strom, and X. Jizhong. Autonomous quadrotor flight using onboard rgb-d visual odometry. In *International Conference on Robotics and Automation (ICRA 2014)*, 2014.

[30] T. Whelan, M. Kaess, M. Fallon, H. Johannsson, J. Leonard, and J. McDonald. Kintinuous: Spatially extended kinectfusion. 2012.

[31] J. Willmann, F. Augugliaro, T. Cadalbert, R D'Andrea, F Gramazio, and M. Kohler. Aerial robotic construction towards a new field of architectural research. *International journal of architectural computing*, 10(3):439–460, 2012.

[32] C. Wu. Siftgpu: A gpu implementation of scale invariant feature transform (sift), 2007.

[33] K. Wurm, A. Hornung, M. Bennewitz, C. Stachniss, and W. Burgard. Octomap: A probabilistic, flexible, and compact 3d map representation for robotic systems. In *Proceedings of the ICRA 2010 workshop on best practice in 3D perception and modeling for mobile manipulation*, volume 2, 2010.

Appendix A

Listings

```
1 sensors:
2   - sensor_plugin: occupancy_map_monitor/
        DepthImageOctomapUpdater
3     point_cloud_topic: /depth
4     max_range: 5.0
5     frame_subsample: 1
6     point_subsample: 1
```

Listing A.1: Example configuration file for MoveIT! to update the Octomap using a depth image

```
1 sensors:
2   - sensor_plugin: occupancy_map_monitor/
        PointCloudOctomapUpdater
3     point_cloud_topic: /points
4     max_range: 5.0
5     frame_subsample: 1
6     point_subsample: 1
```

Listing A.2: Example configuration file for MoveIT! to update the Octomap using a point cloud

```
<launch>
<param name="octomap_frame" type="string" value="odom" />
<param name="octomap_resolution" type="double" value="0.05" />
<param name="max_range" type="double" value="6.0" />
<rosparam command="load" file="$(find notfDrone)/config/
    sensors_rgbd.yaml" />
</launch>
```

Listing A.3: Example launch file for the sensor manager in MoveIT

```
<launch>
 <!-- Set the param that trajectory_execution_manager needs to
     find the controller plugin -->
  <arg name="moveit_controller_manager" default="
      moveit_simple_controller_manager/
      MoveItSimpleControllerManager" />
  <param name="moveit_controller_manager" value="$(arg
      moveit_controller_manager)"/>

  <!-- load controller_list -->
  <rosparam file="$(find notfDrone)/config/controllers.yaml"/>
</launch>
```

Listing A.4: Example launch file for the sensor manager in MoveIT

```
void translator()
{
  if(path.joint_name == Base && path.waypoints > 0)
  {
    for( point in path.waypoints)
    {
```

```
7         voPosition = tf.lookup("/odom", "/camera_link") // looks
              up the camera position
8         currentDirection = voPosition.getRotation()
9         desiredDirection = getDirection(voPosition.getPosition(),
               point.getPosition())
10        angle = getAngleBetweenAxis(currentDirection,
              desiredDirection) //gets the yaw between 2 vectors
11
12        publishRotation(angle)
13
14        distance = getHorizontalDistance(voPosition.getPosition()
              , point.getPosition()) //gets the distance of the x-
              and y-components
15        height = voPosition.getPosition().z() - point.getPosition
              ().z()
16
17        publishTranslation(distance, height)
18      }
19    }
20  }
21
22  void publishRotation(angle)
23  {
24    notDone = true;
25    sentAngle = angle;
26    sign = ((angle < 0 )? -1:1);
27    maxAngle = sign * 1.570796327f ;
28
29    /*publish rotation command in radians/sec*/
30    while(notDone)
31    {
32      if(abs(angle) > abs(maxAngle))
```

```
33      {
34        sentAngle = maxAngle;
35        angle = angle - maxAngle;
36      }
37      else
38      {
39        notDone = false;
40        sentAngle = angle;
41      }
42      cmd.angular.x = cmd.angular.y = cmd.angular.z = 0;
43      cmd.angular.z = sentAngle;
44
45      pub_topic.publish(cmd);
46      sleep(1.0);
47    }
48  }
49
50  void publishTranslation(distance, height)
51  {
52    /*publish translation command, by building up Twist message
          cmd*/
53    cmd.linear.x = distance
54    cmd.linear.y = 0
55    cmd.linear.z = height
56    cmd.angular.z = 0; //rotations are not necessary at this
          point
57
58    pub_topic.publish(cmd)
59    sleep(1.0)
60
61    /*stop the movement*/
```

```
62    cmd.linear.x = cmd.linear.y = cmd.linear.z = cmd.angular.z =
          0
63    pub_topic.publish(cmd) //publish the Twist message to cmd_vel
64  }
```

Listing A.5: Pseudo-code for the actionController. It enforces 4 DOF and translates the commands from waypoints to velocities

Auteursrechtelijke overeenkomst

Ik/wij verlenen het wereldwijde auteursrecht voor de ingediende eindverhandeling:
Investigating the use of UAVs in combination with low-cost sensors for human-UAV interaction

Richting: **master in de informatica-multimedia**
Jaar: **2014**

in alle mogelijke mediaformaten, - bestaande en in de toekomst te ontwikkelen - , aan de Universiteit Hasselt.

Niet tegenstaand deze toekenning van het auteursrecht aan de Universiteit Hasselt behoud ik als auteur het recht om de eindverhandeling, - in zijn geheel of gedeeltelijk -, vrij te reproduceren, (her)publiceren of distribueren zonder de toelating te moeten verkrijgen van de Universiteit Hasselt.

Ik bevestig dat de eindverhandeling mijn origineel werk is, en dat ik het recht heb om de rechten te verlenen die in deze overeenkomst worden beschreven. Ik verklaar tevens dat de eindverhandeling, naar mijn weten, het auteursrecht van anderen niet overtreedt.

Ik verklaar tevens dat ik voor het materiaal in de eindverhandeling dat beschermd wordt door het auteursrecht, de nodige toelatingen heb verkregen zodat ik deze ook aan de Universiteit Hasselt kan overdragen en dat dit duidelijk in de tekst en inhoud van de eindverhandeling werd genotificeerd.

Universiteit Hasselt zal mij als auteur(s) van de eindverhandeling identificeren en zal geen wijzigingen aanbrengen aan de eindverhandeling, uitgezonderd deze toegelaten door deze overeenkomst.

Voor akkoord,

Jorissen, Ben

Datum: **25/06/2014**