Indoor swarm exploration with Pocket Drones

McGuire, Kimberly

DOI
10.4233/uuid:48ed7edc-934e-4dfc-b35c-fe04d55caee1

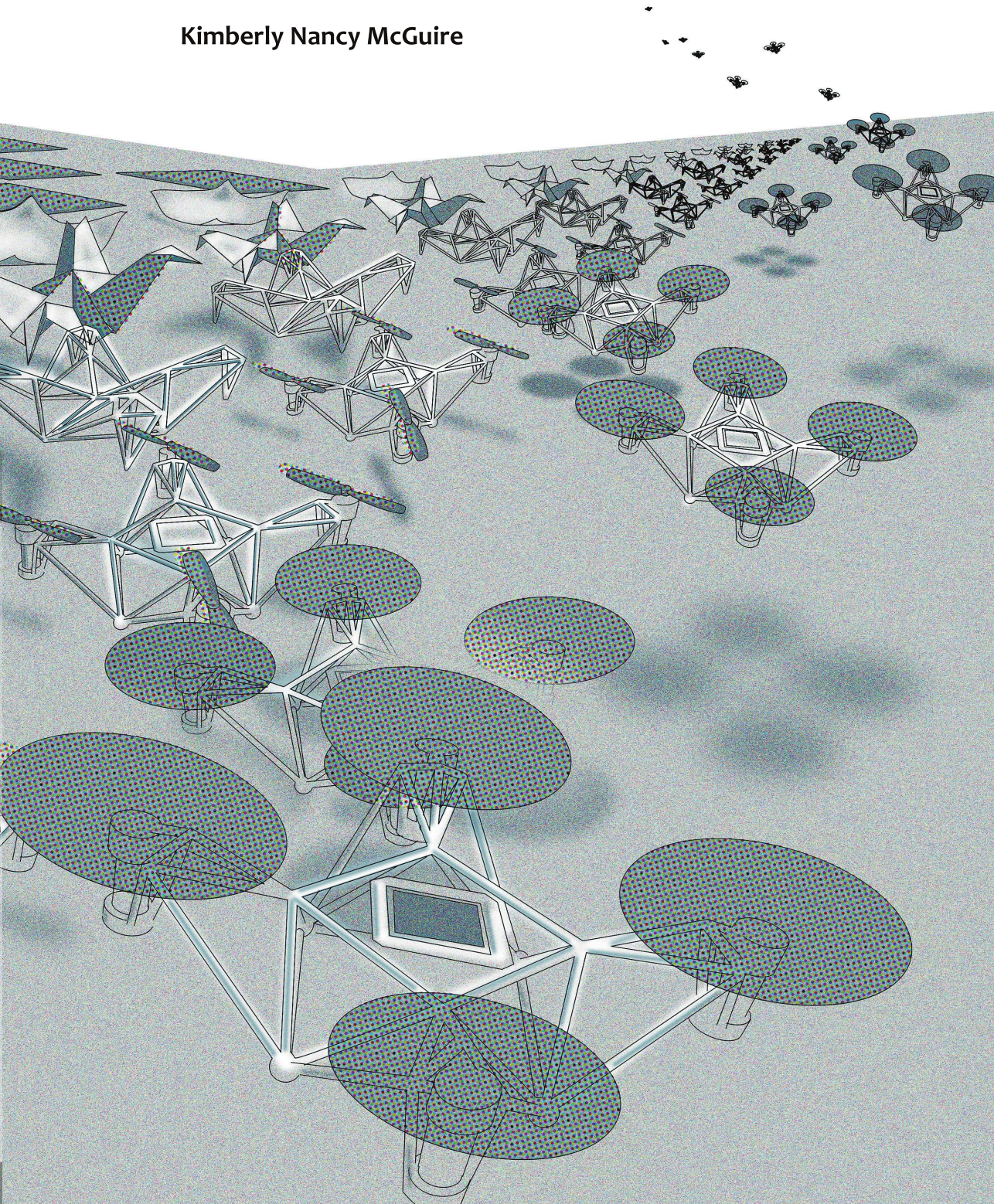
Publication date
2019

Document Version
Final published version

Citation (APA)
McGuire, K. (2019). *Indoor swarm exploration with Pocket Drones*. [Dissertation (TU Delft), Delft University of Technology]. https://doi.org/10.4233/uuid:48ed7edc-934e-4dfc-b35c-fe04d55caee1

Indoor Swarm Exploration with Pocket Drones

Kimberly Nancy McGuire



INDOOR SWARM EXPLORATION WITH POCKET DRONES

INDOOR SWARM EXPLORATION WITH POCKET DRONES

Proefschrift

ter verkrijging van de graad van doctor
aan de Technische Universiteit Delft,
op gezag van de Rector Magnificus prof. dr. ir. T.H.J.J. van der Hagen,
voorzitter van het College voor Promoties,
in het openbaar te verdedigen op *donderdag 14 november 2019 om 10:00 uur*

door

Kimberly Nancy MCGUIRE

Master of Science in Mechanical Engineering,
Technische Universiteit Delft, Delft, Nederland
geboren te Heemstede, Nederland.

Dit proefschrift is goedgekeurd door de

promotor: Dr. G.C.H.E. de Croon
promotor: Prof. dr. K.P. Tuyls
promotor: Prof. dr. H.J. Kappen

Samenstelling promotiecommissie:

Rector Magnificus,	voorzitter
Dr. G.C.H.E. de Croon,	Technische Universiteit Delft
Prof. dr. K.P. Tuyls,	University of Liverpool
Prof. dr. H.J. Kappen,	Radboud Universiteit Nijmegen
Onafhankelijke leden:	
Prof. dr. ir. P.P. Jonker	Technische Universiteit Delft
Prof. dr. P. Campoy	Universidad Politécnica Madrid, Spain
Dr. V. Trianni	Italian National Research Council (CNR), Italy
Dr. S. Viollet	Aix-Marseille Université, France
Prof. dr. ir. M. Mulder (reserve)	Technische Universiteit Delft

Keywords: micro aerial vehicles, swarm robotics, autonomous navigation, pocket drones, optical flow, stereo vision, bug algorithms

Printed by: Off page, www.offpage.nl

Front & Back: K. N. McGuire

ISBN 978-94-6182-976-4

An electronic version of this dissertation is available at
http://repository.tudelft.nl/.

CONTENTS

SUMMARY

SWARMS of tiny Micro Aerial Vehicles (MAVs) are widely sought after in both research and industry. Pocket drones, which fit on the palm of your hand, are small, agile and inherently safe. This makes them suitable for several surveillance tasks such as search and rescue, green-house monitoring and pipe-line inspection. In order for a more efficient search, a swarm of pocket drones would be ideal to explore these types of areas faster. For these exploration tasks in GPS-deprived environments, communication signals over a large distance will be of low quality due to disturbance and interference, therefore the pocket drones cannot make use of human pilots and/or an external computer that is able to choreograph their every movement. Localization systems like GPS or motion-capture systems will not be available in these missions. The swarm of pocket drones must be completely autonomous and only use their on-board sensing and processing capabilities. Current methods and techniques like Simultaneous Localization and Mapping (SLAM), will not be suitable due to their extensive requirements for the platform's computational capabilities and memory storage. This dissertation will therefore focus on designing a new strategy for a swarm of pocket drones for both low-level and high-level navigation in an indoor environment.

The first part of this dissertation focuses on the low-level-navigation capabilities of the swarm of pocket drones, by first looking at the individual. We developed Edge-Flow, which was able to run on-board an STM32F4-based stereo-camera for the detection of optical flow. A pocket drone with a downward-facing stereo-camera was able to estimate its own velocity. It was able to use the measurement directly in its own velocity-based control loop and was able to match it with externally given velocity commands. We extended Edge-Flow to Edge-FS (Flow & Stereo), which enabled the stereo-camera to detect both obstacles and the drone's velocity at the same time. This was implemented on a pocket drone with a forward-facing stereo-camera and we showed its capabilities in a typical office-like room. Here it was able to detect and control its ego-motion and avoid obstacles with a simple finite state machine, and therefore was able to fly autonomously.

A further necessity for swarm operations is for multiple pocket drones to avoid each other. An on-board relative localization scheme based on the Received Signal Strength Intensity (RSSI) of the inter-drone communication was developed to make this possible. Two pocket drones with a downward-facing stereo-camera were communicating with each other by means of Bluetooth and by fusing the RSSI with their velocity (estimated by Edge-Flow), they were able to estimate each other's relative position and perform inter-drone avoidance. This on-board localization scheme was combined with the capabilities of Edge-FS. Two pocket drones with a forward-facing stereo-camera now were able to detect static obstacles and detect each other's position at the same time. Both obstacles and drones were added to a collision disk on-board each drone, which indicated the safe directions to go to. With this, two pocket drones were able to fly together in a room while avoiding the walls and each other.

The second part of this dissertation focuses on high-level-navigation. Since conventional navigation strategies cannot fit on-board the pocket drones, we investigated an alternative method: bug algorithms. This type of navigation does not require a map and uses very little memory. The robot has a general goal to navigate towards and once it hits an obstacle, it will follow its boundary until the path towards the goal is clear again. We present a literature survey of the existing techniques and evaluation on their suitability for deployment for real-world scenarios. We tested a selection of bug algorithms in a simulation, which reenacted different types of on-board sensor values. Here we found that bug-algorithms over-relied on a perfect positioning system. With increasing sensor errors and estimation drift, all existing bug algorithms' performances decreased. This provided us valuable insights for the design of a novel bug algorithm for high-level-navigation.

Finally, we developed and demonstrated a bug-algorithm-based navigation strategy for multiple pocket drones for indoor exploration and homing. We named this technique the swarm gradient bug algorithm (SGBA) and it enabled the pocket drones to explore a floor of an inside building and return to its original position by the RSSI-gradient of a radio beacon. Once two pocket drones come into each other's proximity, one will avoid the other and coordinate its own preferred search direction based on the information it has received (from the other). On the 11th floor of the high-rise building of the Aerospace faculty of TU Delft, up to 6 pocket drones explored the area and returned to their initial position. This is the first time that such a complex task has been performed by a swarm of MAVs of this size. Moreover, we showed an application experiment of a "victim" search task, where 4 pocket drones equipped with cameras searched and found two colored wooden figures in the rooms.

We were able to achieve the main objective of this dissertation with SGBA on the pocket drones. The current solution still relies on a beacon, albeit only at the base station and not for positioning. Future work should study navigation methods that do not rely on any external elements. Throughout this dissertation, an important lesson we learned is that more capabilities inevitably require more sensors / processing / energy, which with current technology translates to less flight time. Future researchers into this topic should be aware of this important trade-off for future implementations on pocket drones.

SAMENVATTING

ZWERMEN van kleine Micro Aerial Vehicles (MAV's, microluchtvaartuigen) zijn zeer gewild in zowel academia als industrie. Pocket drones, die op een handpalm passen, zijn snel, behendig en inherent veilig. Dit maakt ze geschikt voor verschillende surveillancetaken zoals opsporings- en reddingstaken, het monitoren van kassen en inspectie van pijpleidingen. Een zwerm pocket drones zou ideaal zijn om dit soort gebieden sneller en efficiënter te verkennen. Bij dit soort verkenningstaken in GPS-arme gebieden zijn communicatiesignalen die over lange afstanden worden verstuurd van lage kwaliteit door verstoringen en interferentie. Hierdoor kunnen de pocket drones geen gebruik maken van een menselijke piloot en/of externe computer die controle heeft over elke beweging. Lokaliseringstechnieken zoals GPS of motion-capturesystemen zijn niet beschikbaar tijdens dit soort missies. De zwerm pocket drones moet volledig autonoom zijn en kan enkel gebruik maken van de sensoren en rekenkracht die aan boord worden meegenomen. Huidige methodes en technieken zoals Simultaneous Localization and Mapping (SLAM, het gelijktijdig lokaliseren en maken van een kaart) zijn niet geschikt door hun vereisten voor extensieve hoeveelheden rekenkracht en geheugenopslag. Dit proefschrift zal zich daarom focussen op het ontwikkelen van een nieuwe strategie voor pocket drones voor navigatie binnenshuis op zowel laag als hoog hiërarchisch niveau.

Het eerste deel van dit proefschrift focust op navigatiecapaciteiten van de zwerm pocket drones op laag niveau door eerst te kijken naar het individu. We hebben Edge-Flow ontwikkeld, een algoritme voor het detecteren van optische stroom (het bewegen van objecten in een beeld) dat kan draaien op een STM32F4 gebaseerde stereocamera. Een pocket drone met een naar beneden gerichte stereocamera was in staat zijn eigen snelheid te schatten. Deze meting kon direct worden gebruikt als invoer in zijn eigen snelheidsgebaseerde regelkring en de drone was in staat deze overeen te laten komen met extern ingevoerde snelheidscommando's. We hebben Edge-Flow uitgebreid naar Edge-FS (Flow & Stereo), waarmee de stereocamera zowel obstakels kon detecteren als de snelheid van de drone kon bepalen. Dit was geïmplementeerd op een pocket drone met een naar voren gerichte stereocamera en we hebben diens capaciteiten gedemonstreerd in een typische kantooromgeving. Hier was de drone in staat zijn eigen beweging zowel te detecteren als te controleren. Hiernaast konden obstakels worden ontweken met een eenvoudige eindigetoestandsautomaat, waardoor de drone autonoom kon vliegen.

Een andere voorwaarde voor de operatie van een zwerm is de capaciteit van meerdere drones om elkaar te ontwijken. Om dit mogelijk te maken werd een relatief lokalisatieconcept ontwikkeld, gebaseerd op Received Signal Strength Intensity (RSSI, de intensiteit van het ontvangen signaal) van interdronecommunicatie. Twee pocket drones met een naar beneden gerichte stereocamera communiceerden met elkaar via Bluetooth. Door de RSSI te combineren met hun snelheid (geschat door Edge-Flow), waren ze in staat elkaars relatieve positie te schatten en elkaar te ontwijken. Dit lokalisatiecon-

cept, dat aan boord kan worden verwerkt, werd gecombineerd met de mogelijkheden van Edge-FS. Twee pocket drones met een naar voren gerichte stereocamera waren nu in staat gelijktijdig statische obstakels alsmede elkaars positie te detecteren. Zowel obstakels als andere drones werden toegevoegd aan een mogelijke-botsingsveld aan boord van elke drone, waarmee richtingen die veilig waren om heen te vliegen werden aangeduid. Met dit systeem waren twee pocket drones in staat om samen in een kamer te vliegen terwijl zij de muren en elkaar konden vermijden.

Het tweede deel van dit proefschrift focust op navigatie op hoog niveau. Daar conventionele navigatiestrategieën niet aan boord van pocket drones kunnen worden verwerkt, onderzochten wij een alternatieve methode: bug-algoritmes. Dit type navigatie is niet afhankelijk van een kaart en vraagt zeer weinig van het geheugen. De robot navigeert naar een algemeen doel en zodra het een obstakel treft volgt het diens randen tot het pad richting het doel niet langer geblokkeerd is. We presenteren een literatuurstudie van de bestaande technieken en een evaluatie van hun mogelijke geschiktheid voor toepassing in de echte wereld. We testten een selectie van bug-algoritmes in een simulatie waarmee verschillende types sensordata van boordsystemen konden worden geëvalueerd. Wij vonden dat bug-algoritmes teveel afhangen van een perfect positioneringssysteem. Met toenemende sensorfouten en meer afwijkende schattingen nam de prestatie van alle bug-algoritmes af. Dit gaf ons waardevolle inzichten voor het ontwikkelen van een nieuw bug-algoritme voor navigatie op hoog niveau.

Tenslotte hebben we een navigatiestrategie ontwikkeld en gedemonstreerd voor meerdere pocket drones voor verkenning en terugkeren binnen een gebouw, gebaseerd op een bug-algoritme. We noemen deze techniek Swarm Gradient Bug Algorithm (SGBA, zwerm gradiënt insect-algoritme) en het stelt de pocket drones in staat een verdieping van een gebouw te verkennen en terug te keren naar hun originele positie door de RSSI-gradiënt van een radiobaken. Zodra twee pocket drones in elkaars nabijheid kwamen, ontwijkt één drone de andere, waarna hij een eigen geprefereerde zoekrichting coördineerde op basis van de informatie die hij ontvangen heeft van de andere drone. Tot 6 drones verkenden een gebied op de 11e verdieping van de faculteit Aerospace Engineering van de Technische Universiteit Delft waarna zij terugkeerden naar hun initiële positie. Dit is de eerste keer dat zo'n complexe taak is uitgevoerd door een zwerm MAV's van dit formaat. Verder presenteren we een experiment waarin een toepassing wordt bestuurd om "slachtoffers"te zoeken, waarbij 4 pocket drones uitgerust met camera's zochten naar twee houten, gekleurde figuren op de verdieping en deze wisten te lokaliseren.

We zijn er in geslaagd het hoofddoel van dit proefschrift te bereiken met SGBA op de pocket drones. De huidige oplossing is nog altijd afhankelijk van een baken, ofschoon enkel bij het basisstation en niet voor positionering. Toekomstig werk zou zich moeten focussen op navigatiemethodes die onafhankelijk zijn van externe hulpmiddelen. Een belangrijke les die als rode draad door dit proefschrift loopt is dat meer mogelijkheden onvermijdelijk meer sensoren/rekenkracht/energie vereisen, wat zich met de huidige techniek vertaalt in kortere vliegtijd. Toekomstige onderzoekers in dit veld moeten zich bewust zijn van deze belangrijke afweging voor toekomstige implementaties op pocket drones.

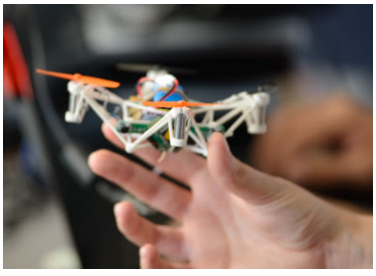
1

INTRODUCTION

HONEYBEES are one of nature's most fascinating creatures. Even though they barely fit on the tip of your finger and only have approximately 960,000 neurons in their brain (Menzel and Giurfa, 2001), a single bee can explore a large field of flowers (Capaldi et al., 2000). Yet, to search through an entire field by itself will take a long time. The bee's real strength lies in the collective, as a swarm can coordinate the exploration more efficiently (Seeley, 2009). In the hive, individuals will point out the most profitable flower patches to their fellow bees, by means of a waggle dance that most likely indicates the direction, range and significance of various food-sources (Menzel and Greggers, 2015, Reinhard and Srinivasan, 2009, Veeraraghavan et al., 2008). The most vividly recommended nectar-source will most likely be revisited by the observing worker bees (Menzel et al., 2012). With these skills, the swarm can search the entire field faster than any single bee ever could (Milius, 2009).

One can imagine how useful a swarm of small Micro Air Vehicles (MAVs) could be within society (Şahin, 2005, Yang et al., 2018). In this dissertation, we want to use such

(a) Ladybird frame with a Lisa-S autopilot



(b) 3D printed frame with a Lisa-MXs autopilot

Figure 1.1: Examples of the pocket drones with a 4 g stereo-camera used in this dissertation.

1

(a) Photoshopped search-and-rescue scenario

(b) Photoshopped greenhouse monitoring scenario.

Figure 1.2: Example of applications for pocket drones.

a swarm of small MAVs for the exploration of indoor environments. To achieve this, we conducted experiments with pocket drones, which are tiny MAVs that fit in the palm of your hand (Fig. 1.1). Their small size makes them able to pass through small windows or holes (Mulgaonkar et al., 2015). For instance, as in Fig. 1.2a, they can explore a collapsed building, find missing people, and explore a structure's instability in search-and-rescues scenarios (Tomic et al., 2012), which would prevent a human search-team to put their lives unnecessarily at risk. Pocket drones would be valuable for greenhouse monitoring as well (Primicerio et al., 2012), as their tiny size would make them unlikely to hurt any of the plants or the workers walking underneath them (Fig. 1.2b), while nobody would feel comfortable if a 3-kilo Unmanned Aerial Vehicle (UAV) with large rotor blades would be flying over their heads (McHenry, 2004).

In the aforementioned examples of using pocket drones, many practical difficulties are involved (Elbanhawi et al., 2017). Controlling them directly over long distances by a human pilot is challenging, as radio signals will interfere with other radio-sources and/or interrupted by the materials of the structure (Hashemi, 1993). This indicates that the pocket drones would need a great deal of autonomy to be able to navigate by themselves without any external control. As many indoor application environments are GPS-deprived, their exact location will be unknown (Nirjon et al., 2014). A single pocket drone should still be capable to explore an indoor building without knowing its current position. Moreover, it would need to handle the unknown obstacles it encounters as well. While exploring, it can collect information that will be useful for the completion of its mission (Toth and Jóźków, 2016). However, transmitting this information back to a home-station, especially bandwidth-demanding videos, is very challenging for a small platform (Dunkley et al., 2014, Elbanhawi et al., 2017). A pocket drone should spend all its energy on flying autonomously, not in compressing and transmitting data over large distances. Therefore, it is necessary to return to the home position to deliver the necessary information to the end-user. However, a single pocket drone will not be able to cover much ground all by itself. If multiple MAVs were deployed, they could cover a larger area in a shorter amount of time (Brambilla et al., 2013). If they meet other pocket drones

on the way, while sensing and avoiding each other, they can also coordinate their search based on the information they share.

The examples that nature has provided us, such as the earlier mentioned honeybees, suggest that there are navigation strategies simple enough to fit on a pocket drone (Collett and Collett, 2006, Collett et al., 2013). The unfortunate truth is that currently, there is no method or technique that can be fully implemented on-board a tiny, resource-limited, flying platform. The biggest problem is that traditional techniques are requesting that an MAV can carry a considerable amount of computing power. One such technique, called Simultaneous Localization and Mapping (SLAM) (Durrant-Whyte and Bailey, 2006), starts off by building a high-resolution 3D map, then trying to localize the robot within it and finally setting out a path for it to follow (Fuentes-Pacheco et al., 2012). Within a swarm, information can be shared with the others in order to coordinate the exploration, which can consist of their map or current positions (Weinstein et al., 2018). This calls for a platform capable of carrying a large on-board computer, high-end sensors and antennas, resulting in a big, crude and most importantly, unsafe MAV. Although using an external computer seems as a logical step to aid the limited processing capabilities of tiny MAVs like pocket drones, like in (Dunkley et al., 2014), it does also impose limitations on their operation space. The communication link must be strong, which means that the MAV needs to always be in close proximity. For indoor exploration, the signal will deteriorate once the distance increases, due to the scattering and interference by the walls and obstacles in between (Hashemi, 1993). The pocket drones cannot rely on the external computer anymore and must do all the processing themselves. Moreover, this independence would also improve the possible scalability of the swarm, since the communication bandwidth will not be the bottle neck for the number of MAVs (Nunnally et al., 2012).

This dissertation will take a novel approach to solve autonomous indoor navigation of tiny pocket drones, which can also be considered for other non-flying platforms. We will not wait for the on-board computers to become more powerful to implement SLAM techniques or any of its equivalents, but we will instead make the navigational package as simple and efficient as possible. SLAM does provide more navigational freedom, as it enables MAVs to go from any point to another point in the map. However, we are focusing on the main principles we believe are fundamental for indoor navigation, which are exploration and the ability to move back to a home-location. This gives us the opportunity to develop extremely computational efficient methods for navigation and to design strategies to enable multiple pocket drones to explore an indoor environment.

1.1. CHALLENGES AND PREVIOUS RESEARCH

This section will discuss the requirements for autonomous navigation with a swarm of pocket drones and show what has been already done in the field. Fig. 1.3 shows a visualization with the necessities that needs to be fulfilled. First of all, the drones would need a good position and/or velocity estimate in order to hover in place and react on velocity commands. Quadrotors, like the pocket drones, are inherently unstable and can easily drift away and collide with obstacles if their velocity is not estimated. It would be quite challenging to do high-level-navigation without these building blocks properly taken care off.

After achieving stable velocity and/or position control, the pocket drones should be capable to avoid the obstacles and each other. Here is also where the sensors becomes important for observing the environment and the other (flying) robots around it. In order to take the leap to high-level-navigation, one has to be sure that the drone has an efficient and robust collision avoidance strategy, since it is much more difficult for a quad-copter to recover from a collision than a ground-bound wheeled robot. The pocket drone should now be able to move out, fly within an environment and should be able to return to its initial location. The next logical step is be able to navigate to any pre-visited location, which requires the platform to make some kind of environment representation during exploration. The latter is not in the scope of this dissertation, as we would first need to solve the behaviors that comes before, however is part of the necessary future work.

The following sections will go into the sub-modules and present the current state-of-the-art research. We will evaluate the research done in this field on their suitability for implementation on pocket drones (< 50 grams, <168 MHz), and therefore will mostly focus on the hardware requirements, such as weight, processor speed and if they are relying on any external systems as a positioning or processing aid.

1.1.1. VELOCITY CONTROL AND OBSTACLE AVOIDANCE

A MAV, especially quadrotors, should be able to hover in place, react to velocity comments, and avoid obstacles. In order to do that, it should be able to estimate its own velocity and/or position and detect the objects surrounding it. The localization of MAVs can easily be estimated with the help of GPS or a motion capture system (MCS), however there have been those who tried to make their platforms more independent from external position systems. For example the work of Kendoul et al. (2009a), where they implemented velocity estimation and control on an X-3D-BL MAV (53cm ⌀, 650 g) with a downward-looking camera. They used an external computer to receive the video-stream and to calculate optical flow at 10 Hz. This was sent back and fused on-board on a Gumstix autopilot (400 MHz, 64 MB RAM). For an example of an implementation in the weight-class we address in this dissertation, we found that Dunkley et al. (2014) used a 26-gram Crazyflie 1.0 (72 MHz, 20KB RAM) with a forward-looking micro PAL camera system to achieve visual-inertial guidance control. The video was streamed to an external computer as well; however, in comparison with Kendoul et al. (2009a), all processing to retrieve the position and velocity estimates was done off-board. Here they reported a delay of 40 ms to receive and to pre-process the video footage; nonetheless, the Crazyflie had a reported drift-free hover.

For indoor exploration, it is essential that the processing of velocity and position estimations is computed all on-board the MAV. It might be an option to choose dedicated software and hardware modules, such as with a camera/sonar combination of the PX4Flow deck (Honegger et al., 2013) (not suitable for pocket drones), the Crazyflie's flowdeck [1] or an ventral optic flow sensor inspired by the facet eyes of insects (Ruffier and Franceschini, 2015). Briod et al. (2013) implemented four ADNS-9500 optical mouse sensors on a 46-gram quadrotor. The optical mouse sensors were designed to detect optical flow and were placed facing four different directions on the MAV. With this setup, the

[1] Bitcraze AB. Flow-deck expansion deck, https://www.bitcraze.io/flow-deck-v2/

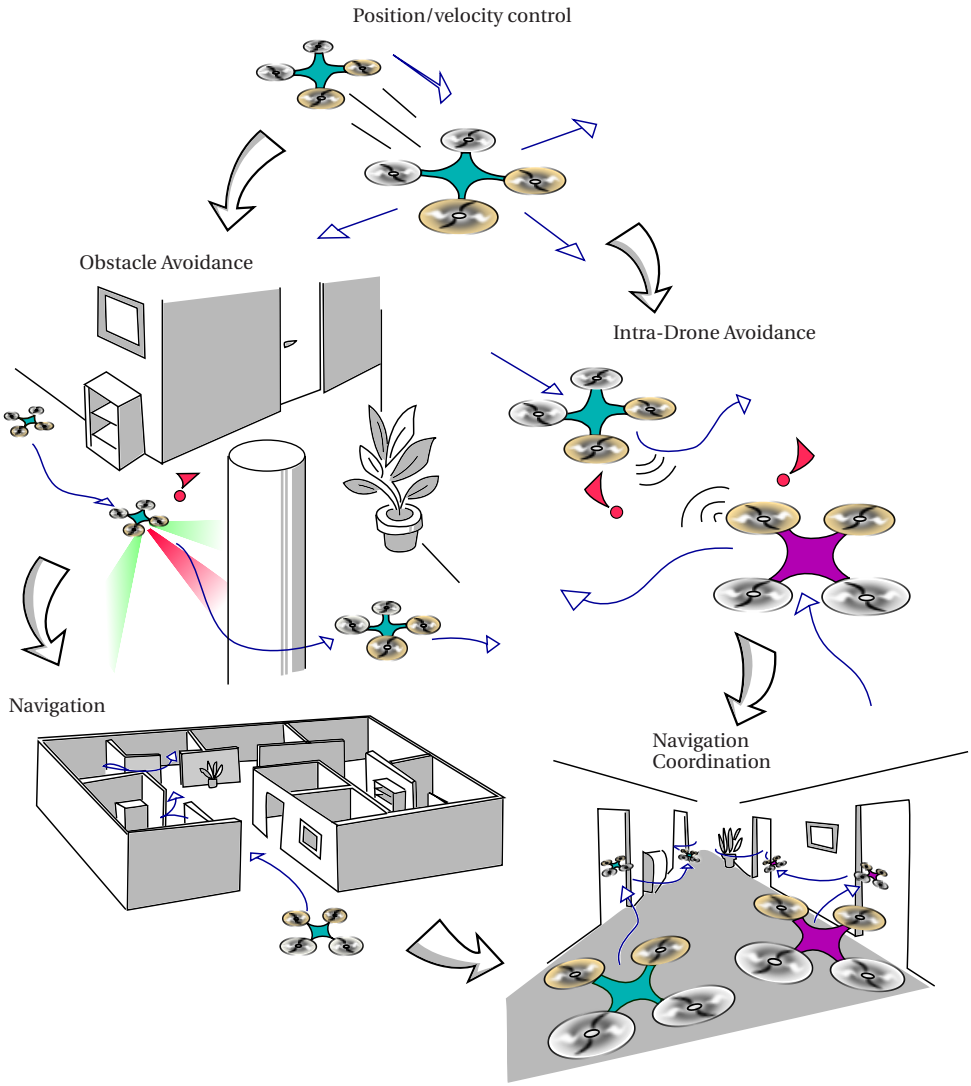


Figure 1.3: Visualization of necessary blocks to achieve high level navigation on resource-limited tiny flying robots.

drone was able to estimate its ego-motion and used it directly within the control loop, all on-board the stm32 processor (120MHz, 16KB RAM). It saved additional processing by receiving the optical flow directly from the sensors; however, to make the velocity estimation observable, the quadrotor had to constantly move in an oscillatory fashion. Therefore, occasional high-level commands were needed to move the small drone away from obstacles, as the hover was not perfect due to these oscillations. The performance improved by adding 4 more optical flow sensors, but the final platform ended up weighing 268 g (Briod et al., 2016). Moore et al. (2014) developed a 2-gram omnidirectional camera system to also do ego-motion estimation in the loop. A Blade MCX21 mini helicopter carried it while flying, totaling the weight to only 30 g. This time, the optical flow was computed from the camera sensors on-board an Atmel AT32UC3B1256 microprocessor (60MHz, 32KB RAM) at a rate of 10 Hz, while the control loop of the mini helicopter existed on a second, parallel, Atmel processor. The present motion-capture-system was used only to measure and control the height; nonetheless, there was still noticeable drift of the micro helicopter while it was flying through the test-environment.

The works of Moore et al. (2014) and Briod et al. (2016) have expressed the desire to also integrate obstacle avoidance into their system – the next step in Fig. 1.3–, but unfortunately no follow-up of the work has been found. There have been bigger MAVs that use stereo-vision based systems for this purpose, like the reactive obstacle avoiding method of Oleynikova et al. (2015) and Matthies et al. (2014). However, there is just a handful of methods that has actually been implemented on tiny MAVs in the same weight-class as in this dissertation. The 20-gram Delfly in De Wagter et al. (2014) and Tijmons et al. (2017) uses a 4-gram stereo-camera to detect and avoid obstacle indoors with solely on-board sensing and processing. The Delfly is an inherently stable platform, therefore it can skip the step of velocity-estimation and -control in Fig. 1.3. Unfortunately, rotor-based MAVs like the pocket drone are inherently unstable and therefore need the help of velocity estimates to function. However, flapping wing MAVs are still limited by a very small payload, which is less the case for quadrotors.

Within low-level-navigation, this section has shown examples of on-board implementations of the same weight-class as the pocket drones. Unfortunately, the platforms that can estimate their own ego-motion and do velocity control in Briod et al. (2016) and Moore et al. (2014) have not reached the level of robustness necessary for stable flight and have not incorporated any obstacle avoidance. On the other hand, the research with the Delfly did focus on obstacle avoidance but did not have the need of any velocity estimation. The challenge in this area is to incorporate both robust velocity estimation and obstacle avoidance on a pocket drone, solely using its on-board sensing and processing capabilities.

1.1.2. SINGLE MAV HIGH-LEVEL-NAVIGATION

There is a lot of research covering high-level-navigation strategies for single MAVs. However, there is one common problem: a large portion of them weigh more than 1 kg. The majority uses the conventional SLAM techniques to perform navigation, which requires a significant amount processing and memory capabilities. In Bachrach et al. (2009) and Grzonka et al. (2012) for instance, they used a 1-2 kilo quadcopter with a Hokuyu laser range scanner, but still needed the help of an external computer to do the computa-

tion of the estimation and mapping. Achtelik et al. (2011) used a 2 kg Astec Pelican with an Intel Atom computer (1.6 Ghz, 1GB RAM), with the same laser-scanner and a downward-looking camera. This time, all the SLAM processing was done on-board, with some high-level commands given by an external computer, which they demonstrated in a stabilization task. Shen et al. (2011) were able, as one of the first, to deploy their drone to do multi-floor mapping of an inside building with all on-board processing (using a similar platform as in Achtelik et al. (2011)), with way-points given beforehand to aid the navigation. A more recent example is the platform presented in Mohta et al. (2018), with an Intel-i7 based computer (3.1 Ghz, 16 GB RAM), which only needed a goal position and was able to do all the localization and mapping fully on-board. The total weight of the platform was just shy of 3 kilos, which is a good indication of the extensive computer-power it needed to carry.

While in the last section some low-level-navigation examples could be found of platforms weighing less than 50 grams, for single-drone high-level-navigation almost none can be found at all. The smallest flying platform that was able to do multiple room exploration was in the work of Scheper et al. (2018). The Delfly, equipped with the 4-gram stereo-camera, was able to avoid the obstacles as in Tijmons et al. (2017), recognized open doors and navigated through them, all fully on-board the stm32F4 processor (168MHz, 192kB RAM). Although no 3D mapping was involved, it indicates that a combination of simple behaviors should be enough to go from location A to B. The same parallels can be found in models describing insect navigation, which inspire new types of robotic navigation as well. Experiments with real bees can already show interesting properties, like that they just use a direction and a distance to describe the path towards the flowers (Menzel et al., 2012) and that they use optical flow to monitor their distance to a food source (Srinivasan et al., 1997). Cartwright and Collett (1983) takes it one step further and suggests that bees could also recognize landmarks around them and save environment "snap-shots" at significant locations.

In essence, if 3D-SLAM is not required for the actual mission, the first step of navigation can be seen as the exploration phase, where the robot follows a certain strategy to get familiar with the environment. The next step would be homing, where the platform is able to return to the home location based on the representation of the environment it made in the exploration phase. To illustrate an example with a non-flying platform, Stelzer et al. (2018) deployed a robot that was able to (manually) traverse an outdoor urban environment, save the snapshots of its surroundings, and was able to get back to its initial location by creating homing-vectors from the difference between its current view and its saved view. Denuelle and Srinivasan (2016) showed the snap-shot principle on a Microkopter platform (2 kg) with a Intel NUC computer (2-core 2.6 GHz). However, both Stelzer et al. (2018) and Denuelle and Srinivasan (2016) do require quite some processing and memory, which is still beyond the reach of the pocket drones used in the experiments of this dissertation.

Full on-board implementations of navigation of single MAVs currently exist only on platforms that weigh at least a few kilos and are not suitable for a pocket drone. Bio-inspired strategies such as the snapshot model are promising options, however, still require full images to be stored. To enable exploration and homing on a pocket drone, we need to find even more efficient methods.

1.1.3. MULTIPLE ROBOT LOCALIZATION AND COORDINATION

Eventually we would like to design exploration strategies for not a single, but a swarm of autonomous pocket drones. We will therefore need to address the work done in the field of multi-MAV navigation. First of all, the MAVs should be able to avoid each other, as it has been depicted in Fig. 1.3. In order to do this, the drones would need to be able to sense each other's presence. One way is to share the GPS-coordinates. Duarte et al. (2016) did this with a swarm of aquatic robots, which were able to do all kinds of swarming behaviors, such as homing, dispersion, clustering and area monitoring. Vásárhelyi et al. (2018) showed a decentralized flocking method with 30 Mikrokopters (1kg) within a 200x200-meter outdoor area, equipped with an extra Odroid C1+ (2-core 1.5Ghz, 1GB RAM) for processing. Here they communicated directly with each other through a WiFi protocol; however, the MAVs did experience package-loss when they were further apart. The separation was necessary due to the 3-4 meters inaccuracy of the GPS position measurements. A more accurate method is to use an MCS. In Preiss et al. (2017), 49 Crazyflie 2.0s (33 g each) were able to do an indoor formation flight in a 6x6-meter area with aid of a Vicon MCS, reaching an accuracy of 2 cm. However, here the ground station computer handled all the communication and coordination of the Crazyflies, as they were not directly communicating with each other.

An absolute positioning system cannot be guaranteed for the exploration of an indoor environment, especially in search- and-rescue scenarios. However, there has been work in relative inter-drone localization, where the individual can determine the position of its neighbors using their own on-board sensors. Roelofsen et al. (2015) implemented this on 3 Asctec Hummingbirds (1 kg, 1.6 GHz), each equipped with forward-looking cameras. All the MAVs were augmented with a big orange ball for easy recognition. Unfortunately, they still needed their own position estimate from an MCS and relied on an external computer to do all the vision processing. Saska et al. (2017) deployed a similar system with 3 Mikrokopters (2kg) with a visible circle pattern, a PX4flow sensor and a forward-looking camera. The vision processing and relative localization were all done on-board of their Gumstix autopilot (400MHz, 16MB RAM), but some of the higher-level coordination commands still had to be given by an external computer. Guo et al. (2017) has added an ultra-wide-band (UWB) module on each of its 3 DJI F450's (approx. 1 kilo) with a Beaglebone Black autopilot (1 GhZ, 512Mb RAM), to perform relative localization. However, the drones would need to keep exact track of their own position, which is subject to drift in a GPS-deprived environment.

Since the start of this decade, there has been work on collaborative SLAM techniques, where the MAVs map the environment, communicate and then merge their created maps with each other. With this updated information, they can localize the other drones without the use of an external positioning system. Forster et al. (2013) was one of the first to implement this, with two 1.5 kilo Asctec Fireflies with a downward-looking camera. The MAVs computed their own visual odometry with an embedded Intel CoreDuo processor (1.86 GHz) but were not communicating directly with each other. They had to send the key frames to an external computer to create the individual maps and merge those together in order to retrieve the individual locations. Weinstein et al. (2018) was able to do more on-board with their 12 Qualcomm Snapdragon-based quadcopters (250 gr, 4-core 2 GHz) with a downward-looking camera. However, the central computer still needed

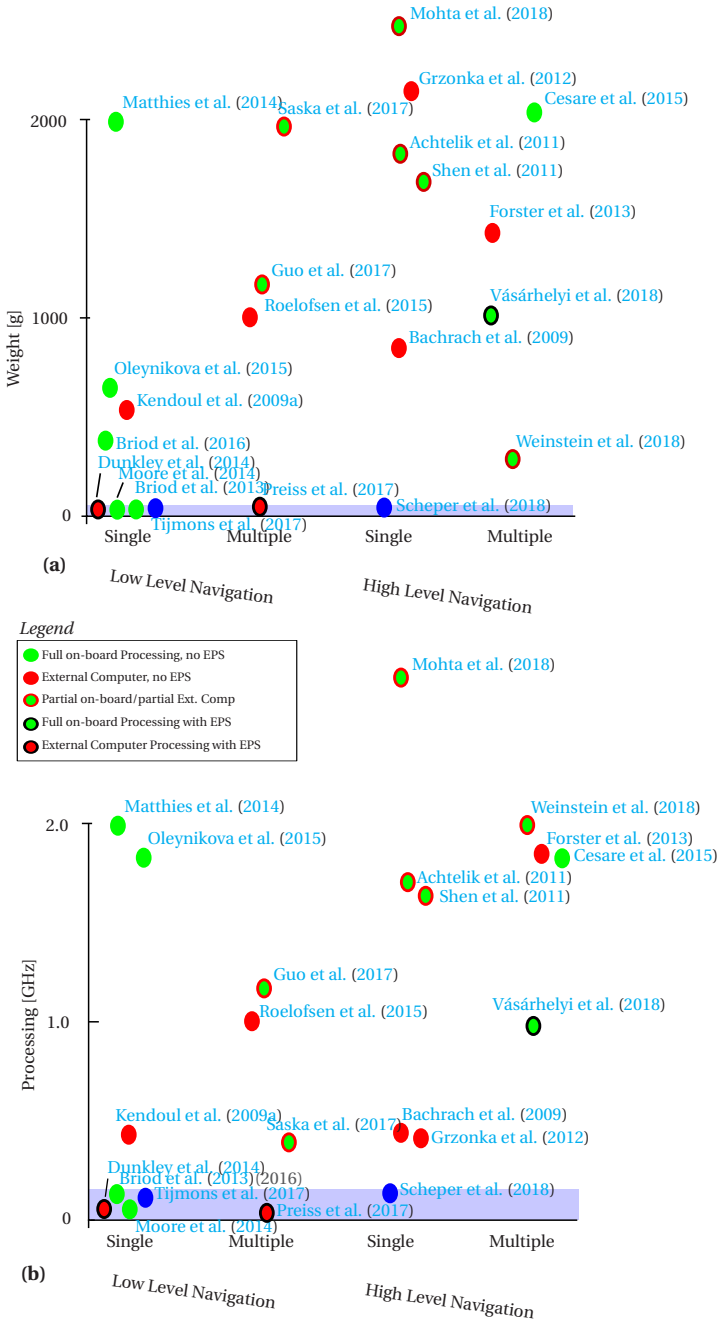


Figure 1.4: Overview of related work. a) shows the weight of the MAVs against the navigational complexity and in b) the processing capabilities against the navigational complexity. The points are color marked if they required any processing help of an external computer or needed an external positioning system (EPS) like GPS or MCS. Works with the Delfly are marked with blue.

to do the planning for them and handled the inter-drone communication. Forster et al. (2013) and Weinstein et al. (2018) were not focusing on any obstacle avoidance with a forward-looking camera, therefore their work is only applicable for open spaces with known or no obstacles.

Until now, multi-MAV navigation has been mostly focused on the inter-drone localization, which some implemented with formation flight, and has not been merged with individual navigational capabilities to explore an unknown environment. More examples can be found for ground-driven robots, for instance the work of Marjovi and Marques (2011). They had a group of iRobot Roomba robots with an Asus EEPC computer perform an odor-source localization task. They developed a graph-like compression of the environment variables, which they enlarged using frontier-based exploration. They were not using a central computer, did not require GPS or MCS, did all the processing themselves and were communicating directly with each other whenever another robot was in range. Cesare et al. (2015) used two custom built quadrotors (1.5 kg, 2-core 1.8 Ghz) for a frontier-based exploration task, while building an Octo-Map of the environment. If one of the MAVs runs out of battery, it will land and act as a relay for the communication with the others, however they did not share the exploration data with each other as they build their maps separately and only covered a single room and a corridor.

Current implementation of multi-MAV exploration of an indoor environment has been distributed over a large set of research, where the focus is mainly on the inter-drone localization and formation/pattern flying. These capabilities exist on MAVs far outside the weight-class and processing capabilities of pocket drones. Moreover, the majority of them still rely on a central computer for processing, high-level monitoring or/and inter-drone communication. The work of Cesare et al. (2015) shows an example of a multi-MAV solution for indoor exploration, however, was only able to show a portion of the mission completion and used 2 kg MAVs. In order to achieve this on the pocket drones, this would need to be built from the ground up, starting with low-level-navigation.

1.1.4. OVERVIEW

Fig. 1.4 shows an overview of the MAVs used in the discussed literature. Fig. 1.4a shows the weight of the used platforms, and Fig. 1.4b shows the processor-speed plotted against the complexity of the navigational task. Note that for some MAVs with 2 on-board autopilots, the one with the highest processing speed would be chosen for the graph. The scatterplot has also been divided in low- and high-level-navigation, and those have been separated in single- and multiple-MAV applications. A blue bar has been added to indicate both the weight-class and the processing capabilities of the pocket drones, to indicate which of the current work falls inside those limitations. We are interested here in implementations on unstable (multi-)rotor vehicles, however the works with the stable Delfly is shown here in blue for comparison. Within the scope of this dissertation, we are looking for solutions that are suitable for collaborative indoor navigation as depicted in Fig. 1.3. Very few of the research for multi-MAV, has implemented the necessary building blocks of low-level-navigation yet, like obstacle and inter-drone avoidance. Moreover, their movement while building the map has been choreographed beforehand. None of these solutions focused on the complete picture: the swarm's ability to avoid physical, unknown, objects, and each other. We therefore consider them only presenting a part of

the solution of our main objective.

It can be noticed that only for the single-MAV-low-level-navigation, work can be found with the same weight- and processor-speed-class as the pocket drone, namely the work of Moore et al. (2014) and Briod et al. (2013). Here they achieved full on-board velocity estimation with a dedicated optical flow sensor and an omnidirectional vision system, however both reported drift and stability issues. Moreover, they have not focused on merging their solution with obstacle detection to avoid crashes, an essential element for indoor navigation. For all the other categories, for the exception of the Delfly, none fall within the necessary requirements. The work of Weinstein et al. (2018) does hold great potential, as their platform also contain a forward looking stereo-camera and enough processing to push their solution to multi-MAV high-level-navigation for indoor environments. Nonetheless, their platform is 5 times heavier and their required processing is 15 times higher than those of pocket drones. At the moment of writing, no work has been found of a full on-board implementation of indoor exploration with a swarm of MAVs, without the help of an external positioning system or an external computer, with the limitations imposed on the objective of this dissertation.

1.2. OBJECTIVES AND APPROACH

1.2.1. RESEARCH QUESTIONS

In this dissertation we would like to have a swarm of pocket drones to explore and navigate through an indoor environment. In the last section we have confirmed that this has not been developed elsewhere, and that the current solutions work on large platforms with a lot of computing power, with help of an external localization and/or a central computer. Due to over-complicated and computationally heavy navigation strategies, the application of pocket drone swarms cannot yet be realized. Hence the main research question of this dissertation can be formalized as follows:

MAIN RESEARCH QUESTION

To what extent can we design a robust and computationally efficient method for multiple pocket drones to explore an unknown, indoor environment and to return to their initial position?

We will hereby focus on the requirements necessary for search-and-rescue applications. In this scenario, it would not be possible to use a ground station for coordination and processing, and there is no external position system available as well. Also, the pocket drones should be able to disperse in an unknown environment and should be able to return to their initial position. The pocket drones can only use on-board sensing and processing and need to have direct inter-drone communication. For the main research question and all the upcoming sub-questions, the following requirements are necessary:

1

REQUIREMENTS

The pocket drones can only rely on their on-board sensors and processing, which implies that:

- *there cannot be any use of an external localization system, e.g. GPS, motion capture system*
- *there cannot be any use of an external computer for additional processing and giving specific local commands*

Before the pocket drones are ready for swarm-exploration, they would first need to perform stable velocity control. Once that has been achieved, they would need to avoid obstacles and each other. Therefore, we will first start with the low-level-navigation of pocket drones by formulating the following sub-question:

SUB-RESEARCH QUESTION I

To what extent can we achieve low-level navigation capabilities on multiple pocket drones, e.g. ego-motion estimation, obstacle avoidance and inter-drone avoidance?

Once the latter sub-research question is solved, we can step into a higher level of intelligence with the following sub-question:

SUB-RESEARCH QUESTION II

To what extent can we achieve high-level navigation capabilities on multiple pocket drones, e.g. exploration, coordination and homing?

This will be done with a "bottom-up" approach. The individual pocket drone needs a purpose, e.g. a goal to navigate to. It will need a method that enables it to do so, while meeting its low computational and memory requirements. Implementing this strategy on a swarm of pocket drones can enhance the exploration process, where they will coordinate their search once they come in close proximity.

These (sub-)research questions will be the red thread within this dissertation. In the next section, the outline will be discussed, where the objectives will be addressed by the content of the main chapters.

1.2.2. DISSERTATION OUTLINE AND HARDWARE

Based on the levels of navigational intelligence discussed earlier, a general outline has been developed for this dissertation (Fig. 1.5). It will be divided into two parts: Part I will present work on low-level-navigation and Part II will be about the high-level-navigation of both a single and multiple pocket drone(s).

In Chapter 2, we will describe a new efficient computer vision technique to compute optical flow, called Edge-Flow, that we developed in McGuire et al. (2016). Edge-Flow enables a small micro-processor to compute otherwise computationally heavy tasks.

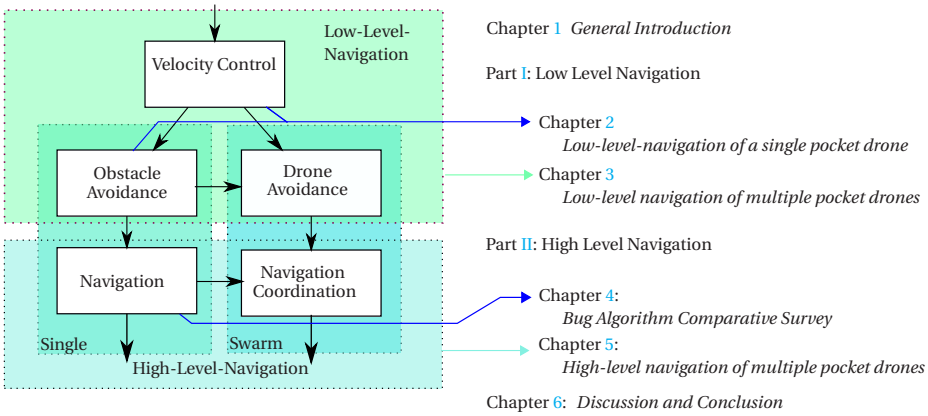
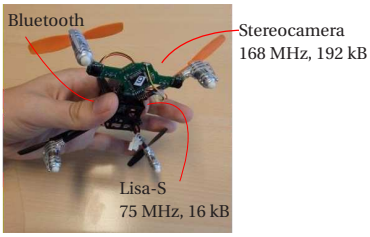
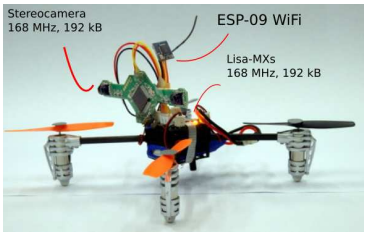


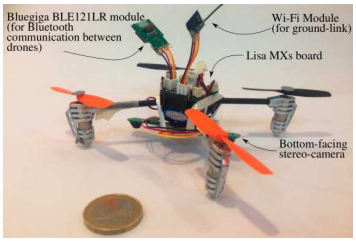
Figure 1.5: The outline of the dissertation



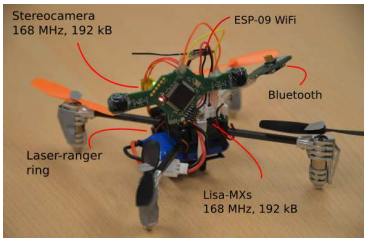
(a) 40 g, Chapter 2: Velocity Control



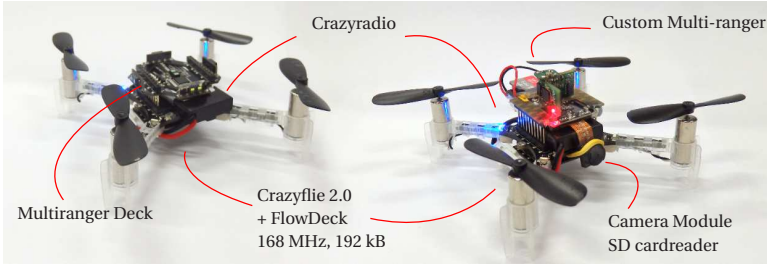
(b) 41.9 g, Chapter 2: Obstacle avoidance



(c) 43 g, Chapter 3: Intra-drone avoidance



(d) 43 g, Chapter 3: Intra-drone & obstacle avoidance



(e) 33 g, Chapter 5: High-level-navigation

(f) 35.7 g, , Chapter 5: Application platform

Figure 1.6: Overview of the platforms used in this dissertation.

Here we demonstrate Edge-flow by letting the pocket drone estimate its own velocity and compensate for it. Fig. 1.6a shows the platform used for this chapter. On a Walkera ladybird-frame with a Lisa-S autopilot (75 MHz,16 kB), we mounted a downward-looking stereo-camera with its own stm32F4 processor (168 MHz, 192 kB RAM). With the implementation of Edge-Flow on the stereo-camera's microprocessor, the pocket drone was able to react on high-level velocity commands given by a remote controller and match it with its own. We have extended Edge-Flow to Edge-FS (McGuire et al., 2017), which was able to compute both optical flow and stereo vision simultaneously. Fig. 1.6b shows the platform with a forward looking stereo-camera and an upgraded autopilot: the Lisa-MXs (168MHz, 192kB). With this assembly, the pocket drone was able to both estimate its own velocity and detect obstacles. In this chapter we also show an experiment where the pocket drone is able to autonomously fly with a simple wall-avoiding finite-state-machine in an indoor, office-like room, without the help of an external positioning system or computer.

In the last chapter of Part I, Chapter 3, we explain the Bluetooth-based inter-drone localization scheme developed in Coppola et al. (2018). Using the received signal strength intensity from the Bluetooth connection and the velocity estimation (by Edge-Flow of Chapter 2), two versions of the platform in Fig. 1.6c were able to localize and avoid each other. Fig. 1.6d, shows the pocket drone now with a forward facing camera, were we combined Edge-FS (Chapter 2) with the relative localization scheme (McGuire et al., 2017). The estimated bearing of the other drones is added on top of the detected obstacles by the stereo-camera on a collision detection disk. With this, two pocket drones were able to fly autonomously inside a room and avoid both the walls and each other by determining the safe directions to go.

Part II will focus on high-level-navigation, and therefore will start with Chapter 4, a survey of Bug Algorithms and its application for robotic platforms (McGuire et al., 2019). As explained in the last sections, the common SLAM techniques are too computational extensive to fit on the stm32f4 processors of the pocket drones. Bug algorithms are ideal candidates as it gives a simple solution to go from A to B with little memory and without the requirement of a map. It does not know the intermediate obstacles, but it will deal with them by following their border. Unfortunately, current implementations of bug algorithms do rely on external positioning systems, and once those are replaced with realistic sensor noise and estimation drift, the simulations in Chapter 4 show that the performances drop dramatically.

Eventually, we developed our own Bug Algorithm technique that was suitable for real-world navigation. With the new platform shown in Fig. 1.6e, the pocket drone was able to navigate based on the signal strength of a single home-beacon, in order to explore and come back to its starting position. Moreover, this method was also extended to multiple pocket drones, where they were able to avoid each other and coordinate the exploration phase. This method is called the swarm gradient bug algorithm (SGBA) and we were able to deploy up to 6 drones to navigate completely autonomously in a multi-room exploration scenario. In this chapter, we also show an application with 4 pocket drones as in Fig. 1.6f, where they reenacted a victim-search scenario and captured two "people" on their on-board cameras.

I

Low Level Navigation

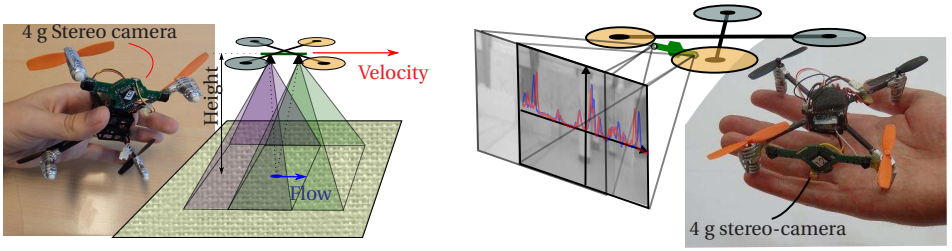
2

LOW-LEVEL NAVIGATION OF A SINGLE POCKET DRONE

Before we can think about designing autonomy for a swarm of pocket drones, we must first handle the individuals most fundamental ability: estimating its own motion. A quadrotor, like the pocket drone, is inherently unstable and should therefore be able to compensate for any side-ways drift. To move forward, it would also need to see obstacles and walls, and react upon it. No adequate technique yet exists to achieve this all on-board the 40g platform.

This chapter will introduce a new, efficient, computer vision technique called Edge-FS. This can run on a small 4g stereo-camera with a separate microprocessor and calculates stereo-vision (for obstacle-detection) and optical flow (for ego-motion-estimation), while matching the cameras frame rate of 25 Hz in speed. We will first go into the specifics of Edge-Flow, where we explain the utilization of a compressed image representation called Edge-Distributions and how to calculate optical flow from a downward-facing camera. We will then show the extension to Edge-FS, which is able to do the same with a forward-facing stereo-camera, and detect obstacles at the same time. By implementing this on a pocket drone, we will demonstrate a full-autonomous flight within a room, without a global localization system, where the pocket drone avoids the walls while maintaining a constant speed.

Parts of this chapter have been published in :
K.N. McGuire, G.C.H.E. de Croon, C. De Wagter, B. Remes, K. Tuyls & H. Kappen *Local histogram matching for efficient optical flow computation applied to velocity estimation on pocket drones*, IEEE International Conference on Robotics and Automation (ICRA) **3255-3260**, (2016)
K.N. McGuire, G.C.H.E. de Croon, C De Wagter, K. Tuyls & H. Kappen, *Efficient Optical Flow and Stereo Vision for Velocity Estimation and Obstacle Avoidance on an Autonomous Pocket Drone*, IEEE Robotics and Automation Letters **1070 - 1076**, 2 (2017).



(a) **(b)**

Figure 2.1: a) shows a pocket drone with velocity estimation using a downward-looking stereo-vision system. A novel efficient optical flow algorithm, Edge-Flow, runs on-board the stereo-camera with an STM32F4 processor running at only 168 MHz and with only 192 kB of memory. The optical flow and height, as determined with the stereo-camera, provide the velocity estimates necessary for the pocket drone's low-level-control. b) shows a pocket drone with the same forward-looking stereo-camera, which determines velocity and depth by Edge-Flow's extension, Edge-FS, which are the necessary components for the pocket drone's visual low-level-navigation.

2.1. INTRODUCTION

DEPLOYMENT of Micro Aerial Vehicles (MAVs) is important for indoor tasks such as inspections, search-and-rescue operations, green house observations and more. Tiny MAVs, also called pocket drones (<50 g, as in Fig. 2.1), are ideal for maneuvering through very narrow spaces, as often occurs in indoor environments. In order for them to autonomously navigate through a GPS-deprived area, there are several on-board sensors to consider (laser rangers, motion sensors, infrared rangers, sonar). The pocket drone's sensor of choice is an RGB camera. It is the most energy efficient and versatile sensing option, as multiple variables can be observed from the image stream: obstacles, motion, object recognition and more.

Using cameras enables the Micro Air Vehicle (MAV) to extract essential information for autonomous navigation. A stereo vision setup with two cameras has been particularly successful, for instance for obstacle avoidance (Hu and Mordohai, 2010). Since there are strict limitations on energy expenditure, sensing, and processing capabilities on a pocket drone, even relatively efficient stereo vision methods (Geiger et al., 2011, Hirschmuller, 2008) are computationally too heavy to run on-board a microprocessor. Therefore, an even more efficient stereo vision algorithm was developed, which is able to run at 10 Hz on a 20 g flapping wing MAV, the DelFly Explorer (De Wagter et al., 2014). It is still the lightest fully autonomous MAV to this date, which can fly through a room and avoid obstacles with purely onboard sensing and processing (Tijmons et al., 2017).

Since tailed flapping wing MAVs such as the DelFly Explorer are passively stable, there is no need to compute their velocity to compensate for drift. However, for inherently unstable platforms like a quadcopter, velocity estimation is necessary for stabilization when navigating in constrained areas. Optical flow is the way in which objects move in two sequential images and is the most important visual cue for velocity estimation. It can be calculated in a dense manner (Farnebäck, 2003, Horn and Schunck, 1981) or a sparse manner, e.g., by tracking features such as Shi-Tomasi (Jianbo Shi and

Tomasi, 1994) or FAST (Rosten and Drummond, 2005) over time with a Lucas-Kanade tracker (Bouguet, 2001). These types of techniques have proven themselves on numerous occasions (Honegger et al., 2013), nonetheless, do require a platform with a decent amount of computing power. On a pocket drone such standard optical flow methods either cannot run in real-time and will occupy a large percentage of its processing power, leaving little to no room for other types of processing. Especially when autonomous flight is the final goal, optical flow determination will only constitute a part of what the MAV has to do, as much more information can be retrieved from the image stream.

In order to design a computationally much more efficient optical flow algorithm, we have drawn inspiration from the study in Lee et al. (2004), which proposed using spatial edge distributions to track motion in the image. This chapter presents *Edge-Flow*, which improved upon the work in Lee et al. (2004) by introducing a variable time horizon for determining sub-pixel flow. Edge-Flow runs embedded at 30 Hz on a lightweight stereo-camera positioned underneath a pocket drone (Fig. 2.1a). The stereo-camera is pointing down and detects optical flow and a global height estimate, assuming that it is looking at a flat ground surface. With these, the MAV determined its own velocity and used this in a guided control, where it autonomously matched externally-given velocity references. However, a 4 g stereo-camera for a 40 g pocket drone is a significant weight, so it is a waste to have this "heavy" sensor looking downward and not using it to avoid obstacles in the flight direction.

This chapter also presents a major extension of Edge-Flow, which enables the stereo-camera to face forward on a MAV, so it can be used for navigation purposes (Fig. 2.1b). As the pocket drone will now be facing hallways, rooms, doors etc., the assumption of looking straight at a flat plane will not hold anymore. The same matching paradigm used to determine Edge-Flow, will now be used to not only calculate optical flow but also stereo depth over the entire image. *Edge-Stereo*, as called for convenience, uses the so-determined distances to properly scale the locally observed optical flow in order to retrieve a velocity estimate. This combination of Edge-Flow and Edge-Stereo will be called *Edge-FS*.

Our main contribution is that the presented method provides both velocity and distance estimates, while still being computationally efficient enough to run close to the frame rate on a very limited embedded processor. As such, the method enables unstable MAVs such as tiny quadcopters to perform fully autonomous flights in unknown environments. The Edge-Flow and Edge-Stereo methods will be explained in more detail in section 2.2.1. Off-line results for velocity estimates with a set of images is shown in section 2.3. From here, the algorithm is embedded on the lightweight stereo-camera and placed on 40 g pocket drone for velocity estimation (section 2.4.3). Finally, the velocity estimate is used together with Edge-Stereo-based obstacle detection to perform fully autonomous navigation in an environment with obstacles (section 2.4.4). This is followed by some concluding remarks.

2.1.1. RELATED WORK

In related research, several works have achieved optical flow based control of a MAV (e.g., Grabe et al. (2015), Kendoul et al. (2009a), Romero et al. (2009)). As mentioned in the introduction, the standard optical flow methods are computationally too heavy to

run on a quadcopter of less than 50 g. For instance, Dunkley et al. have flown with a 25 g quadcopter before, while computing optical flow for visual odometry (Dunkley et al., 2014). However, this was done on an external computer. As miniaturization of hardware also poses a limitation on communication bandwidth, this can result in a significant delay in the controls. To obtain full autonomy, it would be wise to uncouple a MAV of any external dependencies.

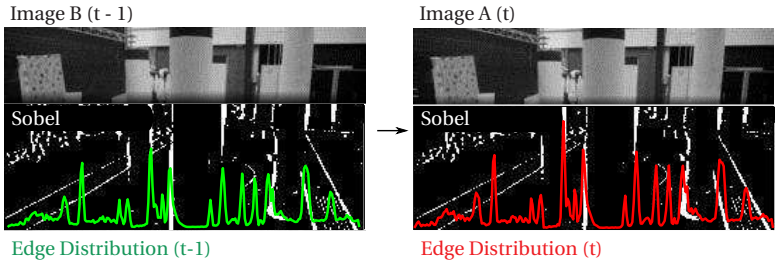
To design extremely lightweight MAVs for autonomous flight, some researchers looked into EMD sensors (Ruffier et al., 2003) and other 1D signal sensors (Green and Oh, 2008). Briod et al. (2013) proposed the design of a 45 g quadcopter for optical flow based control with 1D flow sensors. They followed up with this research on a heavier 278 g platform containing 8 of these sensors pointing in all directions (Briod et al., 2016). With this they could hover the quadcopter in various cluttered environments. The results are impressive, nevertheless they were achieved by using multiple single purpose sensors. As they can only sense motion, it does not leave much room to detect other variables necessary for navigation.

More similar to our research, Moore et al. implemented an efficient optic flow algorithm on a small lightweight (2 g) omnidirectional camera system on a 30 g helicopter (Moore et al., 2014). With a ring of 8 low-resolution image chips (64 x 64 pixels), the MAV could compute optical flow. It did this by computing the edges, compressing the images and calculate the displacement by block matching which resulted in translational optical flow. The vision calculations where done on-board the helicopter with 10 Hz, yet the flight controls where computed off-board. Although the potential of a full on-board implementation is there, the redundancy lies in the ratio of cameras to sensed variables. One camera has the potential of detecting flow in 3 directions; they used 8 to only detect 2 (forward and sideways velocity).

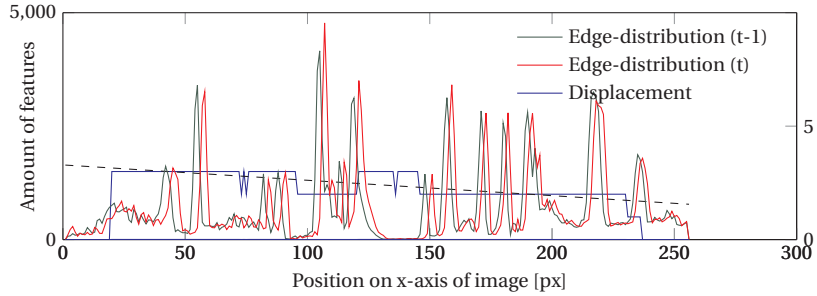
Optical flow can also be used to detect obstacles (Mori and Scherer, 2013), however the MAV needs to be constantly on the move. This is not required if stereo vision is used for depth information. With this, Oleynikova et al. developed a reactive avoidance controller for a quadcopter (30 cm in diameter) (Oleynikova et al., 2015). From the obtained stereo disparity map, they accumulated the values along the columns to get a summed disparity factor. Assuming that the obstacles are vertical and long, these can be detected quickly. The stereo map was calculated over the entire image first before accumulation to a vector. This significantly impacts the amount of computation making it less suitable for implementation on a smaller MAV.

2.2. VELOCITY AND DEPTH FROM EDGES

To achieve autonomous navigation with a camera on an unstable pocket drone, we need to obtain two variables: velocity and depth. In the introduction we mentioned that many of the mainstream computer vision techniques will be computationally too heavy to run on the pocket drone. Edgeflow is able to detect optical flow within the image in a semi-dense but computationally efficient manner, as it reduces the 2D image search problem to 1D signal matching by the use of edge-feature distributions. For a down-ward looking stereo-camera, it can estimate the pocket drone's forward and sideways velocity. This section will also explain the modifications that are necessary to make the stereo-camera point forward and still be able to measure those variables. Edgeflow will be explained in



(a) Vision Loop



(b) Local edge distribution matching

Figure 2.2: a) The vision loop with for creating the edge feature distributions and b) the velocity estimation by measuring optical flow with one camera and height with both cameras of the stereo-camera.

this chapter and subsequently, we will present its extension with Edge-Stereo to Edge-FS, which will be used for obstacle detection in the experiment part of this chapter.

2.2.1. FROM DOWNWARD-LOOKING CAMERA TO VELOCITY

The generated edge distributions are created by first calculating the gradient of the image on the vertical and horizontal axis using a Sobel filter (Fig. 2.2a). From these gradient intensity images, the distribution can be computed for each of the image's dimensions by summing up the intensities. The result is an edge feature distribution of the image gradients in the horizontal and vertical direction of the image coordinates.

From two sequential frames, these edge distributions can be calculated and matched locally with the Sum of Absolute Differences (SAD). In Fig. 2.2b, this is done for a window size of 18 pixels and a maximum search distance of 10 pixels in both ways. The displacement can be fitted to a linear model with least-square line fitting. This model has two parameters: a constant term for translational flow and a slope for divergence. Translational flow stands for the translational motion between the sequential images, which is measured if the camera is moved sideways. The slope/divergence is detected when a camera moves to and from a scene, however this will be revisited later in this section.

Due to the image sensor's resolution, existing variations within pixel boundaries cannot be measured, so only integer flows can be considered. However, this will cause complication if the camera is moving slowly or is well above the ground. If these types of movements result in sub-pixel flow, this cannot be observed with the current state of the

edge flow algorithm. This sub-pixel flow is important to ensure velocity control on an MAV.

To ensure the detection of sub-pixel flow, another factor is added to the algorithm. Instead of the immediate previous frame, the current frame is also compared with a certain time horizon n before that. The longer the time horizon, the more resolution the sub-pixel flow detection will have. However, for higher velocities it will become necessary to compare the current edge distribution to the closest time horizon as possible. Therefore, this time horizon comparison must be adaptive.

Which time horizon to use for the edge distribution matching, is determined by the translational flow calculated in the previous time step (o_{t-1}):

$$n = \min\left(\frac{1}{|o_{t-1}|}, N\right) \tag{2.1}$$

where n is the number of the previous stored edge distribution that the current frame is compared to. The second term, N, stands for the maximum number of edge distributions allowed to be stored in the memory. It needs to be limited due to the strict memory requirements and in our experiments is set to 10. Once the current distribution and time horizon distribution are compared, the resulting flow must be divided by n to obtain the flow per frame.

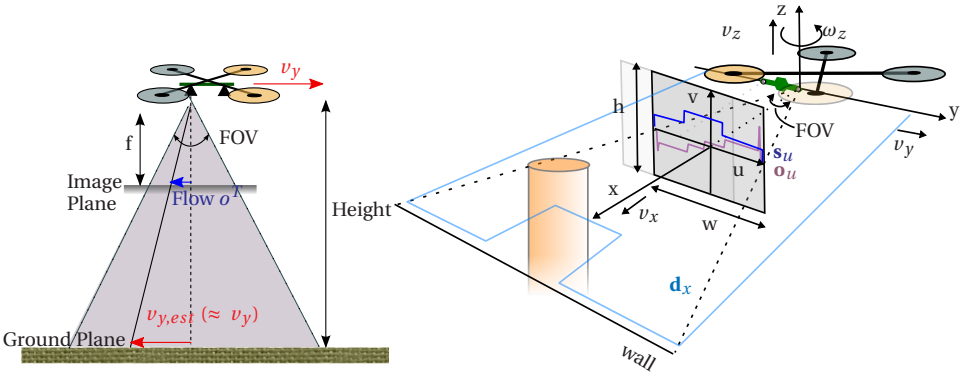
As seen in Fig. 2.3a, the velocity estimation V_{est} can be calculated by means of Edge-Flow combined with the height of the drone and the angle from the center axis of the camera:

$$V_{est} = \frac{h \cdot \tan(o_t^T \cdot \frac{\alpha_{FOV}}{w})}{\Delta t} \tag{2.2}$$

where o_t^T is the translational flow as calculated by the linear model fit, h is the height of the drone relative to the ground, and w stands for the pixels size of the image (now shown in the y-direction). α_{FOV} stands for the Field of View (FOV) of the image sensor. A MAV can monitor its height by means of a sonar, barometer or GPS. In our case we do it differently, as we match the left and right edge distribution from the stereo-camera with global SAD matching. The results for velocity estimation and control with Edge-Flow with a downward camera can be found in section 2.4.2.

2.2.2. FROM FORWARD-LOOKING CAMERA TO VELOCITY AND OBSTACLES

When looking orthogonally at a planar ground surface while moving, the optical flow field is rather simple and allows for easy determination of the forward and sideways velocities with the help of a single height measurement. But to navigate without bumping into anything, the MAV needs to see objects in the direction of motion, which is typically forward. Due to the likely non-planar (3D structure) of the environment in forward direction, the optical flow field will become more complex. Moreover, the forward velocity now can only be observed by means of the divergence of optical flow, which is more difficult to determine, especially close to the focus of expansion. Here we delve into how we determine the velocities with the help of forward facing stereo-images. In principle,



(a) Downward-looking stereo-camera **(b)** Forward-looking stereo-camera

Figure 2.3: a) Velocity estimation by measuring optical flow with one camera and height with both cameras of the downward-looking stereo-camera. b) For a forward-looking stereo-camera: the MAV's body fixed coordinates with respects to the camera axis, shown for the left camera (XYZ). The conventional aircraft coordinates of east-north-up is used for the MAV as the camera. The image coordinates in width and height are represented as u and v respectively.

the unscaled velocities and the rotation rates can be determined from the image alone, according to the paper of Longuet-Higgins and Prazdny (1980). They theorized that measured flow ($\mathbf{o}_u$) is the summation of a translational flow ($\mathbf{o}_u^T$) and rotational component ($\mathbf{o}_u^R$). Before estimating the horizontal planar velocity, we first have to determine $\mathbf{o}_u^R$.

Although Longuet-Higgins and Prazdny (1980) assume rotations in all directions, we can make simpler assumptions for the pocket drone. Fig. 2.3b shows the placement and axis definition of the drone and camera. For obstacle avoidance it is essential to look in the direction of motion, which in this case is the direction of the positive x axis. Here, correctional pitch and roll motion for drift compensation will be relatively small, but yaw rotations will be more common. Assuming that the latter only has significant effect on the optical flow, $\mathbf{o}_u^R$ can be approximated (assuming small angles) using the gyroscopes on the on-board IMU of the pocket drone:

$$o_{u,i}^R \approx \omega_Z \cdot \frac{w}{\alpha_{FOV}} \tag{2.3}$$

$$\mathbf{o}_u^R = [o_{u,1}^R, \ldots, o_{u,w}^R] \tag{2.4}$$

where w is the width of the image, α_{FOV} is the angle of the FOV and ω_Z is the yaw rotation measured from the gyroscopes.

Now that $\mathbf{o}_u^R$ is known, we can isolate $\mathbf{o}_u^T$ to determine the pocket drone's forward (v_x) and sideways velocity (v_y). With the coordinate system we use in this chapter (Fig. 2.3b), Longuet's equation of $\mathbf{o}_u^T$ is expressed as:

$$\mathbf{o}_u^T = (-v_y + \mathbf{x}v_x)/\mathbf{d}_x \tag{2.5}$$

$$\mathbf{d}_x \mathbf{o}_u^T = -v_y + \mathbf{x}v_x \tag{2.6}$$

Where $\mathbf{x}$ is an array of indices of the image columns. Depth, $\mathbf{d}_x$, scales the optical flow resulting in motion parallax, as close objects appear to move faster than objects far

away. In section 2.2.2, a global height estimate was used to scale the optical flow back to velocities, which is sufficient if the camera is looking at a flat floor or perpendicular to a straight wall. This assumption will not hold when the MAV is flying towards a wall at an angle or whenever obstacles at different distances are in the field of view. This non-constant depth needs to be accounted for when scaling the optical flow, therefore the stereo depth is needed over the entire size of the image for a better velocity estimate. Local right-left image disparity from a stereo-camera can be transformed to actual depth in meters by using the camera parameters, with the following approximation:

$$\mathbf{d}_x \approx \frac{w \cdot r}{\alpha_{FOV} \cdot \mathbf{s}_u} \tag{2.7}$$

where r is the baseline between the two cameras, and the stereo disparities in pixels along the image columns is $\mathbf{s}_u$.

With depth $\mathbf{d}_x$ and translational optical flow $\mathbf{o}_u^T$, it is now possible to calculate the MAV's sideways and forward velocity by fitting a linear model to (2.6). In the next section we will explain how to obtain both optical flow and stereo depth from a stream of low resolution stereo-images.

2.2.3. PROCEDURE FOR EDGE-FS

If image A and B from Fig. 2.2a are two temporal sequential images (t with $t-1$) in time, this will result in the pixel flow, thus Edge-Flow as explained in Section 2.2.1. Based on the previous flow value, Edge-Flow adaptively chooses how far in time ($t-n$) it will compare the current edge distribution to. On top of that, the flow shift predicted by a yaw rotation ($\mathbf{o}_u^R$, as calculated in equation (2.4)) will shift the start of the block matching scheme. This and the adaptive time horizon will be present for the experiments in this chapter. Note that in section 2.2.2, also the direction along the image height was used to estimate the forward velocity (as the camera was looking down). For a forward-looking camera, it will not be used as the forward velocity (v_x) will be subtracted from the divergence of Edge-FS.

Previously in Section 2.2.1, the entire edge distribution of the left and right image were matched to obtain a global depth estimate. To get a better velocity estimate with a forward camera, we need to use pixel disparity per column. To calculate both column-wise optical flow and stereo vision and keep the algorithm computationally efficient, the exact same matching principle of Edge-Flow (Fig. 2.2a) is used, resulting in Edge-Stereo. Disparity to depth in meters is calculated with the known camera parameters and (2.7) from the last section. Sequentially, Edge-Stereo scales Edge-Flow to compensate for the motion parallax (see Fig. 2.4d), which results in the left side of (2.6). These values will then be fitted to a linear model (Fig. 2.4e), which gives us the slope and intercept of the line. With the camera parameters, the forward and sideways velocities are estimated (Fig. 2.4f).

2.3. OFF-LINE VISION EXPERIMENTS

Before implementing the algorithm on the actual stereo board, Edge-Flow was run on a set of stereo-images in MATLAB (version R2015b on a Dell Latitude E7450, i7-5600U CPU @ 2.60GHz processor). Fig. 2.5a shows screen shots of the data set used in this section,

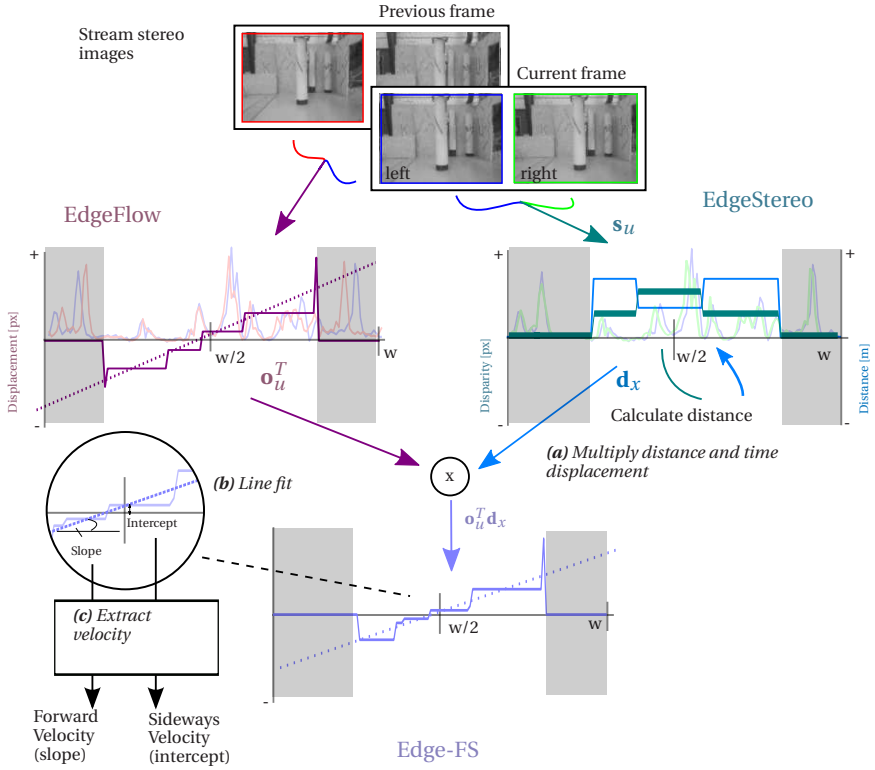


Figure 2.4: The temporal pixel disparities per column of Edge-Flow is a) scaled by Edge-Stereo. Following (2.6), b) a line fit is done on this array of values, c) from which the forward and sideways velocities can be extracted from the slope and intercept, respectively.

where the camera moves towards obstacles at different distances. In Fig. 2.5b, Edge-Flow scaled by Edge-Stereo, now dubbed as Edge-FS, results in the velocity estimates.

Edge-FS is contrasted against the well-known optical flow method developed by Farnebäck (2003), a dense optical flow method (Fig. 2.5). Although less used than a more conventional KLT-tracker (Bouguet, 2001), preliminary analyses indicated it to be more suited for the low-resolution images used here. With its default parameters set as in MATLAB R2015b, the sparse magenta line illustrates that the KLT-tracker indeed has difficulties with the low-quality, low-resolution images (128 x 96 pixels).

For Färneback, depth is determined by matching the stereo-images with each other and converting the resulting pixel disparity to a distance. To get velocity, the same line-fit is used as for Edge-Flow[1], but here the whole image is considered rather than the compressed form like the edge distributions. After comparison of the methods with different parameters, both Edge-FS and Färneback are set up with a window size of 11 pixels and a search range of 15 pixels (Färneback's pyramid level at 1). Both forward (x) and sideways

[1]The Edge-Flow code as embedded on the stereo-camera has a mean computation time for Edge-Flow is 0.00134 seconds (compiled for Linux) and for Färneback is 0.00466 seconds on the same stereo-image data set.

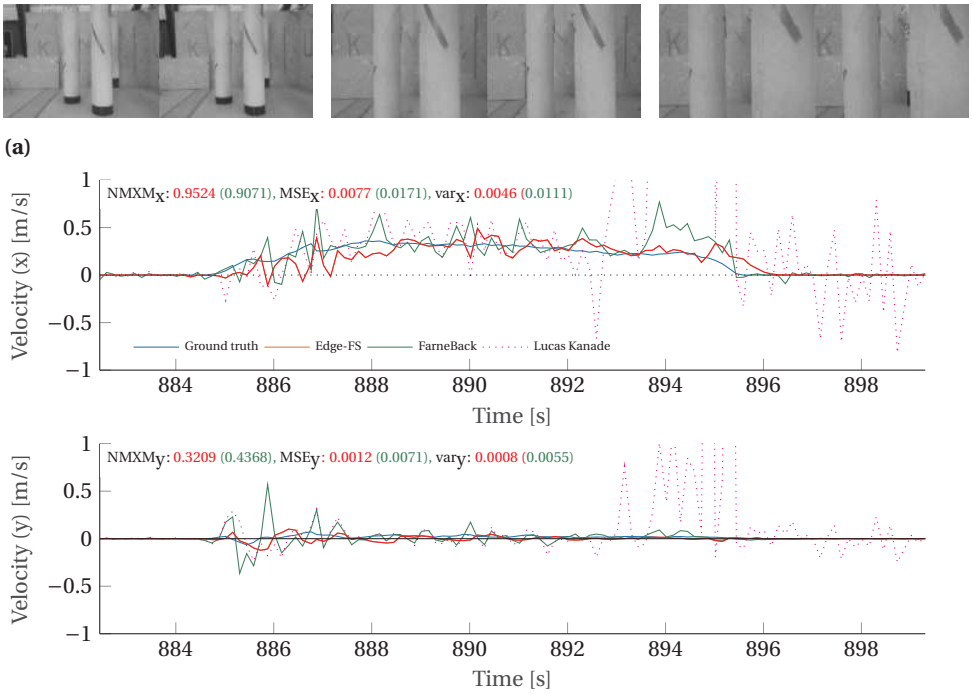


Figure 2.5: a) Several screen shots of the set of images used for off-line estimation of the velocity. Here the diversity in amount of texture can be seen. b) Off-line velocity estimate calculated by Edge-FS and Färneback, held against the ground truth for the forward moving stereo-camera's data set.

(y) velocity measurements, shown in Fig. 2.5, are compared against the "ground truth" as obtained with an OptiTrack motion capture system[2], with 24 infrared cameras. The plots also include several values to determine the quality of the velocity estimates: Mean Squared Error (MSE), Variance (VAR) and Normalized Maximum Cross-Correlation Magnitude (NMXM). A low MSE indicates greater similarity and low VAR is a smaller spread of the measurement from the ground truth. A high NMXM stands for a better shape correlation between the two. All these metrics indicate Edge-FS to obtain more accurate results on this data set than the computationally more expensive Färneback method.

It is important to note that because of the nonlinear relation between pixel disparity and depth in stereo vision, far distances are measured less accurately. The disparities for further distances will become sub-pixel and hard to determine. This is especially relevant to the small stereo board used in this study, which we set up to use 128 x 96 pixel images for the 57.4 x 44.5 deg FOV . Also, the translational optical flow of objects is harder to measure when they are further away, since it becomes sub-pixel as well. Hence, both terms on the left in (2.6), $\mathbf{s}_u$ and $\mathbf{d}_x$ become less accurate at far distances. This correlation between distance and accuracy can be seen in the box plot of Fig. 2.6.

[2] www.optitrack.com

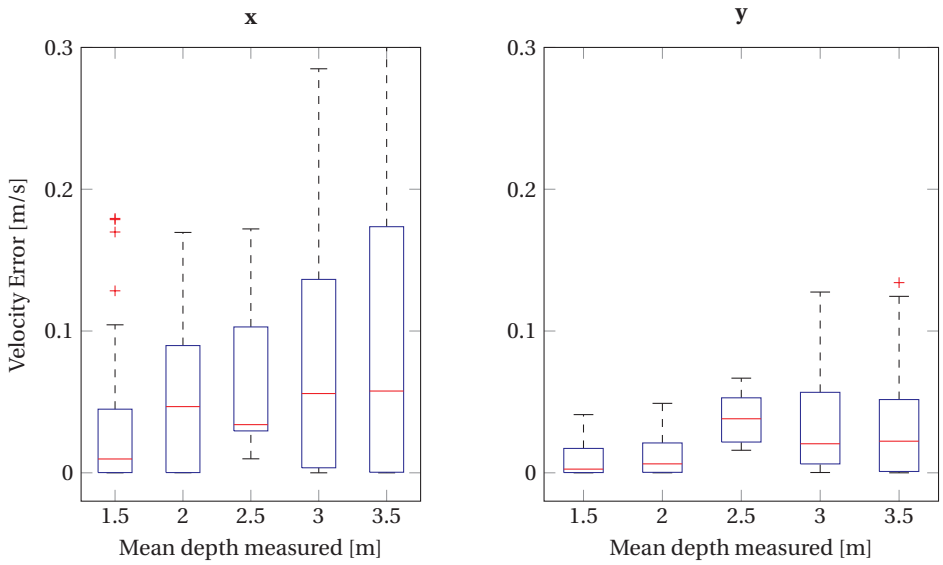


Figure 2.6: Boxplot of the absolute velocity estimate error of Edge-FS, compared against the mean observed depth.

Besides the difficulty with larger distances, which is fundamental to stereo vision, Edge-FS also has some difficulty determining the forward flow when there is a large lateral motion. Sideways velocity has a 0^{th} order effect on the flow, while forward velocity information is captured by the divergence of the flow field, which is a 1^{st} order effect. Therefore, the forward velocity is more subject to noise and harder to estimate (this can be observed in Fig. 2.6 as the errors are generally higher for the x-direction than the y-direction). In this work, the MAV will mostly fly forward. In this situation lateral flow is kept very small while the divergence is larger and more observable. A larger SAD window size and filtering are used to correct for the remaining noise.

2.4. EXPERIMENTS ON THE POCKET DRONE

In this section, we explain the implementation of Edge-FS on-board a pocket drone and how it is used in an autonomous obstacle avoidance task. We will first present a guided control with the velocity estimate with the downward-looking stereo-camera by Edge-Flow. We will then present the velocity estimates by Edge-FS during flight with the forward-looking stereo-camera. Subsequently, a closed loop flight is shown, where the drone autonomously navigates through a room, while maintaining its velocity and avoiding obstacles.

2.4.1. HARDWARE SPECIFICS

Edge-Flow and Edge-FS runs embedded on the stereo-camera (as introduced in De Wagter et al. (2014)). Fig. 2.7 displays the stereo-camera with 1/6 inch image sensors, with a baseline of 6 cm and an FOV of 57.5^o x 44.5^o. The stereo-camera has an embed-

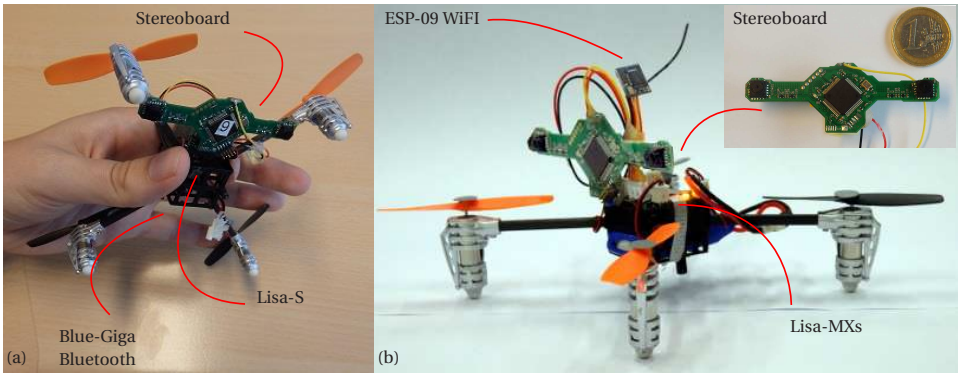


Figure 2.7: The 4 g stereo-camera mounted on the pocket drone in a) a down-ward looking position and b) a forward looking position.

ded microprocessor, an STM32F4 with a speed of 168 MHz and 196 kB of memory in which the largest consecutive memory block spans 128 kB. The cameras are configured to output stereo-images with a size of 128 x 96 pixels to fit within memory and processing constraints. The maximum reachable frame rate of the stereo-camera is 30 Hz, which is not much affected by the computation of Edge-FS (approx. 0.0175 sec).

For the experiments, a pocket drone will be first equipped with a single bottom-facing stereo-camera in Fig. 2.7a (see Section 2.4.2) and then with a front-facing stereo-camera in Fig. 2.7b (see Section 2.4.3 and 2.4.4)). A frame of a Walkera QR LadyBug[3] is adopted as a base. An adapted smaller variant of the Lisa-MX[4] will be used as the auto-pilot[5]. The *Lisa-MXs* also carries an STM32F4 microprocessor, with a speed of 168 Hz and 1 MB of flash memory. With an ESP-09 WiFi module, telemetry can be broadcasted to the computer to receive all the measured variables required for validation[6]. The entire assembly, including stereo-camera and battery, weighs exactly 41.9 g.

The auto-pilot program flashed on the Lisa-MXs is Paparazzi[7]. The software runs entirely on-board the microprocessor which governs all the basic flight controls. An adaptive Incremental Nonlinear Dynamic Inversion (INDI) controller (Smeur et al., 2016) is used for the attitude stabilization of the MAV. The guidance controller resides on top of the stabilization control, to calculate the desired pitch and roll angle, to achieve a desired altitude position or airspeed. In this chapter, it will be applied to maintain a desired velocity. It will need the measurements from the stereo-camera, operating in parallel with the Lisa-MXs.

[3] http://www.walkera.com/
[4] http://wiki.paparazziuav.org/wiki/Lisa/MX
[5] Fig. 2.7a had the previous version of the autopilot, the Lisa-S
[6] Fig. 2.7a had instead a Blue-giga BLE211 module for telemetry
[7] http://wiki.paparazziuav.org/

2.4.2. ON-BOARD VELOCITY ESTIMATION AND CONTROL WITH THE DOWNWARD-FACING CAMERA.

With the platform shown in Fig. 2.7a, we first showed a flight with Edge-Flow's velocity estimate, as in Section 2.2.2, in the control-loop. During a guidance control task with externally given speed references, the pocket drone's flight lasted for 370 seconds. Mostly speed references where given in the x-direction (forward), however occasional speed references in the y-direction (sideways) were necessary to keep the pocket drone flying over the designated testing area. A sample of the velocity estimates during that same flight is displayed in Fig. 2.8a and b. From the MSE and NMXM quality values for the x-direction, it can be determined that Edge-Flow's estimated velocity correlates well with the ground truth. The pocket drone obeys the speed references given to the guidance controller Fig. 2.8d.

Noticeable in Fig. 2.8b is that the NMXM for y-direction is lower than for the x-direction. As most of the speed references sent to the guidance controller were for the x-direction, the correlation in shape is a lot more eminent, hence resulting in a higher NMXM value. Overall, it can be concluded that pocket drone can use the 4g stereo-board for its own velocity controlled guidance. Fig. 2.8b gives a screen-shot of the video of the experiments[8], where it can be seen that the pocket drone is flying over a feature-rich mat.

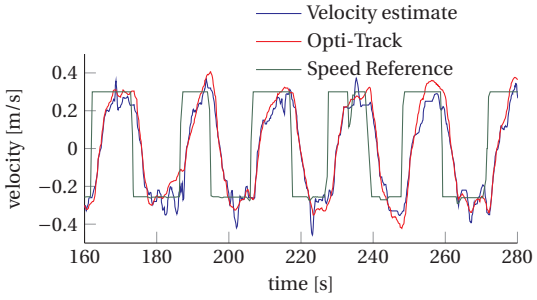
2.4.3. VELOCITY ESTIMATION WITH THE FORWARD CAMERA

We have shown in section 2.3 that Edge-FS can measure the camera velocity based on a collection of images. Now implemented in the forward-looking 4 g stereoboard fixed on the pocket drone, the question remains if it can still retain its quality with all the additional effects caused by motion and vibrations during flight.

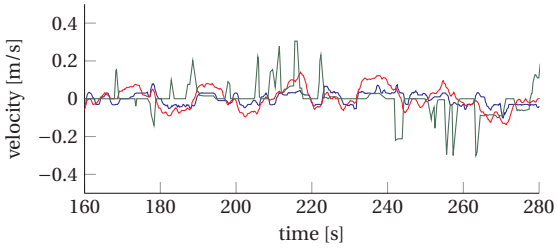
Fig. 2.9a, presents the velocity estimates of Edge-FS, during a manually controlled flight in front of a textured screen (screen shot in Fig. 2.9b and position in Fig. 2.9c). The same OptiTrack system used for the image data set (Fig. 2.5a), is monitoring its real velocity. The raw unfiltered velocity measurements of Edge-FS are contrasted with this ground truth with NMXM, VAR and MSE . Noticeable is that the forward velocity shows more noise peaks than the sideways velocity, as expected (see Section 2.3). However, in both directions, Edge-FS matches the ground truth adequately, which should be sufficient for the closed-loop flight.

To use the actual raw measurements in flight is undesirable. The most common way is to fuse these vision-based velocity estimates with the accelerometers. On a larger MAV than the pocket drone, this would be possible because of the damping. However, many vibrations are generated by the small propellers, which are in close proximity with the autopilot, the accelerometers readings contain too much noise. Therefore, in this chapter, we use a vision-only approach applying a median filter to the 5 last velocity measurements, to keep the delay to a minimum.

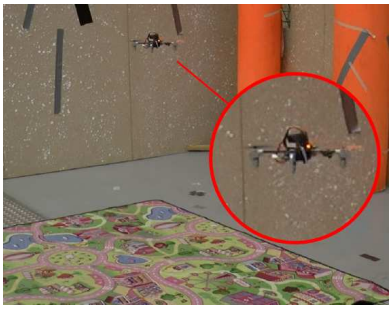
[8] YouTube playlist:
https://www.youtube.com/playlist?list=PL_KSX9GOn2P9TPb5nmFg-yH-UKC9eXbEE



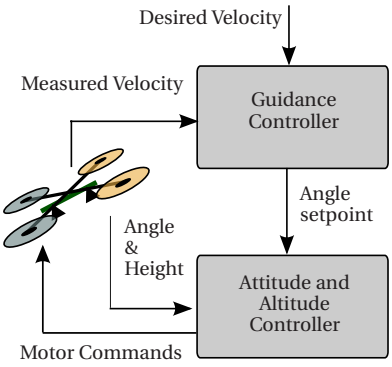
(a) Velocity estimate x-direction (MSE: 0.0041 m, NMXM: 0.9631)



(b) Vertical velocity y-direction (MSE: 0.0025 m, NMXM: 0.7494)



(c)



(d)

Figure 2.8: Velocity estimates calculated by the pocket drone and stereo-board assembly, now using estimated velocity in the control in the a) x and b) y-direction. MSE and NMXM values are calculated for the entire flight which lasted for 370 seconds, where several external speed references were given for guidance. a) A screen-shot of the video of the flight and b) the control scheme of the velocity control.

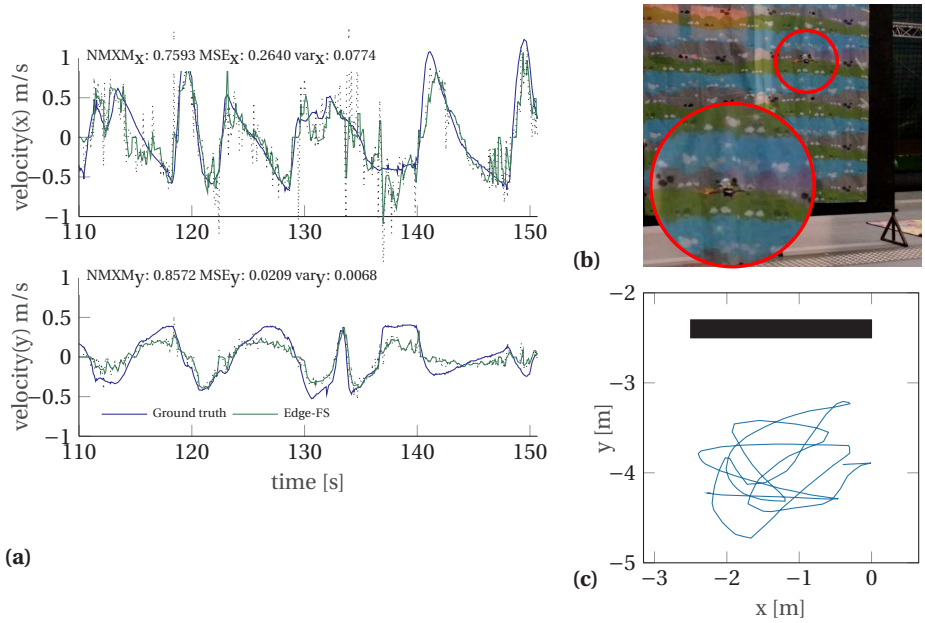


Figure 2.9: a) Velocity estimation by Edge-FS on the pocket drone, where the otted line is the unfiltered velocity estimate by Edge-FS. b) shows a screen shot from the flight in front of a wall and c) the position during a remote controlled maneuver as measured by the MoCap system.

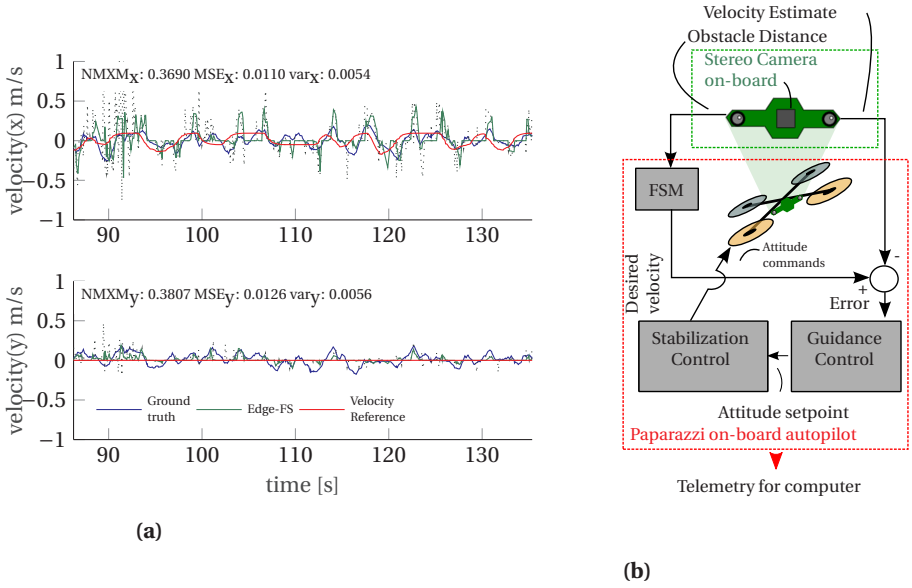


Figure 2.10: a) Velocity control on pocket drone with a wall force field and b) the control scheme used, explaining the hierarchy of the on-board sensors and controllers.

2.4.4. AUTONOMOUS OBSTACLE AVOIDANCE

In the previous subsection, we showed validation of the velocity estimate as calculated by Edge-FS on a forward-looking stereo-camera. Now we will present a closed-loop flight, where the pocket drone avoids obstacles identified by means of its stereo vision, while guided by its velocity estimates. The main goal of this experiment is to show the potential of the proposed algorithms for full autonomous navigation. In this section, the vertical position as measured by OptiTrack is exclusively used for height control, as no position measurement is used in the horizontal plane (solely for validation afterwards). This is where the MAV uses its velocity estimates by Edge-FS.

Fig. 2.10b displays the basic control scheme for the navigation task. It determines a desired velocity to avoid collisions. The error between the estimate and the desired velocity is the input to the velocity guidance controller, which sets an attitude set-point for the stabilization. Subsequently, Edge-Stereo determines the nearest object to camera. If too close, it will produce a backward velocity reference to the guidance controller (a force field), therefore preventing the pocket drone from hitting the wall face-on. Fig. 2.10a shows the readings from a short flight of a simple hover with the obstacle force field.

When encountering a wall/obstacle, the pocket drone will need to move away from the situation. The avoidance scheme is a simple finite state machine (FSM) with 4 behavioral states (see Fig. 2.11b). It starts in *check* mode, where the pocket drone will check if there is a detected obstacle within 1 meter by Edge-Stereo. If the way is clear, the pocket drone moves *forward* with a constant speed (set now to 0.3 m/s), guided by the velocity estimate from Edge-FS. If it detects an object on its path, the MAV will first hover for 1 second actively controlling the forward velocity to zero. Then it will turn quickly with a constant angle relative to the heading (here $\Delta\psi = 60^o$). Immediately thereafter, the MAV will evaluate the situation in the *check* mode and proceeds from there.

We conducted multiple autonomous flights with the pocket drone. Fig 2.11a shows the result of 3 representative flights of the pocket drone with the forward looking stereo-camera. The pocket drone has to navigate in a small room of 4 x 4 meter with varying textured surfaces (screen shot of camera footage). All the flights lasted longer than 90 seconds, from which the longest duration was 122 seconds (flight 3). When the pocket drone brushed against the wall, the safety pilot took over the flight with a remote control for a safe landing. The most common failure case during the test flights, is that the MAV will approach the wall with a small angle. After the turn with constant angle, the drone will fly almost parallel to the wall which it can not detect due to its limited FOV . This is the case for flight 2 and 3, except for flight 1, in which case the pocket drone was facing the observer after a turn. Several flights of the pocket drone have been done within a real-world environment (Fig. 2.11c), which can be observed with the accompanying video and YouTube list[9].

The mentioned failure case for the autonomous flights is difficult to overcome. If the MAV would turn and face a large open space, the distance for Edge-Stereo could be far enough to compromise the quality for the velocity estimate due to the small base line of the stereo-camera. As we already observed in Fig. 2.6, this would cause the pocket drone to drift, which is problematic when near a wall/obstacle after the turn. If an obstacle is not in its FOV , the chances of collision significantly increases. This could be solved by

[9] https://www.youtube.com/playlist?list=PL_KSX9GOn2P812tmddfrTlURHNieRe6YY

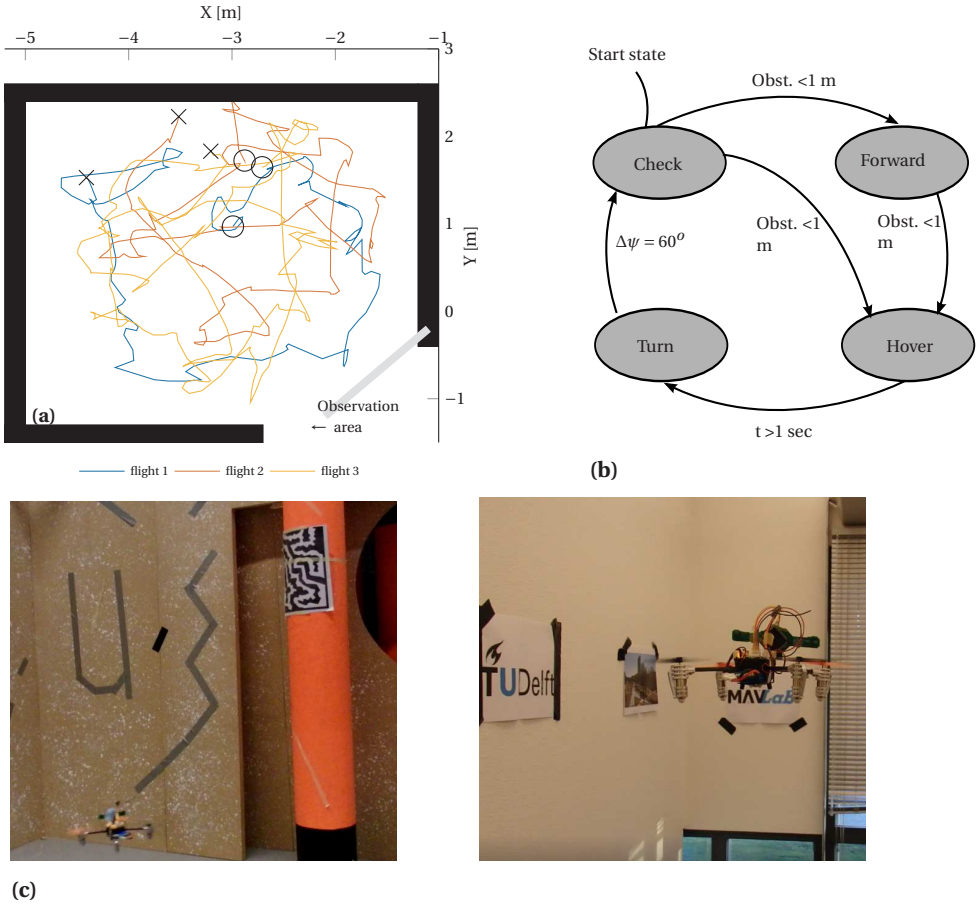


Figure 2.11: a) A position plot of 3 flights, from which the first lasted 91 seconds, the second 101 seconds and the third 122 seconds. b) shows screen shots of the experiments in the flight arena (left) and in a real-world office (right). Some posters were added to the latter to provide extra texture.

merging the *check* and *turn* node of the FSM, so it will only stop turning at a significant clear path. Another solution is to add a lightweight short range sensor on the sides of the pocket drone, so it will detect immediately if the drone is flying close and aside an obstacle.

The obstacle avoidance logic will need some additional work, however the experiments show that Edge-FS can be used in navigation overall. During the autonomous flight, the pocket drone was stabilizing itself using the velocity estimates of its forward camera alone.

2.5. CONCLUSION

A computationally efficient optical flow and stereo algorithm is presented in this chapter, called Edge-FS. It runs embedded on a very lightweight stereo-camera and can be carried by a 40 g pocket drone for determining velocity and depth. The presented algorithm allows the stereo-camera to face forward, a direction in which a complex 3D structure can be expected.

We presented experiments where the pocket drone with the stereo-camera autonomously navigated and avoided obstacles in an area of 4 x 4 meters. A simple finite state machine controller showed that the velocity estimates and the depth measurement can be used for fully autonomous flight. The current work lays the basis for stabilization and collision avoidance on pocket drones with a single, small stereo vision system.

3

Low-Level Navigation of Multiple Pocket Drones

Besides estimating their own velocity and avoiding obstacles (Chapter 2), the pocket drones should also be capable of detecting and avoiding each other. Relative intra-drone localization could be done with a global localization system, of which the global coordinates can be communicated among the MAVs, however we have already established in the introduction that there is no guarantee to apply this in real-world indoor applications. The pocket drones should therefore be able to do the relative localization and avoidance in a decentralized manner. However, no method exists to achieve this on the small scale of the pocket drones.

This chapter will present a complete on-board localization and avoidance scheme for pocket drones. First, we will show a solution based on the received signal strength intensity (RSSI) of Bluetooth, from which the pocket drones can determine their relative position with respect to each other. We will then demonstrate an inter-drone collision avoidance task with 2 pocket drones. We have combine this with wall-avoidance as in Chapter 2, which will enable a team of pocket drones to fly autonomously within an indoor office-like environment, using solely their on-board sensors and computational capabilities.

Parts of this chapter have been published in:

K.N. McGuire, M. Coppola, C. De Wagter & G.C.H.E. de Croon, *Towards Autonomous Navigation of Multiple Pocket Drones in Real-World Environments*, IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) **244-249** (2017)

M. Coppola, **K.N. McGuire**, K.Y.W. Scheper & G.C.H.E. de Croon, *On-board communication-based relative localization for collision avoidance in Micro Air Vehicle teams*, Autonomous Robots **1787–1805** 42-8 (2018)

Contribution: The initial research leading to majority this chapter's work was the result of a MSc. graduation project done by Mario Coppola. He mainly developed the relative localization scheme with Bluetooth and conducted the experiments on the ARdrones. I helped him by porting his method on pocket drones for additional experiments. *Section 3.2 of this chapter has been adopted from the paper of Coppola et al. (2018) to provide the necessary background information for the rest of the chapter.* Although I was the main author of the McGuire et al. (2017)'s paper, Mario helped out significantly with the concept and experiments.

Figure 3.1: Two pocket drones (40 g each) flying autonomously in a real-world indoor environment using only on-board sensing and processing.

3.1. INTRODUCTION

POCKET DRONES are Micro Aerial Vehicles (MAVs) characterized by their low mass and small size. These attributes make them safe for flight near humans and allow maneuvering through narrow indoor areas like corridors, windows, or rooms (as shown in Fig. 3.1). Pocket drones are thus ideal for indoor exploration and surveillance tasks, such as green-house observations or search-and-rescue operations (Scaramuzza et al., 2014). However, real-world applications of pocket drones are limited by the short flight duration and range of a single platform. By using multiple pocket drones together, exploratory tasks would be performed more efficiently and transcend the individual limitations (Brambilla et al., 2013).

In order for teams of pocket drones to perform tasks in indoor spaces, they must be able to avoid collisions with static obstacles *and* with each other. While solutions with a centralized computer or external sensors are possible (e.g. using a motion tracking system (Mulgaonkar et al., 2015) or fixed ultra wide-band beacons (Ledergerber et al., 2015)), they are not applicable to exploration scenarios, where the link to a possible base station can easily be interrupted. Furthermore, the need to set up an external sensor suite would inherently defeat the purpose of the exploration task. It follows that the drones must operate fully autonomously using on-board sensors.

The challenge tackled in this work is to achieve this on real-world pocket drones. This challenge may be broken down into three sub-challenges. The pocket drones must: 1. fly autonomously indoors, 2. detect and avoid obstacles in the environment, and 3. localize and avoid each other. The hardware to achieve this must be small, light-weight, and energy-efficient.

The first and second sub-challenges require an efficient method for own-state estimation and environment detection. Computer vision is often used for such purposes, as it can turn a simple camera into a versatile sensor, capable of measuring multiple variables. Examples are the detection of ego-motion with optical flow, successfully demonstrated by Honegger et al. (2013) and creating 3D maps by stereo-vision, as in Geiger et al. (2011). Many of the latest computer vision techniques do not scale well for small

micro-processors and low resolution cameras. Nevertheless, some efficient methods exists, which are able to run on-board miniature MAVs of 40 grams or less (De Wagter et al., 2014, Moore et al., 2014). Recent work by McGuire et al. (2017), as presented in Chapter 2 demonstrated successful real-world flight by a pocket drone in a room using a light-weight stereo camera. However, the pocket drone was unable to control its own height and, in case of side-wards drift, could collide into obstacles that were not seen by the camera due to its limited Field-Of-View (FOV). The third sub-challenge requires an efficient method for relative localization. In literature, typical methods rely on: high-resolution cameras (Shen et al., 2011) (Conroy et al., 2014), infra-red sensors (Roberts et al., 2009), or mounted microphone arrays (Basiri et al., 2016). Recently, Coppola et al. (2018) have shown that it is possible to use communication between MAVs to achieve relative localization to a sufficient accuracy for collision avoidance (this relative localization scheme will be revisited in this chapter). However, the tests were only performed in a controlled environment. Furthermore, their collision avoidance strategy (a variant of Velocity Obstacle (VO) (Fiorini and Shiller, 1998, Wilkie et al., 2009)) did not account for heading change by a drone.

The main contribution in this chapter is a system for fully autonomous flight by a small group of pocket drones, with active avoidance of static obstacles and other drones. The system is demonstrated in a real-world office with two pocket drones. The work from McGuire et al. (2017) and Coppola et al. (2018), described above, were used as starting points. In this work, we add a binary collision avoidance structure to efficiently store the bearing of static obstacles (as sensed by the camera) and the other drones (as sensed via communication). Furthermore, to allow the drones to control their height and avoid drift, we also introduce a range sensor array. This new light-weight sensor provides accurate ranging data sideways, downwards, and upwards, which allows the drone to control its own height and detect & react on obstacles outside of the stereo camera's FOV.

This chapter is structured as follows. First the method for the communication-based relative localization will be recapped and explained in section 3.2, showing inter-drone avoidance experiments of 2 pocket drones with a downward looking camera. The approach for both obstacle and inter-drone avoidance is explained in detail in section 3.3. The drone's behavior was first tested in simulation, as discussed in section 3.4. Finally, section 3.5 shows the results of the real-world experiments. Section 3.6 provides concluding remarks.

3.2. COMMUNICATION-BASED RELATIVE LOCALIZATION

This section will go into the specifics of relative localization via wireless communication between MAVs, which will be demonstrated with an inter-drone collision avoidance on two pocket drones later on. The MAVs communicate the following states to each other: planar velocity in the body frame, orientation with respect to North, and height from the ground. When communicating, the MAVs can also measure the signal strength; this acts as a measure of distance. For Bluetooth Low Energy (BLE), the technology chosen in our implementation, signal strength measurements are referred to as Received Signal Strength Indication (RSSI). Each MAV fuses the received states, the RSSI, and its own on-board states to estimate the relative position of another MAV. When multiple MAVs are present, each MAV can run multiple parallel instances of the fusion filter so as to keep

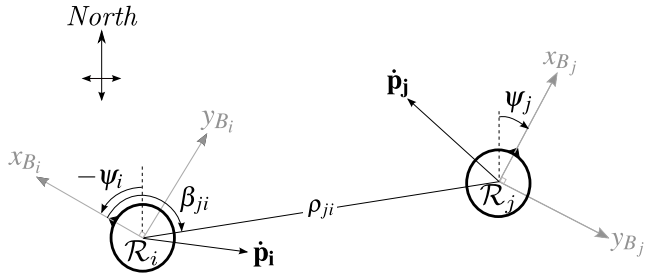
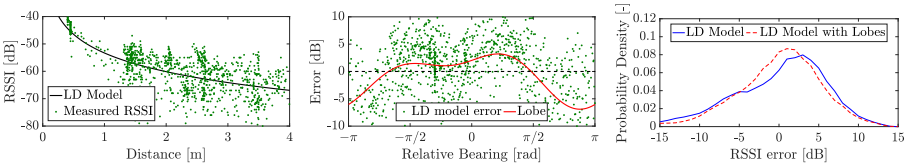


Figure 3.2: Top view of the relative localization framework (x_B and y_B are the planar axis of $\mathcal{F}_B$, while z_B is positive down (conventional aircraft coordinate system of North-East-Up).



(a) RSSI measurements with respect to distance and fitted LD model

(b) Error about LD model with respect to relative bearing fitted with a second order Fourier series

(c) Noise distribution about the LD model without and with lobe effects

Figure 3.3: Results of RSSI measurements during an experiment whereby a Ladybird MAV was carried in circles around a fixed Bluetooth antenna.

track of all others. This section details the design and implementation of the relative localization scheme and presents some preliminary localization results that were obtained in early stages of the research.

3.2.1. Framework Definition for Relative Localization

Consider two MAVs $\mathcal{R}_i$ and $\mathcal{R}_j$ with body-fixed frames $\mathcal{F}_{B_i}$ and $\mathcal{F}_{B_j}$, respectively. We define the relative pose of $\mathcal{R}_j$ with respect to $\mathcal{R}_i$ as the set $P_{ji} = \{\rho_{ji}, \beta_{ji}, z_{ji}, \psi_{ji}\}$, where: ρ_{ji} represents the range between the origins of $\mathcal{F}_{B_i}$ and $\mathcal{F}_{B_j}$, β_{ji} is the horizontal planar bearing of the origin of $\mathcal{F}_{B_j}$ with respect to $\mathcal{F}_{B_i}$, z_{ji} is the height of $\mathcal{R}_j$ with respect to $\mathcal{R}_i$ and ψ_{ji} is the yaw of $\mathcal{F}_j$ with respect to $\mathcal{F}_i$. See Fig. 3.2 for an illustration. Note that ρ_{ji} and β_{ji} are related to their Cartesian counterparts via:

$$\rho_{ji} = \sqrt{x_{ji}^2 + y_{ji}^2 + z_{ji}^2}, \tag{3.1}$$

$$\beta_{ji} = \text{atan}_2(y_{ji}, x_{ji}). \tag{3.2}$$

x_{ji}, y_{ji}, and z_{ji} are the Cartesian coordinates of the origin of $\mathcal{R}_j$ in $\mathcal{F}_{B_i}$.

3.2.2. SIGNAL STRENGTH AS A RANGE MEASUREMENT

Let S_{ji} be the RSSI measurement in dB. It is correlated with ρ_{ji} by a function $\mathscr{L}(\rho_{ji})$. We define this function based on the Log-Distance (LD) model (Seybold, 2005):

$$S_{ji} = \mathscr{L}(\rho_{ji}) = P_n - 10 * \gamma_l * \log_{10}(\rho_{ji}). \tag{3.3}$$

P_n is the RSSI at a nominal distance of 1 m. γ_l is the *space-loss parameter*, which dictates how much the signal strength decays with distance (for free-space: $\gamma_l = 2.0$). [1] The LD model is assumed subject to Gaussian noise (Svečko et al., 2015).

In preliminary tests, we analyzed the LD model with a Ladybird MAV (Remes et al., 2014) (similar as the first platform used in Chapter 2) connected via Bluetooth to a fixed W1049B omni-directional antenna.[2] The MAV was carried in concentric circles at different distances around the antenna whilst RSSI was being recorded with the antenna. Its orientation with respect to North was kept constant, thus varying the relative bearing to the antenna. Ground Truth data was recorded with an Optitrack Motion Capture System. The results from a representative data sample are shown in Fig. 3.3, to which the LD model was fitted using a non-linear least squares estimator as in Fig. 3.3a. Among a set of similar experiments, the Standard Deviation (SD) of the error about the fitted LD model was found to be between 3 dB and 6 dB. This is in line with literature (Nguyen and Luo, 2013, Szabo, 2015).

We also observed a change of the error with the relative bearing. This is shown in Fig. 3.3b, and accounts for the skew in error distributions, see Fig. 3.3c. The disturbances that can cause this could be: uneven directional propagation lobes, interference by the reflection of the signal in the environment, the presence of other signals in the 2.4 GHz spectrum, or other objects that obstruct the signal (Caron et al., 2008, Kushki et al., 2008, Seybold, 2005, Svečko et al., 2015, Szabo, 2015). Such disturbances could be dependent on the environment or on the relative bearing between antennas, both of which are unknown during an exploration task. For this reason, the LD model was not expanded to include this dependency on bearing.

3.2.3. LOCALIZATION VIA FUSION OF RANGE AND ON-BOARD STATES

Achieving a relative pose estimate requires measuring or inferring all four variables in P_{ji}. We can directly measure or observe the following three:

- ρ_{ji} (range), available via RSSI as in Sec. 3.2.2.
- z_{ji} (relative height). Each MAV is expected to measure its height above the ground. This could be done with a pressure sensor (Beard, 2007, Sabatini and Genovese, 2013, Shilov, 2014), sonar, or a downward-facing camera (Kendoul et al., 2009a,b, McGuire et al., 2017). Two MAVs $\mathscr{R}_i$ and $\mathscr{R}_j$ can share their altitude data, such that: $z_{ji} = z_j - z_i$.

[1] Experimentally, it has been found that office buildings can feature $2 \leq \gamma_l \leq 6$ (Kushki et al., 2008). Performing a sensitivity analysis of the LD model shows that an accurate identification of γ_l has a low impact on the distance estimate at small distances.

[2] Pulse, W1049B Datasheet version 1.1, www.cdiweb.com/datasheets/pulse/W1049B.pdf

- ψ_{ji} (relative orientation). It is assumed that all MAVs acknowledge a common planar axis (e.g., magnetic North (Afzal et al., 2011, No et al., 2015)). Through communication, the MAVs exchange their orientation data.

Relative bearing is the only unknown variable. It becomes observable when fusing the three measurements above with velocity measurements (Martinelli and Siegwart, 2005, Martinelli et al., 2005). [3] We chose to perform sensor fusion with a discrete-time Extended Kalman Filter due to its efficient processing and memory requirements (De Silva et al., 2014). The filter uses Cartesian coordinates so that it can directly take the difference between velocities in each axis. The state transition model from time-step k to $k+1$ was defined as in Eq. 3.4.

$$\begin{bmatrix} \vec{p}_{ji} \\ \dot{\vec{p}}_i \\ \dot{\vec{p}}_{jRi} \\ \psi_j \\ \psi_i \\ z_j \\ z_i \end{bmatrix}_{k+1} = \begin{bmatrix} \vec{p}_{ji} + \left(\dot{\vec{p}}_{jRi} - \dot{\vec{p}}_i\right)\Delta t \\ \dot{\vec{p}}_i \\ \dot{\vec{p}}_{jRi} \\ \psi_j \\ \psi_i \\ z_j \\ z_i \end{bmatrix}_k + \vec{v}_k \tag{3.4}$$

$\vec{p}_{ji} = \begin{bmatrix} x_{ji} & y_{ji} \end{bmatrix}^T$ holds Cartesian equivalents of bearing and range. $\dot{\vec{p}}_i = \begin{bmatrix} \dot{x}_i & \dot{y}_i \end{bmatrix}^T$ is a vector of the velocity of $\mathscr{R}_i$ in $\mathscr{F}_{B_i}$ (see Fig. 3.2). $\dot{\vec{p}}_{jRi}$ is $\dot{\vec{p}}_j$ rotated from $\mathscr{F}_{B_j}$ to $\mathscr{F}_{B_i}$. Δt is a discrete time-step between updates, equal to the time between k and $k+1$. $\vec{v}_k$ represents the noise in the process at time-step k. This model assumes that all current velocities and orientations remain constant between time-steps. The observation model for the EKF is given by Eq. 3.5.

$$\begin{bmatrix} S_{ji} \\ \dot{\vec{p}}_i \\ \dot{\vec{p}}_j \\ \psi_j \\ \psi_i \\ z_j \\ z_i \end{bmatrix}_k = \begin{bmatrix} \mathscr{L}(\rho_{ji}) \\ \dot{\vec{p}}_i \\ \mathbf{R_{2D}}(\psi_{\mathbf{ji}}) * \dot{\vec{p}}_{jRi} \\ \psi_j \\ \psi_i \\ z_j \\ z_i \end{bmatrix}_k + \vec{w}_k \tag{3.5}$$

$\mathbf{R_{2D}}(*)$ is a 2D rotation matrix that uses the relative heading ψ_{ji} to rotate the state estimate $\dot{\vec{p}}_{jRi}$ from $\mathscr{F}_{B_i}$ to $\mathscr{F}_{B_j}$. $\vec{w}_k$ represents the noise in the measurements at time-step k. Note that ρ_{ji} is expanded as per Eq. 3.1 so as to observe x_{ji} and y_{ji}.
The EKF cannot be initialized with a correct relative location estimate, since this is not known; it must converge towards the correct value during flight. Appropriate tuning of the EKF noise covariance matrices is key to achieving this. In the EKF, the measurement

[3] An intuitive explanation of the observability is as follows: if robot $\mathscr{R}_i$ is moving towards North with 1 m/s and its distance to $\mathscr{R}_j$ (which, in this example, remains stationary) increases by 1 m each second, then $\mathscr{R}_j$ could infer that $\mathscr{R}_i$ is to its North. Similarly, $\mathscr{R}_i$ would know that $\mathscr{R}_j$ is to its South. This logic can be extended for all directions.

noise matrix **R** is a diagonal matrix with the form shown in Eq. 3.6.

$$\mathbf{R} = \begin{bmatrix} \sigma_m^2 & & & \\ & \sigma_v^2 * \mathbf{I_{4\times4}} & & \\ & & \sigma_\psi^2 * \mathbf{I_{2\times2}} & \\ & & & \sigma_z^2 * \mathbf{I_{2\times2}} \end{bmatrix} \tag{3.6}$$

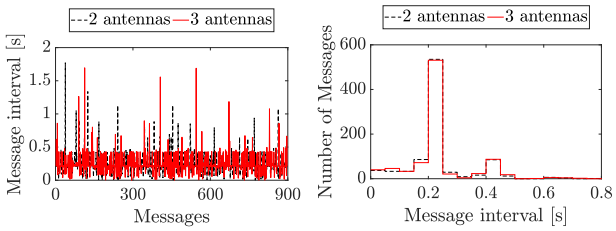
σ_m is the assumed SD of S_{ji}. σ_v is the assumed SD of $\dot{\vec{p}}_i$ and $\dot{\vec{p}}_j$. σ_ψ is the assumed SD of the magnetic orientation measurements. σ_z is the assumed SD of the height measurements. $\mathbf{I_{n\times n}}$ is a $n \times n$ identity matrix. Based on our preliminary RSSI noise analysis, σ_m is tuned to 5 dB. Throughout this chapter, all other SDs were tuned to 0.2, unless otherwise stated. This was based on the measurement noise, either simulated or expected from the sensors.

The process noise matrix **Q** is the diagonal matrix presented in Eq. 3.7.

$$\mathbf{Q} = \begin{bmatrix} \sigma_{Q_p}^2 * \mathbf{I_{2\times2}} & & & \\ & \sigma_{Q_v}^2 * \mathbf{I_{4\times4}} & & \\ & & \sigma_{Q_\psi}^2 * \mathbf{I_{2\times2}} & \\ & & & \sigma_{Q_z}^2 * \mathbf{I_{2\times2}} \end{bmatrix} \tag{3.7}$$

σ_{Q_p} is the SD of the process noise on the relative position update. σ_{Q_v}, σ_{Q_ψ}, and σ_{Q_z} are SDs for the expected updates in velocity, orientation, and height, respectively. By tuning **Q** we can define the validity of the process equations (Malyavej et al., 2013). In this chapter, unless otherwise stated: $\sigma_{Q_p} = 0.1$, while $\sigma_{Q_v} = \sigma_{Q_\psi} = \sigma_{Q_z} = 0.5$. We tuned σ_{Q_p} to 0.1 so as to have a relatively low process noise on the relative position update. This forces the filter to rely less on the (noisy) range measurements and more on other data, which encourages convergence and helps discard the high noise and disturbance in the RSSI measurements. σ_{Q_v}, σ_{Q_ψ}, and σ_{Q_z} were then tuned higher (to 0.5) to enhance the difference, while staying within the order of magnitude of the expected standard deviations of the measurements.

This filter is limited by flip and rotation ambiguity as defined by Cornejo and Nagpal (2015). When the motion of $\mathscr{R}_j$ perfectly matches the motion of $\mathscr{R}_i$, range-only measurements remain constant and are not informative for bearing estimation. Unless the



(a) Message Intervals **(b)** Distribution

Figure 3.4: Messaging rate over a test flight.

MAVs are flying in formation, the probability of this event will be low (Cornejo and Nagpal, 2015). The same ambiguity takes place when both $\mathscr{R}_i$ and $\mathscr{R}_j$ are static. Motion by at least one MAV is required, as the filter operates by taking the difference in velocity. The performance of the filter thus increases as the average difference in velocity between the MAVs increases (and/or the accompanying measurement noise decreases).

3.2.4. IMPLEMENTATION DETAILS AND TESTING THE ON-BOARD LOCALIZATION ON POCKET DRONES

We used BLE to enable communication between the MAVs. The data is sent and received by means of advertising messages scheduled using a Self-Organized Time Division Multiple Access (STDMA) algorithm, as described by Gaugel et al. (2013). This enables ad-hoc communication and circumvents the Master-Slave paradigm otherwise enforced by the BLE standard (Townsend et al., 2014), as each antenna alternates between advertising and listening. The messaging rate is tuned to 5 Hz, which is a compromise between the amount of STDMA communication slots (8 slots) and an acceptable communication rate. 5 Hz keeps the congestion low. Nevertheless, the messaging rate can be affected by differences in clock rates and possible packet losses. Fig. 3.4 shows the interval between received messages over approximately 3 minutes of recording from the point of view of a single antenna in a group of two or three participating antennas. Approximately 80% of messages are received and parsed within 0.25 s and 95% within 0.45 s. A slight increase in packet loss was observed when increasing the number of antennas to three, with 128 missed messages as opposed to 120 when two antennas were used. This corresponds to an increase in packet loss from 13.4% to 14.2%.

An exploration task was developed where multiple MAVs fly in a room at the same altitude and attempt to pass through the center. It was designed to provoke collisions, which the drones would handle by means of the collision-cone avoidance tactic as explained in section 4 of Coppola et al. (2018), The sample task was used to test the performance of the relative localization and collision avoidance, separately and combined. In section 7 of Coppola et al. (2018), experiments were done up to 3 AR.Drones, who were communicating by separate Bluetooth modules. To show that the proposed solution scales to smaller MAVs, we ported the technology to pocket drones. A test-ready MAV and its components are shown in Fig. 3.5. The platform is similar to the pocket drone used in Chapter 2 and McGuire et al. (2017), with an addition of the Bluetooth module.[4] The conclusion of experiments done with the pocket drones in Coppola et al. (2018) is that it did result in a slightly lower performance than the AR.drones. This is due to the worse velocity estimation (lower resolution camera), to the fact that two communication modules (WiFi and Bluetooth) were right next to each other and to the smaller flying space. Nonetheless, the pocket drones were able to adequately avoid each other and therefore we used the same Bluetooth-based relative positioning explained in this section for the remainder of this chapter.

[4]Videos of the experiments are available at: https://www.youtube.com/playlist?list=PL_KSX9GOn2P9f0qyWQNBMj7xpe1HARSpc.

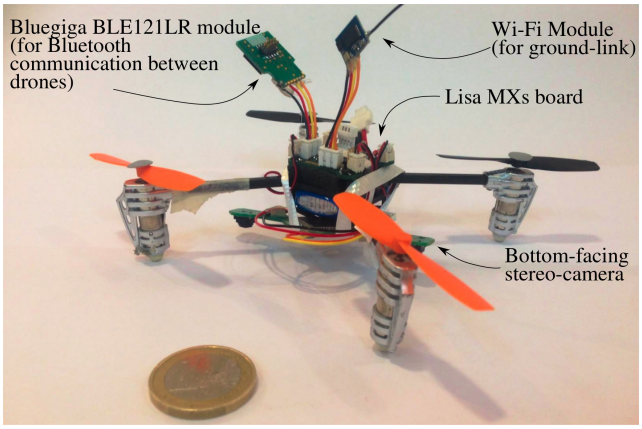


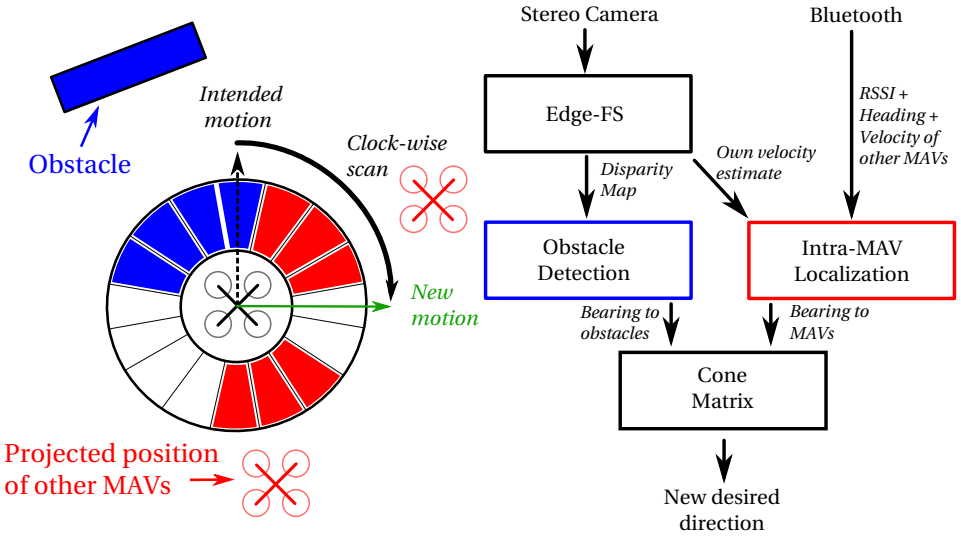
Figure 3.5: Pocket-drone used in the experiments.

3.3. METHOD OBSTACLE AND INTER-DRONE AVOIDANCE

This section explains the separate sub-systems that were implemented on the pocket drone to enable inter-drone and obstacle avoidance. To efficiently select a safe flight direction, we implemented a binary disk array where directions are marked as either safe or unsafe. This is as a substitute of the collision cone-based strategy used in the last section. This array will be referred to as Collision Disk. It is depicted in Fig. 3.6a. The disk is continuously updated using information coming from both the stereo-camera and the inter-MAV localization, as in Fig. 3.6b.

3.3.1. VELOCITY ESTIMATION AND STATIC OBSTACLES DETECTION

Velocity estimation and static obstacle detection can be performed simultaneously with a stereo-camera running Edge-Flow Stereo (Edge-FS). Edge-FS, earlier explained in Chapter 2, stems from the work in Lee et al. (2004). It was followed up by McGuire et al. (2016) and applied for an autonomous flight of a pocket drone in McGuire et al. (2017). This computer vision algorithm is efficient thanks to the use of edge distributions. Its working principle is depicted in Fig. 3.7. First, the gradients of an image are computed using a Sobel filter. Then, these gradients are compressed to an edge distribution. This can be either compared to one of a previous time-step, to compute optical flow (Edge-Flow), or with a stereo camera, to compute depth (Edge-Stereo). By scaling Edge-Flow with Edge-Stereo we can estimate velocity along all axes of the drone's coordinate system in North-East-Down (NED). Using Edge-Stereo, a disparity map can be used to also detect obstacles such as walls or objects (within the FOV of the stereo-camera). If the distance to the obstacle is below a threshold, it is added to the Collision Disk with the angle relative to the current heading of the pocket drone (see Fig. 3.6a). This work currently only considers the closest obstacle.



(a) Visualization of the Collision Disk.

(b) Update flow diagram.

Figure 3.6: Schematic explanation of Collision Disk.

3.3.2. POCKET-DRONE RELATIVE LOCALIZATION

In this Chapter, we use the relative localization method explained in Section 3.2 and Coppola et al. (2018), achieving inter-drone localization via communication between the pocket drones. The height, and the estimated velocity in the horizontal plane (as from Edge-FS) are communicated between drones while the RSSI is measured. Albeit coarse (Root Mean Squared Error (RMSE) was of $\approx 0.8rad$), the method efficiently provides data on un-safe flight directions to include in the Collision Disk. It was experimentally demonstrated that the accuracy is sufficient for collision avoidance even in small rooms.

A finding from Coppola et al. (2018) was that too high cautiousness in collision avoidance leads to a restriction in motion, which is ultimately detrimental. To avoid this, the collision cone only considers drones at an estimated distance below a threshold d_{drone}. To account for the relative motion of the drones, the Collision Disk does not directly use the estimated location, but a projected location of the other MAV a certain time into the future (Fig. 3.8). This is based on the angular velocity of the drone, such that

$$\Delta\beta \approx \kappa_t \cdot \frac{v_\perp}{d}, \tag{3.8}$$

where: d is the distance to another drone; $v_\perp$ is the perpendicular velocity of the moving drone about the observing drone; and κ_t is a factor equal to the amount of seconds in the future that are estimated. In its current implementation, κ_t is set manually. In all the experiments in this chapter $\kappa_t = 1$.

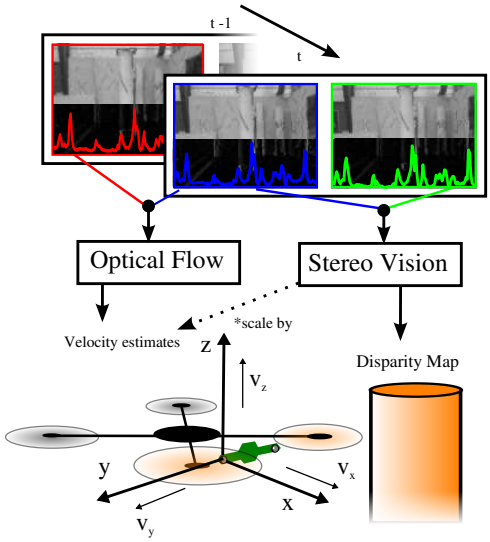


Figure 3.7: Schematic explanation of Edge-FS.

3.3.3. HEIGHT CONTROL AND DRIFT COMPENSATION

The pocket drones will be equipped with four range-sensors, pointed toward the top, bottom, right and left (see Fig. 3.9). The top and bottom range-sensors will be used for basic height control, where the option exist to transverse along the ceiling of an indoor environment. For the sides it will act like a fail safe in case the pocket drone drifts towards an obstacle outside of the stereo-camera's FOV. Here a simple force-field principle is applied with an inner- and outer-border, as illustrated in Fig. 3.9. If the drones drift to a distance between the obstacle and the outer-border, it will get an extra velocity command to get out of this situation. From here on, the magnitude of this command is linearly dependable on the distance between pocket drone and obstacle, which is bounded with a maximum velocity command from the inner-border on. For the experiments, the

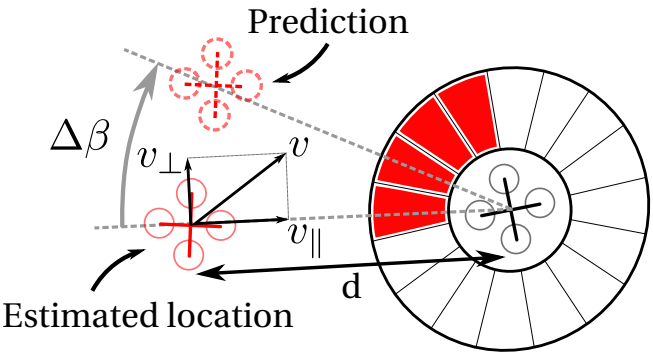


Figure 3.8: MAV location prediction in the Collision Disk

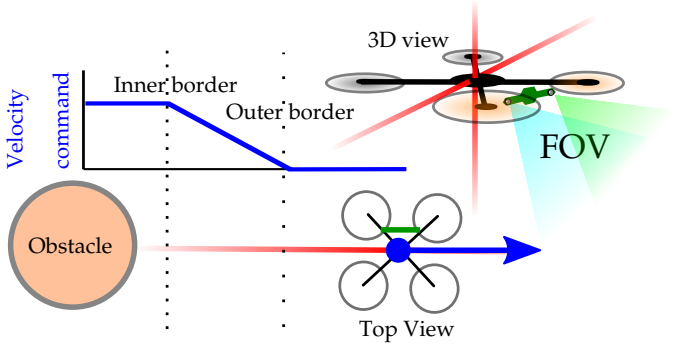


Figure 3.9: Visualization about the forcefield generated by the range sensors where the red lines illustrate their direction of gaze.

inner- and outer border are set on 0.8 and 1.2 meters respectively and the maximum bounded velocity is 0.3 m/s.

3.3.4. BEHAVIOR

To navigate within the FOV of the camera, the MAVs should always be in forward flight so that they can visually detect obstacles (e.g. walls). The avoidance behavior then performs the following operations, in order: 1. velocity is reduced to zero, 2. the MAV rotates to face a new direction that it deems safe, 3. the MAV resumes forward flight in the new direction. If the MAV is flying towards a region marked as unsafe, it will stop and turn clock-wise until it is facing a direction marked *safe*. This is depicted in Fig. 3.6a.

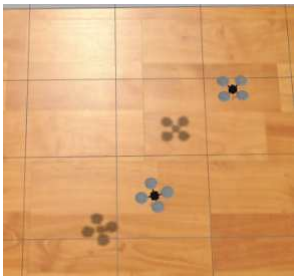
3.4. SIMULATION

Prior to real-world tests, the behavior was tested in simulation using the Robotics Operating System (ROS)[5] and the *hector-quadrotor* simulator within Gazebo[6] (Fig. 3.10a). The simulated MAVs fly at the same height in an arena. The simulated MAV diameter was $0.2m$. The RSSI noise and lobes were set to $5dB$, which is similar to the real-world Bluetooth performance. No simulated range-sensors and stereo-camera were used here, so the height and velocity were taken directly from the ground truth. The velocity and height estimation error were set at $0.2m/s$ and $0.2m$, respectively. The arena was $6m \times 6m$.

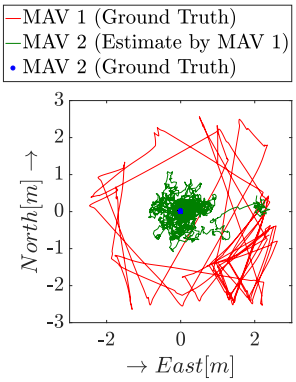
In the first test, one drone was held static while the other was let lose in the space. Repeated simulations of 500 s showed no collisions. The log of a simulation is shown in Fig. 3.10b. The moving MAV could successfully combine knowledge of the other drone and the walls to choose a safe path.

To test out the scalability of the method, trials were conducted with 2 and 3 MAVs in the same arena. The threshold distance from the drone to the wall was $1.2m$. The maximum trial duration was $500s$, but an inter-drone collision will end the trial prema-

[5]Robot Operating System (ROS), http://www.ros.org/
[6]Gazebo, http://gazebosim.org/



(a) Screen-shot of a simulation trial.



(b) Simulation log with a static drone over a 500s flight.

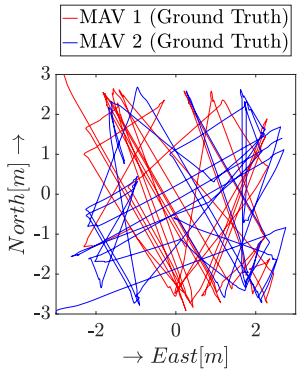
Figure 3.10: Set-up of simulation environment.

	No Avoidance	$d_{drone} = 2m$	$d_{drone} = 5m$
2 MAVs	194s & 10 coll.	266s & 8 coll.	421s & 4 coll.
3 MAVs	66s & 10 coll.	103s & 9 coll.	177s & 10 coll.

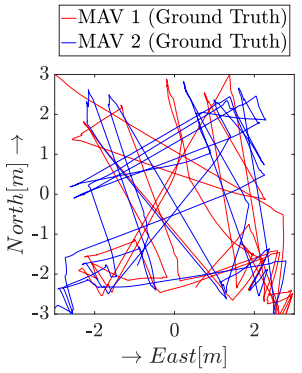
Table 3.1: Simulation statistics (mean flight time and number of collisions). 10 simulated flights were run for each parameter pair.

turely. d_{drone} was set to $2m$ (Fig. 3.11a and 3.11c) and $5m$ (Fig. 3.11b and 3.11d). For each parameter pair, 10 trials were run. An overview of the results can be found in Tab. 3.1. For the 2 and 3 MAVs scenarios, the average flight times (over 10 flights) increased with $d_{drone} = 5m$ instead of $2m$ and the number of collisions went down, except for 3MAVs with d_drone of 5 meters. However, the trajectories (see Fig. 3.11) show that increasing d_{drone} restricts freedom of movement. In 3.11d), MAV 1 and MAV 3 were stuck in one corner and could not move out of their position. To favor unrestricted movement, a smaller d_{drone} would be preferred, although this inherently increases collision risk.

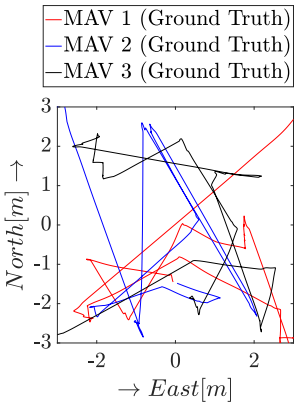
Overall, the results from this behavior do not improve the average flight-times if compared to the simulated results from Coppola et al. (2018). One of the added complication is that now the MAV behaviour is highly non-holonomic, as it constantly stops to change its own heading. Instead, Coppola et al. (2018) simply changed direction of flight. Stopping to change heading decreases the accuracy of the relative localization, which relies on motion. Finally, adding the real wall detection to the Collision Disk adds further complexity. Nevertheless, the simulation results show that 2 pocket drones can fly within a room for the almost full duration of their battery (approx. $5min$), which will be demonstrated in a real-world environment in the next section.



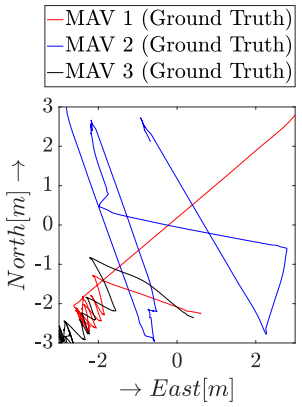
(a) 2 MAVs with $d_{drone} = 2$. Flight duration = 500s.



(b) 2 MAVs with $d_{drone} = 5m$. Flight duration = 500s.



(c) 3 MAVs with $d_{drone} = 2m$. Flight duration = 163s.



(d) 3 MAVs with $d_{drone} = 5m$. Flight duration = 437s.

Figure 3.11: Simulation logs with 2 and 3 MAVs for $d_{drone} = 2m$ and $d_{drone} = 5m$.

3.5. REAL-WORLD EXPERIMENTS

The system was implemented in real pocket drones. In this section, the hardware and software specifics are presented and the experiments are shown of the pocket drones flying autonomously in a real-world environment.

3.5.1. HARDWARE AND SOFTWARE SET-UP

A single pocket drone consists of a Lisa-MXs auto-pilot module (a smaller variant of the Lisa-MX[7]), similar to the one used in McGuire et al. (2017) (see Fig. 3.12). It carries an STM32F4 microprocessor with a speed of 168MHz and 1MB of flash memory. The 4 gram stereo-camera also features an embedded STM32F4 microprocessor with a speed of 168MHz and 196kB of memory in which the largest consecutive memory block spans

[7] Lisa MX Paparazzi Wikipedia manual, http://wiki.paparazziuav.org/wiki/Lisa/MX

128 kB. The processed images are $128 \times 96 px$ and the camera has a $57.4 \times 44.5 deg$ FOV. With this hardware, the Edge-FS algorithm can run in parallel with the regular flight controllers of the Lisa-MXs. Everything is mounted on a Walkera QR LadyBug quad-copter frame.[8] The inter-drone communication and RSSI measurement is done by a Bled112 Bluetooth smart dongle[9] (as used in Coppola et al. (2018)). For testing and validation purposes, an ESP-09 WiFi module was used to broadcast high-speed telemetry to the ground computer. It was not used to send any prior information about the testing area as the pocket drones interprets the environment (by Edge-FS) and perform the inter-MAV localization (RSSI) all on-board.

Finally, a $0.2mm$ thick, $7mm$ wide and $88mm$ long Polyimid Flex-PCB with four VL53L0X Time-of-Flight ranging sensor[10] was designed. The flexible board is bent into a ring and attached to the pocket drone resulting in ranging sensors pointing towards the sides and towards the bottom and the ceiling. By configuring the range sensors into *long range mode*, they can measure an absolute range up to $2m$ at $8Hz$. A local *ATmega328P-MLF28* microcontroller interfaces with all the sensors and sends the combined measurements to the Lisa-MXs over a single wire. The total weight of the board is $0.25g$.With everything combined, the MAV's total mass is approximately 43 gram (including a 11 gram battery).

The auto-pilot program flashed on the Lisa-MXs is Paparazzi UAV.[11] All algorithms and controllers of the software runs entirely on the microprocessor. The basic low levels controllers regulate the attitude of the pocket drone. On top of this, a PID guidance controller coordinates the MAV's velocity in the X (forward) and Y (sideways) direction. In this chapter, the velocity estimated in the X-Y-Z is given by Edge-FS and is actively controlled in the horizontal plane. Since the range sensors provide an accurate position of the altitude of the pocket drone, it can maintain a fixed height for the duration of the flight. The side range sensors will not be used for the main navigation, since individually they have a very narrow receptive angle, but will act as a velocity force field. If the pocket drone comes too close to a wall or obstacle at the sides — which is beyond the FOV of the camera — it will give an opposite velocity commands that is added to the one from the main navigation and steers the MAV away from the lateral obstacle. In the experiments, only one pocket drone (PD1) is equipped with the range sensors and the other one is not (PD2).

3.5.2. EXPERIMENT RESULTS

The experiments with two pocket drones were conducted in a real-world environment: an office at the faculty of Aerospace Engineering from the Delft University of Technology (Fig. 3.13). This office's dimensions are $5.0 \times 4.0 \times 2.7m$ in length, width, and height, respectively. The office features varying types texture as commonly found in such areas. The glass cabinets were given a bit of additional coverage as this is still a difficult scenario

[8] Walkera QR LadyBug quad-copter, http://www.walkera.com/

[9] Scilabs Bled112 Bluetooth smart dongle, http://www.silabs.com/products/wireless/bluetooth/bluetooth-low-energy-modules/ble121lr-bluetooth-smart-long-range-module

[10] STMicroElectronics, VL53L0X Time-of-Flight ranging sensor, http://www.st.com/content/st_com/en/products/imaging-and-photonics-solutions/proximity-sensors/vl53l0x.html

[11] PaparazziUAV Wikipedia manual, http://wiki.paparazziuav.org/

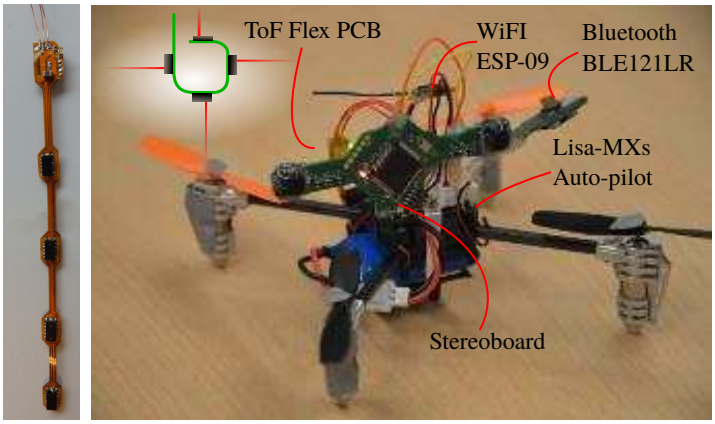


Figure 3.12: Picture of the pocket drone and its hardware.

Figure 3.13: Panorama of testing site.

for Edge-FS as well for the proximity sensors. Four infrared OptiTrack cameras[12] were placed near the ceiling of the room to measure a (sparse) trajectory for determining their coverage, but is used for post flight analysis only.

The pocket drones started out with a manual take-off, from which they switch to autonomous control mid-air. The thresholded distance for both the range for the other MAV as the obstacles was set to $1.5m$. The pocket drone (PD1) carrying the range sensors, would start out first as it is able to maintain its own attitude (which it maintains at $1.5m$, taking the ceiling as a reference). Once PD1 is flying autonomously for a few seconds, the second MAV (PD2) takes off and switched to guided mode. As the drone does not contain the range sensor PCB ring, its height has to be controlled manually. In the horizontal plane however, the same avoidance logic for the turning exists as with the first drone, with some extra velocity guidance of the remote control. This means that only the preferred velocity is given by the remote control, from which it has to match with its own velocity estimate of Edge-FS, and not the exact angle set points as with the

[12] OptiTrack, http://www.optitrack.com/

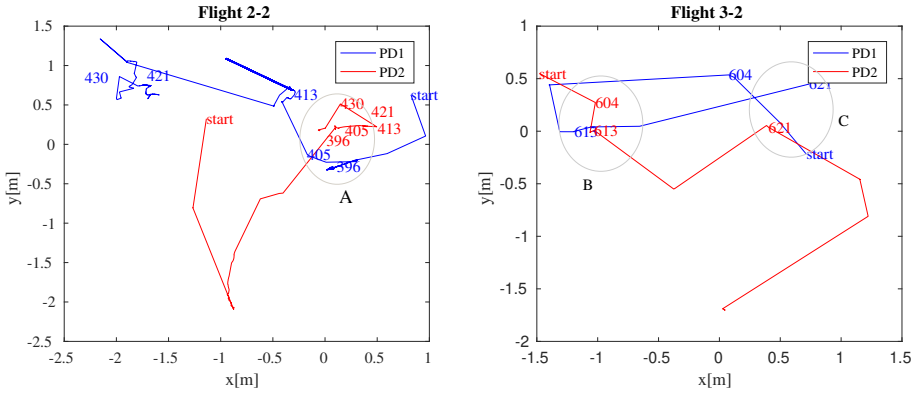


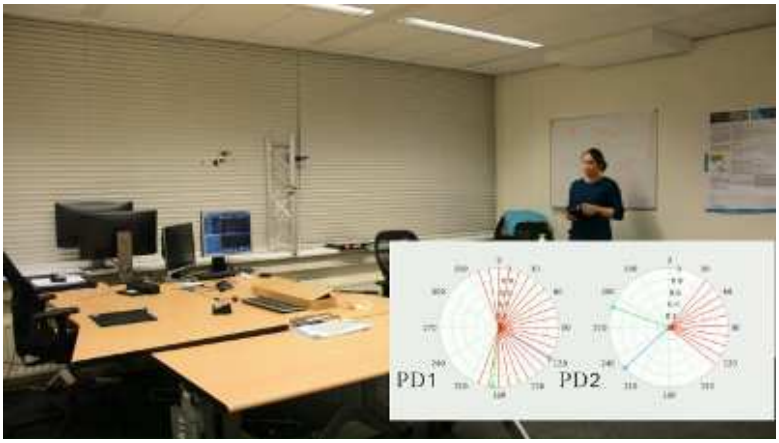
Figure 3.14: Top view flight tracking of flight 2-2 and 3-2

common attitude manual control. However, the difficulty in guiding PD2, is that it is controlling its own heading based on obstacles and the location of PD1. Since this and its velocity control (given by the remote control) are decoupled, the safety pilot had a hard time controlling PD2, resulting in a longer average flight duration for the full autonomous PD1.

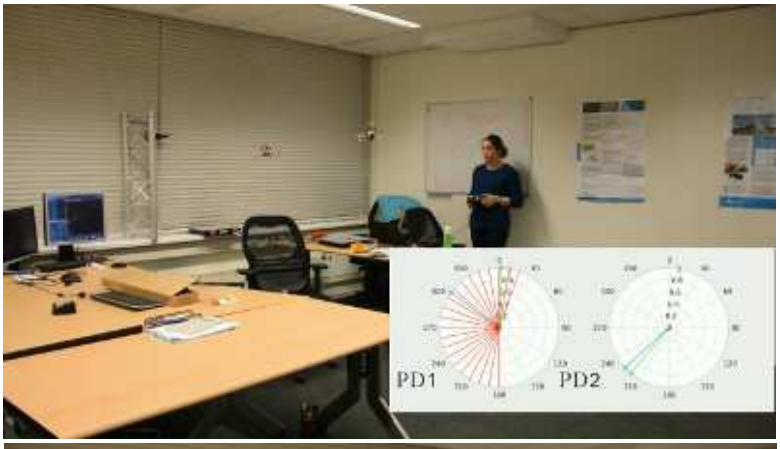
Based on the full duration of PD1's flight, 4 tests were performed, with a duration of $119s$, $311s$, $321s$ and $103s$ respectively. PD1 crashed in the 1^{st} flight because of an undetected static obstacle. In the 2^{nd} and 3^{rd} flight, PD1 flew autonomously until the end of its battery life. In the 4^{th} flight, PD1 was caught in between two pillars and facing a wall, and was unable to escape from this dead-lock. For now, this chapter will focus on flight 2 and 3 specifically because of their length, as more inter-MAV collision avoidance situations can be analyzed. The flights contain multiple restarts of PD2, therefore can be split up in sub-flights (2-1 to 2-3 and 3-1 to 3-4). In Fig. 3.14, the trajectory by the OptiTrack cameras is shown for flight 2-2 and 3-2. The motion capture system was undersized for full room coverage, and was thus unable to track the drones the entire time due to occlusion and their small size. This resulted in some artifacts in the tracked position. However, some moments were identified where the pocket drones came in proximity of each other, which are annotated in the plots of Fig. 3.14.

We shall discuss three representative scenarios, which have been recorded on video and their screen-shots can be found in Fig. 3.15. These images show the moments of close proximity, as annotated in Fig. 3.14. The first screen-shot shows Scenario A, where both drones were able to see each other and changed their heading accordingly. In the Scenario B, PD2 failed to locate PD1 correctly as it is not shown in the Collision Disk. However, PD1 successfully detected PD2 and changed its heading to an obstacle free direction. In the last screen-shot, Scenario C, PD2 did see PD1 (as indicated in the Collision Disk), and only had to adjust its heading slightly. PD1 did also detect PD2, but since it is not on its planned trajectory it did not go into an evasive maneuver at first. However, as it was heading towards an obstacle, it will plan a turn into an collision-free direction. All

A)



B)



C)

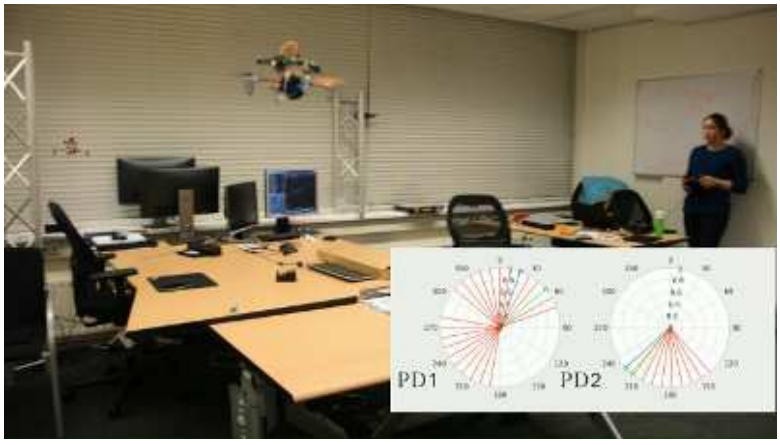


Figure 3.15: Screen shots of flight 2-2 at A) 2:21 sec and flight 3-2 at B) 2:46 and C) 2:54 sec.

video recorded flights can be find in a dedicated YouTube play-list.[13]

3.6. CONCLUSION

To the best of our knowledge, this chapter presents the first attempt to fly a fully autonomous team (duo) of pocket drones in a real-world environment. We combined state-of-the-art methods for own-state estimation and inter-drone tracking for pocket drones, and added additional range sensors to control height and side-ways drift. With this set-up, the pocket drones can achieve stable flight. Using a binary structure called Collision Disk, they could efficiently select collision free paths (from static obstacles and other drones) while exploring their environment. The experiments showed that the pocket drones made the right maneuvers at close proximity of each other. By means of simulation, there are indications that this method can scale to teams of three or more drones. However, the accuracy of the relative localization and the avoidance behavior needs to be developed further in order to achieve this successfully. Nevertheless, this work takes a step closer towards achieving a team of pocket drones, which is able navigate indoor without any external sensors or prior knowledge of the environment.

[13] https://www.youtube.com/playlist?list=PL_KSX9GOn2P9CdtNz_p_tv1cpEs1E3KUx

II

High Level Navigation

4

BUG ALGORITHM LITERATURE SURVEY

In part I, we have covered the low-level-navigation of the pocket drones, for the individual drone in Chapter 2 and for multiple drones in Chapter 3. Now we will take a step towards a higher level of navigation. In the introduction we covered several high-level navigation methods, including several SLAM and insect-inspired navigation techniques, some of which we have also tried to implement ourselves in van Dalen et al. (2018). However, these algorithms were still not fit to be implemented on-board the limited autopilot of the pocket drone as they still require significant memory and processing requirements.

In this chapter, we will dive into a potential alternative, called Bug Algorithms. The general idea is that the robot will have a goal position, while not knowing anything about the obstacles/walls on its way. It will start moving towards its goal, and if it encounters an obstacle, it will follow its boundary until it comes across the possibility to move towards the goal again. Such a strategy only requires limited processing and memory capabilities and is therefore promising for pocket drones. In this chapter, we will provide a survey of Bug Algorithms, their current implementation on robotic platforms and simulation tests which will assess their ability to deal with real-world sensor noise. The conclusion of this chapter's study is that existing bug algorithms rely too much on perfect positioning. The performance drops dramatically once realistic estimation drifts are included. Based on the results, we will develop a novel bug navigation strategy in Chapter 5.

Parts of this chapter have been submitted to
K McGuire, G.C.H.E. de Croon & K. Tuyls , *A Comparative Study of Bug Algorithms for Robot Navigation.*, Robotics and Autonomous Systems **103261**, 121 (2019).

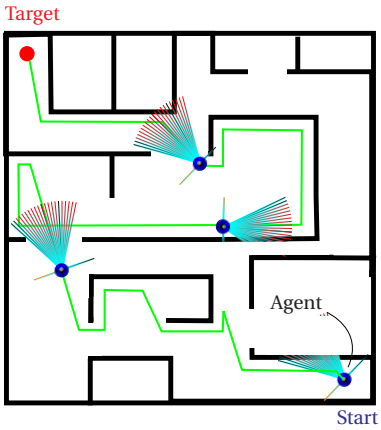


Figure 4.1: An example of an agent performing a Bug Algorithm-like behavior, while navigation in an indoor environment. From a starting position (bottom-right), it moves towards the target (top-left), where it tries to move towards the target whenever it can and follows the obstacles' boundary when it hits an obstacle. Its trajectory is given in green.

4.1. INTRODUCTION

ROBOTIC indoor navigation of robots has been a sought-after topic for the last few decades within the robotic community. An important stimulus for this interest is its potential for a wide range of scenarios, e.g. search-and-rescue, greenhouse observation, industrial inspection. Indoor navigation also comes with a wide range of issues, such as the absence of a reliable GPS-signal and wall interference in long-range communication. An indoor robot should preferably be autonomous and able to navigate based on its on-board sensors and computational capacity.

There has been tremendous progress in autonomous robotic navigation, up to a point that some researchers believe this to be an already solved problem. With the emerging autonomous cars, simultaneous localization and mapping (SLAM) has reached high maturity in development (see Bresson et al. (2017) for a review). SLAM is a notoriously complex and expensive algorithm, consuming much of the robot's on-board processing power. To strive towards computationally efficient methods is advantageous for any robot, but it becomes vital when the application requires the use of tiny, light-weight robots. For instance, small Micro Aerial Vehicles (MAVs), in the order of 50 grams and 15 cm diameter, could be ideal for exploring small and confined spaces. However, their on-board computational resources are so limited that they cannot make use of the current SLAM methods.

Given these strict computational requirement for tiny robotic platforms, an important question is raised: does the actual simple principle of navigation, *going from point A to point B*, need the computational and memory requirements for constructing and maintaining high-resolution metric maps? Should the complexity of the strategy not be proportional to the simplicity of the task?

There are several light-weight alternatives to SLAM to consider, such as Topological

SLAM (see Boal et al. (2014) for a review). Biologically inspired techniques like the Snapshot Model (Cartwright and Collett (1983)) and the Average Landmark Vector (Lambrinos et al. (2000)) can also be considered. These efficient methods, however, still have the tendency to scale up the memory requirements, when navigating in more complex and large environments.

In this chapter, we will look at a navigation method of a different kind: *Bug Algorithms*. Although the name suggests a biological origin, it is a path-planning technique that evolved from maze-solving algorithms. The main principle of Bug Algorithms is that they do not know the obstacles in their environment and only know their target's relative position. They will locally react only upon contact with obstacles and walls, in a way that lets the agents progress towards their goal, by following the obstacle's boundaries ("wall-following"), as illustrated in Fig. 4.1. The nature of Bug Algorithms is ideal for indoor navigation on tiny, resource-limited, robotic systems, as their potential memory and processing requirements are low, therefore expected to take up little space on the on-board computer.

In this chapter we will delve into Bug Algorithms in more detail, by providing an overview of the techniques existing today. Although there have been two comparative studies on Bug Algorithms before (Noborio et al. (2000), Ng and Bräunl (2007)), our distinctive contribution is that we will evaluate how suitable Bug Algorithms are in for becoming a new navigational standard within robotics. Here we will investigate the assumptions for real-world scenarios. An important conclusion of our study is that Bug Algorithms tend to heavily rely on a perfect position estimation, which cannot be taken for granted in a GPS-deprived indoor environment. Global positioning systems could be set up beforehand, such as a motion capture or Ultra-Wide-Band (UWB) localization system (like in Mueller et al. (2015)). However, in cases like search-and-rescue scenarios, it is undesirable to have humans prepare the robot's environment. The robots would need to rely on their estimated position, obtained by their own, noisy, on-board sensors. With ground-bound robots, wheel slippage (Borenstein and Liqiang Feng (1996)) can cause an increasing error between the real and estimated position. The same goes for visual odometry (Scaramuzza and Fraundorfer (2011)), used by MAVs or hovercraft-like vehicles, where the error of the noisy velocity estimate will get accumulated over time. This is especially the case in a texture-poor environment.

We will compare a representative subset of Bug Algorithms in the ARGoS simulator, which is capable of modeling realistic physical interactions with objects in the environment. Although we will not implement as many Bug Algorithms as Ng and Bräunl (2007) did, we will test them in more realistic real-world conditions, containing elements such as odometry-drift or recognition-failures. We investigate their behaviors on hundreds of procedurally generated indoor environments, to compare their performance statistically. Here it is shown that the increased measurement noise on the on-board sensors causes a dramatic drop in overall performances of the Bug Algorithms. We will discuss how this affects the potential of Bug Algorithms in robotic navigation and what type of assumptions we can make about the environment, which can point us to the variations that are the most suitable.

An overview of Bug Algorithms is given in section 4.2, starting from their "maze-solver" origins, to the fundamental contact-based Bug Algorithms, to the more recent

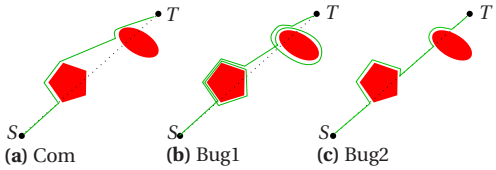


Figure 4.2: The behavior of simple Bug Algorithms: a) Com, b) Bug1 and c) Bug2. The *S* and *T* depicts the start and target position respectively.

extended range-based versions and hybrid solutions. This is followed by a sum-up of the methods already used in robotic navigation in section 4.3. Subsequently, we perform a quantitative comparison of the Bug Algorithms performances, of which the setup is explained in section 4.4. The experiments themselves are discussed in section 4.5, and involve various degrees of sensor-noise and -failures. The findings of this chapter will be discussed in section 4.6, from which we will present our conclusions in section 4.7.

4.2. THEORY AND VARIANTS OF BUG ALGORITHMS

The late 80s is when the term *Bug Algorithms* (BAs) first came into existence, evolving from the existing path planning algorithms like Dijkstra (Dijkstra (1959)) and A* (Hart et al. (1968)). However, the latter methods need to know their environment in advance, which includes start and goal positions, all obstacles and their position along the way. Maze-solving algorithms first explored the navigational problem without knowledge about the environment for enclosed spaces with walls and only one entrance and exit. If the target is reachable through a series of interconnected walls, a *wall-follower* would guarantee a quicker solution than a random walker (Mishra and Bande (2008)). As long as its left or right side is constantly in contact with a wall while moving through the maze, it will always reach the exit. However, if the environment is not an interconnected maze and contains disjoint obstacles between the start- and end-location, the wall-follower might get stuck in an endless loop. For typical indoor, non-maze, environments with disjoint obstacles, Bug Algorithms will be more suitable for the task.

4.2.1. CONTACT BUG ALGORITHMS

Lumelsky and Stepanov (1986) were the pioneers in developing bug algorithms. At first, they described a very simplistic BA, called the "common sense algorithm" which can be abbreviated as *Com*. Its behavior is illustrated in Fig. 4.2a. The position where a BA hits the obstacle for the first time is called a *hit-point*, and it has a *leave-point* as soon as the direction to the target is free. Intuitively, Com could solve many situations; however, Lumelsky and Stepanov (1986) pointed out that there are scenarios in which it cannot reach the goal. This happens when introduced to a more complex environment as, for instance, the one illustrated in Fig. 4.3a.

In the same paper of Lumelsky and Stepanov (1986), the *Bug1* algorithm was introduced, following a different strategy to overcome the issues that Com is facing. Every obstacle Bug1 comes across, it first has to "explore" the obstacle by following its entire border, while simultaneously keeping track of which position is the closest to the tar-

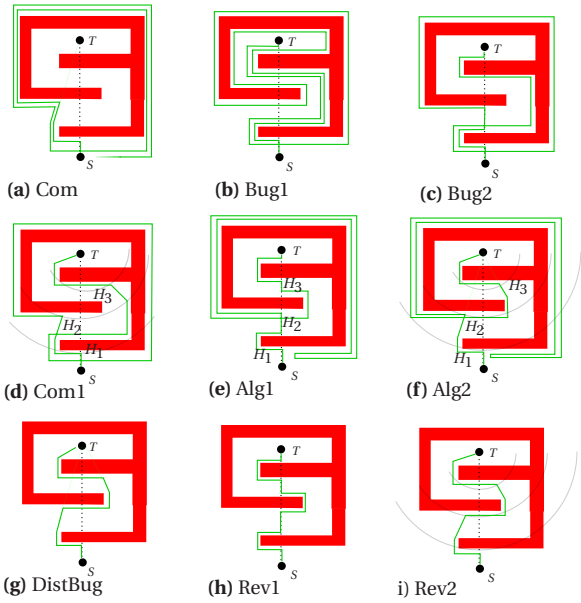
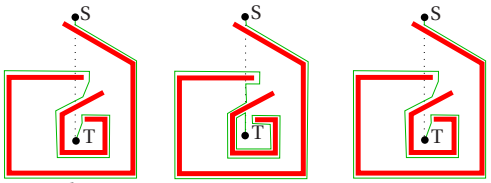


Figure 4.3: Generated paths by the Bug Algorithms a) Com, b) Bug1, c) Bug2, d) Com1, e) Alg1, f) Alg2, g) DistBug, h) Rev1 and i) Rev2 in a more challenging environment. The S and T depicts the start and target position respectively and H_i means the ith hit-point. x is the current position of the agent and CW and CCW stand for Clock Wise and Counter Clock Wise respectively.

get, as shown in the simple environment in Fig. 4.2b. After it encounters its original hit-point, Bug1 will continue and move towards the position closest to the target, from which it will leave the obstacle. Bug1 is able to handle environments where Com failed (as seen in Fig. 4.3b); however, it is a less intuitive approach. As it needs to know the entire border of the obstacle, this will naturally create unnecessary long paths. Lumelsky and Stepanov (1987) therefore proposed an alternative: *Bug2*. Between the beginning and end position, an imaginary line is drawn, called the *M-line*. In the simple scenario of Fig. 4.2(c), this means that the bug will follow the obstacles border until it hits the same M-line at the other side. As long as that point is closer towards the target than the hit-point's position, it will depart from the obstacle, as illustrated by Fig. 4.3c.

Sankaranarayanar and Vidyasagar (1990) were able to reduce the path length even further by adding memory. They extended the Bug2 algorithm with the following principle: to change its wall-following direction if it comes across a previously visited hit-point along the border of the obstacle. It has been dubbed as *Alg1*. It is true that in some situations a shorter path will be generated, however in others it will increase the path length, as can be seen in Fig. 4.3e. Sankaranarayanar and Vidyasagar (1990) also suggested an extended version of Com, *Com1*[1], which remembers the previous hit-point's distance to the target. Com1 will utilize this as an extra argument to initiate the departure from the obstacle boundary, as seen in Fig. 4.3d. Based on Com1, *Alg2* was created in the same pa-

[1]This is also being referred to as *Class1* in the studies of Noborio et al. (2000) and Ng and Bräunl (2007)



(a) Distbug **(b)** Rev1 **(c)** Rev2

Figure 4.4: An alternative complex environment to show a case that would produce a long path-length for a) DistBug, b) Rev1 and c) Rev2.

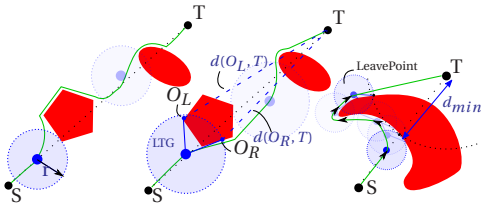
per of Sankaranarayanar and Vidyasagar (1990) as well, where it, similar to Alg1, reverses the wall-following direction if it encounters a previous saved hit-point. Alg2 therefore needs to keep track of all previous hit-points on its way for the reverse local direction condition, as this will occasionally occur (Fig. 4.3f). Kamon and Rivlin (1997) created a BA quite similar to Alg2, *DistBug*[2]. The only difference is that it will not remember the positions of all the previous hit-points, but solely the last one, therefore making it more memory efficient. Another intriguing aspect of DistBug, is that the local wall-following direction depends on the orientation with which the BA touches the hit-point. Most times, this will naturally lead it to the target and result in a shorter path, which is noticeable in the environment illustrated in Fig. 4.3g. However, there are situations where such a policy will fail, as in Fig. 4.4a. At the first hit point, it would be better to follow the wall in the other direction. The same goes for *Rev1* and *Rev2*, extensions to both Alg1 and Alg2 respectively, proposed by Horiuchi and Noborio (2001). Both BAs will alternate their local direction at each (new) hit-point, which is a good strategy for environments like in Fig. 4.3h and i. However, Fig. 4.4b and c show a situation where alternating the local wall-following direction is not the best policy. These examples do show that the best choice of local direction depends on the environment. It is, therefore, difficult to find a generic strategy for determining the best wall-following direction.

4.2.2. BUG ALGORITHMS WITH A RANGE SENSOR

What if the robot is able to sense obstacles already at a certain range and therefore act before touching the obstacles physically? Lumelsky and Stepanov (1986) already mentioned this idea in their first paper, which has been materialized in the papers of Lumelsky and Skewis (1988) and Lumelsky and Skewis (1990) as *VisBug 21 & 22*. Both are based on Bug2, but are now equipped with a range sensor able to sense up to a given maximum range. The BA will still follow the M-line but they can detect "short-cuts" to the next obstacle which should reduce the total path traveled, as can be seen in Fig. 4.5a.

Kamon et al. (1996) introduced a successful version of the range-based Bug Algorithms, called *TangentBug*. Within the maximum range of its sensor, a local tangent graph (LTG) is constructed, as illustrated in Fig. 4.5b. The LTG represents the discontinuities/borders of the detectable obstacle field around the robot. It starts out by moving towards the target while traversing the LTG edge that results in the quickest path D to the target (T) from its current position (x). However, if D of that edge increases, it will save

[2]Here we are referring to the extended DistBug algorithm of the same paper, with the search manager and local-direction choice based on the slope of the wall.



(a) VisBug **(b)** TangentBug **(c)** Special Scenario Tangentbug

Figure 4.5: The Bug Algorithms developed with obstacle detection with range sensors: a) VisBug and b) TangentBug. The S and T depicts the start and target position respectively. r stands for radius of the range sensor. LTG stands for local tangent graph and O_L and O_R stand for the left and right border of the detected obstacle within the range sensor respectively. $d(O_R, T)$ and $(d(O_L, T)$ stand for the distance between the left and right obstacle boundary to the target, respectively. c) A close up of a scenario in which Tangent bug is able to handle local minima.

the current range to the target as a local minimum (d_{min}) and will continue following the remaining boundary of that obstacle. If the robot senses a node on the boundary of the obstacle that is smaller than d_{min}, it trigger a leave-condition and, if possible, moves directly to the target (see Fig. 4.5c). Kamon et al. (1999) extended TangentBug to operate in 3D-environments as well (*3DBug*).

TangentBug is probably the most referred work in the field of BAs and many variants of it exists. The 360° range sensor assumption is changed to a sensor with a limited field of view with *WedgeBug* (Laubach and Burdick (1999)), for instance, to represent a stereo camera. Magid and Rivlin (2004) developed a BA which will actively search for the right local wall-following direction, to prevent a long-path length. Their *CautiousBug* will not choose a direction based on the angle of attack on the hit-point, as DistBug, but will first do a spiral search along the border, with the hit-point in the center. A disadvantage of this method seems that the spiral search by itself will also produce a long path, therefore it has less of an advantage over Tangentbug. A newer variation is *InsertBug* by Xu and Tang (2013), which navigates by means of way-points, placed on a safe distance from the obstacle's boundary. This could be seen as a version of TangentBug that adds a safety margin to each obstacle detected.

4.2.3. SPECIAL BUG ALGORITHMS

Some BAs either take a special approach or are combined with other existing methods (*HybridBugs*). Lee et al. (1997) used fuzzy logic with an adjusted Com method, a.k.a *FuzzyBug*. Assuming the BA is equipped with two single-beam sensors, pointed forward on both sides, it can detect if an obstacle is closer to its right or left. Based on a fuzzy membership function, FuzzyBug decides to follow the obstacle's boundary on its right or left, which a similar approach to DistBug's strategy. Noborio et al. (1999) developed HB-I, which is another HybridBug. After each hit-point of the obstacle, HB-I moves along the border in both directions until it hits a corner. It will then select the best direction first, based on the best-first search of a decision tree. Xu (2014) used a different approach with *RandomBug*. Once it detects an obstacle, it generates random points within the range of

its sensor. From these points, RandomBug selects the optimal one, dependent on how close the point is to the target, and generates a motion vector towards it. This produces a path quite similar to InsertBug, but the process is highly related to rapidly-exploring random trees (LaValle and James J. Kuffner (2001)).

Taylor and LaValle (2009) developed *IBug*, which is short for Intensity-bug. Its only information about its target is a wireless beacon on a specified location, of which it will navigate towards by means of the signal strength. Since they assume that IBug can make use of a "tower-orientation" sensor, the agent will move towards the beacon location. When it leaves the obstacle, IBug will temporarily save the value of the intensity (i_H) at that very moment. Here, a high intensity (signal strength) means a short distance to the target and a low intensity a large distance. While the robot follows the obstacle's boundary (always with a predefined Local direction of CW or CCW), it compares the current intensity level with i_H, as well as the current intensity and of time-steps back. If the signal strength decreases after increasing, the agent will have detected a local minima and a leave-condition is triggered, but only if the current measured intensity is larger than i_H. Although the leave condition is different, the latter comparison of intensity levels at the hit- and leave-points is quite similar in approach to Com1, with intensities substituted for distances.

4.2.4. OVERVIEW BUG ALGORITHMS

The BAs discussed in the previous sections are visualized in the overview of Fig. 4.6, where they are connected based on their development and added features. We subdivide the algorithms in a few major categories. The main division already started in the paper of Lumelsky and Stepanov (1986), where they presented Com, Bug1 and Bug2. Com led to a series of Bug Algorithms that navigated in an azimuth angle towards the goal whenever it had the chance to do so. Hence, here we categorize them as *Angle-Bugs*. Lumelsky and Stepanov (1986) realized that their next creation, Bug1, would create long trajectories by default. The community seems to have agreed as no extension or variation of Bug1 was developed here after, so therefore no similar Bug Algorithms have emerged since. Lumelsky and Stepanov (1986)'s alternative solution, Bug2, did have more potential, so new variations of *M-line-Bugs* have been presented, leading to a separate category of BAs in the overview.

Com is arguably the simplest BA, as it uses no memory, nor determines any M-line. With Com1, Sankaranarayanar and Vidyasagar (1990) added a distance-based leave condition, where it will only leave the obstacle if it reaches a position closer to the goal than before. Sankaranarayanar and Vidyasagar (1990)'s Alg1 and 2 are given additional memorization tasks as increments. Not only do these BAs remember the previous minimal distance to the goal, but all the hit-points' locations in between as well to reverse their local direction. Horiuchi and Noborio (2001) made Rev1&2 remember their last local wall-following direction, together with the local direction chosen at each hit-point. DistBug uses a more memory-friendly approach to determine its local wall-following direction, which is purely based on the detected slope of the approached obstacle. Eventually, the BAs started to use range-sensors at one point, creating the *Range-Bugs* category.

We can make some general observations about the overview in Fig. 4.6. Firstly, there are more Angle-bug-based BAs than M-line bugs. This is likely thanks to their more in-

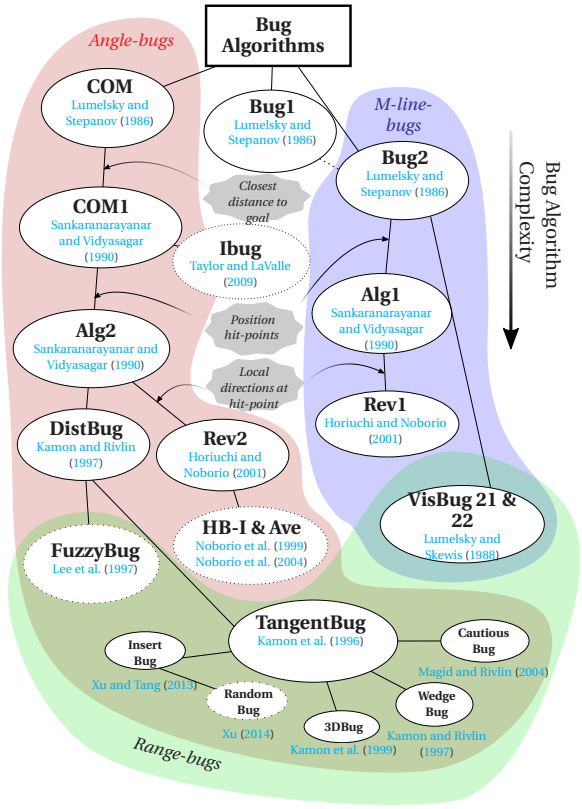


Figure 4.6: An overview of all the discussed Bug Algorithms (BAs) in section 4.2. These BAs are presented in a development tree of increasing complexity. It makes a distinction between Angle-Bugs, BAs that move to the target's azimuth direction, M-line-Bugs, BAs that use an M-line to navigate, and Range-Bugs, which use a range sensor to detect obstacles. The BAs noted in a dotted circle are special/ hybrid-bugs. The gray blobs indicate the type of memory and leave-condition added to the method. The latter is only shown until Rev1&2.

tuitive and less restrictive navigation strategy towards the target. Secondly, more and more features are added to the BAs as time progresses. Each new BA builds on top of an other, adding new leaving conditions and memory capabilities, therefore increasing the bug's complexity in the hope to find more efficient paths. The sole exception is the more recent Ibug, which is a recent BA variation, but is only one step away from Com1 in complexity.

4.3. BUG ALGORITHMS FOR ROBOTIC NAVIGATION

The BAs presented in the last section are considered as a potential new robotic path planning paradigm, because of their simplicity and low memory footprint. We first will discuss how the principle of BAs translates to realistic operating conditions. Afterwards, existing BA robotic implementations will be presented and discussed on how well these studies represent real-world scenarios.

Table 4.1: Robotic implementations of various Bug Algorithms (BAs). These are evaluated on the type of platform used, whether the environment was real or simulated and which BA type was used. Moreover, it shows the used local sensors for obstacle detection and the used global sensor for a position estimate.

Paper	Platform	Environment	Bug algorithm	Local sensors	Global sensors
Kamon and Rivlin (1997)	Wheeled robot	Real	DistBug	Range sensors	Global localization (system not given)
Laubach and Burdick (2000)	Microrover	Real	RoverBug (wedgebug extended)	Stereo images	Guiding operator (First person view)
Mastrogiovanni et al. (2009)	Wheeled robot Hexapod robot	Real	μNAV	Ultrasound range sensor Wheel odometry	Azimuth angle by photo diodes (only for hexapod)
Zhu et al. (2010)	Wheeled robot	Real	Bug2 and a DistBug variant	Laser scanner (180 deg)	Global localization (system not given)
Kim et al. (2013)	Wheeled robot	Real	Tangentbug (adjusted)	Ultrasound range sensor Wheel odometry	Global localization (system not given)
Taylor and LaValle (2014)	Wheeled robot	Real	Ibug	Touch sensors	IR Beacon
Ebrahimi et al. (2014)	Quadcopter	Simulation	UavisBug	Camera	Motion Capture System
Gulzar et al. (2015)	Wheeled robot	Real	Not Given	Ultrasound	Motion Capture System
Marino et al. (2016)	Quadcopter	Simulation	Bug2	Laser scanner (180 deg)	UWB localization

4.3.1. BUG ALGORITHMS IN REAL-WORLD CONDITIONS

In the earlier literature overview in section 4.2, it seems to be the case that BAs heavily rely on perfect localization. They almost all assume that the BA does not know the exact location and shape of the obstacles, however they almost all need to know the exact coordinates of their goal and their own position. The latter is used for more aspects of BAs than first meets the eye. Angle Bugs need to know the distance and azimuth angle to the target at any point. M-line-Bugs (i.e. Bug2, Alg1, Rev1) both remember the exact line (and direction) between the starting position and the goal, and recognize if they have reached it. Hit-point memorizing BAs (i.e Alg1&2) need to match their current position estimate with previously hit-point coordinates. In a typical indoor GPS-deprived environment, obtaining and maintaining a world position is a significant challenge. Real-world robots that do not have maps then will need to rely on odometry, which is prone to errors and has the tendency to drift in time from the ground truth. Some BAs (i.e. Alg1&2 and Rev1&2) have to remember the exact coordinates of where they have been, which ensures a convergence to the target. This could be done by odometry as well, or by memorizing features in the environment with scene descriptors locally with SIFT (Goedemé et al. (2007)) or globally with Bag-of-Words (Fraundorfer et al. (2007)). As with visual

odometry, the descriptor's performance highly depend on the texture of the environment. Most BAs use a Distance-to-Target (DT) measurement in their leave-condition. Next to using the drift-susceptible odometry, they could also retrieve the DT in ways such as received signal strength intensity (RSSI) of Bluetooth (Bargh and de Groote (2008)) or Ultra-Wide Band (UWB, Guo et al. (2017)). This does of course require the placement of a wireless transmitter at the target location. Moreover, none of these sensors have a perfect ranging measurement, with errors ranging from 10 cm to 5 meters.

4.3.2. EXISTING IMPLEMENTED BUG ALGORITHMS FOR ROBOTIC NAVIGATION

This section will look at current robotic BA implementation, either in a real-world environment or a simulated scenario. An overview of these methods is presented in Tab. 4.1, which lists the platform they used and shows the sensors the robot was equipped with for local obstacle sensing and global position estimations. Several works have focused on implementation of simulated systems. The work of Ebrahimi et al. (2014) developed *UavisBug* for a simulated MAV for visual guided navigation. However, they combined the BA with SLAM for the obstacle detection and boundary following, from which they used a potential force field to navigate around the obstacle. Moreover, they mentioned the requirement of a motion-capture-system if their technique would be implemented in the real world. Marino et al. (2016) also mentioned that they assumed the existence of an UWB-localization system. Moreover, if the agent believes it is at the right goal position but on a different floor, it will use the Dijkstra method to compute the shortest path. Even though Ebrahimi et al. (2014) and Marino et al. (2016) acknowledged the limited sensing, computing and energy capability of MAVs, they still combine the efficient BAs with computationally-heavy navigation techniques.

With DistBug, Kamon and Rivlin (1997) showed, as one of the first, a BA implemented on an actual wheeled robot, a Nomad200. In their paper they mention that the robot, while moving to the target, only responds to local measurements by the contact sensors. However, the robot always moves towards the target after boundary following, therefore, it must also know its own and the targets position in global coordinates. Although the paper of Kamon and Rivlin (1997) has not specified this, their BA would need to use a global localization system. Zhu et al. (2010), Kim et al. (2013) and Gulzar et al. (2015) have implemented a BA on autonomous real-world wheeled robots without a tele-operator. In all cases, they were using single beam range sensors and/or a laser scanner. Again, the exact location of the robots is needed in order for the BA to move towards the target.

Mastrogiovanni et al. (2009) acknowledged that a robot would not be able to know its exact position, but would need to infer it from its noisy on-board sensors. They implemented *μNav* on a real-world wheeled robot, AmigoBot and a hexapod, Sistino. The first platform used ultra-sonic sensors for obstacle detection and wheel-odometry for global localization. Since the wheeled robot combined its wheel-odometry with the azimuth angle toward the target, it could reach the target location from one room to another, even if the orientation was manually perturbed by the researchers. However, the operation area only spanned across 2 rooms and no notion was given of what the navigational limit was, based on accumulated errors of odometry drift. Their second robot, the hexapod, was not able to use odometry, so the azimuth angle had to be given by an external

source through photo diodes.

Taylor and LaValle (2014) implemented IBug on a small wheeled robot for several small-scale environments. In their previous work (Taylor and LaValle (2009)), they described the BA to be suitable for navigating towards a single wireless beacon. Nevertheless, for the test on a real robot, a Lego-Mindstorm-based platform, Taylor and LaValle (2014) used an infra-red (IR) beacon instead. It proved to be challenging for their tests to use the signal strength of i.e. a WiFi beacon at a large range. The use of the IR beacon did necessitate a constant line of sight, which required the obstacles and walls to be lower than the robot itself.

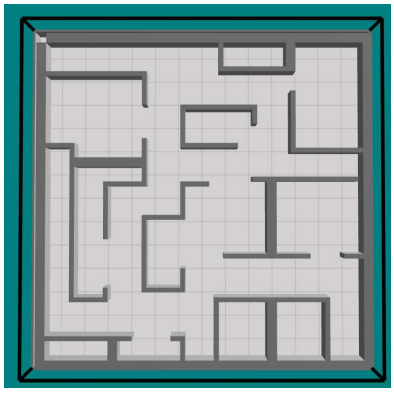
If we look at the existing implementations of BAs in real or simulated robots, they all assume or need explicit global localization, either by a UWB localization system, a motion capture system or a guiding navigator, for the exception of IBug, which used a visual beacon. Mastrogiovanni et al. (2009) is actually the only one that used the odometry of a (bigger) wheeled robot to recover its own position and to update the azimuth angle towards the target; however, the real-life test was too small to draw any conclusions about the suitability of BAs for robotic navigation. In the comparative study, presented in the next sections, we will test various BAs with varying amounts of odometry drift, recognition failures and distance noise. This will show that these real-world conditions will have significant effect on the BAs' performances.

4.4. EXPERIMENTAL SET-UP COMPARATIVE STUDY BUG ALGORITHMS

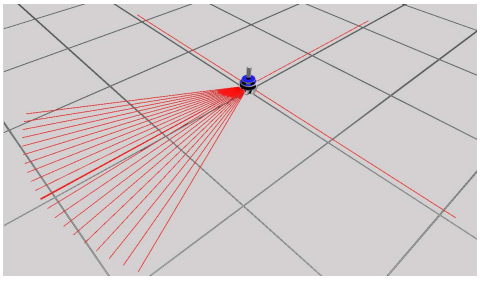
In this chapter, we study whether BAs could be used for real-world robotic navigation. Most indoor environments have more complex obstacle configurations than an open environment with a few convex obstacles. There are many situations where the robots could get stuck on their way, particularly in rooms. In this section, we will present our motivation for this study and the chosen set of BAs to be evaluated. We will then provide the details of the simulation used and the procedural environment generator for typical indoor environments. Afterwards, the implementation specifics of the BAs will be presented, by explaining a wall-following paradigm, which is the foundation for all BAs to be implemented.

4.4.1. MOTIVATION AND CHOICE BUG ALGORITHMS

There have been previous comparative studies between existing BAs. In the paper of Noborio et al. (2000), Class1, Bug2, Alg1, Alg2 and HB-I, were compared and evaluated on their generated path-length within a complex maze. They based their observations on just one indoor environment. A newer comparative study was performed by Ng and Bräunl (2007), on: Bug1, Bug2, Alg1, Alg2, DistBug, TangentBug, OneBug, LeaveBug, Rev1, Rev2 and Class1. They presented the BAs with four types of environments and recorded the total path length for each run. However, the performance could not be adequately compared due to the inconsistent results. In this chapter, we test a set of BAs in hundreds of procedurally generated environments, so we can statistically evaluate their performances. Moreover, set up the simulation environment to include the possibility to simulate real-world properties such as sensor noise and odometry drift.



(a) Generated environment in ArGos



(b) Modified ArGos Foot-bot

Figure 4.7: a) The resulting environment from Fig. A.1f generated within the ARGoS simulator and b) a modified Foot-bot simulated robot with range sensors (red lines) used for wall following.

In the overview of the BA-methods existing today (section 4.2.4), it can be seen that many of the methods are natural increments of one another with increasing complexity. If the fundamental BAs can be tested with these real-world conditions, we would automatically find the issues that their descendants are facing as well. Specifically, we have selected Com, Com1, Bug2 and Alg1&2. Range-bugs will not be considered as the selected BAs are the base of those more complex versions. Moreover, the selected BAs presents a mix of different types of strategies (Angle-Bugs and M-line-Bugs) and memory-use (distance and/or hit-points). We will exclude bugs that determine a local wall-following direction, as the policy for this choice is heavily influenced by the structure of the environment, as previously discussed in section 4.2.1. Moreover, any special bugs will not be considered either, since they contain aspects and enhancements that no other BA-related research followed up on.

4.4.2. SIMULATION AND PROCEDURALLY ENVIRONMENT GENERATOR

It is our ambition to test the earlier mentioned BAs in a simulator with realistic and swift physics calculations. ARGoS, a multi-physics robot simulator developed by Pinciroli et al. (2012), is used for our comparative study. Its main trait is its efficiency, which enables the simulator to run many times faster than real-time, which will be essential if the BAs are evaluated in many environments. Although ARGoS does have the capability to incorporate its own, C++ based, controller for the robots, the ROS framework is used to enable Python-based controllers. The ROS messaging system is also ideal to modulate whether a new environment needs to be generated, to vary the measurement noise and select the right bug algorithm.

Since the BAs will be evaluated in many indoor environments, it would be unfeasible to design these by hand. Therefore, a procedural indoor environment generator will automatically generate a new arena for the bugs to navigate in. This process is depicted in appendix A.2, where a standard indoor environment with corridors and rooms can be generated in different configurations. Rooms are especially challenging, as they can lead

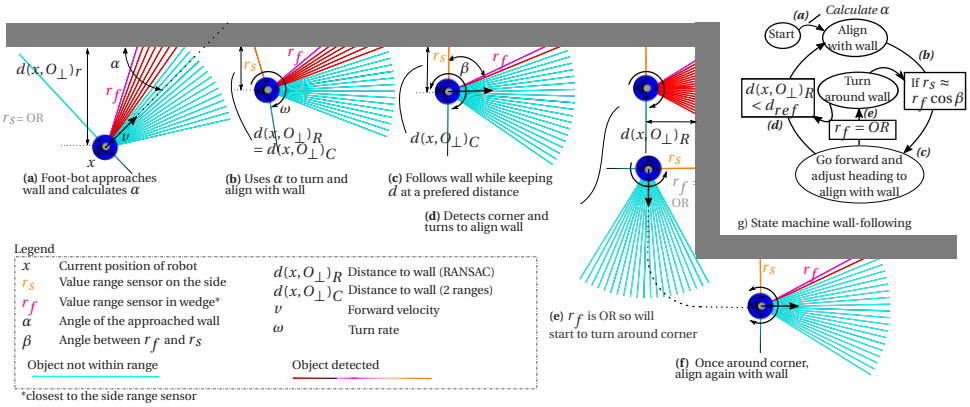


Figure 4.8: a)-f) Schematics to explain the wall-following paradigm for a right-sided local direction with (g) the corresponding state machine . OR stand for out of range.

to agents getting stuck in loops, which will showcase the strengths and weaknesses of the evaluated BAs. The resulting environment in the ARGoS simulation is shown in Fig. 4.7a.

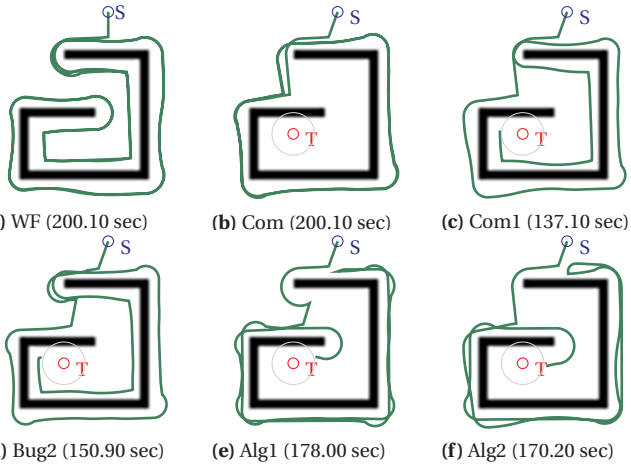
The ARGoS FootBot is used for our experiments, which is a simulated wheeled mobile platform (see Pinciroli et al. (2012) for specifications). The FootBot contains many options to attach various types of sensors, however for our experiments we will only use the proximity sensors. We adapted FootBot to turn the proximity sensors into single beam range sensors with a maximum measurable distance of two meters, placed in the configuration shown in Fig. 4.7b. The robot has two separate single beam range sensors located on each side and 20 range sensors pointing to the front in a wedge shape. This is to simulate a depth sensor/stereo-camera for obstacle detection with a few additional range sensors on the side. Since the robot must move towards the range-wedge configuration, its movement will be non-holonomic.

4.4.3. IMPLEMENTATION DETAILS BUG ALGORITHMS

The most important element of any BA, is the ability to follow a boundary of an obstacle or wall. Based on the robot configuration in Fig. 4.7b, we developed a wall following principle. Fig. 4.8a-f shows the wall following of the footbot for a right-sided local direction and Fig. 4.8g shows the state machine, which can also be found as pseudo code in appendix A.3.2, Alg. A.3.1. If the robot moves forward and hits a wall, like in Fig. 4.8a, the angle of the wall can be estimated by using a RANSAC line-fit method (Fischler and Bolles (1987)) in the wedge of range sensors.[3] This is done so that the true distance $d(x, O_\perp)_R$ can be estimated from the robot to the wall.[4] If this distance becomes smaller than d_{ref}, the preferred distance from the wall, it will keep turning either CW or CCW

[3] Since RANSAC uses random samples to determine the slope of the plane, some stochasticity is expected in the wall-following behavior.

[4] This only goes with the assumption that the robot will always encounter walls and no single objects, such as plants. The later will not be simulated in ARGoS; however, a classifier able to distinguish walls from small obstacles, must be added if this principle is implemented on a real robot.



(a) WF (200.10 sec) **(b)** Com (200.10 sec) **(c)** Com1 (137.10 sec)

(d) Bug2 (150.90 sec) **(e)** Alg1 (178.00 sec) **(f)** Alg2 (170.20 sec)

Figure 4.9: The results of a) the wall-following only (WF) and the implemented bug algorithms that use the same WF b)-e) as part of their navigation strategy. The time limit is 200 sec.

until it is aligned with the wall. Fig. 4.8b and c shows this alignment for a right-side local direction. This will be the case if the measurement of the side range sensor r_s is equal $r_f \cdot \cos\beta$, where r_f is the element from the range wedge that is the closest to r_s and β is the angle between r_s and r_f.

After the robot is aligned, it will need to follow the wall, as in Fig. 4.8c. Now the true distance to the wall $(d(x, O_\perp)_C)$[5] will be calculated as follows:

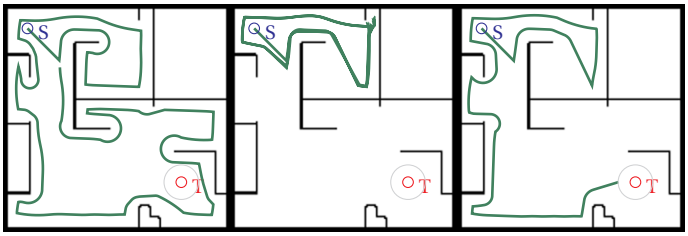
$$d(x, O_\perp)_C = \frac{r_s \cdot r_f \sin\beta}{\sqrt{r_s^2 + r_f^2 - 2 \cdot r_s \cdot r_f \cos\beta}} \tag{4.1}$$

The derivation of the latter equation can be found in appendix A.3.1. The FootBot will maintain $d(x, O_\perp)_C$ to be near d_{ref}, and to keep being aligned in the process. However, since the robot's heading and the measurements of r_s and r_f are coupled, a separate control heuristic is developed to make the wall alignment possible. The details of this wall-alignment method can be found in appendix A.3.2, Alg. A.3.2.

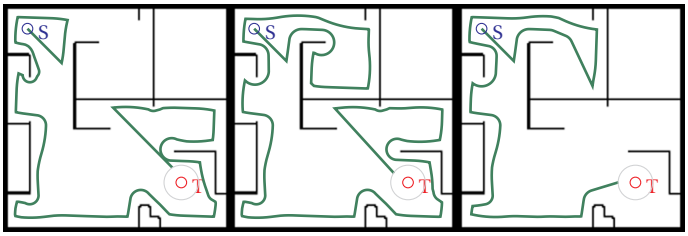
When the FootBot hits another wall during its forward motion, as in the corner in Fig. 4.8d, while in its wall-following state, it will turn away from the wall until it is aligned (similar condition as with Fig. 4.8b). If during a forward motion, the front-range sensor is out of range, as in Fig. 4.8e, the foot-bot will initiate a wide turn, to find the wall on the other side as in Fig. 4.8f. The state machine for the wall-following behavior can be found in Fig 4.8g, of which the pseudo code can be found in appendix A.3.2, Alg. A.3.1.

This control heuristic should result in a robust wall-following behavior, in particular for indoor environments with straight walls. The resulting wall-following behavior is shown in Fig. 4.9a. Here it can be seen that the wall-following produces a smooth path all

[5]The R and C subscript of $d(x, O_\perp)$ enables separation of the two methods (RANSAC or only 2 ranges) of retrieving the true distance to the walls.



(a) WF (300.10 sec) **(b)** Com (300.10 sec) **(c)** Com1 (173.70 sec)



(d) Bug2 (226.60 sec) **(e)** Alg1 (282.90 sec) **(f)** Alg2 (172.10 sec)

Figure 4.10: Behaviors of the implemented a) wall-follower (WF) and Bug Algorithms (BAs): b) Com, c) Com1, d) Bug2, e) Alg1 and f) Alg2 in one generated environment. The BA starts in the top left corner at Start (S) and ends withing 1 meter radius from the Target (T), with a time-limit of 300 seconds.

along the walls of the mirrored "G". All the implemented bug algorithms, from which the pseudo-code can be found in appendix A.1.1, will make use of this exact same wall following behavior in their state machine. [6] The resulting trajectories of the implemented BAs in the ARGos simulated environment are shown in Fig. 4.9b-f.

4.5. EXPERIMENTAL RESULTS OF BUG ALGORITHMS IN REAL-WORLD CONDITIONS

In this section, the BAs will be compared against each other on a wide range of procedurally generated environments. Moreover, we will investigate how sensitive these algorithms are to real-world conditions, subjecting them to the experimental setup explained in section 4.4. The selected BAs: Com, Com1, Bug2, Alg1 and Alg2, will be evaluated first with perfect localization. After that, they will be subjected to increasing odometry drift, varying hit-point recognition failures and Distance-to-Target (DT) noise. From this, we hope to properly evaluate the BAs ability to handle real-world conditions.

4.5.1. EXPERIMENTS WITH PERFECT LOCALIZATION

The implemented BAs' performances are evaluated in 200 procedurally generated environments, with a constant size of 14 by 14 meters. Each BA will have one chance to navigate through the same environment with a time limit of 300 seconds. This should be sufficient to reach the target, while preventing the simulation to run endlessly, if one of the BAs gets stuck in a loop. Each BA's success percentage is recorded, which is the percentage of when the target is reached out of the 200 environments. Fig. 4.11a shows the percentage of BAs that made it to the goal within the required 300 seconds with perfect localization (indicated with σ=0.00), where the goal is considered reached if the BA is able to get within one meter radius.

The BA's total trajectory length is recorded as well, which is normalized by dividing by the optimal path length as calculated by the A* path planning algorithm. A* will get an occupancy grid identical to the procedurally generated environment and is able to visit all the 8 neighboring cells at each step.[7]. Note that the grid not available to the bug algorithms by any means. The normalization is applied in order to compare the performances adequately across the generated environments, as the optimal path will be different at each iteration. Fig. 4.11b shows a box-plot of the length of the BAs' trajectories with perfect localization. For all BAs, all path-lengths are taken into account, including the ones that did not reach the goal. Although this skews the statistics, a time limitation of 300 seconds will be held constant throughout the experiments to ensure consistency.

In the simulation set-up, the BA would need to leave the walls physically in order to reach the goal, which results in the wall-follower (WF) to not be able to complete the experiment at all. Com is only capable to reach the goal about 60 % of the time with perfect localization. Com does not use memory and this results in it occasionally getting stuck in loops as shown in Fig. 4.10b. The last four BAs, Com1, Bug2, Alg1 and 2, have a success percentage of around 90% in Fig. 4.11a with perfect localization.

In Fig. 4.10d and e, it can be seen that Alg1 and its ancestor Bug2 need to find the M-line first before it can leave the wall. However, this restriction results in longer trajectories. Com1 and Alg2, on the other hand, will move towards the target if the chance arises, hence have more leave-opportunities along their path (Fig. 4.10c and f). The outcome is that in the 200 generated environments, Com1 and Alg2 have a shorter path-length than Bug2 and Alg1 in average (Fig. 4.11b).[8]

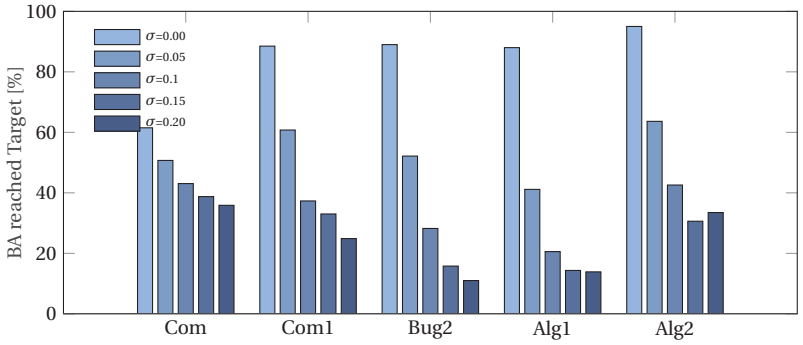
4.5.2. EXPERIMENTS WITH ODOMETRY DRIFT

In this chapter, we test the BAs' potential for real-world navigation purposes. Therefore, we have added more realistic elements to the simulation, based on our discussion in section 4.3.1. In the absence of an exact global position, BAs will need to rely on odometry. Therefore, this section will investigate the effects of odometry drift. We assume that

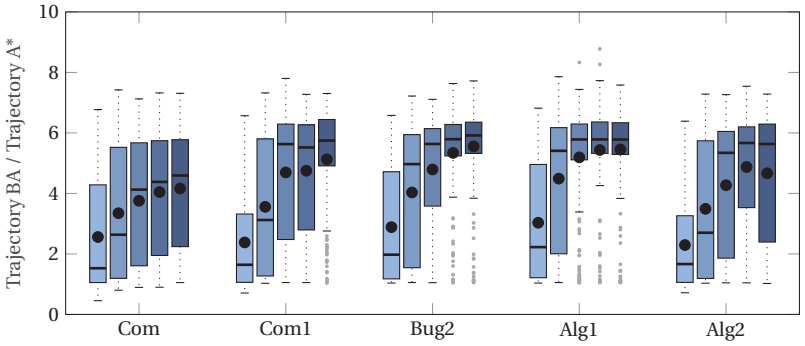
[6]The bug algorithms are implemented in Python, but to test the claim of the computational efficiency of bug algorithms, we have also implemented Com1 on a STM32F4 processor (specs: 168MHz processor speed and 128kB RAM memory). The average computation time is 0.04 milliseconds.

[7]An 8-connection A* will cause the path to go through a corner of an obstacle. The grid that is available A*, will include padded wall and obstacles compared to the actual environment where the Bug Algorithms will navigate in.

[8]A bootstrapping based statistical similarity analysis of both the success rate and the trajectory length can be found in appendix A.4.1.



(a) Success percentage.



(b) Normalized trajectory length

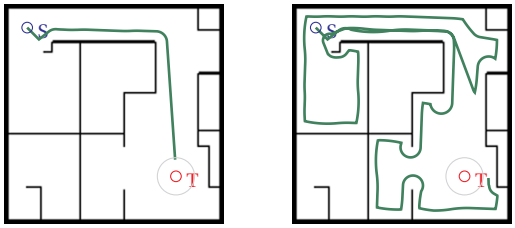
Figure 4.11: The a) percentage of the Bug Algorithms Com, Com1, Bug2, Alg1 and Alg2, which made it to the goal of increasing velocity measurement noise (σ) which causes odometry drift, and b) the trajectory length normalized by the optimal path calculated by A*.

the BA will know its own and the target's position at the start of the experiments, but it has to keep them up-to-date with its own, noisy, velocity measurements. For these experiments, we assume that the position estimate is acquired by the latter assumption, namely:

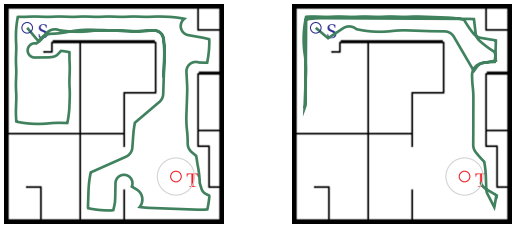
$$\tilde{\mathbf{x}}_t = \tilde{\mathbf{x}}_{t-1} + \dot{\tilde{\mathbf{x}}}_{t-1} \tag{4.2}$$

, where $\tilde{\mathbf{x}}_\mathbf{t}$ is the x- and y-position estimate at a given time. $\dot{\tilde{\mathbf{x}}}_{t-1}$ is assumed to be $\mathcal{N}(\mathbf{u}_{t-1}, \sigma)$, where $\mathbf{u}$ is the actual velocity, from which the outcome on the system consists of noise with a standard deviation of σ.

Fig. 4.11 shows the impact on the performances of the BAs when exposed to odometry drift due to noisy velocity estimates, with a σ of 0.05, 0.10, 0.15 and 0.2. In Fig. 4.11a indicates a significant drop in all the BAs' success percentage with an increasing σ. In Fig. 4.11b we see that it has a large effect on the trajectory length overall, although there is a less significant degeneration of the Angle-Bugs' performance (Com, Com1 and Alg2). Bug2's and Alg2's performances took the deepest dive with a relatively small increment of



(a) σ=0.05 (300.10 sec) **(b)** σ=0.10 (59.70 sec)



(c) σ=0.15 (70.50 sec) **(d)** σ=0.20 (300.10 sec)

Figure 4.12: Example of Alg2 in environment # 123 of the experimental testing, with increasing noise variance of σ = a) 0.05, b) 0.10, c) 0.15 and d) 0.20. The BA starts in the top left corner at Start (S) and ends withing 1 meter radius from the Target (T), with a time-limit of 300 seconds. In d) Alg2 suddenly turns 180 degrees on the left side of the environment, without having seen that hit-point before.

the odometry drift, whereas Com's performance only gradually decreased. As Com does not save any position or distance-to-goal at hit-points, only its bearing estimate towards the goal is affected by faulty velocity estimates, resulting in the simplest BA outperforming the rest with $\sigma > 0.05$. Alg2 already lost its advantage to recognize previously visited places, as its success-rate is similar, if not lower, than Com1 at a σ of 0.2.

However, both Alg1 and Alg2 show signs of stagnation from $\sigma = 0.15$ and on, as their performances does not seem to decrease any further and even seem to improve slightly. At that point, it could be that it would accidentally recognize a previously hit-point at a location where it has not been before due to the odometry drift. Although seemingly unwanted, this randomness could have helped the BA to get out of difficult situations, as in Fig. 4.12 with Alg2.[9]

4.5.3. EXPERIMENTS WITH FALSE POSITIVE AND FALSE NEGATIVE RECOGNITION RATE

BAs can also recognize previous hit-points based on scene-recognition. In this chapter, we will not use the techniques and descriptors discussed in subsection 4.3.1, but will simulate their performance through false-negative (FN) and false-positive (FP) recall rates. With an increasing probability (p) of a uniform distribution, the chances of a previously visited hit-point being falsely recognized at a different location (FP) or not

[9] Statistical correlation analysis of the effect of the increasing odometery noise both the success rate and trajectory length can be found in appendix A.4.2.

being recognized at the right position (FN) will increase.

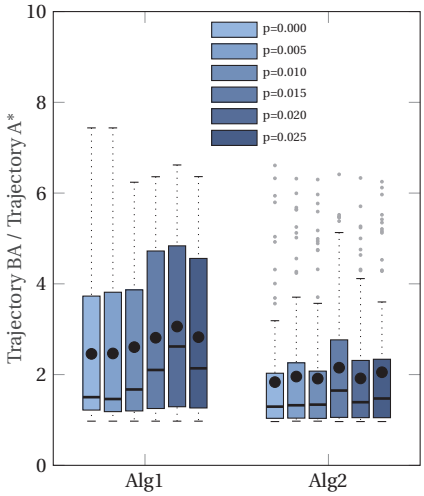
Of the implemented BAs, only Alg1 and Alg2 specifically use previously visited locations to change their local wall-following direction from right- to left- sided. In Fig. 4.13, they are being evaluated with an increasing p(FP) in Fig. 4.13a&b or p(FN) in Fig. 4.13c&d over 100 generated environments. At a p(FP)=0.005, there is a chance of FP occurring 1-2 times (0.5 %) during the run-time of 300 seconds and at p(FP)=0.025 a chance of 7-8 times (2.5 %). At p(FN) = 0.2, every time the BA encounters a previous hit-point, there is a 20 % chance that it will not recognize it and at p(FN)=1.0, the hit-point will never be recalled. Fig. 4.13a and b shows that increasing the p(FP) has more effect on the performance of Alg1 than Alg2. This can be due to Alg2's fewer leave-restrictions, which makes it less sensitive to more frequent occurrences of FP. Fig. 4.13c and d show that both Alg1 and Alg2 seemed to be hardly effected by an increasing p(FN). The only trend that could be noticed is for Alg2 as the variance of the trajectory length slowly creeps up with an increasing p(FN) in Fig. 4.13c. With p(FN) = 1.0, Alg1 and Alg2 are not able to recall previous hit-points, which makes them identical to their ancestors Bug2 and Com1. However, no significant difference can be noticed in the success rate of Fig. 4.13d with p(FN) = 0.0 and 1.0 for both Alg1 and Alg2.[10]

4.5.4. EXPERIMENTS WITH DISTANCE MEASUREMENT NOISE

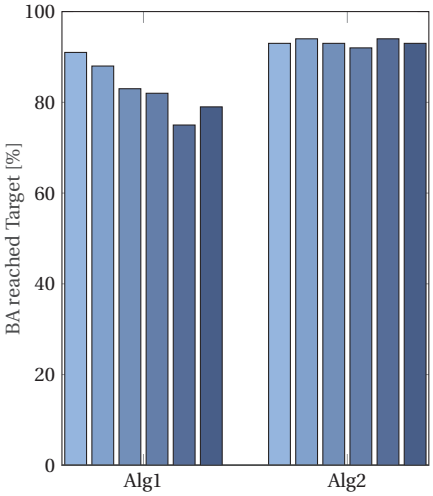
BAs could also use a Distance-to-Target (DT) measurement, so here we assume that the agents are carrying a sensor able to determine this. Com1 and Alg2 both save previous DT measurements to prevent getting stuck in a loop in some situations. In Fig. 4.14, we are showing the (a) trajectory length and (b) success rate of the BAs subjected to increasing DT noise, while keeping both the velocity measurement noise (odometry drift) and the FP & FN rate at 0.0. The noisy DT measurements ($\tilde{d}(x,T)_t$) at time t are modeled by $\tilde{d}(x,T)_t = \mathcal{N}(d(x,T)_t, \sigma)$, where $d(x,T)_t$ is a scalar that stands for the true DT at time t and σ is the standard deviation of the noise. The degrading performance in both trajectory length and success percentage for increasing σ is more noticeable for Com1 than for Alg2. Com1's only mechanism to get out of a potential loop is to compare its current DT with a saved one to decide when to leave the wall. Once it is gradually losing this capability with the noisier DT measurements, its behavior will become more and more similar to Com's, as observed in Fig. 4.15. Moreover, Com1's success percentages in Fig. 4.15b drops to around 60 percent at a σ=6 meters, which is equivalent to Com's score in Fig. 4.11b with perfect localization. Alg2 is less affected by the increasing DT noise, which is likely because it can rely on memorized position as an additional leave condition.[11]

[10] Statistical correlation analyses of the effect of the increasing recognition failure rate on both the success rate and trajectory length can be found in appendix A.4.3.

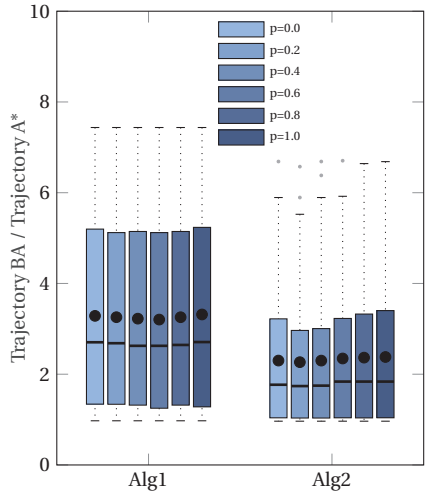
[11] Statistical correlation analysis of the effect of the increasing DT measurement noise on both the success rate and trajectory length can be found in appendix A.4.4.



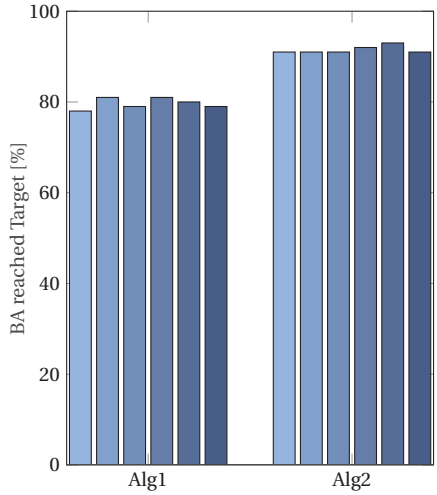
(a) Trajectory length (FP)



(b) Goal reached (FP)

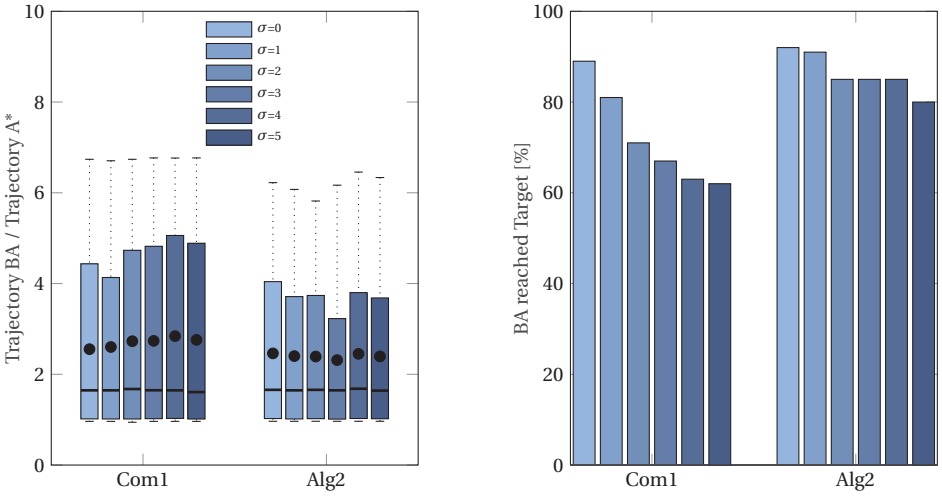


(c) Trajectory length (FN)



(d) Goal reached (FN)

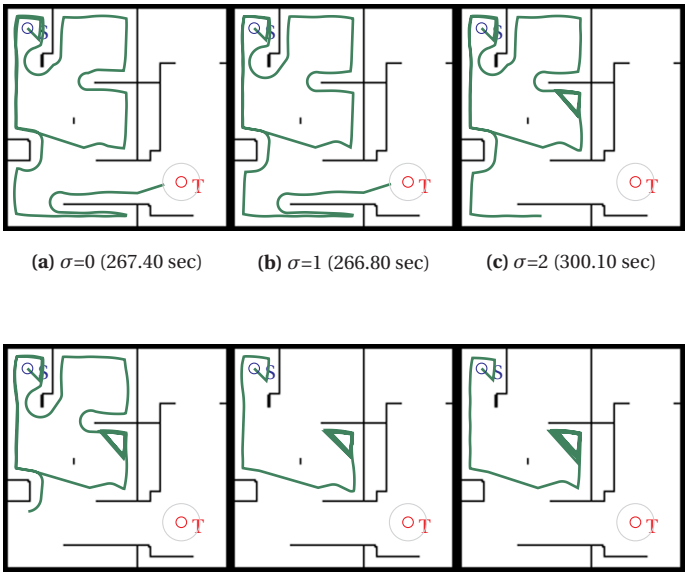
Figure 4.13: The a)&c) measured trajectory length and (b&d) percentage of Alg1 and Alg2 reaching the goal, with a varying chance (p) of a false-positive (FP - a&b) or a false-negative (FN - c&d) of a previous recognized point to occur.



(a) Trajectory length

(b) Goal reached

Figure 4.14: The performance of Com1 and Alg2 with varying distance measurement noise (σ) in meters, in the a) normalized trajectory length and percentage b) of bugs who made it to the goal.



(a) σ=0 (267.40 sec) **(b)** σ=1 (266.80 sec) **(c)** σ=2 (300.10 sec)

(d) σ=3 (300.10 sec) **(e)** σ=4 (300.10 sec) **(f)** σ=5 (300.10 sec)

Figure 4.15: An example environment with the trajectories of Com1 with increasing distance measurement noise variance (σ) in meters of a) 0, b) 1, c) 2, d) 3, e) 4 and f) 5 meters. The BA starts in the top left corner at Start (S) and ends within 1 meter radius from the Target (T), with a time-limit of 300 seconds.

4.6. DISCUSSION

This section will discuss both the experimental set-up and results and will accommodate the observations made in section 4.3. The modeled real-world conditions will be discussed first, including the implementation details of the simulation and the chosen noise-models and Bug Algorithms (BAs). Here we will give some suggestions for future development in this topic. Afterwards, we will discuss the results from our experiments, from which we will determine which aspects of BAs are suitable for real-world scenarios.

4.6.1. MODELING REAL-WORLD CONDITIONS

BAs are potentially an ideal indoor navigation paradigm for tiny robotic platforms with limited resources. Although the results show that the paths generated are sub-optimal compared to path-planning algorithms as A*, no map is needed for navigation. Nevertheless, we established that the BAs, presented in section 4.2, tend to over-rely on a perfect localization, which cannot be guaranteed for all indoor environments. If no global localization system is available, the BA needs to rely on its noisy on-board sensors to know where it is and integrate this knowledge with the target's position. In section 4.3, we reflected on several issues that a robotic implementation of a BA will come across. This includes: an increasing odometry drift, a mismatch between its measured position and the ground truth; recognition failures, i.e. when it fails to recognize a previous location or falsely detects one; and noisy distance to target measurements, which could interfere with the suitable leaving-condition. There are other types of sensor-noise and failures to consider, such as the noise in the laser-range sensors or (stereo-)cameras for (local) boundary/wall following. However, in this chapter, we focused on the global position estimation instead, as this is an issue that all bug algorithms must deal with.

For the experiments themselves in section 4.5 we used simple noise models, i.e. using a Gaussian probability distribution for the odometry drift and noisy range measurements, and a pseudo-random number generator for FNs and FPs occurrences. Future work could look at more realistic noise characteristics. For ground-bound robots, for instance, wheel slippage is determined by the materials used and the friction with the floor. If visual-odometry is used, Gaussian noise could very well be applied, however the texture of the environment is crucial to the variance. The FP & FN occurrences are also very much determined by the features of the environment, as aliasing could occur at areas that are very similar. There is no equal probability of these failures to happen throughout the trajectory of the bug. Moreover, distance measurements by radio beacons not only suffer from regular noise around the mean, but have to endure a whole range of disturbances. This includes uneven directional propagation noise, the reflection off the walls and interference of other signals. For the experiments in this chapter, we wanted to have more control over the noisy measurements to find a clear correlation between the noise severity and the performance of the BAs, so we restricted ourselves to use the basic versions of the noise models.

The ARGoS simulator and environment generator was very useful for this chapter's experiments, as it generated new environments at a high pace and run the experiments faster than real-time. This enabled us to test the BAs on hundreds of environments, leading to more reliable results. Nevertheless, further development of these experiments must be performed in a more realistic simulation, with more types of obstacles and

visual representations, to induce more challenges of a typical indoor navigation task. Moreover, as mentioned earlier, we should also include more realistic sensor models to further increase the realism of the simulation.

4.6.2. BUG ALGORITHMS PERFORMANCE IN SIMULATED REAL-WORLD CONDITIONS

Generally, our experiments showed that all BAs performed worse with a higher odometry drift, noisier range measurements and increasing failure cases. The most noticeable feature, is that the BAs did not all have a similar drop in performance, which is especially noticeable with increasing odometry drift in Fig. 4.11. Some had a more severe response than others, namely those using memory. Com, being the simplest of all BAs, started out as the worst one of the six, to the best performing with only standard deviation of 0.1 m/s in the velocity estimation. As it only uses the odometry to get a range and bearing to the goal and nothing else, there are less "bad" decisions it could make. Since odometry is likely to be noisy on very small robots, such simplicity may be the better strategy. Nevertheless, although Com is less influenced by odometry drift, it success rate still drops to 40%, which is a low score. In general, it is ill-advised to have BAs solely rely on odometry alone.

In this chapter we have not experimented with different environment sizes. However,this could also affect the performance of the BAs dealing with sensor noise. In smaller environments for instance, the odometry drift will be less severe since the optimal paths to the goal will be shorter than for larger environments. The focus of this chapter was on the onboard sensors used for position estimation and to fix elements like the environment dimensions in order to properly isolate and evaluate the associating issues with real-world-like conditions. However, for future research we should also experiment with varying sizes of the experiment space.

In section 4.5.3 and 4.5.4, we also assumed that the BAs will also have access to measurements other than odometry. Although a decrease in performance was noticed in all the tested BAs with these specific features (Com1, Alg1 and Alg2), it became evident that Alg2 is the most resilient algorithm. With increasing FN & FP occurrences, Alg1's performance was noticeably decreasing but Alg2 was hardly affected. This indicates that the M-line-Bugs, as Alg1, seem to have a disadvantage over Angle-Bugs, as Alg2, due to their restrictive leave-condition. This is also noticeable in section 4.5.2, as M-line-Bugs suffered the most from increasing odometry drift. If real-world conditions apply, BAs should rather be able to leave the wall/obstacle whenever there is the possibility to do so.

The same goes for noisy distance-to-target measurements (section 4.5.4), where Com1 is performing worse than Alg2. The reasoning behind this observation is simple: Alg2 is using more mechanisms to get out of complex situations, namely remembering range measurements and locations of previous hit-points. If one of these mechanisms perform badly, then Alg2 can fall back on the other one. Now these measures are operating separately and have a different behavioral outcome; however, it could be more beneficial to a BA if they were fused together or used for cross validation and checking if the BA is stuck in a loop. Nevertheless, it is of great interest to have multiple types of measurements to rely on, either concerning position of the robot itself or the relative

position of the goal.

4.7. CONCLUSION

This chapter investigates the potential of Bug Algorithms as a computationally efficient method for robotic navigation. Although the general idea behind the methods seems ideal for implementation of light-weight robots, the literature survey shows that many of their variants rely on either a global localization system or perfect on-board sensors. Our simulation experiments evaluated several implemented Bug Algorithms with varying noisy measurements and failure cases, which showed a significant performance degradation of all algorithms. This indicates that Bug Algorithms cannot simply be implemented as they are on a navigating robot, which has to rely on only on-board sensors without any external help. The experimental results did, however, shed some light on how these techniques can be enhanced. Simplicity is a key element, as the most basic Bug Algorithm, Com, was also the one that was the most resilient to odometry drift. Another crucial element is a robust loop detection system, where the robot should not just rely on one but on multiple measured variables, especially in realistic, noise-inducing, environments. Considering these observations in the design of new Bug Algorithms, will make them suitable for the autonomous navigation of tiny robotic platform with limited computational resources.

5

SWARM GRADIENT BUG ALGORITHM

In Chapter 4, we have looked into a potential navigation strategy for a pocket-drone, namely Bug Algorithms. We did find that previous implementation of this technique over-relied on perfect localization. The navigation strategy would need to fit on-board the pocket drone, should not use any external localization system for positing and not use an external computer as a processing aid. We would therefore need to modify the bug algorithm principle to better adhere realistic sensor noise in the real-world and therefore be suitable for an indoor exploration task.

We developed the Swarm Gradient Bug Algorithm (SGBA), which enables a pocket drone to explore a building and return to the initial position based on the signal strength of the home beacon. Moreover, the pocket drones can avoid each-other based on the signal strength of the intra-drone communication, and — based on their transmitted preferred exploration direction— can also locally coordinate their search. This is the first time that a group of 6 small pocket-drones were able to perform an indoor exploration task, while adhering to all the requirements of full on-board autonomy we have established for this dissertation.

Parts of this chapter have been submitted to:
K. N. McGuire, C. De Wagter K. Tuyls, H. Kappen & G.C.H.E. de Croon, *Minimal navigation solution for a swarm of tiny flying robots to explore an unknown environment*, Science Robotics (2019) [In press].

5.1. INTRODUCTION

SWARMS of tiny autonomous flying robots hold great promise in the field of robotics. Tiny flying robots can move in narrow spaces, can be so cheap that they may become disposable, and are safe in the presence of humans (Floreano and Wood, 2015, Ma et al., 2013). Moreover, while the individual robots may be inherently limited in their abilities both in terms of cognition and in terms of actions, together they may solve very complex problems. This kind of problem-solving ability is abundant in nature. Perhaps the most well-known examples are the shortest path finding by swarms of ants (Reid et al., 2011) and the food location communication by waggle dances of honeybees (Menzel et al., 2012).

The core principle of swarming is that the individual robots obey simple control rules, merely based on their local sensory inputs and local communication with their neighbors. This principle fits well with the limited resources of tiny robots. Moreover, not relying on any central processing promises a high robustness of the system. A single failing robot will not endanger task execution, as its role will be fulfilled by one of the many other robots. In addition, together, small robots will potentially be able to perform tasks quicker and more robustly, such as surveillance, construction, or exploration. In the past few decades, a large body of research has been formed investigating swarm robotics. For instance, in the Swarm-Bots project, controllers have been evolved for small driving robots to complete tasks such as gap-crossing, which requires them to attach themselves to each other to form a bridge, and to move objects that are bigger than each individual (Mondada et al., 2005). Moreover, swarms of robots have been demonstrated in applications ranging from constructing small pre-planned structures (Durrant-Whyte et al., 2012) to forming shapes with their own bodies (Kushleyev et al., 2013, Rubenstein et al., 2014), or performing tasks such as dispersion, aggregation, and surveillance (Duarte et al., 2016).

Concerning swarms of flying robots, the major challenge lies in achieving autonomous robot navigation and coordination between the robots in real-world environments. There are already impressive commercial shows with many simultaneously flying robots, such as Intel's Shooting Star drones[1] that were used in the 2017 super bowl half time and in the 2018 winter Olympics. However, the robots in these shows purely follow pre-programmed GPS-based trajectories, so they do not take local decisions based on their surroundings. In contrast, in Hauert et al. (2011), Vásárhelyi et al. (2014) and, more recently, Vásárhelyi et al. (2018), a swarm of flying robots performed coordinated swarming behaviors together in outdoor environments. In the latter study, the main behavioral parameters were optimized with an evolutionary process such that robots would stay together, even in the presence of no-fly-zones. The studies (Hauert et al., 2011, Vásárhelyi et al., 2014, Vásárhelyi et al., 2018) all still crucially rely on GPS. The flying robots communicate their GPS-locations to each other in order to determine the relative locations to other robots that serve as input to the local controllers. In all the above studies, the swarms essentially fly in open environments, or in the case of Vásárhelyi et al. (2018) have access to a global map of where no-fly-zones are.

Navigation of a swarm of tiny flying robots in a cluttered, GPS-denied environment is

[1] https://newsroom.intel.com/news-releases/intel-drone-light-show-breaks-guinness-world-records-title-olympic-winter-games-pyeongchang-2018

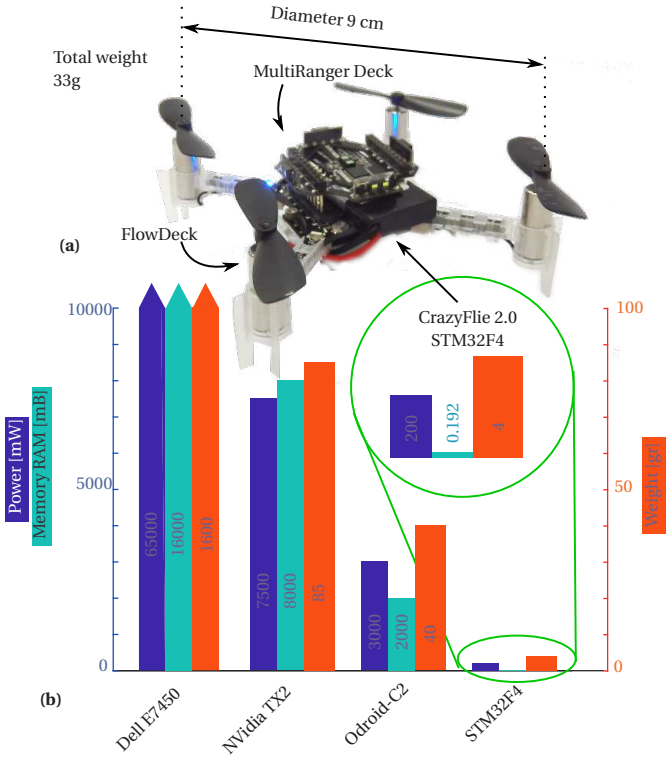


Figure 5.1: Hardware specifications and comparison. a) The Crazyflie 2.0 with the flow and multi-ranger expansion decks and b) the autopilot (STM32F4) compared to the specifications of the Nvidia TX2, Odroid-C2 and a laptop (Dell Latitude E7450). Please note that the Dell specifications do not fit within the chart (as indicated with the top triangles).

until now an unsolved problem. The major challenge derives from the highly restricted nature of these tiny robots. The mainstream solution to navigation consists of Simultaneous Localization And Mapping (SLAM) based on camera images (Fuentes-Pacheco et al., 2012) or laser range finders (Lázaro et al., 2013). However, typical, metric SLAM methods make detailed 3D maps, which is very demanding in terms of computational and memory resources. State-of-the-art SLAM methods like LSD-SLAM (Engel et al., 2014) often need to be computed by an external ground station computer (López et al., 2017). Multi-robot SLAM, in which a group of robots jointly creates and maintains a shared map of the environment (Forster et al., 2013), places an additional load on the communication bandwidth. One can also opt for the slightly more efficient visual inertial odometry (VIO) (Weinstein et al., 2018). However, this is subject to drift. To illustrate the challenge of navigating with tiny flying robots, Fig. 5.1 shows the processing power of the robot used in our experiments (Bitcraze's Crazyflie 2.0) besides two recent, state-of-the-art embedded processing units used in SLAM approaches. Flying robots like the Crazyflie use approximately 7 Watt to fly. To not significantly affect their flight time, the processing should therefore use only a fraction of this power. The Crazyflie carries an

STM32F4 microprocessor, with a clock speed of 168MHz and 192kB of RAM memory. Typical state-of-the-art robots used for autonomous flight (e.g. Jung et al. (2018), Sanket et al. (2018)) use processors like the NVidia TX2, which has a 6-core CPU each with a clock speed of 2GHz, a 256-core NVidia GPU, and 8GB of RAM. Hence, we need to solve the navigation problem with orders of magnitudes less memory and processor-speed. This calls for a completely different navigation strategy.

One solution avenue to efficient navigation is to draw inspiration from biology, for instance by looking at the navigation strategies of honeybees. Honeybees navigate by combining path integration with landmark recognition (Srinivasan, 2011). Path integration is well understood and can be implemented with very limited systems, as in the recently presented AntBot (Dupeyroux et al., 2019). While walking insects can count their steps for path integration, flying insects such as honeybees rely more heavily on the integration of optical flow (Srinivasan et al., 1997). Path integration alone does not suffice for navigation, since it drifts over time. This drift can be canceled by means of landmark detection and visual homing, but it is not obvious how landmark recognition is performed by biological systems. The dominant model is the snapshot model (Cartwright and Collett, 1983), in which pictures are stored of the surroundings and later compared to the visual inputs. Unfortunately, current implementations of landmark recognition still require substantial processing and memory (e.g., Denuelle and Srinivasan (2016)), making it unsuitable for navigation by tiny robots. Moreover, they mostly thrive on texture-rich environments, which is more commonly found in nature, but not as much in repetitive man-made environments.

Biological systems do provide interesting suggestions for arriving at the minimal requirements for navigation. It is evident that the maps created by metric SLAM can be used for navigating from any point to any other point in the map. The navigation strategies followed by insects suggest that it may be possible to save on computation and memory by requiring less accurate maps. Indeed, biological navigation strategies show a parallel with topological SLAM (Garcia-Fidalgo and Ortiz, 2015), in which a robot only stores important landmarks and their relations in terms of distance and direction. This no longer allows a robot to travel anywhere in the explored space with high accuracy, but it is questionable if this is necessary for successful behavior. In fact, in some cases, it may only be important to explore and come back to the "nest", i.e., to only perform accurate homing. A navigation strategy that only demands homing and does not rely on computationally complex visual navigation would have a strong potential for downscaling to tiny robots.

5.1.1. A MINIMAL NAVIGATION SOLUTION

The main contribution of this chapter is a minimal autonomous navigation solution for a swarm of tiny flying robots to explore an unknown, unstructured environment, and subsequently come back to the departing point. "Exploration" here means to move through as large a part of an unknown environment as possible, with as goal to gather application-dependent information. The proposed navigation solution is implemented in a swarm of tiny flying robots and is shown to work in a large real-world indoor environment that has no external infrastructure for exact positioning. Moreover, in the same environment we illustrate how the solution enables a specific proof-of-concept

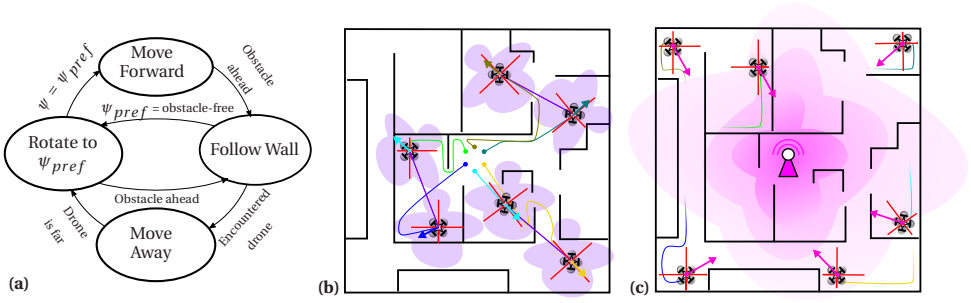


Figure 5.2: Main concept swarm gradient bug. a) A simplified state machine of SGBA derived from the one presented in Materials and Methods. b) The outbound travel of SGBA. The purple shading illustrates the local signal strength around each drone, used for intra-swarm avoidance. c) The inbound travel. The pink shading represents the signal strength of the wireless beacon at the ground station, to which the drones navigate. Note that the intra-swarm avoidance is still active on the inbound flight but is not depicted. The fuchsia arrow at each drone's position illustrates the robot's estimated direction to the beacon.

search-and-rescue exploration mission, in which the swarm has to gather images to find "victims" in the environment.

In particular, we introduce the *swarm gradient bug algorithm* (SGBA). As the name suggests, the method is inspired by "bug algorithms" (Kamon and Rivlin, 1997, Lumelsky and Stepanov, 1986, Sankaranarayanar and Vidyasagar, 1990), which originated as simple maze solving algorithms. The core concept behind these algorithms is that navigating from A to B is not performed by planning in a global map with known obstacles, but by reacting to obstacles on-the-fly as they come within range of the sensors. This way of dealing with obstacles results in a computationally highly efficient type of navigation. However, existing bug algorithms in the literature remain rather theoretical and are not suitable for application to navigation in real, GPS-denied environments since they typically rely on either a known global position or perfect odometry. Examples are the works in Mastrogiovanni et al. (2009), Nguyen et al. (2018), where real-world robots used their wheel odometry for navigation within an indoor environment; nevertheless, the testing environments were too small to experience the full extent of the possible odometry drift. A flying robot typically relies on visual odometry, and due to the vibrations and texture-dependence, is even more prone to odometry inaccuracies than driving robots. When realistic levels of odometry drift are introduced, the navigation performance of bug algorithms from the literature drops steeply (McGuire et al., 2019).

The bug algorithm proposed in this chapter, SGBA, departs significantly from existing bug algorithms, since it has been designed explicitly for allowing a swarm of tiny robots to explore a real-world, GPS-denied environment. Fig. 5.2b shows the main concept: A swarm of robots departs at a base station for their outbound travel. Each robot has a different preferential direction, towards which it will try to go. When robots encounter obstacles, they will follow the obstacles' contours. This process is called "wall-following" in the bug algorithm literature. When the robots' preferred direction is free of obstacles again, they will continue to follow that direction. As soon as the robots' battery

is around 60%, they start their inbound travel (Fig. 5.2c). To come back to the original location, the robots use a mix of (coarse) odometry and – on longer time scales – an observable gradient to the base station. In our experiments, we will use the received signal strength intensity (RSSI) to a radio beacon located at the base station. Both during the outbound and the inbound travel, the odometry is also exploited to detect short-term loops that could result in robots getting stuck in a particular part of the environment. The robots also need to avoid each other and communicate their desired direction to each other. In the experiments, we will use wireless onboard intra-robot communication to both these ends. Specifically, for the intra-swarm collision avoidance the intra-robot RSSI is used, instead of communicating a global position (which is not known by the robots). Moreover, when robots notice the presence of other robots in the direction of their preferred heading, they will adapt their preference, thus enhancing the exploration for the outbound flight.

A simplified version of the finite state machine (FSM) of SGBA can be found in Fig. 5.2a. The SGBA method and the FSM are presented in more detail in Materials and Methods. The innovation of SGBA lies in its suitability for the real-world properties of tiny, computationally extremely restricted robots, and in the combination of the various sub-components. Many of the sub-components themselves have already been proposed in the literature. For instance, traveling towards a wireless beacon with a bug algorithm was also proposed by Taylor and LaValle (2009), Taylor and LaValle (2014). However, the proposed methods in those studies were too sensitive to the real-world noise of RSSI measurements. Due to the difficulties of real-world interference, refraction and scattering of the signal, the experiments eventually involved an infrared beacon instead, which was visible from all locations in the environment. This setup would not be useful in a real scenario. The work in Twigg et al. (2012) does use the gradient of real-world, noisy RSSI values to guide exploration on a real robot (without a bug algorithm type of behavior). However, the platform still required a full SLAM method on-board, as the precise positions of the RSSI samples were needed to estimate the location of the Wi-Fi source. In contrast to these methods, we have implemented a home-beacon search tactic that is able to deal with real-world, noisy, 2.4 GHz, Wi-Fi RSSI values and does not rely on exact positioning. Another example is SGBA's use of multiple robots. The idea of using multiple robots for bug algorithms was first forwarded and studied in simulation by Chinnaaiah et al. (2018). However, they used it to explore the local obstacle boundary and not for efficiently exploring the environment. The swarming mechanisms of SGBA involve (i) the imprinting of different initial preferential directions, (ii) collision avoidance, and (iii) the adaptation of preferential directions when robots notice that their preferred direction overlaps too much with that of another robot. More elaborate swarming mechanisms are possible, but the results will show that these straightforward mechanisms, which do not require accurate relative positions between the robots, already significantly increase the exploration efficiency.

From the explanation above it can be deduced that SGBA requires five main functionalities: (1) following a given direction, (2) wall-following, (3) odometry, (4) intra-robot detection, communication and avoidance, and (5) a gradient-based search towards the departure point. Although we mainly focus on flying robots in this chapter, the five functionalities can be implemented with various types of hardware and software on different

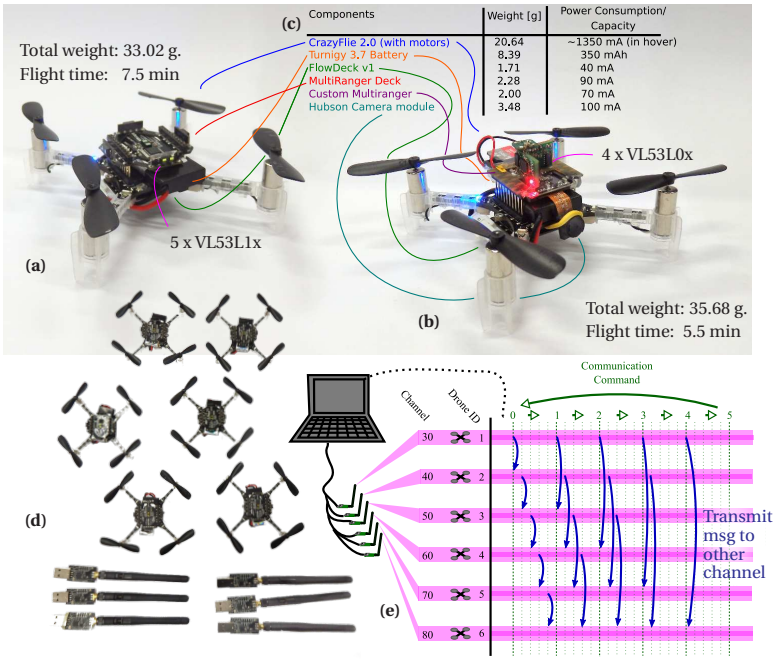


Figure 5.3: Hardware and Communication Specifics. a) The Crazyflie used for the outbound and inbound travel and b) the assembly used for the video recording of the environment. c) The table with all the components on the Crazyflie, including weight and battery consumption. d) The total of 6 Crazyflie's used including 6 Crazyradio PAs and e) the communication scheme shown for the 6-drone experiment. Here a counter is regulating when the drone will transmit a message (msg) to another drone (for counter 1: drone 1 to 2, drone 2 to 3 etc.). Between the regulated counter, the drone will transmit its message to another drone with a time offset based on its ID. Note that currently 6 PAs are used for the 6 communication channels to receive logging of the Crazyflies for the chapter's statistics, however this can be replaced by only one if no telemetry would be required.

types of robots. In fact, while in the real-world experiments we will use flying robots, the simulation experiments detailed below make use of driving robots. The difference in implementation of SGBA includes for instance that functionality (3) is performed with optical-flow-based odometry on the flying robot and wheel-based odometry on the driving robot. Hence, the SGBA algorithm can be applied to different types of mobile robots with limited resources, as long as they are endowed with the above required functionalities.

5.2. MATERIALS AND METHODS

In this section, we will explain the hardware used for the real-world experiments. Afterwards we will explain the exploration and homing strategy of SGBA, starting with the navigation of a single robot and then expanding to larger numbers of robots.

5.2.1. HARDWARE

For the experiments we used Bitcraze's platform Crazyflie 2.0[2], augmented with the flow-deck v2.0[3] and the multi-ranger[4] expansion decks, which can be seen in Fig. 5.3a. An alternative battery with more capacity has been added for a longer flight time, namely the Turnigy Nano-tech 300mAh (1s 45 90C) LiPo battery, providing an average power supply of 3.7 V. To make sure the entire path in front of the drone is free of obstacles using the 20 deg FOV laser ranger, the minimum required detection range is 50cm, for which the VL53L1xs[5] on the multi-ranger are sufficient. The flow-deck contains a PMW3901MB optical flow sensor[6] to detect motion, with an additional VL53L1x for height detection and control. Within the existing state-estimation (Mueller et al., 2017), the Crazyflie can achieve excellent hover and velocity control, as optical flow can be detected on most surfaces. Nevertheless, dark colors should be avoided. The dark low-texture floors in our real-world environment were therefore challenging (Fig. 5.6a), as the flyable height where motion detection was still reliable was only 0.5 meters.

For the on-board video recording experiments, we designed a custom expansion board, which included configuration of the lower power VL53L0x ToF[7] sensors (the predecessor of the VL53L1xs on Bitcraze's multi-ranger deck) and a camera module, meant as a spare part of the Hubsan X4 H107C RC Quadcopter[8]. This camera module carries an SD-card, to record the videos captured during the SGBA navigation of the Crazyflies. This configuration is displayed in Fig. 5.3b. The weight of the platforms and the average power consumption per expansion board are shown in Fig. 5.3c which results in an approximate flight time of 7.5 min for the left-hand Crazyflie configuration and 5.5 min for the right-hand Crazyflie configuration in Fig. 5.3.

To fully execute the SGBA, a communication protocol (Fig. 5.3e) has been flashed into the NRF51 microprocessor, which handles the Crazyradio communication (2.4 GHz Wi-Fi protocol and Bluetooth), and the power distribution. Each drone has its own unique identification number (ID), of which we will consider numbers 1 to 6 in this explanation. This ID will also indicate in which channel (ID*10+20) the Crazyflie will be communicating with the computer for logging the onboard variables, as can be seen in Fig. 5.3e. This separation of channels is done to reduce interference between the Crazyflies. The variable logging of each Crazyflie to the computer was done at 0.5 Hz, which includes the position estimation, the RSSI of the beacon and other robots, the SGBA status of the state-machine etc. In our experiments, since we needed to receive the onboard data of each Crazyflie separately for the statistics in this chapter, each drone had its own individual Crazyradio PA (Fig. 5.3d). This was to reduce the possibility of package-loss of the telemetry data; however, technically only one beacon is necessary. If the Crazyradio PA quickly switches and transmits empty packages on all the available

[2] Bitcraze AB. Crazyflie 2.0, https://www.bitcraze.io/crazyflie-2/
[3] Bitcraze AB. Flowdeck, https://www.bitcraze.io/flow-deck-v2/
[4] Bitcraze AB. Multiranger deck, https://www.bitcraze.io/multi-ranger-deck/
[5] STMicroelectronics. VL53L1X, https://www.st.com/en/imaging-and-photonics-solutions/vl53l1x.html
[6] P. I. Inc. PMW3901MB-TXQT
[7] STMicroelectronics. VL53L0x, https://www.st.com/en/imaging-and-photonics-solutions/vl53l0x.html
[8] Hubsan, HD Camera module 720P

channels, the SGBA will not require any additional knowledge except for the RSSI value.

The communication between the Crazyflies was done by means of a counter to prevent package-loss due to message collisions (Fig. 5.3e). The counter regulates when one drone will send a package to another drone, which will be incremented every 0.5 seconds. For this, it switches to the primary transmitter mode (PT), changes its communication channel to the other drone's channel, sends the message within a short timeframe, and switches back immediately to its own channel in primary receiver mode to receive the messages from the other Crazyflies and receive an RSSI of the home-beacon. In between the regulating counter increments, the moment to switch to PT depends on the drone's ID. This should prevent the Crazyflies to simultaneously send a message, therefore reducing the possibility of intra-drone package-loss. The information that the Crazyflies send to each other, are their ID and preferred heading. This is necessary for changing direction on the outbound travel. At the same time, the receiving drone will also know the signal strength and hereby have an indication of the proximity of the other robot. Not all Crazyflies will send a similar number of messages. The highest priority robots (lower ID = higher priority) transmits to every channel, as all others would need to avoid it, and the lowest priority robot does not send a message at all, since it needs to avoid everybody else.

In the experiments, the earlier-mentioned counter is for now regulated by the computer; however, each Crazyflie would be able to do this by themselves after clock synchronization of the autopilots. The separation of channels on the Crazyradio modules is necessary to enable stable communication between Crazyflies. It should be possible to put multiple Crazyflies on one channel, however the number of usable channels is limited, and the number of robots per channel as well. This poses a restriction for the total number of robots the 2.4 GHz Crazyradio protocol is useful for, however this could be further scaled by using UWB instead and a more sophisticated scheduling protocol such as STDMA as in Coppola et al. (2018).

5.2.2. OUTBOUND TRAVEL

We start our explanation of the FSM with the outbound travel of a single robot. Fig. 5.4a and b illustrate the entire FSM, where the robot starts at "Init". For the outbound travel, it is important to realize that the robot just needs to explore the available space and does not need to go to a specific location. Therefore, it will only be assigned a preferred heading.[9] After it encounters, follows and then leaves an obstacle, it will follow that same heading again (Fig. 5.4c). Of course, there will be heading drift over time. In the case of the Crazyflie robots used in the real-world experiments, the drift was 0.10 degrees / second (48 degrees over the 8-minute flight time). Still, since the main goal of the heading estimate is to send multiple robots into roughly different directions, the drift does not significantly affect SGBA's performance.

After the robot detects an obstacle with its front laser range sensor, it will start the wall-following behavior. First, it chooses an initial "local direction", which decides

[9] In Chapter 4, we made the distinction between Angle and M-line-bugs. However, the outbound-SGBA falls in neither of those category, since it only keeps preferred heading and does not adjust it based on its current location or only moves away from the obstacle if the M-line is hit. The preferred heading is more of an exploration driver to get the bug to move through the building than to go from point A to B.

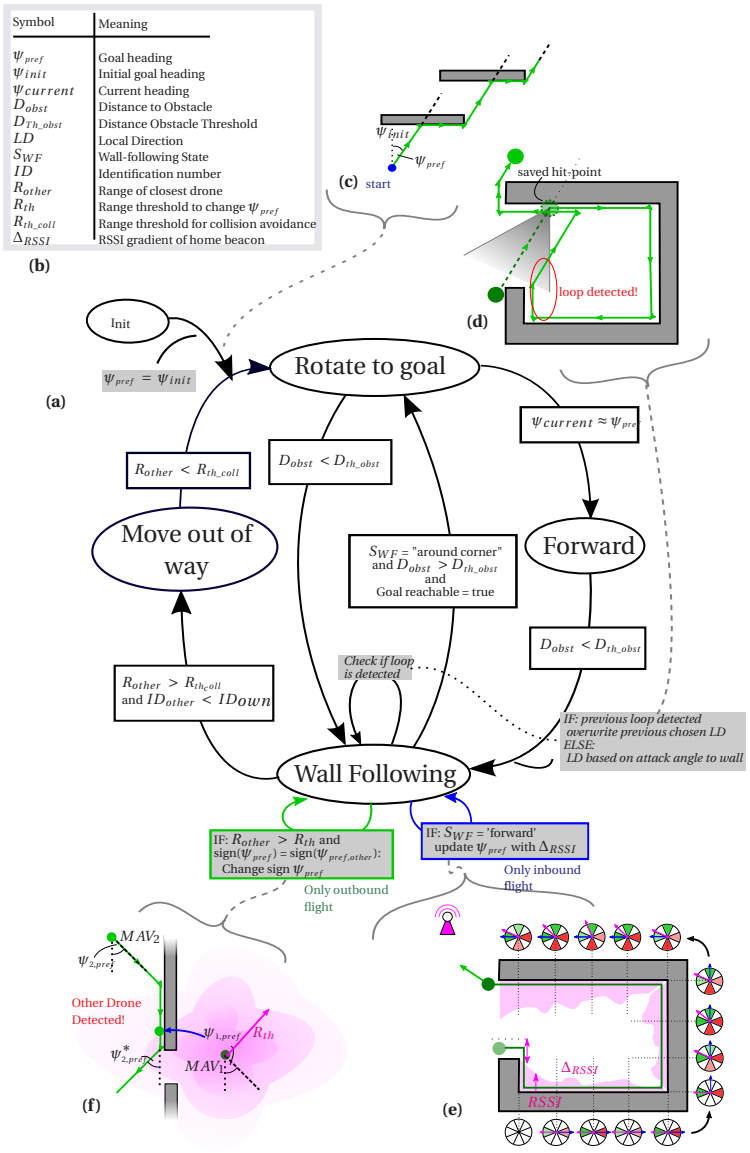


Figure 5.4: SGBA Finite State Machine. a) The finite state machine of the swarm gradient bug algorithm, with a legend of symbols in b). Its individual subsystems are explained: c) outbound navigation based on heading, d) local direction preferences based on angle of attack and the principle of the loop detection. The addition to the state machine of the gradient search of the beacon home location for the inbound travel is given in blue, with e) the gradient search method during the straight parts of the wall-following. Here the robot tries to estimate the direction towards the beacon, by keeping up a score system based on its heading along the way. Swarming gradient bug additions in the state machine f) for the outbound flight (green), where the robot will change his goal heading if the other drone (with the highest priority) has its preferred heading in the same direction. In case the drones are even closer, the one with the lowest priority will move out of the way completely for both the inbound and outbound travel.

whether to follow the wall on the right- or left-hand side. We choose a local direction policy based on the strategies of DistBug (Kamon et al., 1999) and FuzzyBug (Lee et al., 1997), namely by adopting the "angle of attack" as the robot approaches the wall. With the current hardware of the 4 laser-range sensors in all 4 directions of the horizontal plane, the robot can easily determine the angle of the wall by evaluating if the side range sensors are triggered in combination with the front one. The main assumption here is that the wall needs to be straight. However, if this is not the case, this does not mean that the strategy will fail. If the local direction ends up being a less optimal one, this will be corrected for at a later time. From here on, the robot starts following the boundary of the obstacle and the wall.

SGBA uses memory for loop detection. Memoryless bug algorithms are prone to getting stuck in loops, because they may encounter an obstacle, perform wall-following, and then leave the obstacle in a direction which will lead them back to exactly the same obstacle. This will lead to an endless loop, devastating the navigation performance. This is why during wall-following SGBA keeps track of its position relative to the location where the robot first hit the obstacle (termed the "hit-point"). If the robot tracks back, due to the environment characteristics, and crosses the area behind the hit-point, it will detect that as a loop. This means that once the robot leaves the obstacle and encounters another, which is usually the same hit-point as last time, it will not base its local direction on the current wall-angle, but on the reverse of the direction chosen at the previous saved hit-point. This position tracking is illustrated in Fig. 5.4d and is done completely with relative position estimations of the on-board odometry. Since this procedure is only used for local decision-making within small rooms, this should be a sufficient tactic to handle loops within our experiment environment. However, this probably will not prevent a potential loop in large areas, as the drift will then be too severe. We have studied the effect that SGBA's loop detection has on the return rate (Appendix, Text B.7).

5.2.3. WIRELESS-COMMUNICATION-BASED INBOUND TRAVEL

After a few minutes, either after a time threshold has passed or based on the remaining voltage of the battery, the robot needs to return to its base station. This is extremely important for robots that store their measurements onboard and do not stream their results to the operator. To achieve this, SGBA keeps track of the gradient of the filtered received signal strength intensity (RSSI, see Text B.3 for raw measurements) while it is performing the wall following, as seen in Fig. 5.4e. During the straight parts of the procedure, it will have a circular buffer, corresponding to the heading of the robot, where it will keep track in which directions the RSSI has increased over time. This weight will be incremented, while +/- 180^o of that direction will be decremented to give it a lower influence. For an RSSI decrease, the exact opposite procedure will be done and for no RSSI change, the scores will stay the same. Both incrementation and decrementation, based on the RSSI's derivative, is done for every N meter, where N is a decimal number that can be defined by the user. Every N*k meter, where k is a scalar value, a vanishing function is applied, to decrease the influence of older RSSI measurements. This RSSI change in function of the heading allows the robots to estimate the direction to the home beacon, which they

will use for the return travel any time it is not forced to follow an obstacle or wall.[10] As the RSSI increase is noisy and irregular, this will usually not be an exact angle, but coarse indication of where the beacon is. This proves to be enough for the robot to return to its home base. Please note that any drift in the robot's heading estimate is not problematic for the inbound travel, since the direction to the home beacon is determined with respect to this internal heading representation.

5.2.4. MULTI-ROBOT COORDINATION

A single robot could use the SGBA-FSM by itself in order to navigate. However, it only has a limited battery capacity and therefore will not be able to explore the entire environment. For this reason, it is more advantageous to use a swarm of robots. However, using multiple robots poses a new set of problems. Firstly, the robots need to avoid each other, and secondly, it is better to have them coordinating the search with each other in order to achieve maximum dispersion and avoid conflicts. This is all done with their communicated information and the filtered RSSI measurements, and is implemented by the "move out of way" state in Fig. 5.4a. On the outbound travel, each robot is assigned a preferred heading, chosen out of K different directions. The robot with the highest priority (lowest ID) can leave first and always will have right of way. If a second robot comes in the proximity of another, and the RSSI of the connection exceeds the first threshold and the other robot has a similar preferred heading, it will change the sign of its preferred heading and carry on (Fig. 5.4f). The next time it leaves the obstacle, it will therefore move away from the search area of the robot with the higher priority. This element will only occur for the outbound travel, as there can only be one preferred heading for the inbound travel, namely the one pointing to the home beacon. If the moment would arise that the robots come even closer to each other, either if the first avoidance strategy illustrated in Fig. 5.4f is insufficient or both robots are returning, a second RSSI threshold is triggered and the robot with the lowest priority will perform an action that will enable the higher priority robot to smoothly move past it. The avoiding robot will then start moving towards its preferred direction again.[11]

5.3. RESULTS

We have performed both simulation and real-world experiments to gauge the performance of SGBA as an autonomous navigation solution for exploration missions. Navigation here means spreading out in the environment, covering the environment as much as possible, and coming back to the departure point. The two main performance metrics are 1) the area coverage, and 2) the return rate of the robots. With these metrics, we mainly assess the navigational characteristics of SGBA, as these are important to exploration missions in general. We vary the number of robots in order to investigate the advantages of the swarming aspect of SGBA. After the main simulation and real-world experiments focusing on the navigation performance, we will also perform an experiment as a proof of concept of a specific exploration mission. In particular, we investigate

[10] This makes the inbound-SGBA closer to Angle-bugs as depicted in Chapter 5, although the angle towards the goal estimated and can be slightly different at every leave-point.

[11] The different implementations for both the simulated and real-world robotic platforms used later in the experiments can be found in the Appendix as Text B.6

a search-and-rescue scenario in which the robots have to find possible victims in the environment. For this scenario, the robots carry onboard cameras and SD cards for storing images of the environment, since transmitting live streams is not feasible at this scale. When returning to the base station, the robots can upload the images to the base station and a human end user can look at the images to find the victims. Hence, only returning robots will provide useful information on the task. Since this will be the case in many real-world exploration scenarios where SGBA is a suitable method, we consider as a third performance metric, 3) the area covered by returning robots. In the specific search-and-rescue experiment we evaluate whether the victims are present in the images.

5.3.1. SIMULATION EXPERIMENT RESULTS

We first implemented the swarm gradient bug algorithm (SGBA) in simulation. The goal of the simulation experiments is to investigate the performance of the algorithm in many artificially generated environments. Moreover, in simulation we can perform extensive experiments for gathering sufficient statistics on trends such as the relation between the performance and the number of robots. As a simulator, we have chosen ARGoS (Fig. 5.5a), as it has especially been developed for multi-robot systems (Pinciroli et al., 2012). A ROS environment is used to connect the SGBA controller, the automatic environment generator and the simulator together, in a similar manner as in McGuire et al. (2019), by using ARGoS for ROS[12] (Text (B.1 in the Appendix explains the procedure to get the code repositories for the simulation experiments). In simulation the robots are adapted ARGoS foot-bots, which are originally modeled based on the MarXbot (Bonani et al., 2010). These ground-based, non-holonomic, robots are different from the airborne, holonomic, robots used in the real-world experiments. See Text B.6 the Appendix for an overview of how we implemented SGBA's five functionalities on the simulated Footbot and on the flying robots used in the real-world experiments. The use of ground-based robots in the simulation experiments illustrates that SGBA can be applied to different types of robots.

The simulated robots will start around the home beacon in the middle of the environment. With SGBA, each of them sequentially starts moving into their preferred direction, which in this case we consider the angles 45^o, 135^o, -135^o and -45^o (the modulus of the robot's ID from 4 determines the preferred direction). In simulation, the outbound travel will last for 5 minutes. After spreading out into the environment, the simulated robots will try to head back to the home beacon within another 5 minutes (10 minutes of total simulation time), for which they will use the noisy and locally perturbed RSSI of the base station beacon. Fig. 5.5b and c show two examples of the simulated experiments with 4 and 6 robots. We experimented with 2, 4, 6, 8 and 10 robots per simulated environment. Per test configuration, 100 environments were produced with the procedural environment generator, as developed in McGuire et al. (2019). The coverage statistics can be found in Fig. 5.5d and the return rates in Fig. 5.5e. Despite their extremely restricted onboard resources, 10 small robots are able to explore on average 90% of a simulated 20 x 20-meter environment in 10 minutes (dark blue bar in Fig. 5.5d, example trajectory in Fig. 5.5b and c).

The utility of the collective aspect of SGBA is shown by the dark blue bars in Fig. 5.5d;

[12] ARGoS Bridge. GitHub repository. from https://github.com/BOTSlab/argos_bridge.

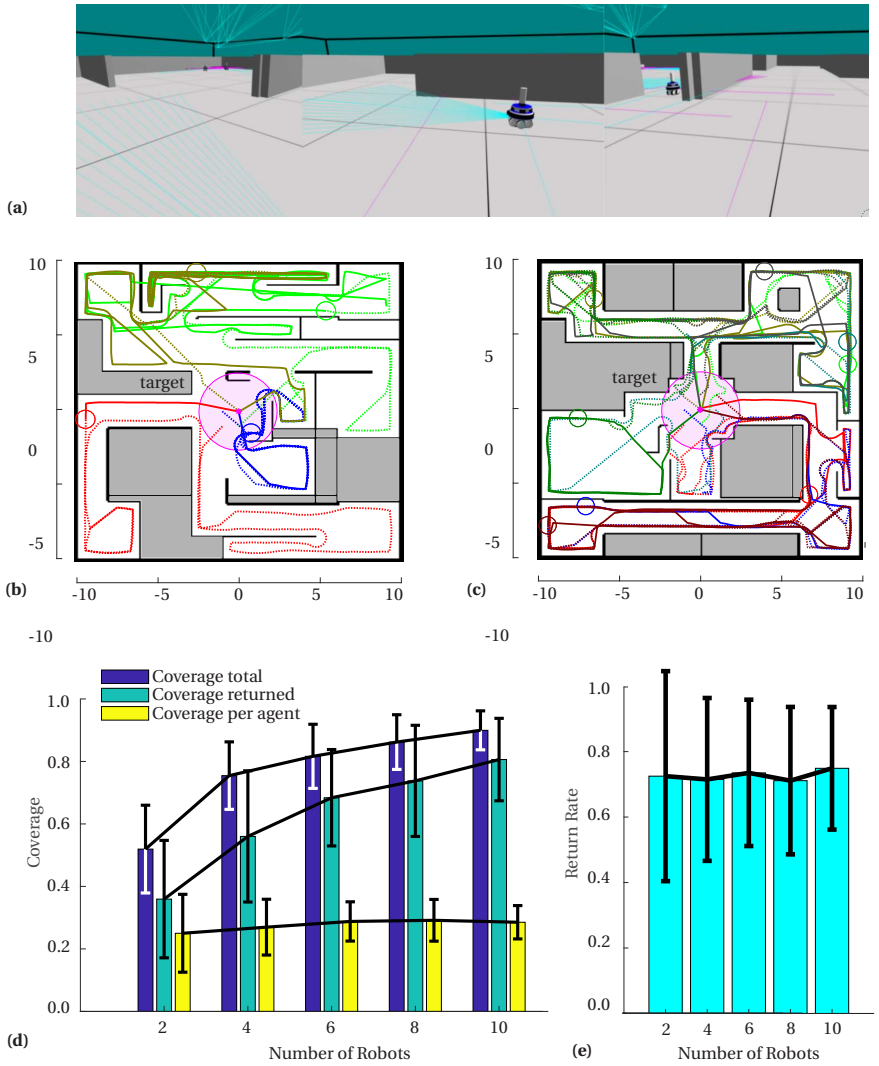


Figure 5.5: Simulation results. The results of the simulation environments with a) a representation of the ARGoS simulator and the modified simulated Footbot. Two example environments and trajectories are shown for b) 6 robots and c) for 4 robots. d) and e) shows the results of [2, 4, 6, 8 10] robots, in 100 procedurally generated environments for each configuration, in the coverage (not including non-accessible areas), and the return rate. Three types of coverage are shown in d): coverage total (area covered by all robots), coverage returned (area covered only by the robots that have returned) and coverage per robot (area that a single robot has covered). The exact computation of the covered area can be found in the appendix in Section B.5.1. Finally, in e) the return rate is shown, i.e., the portion of robots that successfully returns to the base station after exploration. Both bar graphs of d) and e) show the mean as the standard deviation, of which the specific values can be found in Appendix B.5.1.

Adding more robots leads to a higher coverage in the same amount of time. The trend of the bars indicates that the coverage is subject to a law of diminishing returns; Adding two more robots has more effect when going from two to four robots than when going from eight to ten robots. The results suggest that this effect is mainly due to covering the same areas in the environment and not due to robots getting too much into each other's way. Namely, the coverage per robot (Fig. 5.5d, yellow bars) and the return rate (Fig. 5.5e, light blue bars) do not decrease for the studied numbers of robots. The return rate is also of interest for the envisaged proof-of-concept search-and-rescue mission, in which the robots would store images onboard and only robots that return to the base station provide information on the task. The return rate is lower than 100%, mainly because SGBA can lead to suboptimal paths back to the base station (too slow for the total mission time). The coverage by the group of returned robots is shown in turquoise in Fig. 5.5d. Finally, the variances of all performance characteristics become smaller for higher number of drones, showing that adding more agents increases the certainty of the outcome. This seems mainly due to the reduced effect that variations in individual performance have on the total coverage. For instance, with two robots an early failing robot could halve the coverage, while with ten robots the effect would be much less noticeable, underlining swarm robustness. The logging data and statistical tests of the simulation results can be found in Text B.5.1 of the Appendix, which show that the trend of the total coverage and coverage returned are significantly related to the total number of robots. This is not the case for the coverage per robot and the return-rate.

Finally, we have studied the contributions of the different swarming mechanisms in SGBA: (i) sending them off in different directions, (ii) performing an avoidance maneuver when close to another robot, and (iii) changing preferential direction. All three mechanisms contribute to reducing collisions and increasing coverage. For example, for the "full" SGBA with 6 robots there are on average 0.2 collisions per exploration trial. When sending off all robots in the same direction, this rises to 1.7 collisions. When only switching off the avoidance maneuvers, there are on average 1.2 collisions per trial. Text B.7 in the Appendix contains all results of these tests.

5.3.2. REAL-WORLD EXPERIMENT RESULTS

Subsequently, we performed real-world experiments. The goal of these experiments is to show that SGBA works in the real world, and to investigate if the results align with the findings from simulation. In particular, we implemented SGBA on the small commercial off-the-shelf (COTS) Crazyflie 2.0 drone developed by Bitcraze AB.[13] The hardware package of the drones consisted of the following modules. The multi-ranger deck[14] is used for obstacle detection and wall-following. It has four tiny laser rangers that point to the front, left, right, and back. The flow-deck[15] is used for coarse visual odometry. It consists of a downward looking camera that determines translational optical flow and a downward pointing laser ranger that scales the flow to obtain height and translational velocity. Bitcraze's own communication hardware "Crazyradio" with the 2.4 GHz Wi-Fi

[13] Bitcraze AB. Crazyflie 2.0, https://www.bitcraze.io/crazyflie-2/
[14] Bitcraze AB. Multiranger deck, https://www.bitcraze.io/multi-ranger-deck/
[15] Bitcraze AB. Flowdeck 2.0, https://www.bitcraze.io/flow-deck-v2/

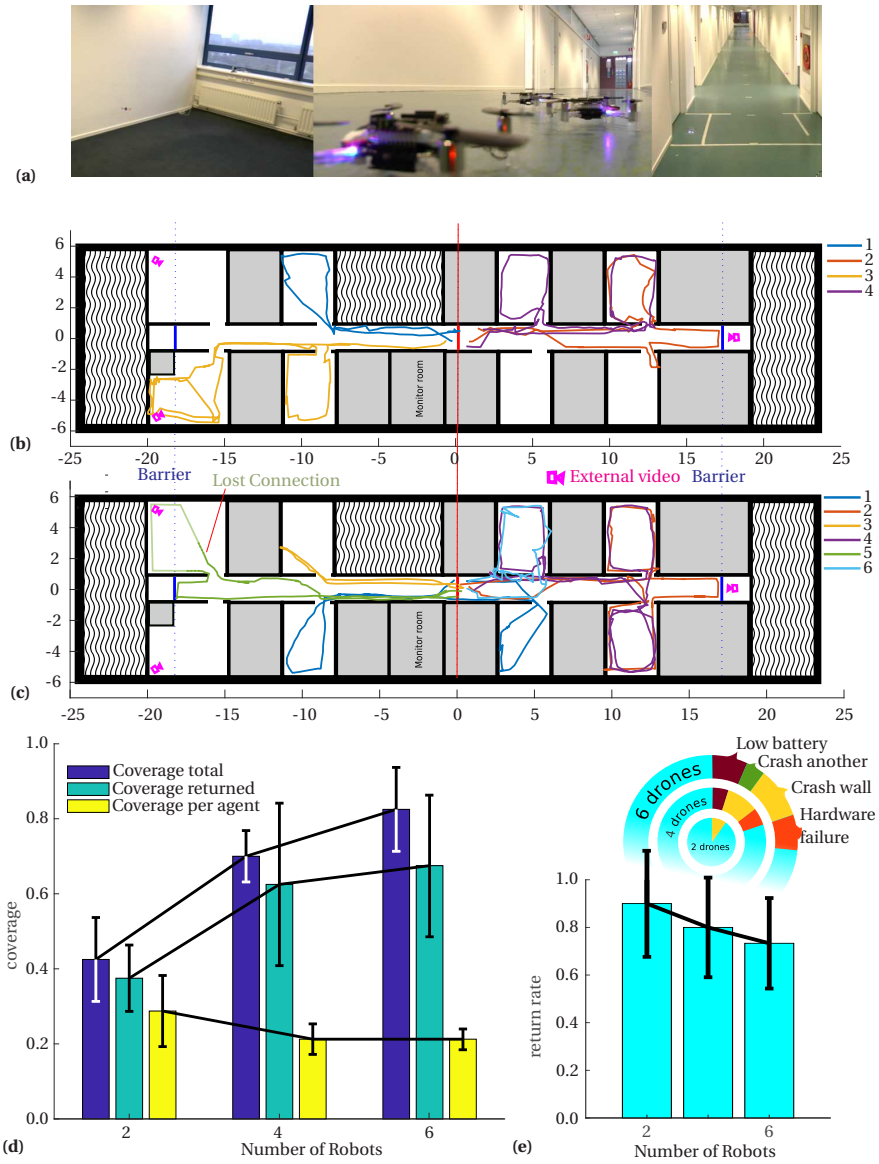


Figure 5.6: Real world results. The results of the real-world experiments with a) a representation of the environment used and the Crazyflie 2.0's with the necessary expansion decks. Several example trajectories are shown for b) 4 robots and c) for 6 robots from their onboard odometry (adjusted by means of the external cameras). The results of [2, 4, 6] robots for 5 flights in each configuration are shown in d) for the coverage (not including the non-accessible areas in gray), and in e) for the return rate (cyan bars), with a pie chart additionally indicating the percentages of real-world related issues which prevented a successful return to the base station. Both bar graphs of d) and e) show the mean as the standard deviation, of which the specific values can be found in Appendix in Section B.5.2.

band[16] is used for ranging to other drones and to the wireless beacon. It also serves as a communication channel between the drones for exchanging desired headings. These three light-weight and low-power hardware modules were sufficient for our navigation solution. We performed the experiments in an empty hallway of the faculty of Aerospace Engineering at TU Delft, as it allowed us to perform extensive testing in the real world (Fig. 5.6a and see Text B.2 for a more detailed description). We conducted real-world tests with 2, 4 and 6 Crazyflies at the same time. For each number of drones, five different flights were performed.

As in the simulation experiments, the robots started in the middle of the environment. From here, they flew with SGBA to their own preferred outbound flight direction for about $\frac{1}{3}$ of their battery life, which was about 2 minutes. Afterwards, they needed to return, again using the RSSI of the home beacon. Along the entire path, they avoided each other by using the RSSI of the intra-robot connection, which was handled by the communication scheme presented in Materials and Methods.

SGBA also allows tiny robots in the real world to successfully explore the environment, with 6 tiny drones on average flying into 83% of the open rooms in the 40 x 12-meter environment within 7.5 minutes (dark blue bar, Fig. 5.6d). Fig. 5.6b and c show two example trajectories with 4 and 6 Crazyflies, respectively. The trajectories show that when a drone enters a room it typically flies along its complete boundaries. Hence, upon entry we consider the room "covered". Please note that the rather accurate trajectories in Fig. 5.6b and c have only been re-constructed for visualization purposes, and do not play any role in the navigation. The trajectories are plotted based on the coarse onboard odometry, adjusted with the video footage of the external cameras on the scene using post-processing (Appendix's Text B.4 explains the procedure and shows the difference between the original and adjusted odometry). The trajectories show how generally, the drones explore different parts of the environment thanks to the different preferential directions. In Fig. 5.6c, an example can be seen where drone 5 loses connection with the beacon in the far top-left room of the environment, so therefore the external camera had to provide the additional trajectory information. Please note that losing the connection was no problem for the autonomous navigation, as the FSM runs onboard of the Crazyflie. The visual odometry and wall-following behaviors allowed the drone to escape the room and reconnect with the home beacon. Video compilations will be accommodating with the original publication that this chapter has been based on.

Since we do not have access to ground-truth global coordinates, in the real-world experiments we determine the coverage performance in terms of the number of visited rooms, excluding the hallway. Fig. 5.6d shows that, as in simulation, a clear trend can be seen that the total coverage increases with the number of drones. However, the increase of the coverage by returned drones seems less steep than in simulation. The reason for this is that both the coverage per robot and the return-rate (Fig. 5.6e) slightly decrease when adding more Crazyflies. This is due to many issues that occur in reality, such as hardware malfunctions, sensing failures and even – for 6 drones – a collision between two drones (see the pie chart of Fig. 5.6e). Having a limited battery capacity is a real-world problem as well but was taken into account in the simulation in the form of a time limit for the results in Fig. 5.5. Concerning the collision avoidance, in all fifteen real-

[16] Bitcraze AB. Crazyradio PA, https://www.bitcraze.io/crazyradio-pa/

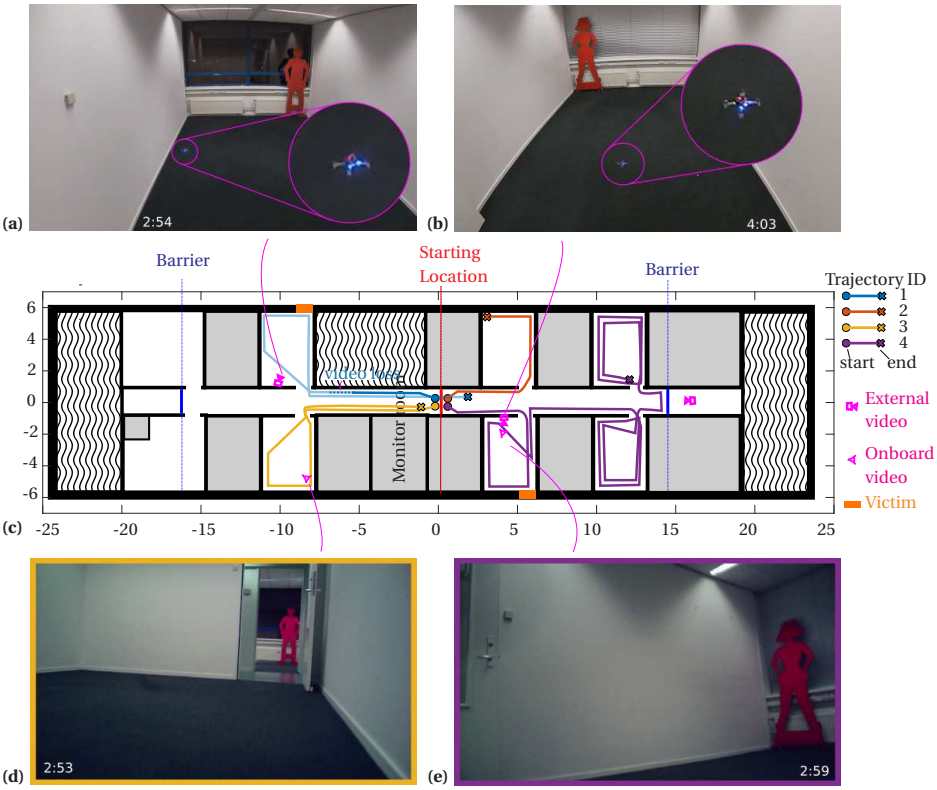


Figure 5.7: Proof-of-concept search-and-rescue mission. This figure presents the results of the experiment in which the Crazyflies carry a camera to detect "victims" in the environment. a) and b) shows the screenshots of the external cameras capturing the MAVs during their flight, c) shows the trajectory of the 4 Crazyflies (inferred from the onboard and external camera), while d) and e) show the screenshots of the onboard Hubsan camera, with the two human-shaped silhouettes captured during the exploration flight.

world experiments combined there were 54 encounters between drones and only one collision. This corresponds to a 98% success rate of the implemented avoidance maneuver. The logged data and statistical tests of the real-world results can be found together with a table with the numbers of encounters and collisions in the Appendix as Text B.5.2. Also the real-world results show that the trend of the total coverage is significantly related to the total number of drones.

5.3.3. PROOF-OF-CONCEPT SEARCH-AND-RESCUE MISSION

Finally, we have performed one experiment in which we applied SGBA to a specific search-and-rescue exploration mission. The goal of this experiment is to serve as a proof of concept of how SGBA can be used for a specific exploration mission. To this end, the light-weight Crazyflies carry a mission-relevant payload. In particular, we additionally equipped the flying robots with a forward-looking camera and an SD card (see Materials and Methods for the exact setup), which resulted in a flight-time of 5.5 minutes in total.

This extra camera allowed storage of images captured during flight on the SD-card, for inspection by a human end-user. While the proposed navigation solution works with COTS Crazyflie modules, for the proof-of-concept search-and-rescue mission we had to make a custom, lighter weight and lower power laser ranging module in order to accommodate the extra camera and SD card. This custom ranging module replaced the Crazyflie's multi-ranger module.

The experiments simulated a search-and-rescue scenario, in which two human-size wooden figures were placed in two different rooms in the hallway. The same starting position was used for the 4 camera-equipped Crazyflies as in the previous testing set-up. To cope with the limited flight time of the prototypes, it was chosen to perform only the outbound flight. The trajectories in Fig. 5.7c were inferred from the onboard-camera footage, in combination with the external cameras.

Both "victims" were found by the drones. In Fig. 5.7a and b, we can see that Crazyflie 1 and 4 were flying in the rooms where the victims were located. Drone 4 was able to capture the victim on its onboard camera (Fig. 5.7e). However, Crazyflie 1 suddenly stopped recording, right before it flew into the room with the victim. Luckily, the victim was spotted by drone 3 from another angle in Fig. 5.7d. This example shows the advantage of using swarming, which can lead to redundant observations in an exploration task. A video compilation can be found in the original publication of the source of this chapter.

As explained, the drones do not make a map of their environment for navigation. After the drones come back, their collected images can be downloaded to the base station. A human end-user can then go through the images, and, when finding a "victim", look at the video of the robot in fast-forward to find out where the victim is located. If a map is desired by the end user, the base station computer could also generate a map based on the onboard images with state-of-the-art SLAM methods (e.g., Engel et al. (2014)). This map generation may be challenging though, due to real-world factors such as quick motions, motion blur from vibrations, lack of texture, etc.

5.4. DISCUSSION

The purpose of this chapter was to present an alternative for navigating a group of tiny, resource-limited flying robots in an unknown, GPS-denied environment. We presented the SGBA, which fit onboard the commercially available flying Crazyflie robot, weighing a mere 30 grams. With the STM32F4 microprocessor and added sensing capabilities (multi-ranger and flow-deck expansion decks), it navigated through a real office environment. Moreover, while communicating with its peers, it avoided other Crazyflies and increased coverage in the overall exploration task. The algorithm enabled a group of small and limited flying robots to fully autonomously navigate in a real environment by using their on-board sensors and processing capabilities. Still, there are several elements to consider for extending this work to bigger and more complex environments as seen for instance in real-world search-and-rescue scenarios. This section will reflect on the limitations, their impact on the results and possible improvements, and on the choices made in terms of sensing, processing and navigational strategies.

There are several options that would allow for a better navigation performance. First, we expect that using Ultra Wide Band (UWB) instead of the Crazyradio PA would significantly improve both the localization with respect to the beacon, and the sensing of other

drones (see e.g. van der Helm et al. (2019)). The Crazyradio is heavily influenced by other 2.4 GHz sources (Wi-Fi) and the intra-Crazyflie chatter and the RSSI-based distance measurements are particularly noisy and heading-dependent. The good overall results show the robustness of SGBA. Still, communication could be significantly improved by using UWB communication with time-of-flight ranging. For instance, a single DecaWave DWM1000 module can provide ranging to another such module for a distance up to 290 meters, through walls and obstacles, with 10 cm accuracy[17]. Since SGBA only needs ranging to a single beacon, it can use the full extent of the UWB range to significantly improve the inbound flight.

Also, the collision avoidance between drones will benefit from using UWB. The experimental results showed an increase in coverage when adding more drones to the swarm. However, the increase follows a law of diminishing returns. We expect this phenomenon to be fundamental, as having more robots necessarily means covering more of the same area when they start from the same point, and it also means that robots will spend more time avoiding each other. Still, in our current implementation, using too many drones also led to communication-overload during the flight. In our current communication scheme (see Materials and Methods), the more drones there are, the slower they can communicate with each other. Increasing the number of drones can cause problems of miscoordination, or even worse, a higher likelihood of inter-drone-crashes. We saw in the real-world results (Fig. 5.6e) that the latter did not occur with 2 and 4 drones, but it did once with 6 drones. The optimal number of drones will depend on the size of the environment and – with the current implementation of intra-swarm avoidance – on the communication hardware and protocol. Due to UWB's greater robustness with respect to interference and its higher throughput, we expect it to also improve the scalability of the current proposed scheme for drone collision avoidance.

During the real-world experiments, there was almost always a connection between the home beacon and all drones. In Fig. 5.6c, a disconnected drone kept executing its tasks autonomously, since the FSM runs fully on-board. It was therefore able to get out of a communication dead-zone eventually. However, the question still arises: How will the robots still be able to get home if the beacon is lost completely? Even with the earlier mentioned UWB improvement, there are situations where the environment is larger than the range of the beacon. A useful addition to cope with the home-beacon-loss problem in bigger environments is to make more use of the swarm. As the Crazyflies are communicating with each-other, they can also be used as a beacon themselves. As soon as a drone loses connection with the homing beacon, it could try to find another Crazyflie that is still connected to the beacon and navigate towards that position first, reconnect with the original home-beacon and resume its navigation to the starting position (a strategy that reminds of "chains" of robots as used in, e.g., Dorigo et al. (2013), but that would be more economic in terms of the number of used robots). Yet, this requires that at least several Crazyflies always need to stay connected to the home beacon and therefore are limited in their own mission.

Improving the robots' sensing capabilities would also improve the results. Specifically, the multi-ranger deck proved to be sufficient in the environment we tested in, yet there are limitations. For instance, it cannot see very thin objects and relies on the flow-

[17] Decawave, "DWM1000 Productbrief p. 1."

deck to work properly. Although the flow-deck and the existing sensor fusion provided stable velocity-driven flight, the dark floor in the office environment turned out to be challenging. Therefore, situations occurred in which the Crazyflie would drift and move into a direction where obstacles were present in the blind spots of the multi-ranger. Of course, a higher robustness to collisions by means of a protective cage as proposed in Mulgaonkar et al. (2018), would help. Still, the wall-following and obstacle detection can also be made more robust. A possible solution is to add a light-weight vision system, as in De Wagter et al. (2014), Palossi et al. (2019). Vision can provide distance estimates in an entire field of view and aid the velocity estimation and odometry by means of frontal optical flow, cf. McGuire et al. (2017). This would reduce problems with textureless floors. Even so, a fundamental limitation of using the multi-rangers, cameras or optical flow sensors, is that they will be ineffective if the surroundings are filled with smoke. In that case, different sensors such as sonar or radar can be used, while the navigation can remain identical.

The high efficiency of SGBA in terms of sensing, computation, and memory comes at the cost of navigation efficiency. Not building a global map and not performing computationally expensive optimal path planning results in suboptimal paths. Drones can revisit rooms multiple times or can visit rooms that were already visited by other drones. This could perhaps be solved in a relatively efficient manner, e.g., involving visual landmark recognition. Still, the experimental results have shown that multiple measurements from the same area can be beneficial. Camera footage can get temporarily occluded or even lost, as happened with drone 1 in Fig. 5.7c. Moreover, the fact that drones' views overlap with each other can make a significant difference in data collection if not all the robots are able to return.

We illustrated the potential of SGBA by implementing it on the smallest possible commercially available quadrotor. However, the discussion above suggests that the method would perhaps be even more successful on a custom-designed drone. One option is to implement SGBA on a smaller platform, while keeping a similar performance. Making SGBA work on a smaller drone is possible, since a custom design would not have to be as modular and easy to use as the Crazyflie decks. That this is possible is already shown by the lighter and more energy efficient custom laser ranger deck that was made for the proof-of-concept search-and-rescue mission. Another option is to implement SGBA on a slightly bigger drone for better performance. We expect that using a slightly bigger drone with better sensing, communication devices, and more battery capacity would significantly improve the return rate of the drones, since it would reduce collisions both with obstacles and with other drones, make the inbound flight more efficient, and extend the flight time available for returning. Even if such a drone could have a bit more processing available, the current proposed navigation solution remains of high interest, since it will leave much room for other types of functionalities. This can be used by vision algorithms to enhance the navigation, or by other algorithms performing mission-specific tasks.

In the future, more processing power will become available to small robots (see, e.g., Jones et al. (2018)). In comparison with 3D SLAM, SGBA will always be available to smaller robots. For instance, it is not unthinkable that SGBA can one day be applied to the 80 mg Robobee (Jafferis et al., 2019). Furthermore, small robots will have to use

their onboard computing power for all tasks they need to perform autonomously. It follows that it is essential for small robots to have computationally efficient algorithms for all tasks they perform. Using SGBA implies that there is more computational power and memory available for other mission-relevant tasks. Hence, we expect SGBA to remain relevant even with the further progress in the miniaturization of computing devices.

5.5. CONCLUSION

To conclude, we presented a minimal navigation solution, the swarm gradient bug algorithm, that allows tiny flying robots to successfully explore a real-world environment. The flying robots only use 4 tiny one-dimensional laser range finders, 1 optical flow integrated circuit and a very light 2.4 GHz radio. The processing fits easily in the single 32-bit 168MHz 196kB RAM micro controller of the Crazyflie on top of all the flight control code. Instead of building a map of the environment like conventional SLAM techniques, our navigation solution consists of an intricate combination of simple behaviors and behavioral transitions, in order to accomplish the complex tasks of autonomous exploration and homing. The guiding principle here is to trade off properties such as path optimality and accuracy with resource efficiency, allowing for swarm operations. We believe that this principle can offer inspiration for solving other complex robotic tasks as well with swarms of cheap and safe tiny robots.

6

DISCUSSION

At the start of the work done in this dissertation, there was no known implementation of an indoor autonomous navigation strategy for multiple flying robots with a weight less than 50g. Such a system could have great potential for surveillance tasks such as search & rescue, green-house monitoring and pipeline inspection. Therefore, the main research question for this dissertation was formulated in the introduction as follows:

MAIN RESEARCH QUESTION

To what extent can we design a robust and computationally efficient method for multiple pocket drones to explore an unknown, indoor environment and to return to their initial position?

The requirements for the main research objective are the following: the pocket drones are only allowed to use their on-board sensing and processing capabilities. Therefore, they are not allowed to use any external positioning system and not allowed to use an external computer for memory, processing or guidance. This discussion will answer the main research question by first addressing the smaller sub-questions in Section 6.1. Afterwards we will step back and discuss the trade-off made in this dissertation between capabilities, hardware and energy (Section 6.2). We will then discuss the choices made in terms of perception (Section 6.3), communication (Section 6.4) and navigation strategies (Section 6.5). This chapter will end with a conclusion in Section 6.6 which will reflect on the achievement of the main research objective, and several thoughts and ideas on future work in Section 6.7.

6.1. SUB-QUESTIONS

In the introduction we divided the main research-question into two sub-questions, which we will evaluate by means of their fulfillment of the requirements given in Section 1.2.1. The first sub-research question is formulated as follows:

SUB-RESEARCH QUESTION I

To what extent can we achieve low-level navigation capabilities on multiple pocket drones, e.g. ego-motion estimation, obstacle avoidance and inter-drone avoidance?

Show-casing Edge-FS on a pocket drone in Chapter 2 is a major contribution of this dissertation for low-level-navigation of a single pocket drone. Before, optical flow detection and velocity estimation was mostly done with downward-looking sensors and cameras. Nevertheless, for small limited flying platforms researchers need to be conservative with weight. Adding another sensor for horizontal obstacle avoidance is usually challenging because of this limitation. Implementing Edge-FS on a front-facing stereo-camera with the ability of detecting both obstacles and velocity, will open new possibilities for autonomous navigation, as there is simply more to see looking forward than looking downward. The pocket-drone became a platform capable of flying by itself in an office-like room, fulfilling almost all autonomy requirements. The side-note here is that in the horizontal plane, the pocket-drone was fully autonomous, but for the vertical plane it still needed throttle command from a remote control, as it did not have a reliable height estimate. However, in general the ability of forward-looking perception will enable future researchers to add more functionalities to the vision algorithm to increase the autonomy.

Chapter 3 added inter-drone avoidance, extending the low-level-navigation to multiple pocket drones. We show-cased several pocket drones localizing each other by means of a small on-board communication chip. This eliminates the need of using an external positioning system for multiple MAVs, even for these small and limited platforms. Moreover, we also showed that the on-board relative localization scheme is possible even with the noisy received signal strength intensity (RSSI) of Bluetooth, as long as on-board velocity estimates are available. The combination of the low-level-navigation of a single drone (Edge-FS), and the inter-drone localization and avoidance, resulted in two platforms capable of flying autonomously in an indoor environment. However, due to hardware issues, one of the pocket drones needed to be assisted in flight by giving it additional velocity guidance commands.

Although the low-level-navigation strategy of Part I was only partially fulfilled, we were still able to go to the next sub-research question, namely:

SUB-RESEARCH QUESTION II

To what extent can we achieve high-level navigation capabilities on multiple pocket drones, e.g. exploration, coordination and homing?

In Part II, we investigated Bug Algorithms as a computational efficient alternative to other types of navigation strategies. Although in Chapter 4 no physical tests were performed with real platforms, the simulation experiments were crucial in understanding this navigational paradigm. We contributed with the first comparative study that really focuses on the real-world issues in hundreds of procedurally generated environments, therefore providing statistical analysis that shows the areas of improvement for future

bug algorithm-based robotic navigation. In Chapter 5, we developed a new bug algorithm, called the swarm gradient bug algorithm (SGBA). We used the knowledge gained from the previous comparative study to introduce elements to the bug algorithm principle to make it more suitable for operation in a real-world environment. These elements include the beacon-based RSSI-gradient navigation and the loop-detection on the fly. We showed the first full on-board exploration task of 6 pocket drones in an indoor environment, and therefore presented the SGBA method as a suitable alternative for navigation of limited platforms.

Although the two sub-questions were mostly fulfilled, the connection between them is less apparent. We were not able to use the solutions developed in Part I for the fulfillment of sub-research question II. The pocket drone had very little flight-time left and many robustness issues, which forced us to choose the commercially available Crazyflie with better low-level-navigation qualities but more limited in its distance perception capabilities. The transition of hardware for the progression from low-level-navigation to high-level-navigation seems to be at the core of this discussion. Researchers working with these limited systems should be made aware of the fundamental trade-offs between capabilities, hardware and energy.

6.2. CAPABILITIES, HARDWARE AND ENERGY

The build up of the capabilities necessary for autonomous indoor exploration on limited pocket drones, reflects directly on the used hardware. In Fig. 6.1, the used platforms for Part I are being shown. Each paper presented in Part I used a different sensor-combination, and the total number increased as time progressed. Immediately noticeable is that this had a significant effect on the total flight-time. This peaked at around 8 minutes for the first platform (Chapter 2), decreased to 6 minutes for the second, to 3 minutes with the first platform of Chapter 3, to finally end up at a mere 1.5 minutes of the last Lisa-MXs based platform. The more sensors/hardware we added over time; the more flight-time needed to be sacrificed. This overview is important matter for the discussion, since the last state posed to be a significant threat to the final goal of this dissertation. A mere flight-time of 1.5 minutes means that the pocket drones will not be able to explore much of the inside of a building, even if they are with a swarm. It was necessary to maximize the flight-time, which introduced the switch to the Crazyflie for Part II. The flight-time went back to 8 minutes in Chapter 5. Although the SGBA's processing inflicted minimal load on the on-board computer, we still had to make concessions on the hardware. We switched from the camera and the laser-range-ring to only the multi-ranger deck and combined the communication capabilities of telemetry (formerly on Wi-Fi) and inter-drone communication (formerly on Bluetooth) on just one module (Wi-Fi).

As the low-level navigation and the high-level-navigation are so intertwined with each other, we will explain the considerations made in terms of perception, communication and navigation. The first two sections are quite focused on the physical element of the project: the evolution of the hardware and the effect it had on the obstacle avoidance, state estimation and inter-drone coordination. Afterwards, we will investigate the new navigation strategy we proposed and how this differs with other more common techniques in terms of optimality. We hope to make the reader understand the

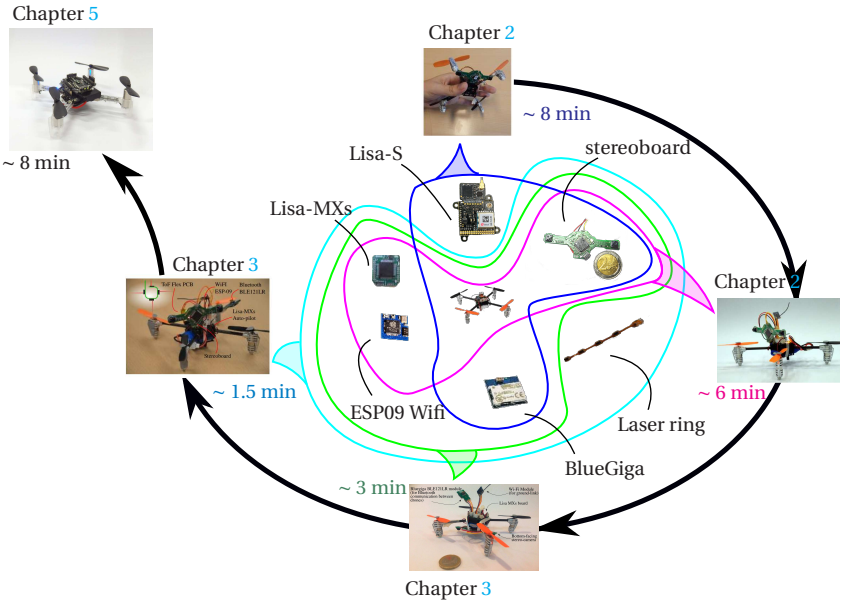


Figure 6.1: Visualization of the hardware development of the pocketdrone. It shows the combination of hardware-modules used of the pocket drones per chapter in Part I and the effect it had on the total flight-time. This is an illustration that marked the choices to go for the Crazyflie platforms in the final chapter.

delicate balance between hardware and software, which is something future researchers on this topic will need to deal with on a regular basis.

6.3. VERSATILE TO DEDICATED PERCEPTION

Throughout the majority the work of this dissertation's Part I, the pocket drones have used the stereo-camera as the main sensor for both state-estimation and perception. In Chapter 2, it first looked downward to detect the motion while giving a crude estimation of the distance height measurement. Once the stereo-board was looking forward, the pocket-drone used it to both detect velocity and the obstacles in its path. The reason for this assembly is clear: a camera is able to detect multiple variables and is therefore considered a versatile sensor. A versatile sensor, opposed to using several dedicated (single-purpose) sensors, usually does not require as much energy than the sum of the single-purpose sensors for the same use. It is usually assumed that designing low-energy solutions are always beneficial for small platforms as the pocket-drone and the Crazyflie.

Unfortunately, we have experienced that this is not the rule of thumb. Whether designing low-energy solutions is beneficial or not, is interlinked with the capabilities necessary for the overall mission. Versatility usually does not coincide with robustness. Taking an example of motion-detection: a camera can infer optical flow from its video-stream; however, a dedicated optical flow sensor is much more reliable as it has been designed optimally to do exactly that. Obviously, it suffers from similar limitations such as cameras, since they are both optical based and therefore sensitive to low-lighting con-

ditions. However, usually the dedicated optical flow sensor has a bigger operating range, as in, it can handle lower texture- and lightning-conditions than the camera could, given the restriction that it is looking at a flat surface. For a pocket-drone, that means that such a dedicated sensor can significantly improve the state-estimation and therefore makes it a more stable platform. Switching from the forward stereo-camera for velocity estimation on the Lisa-MXs based autopilots (Part I), to the optical flow sensor on the flow deck of the Crazyflie (Part II), enabled to have hover- and velocity-control that the designer can trust with her/his eyes closed. Since the stereo-camera had to compute both the obstacle field and velocity at the same time, the velocity estimation's quality was not as good and made the pocket-drone drift into the direction were obstacles could not be detected. It must be noted that if this project had more time and resources to invest in perfecting forward-facing visual velocity estimation and increase the technology-readiness-level by thorough testing and bug-fixes, it would have had the preference to keep on using the stereo-camera for the high-level-navigation. If the stereo-camera would have needed to detect additional variables, such as the recognition of people or cracks in buildings, it would have the capability to do so. However, considering the final objective of the dissertation and the limitation of time and resources, the choice to switch to dedicated sensors was necessary.

Once stable velocity control is achieved, you can go to the next step, namely obstacle avoidance. For this we go back to Part I, as the stereo-board performed both the velocity estimation and stereo vision. The obstacle detection was sufficient for the pocket-drone in Part I to fly autonomously in a room, and it was able to detect multiple obstacles in one frame, but it was limited to the field of view (FOV) of the camera. The cameras on the stereo-board on the lisa-MXs pocket drones, only had a FOV of about 60deg, which reduced the stereo-vision's FOV to only 40 degrees. The imperfect velocity estimation discussed in the last paragraph, caused the drone to have sideways drift and had a higher chance to fly into obstacles (Chapter 2), which was especially the case for low-texture environments. This is still manageable for the navigation flight of one pocket-drone; however, once more is added to the mix, the robustness of one becomes even more important. Therefore, in Chapter 3 we added extra laser range sensors as a fail-safe: to prevent the sideway drift to result in a crash. These laser range sensors we used, did have problems when it encountered dark and reflective surfaces and had a much smaller FOV. However, when you look at the total detection range, these sensors can be placed in any angle and would also work in low texture environments. Moreover, the multi-ranger deck requires less power than the stereo-camera. Although this switch of the primary navigating sensor was necessary to achieve the results of Chapter 5, for future applications it does have value to stick with the camera solution for forward-looking perception, once the same level of robust flight can be achieved.

Fig. 6.2a shows the evolution of the hardware used for perception. Here it can be seen that the pocket drone first started out with one versatile sensor for both (primary) obstacle detection and state estimation, and later added the laser ring for secondary obstacle detection. As this drew too much power and still did not produce a platform that was robust enough to use in a swarm, I switched up the tasks and priority. For motion detection, I switched to the optical flow sensor (flow deck) of the CrazyFlie for state-estimation, and for the laser-range-finders (multi-ranger deck), I switched its task to be

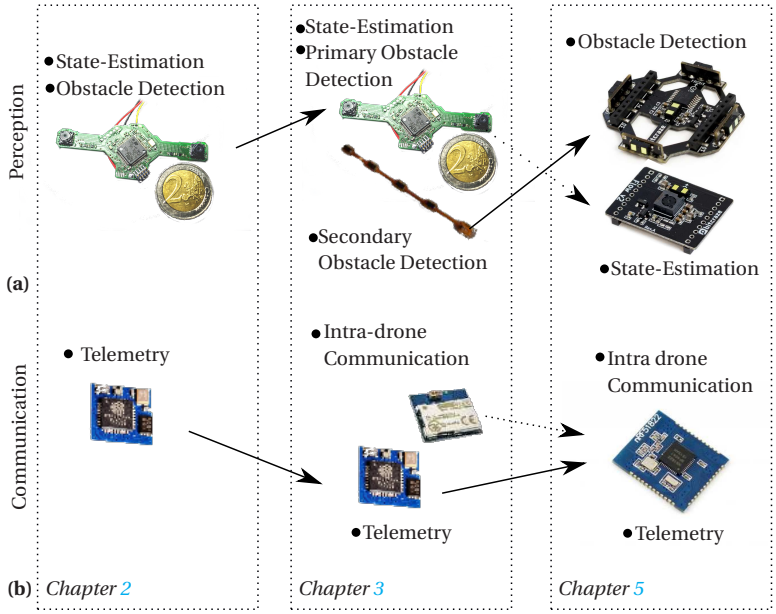


Figure 6.2: Visualization of the hardware development of certain capabilities of the pocket-drone throughout this dissertation. a) shows how the pocket drone's perception went from the use of a single stereo-camera for primary obstacle detection, to the addition of a laser range ring for extra robustness. Eventually the multi-ranger was promoted as the solely obstacle detection and the flow deck for only state estimation (versatile to dedicated sensing). b) shows how the pocket-drone first used a single Wi-Fi-chip for telemetry, and afterwards an extra Bluetooth module was added for inter-drone communication. These two functionalities were eventually merged into one Wi-Fi-based module (dedicated to versatile sensing).

the sole obstacle detector. The Crazyflie and its extension decks have a much higher technology-readiness-level, as they have a large community of hobbyists and researchers that are using them. This is the main difference compared to the stereo-camera and the laser-ring used in Part I, as it was mainly developed for the use within our own flight-lab. However, building the pocket-drone with custom hardware does enable the designer to optimize its functionalities and save weight. The Crazyflie is meant to be modular, and therefore use attachment pins, which are easy to use but weigh more than directly soldered light-weight copper wires. Nevertheless, the need to design navigation strategies for a swarm drew me towards a platform that has been tested by many and proved itself to be a stable platform. Although in theory it would have been a better idea to stick with the stereo-camera for both state-estimation and obstacle detection, the conscious decision was made to change the platform and to split those functionalities into two dedicated sensor-modules. Unfortunately, this also resulted in concessions that had to be made elsewhere, namely communication.

6.4. DEDICATED TO VERSATILE COMMUNICATION

For the experiments of Chapter 2, we only needed a communication scheme to receive telemetry from the on-board computer with an ESP-09 Wi-Fi chip. However, when

designing algorithms for a swarm of pocket drones, the inter-drone communication needed to be dealt with as well. Therefore, in Chapter 3 we implemented on-board relative localization with a BlueGiga Bluetooth module, which worked both as the signal strength indicator as the communicator of each-other's state-estimation. Again, the separate Wi-Fi-chip was used for telemetry. Unfortunately, there is a limit to the amount of hardware one can add until the energy threshold is reached, as we experienced during those experiments. To only have a flight time of 2 minutes was a deal-breaker for the final objective of this dissertation. Although the setup used in Chapter 3 was sufficient to show low-level-navigation with multiple pocket drones in a single room, exploring a building was out of the question with such a flight-time.

In Chapter 5, both the inter-drone communication and the telemetry were performed by the same module. Moreover, the connection with the main computer (telemetry) has also changed its functionality. It now governs the navigational drive of the SGBA algorithm, as it used the signal strength of the telemetry as an indication of progression. This combination of functionalities and the absence of the 4-gram vision system saved a significant amount of power that drove the total flight-time of the Crazyflie up to 8 minutes, which was a more acceptable time-window for indoor navigation. However, there was a catch: instead of 10 Hz telemetry and 5 Hz inter-communication in Chapter 3, it dropped to 2 Hz for both in Chapter 5. Also, in comparison to Chapter 3, the drones did not know where the other drones were coming from: only that they were close. The avoidance in Chapter 5 therefore needed to be conservative, which limited the lower-priority drone in its own ability to explore.

Fig. 6.2b shows the evolution of the communication modules used for the platforms, from only telemetry with Wi-Fi in Chapter 2, to Wi-Fi + Bluetooth for telemetry + inter-drone communication in Chapter 3, to doing all of the functionalities only on the native NRF512188 chip on the Crazyflie in Chapter 5. This was a conscious decision that was based on the total power supply of the Crazyflie, where the perception was more important than the communication element of the platform. For my final solution in Chapter 5, the Crazyflies did not need fast telemetry and exact localization for avoidance. However, if there is anything on the whole system that I could change if the power supply would let me, then I would have added an Ultra-Wide-Band (UWB) module to the system instead for the inter-drone and home-beacon ranging with a separate Wi-Fi module specifically for sharing data. Unfortunately, on the current platform, UWB would have added an extra 120 mAh load on the 3.7 volt battery and would probably have resulted in a much shorter flight-time. For now, the combined perception and communication solution in Chapter 5 was to optimize the flight-time and robustness based on the limitations the platform had, in order to achieve the final objective.

6.5. OPTIMAL VERSUS SUB-OPTIMAL NAVIGATION

Once the combination of hardware, sensors and processing has converged to a stable platform, one can start thinking about a step higher than low-level-navigation: going somewhere with a purpose. In the introduction we explained in detail on why most conventional navigation strategies do not qualify for the pocket drones, so we will not discuss those again in this section. What we will do is to discuss the implications of the navigational strategy we created, and on how our philosophy can also apply for larger

vehicles than the ones we flew with during our experiments.

In Chapter 5, we developed the SGBA algorithm as the indoor navigation strategy. Bug algorithms do not depend on a map-building process. However, such techniques produce sub-optimal path-planning alternatives. Looking at a flight-time perspective — the common thread in our discussion — it is seemingly good to have more optimal path planning to avoid any detours, which would only be possible in the presence of a known map of the environment. The physical platform used in Chapter 5 could have implemented a SLAM algorithm since it has visual odometry from the flow deck and obstacle detection from the multi-ranger deck. However, such a computation would not have been possible on-board the small STM32F4's on the Crazyflies. SGBA, on the other hand, takes up very little space on the on-board computer. A bigger drone or robot could also use this same principle and easily perform auxiliary tasks in parallel, like in the application experiments in Chapter 5. This can consist of multiple tasks, dependent on the capabilities and hardware of the drone, which can range from video-recording to gas-source localization and victim-search. Any possible map-building can be part of the sub-tasks but does not exist within the same control-loop as the navigation. In other words, it has been decoupled of the navigation strategy and therefore leads to having more room to customize your robot for the tasks and missions required.

We benefited from one of the earlier design choices to deal with the sub-optimality of the SGBA. Switching to the flow deck instead of the forward stereo-camera, enabled the Crazyflies to have better visual odometry. Although the eventual drift was too severe for the total navigation strategy, it enabled the SGBA to detect dead-locks and recognize if a Crazyflie got into a loop. However, these were solutions on a local level, since on the macro-level SGBA still needed the signal strength of a home-beacon in order to navigate. To remove this dependency, it still could have been possible to further correct for the odometry drift by recognizing scenes and landmarks in the environment. In this case, it would have been better if a forward-looking or omnidirectional camera was included on the system. However, for the Crazyflie, the laser-range sensors would have needed to be sacrificed to save energy, which would have a significant effect on the robustness against obstacles. For this addition, a slightly larger quadrotor with stronger motors and a bigger battery would have been necessary.

The sub-optimality was also counteracted by making use of the swarm. Whenever one Crazyflie would meet another, the one with the highest priority would share its intention and the lower one would change his own preferred heading, which enabled the Crazyflies to spread out. In comparison with Chapter 3, in Chapter 5 the drones did not know each-others relative position. This can be a problem when the Crazyflie's communicate their intentions to one another. If their coordinate system has diverged from the original at the start of the experiment due to heading drift, the intention (preferred heading) can be misinterpreted. Chapter 5's assumption was that for the outbound flight, this would not have been as severe to cause significant problems. However, for longer flight-times and if the drones would have also communicated the estimated home-beacons direction to each other on the inbound flight, this would have become an issue. Relative localization will be necessary in that case, and the preferred ranging sensor would have been UWB, like in van der Helm et al. (2019). However, as discussed before, adding an UWB on the current system would have not been feasible.

Designing a navigation strategy for such small platforms means that the high-level-navigation is highly interlinked with all the hardware/processing choices made for the low-level-navigation. The sensors used for stable flight put limitations on the extent of the designed navigation strategy. This resulted in a sub-optimal navigation strategy in terms of flying the shortest path. However, the approach was necessary to get it to work fully on-board the constrained resources of the platform, and its limitations is counteracted by having multiple Crazyflies performing the same task. All these choices were made in order to extend the flight-time as long as possible and to have a robust flying platform, given the available hardware, battery and processing capabilities.

6.6. CONCLUSION

In the introduction of this dissertation, we formulated the following main research-question:

> To what extent can we design a robust and computationally efficient method for multiple pocket drones to explore in an unknown, indoor environment and to return to their initial position?

We were able to show a swarm of pocket drones navigating through an indoor environment in the results of Chapter 5, therefore achieving the final objective of this dissertation. There are still a couple of side-notes to these results. First of all, the RSSI gradient search for homing is still dependent on an external element, namely the home-beacon. Secondly, there are some limitations involved with the hardware. The flow deck on the Crazyflie had difficulties with the dark floors and illumination of the environment. The multi-ranger deck, although it was able to work with the environment we tested in, is not suitable for all types of obstacles. However, despite these limitations, the final results did show case that a group of pocket drones is capable of performing such a complex task as indoor exploration.

Throughout the dissertation I came to experience that achieving the final results was not simply a matter of solving the low-level-navigation problem of state-estimation and drone/obstacle avoidance and then move on to the high-level-navigation problem of exploration, homing and inter-drone coordination without any consequence. Working on on-board solutions on such limited systems means that each decision made in the earlier phase has a direct influence on the next. With platforms as small as pocket drones, one should be aware that any enhancement to the elements perception, communication, behavior and their connections, need to share a limited supply of Energy (Fig. 6.3). Each of my choices had a cost: focusing more on the perception of a single drone for robust flight than on the inter-drone communication, had a major influence for the creation of SGBA. However, for indoor-navigation, the first-most priority is to have a single pocket-drone that you can trust with your eyes closed, since there is going to be more obstacles and walls in an indoor environment than inter-drone-encounters. By making the necessary choices in terms of hardware & processing and understanding their consequences, we were able to show a swarm of pocket drones navigating an indoor environment in the results of Chapter 5, therefore achieving the final objective of this dissertation.

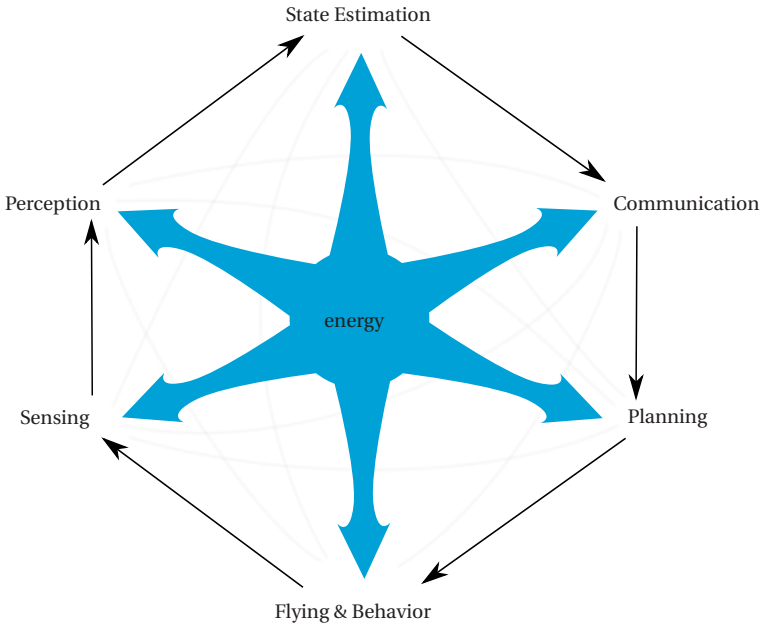


Figure 6.3: Balance between capabilities and energy.

6.7. FUTURE WORK

The work done in this dissertation was a case-study in the possibility of autonomous exploration with a swarm of pocket drones in an indoor environment. This does not represent the final chapter on autonomous swarm navigation on limited systems and there are still matters that need to be developed further. The pocket drones could be optimized even further in both hardware and software, which should make the very same techniques discussed in this dissertation available on even smaller platforms. To enable more functionalities, it would be good to again investigate the addition of a camera. We have abandoned this element for Part II, however it would be very beneficial for the overall system to enable the computation of multiple variables in one image stream and tested it thoroughly, such that e.g. the optical flow of a forward camera would be as suitable for state estimation as for a downward camera. For the inter-drone communication, it would be beneficial to switch to UWB for better ranging properties, so once light-weight – higher capacity – batteries become available. This is the first hardware element of SGBA that should be enhanced.

It will also benefit the homing performance by replacing the home-beacon with UWB. In this dissertation, this minor aid of the high-level navigation was necessary to counteract the drift of the visual odometry, and although no close to a full external positioning system, it would be good to make the swarm completely independent of such elements. There are several alternatives to consider. The Crazyflie will soon have a new extension deck for AI on the market, complete with a single forward-looking camera and capable of running a deep neural network onboard (Palossi et al., 2018), which has al-

ready been shown to perform in obstacle avoidance and corridor following experiments (Palossi et al., 2019). During the exploration phase, the views experienced during that flight can be learned and stored in a manner similar to van Dalen et al. (2018), in case the learning itself can be done online. With an added camera, the pocket drone should be able to recognizes features in its environment, which can enable it to navigate back home more swiftly than with SGBA. This would be required to make the swarm of pocket drones completely autonomous and independent of any system other than themselves.

A

APPENDICES: BUG ALGORITHM LITERATURE SURVEY

A.1. SPECIFIC INFORMATION ABOUT THE EXPERIMENTS WITH THE BUG-ALGORITHMS

A.1.1. PSEUDO-CODE BUG ALGORITHMS

The pseudo-code for Com, Com1, Bug2, Alg1 and Alg2, is listed in Algorithm A.1.1, A.1.2, A.1.3, A.1.4 and A.1.5 respectively. T stands for target and s_{WF} is a variable that determines if the wall-following is right- (s_{WF}=1) or left-sided (s_{WF}=-1). r_{local} stand for local sensor measurements, which can be either contact- or range-sensors. x_global stands for the global position estimate of the Bug Algorithm. $d(H,T)_{prev}$ stands for the previous distance of the hit-point to target and $d(x_{global},T)$ stands for the current distance from BA to target. $list_{hp}$ stand for a list of previously encountered hit-points. v is the control output for the forward velocity of the robot and c_v is the fixed velocity constant. ω is the control output for the heading of the robot in rad/s and c_ω is a fixed rate constant, to control the speed of the robot's turns.

A

Algorithm A.1.1 The pseudo-code for the state-machine of Com.

Init: $state$ = "forward", s_{WF}
Require: $c_v, c_\omega, x_{global} r_{local}, list_{hp}$
function COM
if $state$ is "forward" **then**
$v \leftarrow c_v$
$\omega \leftarrow 0$
if Obstacle is hit **then**
$state \leftarrow$ "wall_following"
else if $state$ is "wall_following" **then**
$[v, \omega] \leftarrow$ `Wall_Following`$(c_v, c_\omega, s_{WF}, r_{local})$ ▷ See A.3.2
if Way towards T is free **then** ▷ Based on r_{local}
$state \leftarrow$ "rotate_to_target"
else if $state$ is "rotate_to_target" **then**
$v \leftarrow 0$
$\omega \leftarrow c_\omega$
if Heading BA same as direction T **then**
$state \leftarrow$ "forward"
return v, ω

Algorithm A.1.2 The pseudo-code for the state-machine of Com1.

Init: $state$ = "forward", $s_{WF} = 1$
Require: c_v, c_ω, r_{local}
function COM
if $state$ is "forward" **then**
$v \leftarrow c_v$
$\omega \leftarrow 0$
if Obstacle is hit **then**
$d(H, T) \leftarrow d(x_{global}, T)$
$state \leftarrow$ "wall_following"
else if $state$ is "wall_following" **then**
$[v, \omega] \leftarrow$ `Wall_Following`$(c_v, c_\omega, s_{WF}, r_{local})$ ▷ See A.3.2
if Way towards T is free and $d(x_{global}, T) < d(H, T)$ **then**
$state \leftarrow$ "rotate_to_target"
else if $state$ is "rotate_to_target" **then**
$v \leftarrow 0$
$\omega \leftarrow c_\omega$
if Heading BA same as direction T **then**
$state \leftarrow$ "forward"
return v, ω

Algorithm A.1.3 The pseudo-code for the state-machine of Bug2.

Init: $state$ = "forward", $s_{WF} = 1$
Require: $M-line$,c_v, c_ω, $x_{global}r_{local}$
function COM
if $state$ is "forward" **then**
$v \leftarrow c_v$
$\omega \leftarrow 0$
if Obstacle is hit **then**
$state \leftarrow$ "wall_following"
else if $state$ is "wall_following" **then**
$[v, \omega] \leftarrow$ `Wall_Following`$(c_v, c_\omega, s_{WF}, r_{local})$ ▷ See A.3.2
if $M-line$ is hit and BA is closer to T **then**
$state \leftarrow$ "rotate_to_target"
else if $state$ is "rotate_to_target" **then**
$v \leftarrow 0$
$\omega \leftarrow c_\omega$
if Heading BA same as direction T **then**
$state \leftarrow$ "forward"
return v,ω

Algorithm A.1.4 The pseudo-code for the state-machine of Alg1.

Init: $state$ = "forward", $s_{WF} = 1$, , $list_{HP}$=[ ]
Require: $M-line$,c_v, c_ω, $x_{global}r_{local}$
function COM
if $state$ is "forward" **then**
$v \leftarrow c_v$
$\omega \leftarrow 0$
if Obstacle is hit **then**
$list_{HP} \leftarrow [list_{HP}, x_{global}]$
$state \leftarrow$ "wall_following"
else if $state$ is "wall_following" **then**
$[v, \omega] \leftarrow$ `Wall_Following`$(c_v, c_\omega, s_{WF}, r_{local})$ ▷ See A.3.2
if x_{global} is in $list_{HP}$ **then**
$state$ is "change_local_direction"
if $M-line$ is hit and BA is closer to T **then**
$state \leftarrow$ "rotate_to_target"
else if $state$ is "rotate_to_target" **then**
$v \leftarrow 0$
$\omega \leftarrow c_\omega$
if Heading BA same as direction T **then**
$state \leftarrow$ "forward"
else if $state$ is "change_local_direction" **then**
$v \leftarrow 0$
$\omega \leftarrow c_\omega$
$s_{WF} = -1$
if BA has rotated 18^o **then**
$state \leftarrow$ "wall_following"
return v,ω

A

Algorithm A.1.5 The pseudo-code for the state-machine of Alg2.

Init: $state$ = "forward", $s_{WF} = 1$, $list_{HP}$=[ ]
Require: c_v, c_ω, r_{local}
  function COM
    if $state$ is "forward" **then**
      $v \leftarrow c_v$
      $\omega \leftarrow 0$
      if Obstacle is hit **then**
        $s_{WF} = 1$
        $d(H,T) \leftarrow d(x_{global}, T) for$
        $list_{HP} \leftarrow [list_{HP}, x_{global}]$
        $state \leftarrow$ "wall_following"
    else if $state$ is "wall_following" **then**
      $[v, \omega] \leftarrow$ `Wall_Following`$(c_v, c_\omega, s_{WF}, r_{local})$ ▷ See A.3.2
      if x_{global} is in $list_{HP}$ **then**
        $state$ is "change_local_direction"
      if Way towards T is free and $d(x_{global}, T) < d(H,T)$ **then**
        $state \leftarrow$ "rotate_to_target"
    else if $state$ is "rotate_to_target" **then**
      $v \leftarrow 0$
      $\omega \leftarrow c_\omega$
      if Heading BA same as direction T **then**
        $state \leftarrow$ "forward"
    else if $state$ is "change_local_direction" **then**
      $v \leftarrow 0$
      $\omega \leftarrow c_\omega$
      $s_{WF} = -1$
      if BA has rotated 18^o **then**
        $state \leftarrow$ "wall_following"
    return v,ω

A.2. PROCEDURAL ENVIRONMENT GENERATOR

The procedural environment generator specifics is shown in Fig. A.1. First, in a coarse grid world (a), two entities are initialized on the exact position of the start and target position to be in the eventual task. They will perform a simple 4-connected path generation, where they will have a certain chance of going straight (p_{str}). The chance of either going left or right is equal to $1-p_{str}$. Each agent will leave a corridor trace, as can be seen in (b), until, in (c), the amount of corridors hit a density threshold ($t_{cor} = 0.4$), which is the number of grid-cells occupied with a corridor divided by the total number of existing grid cells in the environment. A connectivity check is performed, to check if the initial position of the robots are connected by these corridors, which will re-initiate the process in case it fails. This is to ensure that the BA is always able to reach its final destination. Next, walls will be added to these corridors (d). The remaining areas will act as rooms and are divided when they are too large in (e). Finally, in (f), random openings are added along the border of the corridors to create passages these areas.

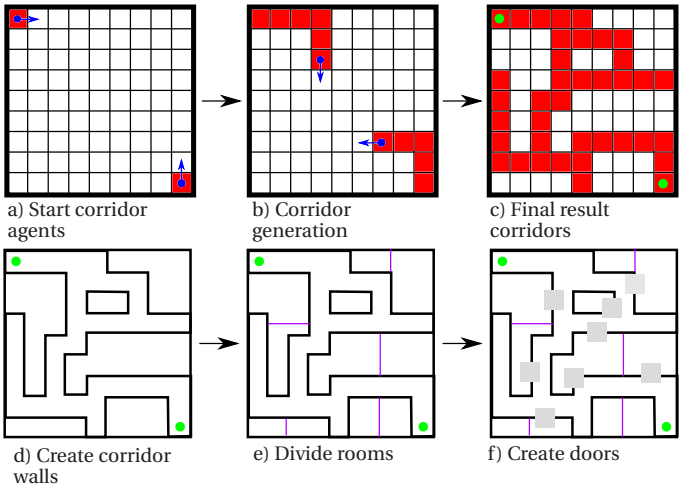


Figure A.1: The steps of the procedural generated environment method will be explained here. The corridor-generating random agents (blue circles) start in (a) at the same positions as the start and target locations of the experiment. These will move forward in (b), while occasionally turning left and right, while leaving a corridor trace (red blocks). Once it reaches the corridor-density threshold in (c), the corridors-cells are tested for interconnectivity, such that the target position can be reached from the starting position (green circles). The corridor walls are created in (d) and then, in (e), remaining non-corridor spaces are then divided into rooms (purple stripes) and random door-openings (gray blocks) are created along the border of the corridors in (f).

A.3. WALL FOLLOWING

A.3.1. CALCULATION REAL DISTANCE FROM WALL

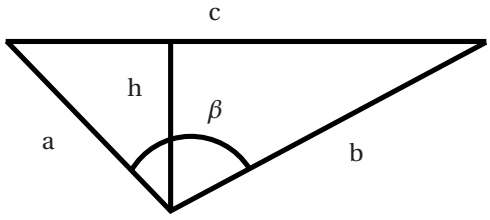


Figure A.2: Visualization of the triangle configuration for the derivation.

In Fig. A.2, the configurations of the solved triangle is solved, where we want to calculate h (height triangle) with the triangle sides of a and b and the angle β. c is the triangle side that will be unknown, so a formula will be derived that will only use a, b and β.

The geometrical equations used to achieve the ranges are the triangle area formula:

$$A = \frac{c \cdot h}{2} \tag{A.1}$$

, the SAS triangle rule:

$$A = \frac{a \cdot b \cdot \sin\beta}{2} \tag{A.2}$$

, the cosine-rule:

$$c = \sqrt{a^2 + b^2 - 2 \cdot a \cdot b \cos\beta} \tag{A.3}$$

with A is the area of the triangle.

Substitute A in Eq. A.2 for the right side of Eq. A.1, and solve for h:

$$h = \frac{a \cdot b \cdot \sin\beta}{c} \tag{A.4}$$

Now substitute c in Eq. A.4, for the right side of Eq. A.3, which results in the following equation:

$$h = \frac{a \cdot b \sin\beta}{\sqrt{a^2 + b^2 - 2 \cdot a \cdot b \cos\beta}} \tag{A.5}$$

A.3.2. PSEUDO CODE WALL FOLLOWING

The procedure of the wall-following behavior is listed in this appendix in Algorithm A.3.1 and A.3.2. Ts_{WF} is a variable that determines if the wall-following is right- (s_{WF}=1) or left-sided (s_{WF}=-1), $d(x, O_\perp)$ is the current distance to the robot calculated perpendicular from the wall and d_{ref} is the preferred distance from the wall in meters and t_d is the threshold to determine if the robot near d_{ref}. r_s and r_f are the side and range sensor's measurement in meters and β is the angle between them. If s_{WF}=1, then r_s is the right range sensor and if s_{WF}=-1, then r_s is the left range sensor. v is the control output for the forward velocity of the robot and c_v is the fixed velocity constant. ω is the control output for the heading of the robot in rad/s and c_ω is a fixed rate constant, to control the speed of the robot's turns.

Algorithm A.3.1 The procedure of the wall-following behavior.

```
   Init: state = "rotate_to_align_wall"
Require: s_WF, d(x,O⊥), d_ref, r_s, r_f, c_v, c_ω, β
   β = 60 deg
   function WALL_FOLLOWING
      if state is "rotate_to_align_wall" then
         v ← 0
         ω ← −1 · s_WF · c_ω                          ▷ Turn away from the wall
         if r_s ≈ r_f · cos(β) then
            state ← "wall_following_and_aligning"
         if r_f = OR then
            state ← "rotate_around_corner"
      else if s_WF is "wall_following_and_aligning" then
         v ← c_v
         ω ← Wall_Following_and_Aligning()            ▷ See A.3.2
         if d(x,O_min) < d_ref then
            state ← "rotate_to_align_wall"
         if r_f is OR then
            state ← "rotate_around_corner"
      else if state is "rotate_around_corner" then
         v ← c_v
         ω ← s_WF · v / d_ref                         ▷ Wide turn, radius = d_ref
         if r_s ≈ r_f · cos(β) then
            state ← "wall_following_and_aligning"
         if d(x,O_min) < d_ref then
            state ← "rotate_to_align_wall"
       return ω
```

Algorithm A.3.2 The procedure of keeping the heading of the FootBot aligned with the wall during wall following.

```
Require: s_WF, d(x,O⊥), d_ref, r_s, r_f, c_w, β
   function WALL_FOLLOWING_AND_ALIGNING
      if |d_ref − d(x,O⊥)| > −t_d ) then             ▷ If too far from d_ref
         if d_ref − d(x,O⊥) > t_d then               ▷ If too far from wall
            ω = s_WF · c_ω                           ▷ Turn towards the wall
         else                                        ▷ If too close to wall
            ω = −s_WF · c_ω                          ▷ Turn from the wall
      else if |d_ref − d(x,O⊥)| < t_d then           ▷ If close to d_ref
         if r_s > r_f · cos β then                   ▷ Fine tune alignment
            ω = s_WF · c_ω                           ▷ Turn towards the wall
         else
            ω = −s_WF · c_ω                          ▷ Turn from the wall
      else                                           ▷ Do not adjust the turn
         ω = 0
       return ω
```

A.4. STATISTICAL TESTS

A.4.1. BOOTSTRAPPING BUG ALGORITHMS

In Fig. 4.11 for perfect localization ($\sigma = 0$), the resulting performance values per bug algorithm was shown. Here, both the success rate and the trajectory length are subjected

A

to a bootstrapping test, to evaluate whether the bug algorithms belong to the same distribution (null-hypothesis). Table A.1 contains the bootstrapping tests from the data presented in Fig. 4.11(a) and Table A.2 for Fig. 4.11(b) for σ=0.

	Com	Com1	Bug2	Alg1	Alg2
Com	1	1	0	0	1
Com1	1	1	0	0	1
Bug2	0	0	1	1	0
Alg1	0	0	1	1	0
Alg2	1	1	0	0	1

Table A.1: Bootstrapping results on the trajectory length of the evaluated bug algorithms with a sample size 10000. The value "1" means that the null-hypothesis (the evaluated data comes from the same distribution) holds, while "0" means it is rejected.

	Com	Com1	Bug2	Alg1	Alg2
Com	1	0	0	0	0
Com1	0	1	1	1	0
Bug2	0	1	1	1	0
Alg1	0	1	1	1	0
Alg2	0	0	0	0	1

Table A.2: Bootstrapping results on the success rate of the evaluated bug algorithms with a sample size 10000. The value "1" means that the null-hypothesis (the evaluated data comes from the same distribution) holds, while "0" means it is rejected.

A.4.2. CORRELATION ANALYSIS ODOMETRY NOISE

In order to evaluate whether an relationship exists between the increasing odometry noise and the degeneration of the performances of the bug algorithms, the data presented in Fig. 4.11 are subjected to regression analysis. Table A.4 contains the logistic regression analysis with a R2 value, from the trajectory length data presented in Fig. 4.11(a) and Table A.3 contains the logistic regression analysis with a pseudo-R2 value, from the success rate data presented in Fig. 4.11(b).

	Com	Com1	Bug2	Alg1	Alg2
Slope	8.081	13.642	13.561	11.857	12.523
Intercept	2.752	2.724	3.152	3.522	2.654
R2	0.076	0.189	0.217	0.173	0.161

Table A.3: Linear regression evaluation of the trajectory lengths against the measurement noise, including the intercept, slope and R2 value per bug algorithm.

	Com	Com1	Bug2	Alg1	Alg2
Slope	-1.240	-3.100	-3.860	-3.480	-3.110
Intercept	0.587	0.800	0.779	0.706	0.847
R2	0.035	0.189	0.343	0.323	0.198

Table A.4: Logistic regression evaluation of the success rate against the measurement noise, including the intercept, slope and (psuedo) R2 value per bug algorithm.

A.4.3. CORRELATION ANALYSIS RECOGNITION FAILURES

In order to evaluate whether an relationship exists between the increasing failing recognition rate and the degeneration of the performances of the bug algorithms Alg1 and Alg2, the data presented in Fig. 4.13 are subjected to regression analysis. Table A.6 contains the logistic regression analysis with a R2 value, from the trajectory length data presented in Fig. 4.13(a) and Table A.5 contains the logistic regression analysis with a pseudo-R2 value, from the success rate data presented in Fig. 4.13(b). Table A.8 contains the logistic regression analysis with a R2 value, from the trajectory length data presented in Fig. 4.13(c) and Table A.7 contains the logistic regression analysis with a pseudo-R2 value, from the success rate data presented in Fig. 4.13(d).

	Alg1	Alg2
Slope	0.5472	0.1722
Intercept	2.4302	1.8843
R2	0.0112	0.0020

Table A.5: Linear regression evaluation of the trajectory lengths against the False Positive recognition rate, including the intercept, slope and R2 value per bug algorithm.

	Alg1	Alg2
Slope	-0.1443	-0.0014
Intercept	0.9105	0.9418
R2	0.4652	0.7773

Table A.6: Logistic regression evaluation of the success rate against the False Positive recognition rate, including the intercept, slope and (psuedo) R2 value per bug algorithm.

	Alg1	Alg2
Slope	0.1873	1.0584
Intercept	3.2478	2.2733
R2	0.0000	0.0006

Table A.7: Linear regression evaluation of the trajectory lengths against the False Negative recognition rate, including the intercept, slope and R2 value per bug algorithm.

	Alg1	Alg2
Slope	0.0577	0.1010
Intercept	0.8018	0.9192
R2	0.3668	0.7171

Table A.8: Logistic regression evaluation of the success rate against the False Negative recognition rate, including the intercept, slope and (psuedo) R2 value per bug algorithm.

A.4.4. CORRELATION ANALYSIS DISTANCE SENSOR NOISE

In order to evaluate whether an relationship exists between the increasing distance measurement noise and the degeneration of the performances of the bug algorithms Alg1 and Alg2, the data presented in Fig. 4.14 are subjected to regression analysis. Table A.10

contains the logistic regression analysis with a R2 value, from the trajectory length data presented in Fig. 4.14(a) and Table A.9 contains the logistic regression analysis with a pseudo-R2 value, from the success rate data presented in Fig. 4.14(b).

	Com1	Alg2
Slope	0.0501	-0.0075
Intercept	2.5783	2.4204
R2	0.0019	0.0001

Table A.9: Linear regression evaluation of the trajectory lengths against the distance measurement noise, including the intercept, slope and R2 value per bug algorithm.

	Com1	Alg2
Slope	-0.0557	-0.0225
Intercept	0.8682	0.9283
R2	0.2412	0.5583

Table A.10: Logistic regression evaluation of the success rate against the distance measurement noise, including the intercept, slope and (psuedo) R2 value per bug algorithm.

B

APPENDICES: SWARM GRADIENT BUG ALGORITHM

B.1. CODE REPOSITORIES FOR SIMULATION AND REAL-WORLD TESTING

All repositories can be found by following the instructions of the future publication of:

K. N. McGuire, C. De Wagter K. Tuyls, H. Kappen & G.C.H.E. de Croon, *Minimal navigation solution for a swarm of tiny flying robots to explore an unknown environment*, Science Robotics (2019)[In press].

B.2. REAL-WORLD TEST ENVIRONMENT

The environment used for the real-world testing, was the 11th floor of the high-rise building of the Faculty of Aerospace Engineering, TU Delft (Kluyverweg 1, Delft, the Netherlands). The floorplan and several photographs of the surroundings can be found in Fig. B.1. Every second room in the building (grayed out areas in Fig. B.1E) was closed during testing, in order to be able to track the robot trajectory and register the rooms it visits from the hall way camera. Moreover, the limited flight-time due to the battery did not allow exploration of all the available rooms. One room (# 10) was selected to be a monitor room, where the responsible person was able to initiate the different test configurations. The home beacon location was shown in Fig. B.1H, where the flying robots took off and tried to return to.

B.3. RSSI MEASUREMENTS

To illustrate the Received Signal Strength Intensity (RSSI) measurements within the rooms of the environment described in Text B.2, we performed 3 nearly identical flights, where one Crazyflie performed a wall-following behavior in room numbers 20, 21 and 22. In the meantime, it was communicating with both the home beacon and another

Figure B.1: Overview of the real-world environment. This figure shows the pictures of the environment used for the real-world testing. E) has the floor plan of the 11th floor of the Aerospace building of the TU Delft, with A)-D) and F)-I), shows photographs of certain locations. All photographs were taken after the testing took place, for the exception of H), which shows the starting position of one of the 6-drone test configuration.

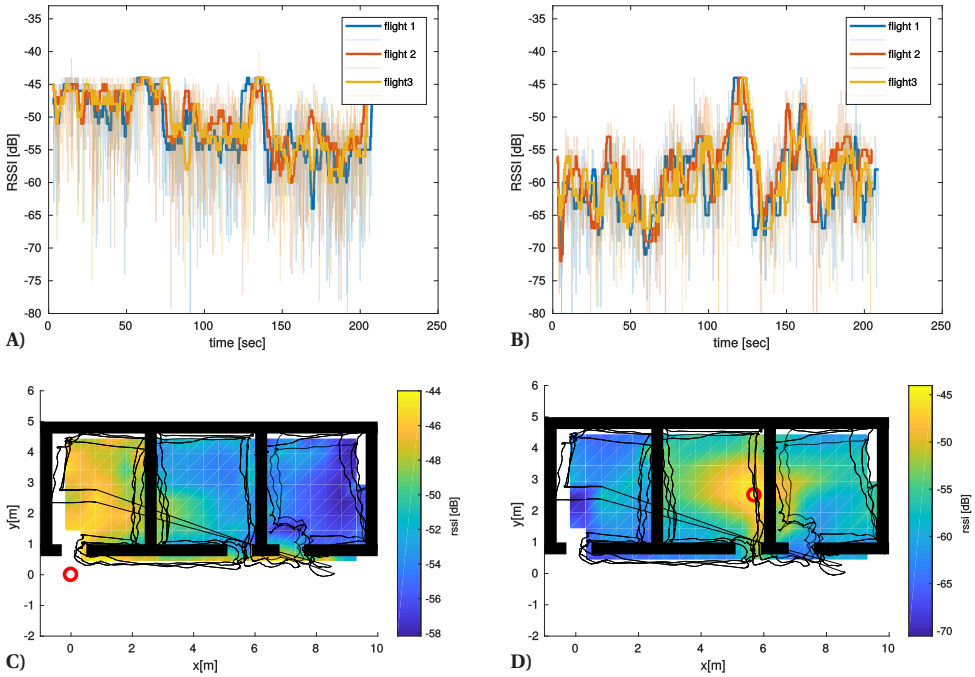


Figure B.2: RSSI measurements. The RSSI measurements of three flights with the Crazyflie performing wall following while communicating with A) the home beacon and B) another Crazyflie. The surface plots of the averaged RSSI measurements over 3 flights with the corrected odometry of the communication of the Crazyflie with C) the home beacon and D) another Crazyflie. The red circle in C) is the location of the home-beacon and the red circle in D) is the other Crazyflie.

Crazyflie that lay on the floor of room 21 (the communication rate was higher than during the final tests). Fig. B.2A shows the raw RSSI measurements with the home-beacon where 3 distinct plateaus can be distinguished which indicate the rooms the Crazyflie has entered, and the difference can also be seen in the surface plot of the average RSSI in Fig. B.2C. For the RSSI between the two Crazyflies (Fig. B.2B), this is less significant. Nevertheless, a high RSSI peak can be seen at 130 seconds, which corresponds with passing near the ground-bound Crazyflie (Fig. B.2D). Interestingly enough, another peak can also be measured at the other side of the wall at 160 seconds, which is not as high but possibly could trigger an avoidance maneuver in SGBA if the threshold is not chosen carefully.

B.4. FROM ODOMETRY TO TRAJECTORY

For the real-world experiments, we connected each drone with a Crazyradio PA to retrieve on-board estimates and data. We used it to know the onboard odometry, SGBA's state etc. and provide the data for the position plots like Fig. B.3. As to be expected, due to drift, the position odometry did not correspond to the actual flight trajectory seen on the external cameras (Fig. B.3A and B). We therefore needed to adjust the odometry-based

position estimate to fit within the floor plan. We used the following strategy:

1. Transcripts: We made transcripts of the flights by using the hallway camera. For each drone, the hallway trajectory was approximated. This included information about which rooms they entered (see Fig. B.4A and B). To be able to see which rooms the robots entered, especially the flights at the far left, each second room was closed. Beside the hallway, only room 03 and 06 were equipped with a fixed camera, so we would need to rely on the odometry data to analyze the robot's behaviors inside most rooms.

2. Global correction: From the odometry, we first made a global adjustment. After a light Gaussian filter, we translated the starting position, scaled in the x and y axis and adjusted the heading drift, by rotating the trajectory by a heading increment at every time-step. This all to make an initial correction in order to get the trajectory a S3little closer that what we observed in the flight transcripts. These parameters (translation, rotation and its derivative, and scaling) unfortunately are different for every drone. See Fig. B.5A for the effects of this correction.

3. Local correction: Although the global correction would be enough for some (short) flights, the degree of drift is unfortunately not exactly the same throughout the duration of the flight. Therefore, a local correction needs to be made (Fig. B.5A), where we need to go through every time-step and see if the trajectory needs additional adjusting to fit the environment in a realistic manner. We developed a GUI in Matlab 2017 for this purpose (Fig. B.5B).

Going through these 3 steps enabled us to build the plots we show in Fig.5.6B and C in the main body of the chapter. Please remark that the above procedure, which takes 30 minutes, is only necessary for producing the trajectory plots for the scientific manuscript. The procedure plays no role whatsoever in the autonomous navigation and is also not necessary for a future real-world application of SGBA.

B.5. ANALYSIS AND STATISTICS

This section presents the data and analysis of both the simulation and real-world experiments.

All databases can be found by following the instructions in the future publication of:

K. N. McGuire, C. De Wagter K. Tuyls, H. Kappen & G.C.H.E. de Croon, *Minimal navigation solution for a swarm of tiny flying robots to explore an unknown environment*, Science Robotics (2019) [In press].

B.5.1. SIMULATION RESULTS ANALYSIS

Table B.1 shows the statistics of the simulation results. For each test configuration, 100 environments were generated. We calculated the coverage by means of a cell grid, as illustrated in Fig. B.6. The statistical analysis results can be found in Table B.1. In the 2nd to last row, the R2 and P-value of the relationship between the number of drones and the variables "coverage total", "coverage returned", "coverage individual" and "return rate", as calculated by the Matlab function Fitml for a quadratic approximation (P-value based

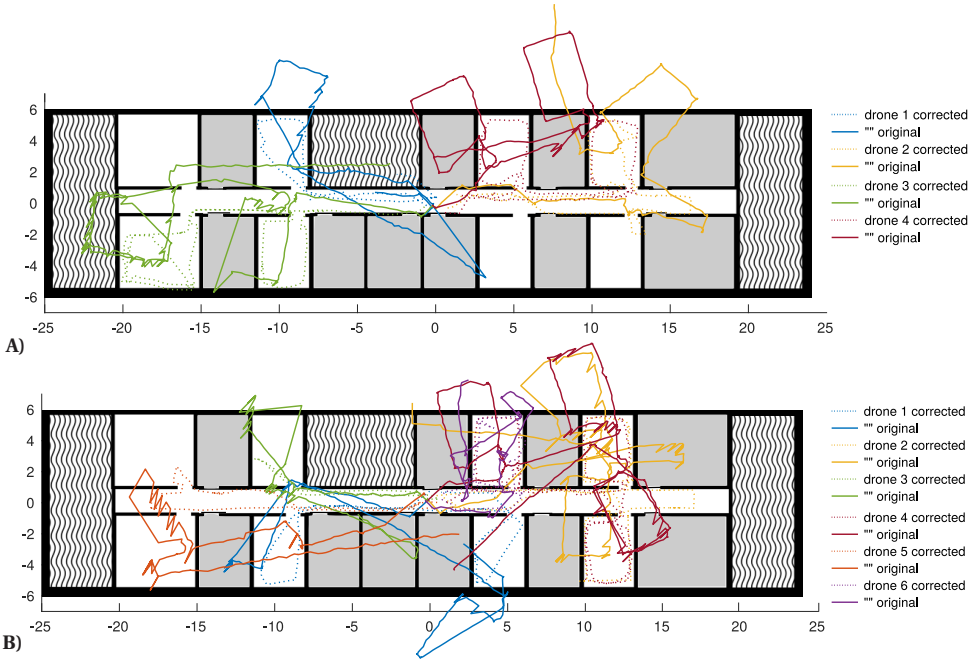


Figure B.3: Odometry versus trajectory. Here the real on-board calculated odometry data is shown, versus the corrected version that is used in the main body of this chapter. A) shows the original odometry of Fig. 5.6B and B) shows the original odometry of Fig. 5.6C.

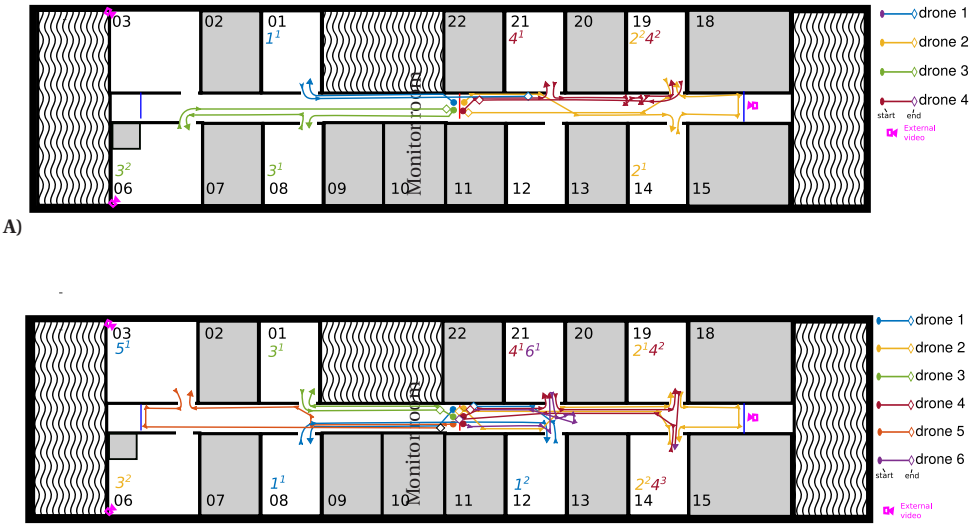


Figure B.4: Transcripts Hallway Video. This figure presents the visual transcripts of A) the 4-drone configuration (flight #2 of 5) and B) the 6-drone configuration (flight #3 of 5). These transcripts were drawn from the hallway video recorder on the far right, so only the trajectory in the hallway and room entering has been noted. Each room has its number (in black) and also contain the ID of the drone that entered that room (in the color of its trajectory). The superscript of that ID number stands for the sequence of rooms visited by that very same drone. The gray rooms had closed doors and were not accessible by the drones and functioned as walls.

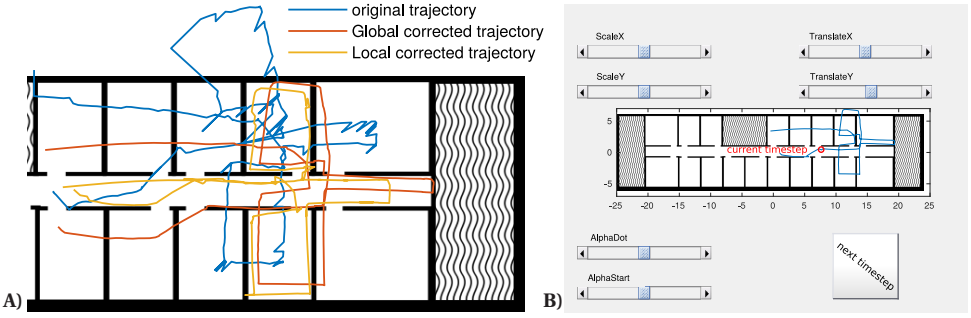


Figure B.5: Odometry Correction. This figure shows A) the evolution of the original odometry measurements (in blue), to undergo a global correction (in red) in terms of translation, rotation and a rotation derivative and scaling in both the x- and y-direction. These correspond in integration error in heading from the gyro bias, scaling error of the forward optical flow and integration errors in optical flow. For the final result, a time-wise local correction is applied (in yellow), with help of the developed GUI in B) where those same parameters are adjusted during each time step.

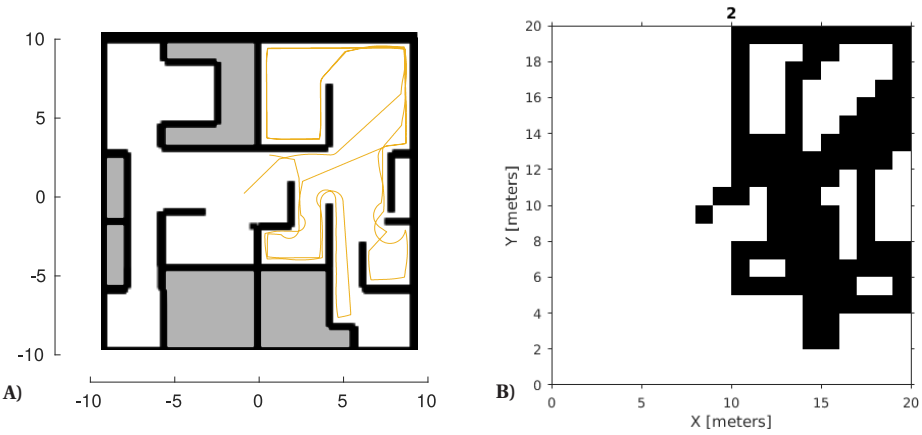


Figure B.6: Coverage calculation simulation. In A) the trajectory (in yellow) in the procedurally simulated environment is shown, with the inaccessible areas marked in grey. The trajectory is turned into an occupancy map in B), which will be used to calculate the coverage, taking into account which areas are not accessible for the agent.

Configuration	Coverage Total		Coverage Return		Coverage Indiv.		Return Rate	
	Avg	Std	Avg	Std	Avg	Std	Avg	Std
2	0.5195	0.1404	0.3594	0.3594	0.3000	0.0987	0.7250	0.3208
4	0.7546	0.1080	0.5601	0.5601	0.3094	0.0775	0.7150	0.2488
6	0.8164	0.1026	0.6838	0.6838	0.3142	0.0681	0.7350	0.2236
8	0.8620	0.0875	0.7378	0.7378	0.3210	0.0647	0.7113	0.2252
10	0.8996	0.0624	0.8060	0.8060	0.3134	0.0555	0.7490	0.1872
Intercept	0.30449		0.14087		0.28396		0.73965	
x1	0.13152		0.12524		0.0088293		-0.0088768	
x12	-0.00734		-0.00597		-0.00057529		0.00092411	
R2	0.6095		0.4472		0.00806		0.0013	
P-value	3.28e-102		1.05e-64		0.134		0.724	

Table B.1: Statistics of the simulation tests. This table presents the average and standard deviation of the total coverage, coverage of the returned agents, coverage of the individual agents and the return rate per configuration of 2, 4, 6, 8, 10 drones from the simulation experiments. It also includes the intercept and coefficients of the quadratic model fit, along with the R2 value and F-test based p-value which indicates the quality and significance of the model. These values were calculated with the Matlab function fitml for a quadradic function. The latter value is indicated in bold if the relationship is considered significant, based on the 5-percentile rule.

on a F-test). The approximation is done on the results of all the 100 environments per drone-number configuration. This indicated a significant relationship between “coverage total”, “coverage returned” and the increasing number of drones, but not for the “coverage individual” and “return rate”.

B.5.2. REAL-WORLD RESULTS ANALYSIS

This section explains the statistical data of the real-world experiments. Table B.2, B.3 and B.4 show the end status per drone after each of the 5 flights, for the 2-, 4- and 6-drone configurations respectively. There are 5 statuses to consider:

- "Made it”: This status indicates that the drone was able to return safely to the home beacon for its inbound flight.

- "Low Battery": This status indicates that the drone was not able to return to the home beacon and landed remotely because of a low battery.
- "Crash wall": This is a failure case that the wall-following was not sufficient to handle a particular obstacle and the drone's rotors came in contact and crashed because of it. (If the reason cannot be determined, this is the standard failure mode).
- "Crash drone": This is a failure case that the drones could not sense each-other in time and that one or two drones crashed because they collided against each-other.
- "Hardware": This failure case is due to a hardware error that caused the drone to not detect obstacles in time or failed to fly. E.g. due to the vibrations, the multi-ranger deck could get lose during flight, which leaves the drone blind to its surroundings, or the loss of a propeller.

Additionally, we also present the data of the visited rooms per drone per flight in Table B.5, B.6 and B.7 for the configuration of 2, 4 and 6 drones. The rooms in the table are shown in sequence and are based on the video footage that we took per flight (available at the database mentioned before). The room numbers are the real office numbers, which are shown in Fig. B.1E. Moreover, the number of unique flights is also listed in brackets after the room sequence, and the rooms visited by the drones that were able to return (end-status of "Made it") are listed in bold.

In Table B.8, we show all the averages for each flight number for each configuration. The coverage total sums all the unique rooms all the drones have visited from Table B.5, B.6 and B.7 and divide them by the total number of existing rooms (8). The same is done for the coverage returned, however only the rooms that the returned robots have explored are included in the calculations. The coverage per agent is also calculated in a similar manner, by taking the average of the rooms listed after the sequences in B.5, B.6 and B.7 and dividing it by the total number of agents.

Finally, the return rate is the number of returned drones divided by the total number of drones flying in the experiment. The average standard deviation per variable and per configuration is listed in Table B.8. In the 2nd to last row, the R2 value and P-value of the relationship between the number of drones and the variables "coverage total", "coverage returned", "coverage individual" and "return rate", as calculated by the Matlab function Fitml for a quadratic approximation (P-value based on a F-test). Immediately noticeable is that there is a significant relationship between the number of agents versus total coverage but not with the other performance measures. We also present a collision count for each of the real-world experiments in Table B.9. Since only one collision occurred out of the 54 encounters, the collision avoidance success percentage is 98 %.

Flight number	Returned	Drone 1	Drone 2
1	2\2	Made it	Made it
2	2\2	Made it	Made it
3	2\2	Made it	Made it
4	2\2	Made it	Made it
5	2\2	Made it	Crash wall

Table B.2: End status of the real-world tests with 2 drones. The return rate and end-status of the 5 flights of the 2-drone experiment configuration in the real-world environment.

Flight number	Returned	Drone 1	Drone 2	Drone 3	Drone 4
1	4/4	Made it	Made it	Made it	Made it
2	4/4	Made it	Made it	Made it	Made it
3	2/4	Made it	Low battery	Made it	Hardware
4	3/4	Made it	Crash wall	Made it	Made it
5	3/4	Made it	Made it	Made it	Crash wall

Table B.3: End status of the real-world tests with 4 drones. The return rate and end-status of the 5 flights of the 4-drone experiment configuration in the real-world environment.

Flight number	Returned	Drone 1	Drone 2	Drone 3	Drone 4	Drone 5	Drone 6
1	4/6	Made it	Made it	Made it	Made it	Low b.	Crash w.
2	4/6	Made it	Made it	Low b.	Made it	Made it	Hardware
3	6/6	Made it	Made it	Made it	Made it	Made it	Made it
4	3/6	Made it	Crash w.	Hardware	Crash D.	Made it	Made it
5	5/6	Made it	Made it	Crash w.	Made it	Made it	Made it

Table B.4: End status of the real-world tests with 6 drones. The return rate and end-status of the 5 flights of the 6-drone experiment configuration in the real-world environment.

Flight number	Drone 1	Drone 2
1	08→12 (2)	12→21→12→14→19 (4)
2	14 (1)	08→12→14 (3)
3	08→12→12 (2)	03 (1)
4	21→14 (2)	12→14→14 (2)
5	08→12 (2)	19 (1)

Table B.5: Coverage of the real-world tests with 2 drones. This table presents the room numbers visited in the real-world environment per flight and per drone of the 2-drone experiment configuration, with the number of unique rooms in brackets. The visited rooms of the flying robots that have returned, are shown in bold.

Flight number	Drone 1	Drone 2	Drone 3	Drone 4
1	01→19 (2)	21→14 (2)	08→08→06 (2)	14→14 (2)
2	01 (1)	14→19 (2)	08→06 (2)	21→19 (2)
3	01→21 (2)	12→14→19→19 (3)	01 (1)	- (0)
4	01→03 (2)	21 (1)	08→08 (1)	12→21→19 (3)
5	06→06 (1)	21→19 (2)	08→03 (2)	- (0)

Table B.6: Coverage of the real-world tests with 4 drones. This table presents the room numbers visited in the real-world environment per flight and per drone of the 4-drone experiment configuration, with the number of unique rooms in brackets. The visited rooms of the flying robots that have returned, are shown in bold.

Flight number	Drone 1	Drone 2	Drone 3	Drone 4	Drone 5	Drone 6
1	08 (1)	21→19 (2)	- (0)	14 (1)	08→01→08→06 (3)	- (0)
2	08→12 (2)	21 (1)	01→01→03 →03→03 →03→01 (2)	14 (1)	08→12 (2)	- (0)
3	08→12 (2)	19→14 (2)	01 (1)	21→19→14 (3)	03 (1)	21 (1)
4	08→12 (2)	12 (1)	01 (1)	19 (1)	03→08→03 (2)	19→14→19→14 (2)
5	08→12 (2)	12 (1)	01→21 (2)	21→19→14→14 (3)	03→06→06→06 (2)	21 (1)

Table B.7: Coverage of the real-world tests with 6 drones. This table presents the room numbers visited in the real-world environment per flight and per drone of the 6-drone experiment configuration, with the number of unique rooms in brackets. The visited rooms of the flying robots that have returned, are shown in bold.

Flight #	Total Coverage			Coverage Returned			Indiv. Coverage			Return rate		
p. Conf.	2	4	6	2	4	6	2	4	6	2	4	6
1	0.625	0.750	0.750	0.625	0.750	0.500	0.375	0.219	0.146	1.000	1.000	0.667
2	0.375	0.750	0.750	0.375	0.750	0.500	0.250	0.219	0.167	1.000	1.000	0.667
3	0.375	0.625	0.875	0.375	0.250	0.875	0.188	0.188	0.208	1.000	0.500	1.000
4	0.375	0.750	0.750	0.375	0.750	0.625	0.25	0.219	0.188	1.000	0.750	0.500
5	0.375	0.625	1.000	0.250	0.625	0.875	0.188	0.156	0.229	0.500	0.750	0.833
Average	0.425	0.7	0.825	0.4	0.625	0.675	0.250	0.200	0.188	0.900	0.800	0.733
Std	0.112	0.069	0.112	0.137	0.217	0.187	0.077	0.036	0.024	0.224	0.209	0.190
Interc.	0.0000			0.000			0.33125			10.333		
x1	0.2500			0.24375			-0.05000			-0.0750		
x12	-0.01875			-0.021875			0.0046875			0.0041667		
R2	0.7791			0.3456			0.1943			0.1193		
P-value	0.000116			0.0785			0.274			0.467		

Table B.8: Statistics of the real-world tests. This table shows the coverage rates (total and of the returned robots) of each flight and each drone configuration (2, 4, 6 drones), including the return rate. The average and standard deviation are given, with the R2 value and P-value (bold-type indicates significance) from the quadratic model fit.

	Collisions / Avoid attempts		
Trail nr.	2 CFs	4 CFs	6 CFs
1	0/1	0 / 5	0 / 9
2	0/2	0 / 3	0 / 3
3	0/1	0 / 2	0 / 5
4	0/3	0 / 4	1 / 8
5	0/0	0 / 3	0 / 5
Total per config	0/6	0/17	1/30
Total	1/54		

Table B.9: Real world collisions. Counted intra drone avoidance attempts and collisions of the Swarm Gradient Bug Algorithm real-world experiments.

B.6. SGBA IMPLEMENTATION DETAILS SIMULATION VERSUS REAL-WORLD

The swarm gradient bug algorithm (SGBA), as explained in the Materials and Methods section, has been implemented on two different platforms for the results of the main body of this chapter. Fig. B.7 shows the strategies used for the different sub-modules of SGBA for the simulated Footbot in ARGoS, and the real-world Crazyflie. These sub-modules originate from the state-machine presented in Fig. 5.4 of the main body of the chapter, which are: 1) the initial direction choice, 2) wall-following strategy, 3) odometry measurement for loop detection, 4) intra-drone avoidance strategy and 5) gradient search towards the departure point.

For both the simulation as the real-world experiments, the initial choices for the preferred direction for the outbound travel are 45o, 135o, -135o and -45o (Fig. B.7A), where the modulus of the robot's ID from 4 determines the preferred direction (ID 1 = 45^o, ID 2 = 135^o, ID 3 = -135^o, ID 4 = -45^o, ID 5 = 45^o, ...). This has been implemented the same for both the Footbot and Crazyflie. Of course, other schemes are possible, such as having smaller differences between the angles or evenly distributing the robots' preferential angles over the 360 degrees (then the angle between two robots is 360^o / number of robots). However, this influences the interaction with the other robots, as they change their preferred direction if they are in proximity of another robot. The current strategy is to flip the sign of the preferred angle, however, if more choices for the initial preferred direction become available, other options such as adding 90^o to their preferred direction may be better.

The wall-following strategy is different between the Footbot and the Crazyflie (Fig. B.7B). The Footbot is a non-holonomic robotic platform which can only move forward/backwards and rotate around its z-axis. Moreover, it uses a different sensor configuration, namely two laser range sensors on its sides and a distance detection wedge on its front. It therefore uses the side range sensor and the measurement of the range wedge for to keep its distance from the wall and align its heading with the wall along the way. The Crazyflie has more freedom of motion as it is a non-holonomic platform. This means that it can move backwards/forwards, sideways and rotate around its z-axis, given that it flies at a constant height in our experiments. However, it only uses four laser range sensors for navigation. This calls for a different wall-following strategy than the Footbot, as it only uses one side range sensor to keep its distance from the wall. This means that it can only determine the wall alignment at the beginning when it encounters the wall and rotates. The wall-following details can be found in the code given in Text B.1.

For SGBA, it is necessary for both the simulation and the real-world experiments for the robots to detect if they are moving in an endless loop, which they can determine by means of odometry (Fig. B.7C). The Footbot is a ground-bound wheeled robot, so it uses its wheel-odometry to determine its position. In the simulation we added extra noise to make the odometry less perfect, such that the loop detection will only work for short distances. The Crazyflie is a flying robot and hence makes use of optical flow measurement to detect its own motion. In Material and Methods, we explained that we use the Flowdeck to enable this measurement. The Flowdeck also contains a laser

range sensor, since the optical flow measurement needs to be scaled to output a velocity estimate, which can then be integrated to a position. Optical flow based odometry is known to be more sensitive to drift, due to its reliance on texture, lighting and due to the platform's vibrations during flying.

The intra-drone coordination is also handled differently for the Footbot and the Crazyflie (Fig. B.7D). First let us discuss the determination of the other robot's proximity. In ARGoS, that measurement is the plain distance in meters and in the real-world experiments it is the received signal strength intensity (RSSI) of the intra drone communication. Once this proximity measurement hits a certain threshold, the robot with the lowest priority will perform an avoidance maneuver to make way for the other. In simulation, the Footbot with the lowest priority (highest ID) will stand still and let the higher-priority robot move around it as if the static robot was an ordinary obstacle. This is possible with the driving robots, as their IR sensors can easily detect the other robot. This will not be done in the real-world testing as the effective field of view of the four laser range sensors for detection is much smaller. The laser rangers on the flying Crazyflie robots can easily miss another robot due to the small robot size and the possible differences in height. Here, the low priority Crazyflie will physically move out of the way, while the other passes along the wall. The avoiding robot makes sure that all its four horizontal laser range finders (front, sides and back) are at least a meter away from any obstacle, which is twice the normal wall following distance (Fig. B.7G). It will stay at its safe position until the intra-drone RSSI has lowered to a level that indicates that the other, higher priority drone has passed. The Crazyflies are performing a much more conservative way of avoidance than the Footbots, since intra-robot collisions are more problematic for flying robots than for slow-moving wheeled robots.

The gradient search strategy to the starting position for both the simulation experiments and real-world experiments are implemented in the same way (Fig. B.7E). The main difference is that the RSSI between the Footbots and the home beacon is from a simulated model of noisy RSSI based on actual distance and the bearing between them (details can be found in the "Indoor_environment_generator" repository explained in Text B.1) and the Crazyflie uses actually measured RSSI. The simulated model does not take into account the local minima as observed in Text B.3, which is due to the environment's characteristics. This makes the gradient search more difficult for the Crazyflie than the Footbot.

Even though most sub-modules of SGBA were implemented differently in the simulated Footbot and the real Crazyflie, the concept as explained in Materials and Methods stays the same. As long as there is a way to accommodate or replace the functionalities explained in Fig. B.7A-E, it should be possible to implement SGBA on other robotic platforms than the Footbot and the Crazyflie. All details of the implementation can be found in the code presented in Text B.1.

B.7. SGBA SUBMODULE ANALYSIS

In order to evaluate the different sub-modules of SGBA and the effect they have on the performances of the robots, we have redone several simulation experiments where these are turned off. This has been done on 6 Footbots where we evaluated the different cases in 30 procedurally generated environments.

A)

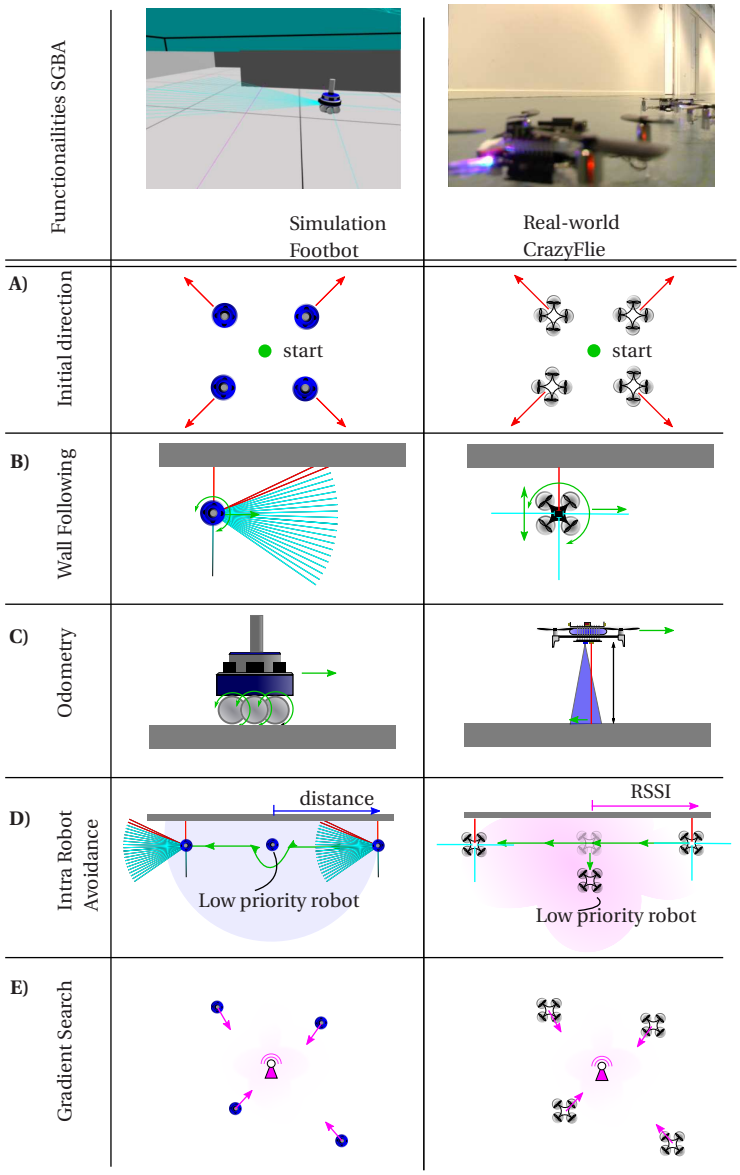


Figure B.7: SGBA simulation vs. real-world. This figure explains the differences of the implementation of the main functionalities of SGBA in the simulated Footbot and the real-world Crazyflie. A) shows how both the Footbot and the Crazyflie has four starting directions to choose from. B) The wall-following has to be handled differently due to difference in motion capabilities and sensor configuration. C) shows that the Footbot has to use its wheel-odometry for loop detection, and the Crazyflie has to use visual odometry for the same functionality. D) show a difference in intra drone avoidance: The low priority Footbot will stand still and let the high-priority robot move around it, but the low priority Crazyflie moves away from the wall to make way for the other. In order to get back to their home position, E) both the Footbots and Crazyflies use the gradient of the signal strength of a wireless beacon.

We first looked at the following scenarios to investigate the effect of some of the swarming mechanisms on the number of collisions:

- Normal': This is the unmodified SGBA from the main body of the chapter.
- No dir change': The same as normal, except that the robots do not change their preferred direction on the outbound travel when encountering other robots.
- 'No avoid': The same as normal, except that the robots will not actively avoid each other when they are in proximity.
- 'No avoid, No dir change': Here there is no intra-robot interaction, so no avoidance and no preferred direction change.
- 'Same init dir': The same as normal, except that here all robots start off going to the same preferred direction.

A significant change in the number of collisions can be seen in Fig. B.8, as it increases by inactivating more sub-modules of SGBA. For the 6 Footbots performances, the collisions will not make a difference as they can still go on continue their journey. However, collisions will have a significant effect for the real-world experiments, as the Crazyflies would not be able to recover after a crash.

We also looked at the loop detection sub-module, by turning it on and off for 1 and 6 robots. The resulting return rate can be found in Fig. B.9. Here it can be seen that for 1 robot the return rate drops when there is no loop detection, but for 6 robots there is no significant drop to be found. Upon detailed inspection, we noticed that intra-robot encounters are responsible of getting stuck robots out of a loop. With this they can cope with the lack of a proper loop detection, which is an interesting feature of the swarming element of SGBA.

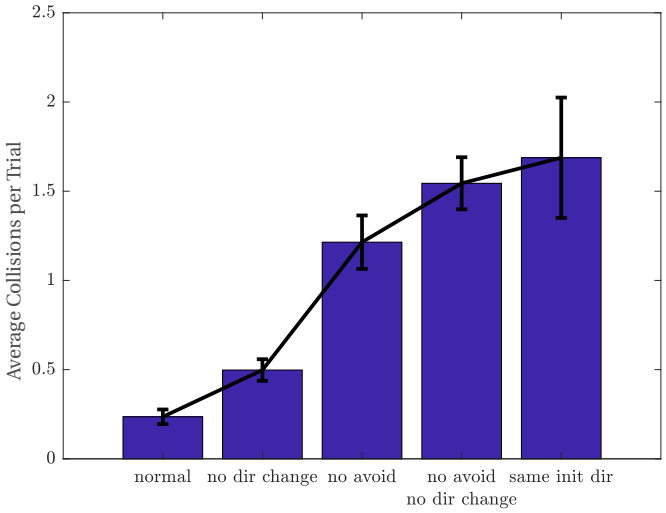


Figure B.8: Simulated collisions SGBA. Average collision per trail with various elements of SGBA turned off for 30 generated environments. The error bars stand for the standard error.

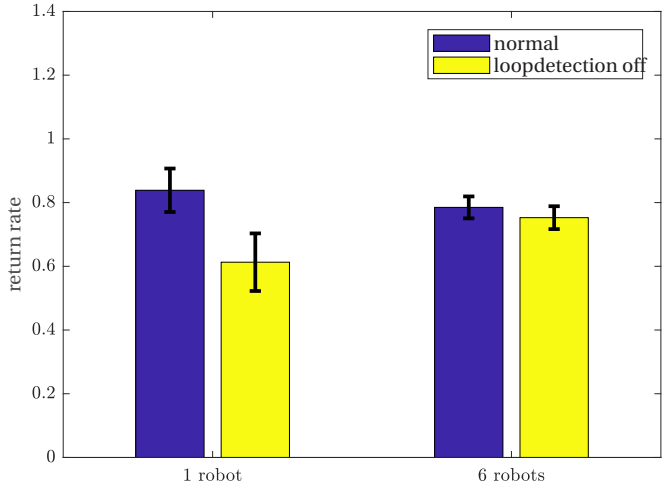


Figure B.9: Loop detection check. Return rate for 1 and 6 robots with the loop detection turned on (blue) and off (yellow). The error bars stand for the standard error.

BIBLIOGRAPHY

REFERENCES

M. Achtelik, M. Achtelik, S. Weiss, and R. Siegwart. Onboard imu and monocular vision based control for mavs in unknown in- and outdoor environments. In *2011 IEEE International Conference on Robotics and Automation*, pages 3056–3063, May 2011. doi: 10.1109/ICRA.2011.5980343.

M. H. Afzal, V. Renaudin, and G. Lachapelle. Magnetic field based heading estimation for pedestrian navigation environments. In *2011 International Conference on Indoor Positioning and Indoor Navigation*, pages 1–10, Sep. 2011. doi: 10.1109/IPIN.2011.6071947.

A. Bachrach, R. He, and N. Roy. Autonomous flight in unknown indoor environments. *International Journal of Micro Air Vehicles*, 1(4):217–228, 2009. doi: 10.1260/175682909790291492.

M. S. Bargh and R. de Groote. Indoor localization based on response rate of bluetooth inquiries. In *Proceedings of the First ACM International Workshop on Mobile Entity Localization and Tracking in GPS-less Environments*, MELT '08, pages 49–54, New York, NY, USA, 2008. ACM. ISBN 978-1-60558-189-7. doi: 10.1145/1410012.1410024.

M. Basiri, F. Schill, P. Lima, and D. Floreano. On-board relative bearing estimation for teams of drones using sound. *IEEE Robotics and Automation Letters*, 1(2):820–827, July 2016. ISSN 2377-3766. doi: 10.1109/LRA.2016.2527833.

R. W. Beard. State estimation for micro air vehicles. In J. S. Chahl, L. C. Jain, A. Mizutani, and M. Sato-Ilic, editors, *Innovations in Intelligent Machines - 1*, pages 173–199. Springer Berlin Heidelberg, Berlin, Heidelberg, 2007. ISBN 978-3-540-72696-8. doi: 10.1007/978-3-540-72696-8_7.

J. Boal, A. Sánchez-Miralles, and A. Arranz. Topological simultaneous localization and mapping: a survey. *Robotica*, 32(05):803–821, 2014. ISSN 0263-5747. doi: 10.1017/S0263574713001070.

M. Bonani, V. Longchamp, S. Magnenat, P. Rétornaz, D. Burnier, G. Roulet, F. Vaussard, H. Bleuler, and F. Mondada. The marxbot, a miniature mobile robot opening new perspectives for the collective-robotic research. In *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4187–4193, Oct 2010. doi: 10.1109/IROS.2010.5649153.

J. Borenstein and Liqiang Feng. Measurement and correction of systematic odometry errors in mobile robots. *IEEE Transactions on Robotics and Automation*, 12(6):869–880, Dec 1996. doi: 10.1109/70.544770.

J.-Y. Bouguet. Pyramidal implementation of the affine lucas kanade feature tracker description of the algorithm. *Intel Corporation*, 5(1-10):4, 2001.

M. Brambilla, E. Ferrante, M. Birattari, and M. Dorigo. Swarm robotics: a review from the swarm engineering perspective. *Swarm Intelligence*, 7(1):1–41, 2013. ISSN 1935-3812. doi: 10.1007/s11721-012-0075-2.

G. Bresson, Z. Alsayed, L. Yu, and S. Glaser. Simultaneous localization and mapping: A survey of current trends in autonomous driving. *IEEE Transactions on Intelligent Vehicles*, 2(3):194–220, Sep. 2017. doi: 10.1109/TIV.2017.2749181.

A. Briod, J.-C. Zufferey, and D. Floreano. Optic-flow based control of a 46g quadrotor. In *IROS 2013, Vision-based Closed-Loop Control and Navigation of Micro Helicopters in GPS-denied Environments Workshop*, number EPFL-CONF-189879, 2013.

A. Briod, J. C. Zufferey, and D. Floreano. A method for ego-motion estimation in micro-hovering platforms flying in very cluttered environments. *Autonomous Robots*, 40(5): 789–803, 2016. ISSN 15737527. doi: 10.1007/s10514-015-9494-4.

E. A. Capaldi, A. D. Smith, J. L. Osborne, S. E. Fahrbach, S. M. Farris, D. R. Reynolds, A. S. Edwards, A. Martin, G. E. Robinson, G. M. Poppy, et al. Ontogeny of orientation flight in the honeybee revealed by harmonic radar. *Nature*, 403(6769):537, 2000. doi: 10.1038/35000564.

C. Caron, D. Chamberland-Tremblay, C. Lapierre, P. Hadaya, S. Roche, and M. Saada. Indoor positioning. In S. Shekhar and H. Xiong, editors, *Encyclopedia of GIS*, pages 553–559. Springer US, Boston, MA, 2008. ISBN 978-0-387-35973-1. doi: 10.1007/978-0-387-35973-1_626.

B. A. Cartwright and T. S. Collett. Landmark learning in bees - experiments and models. *Journal of Comparative Physiology*, 151(4):521–543, 1983. ISSN 0340-7594 1432-1351. doi: 10.1007/Bf00605469.

K. Cesare, R. Skeele, Soo-Hyun Yoo, Yawei Zhang, and G. Hollinger. Multi-uav exploration with limited communication and battery. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2230–2235, May 2015. doi: 10.1109/ICRA.2015.7139494.

M. C. Chinnaaiah, K. Anusha, B. Bharat, M. Divya, P. S. Raju, and S. Dubey. Deliberation of curvature type obstacles: A new approach using fpga based robot. In *2018 International Conference on Control, Power, Communication and Computing Technologies (ICCPCCT)*, pages 519–522, March 2018. doi: 10.1109/ICCPCCT.2018.8574255.

M. Collett and T. S. Collett. Insect navigation: No map at the end of the trail? *Current Biology*, 16(2):R48 – R51, 2006. ISSN 0960-9822. doi: https://doi.org/10.1016/j.cub.2006.01.007.

M. Collett, L. Chittka, and T. Collett. Spatial memory in insect navigation. *Current Biology*, 23(17):R789 – R800, 2013. ISSN 0960-9822. doi: https://doi.org/10.1016/j.cub.2013.07.020.

P. Conroy, D. Bareiss, M. Beall, and J. v. d. Berg. 3-d reciprocal collision avoidance on physical quadrotor helicopters with on-board sensing for relative positioning. *arXiv preprint arXiv:1411.3794*, 2014.

M. Coppola, K. N. McGuire, K. Y. W. Scheper, and G. C. H. E. de Croon. On-board communication-based relative localization for collision avoidance in micro air vehicle teams. *Autonomous Robots*, 42(8):1787–1805, 2018. ISSN 0929-5593. doi: 10.1007/s10514-018-9760-3.

A. Cornejo and R. Nagpal. Distributed range-based relative localization of robot swarms. In H. L. Akin, N. M. Amato, V. Isler, and A. F. van der Stappen, editors, *Algorithmic Foundations of Robotics XI: Selected Contributions of the Eleventh International Workshop on the Algorithmic Foundations of Robotics*, pages 91–107. Springer International Publishing, Cham, 2015. ISBN 978-3-319-16595-0. doi: 10.1007/978-3-319-16595-0_6.

O. De Silva, G. K. I. Mann, and R. G. Gosine. Relative localization with symmetry preserving observers. In *2014 IEEE 27th Canadian Conference on Electrical and Computer Engineering (CCECE)*, pages 1–6, May 2014. doi: 10.1109/CCECE.2014.6901032.

C. De Wagter, S. Tijmons, B. D. W. Remes, and G. C. H. E. de Croon. Autonomous flight of a 20-gram flapping wing mav with a 4-gram onboard stereo vision system. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4982–4987, May 2014. doi: 10.1109/ICRA.2014.6907589.

A. Denuelle and M. V. Srinivasan. A sparse snapshot-based navigation strategy for uas guidance in natural environments. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3455–3462, May 2016. doi: 10.1109/ICRA.2016.7487524.

E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1(1):269–271, Dec 1959. ISSN 0945-3245. doi: 10.1007/BF01386390.

M. Dorigo, D. Floreano, L. M. Gambardella, F. Mondada, S. Nolfi, T. Baaboura, M. Birattari, M. Bonani, M. Brambilla, A. Brutschy, D. Burnier, A. Campo, A. L. Christensen, A. Decugniere, G. Di Caro, F. Ducatelle, E. Ferrante, A. Förster, J. M. Gonzales, J. Guzzi, V. Longchamp, S. Magnenat, N. Mathews, M. Montes De Oca, R. O'Grady, C. Pinciroli, G. Pini, P. Rétornaz, J. Roberts, V. Sperati, T. Stirling, A. Stranieri, T. Stützle, V. Trianni, E. Tuci, A. E. Turgut, and F. Vaussard. Swarmanoid: A novel concept for the study of heterogeneous robotic swarms. *IEEE Robotics and Automation Magazine*, 20 (December):60–71, 2013. ISSN 10709932. doi: 10.1109/MRA.2013.2252996.

M. Duarte, V. Costa, J. Gomes, T. Rodrigues, F. Silva, S. M. Oliveira, and A. L. Christensen. Evolution of collective behaviors for a real swarm of aquatic surface robots. *PLoS One*, 11(3):e0151834, 2016. ISSN 1932-6203. doi: 10.1371/journal.pone.0151834.

O. Dunkley, J. Engel, J. Sturm, and D. Cremers. Visual-inertial navigation for a camera-equipped 25g nano-quadrotor. In *IROS2014 aerial open source robotics workshop*, page 2, 2014.

J. Dupeyroux, J. R. Serres, and S. Viollet. Antbot: A six-legged walking robot able to home like desert ants in outdoor environments. *Science Robotics*, 4(27):eaau0307, 2019. ISSN 2470-9476. doi: 10.1126/scirobotics.aau0307.

H. Durrant-Whyte and T. Bailey. Simultaneous localization and mapping: part i. *IEEE Robotics Automation Magazine*, 13(2):99–110, June 2006. doi: 10.1109/MRA.2006.1638022.

H. Durrant-Whyte, N. Roy, and P. Abbeel. *TERMES: An Autonomous Robotic System for Three-Dimensional Collective Construction*. MITP, 2012.

A. Ebrahimi, F. Janabi-Sharifi, and A. Ghanbari. Uavisbug: vision-based 3d motion planning and obstacle avoidance for a mini-uav in an unknown indoor environment. *Canadian Aeronautics and Space Journal*, 60(01):9–21, 2014. doi: 10.5589/q14-005.

M. Elbanhawi, A. Mohamed, R. Clothier, J. Palmer, M. Simic, and S. Watkins. Enabling technologies for autonomous mav operations. *Progress in Aerospace Sciences*, 91:27 – 52, 2017. ISSN 0376-0421. doi: https://doi.org/10.1016/j.paerosci.2017.03.002.

J. Engel, T. Schöps, and D. Cremers. Lsd-slam: Large-scale direct monocular slam. In D. Fleet, T. Pajdla, B. Schiele, and T. Tuytelaars, editors, *Computer Vision – ECCV 2014*, pages 834–849, Cham, 2014. Springer International Publishing. ISBN 978-3-319-10605-2.

G. Farnebäck. Two-frame motion estimation based on polynomial expansion. In J. Bigun and T. Gustavsson, editors, *Image Analysis*, pages 363–370, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg. ISBN 978-3-540-45103-7.

P. Fiorini and Z. Shiller. Motion planning in dynamic environments using velocity obstacles. *The International Journal of Robotics Research*, 17(7):760–772, 1998. doi: 10.1177/027836499801700706.

M. A. Fischler and R. C. Bolles. Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. In M. A. Fischler and O. Firschein, editors, *Readings in Computer Vision*, pages 726 – 740. Morgan Kaufmann, San Francisco (CA), 1987. ISBN 978-0-08-051581-6. doi: https://doi.org/10.1016/B978-0-08-051581-6.50070-2.

D. Floreano and R. J. Wood. Science, technology and the future of small autonomous drones. *Nature*, 521(7553):460, 2015. ISSN 1476-4687. doi: 10.1038/nature14542.

C. Forster, S. Lynen, L. Kneip, and D. Scaramuzza. Collaborative monocular slam with multiple micro aerial vehicles. In *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3962–3970, Nov 2013. doi: 10.1109/IROS.2013.6696923.

F. Fraundorfer, C. Engels, and D. Nistér. Topological mapping, localization and navigation using image collections. In *IEEE International Conference on Intelligent Robots and Systems*, pages 3872–3877. IEEE, 2007. ISBN 1424409128. doi: 10.1109/IROS.2007.4399123.

J. Fuentes-Pacheco, J. Ruiz-Ascencio, and J. M. Rendón-Mancha. Visual simultaneous localization and mapping: a survey. *Artificial Intelligence Review*, 43(1):55–81, 2012. ISSN 0269-2821 1573-7462. doi: 10.1007/s10462-012-9365-8.

E. Garcia-Fidalgo and A. Ortiz. Vision-based topological mapping and localization methods: A survey. *Robotics and Autonomous Systems*, 64:1–20, 2015. ISSN 09218890. doi: 10.1016/j.robot.2014.11.009.

T. Gaugel, J. Mittag, H. Hartenstein, S. Papanastasiou, and E. G. Ström. In-depth analysis and evaluation of self-organizing tdma. In *2013 IEEE Vehicular Networking Conference*, pages 79–86, Dec 2013. doi: 10.1109/VNC.2013.6737593.

A. Geiger, J. Ziegler, and C. Stiller. Stereoscan: Dense 3d reconstruction in real-time. In *2011 IEEE Intelligent Vehicles Symposium (IV)*, pages 963–968, June 2011. doi: 10.1109/IVS.2011.5940405.

T. Goedemé, M. Nuttin, T. Tuytelaars, and L. Van Gool. Omnidirectional vision based topological navigation. *International Journal of Computer Vision*, 74(3):219–236, 2007. ISSN 09205691. doi: 10.1007/s11263-006-0025-9.

V. Grabe, H. H. Bulthoff, D. Scaramuzza, and P. R. Giordano. Nonlinear ego-motion estimation from optical flow for online control of a quadrotor UAV. *The International Journal of Robotics Research*, 34(8):1114–1135, 2015. ISSN 0278-3649. doi: 10.1177/0278364915578646.

W. E. Green and P. Y. Oh. Optic-flow-based collision avoidance. *IEEE Robotics Automation Magazine*, 15(1):96–103, March 2008. doi: 10.1109/MRA.2008.919023.

S. Grzonka, G. Grisetti, and W. Burgard. A fully autonomous indoor quadrotor. *IEEE Transactions on Robotics*, 28(1):90–100, Feb 2012. doi: 10.1109/TRO.2011.2162999.

M. M. Gulzar, Q. Ling, M. Yaqoob, and S. Iqbal. Realization of an improved path planning strategy. In *Proc. Automation and Information Sciences (ICCAIS) 2015 Int. Conf. Control*, pages 384–389, #oct# 2015. doi: 10.1109/ICCAIS.2015.7338698.

K. Guo, Z. Qiu, W. Meng, L. Xie, and R. Teo. Ultra-wideband based cooperative relative localization algorithm and experiments for multiple unmanned aerial vehicles in gps denied environments. *International Journal of Micro Air Vehicles*, 9(3):169–186, 2017. doi: 10.1177/1756829317695564.

P. E. Hart, N. J. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2): 100–107, July 1968. doi: 10.1109/TSSC.1968.300136.

H. Hashemi. The indoor radio propagation channel. *Proceedings of the IEEE*, 81(7):943–968, July 1993. doi: 10.1109/5.231342.

S. Hauert, S. Leven, M. Varga, F. Ruini, A. Cangelosi, J.-C. Zufferey, and D. Floreano. Reynolds flocking in reality with fixed-wing robots: communication range vs. maximum turning rate. In *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5015–5020. IEEE, 2011. ISBN 1612844561.

H. Hirschmuller. Stereo processing by semiglobal matching and mutual information. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 30(2):328–341, Feb 2008. doi: 10.1109/TPAMI.2007.1166.

D. Honegger, L. Meier, P. Tanskanen, and M. Pollefeys. An open source and open hardware embedded metric optical flow cmos camera for indoor and outdoor applications. In *2013 IEEE International Conference on Robotics and Automation*, pages 1736–1741, May 2013. doi: 10.1109/ICRA.2013.6630805.

Y. Horiuchi and H. Noborio. Evaluation of path length made in sensor-based path-planning with the alternative following. In *Proceedings 2001 ICRA. IEEE International Conference on Robotics and Automation (Cat. No.01CH37164)*, volume 2, pages 1728–1735 vol.2, May 2001. doi: 10.1109/ROBOT.2001.932860.

B. K. Horn and B. G. Schunck. Determining optical flow. *Artificial Intelligence*, 17(1-3): 185–203, 1981. ISSN 00043702. doi: 10.1016/0004-3702(81)90024-2.

X. Hu and P. Mordohai. Evaluation of stereo confidence indoors and outdoors. In *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 1466–1473, June 2010. doi: 10.1109/CVPR.2010.5539798.

N. T. Jafferis, E. F. Helbling, M. Karpelson, and R. J. Wood. Untethered flight of an insect-sized flapping-wing microscale aerial vehicle. *Nature*, 570(7762):491, 2019. doi: 10.1038/s41586-019-1322-0.

Jianbo Shi and Tomasi. Good features to track. In *1994 Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, pages 593–600, June 1994. doi: 10.1109/CVPR.1994.323794.

S. Jones, M. Studley, S. Hauert, and A. F. T. Winfield. A two teraflop swarm. *Frontiers in Robotics and AI*, 5:11, 2018. ISSN 2296-9144. doi: 10.3389/frobt.2018.00011.

S. Jung, S. Hwang, H. Shin, and D. H. Shim. Perception, guidance, and navigation for indoor autonomous drone racing using deep learning. *IEEE Robotics and Automation Letters*, 3(3):2539–2544, July 2018. doi: 10.1109/LRA.2018.2808368.

I. Kamon and E. Rivlin. Sensory-based motion planning with global proofs. *IEEE Transactions on Robotics and Automation*, 13(6):814–822, 1997. ISSN 1042296X. doi: 10.1109/70.650160.

I. Kamon, E. Rivlin, and E. Rimon. A new range-sensor based globally convergent navigation algorithm for mobile robots. In *Proceedings of IEEE International Conference on Robotics and Automation*, volume 1, pages 429–435 vol.1, April 1996. doi: 10.1109/ROBOT.1996.503814.

I. Kamon, E. Rimon, and E. Rivlin. Range-sensor based navigation in three dimensions. In *Proc. IEEE Int. Conf. Robotics and Automation (Cat. No.99CH36288C)*, volume 1, pages 163–169 vol.1, 1999. doi: 10.1109/ROBOT.1999.769955.

F. Kendoul, I. Fantoni, and K. Nonami. Optic flow-based vision system for autonomous 3d localization and control of small aerial vehicles. *Robotics and Autonomous Systems*, 57(6):591 – 602, 2009a. ISSN 0921-8890. doi: 10.1016/j.robot.2009.02.001.

F. Kendoul, K. Nonami, I. Fantoni, and R. Lozano. An adaptive vision-based autopilot for mini flying machines guidance, navigation and control. *Autonomous Robots*, 27(3): 165, Aug 2009b. ISSN 1573-7527. doi: 10.1007/s10514-009-9135-x.

D.-H. Kim, K. Shin, C.-S. Han, and J. Y. Lee. Sensor-based navigation of a car-like robot based on bug family algorithms. *Proceedings of the Institution of Mechanical Engineers, Part C: Journal of Mechanical Engineering Science*, 227(6):1224–1241, 2013. doi: 10.1177/0954406212458202.

A. Kushki, K. Plataniotis, and A. Venetsanopoulos. Indoor positioning with wireless local area networks (wlan). In S. Shekhar and H. Xiong, editors, *Encyclopedia of GIS*, pages 566–571. Springer US, Boston, MA, 2008. ISBN 978-0-387-35973-1. doi: 10.1007/978-0-387-35973-1_629.

A. Kushleyev, D. Mellinger, C. Powers, and V. Kumar. Towards a swarm of agile micro quadrotors. *Autonomous Robots*, 35(4):287–300, 2013. ISSN 0929-5593. doi: 10.1007/s10514-013-9349-9.

D. Lambrinos, R. Möller, T. Labhart, R. Pfeifer, and R. Wehner. A mobile robot employing insect strategies for navigation. *Robotics and Autonomous Systems*, 30(1-2):39–64, jan 2000. ISSN 09218890. doi: 10.1016/S0921-8890(99)00064-0.

S. Laubach and J. Burdick. Roverbug: Long range navigation for mars rovers. *Experimental Robotics VI*, Jan. 2000. doi: 10.1007/BFb0119412.

S. L. Laubach and J. W. Burdick. An autonomous sensor-based path-planner for planetary microrovers. In *Proc. IEEE Int. Conf. Robotics and Automation (Cat. No.99CH36288C)*, volume 1, pages 347–354 vol.1, 1999. doi: 10.1109/ROBOT.1999.770003.

S. M. LaValle and J. James J. Kuffner. Randomized kinodynamic planning. *The International Journal of Robotics Research*, 20(5):378–400, 2001. doi: 10.1177/02783640122067453.

A. Ledergerber, M. Hamer, and R. D'Andrea. A robot self-localization system using one-way ultra-wideband communication. In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3131–3137, Sep. 2015. doi: 10.1109/IROS.2015.7353810.

D.-J. Lee, R. W. Beard, P. C. Merrell, and P. Zhan. See and avoidance behaviors for autonomous navigation. In D. W. Gage, editor, *Mobile Robots XVII*, volume 5609, pages 23 – 34. International Society for Optics and Photonics, SPIE, 2004. doi: 10.1117/12.571550.

S. Lee, T. M. Adams, and B. yeol Ryoo. A fuzzy navigation system for mobile construction robots. *Automation in Construction*, 6(2):97 – 107, 1997. ISSN 0926-5805. doi: 10.1016/S0926-5805(96)00185-9.

H. C. Longuet-Higgins and K. Prazdny. The interpretation of a moving retinal image. *Proceedings of the Royal Society of London B: Biological Sciences*, 208(1173):385–397, 1980. doi: 10.1098/rspb.1980.0057.

V. Lumelsky and T. Skewis. A paradigm for incorporating vision in the robot navigation function. In *Proceedings. 1988 IEEE International Conference on Robotics and Automation*, pages 734–739 vol.2, April 1988. doi: 10.1109/ROBOT.1988.12146.

V. Lumelsky and A. Stepanov. Dynamic path planning for a mobile automaton with limited information on the environment. *IEEE Transactions on Automatic Control*, 31(11): 1058–1063, November 1986. doi: 10.1109/TAC.1986.1104175.

V. J. Lumelsky and T. Skewis. Incorporating range sensing in the robot navigation function. *IEEE Transactions on Systems, Man, and Cybernetics*, 20(5):1058–1069, Sep. 1990. doi: 10.1109/21.59969.

V. J. Lumelsky and A. A. Stepanov. Path-planning strategies for a point mobile automaton moving amidst unknown obstacles of arbitrary shape. *Algorithmica*, 2(1):403–430, Nov 1987. ISSN 1432-0541. doi: 10.1007/BF01840369.

M. T. Lázaro, L. M. Paz, P. Piniés, J. A. Castellanos, and G. Grisetti. Multi-robot slam using condensed measurements. In *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1069–1076, Nov 2013. doi: 10.1109/IROS.2013.6696483.

E. López, S. García, R. Barea, L. M. Bergasa, E. J. Molinos, R. Arroyo, E. Romera, and S. Pardo. A multi-sensorial simultaneous localization and mapping (slam) system for low-cost micro aerial vehicles in gps-denied environments. *Sensors*, 17(4), 2017. ISSN 1424-8220. doi: 10.3390/s17040802.

K. Y. Ma, P. Chirarattananon, S. B. Fuller, and R. J. Wood. Controlled flight of a biologically inspired, insect-scale robot. *Science*, 340(6132):603–7, 2013. ISSN 1095-9203 (Electronic) 0036-8075 (Linking). doi: 10.1126/science.1231806.

E. Magid and E. Rivlin. Cautiousbug: a competitive algorithm for sensory-based robot navigation. In *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566)*, volume 3, pages 2757–2762 vol.3, Sep. 2004. doi: 10.1109/IROS.2004.1389826.

V. Malyavej, W. Kumkeaw, and M. Aorpimai. Indoor robot localization by rssi/imu sensor fusion. In *2013 10th International Conference on Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology*, pages 1–6, May 2013. doi: 10.1109/ECTICon.2013.6559517.

R. Marino, F. Mastrogiovanni, A. Sgorbissa, and R. Zaccaria. A minimalistic quadrotor navigation strategy for indoor multi-floor scenarios. In E. Menegatti, N. Michael,

K. Berns, and H. Yamaguchi, editors, *Intelligent Autonomous Systems 13*, pages 1561–1570. Springer International Publishing, Cham, 2016. ISBN 978-3-319-08338-4.

A. Marjovi and L. Marques. Multi-robot olfactory search in structured environments. *Robotics and Autonomous Systems*, 59(11):867–881, 2011. ISSN 09218890. doi: 10.1016/j.robot.2011.07.010.

A. Martinelli and R. Siegwart. Observability analysis for mobile robot localization. In *2005 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1471–1476, Aug 2005. doi: 10.1109/IROS.2005.1545153.

A. Martinelli, F. Pont, and R. Siegwart. Multi-robot localization using relative observations. In *Proceedings of the 2005 IEEE International Conference on Robotics and Automation*, pages 2797–2802, April 2005. doi: 10.1109/ROBOT.2005.1570537.

F. Mastrogiovanni, A. Sgorbissa, and R. Zaccaria. Robust navigation in an unknown environment with minimal sensing and representation. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 39(1):212–229, Feb 2009. doi: 10.1109/TSMCB.2008.2004505.

L. Matthies, R. Brockers, Y. Kuwata, and S. Weiss. Stereo vision-based obstacle avoidance for micro air vehicles using disparity space. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3242–3249, May 2014. doi: 10.1109/ICRA.2014.6907325.

K. McGuire, G. de Croon, C. de Wagter, B. Remes, K. Tuyls, and H. Kappen. Local histogram matching for efficient optical flow computation applied to velocity estimation on pocket drones. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3255–3260, May 2016. doi: 10.1109/ICRA.2016.7487496.

K. McGuire, M. Coppola, C. de Wagter, and G. de Croon. Towards autonomous navigation of multiple pocket-drones in real-world environments. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 244–249, Sep. 2017. doi: 10.1109/IROS.2017.8202164.

K. McGuire, G. de Croon, C. De Wagter, K. Tuyls, and H. Kappen. Efficient optical flow and stereo vision for velocity estimation and obstacle avoidance on an autonomous pocket drone. *IEEE Robotics and Automation Letters*, 2(2):1070–1076, 2017. ISSN 2377-3766 2377-3774. doi: 10.1109/lra.2017.2658940.

K. McGuire, G. de Croon, and K. Tuyls. A comparative study of bug algorithms for robot navigation. *Robotics and Autonomous Systems*, 121:103261, 2019. ISSN 0921-8890. doi: 10.1016/j.robot.2019.103261.

B. G. McHenry. Head injury criterion and the atb. *ATB Users' group*, pages 5–8, 2004.

R. Menzel and M. Giurfa. Cognitive architecture of a mini-brain: the honeybee. *Trends in Cognitive Sciences*, 5(2):62 – 71, 2001. ISSN 1364-6613. doi: 10.1016/S1364-6613(00)01601-6.

R. Menzel and U. Greggers. The memory structure of navigation in honeybees. *Journal of Comparative Physiology A: Neuroethology, Sensory, Neural, and Behavioral Physiology*, 201(6):547–561, 2015. ISSN 14321351. doi: 10.1007/s00359-015-0987-6.

R. Menzel, J. Fuchs, A. Kirbach, K. Lehmann, and U. Greggers. *Navigation and Communication in Honey Bees*. Springer Netherlands, Dordrecht, 2012. ISBN 978-94-007-2099-2. doi: 10.1007/978-94-007-2099-2_9.

S. Milius. Swarm savvy, how bees, ants and other animals avoid dumb collective decisions. *Science News*, 9:16–21, 2009.

S. Mishra and P. Bande. Maze solving algorithms for micro mouse. In *2008 IEEE International Conference on Signal Image Technology and Internet Based Systems*, pages 86–93, Nov 2008. doi: 10.1109/SITIS.2008.104.

K. Mohta, M. Watterson, Y. Mulgaonkar, S. Liu, C. Qu, A. Makineni, K. Saulnier, K. Sun, A. Zhu, J. Delmerico, K. Karydis, N. Atanasov, G. Loianno, D. Scaramuzza, K. Daniilidis, C. J. Taylor, and V. Kumar. Fast, autonomous flight in gps-denied and cluttered environments. *Journal of Field Robotics*, 35(1):101–120, 2018. doi: 10.1002/rob.21774.

F. Mondada, L. M. Gambardella, D. Floreano, S. Nolfi, J. . Deneuborg, and M. Dorigo. The cooperation of swarm-bots: physical interactions in collective robotics. *IEEE Robotics Automation Magazine*, 12(2):21–28, June 2005. doi: 10.1109/MRA.2005.1458313.

R. J. D. Moore, K. Dantu, G. L. Barrows, and R. Nagpal. Autonomous mav guidance with a lightweight omnidirectional vision sensor. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3856–3861, May 2014. doi: 10.1109/ICRA.2014.6907418.

T. Mori and S. Scherer. First results in detecting and avoiding frontal obstacles from a monocular camera for micro unmanned aerial vehicles. In *2013 IEEE International Conference on Robotics and Automation*, pages 1750–1757, May 2013. doi: 10.1109/ICRA.2013.6630807.

M. W. Mueller, M. Hamer, and R. D'Andrea. Fusing ultra-wideband range measurements with accelerometers and rate gyroscopes for quadrocopter state estimation. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1730–1736, May 2015. doi: 10.1109/ICRA.2015.7139421.

M. W. Mueller, M. Hehn, and R. D'Andrea. Covariance correction step for kalman filtering with an attitude. *Journal of Guidance, Control, and Dynamics*, 40(9):2301–2306, 2017. doi: 10.2514/1.G000848.

Y. Mulgaonkar, G. Cross, and V. Kumar. Design of small, safe and robust quadrotor swarms. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2208–2215, May 2015. doi: 10.1109/ICRA.2015.7139491.

Y. Mulgaonkar, A. Makineni, L. Guerrero-Bonilla, and V. Kumar. Robust aerial robot swarms without collision avoidance. *IEEE Robotics and Automation Letters*, 3(1):596–603, 2018. ISSN 2377-3766. doi: 10.1109/Lra.2017.2775699.

J. Ng and T. Bräunl. Performance comparison of bug navigation algorithms. *Journal of Intelligent and Robotic Systems*, 50(1):73–84, Sep 2007. ISSN 1573-0409. doi: 10.1007/s10846-007-9157-6.

K. Nguyen and Z. Luo. Evaluation of bluetooth properties for indoor localisation. In J. M. Krisp, editor, *Progress in Location-Based Services*, pages 127–149. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013. ISBN 978-3-642-34203-5. doi: 10.1007/978-3-642-34203-5_8.

Q. M. Nguyen, L. N. M. Tran, and T. C. Phung. A study on building optimal path planning algorithms for mobile robot. In *2018 4th International Conference on Green Technology and Sustainable Development (GTSD)*, pages 341–346, Nov 2018. doi: 10.1109/GTSD.2018.8595558.

S. Nirjon, J. Liu, G. DeJean, B. Priyantha, Y. Jin, and T. Hart. Coin-gps: Indoor localization from direct gps receiving. In *Proceedings of the 12th Annual International Conference on Mobile Systems, Applications, and Services*, MobiSys '14, pages 301–314, New York, NY, USA, 2014. ACM. ISBN 978-1-4503-2793-0. doi: 10.1145/2594368.2594378.

H. No, A. Cho, and C. Kee. Attitude estimation method for small uav under accelerative environment. *GPS Solutions*, 19(3):343–355, Jul 2015. ISSN 1521-1886. doi: 10.1007/s10291-014-0391-7.

H. Noborio, Y. Maeda, and K. Urakawa. Three or more dimensional sensor-based path-planning algorithm hd-i. In *Proceedings 1999 IEEE/RSJ International Conference on Intelligent Robots and Systems. Human and Environment Friendly Robots with High Intelligence and Emotional Quotients (Cat. No.99CH36289)*, volume 3, pages 1699–1706 vol.3, Oct 1999. doi: 10.1109/IROS.1999.811723.

H. Noborio, K. Fujimura, and Y. Horiuchi. A comparative study of sensor-based path-planning algorithms in an unknown maze. In *Proceedings. 2000 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2000) (Cat. No.00CH37113)*, volume 2, pages 909–916 vol.2, Oct 2000. doi: 10.1109/IROS.2000.893135.

H. Noborio, R. Nogami, and S. Hirao. A new sensor-based path-planning algorithm whose path length is shorter on the average. In *IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA '04. 2004*, volume 3, pages 2832–2839 Vol.3, April 2004. doi: 10.1109/ROBOT.2004.1307490.

S. Nunnally, P. Walker, A. Kolling, N. Chakraborty, M. Lewis, K. Sycara, and M. Goodrich. Human influence of robotic swarms with bandwidth and localization issues. In *2012 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, pages 333–338, Oct 2012. doi: 10.1109/ICSMC.2012.6377723.

H. Oleynikova, D. Honegger, and M. Pollefeys. Reactive avoidance using embedded stereo vision for MAV flight. *IEEE International Conference on Robotics and Automation*, pages 50–56, 2015. doi: 10.1109/ICRA.2015.7138979.

D. Palossi, A. Loquercio, F. Conti, E. Flamand, D. Scaramuzza, and L. Benini. Ultra low power deep-learning-powered autonomous nano drones. *arXiv preprint arXiv:1805.01831*, 2018.

D. Palossi, A. Loquercio, F. Conti, F. Conti, E. Flamand, E. Flamand, D. Scaramuzza, L. Benini, and L. Benini. A 64mw dnn-based visual navigation engine for autonomous nano-drones. *IEEE Internet of Things Journal*, pages 1–1, 2019. doi: 10.1109/JIOT.2019.2917066.

C. Pinciroli, V. Trianni, R. O'Grady, G. Pini, A. Brutschy, M. Brambilla, N. Mathews, E. Ferrante, G. Di Caro, F. Ducatelle, M. Birattari, L. M. Gambardella, and M. Dorigo. ARGoS: A modular, parallel, multi-engine simulator for multi-robot systems. *Swarm Intelligence*, 6(4):271–295, 2012. ISSN 19353812. doi: 10.1007/s11721-012-0072-5.

J. A. Preiss, W. Honig, G. S. Sukhatme, and N. Ayanian. Crazyswarm: A large nano-quadcopter swarm. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3299–3304, May 2017. doi: 10.1109/ICRA.2017.7989376.

J. Primicerio, S. F. Di Gennaro, E. Fiorillo, L. Genesio, E. Lugato, A. Matese, and F. P. Vaccari. A flexible unmanned aerial vehicle for precision agriculture. *Precision Agriculture*, 13(4):517–523, Aug 2012. ISSN 1573-1618. doi: 10.1007/s11119-012-9257-6.

C. R. Reid, D. J. Sumpter, and M. Beekman. Optimisation in a natural system: Argentine ants solve the towers of hanoi. *Journal of Experimental Biology*, 214(Pt 1):50–8, 2011. ISSN 1477-9145 (Electronic) 0022-0949 (Linking). doi: 10.1242/jeb.048173.

J. Reinhard and M. V. Srinivasan. The role of scents in honey bee foraging and recruitment. *Food exploitation by social insects: ecological, behavioral, and theoretical approaches*, 1:165–182, 2009.

B. Remes, P. Esden-Tempski, F. Van Tienen, E. Smeur, C. De Wagter, and G. De Croon. Lisa-S 2.8g autopilot for GPS-based flight of MAVs. In *IMAV 2014: International Micro Air Vehicle Conference and Competition 2014*, pages 280–285. Delft University of Technology, 2014. ISBN doi:10.4233/uuid:29e5367f-6c16-43e1-989e-d51cf8b51f7d. doi: 10.4233.

J. F. Roberts, T. S. Stirling, J. Zufferey, and D. Floreano. 2.5d infrared range and bearing system for collective robotics. In *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3659–3664, Oct 2009. doi: 10.1109/IROS.2009.5354263.

S. Roelofsen, D. Gillet, and A. Martinoli. Reciprocal collision avoidance for quadrotors using on-board visual detection. In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4810–4817, Sep. 2015. doi: 10.1109/IROS.2015.7354053.

H. Romero, S. Salazar, and R. Lozano. Real-time stabilization of an eight-rotor UAV using optical flow. *IEEE Transactions on Robotics*, 25(4):809–817, 2009. ISSN 15523098. doi: 10.1109/TRO.2009.2018972.

E. Rosten and T. Drummond. Fusing points and lines for high performance tracking. In *Proceedings of the IEEE International Conference on Computer Vision*, volume II, pages 1508–1515, 2005. ISBN 076952334X. doi: 10.1109/ICCV.2005.104.

M. Rubenstein, A. Cornejo, and R. Nagpal. Robotics. programmable self-assembly in a thousand-robot swarm. *Science*, 345(6198):795–9, 2014. ISSN 1095-9203 (Electronic) 0036-8075 (Linking). doi: 10.1126/science.1254295.

F. Ruffier and N. Franceschini. Optic flow regulation in unsteady environments: A tethered mav achieves terrain following and targeted landing over a moving platform. *Journal of Intelligent & Robotic Systems*, 79(2):275–293, Aug 2015. ISSN 1573-0409. doi: 10.1007/s10846-014-0062-5.

F. Ruffier, S. Viollet, S. Amic, and N. Franceschini. Bio-inspired optical flow circuits for the visual guidance of micro air vehicles. *Proceedings of the 2003 International Symposium on Circuits and Systems, 2003. ISCAS '03.*, 3:846–849, 2003. doi: 10.1109/ISCAS.2003.1205152.

A. M. Sabatini and V. Genovese. A stochastic approach to noise modeling for barometric altimeters. *Sensors*, 13(11):15692–15707, 2013. ISSN 1424-8220. doi: 10.3390/s131115692.

E. Şahin. Swarm robotics: From sources of inspiration to domains of application. In E. Şahin and W. M. Spears, editors, *Swarm Robotics*, pages 10–20, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg. ISBN 978-3-540-30552-1.

A. Sankaranarayanar and M. Vidyasagar. Path planning for moving a point object amidst unknown obstacles in a plane: a new algorithm and a general theory for algorithm development. In *29th IEEE Conference on Decision and Control*, pages 1111–1119 vol.2, Dec 1990. doi: 10.1109/CDC.1990.203773.

N. J. Sanket, C. D. Singh, K. Ganguly, C. Fermüller, and Y. Aloimonos. Gapflyt: Active vision based minimalist structure-less gap detection for quadrotor flight. *IEEE Robotics and Automation Letters*, 3(4):2799–2806, 2018. ISSN 2377-3766. doi: 10.1109/Lra.2018.2843445.

M. Saska, T. Baca, J. Thomas, J. Chudoba, L. Preucil, T. Krajnik, J. Faigl, G. Loianno, and V. Kumar. System for deployment of groups of unmanned micro aerial vehicles in gps-denied environments using onboard visual relative localization. *Autonomous Robots*, 41(4):919–944, Apr 2017. ISSN 1573-7527. doi: 10.1007/s10514-016-9567-z.

D. Scaramuzza and F. Fraundorfer. Visual odometry [tutorial]. *IEEE Robotics Automation Magazine*, 18(4):80–92, Dec 2011. doi: 10.1109/MRA.2011.943233.

D. Scaramuzza, M. C. Achtelik, L. Doitsidis, F. Friedrich, E. Kosmatopoulos, A. Martinelli, M. W. Achtelik, M. Chli, S. Chatzichristofis, L. Kneip, D. Gurdan, L. Heng, G. H. Lee, S. Lynen, M. Pollefeys, A. Renzaglia, R. Siegwart, J. C. Stumpf, P. Tanskanen, C. Troiani, S. Weiss, and L. Meier. Vision-controlled micro flying robots: From system design to autonomous navigation and mapping in gps-denied environments. *IEEE Robotics*

Automation Magazine, 21(3):26–40, Sept 2014. ISSN 1070-9932. doi: 10.1109/MRA.2014.2322295.

K. Y. W. Scheper, M. Karásek, C. De Wagter, B. D. W. Remes, and G. C. H. E. De Croon. First autonomous multi-room exploration with an insect-inspired flapping wing vehicle. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1–7, May 2018. doi: 10.1109/ICRA.2018.8460702.

T. D. Seeley. *The wisdom of the hive: the social physiology of honey bee colonies*. Harvard University Press, 2009.

J. S. Seybold. *Introduction to RF propagation*. John Wiley & Sons, 2005.

S. Shen, N. Michael, and V. Kumar. 3d indoor exploration with a computationally constrained mav. In *Robotics: Science and Systems*, 2011.

S. Shen, N. Michael, and V. Kumar. Autonomous multi-floor indoor navigation with a computationally constrained micro aerial vehicle. In *2011 IEEE International Conference on Robotics and Automation*, pages 2968–2969, May 2011. doi: 10.1109/ICRA.2011.5980364.

K. Shilov. The next generation design of autonomous mav flight control system smartap. In *IMAV 2014: International Micro Air Vehicle Conference and Competition 2014, Delft, The Netherlands, August 12-15, 2014*. Delft University of Technology, 2014.

E. J. J. Smeur, Q. Chu, and G. C. H. E. de Croon. Adaptive incremental nonlinear dynamic inversion for attitude control of micro air vehicles. *Journal of Guidance, Control, and Dynamics*, 39(3):450–461, 2016. doi: 10.2514/1.G001490.

M. V. Srinivasan. Honeybees as a model for the study of visually guided flight, navigation, and biologically inspired robotics. *Physiological Reviews*, 91(2):413–460, 2011. ISSN 0031-9333. doi: 10.1152/physrev.00005.2010.

M. V. Srinivasan, S. Zhang, and N. Bidwell. Visually mediated odometry in honeybees. *Journal of Experimental Biology*, 200(19):2513–2522, 1997. ISSN 0022-0949.

A. Stelzer, M. Vayugundla, E. Mair, M. Suppa, and W. Burgard. Towards efficient and scalable visual homing. *The International Journal of Robotics Research*, 37(2-3):225–248, 2018. doi: 10.1177/0278364918761115.

J. Svečko, M. Malajner, and D. Gleich. Distance estimation using rssi and particle filter. *ISA Transactions*, 55:275 – 285, 2015. ISSN 0019-0578.

T. Szabo. Autonomous collision avoidance for swarms of mavs based solely on rssi measurements. Master's thesis, Delft University of Technology, 2015.

K. Taylor and S. M. LaValle. I-bug: An intensity-based bug algorithm. In *2009 IEEE International Conference on Robotics and Automation*, pages 3981–3986, May 2009. doi: 10.1109/ROBOT.2009.5152728.

K. Taylor and S. M. LaValle. Intensity-based navigation with global guarantees. *Autonomous Robots*, 36(4):349–364, Apr 2014. ISSN 1573-7527. doi: 10.1007/s10514-013-9356-x.

S. Tijmons, G. C. H. E. de Croon, B. D. W. Remes, C. De Wagter, and M. Mulder. Obstacle avoidance strategy using onboard stereo vision on a flapping wing mav. *IEEE Transactions on Robotics*, 33(4):858–874, Aug 2017. doi: 10.1109/TRO.2017.2683530.

T. Tomic, K. Schmid, P. Lutz, A. Domel, M. Kassecker, E. Mair, I. L. Grixa, F. Ruess, M. Suppa, and D. Burschka. Toward a fully autonomous uav: Research platform for indoor and outdoor urban search and rescue. *IEEE Robotics Automation Magazine*, 19 (3):46–56, Sep. 2012. doi: 10.1109/MRA.2012.2206473.

C. Toth and G. Jóźków. Remote sensing platforms and sensors: A survey. *ISPRS Journal of Photogrammetry and Remote Sensing*, 115:22 – 36, 2016. ISSN 0924-2716. doi: https://doi.org/10.1016/j.isprsjprs.2015.10.004. Theme issue 'State-of-the-art in photogrammetry, remote sensing and spatial information science'.

K. Townsend, C. Cufí, Akiba, and R. Davidson. *Getting started with Bluetooth low energy: Tools and techniques for low-power networking*. O'Reilly Media, Inc., 2014.

J. N. Twigg, J. R. Fink, P. L. Yu, and B. M. Sadler. Rss gradient-assisted frontier exploration and radio source localization. In *2012 IEEE International Conference on Robotics and Automation*, pages 889–895, May 2012. doi: 10.1109/ICRA.2012.6225059.

G. J. J. van Dalen, K. N. McGuire, and G. C. H. E. de Croon. Visual homing for micro aerial vehicles using scene familiarity. *Unmanned Systems*, 06(02):119–130, 2018. doi: 10.1142/S230138501850005X.

S. van der Helm, M. Coppola, K. N. McGuire, and G. C. H. E. de Croon. On-board range-based relative localization for micro air vehicles in indoor leader–follower flight. *Autonomous Robots*, Mar 2019. ISSN 1573-7527. doi: 10.1007/s10514-019-09843-6.

A. Veeraraghavan, R. Chellappa, and M. Srinivasan. Shape-and-behavior encoded tracking of bee dances. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 30 (3):463–476, March 2008. doi: 10.1109/TPAMI.2007.70707.

G. Vásárhelyi, C. Virágh, G. Somorjai, N. Tarcai, T. Szörényi, T. Nepusz, and T. Vicsek. Outdoor flocking and formation flight with autonomous aerial robots. In *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3866–3873, Sep. 2014. doi: 10.1109/IROS.2014.6943105.

G. Vásárhelyi, C. Virágh, G. Somorjai, T. Nepusz, A. E. Eiben, and T. Vicsek. Optimized flocking of autonomous drones in confined environments. *Science Robotics*, 3(20): eaat3536, 2018. ISSN 2470-9476. doi: 10.1126/scirobotics.aat3536.

A. Weinstein, A. Cho, G. Loianno, and V. Kumar. Visual inertial odometry swarm: An autonomous swarm of vision-based quadrotors. *IEEE Robotics and Automation Letters*, 3(3):1801–1807, July 2018. doi: 10.1109/LRA.2018.2800119.

D. Wilkie, J. van den Berg, and D. Manocha. Generalized velocity obstacles. In *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5573–5578, Oct 2009. doi: 10.1109/IROS.2009.5354175.

Q.-l. Xu. Randombug: Novel path planning algorithm in unknown environment. *Open Electrical & Electronic Engineering Journal*, 8:252–257, 2014.

Q.-L. Xu and G.-Y. Tang. Vectorization path planning for autonomous mobile agent in unknown environment. *Neural Computing and Applications*, 23(7-8):2129, #dec# 2013. ISSN 1433-3058. doi: 10.1007/s00521-012-1163-3.

G.-Z. Yang, J. Bellingham, P. E. Dupont, P. Fischer, L. Floridi, R. Full, N. Jacobstein, V. Kumar, M. McNutt, R. Merrifield, B. J. Nelson, B. Scassellati, M. Taddeo, R. Taylor, M. Veloso, Z. L. Wang, and R. Wood. The grand challenges of science robotics. *Science Robotics*, 3(14), 2018. doi: 10.1126/scirobotics.aar7650.

Y. Zhu, T. Zhang, J. Song, and X. Li. A new bug-type navigation algorithm considering practical implementation issues for mobile robots. In *Proc. IEEE Int. Conf. Robotics and Biomimetics*, pages 531–536, Dec. 2010. doi: 10.1109/ROBIO.2010.5723382.

ACKNOWLEDGEMENTS

The last four years of doing my PhD have been quite the ride! I learned so much and even though I was not able to achieve everything I wanted to do initially, I did end up with a result that I'm very satisfied with. This was all possible thanks to many people that were there and supported me through the process.

I would like to thank my supervisory team, especially my promoter Guido de Croon. You were always around for a quick chat to discuss my process, and we were mostly always aligned of where we wanted to go to with the project. I would like to thank you as well for your faith when I wanted to change direction to achieve the final goal of this dissertation. I would also like to give my gratitude to my other promoters, Karl Tuyls and Bert Kappen. Even though you were not located at the TU Delft itself, you always tried to support me during my process and helped me out with formulating my ideas and the planning of my research.

I had the pleasure of working together with the big pool of talents which is the MAVlab. Christophe, your experience & endless patience have been a great help during a lot of my experiments, so I'm very grateful for that. Erik, your door was always open if I had questions of any kind and you were always willing to help me out. I would also like to thank the other (past) members of the MAVlab: Bart R, Freek, Kevin, Bart S. Andries and Roland. Also, a lot of gratitude to the PhD-students (and Postdoc) at the MAVlab. Kirk, I couldn't imagine a better person to start my PhD time with. You were always around to drag me and others to the Atmosfeer, and we discussed a lot about our crazy ideas. Thank you for being around and being a true friend. Matěj, you were always in for a coffee, drink and chat to talk about our troubles and adventures. You were also the driving factor for our yearly ski-trips, which I very much enjoyed. Diana, I loved our conversations at the coffee corner about our common problems we were facing during our PhD. Tom and Mario, I'm happy that I was your MSc supervisor and really enjoyed working together with you as fellow PhD students as well. I would also like to also thank Ewoud, Sjoerd, Lodewijk, Hann-Woei, Shushuai and Yingfu. Thank you for the "vrijmibos", coffee breaks and being around for a chat.

I very much enjoyed my time at C&S as well. Max, even though you were not part of my supervisory team, you still expressed a lot of interest into my work and well-being. I wish I had your talent for remembering names! Bertine, you are definitely the main drive of C&S, and I loved it when we could share our passions for our cats. I also enjoyed the "heidagen", BBQs and the many coffeecorner chats with many of the C&S staff: Dr. Chu, Clark, Coen, Daan, Olaf, Erik-Jan, Ferdinand, Alwin, Andries, Harold, Rene, Jacco, Menno, Marilena, Hans and Bob. This also goes the rest of the C&S PhD students, starting with my office mates. Ye and Wei, thanks for being the great desk neighbors. My back neighbor Jaime, thank you for your awesome radiating presence! You were always the main advocate for "gezelligheid" and C&S hasn't been the same since you are gone. My opposite neighbor, Sophie, I loved the coffee breaks and the ski-trips! Thanks for being

around when I needed a chat. Also to the rest of my (past) office mates: Jelmer, Sihau, Ivan, thank you for making our office a fun place to work. I want mention the other PhD students as well. Tommaso, I've loved our crazy philosophical "vrijmibo" and coffee corner talks, and your board game nights. Isabel, even though you are only around ever so often, you fill up C&S with your awesome presence! Daniel, you always managed to crack me up with laughter. Dirk, thank you for always arranging everything within C&S with your endless enthusiasm and passion. Jerom, thank you for your openhearted discussions and being a great coffee corner mate. To my other fellow (past) PhD-ers, Sherry, Yingzhi, Bo, Annemarie, Emmanuel, Dyah, Julia, Ezgi, Rolf, Julie, Junzi, Maarten, Tim, Tao, Kasper, João: thank you all for being there!

I also had the support people outside of C&S as well. The people of NoVam: Arijana, Wouter P, Wouter V, and Paul, thank you for adopting me in times that everybody else already left the Atmosfeer. A fellow karaoke enthusiast from the Tohoku summerschool, Victor, thanks for being awesome and cynical at the same time. I was also able to to stay for 3 months at the Smartlab in the University of Liverpool. I had the enjoyment of meeting James, Greg, Benjamin and Jacopo, who were able to show me another side of Liverpool that I've never seen before, so thank you so much for that. Benjamin... I hope where ever you are, you are at peace. I also would like to thank my current colleagues at Bitcraze, Arnaud, Kristoffer, Tobias and Marcus, for putting up with me trying to finish my dissertation while already started working with you in Sweden. Maarten, Patsie, Randy, Emma and Patricia, I very much enjoy our times hanging out together, on which I could always count on instant "gezelligheid". Kirsten & Linda, thank you for being true friends all the way back from IO. I can always count on you to be around, talk about our common problems and enjoy each others presence.

Last but not least I would like to thank my family and significant others. Evy and Yara, even though much of your affection is probably food oriented, you two helped me to keep my sanity during the tough moments of my PhD. My sister Stephanie, although we had our differences in the past, I'm happy that we are able to get along now. I'm proud of you in the way you have grown into the successful person that you are. My half sister Jamie, I'm happy that I got a chance to get to know you and meet up with you and hope to continue doing that in the future. Chris' parents, John, Irene and his sister Lotte's family with Guy and Petite Blandine, thank you for taking me into the family and surrounding me with love and affection. Koos, thank you for being the voice of reason in our crazy household. You are the force that makes us come together. Maria, thank you for being a great stepmom and friend. We always managed to get along in the few times that I go to the US, go shopping or eating out together, which I very much enjoy. I would like to thank my mom, which always tries to support me in the best way she can, even when I do not ask for it. I know I should try to accept your support more often, even if my ego doesn't let me. Thanks for being my mom and I love you. Dad, you taught me the skills to never give up, which definitely came in handy during my PhD. We are so much alike and I wished we were able to see each other more often. I love you and thanks for being my crazy old dad. Christian, you have been there for me for over 8 years. You have seen my ups and downs, and know me better than myself sometimes. Thank you for putting up with my quirky traits and supporting me in my decision to work abroad. I love you with all my heart.

Curriculum Vitæ

Kimberly Nancy McGuire

14-06-1989 Born in Heemstede, The Netherlands.

Education

1993-2007	Primary school: BosBouwers School, Hoofddorp (NL) Secondary School: Kaj Munk College, Hoofddorp (NL)
2007-2012	Bachelor of Science in Industrial Design Engineering Delft University of Technology, Delft (NL)
2012-2014	Master of Science in Mechanical Engineering Delft University of Technology, Delft (NL)
2015-2019	PhD in AeroSpace Engineering Delft University of Technology, Delft (NL) *Thesis:* Indoor Swarm Exploration with Pocket Drones *Promotors:* Dr. G.C.H.E. de Croon, Prof. Dr. K.P. Tuyls and Prof. Dr. H.J. Kappen

Awards

2017	PhD Symposium Aerospace Engineering Best presentation of session: Path planning, Delft (NL)
2018	PhD Symposium Aerospace Engineering Best poster award, Delft (NL)

List of Publications

9. **K. N. McGuire**, C. De Wagter K. Tuyls, H. Kappen & G.C.H.E. de Croon, *Minimal navigation solution for a swarm of tiny flying robots to explore an unknown environment*, Science Robotics (2019)[In press].

8. **K McGuire**, G.C.H.E. de Croon & K. Tuyls , *A Comparative Study of Bug Algorithms for Robot Navigation.*, Robotics and Autonomous systems **103261**, 121 (2019).

7. S. van der Helm, M. Coppola, **K.N. McGuire** & G.C.H.E. de Croon, *On-board range-based relative localization for micro air vehicles in indoor leader–follower flight*, Autonomous Robots **1-27**, March (2019).

6. G. J. J. van Dalen, **K.N. McGuire**. & G.C.H.E. de Croon, *Visual Homing for Micro Aerial Vehicles Using Scene Familiarity*, Unmanned Systems **119-130**, 06-08 (2018).

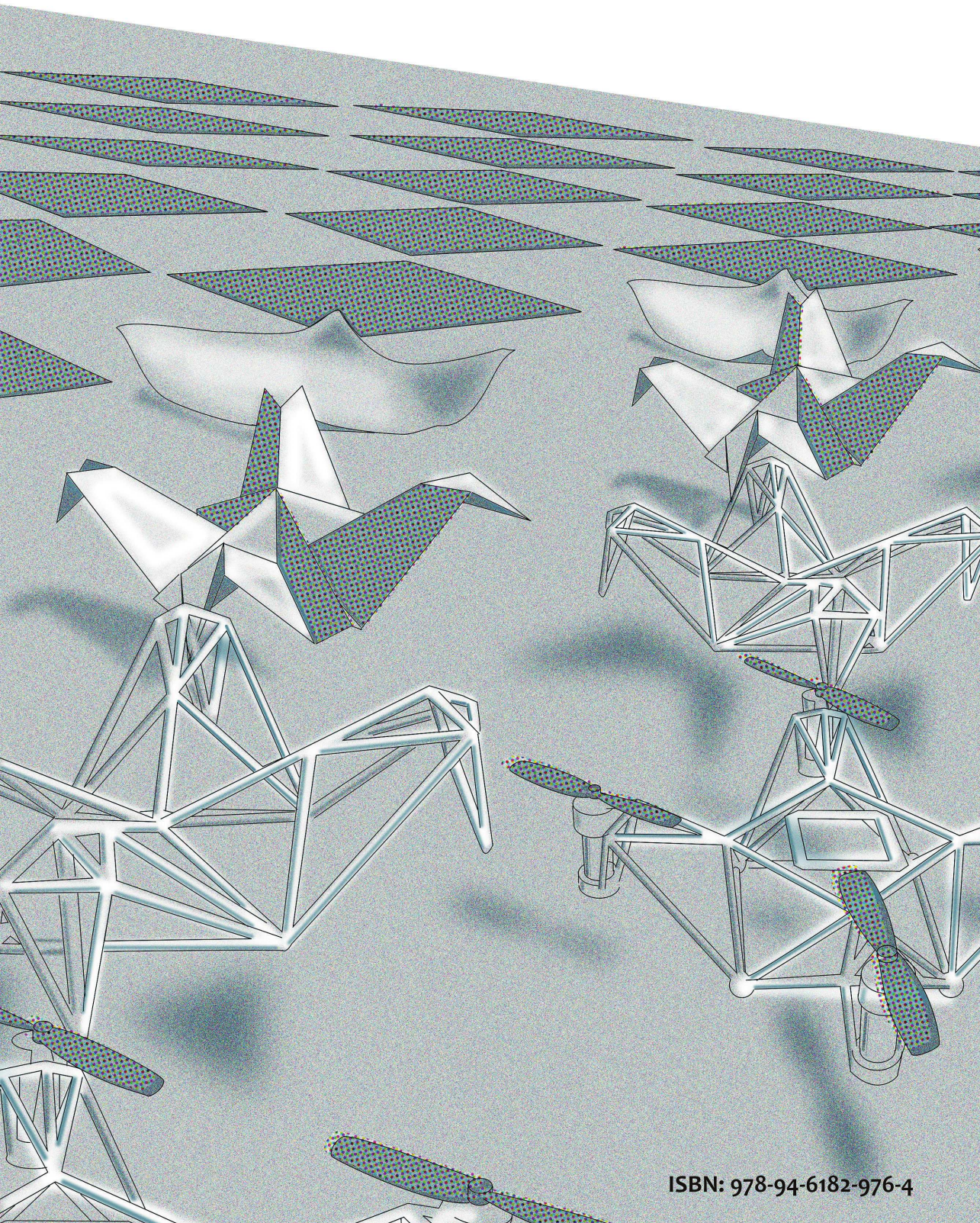
5. M. Coppola, **K.N. McGuire**, K.Y.W. Scheper & G.C.H.E. de Croon, *On-board communication-based relative localization for collision avoidance in Micro Air Vehicle teams*, Autonomous Robots **1787–1805**, 42-8 (2018).

4. **K.N. McGuire**, M. Coppola, C. De Wagter & G.C.H.E. de Croon, *Towards Autonomous Navigation of Multiple Pocket-Drones in Real-World Environments*, IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) **244-249** (2017).

3. **K.N. McGuire**, G.C.H.E. de Croon, C De Wagter, K. Tuyls & H. Kappen, *Efficient Optical Flow and Stereo Vision for Velocity Estimation and Obstacle Avoidance on an Autonomous Pocket Drone*, IEEE Robotics and Automation Letters **1070 - 1076**, 2 (2017).

2. G. J. J. van Dalen, **K.N. McGuire**. & G.C.H.E. de Croon, *Visual Homing for Micro Aerial Vehicles using Scene Familiarity*, International Micro Air Vehicle Competition and Conference (IMAV) **307-313**, (2016).

1. **K.N. McGuire**, G.C.H.E. de Croon, C. De Wagter, B. Remes, K. Tuyls & H. Kappen *Local histogram matching for efficient optical flow computation applied to velocity estimation on pocket drones*, IEEE International Conference on Robotics and Automation (ICRA) **3255-3260**, (2016).



ISBN: 978-94-6182-976-4

Propositions

accompanying the dissertation

INDOOR SWARM EXPLORATION WITH POCKET DRONES

by

Kimberly Nancy MCGUIRE

1. Implementing Edge-FS on a stereo-camera enabled pocket drone, will facilitate more exploration-based capabilities. However, the sensor's versatility comes at the price of precision of both the measured velocity and obstacle field. (this thesis)
2. Bug Algorithms-based navigation strategies are very promising for indoor exploration with extremely limited platforms, if adapted to the real-world properties of the robots. (this thesis)
3. For indoor exploration of multiple pocket drones, it is more important for the individual pocket drone to fly reliably and avoid obstacles, than to avoid each other. (this thesis)
4. When designing full on-board solutions for autonomous exploration of limited pocket drones, it is crucial to balance the desired capabilities with energy necessary to execute these capabilities. (this thesis)
5. In robotics, complex strategies to tackle a task based on physical and mathematical modeling are appreciated more than simple strategies, while the latter are more challenging to design.
6. In order to design robots that are able to operate in the real world, it is necessary for the designer to combine multiple proven strategies and components, and make them work as a whole. However, proper system integration is undervalued within the robotics community.
7. Popular media have demonized the applications for swarms of tiny autonomous MAVs. Although it is important to realize the potential dangers, it also makes the public blind for the potential good they can do in society.
8. It will be difficult to recognize once machine intelligence has surpassed us, because it might not be the kind of intelligence that we expect.
9. The medical condition of blind-sight shows that the subconsciousness of our brain handles much more than we want to believe.
10. Learning a new language with limited linguistic capacity, one will sacrifice previously learned languages for the sake of fluency of the new.

These propositions are regarded as opposable and defendable, and have been approved as such by the promotors dr. G.C.H.E. de Croon, Prof. dr. K.P. Tuyls and Prof. dr. H.J. Kappen

Stellingen

behorende bij het proefschrift

INDOOR SWARM EXPLORATION WITH POCKET DRONES

door

Kimberly Nancy MCGUIRE

1. Het implementeren van Edge-FS op een met stereocamera uitgeruste pocket drone opent meer mogelijkheden voor verkenningstaken. Echter, de veelzijdigheid van de sensor gaat ten koste van metingen van zowel snelheid als obstakelveld. (deze thesis)
2. Navigatiestrategieën gebaseerd op bug-algoritmes zijn zeer veelbelovend voor verkenning binnenshuis met extreem beperkte platformen wanneer ze aangepast worden aan eigenschappen van robots in de echte wereld. (deze thesis)
3. Voor verkenning binnenshuis met meerdere pocket drones is het meer van belang dat individuele drones betrouwbaar kunnen vliegen en obstakels kunnen ontwijken, dan dat ze elkaar kunnen ontwijken. (deze thesis)
4. Wanneer men volledige zelfstandig werkende oplossingen ontwikkelt voor autonome navigatie van beperkte pocket drones, is het van cruciaal belang om de gewenste functionaliteit in balans te houden met het energiegebruik voor het bewerkstelligen van deze functionaliteit. (deze thesis)
5. In robotica wordt meer waarde gehecht aan complexe strategieën, gebaseerd op fysische en mathematische modellen, dan eenvoudige strategieën om taken uit te kunnen voeren, terwijl deze laatste oplossingen uitdagender zijn om te bedenken.
6. Om robots te ontwerpen die kunnen opereren in de echte wereld is het van belang dat de ontwerper meerdere bewezen strategieën en componenten combineert en samen laat werken. Dergelijke systeemintegratie is echter ondergewaardeerd in de roboticagemeenschap.
7. Populaire media hebben toepassingen voor zwermen kleine, autonome MAV's gedemoniseerd. Hoewel het van belang is bewust te zijn van potentiële gevaren, maakt dit het algemene publiek blind voor de mogelijke baten voor de gemeenschap.
8. Het zal moeilijk zijn om waar te nemen wanneer kunstmatige intelligentie ons voorbijgestreefd is, daar het een vorm van intelligentie zou kunnen zijn die we niet verwachten.
9. De medische conditie "blindzien" maakt duidelijk dat het onderbewustzijn van ons brein meer taken afhandelt dan we willen toegeven.
10. Wanneer men met een beperkt linguïstisch vermogen een nieuwe taal leert, zal men eerder geleerde talen moeten opofferen voor beheersing van de nieuwe taal.

Deze stellingen worden opponeerbaar en verdedigbaar geacht en zijn als zodanig goedgekeurd door de promotoren dr. G.C.H.E. de Croon, Prof. dr. K.P. Tuyls and Prof. dr. H.J. Kappen.