Vision-Based Sense and Avoid Algorithms for Unmanned Aerial Vehicles

THÈSE N^{O} 8043 (2017)

PRÉSENTÉE LE 27 OCTOBRE 2017
À LA FACULTÉ DE L'ENVIRONNEMENT NATUREL, ARCHITECTURAL ET CONSTRUIT
LABORATOIRE DE SYSTÈMES ET ALGORITHMES INTELLIGENTS DISTRIBUÉS
PROGRAMME DOCTORAL EN ROBOTIQUE, CONTRÔLE ET SYSTÈMES INTELLIGENTS

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

POUR L'OBTENTION DU GRADE DE DOCTEUR ÈS SCIENCES

PAR

Steven Adriaan ROELOFSEN

acceptée sur proposition du jury:

Dr J. Skaloud, président du jury
Prof. A. Martinoli, Dr D. Gillet, directeurs de thèse
Prof. D. Scaramuzza, rapporteur
Prof. N. Michael, rapporteur
Prof. C. Jones, rapporteur

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE
Suisse
2017

Acknowledgements

Although only my name appears on the front page of this thesis, this work could not be done without the help and collaboration of many people. I am grateful to the ones that helped me over the years, from their contributions to my work to the friendship we shared.
First and foremost, I would like to thank my advisors Alcherio Martinoli and Denis Gillet for giving me the opportunity to do this thesis and the unconditional support they provided over the years. I am also grateful to Honeywell for sponsoring my thesis, as well as the support and technical guidance provided over the course of the project. In particular, I would like to thank George Papageorgiou and Tomas Kabrt, our contact points woth Honeywell.
I am also thankful to all the people that contributed to my thesis. In particular, I would like to acknowledge the contributions of Duarte Dias, with whom I collaborated with me on the quadrotors, and significantly contributed to software used to fly the quadrotors or simulate them. Several people also contributed by providing me insight on my work and share their advises and experience. For that, I would like to thank Duarte Dias, Ali Marjovi, Milos Vasic, Iñiaki Navarro and William Christopher Evans. I would also thank the CVLab which was responsible of the computer vison aspect of the project. In particular, I would like to thank Artem Rozantsev with whom I had the chance to work with several times over the years.
I had the chance to supervise the projects of several talented students. I would like to thank Pierangelo Rothenbuehler, Walid Amanhoud, Algis Karpavicius, Lodovico Oldani, Quentin Kuenlin and Yao Di for their commitment and contributions.
I would like to thank the DISAL and REACT group as whole for all the wonderful moments we shared. In particular, I am thankful thank Duarte Dias, José Nuno Pereira, William Christopher Evans, Alicja Wąsik, Ali Marjovi, Iñiaki Navarro, Milos Vasic and Maria Boberg for their friendship. I will forever cherish the memories of those times where we shared a coffee, a beer, or simply a conversation in the corridor.
I would also want to thank my family for their support. Without them, I wouldn't be were I am now. In particular, I will be eternally grateful to my parents Jan and Lisa, for their unconditional support and for always believing in me. Last but not least, I would like to thank Alicja Wąsik for being an amazing partner and a truly great person, and who despite the distance, always supported me and made every day more enjoyable.

Lausanne, October 5th 2017 Steven Roelofsen

Abstract

The field of Unmanned Aerial Vehicles (UAVs), also known as drones, is rapidly growing, both in terms of size and of number of applications. Civil applications range from mapping, inspection, search and rescue, taking aerial footage, to art show, entertainment and more. Currently, most applications have a human pilot supervising or controlling the vehicles, but UAVs are expected to gain more autonomy with time.

To fly in general airspace, used by both general and commercial aviation, a high level of autonomy is required from UAVs. A core functionality required to fly in general airspace is the UAVs' ability to detect and avoid collisions with other aircraft or objects. This functionality is handled by a so called Sense And Avoid (SAA) system. From among several sensors investigated to be used for a SAA system, a vision-based sensor is seen as a good candidate for a SAA system due to its ability to detect and identify a large variety of objects, as well as being close to the human's main mean to detect aircraft and other objects. To be as general as possible, this work focuses on non-cooperative algorithms that do not take assumptions on the motion of other aircraft.

This thesis presents algorithms for a vision-based SAA system. It focuses on the relationship between sensing and avoidance, and how the limitations of one constrain the second. In particular, this thesis studies the consequences of the limited Field Of View (FOV) of a camera sensor on the collision avoidance algorithms.

Given the assumptions above, the sensing and tracking of other UAVs is performed using cameras with fish-eye lenses that have a large enough FOV for the collision avoidance algorithms to guarantee to be collision-free. The detection of other UAVs is performed using two methods: a marker-based or a marker-less computer vision algorithms. Using the measurements from the computer vision algorithm, the positions and velocities of neighboring UAVs are tracked using a Gaussian mixture probability hypothesis density filter. This tracking algorithm is able to track multiple UAVs while requiring little computational resources, therefore representing a suitable candidate for on-board deployment.

In this work, it is mathematically proven that the motion of an UAV has to be constrained according to the FOV of its sensor. Following that result, several collision avoidance algorithms are adapted to ensure collision-free navigation when used with a sensor with a limited FOV.

Sensory limitations such as noise, lag, limited range and FOV, and their effects on the performance of collision avoidance algorithms are studied. Experimental work using high-fidelity simulation and real robots shows that algorithms that only use position information from the sensors are overall more reliable, although less efficient (in terms of distance traveled

or trajectory smoothness) than algorithms that also use velocity estimates from the sensing system.

Key words: sense and avoid, sensing, collision avoidance, field of view

Résumé

Le domaine des Véhicules Volants Autonomes (VVA, UAV en anglais), aussi connu sous le nom de drones, est en expansion rapide, à la fois en termes de taille et du nombre d'applications. Les applications civiles vont de la cartographie, de l'inspection, du secours, des prises de vues aériennes, aux spectacles ou aux loisirs. Actuellement, la plupart des applications reposent un humain qui supervise ou contrôle le véhicule, mais il est attendu que les VVAs deviennent plus autonomes au cours du temps.

Pour voler dans l'espace aérien général, utilisé à la fois par l'aviation civile et de tourisme, un haut niveau d'autonomie des VAAS est nécessaire. Une fonctionnalité centrale nécessaire pour voler dans l'espace aérien général est la capacité des VAAs de détecter et d'éviter les collisions avec d'autre véhicules volants ou objets. Cette fonctionnalité serait réalisée par un système de perception et évitement (SAA en anglais). Plusieurs capteurs sont étudiés pour être utilisés dans un système SAA. Un capteur basé sur la vision est vu comme un bon candidat non seulement à cause de sa capacité de détecter et identifier une large variété d'objets, mais aussi car elle est proche du moyen utilisé par les humains pour détecter des avions ou objets volants. Pour rester aussi général que possible, ce travail se concentre sur des algorithmes non-coopératifs qui n'ont pas assomptions à propos du mouvement d'autres véhicules volants.

Cette thèse présente des algorithmes pour un système SAA basé sur la vision. Elle est centrée sur le lien entre la perception et l'évitement, et comment les limitations de l'un contraignent l'autre. En particulier, cette thèse étudie les conséquences du champ de vision (FOV en anglais) limité d'une caméra sur l'algorithme d'évitement de collision.

Dans ce travail, la perception et le pistage d'autres VVA sont réalisés en utilisant des caméras avec des objectifs de type fisheye qui ont un FOV assez grand pour que l'algorithme d'évitement de collision puisse garantir l'absence de sans collision : La détection d'autres VAAs est réalisée avec deux méthodes : une est basée sur l'utilisation de marqueurs, l'autre est sans marqueurs. En utilisant les mesures de l'algorithme de vision par ordinateur, la position et la vitesse de VAAs voisins sont estimées en utilisant un filtre "Gaussian mixture probability hypothesis density". Cet algorithme de pistage est capable de suivre de multiple VAAs tout en nécessitant peu de puissance de calcul, donc représentant un bon candidat pour être utilisé à bord d'un robot.

Dans ce travail, il est prouvé mathématiquement que le mouvement d'un VAA doit être contraint selon the FOV du capteur. Suivant ce résultat, plusieurs algorithmes d'évitement de collision ont été adaptés pour assurer une navigation sans collision quand ils sont utilisés

avec un capteur avec un FOV limité.
Les limitations sensorielles tels que le bruit, le décalage, la portée et le FOV limités, et leurs effets sur la performance d'algorithmes d'évitement de collision sont étudiées. Un travail expérimental en utilisant un simulateur de haute fidélité et avec de vrais robots montre que les algorithmes qui n'utilisent que la position obtenue d'un capteur sont en général plus fiables bien que moins moins efficient (en distance parcourue ou en douceur de trajectoire) que des algorithmes qui utilisent aussi l'estimation de vitesse du système de perception.

Mots clefs : capter et éviter, perception, évitement de collision, champ de vision

Contents

Introduction Part I

1 Introduction

UNMANNED Aerial Vehicles (UAVs) have seen an increased interest over the past years, both in academic and industrial fields. UAVs are now used in many applications such as mapping [1], inspection [2], [3], search and rescue [4], [5], environmental monitoring [6], [7], surveillance [8], art shows [9], entertainment [10], and more. Additionally, more applications are expected to appear, such as delivery of goods [11] and medical supply [12], or agriculture [13]. As a result, the UAVs are getting more and more sophisticated, in particular regarding their autonomy. UAVs started as being purely remote controlled, with a human always in the loop controlling the thrust or pitch directly. Nowadays, most UAVs (including those for entertainment) are not only able to fly steadily, but also, for example, to navigate between waypoints with failure detection and automatic fallback procedures; the human supervises the UAV more than controlling it. However, in most cases the UAV strictly follows a predefined plan or takes direct command from a human. But not all plans go as expected, and sometimes it is too late to recover from a failure. This is the case with aircraft colliding. A fully autonomous UAV is thus not possible without a fully autonomous Sense and Avoid (SAA) system.

1.1 Unmanned Aerial Vehicle and Robots

An UAV (also known as drone) is an aircraft which flies without a human pilot. The UAVs can be of various types, including the popular fixed-wing and quadrotor types. Their weights varies from a few grams in the case of toys, to several tons in the case of military applications, with most civil applications using a UAV with a weight around a kilogram.

The fixed-wing aircraft is the traditional type of UAV used in civil and military applications. In particular, these vehicles are used in applications where flight time is more valuable than maneuverability or hovering, such as mapping or environmental monitoring. Even if the size of fixed-wing UAVs ranges greatly (the wingspan ranging from less than a meter to several tens of meters), the fixed wing shape is usually composed of a central core with a wing on each side and a rotor at the rear (although other types of propulsion are possible). A fixed-wing UAV is

a) b)

Figure 1.1 – Examples of UAVs. Left: The Ebee, a fixed wing type UAV from senseFly. Right: The FOX-C8, a coaxial quadrotor UAV from Onyxstar[1].

shown in Figure 1.1a.

Recently, quadrotors (or quadcopter) and other variants got popular as a cheap, easy to control and able of hovering aircraft, and now represent the main type of UAV available. Quadrotors are used in applications where high maneuverability, hovering and vertical takeoff and landing are required, such as inspection. The classical quadrotor is composed of four actuated propellers in a square configuration and facing upwards. Those four rotors give the thrust necessary to provide the lift of the vehicle. Other configurations are possible, such as more rotors, rotors with controllable orientations and more. An example of a quadrotor type UAV is shown in Figure 1.1b.

Current research [14] now merges the quadrotor with a fixed-wing to obtain an aircraft that is more efficient at flying while keeping the aptitude of hovering and vertical takeoff. Furthermore, other types of aircraft are possible, such as the blimp [15], [16] or the helicopter [17], [18].

Because of the great variety of types of UAVs, this work does not leverage detailed dynamical models of specific aircraft, but rather uses general kinematic models representative of the two main classes of UAVs mentioned above, airplanes and quadrotors. It is assumed that there is a low-level controller that translates the general velocity commands from the algorithms into some command compatible with the aircraft dynamics. Furthermore, when the concepts explained are general enough to be applied to mobile robots, the term “robot” will be used instead of UAV.

[1] a) Image from the senseFly website (CC BY-NC-ND 3.0): https://www.sensefly.com/about/news-press.html
b) Image from Wikipedia (CC BY-SA 4.0): https://en.wikipedia.org/wiki/Unmanned_aerial_vehicle

1.2 Sense and Avoid Systems

With manned aircraft, the pilot is in charge of detecting surrounding traffic and obstacles, and avoid any potential risk of collision. For staying safe, a UAV has to follow the regulations of the airspace it is flying in, such as Visual Flight Rules in general airspace. For UAVs, the Sense And Avoid (SAA) system aims to perform this task. A SAA system is divided in two main sub-systems, and which are usually considered separately in the literature. In [19], the authors present a review on the different possible solutions for a SAA system.

1.2.1 Sensing

Several sensing modalities are possible to detect potential threats (e.g., other UAVs, human-piloted aircraft, birds) surrounding an UAV, such as radars, laser scanners, cameras, acoustic sensors or transponders. Independently of the chosen sensing method, a SAA system has to meet some requirements. Based on general airspace regulations, a SAA system is expected to have a range of three statute miles (a bit less than 5 km), a field of view of 220° horizontally and 30° vertically, and to be able to fly in day and night conditions, as long as there is three statute miles of visibility [20]. This section provides a review of sensors that have been investigated to be used in the SAA system presented in Chapter 4.

The transponder relies on communication to share some states of the aircraft, usually the position. It might also use the communication channel to agree on some escape maneuver, such as the Traffic Collision Avoidance System (TCAS). A common transponder method is to share the GNSS position over some shared local wireless communication channel. Another technology is the Automatic Dependent Surveillance – Broadcast (ADS-B) system, where all aircraft broadcast their identification, position velocity and intent to other aircraft. Another transponder system is the FLARM [21]. Although the FLARM is designed to be used by gliders with human pilots, the transponder messages could be used by a UAV to identify gliders. Those solutions provide excellent accuracy and reliability, provided that everything in the airspace uses the same transponder system. As this cannot be guaranteed, the SAA needs to be equipped up by at least one non-communicative sensor to get the required level of safety [22].

Radar technology is a commonly used sensor in aerospace applications. A radar system can estimate the range, velocity and sometimes bearing of an object by emitting radio waves to the object and analyzing the radio signal that bounces back. Radars are usually bulky and heavy, and only a few radars are lightweight enough to be used in UAVs applications. A radar solution is presented in [23] with a relatively lightweight radar (230 g), but the field of view is too small for SAA applications. Furthermore, the size and weight becomes prohibitive for ranges required for SAA [24].

LIDARs (for Light Imaging, Detection And Ranging) is also a sensor considered to be used with UAVs, in particular for obstacle avoidance [25] and SAA [26]. A LIDAR uses a laser to estimate the distance to some surface. Laser scanning is used to obtain a map of the environment

instead of only a single range measurement in some direction. LIDARs can provide very accurate angular and range measurements, but are usually heavier than other technologies, and their refresh rates can be slow [24] (in particular for 3D scans). LIDARs also suffer from a small FOV and a high cost [19].

Acoustic-based sensing uses a set of microphones to detect the noise produced by other aircraft [27]. It is possible to localize the other aircraft by comparing the time shift of the sound between the microphones. Although this method is passive, low weight and omnidirectional, it requires that the other aircraft to be noisy, and that the host UAV is silent enough to be able to distinguish the sound of the other aircraft. Furthermore, the range of this method is usually smaller than the range of vision or LIDAR, and the method is dependent on weather conditions [19].

Vision-based sensing is lightweight (cameras can be made very small), able to detect a large set of potential threats and possibly identify them. Vision also has several drawbacks such as being dependent on environmental conditions (it does not work at night or in foggy conditions), computationally expensive, and does give a poor distance estimation. Another drawback is the high resolution required in order to have a large FOV. In [28], the authors state that at least 5 cameras with a resolution of 4872 by 3248 pixels are required to achieve a range and FOV described in [20] (but this thesis will show that more are required to cover the FOV needed to guarantee safe navigation). Nonetheless, vision is a commonly investigated sensor to be used in a SAA system because it is the most alike the senses used by humans to detect and avoid collisions, and as a consequence vision-based systems have the potential to achieve similar performance as a human pilot [24] (which might be required by governmental agencies).

1.2.2 Avoidance

In SAA systems, the avoidance has two distinctive phases. First, when a potential collision is detected, the UAV will attempt to generate a trajectory that safely and efficiently avoids the collision. This phase, called self-separation, is performed when the UAVs are far from the collision point, and requires long range sensors to be effective. If the self-separation phase fails, an aggressive collision avoidance maneuver is attempted. Because this maneuver is performed at a close range, the maneuver has to be quickly computed, and relies on sensors with short range. In this section, a brief overview of the state of avoidance algorithms for a SAA system is given. More in depth literature review of collision avoidance algorithms will be reported in Chapter 9.

The distinction between a deliberative self-separation and a reactive collision avoidance algorithm is specific to the field of SAA. Although self-separation for SAA is a topic that is researched upon, the work mostly focuses on the requirements needed in order for UAVs to safely carry out such maneuver [29]–[31]. From a broader perspective, the state of the art on trajectory generation does include work on UAVs [32], [33], but focuses on static and cluttered environment (i.e., navigating between buildings and vegetation), which differs significantly

from the sparse but dynamic environment encountered in self-separation. There seem to be a disparity between the research on the requirements of the SAA to fly in common airspace, and the research on trajectory generation for UAV. This difference might be an outcome of the fact that current sensing technology does not provide enough range for SAA self-separation, but is enough for a UAV to navigate between buildings.

If self-separation fails, a collision avoidance algorithm attempts an escape maneuver that is not necessarily efficient, but quickly brings the UAV out of danger. The decision time does not allow to compute complex trajectories, and reactive methods are preferred. The literature for collision avoidance algorithms is rich, and several collision avoidance paradigms have been developed over time. A review of different collision avoidance methods for SAA is given in [34].

1.3 Scope of the Thesis

Developing the necessary algorithms for a vision-based sense and avoid system targeting unmanned aerial vehicles is the scope of this thesis. Most works clearly separate the two problems of sensing and avoidance, without regard on how one affects the other. In this work, we aim to understand the interaction between the sensing and the collision avoidance components and what constraints the sensing imposes on the collision avoidance (and vice versa). In particular, this thesis focuses on the case of limited FOV sensing and how it affects collision avoidance. Solutions are proposed to mitigate the risk of navigating with a limited FOV.

The thesis is structured as follows: First, the work is introduced in Part I. In Chapter 1, the state of the art of UAVs and SAA is reviewed, and the scope of the thesis as well as the contributions to the state of the art are summarized. Chapter 2 presents the setup and tools used in this thesis. Part II and Part III present the SAA algorithms developed during this thesis, and the two parts are separated following the common split between tracking and avoidance in SAA. Part II concerns the tracking algorithms used in this work. It starts with the related work and background in Chapter 4. The contributions for visual tracking using different setups and computer vision algorithms are presented in Chapters 5, 6 and 7. Part III presents our contributions in collision avoidance. An introduction to Part III and related work is reviewed in Chapter 9. The general study of collision avoidance with limited FOV sensing is presented in Chapter 10. Several algorithms that guarantee collision avoidance under limited FOV sensing are given in Chapters 11, 12, 13, 14 and 15. Finally, Chapter 16 presents a study of the effect of sensing limitations on different collision avoidance algorithms. Part IV concludes this thesis.

The separations between the parts follow the structure of the SAA pipeline used in our work. Part I presents our work on algorithms and systems that are outside of the scope of a SAA system. Part II presents our contributions to the tracking aspect of SAA and Part III presents our contributions to collision avoidance algorithms. This structure is illustrated in Figure 1.2. For experiments that run the full SAA system all elements of Figure 1.2 are used.

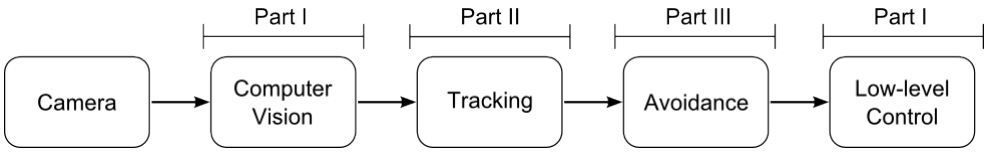


Figure 1.2 – The structure of this thesis.

1.3.1 Contributions

1. **Experimental Setup -** We developed an experimental setup composed of a real-world robotic platform and a simulation platform. The robotic platform is composed of a quadrotor equipped with a computational unit and one or more cameras. The robotic platform is able to run on-board all the software necessary for a SAA system. The simulation framework we developed reproduces our real-world setup. We calibrated the simulations using real-world data acquired from our robotic platform. The real-world robotic platform and simulation framework share the same interface with the SAA algorithms, which allows to seamlessly use the SAA software we developed both in simulation and with the real-world robotic platform with virtually no modification necessary.

2. **Visual tracking of quadrotors -** We developed several visual tracking algorithms for SAA system. Our implementations use the Probability Hypothesis Density (PHD) filter, an efficient multi-target tracking algorithm, to track other aircraft using vision. The proposed solution uses cameras with fish-eye lenses to obtain the FOV required by the collision avoidance algorithms. Two different computer-vision algorithms are used to detect other aircraft on the image. Some are marker-based tracking solutions using both a single camera and multiple cameras setups and are presented in Chapters 5 and 7, respectively. A marker-less solution is also presented in Chapter 6. The visual tracking algorithms are designed to run in real time on hardware commonly found on small UAVs. Furthermore, the tracking algorithms have been validated with real experiments, combined with some collision avoidance algorithms. Relevant publications for this contribution are:

   - S. Roelofsen, D. Gillet, and A. Martinoli, “Reciprocal collision avoidance for quadrotors using on-board visual detection”, in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2015, pp. 4810–4817.
   - K. R. Sapkota, S. Roelofsen, A. Rozantsev, V. Lepetit, D. Gillet, P. Fua, and A. Martinoli, “Vision-based unmanned aerial vehicle detection and tracking for sense and avoid systems”, in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2016, pp. 1556–1561.

3. **Collision avoidance with limited field of view sensing -** This work is the first that explicitly studies how to perform collision avoidance with FOV-limited sensing. To ground our work, we demonstrate in Chapter 10 the necessary and sufficient conditions on the

motion to guarantee that no collision can be caused by FOV-limited sensing. In the same chapter, we also present scenarios that we designed to show that ignorance of the FOV constraints can lead to a collision. This work also provides guidelines to modify any collision avoidance to make it compatible with a sensory system that has a limited FOV. We developed five different collision avoidance algorithms that guarantee collision avoidance under limited FOV sensing. Several collision avoidance paradigms are investigated, including Rule Based (RB) in Chapter 11, Potential Fields (PF) in Chapters 12, 13 and 14, and Velocity Obstacles (VO) in Chapter 15. The algorithms are validated through simulation and real-world experiments, leveraging the visual tracking algorithms. Relevant publications for this contribution are:

- S. Roelofsen, A. Martinoli, and D. Gillet, "Distributed deconfliction algorithm for unmanned aerial vehicles with limited range and field of view sensors", in *American Control Conference*, 2015, pp. 4356–4361.
- S. Roelofsen, A. Martinoli, and D. Gillet, "3D collision avoidance algorithm for unmanned aerial vehicles with limited field of view constraints", in *Conference on Decision and Control*, 2016, pp. 2555–2560.
- S. Roelofsen, D. Gillet, and A. Martinoli, "Collision avoidance with limited field of view sensing: a velocity obstacle approach", in *IEEE International Conference on Robotics and Automation*, 2017, pp. 1922–1927.

4. **Comparison of collision avoidance algorithms -** Leveraging our experimental set-up in simulation and physical reality, we compared the performance and safety of different collision avoidance algorithms for SAA systems. In particular, we investigated the relation between sensor limitations and their effects on different collision avoidance algorithms. Six different algorithms have been evaluated in simulation with four different types of sensor limitations (noise, lag, range and FOV). We show that PF algorithms are usually more robust to sensor imperfections, at the expense of efficiency. The relevant publications for this part are:

   - S. Roelofsen, D. Gillet, and A. Martinoli, "A comparative study of collision avoidance algorithms for unmanned aerial vehicles: performance and robustness to noise", in *International Symposium on Distributed Autonomous Robotic Systems; Springer Proceedings in Advanced Robotics (2018)*, 2016.
   - S. Roelofsen, D. Gillet, and A. Martinoli, "The effects of sensory limitations on avoidance: a comparative study of collision avoidance algorithms for unmanned aerial vehicles", in *Autonomous Robots, to be revised and resubmitted*, 2018.

The work of this thesis took place within a collaborative project in which the CVLab and Honeywell participated. We would like to give credit to the CVLab for the computer vision algorithm described in Section 2.2.3. Furthermore, the content presented in Chapter 6 was made possible with the close collaboration of the CVLab. We also want to give credit to our

industrial partner Honeywell for the financial support and technical guidance throughout the whole project.

Furthermore, this work has seen contributions from other DISAL collaborators, and although we were directly involved with all the work of this thesis, we would like to give credit to Duarte Dias for his contributions to the hardware and software tools presented in Chapter 2.

Summary

This chapter gave an introduction about UAVs and SAA systems and provided a short review of the state of the art on these subjects. Furthermore, the scope and objectives of this thesis have been presented. The contributions were also highlighted, namea visual tracking method for SAA systems, a novel approach to collision avoidance that includes the constraints of limited field of view, and the comparison of different collision avoidance algorithms, in particular in terms of robustness to sensor limitations.

2 Tools and Setup

VALIDATION, might it be done through simulation or real-world experiments, is a fundamental part of robotics. Without validation, it is impossible to know whether the assumptions taken while designing the theory and algorithms are correct or not. In this work, our algorithms are validated using both simulations and physical setups. In this chapter, we present the corresponding tools we developed or assembled in a customized fashion. Furthermore, all the algorithms that were implemented but that are not part of the core contributions of this thesis (e.g., computer vision algorithms or low-level feedback loops) are also presented in this chapter.

2.1 Hardware

To validate the algorithms developed during this thesis with real-world experiments, a suitable hardware platform was needed. Our physical setup is built around two main components, the quadrotors which serve as our UAV platform, and the experiment room equipped with a Motion Capture System (MCS) to perform controlled experiments within.

2.1.1 Unmanned Aerial Vehicle platform

The robotic platform used in this work is the quadrotor, a small flying platform that is propelled with four vertical facing rotors arranged in a square. Our UAV fleet consists of AscTec Hummingbirds, a 500g quadrotor controlled by a proprietary flight controller. To extend the capability of our quadrotors, they are equipped with a computation module in the form of a Gumstix Overo AirStorm, which is an ARM-based single-core processor running at 800MHz with 512 MB of RAM, and has on-board WiFi. The computation module runs a lightweight Linux distribution called Linaro.

When used with the color blob detection computer vision algorithm (c.f. Section 2.2.3), the quadrotors where equipped with a 15cm large, painted Styrofoam sphere in which both the

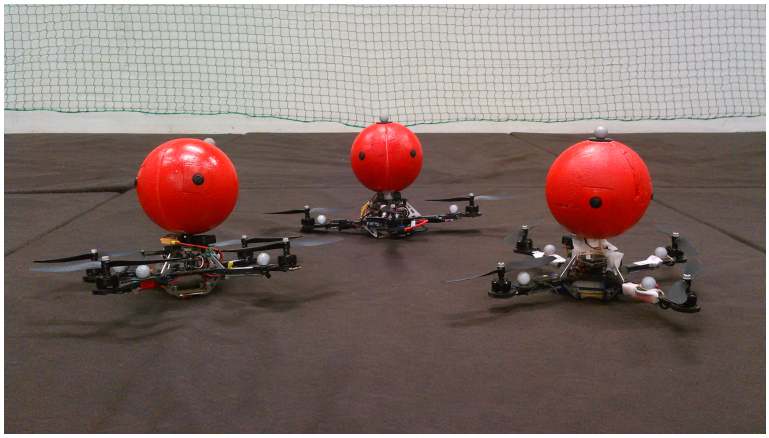


Figure 2.1 – Three quadrotors equipped with the red markers. The black disks are the cameras. The gray spheres are the markers used by the MCS.

Table 2.1 – Characteristics of the experiment rooms.

Name	Length	Width	Height	Number of cameras
Maillefer	8 m	6 m	3 m	20
Jordils	11 m	9 m	5 m	28

computation unit and the cameras were embedded. An example of those markers is shown in Figure 2.1. The cameras used in this work are off the shelf, webcam-type USB cameras equipped with fish-eye lenses.

2.1.2 Experimental Facility

The experiments were performed in an indoor environment. Figure 2.2 illustrates the setup used for our real-world experiments. The experimental infrastructure is composed of a room protected with nets on the walls and the ceiling, and 10cm thick mattresses on the floor. Over the course of this project, two different rooms were used, called Maillefer and Jordils (the names are from their respective locations). Their characteristics are given in Table 2.1. The rooms are equipped with a MCS, which is able to track the pose objects with millimetric accuracy at 100 Hz. The MCS relies on a set of cameras anchored to a structure at the ceiling level of the flying arena to track retro-reflective sphere of 25 mm diameter. The data from the cameras are sent to a Windows computer, which estimates the pose of the objects from the markers attached to the object. The pose estimates obtained are then sent to the quadrotors through WiFi. Another computer running Linux is connected to the WiFi router to send commands to the quadrotors and record ground-truth data from the MCS.

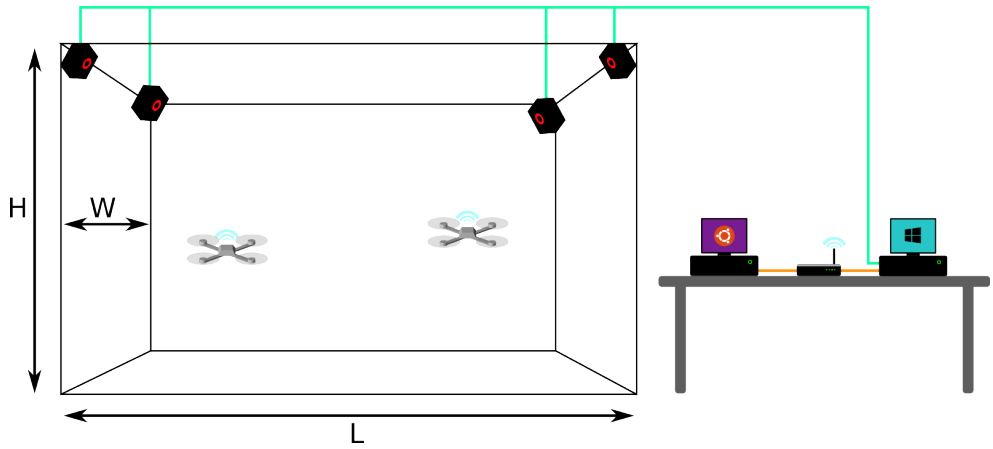


Figure 2.2 – Illustration of the experimental facility used for some of our experiments. Illustration not to scale. The black cubes with a red circles represent the cameras, which track the position of the quadrotors.

2.2 Software

To ease the implementation and validation of the various algorithms presented in this work, several software frameworks are used. In particular, the Robotic Operating System (ROS) and the high-fidelity robotic simulator Webots are used. Both frameworks are presented in this section.

2.2.1 ROS

The ROS framework is a collection of libraries and software that ease the development of robotic applications. The main concept behind ROS is that the functionalities in a robotic application can be broken down in separate processes running concurrently. With ROS, each process is implemented in its own program, called node. Each node can communicate to other nodes through the internet protocol suite, usually UDP/IP. In this work, the nodes communicate through a publisher/subscriber architecture, where nodes can subscribe to topics and receive data from other nodes that publish on that same topic. The nodes used in this work are the following:

Main Control The main task of this node is to translate the high-level control commands (such as position and velocity) into low-level commands for the AscTec MAV framework node. It is also responsible to perform critical maneuvers such as takeoff or landing.

User Control To send commands to the quadrotor (e.g., takeoff, move the quadrotor to a certain position, etc.), a ROS nodes waits for string commands sent over a UDP socket. The string commands are then published on a topic available to all other ROS nodes. All ROS nodes get all commands and only react to those that are concerned with (e.g., the ”takeoff´´ command only affects the main control node, and not the avoidance node).

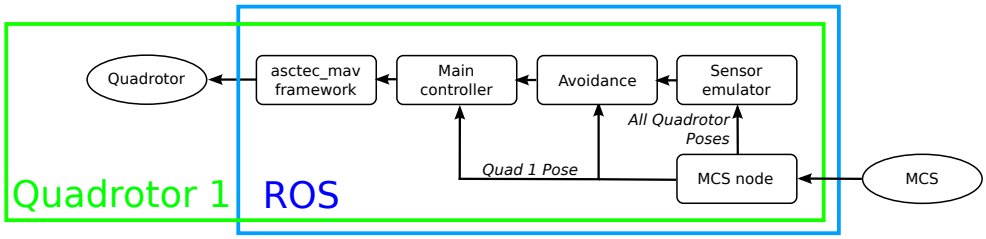


Figure 2.3 – Interaction between the different ROS nodes on a real quadrotors used in the experiments.

Avoidance The avoidance node contains the implementation of the different collision avoidance algorithms. It takes as input the detections from the PHD filter (or the emulation thereof) and the position of the host quadrotor provided by the MCS, and computes the high-level command required for the quadrotor to safely get to its destination.

USB Camera Driver The camera driver acquires the image stream from the camera and makes it available to other nodes. To reduce the computational load, it is also able to perform color segmentation, extracting the positions and sizes of patches with predetermined color.

PHD Filter The PHD filter node uses the detections provided by the computer vision algorithm (including the camera driver) and estimates the location of other quadrotors using one of the PHD filters described in Part II.

Tracking Emulator The PHD filter can be emulated using the ground truth data. Sensor limitations and inaccuracies can be added to the ground truth data to make the obtained tracks more realistic. This node replaces the USB camera driver and the PHD filter.

MCS Node The ground truth is provided by the MCS. The main MCS software runs on a dedicated computer, and this node implements the SDK to communicate with the MCS software. It provides the pose of all objects tracked by the MCS. In the Webots simulation environment, this node is replaced by a Webots controller.

AscTec MAV Framework This node is the link between other algorithms and the quadrotor's low level control board. It takes as input the desired command, in the form of desired roll and pitch angle, yaw rate, and motor trust. It also makes the information from the quadrotor (i.e., IMU data, battery level, etc.) available to the other nodes. In the Webots simulation environment, the functionalities of this node are provided by the Webots robot controller. An implementation of this node can be downloaded from a public repository [42].

2.2.2 Control

The main control node is responsible of translating the desired commands, such as velocity and yaw rate, in commands understandable by the quadrotor. The Hummingbird quadrotor

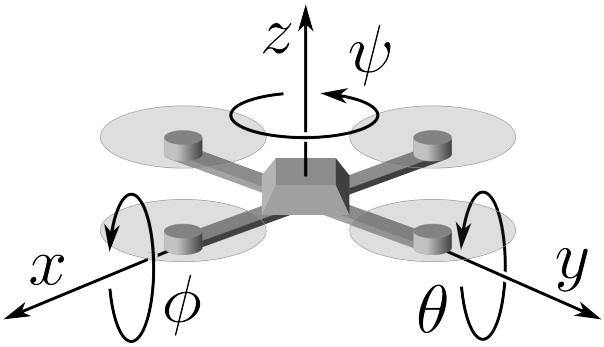


Figure 2.4 – Illustration of the angles on the quadrotor.

is already performing the attitude control, leaving to the main control node to compute the desired thrust and attitude. The inputs are the thrust, roll and pitch angles, and the yaw rate (the angles are illustrated in Figure 2.4). The yaw rate from the collision avoidance algorithm is directly fed to the quadrotor. The main control node has several control modes and can easily switch between them. In this section, the position and velocity control algorithm is presented.

First, the control acceleration $\mathbf{a}_c$, needed to obtain the desired velocity, is computed:

$$\mathbf{a}_c = K_p(\mathbf{p}_t - \mathbf{p}) + K_{I_z} I_z \mathbf{z} + K_v(\mathbf{v}_t - \mathbf{v}) + mg\mathbf{z} + \mathbf{a}_t, \tag{2.1}$$

with $mg\mathbf{z}$ the gravity compensation, $\mathbf{p}_t$, $\mathbf{v}_t$ and $\mathbf{a}_t$ quadrotor's target position, velocity and acceleration, respectively. I_z is the integral term for the height and serves as weight compensation. K_p, K_v and K_{I_z} gains. If exclusively velocity control is desired, K_p is set to 0. The integral term is only updated if the control in position is turned on:

$$I_z[k+1] = \begin{cases} I_z[k] + (\mathbf{p}_t - \mathbf{p})\Delta t & \text{if } K_p \neq 0 \\ I_z[k] & \text{otherwise.} \end{cases} \tag{2.2}$$

The control acceleration is used to compute the thrust and the roll and pitch angles. To be consistent with the quadrotor commands, the control acceleration first needs to be aligned with the quadrotor's yaw angle:

$$\mathbf{a}_b = R_\psi^T \mathbf{a}_c. \tag{2.3}$$

The thrust is then given by the vertical component of control acceleration $a_{b,z}$. The thrust is also compensated for the orientation of the quadrotor to ensure that its real vertical acceleration corresponds to the control acceleration:

$$u = \frac{K_T a_{b,z}}{\cos(\phi)\cos(\theta)}, \tag{2.4}$$

with K_T the acceleration to thrust factor. The roll ϕ_c and pitch θ_c commands are given by the

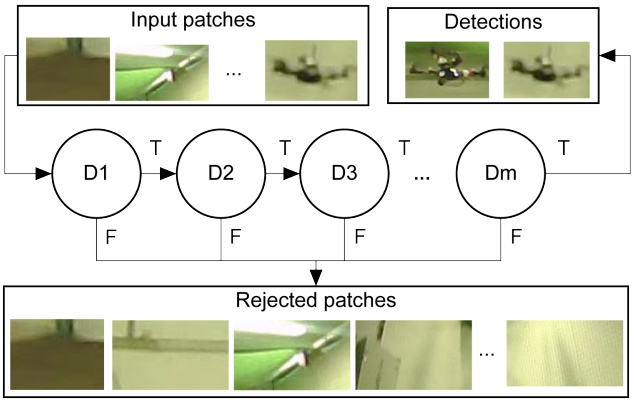


Figure 2.5 – Illustration of the cascade of detectors. Courtesy of Artem Rozantsev.

control acceleration direction

$$\phi_c = \arctan(a_{b,z}, a_{b,y}) \tag{2.5}$$

$$\theta_c = \arctan(a_{b,z}, a_{b,x}). \tag{2.6}$$

The thrust T, roll angle ϕ_c, pitch angle θ_c and yaw rate are then sent to the quadrotor using the *asctec_hl_interface* package [42].

2.2.3 Computer Vision

In this section, the two different computer vision algorithms used in this work are presented.

Quadrotor Shape Detection

Computer vision is one of the proposed sensing solutions for SAA systems. Over the years, several algorithms have been developed to detect aircraft in the sky, based on different concepts. A popular solution is based on morphological characteristics in the image [43][44][45][46], which allows to detect other aircraft that appear as only a couple pixels on the image. The drawback of this solution is that it does not allow to recognize other aircraft and generates a large number of false positives.

Another approach is to learn the appearance of aircraft and detect them on the image. This is the approach developed by the Computer Vision Laboratory (CVLab) at the School of Computer and Communication Sciences, EPFL. In this section, we will briefly describe the algorithm, although the main credit goes to CVLab, and in particular to Krishna Raj Sapkota, for the implementation.

The algorithm is composed of two parts. First, a cascade of detectors aims at detecting the

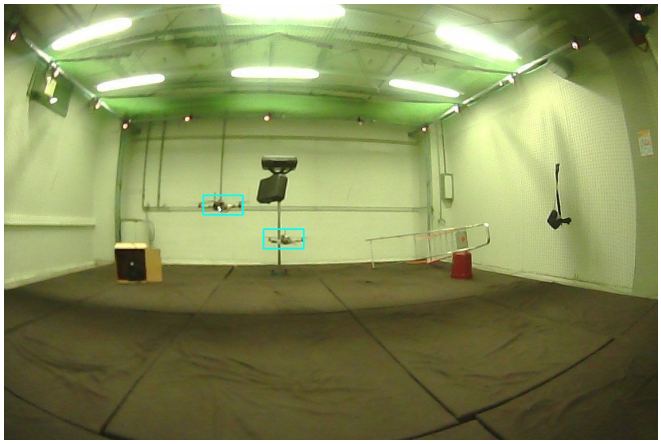


Figure 2.6 – Example of the detections obtained using the shape detection computer vision algorithm with the visual tracker.

shape of a quadrotor in the image. This is performed using a classical cascade of detectors implementation: The image is subdivided into patches, and each patch is compared against a data-set trained using the histogram of oriented gradients. Each detector returns a binary value, depending whether the image patch matches the sufficiently the features trained in that given detector. If the patch matches a large enough number of detectors, it is detected as being a quadrotor. The detectors are separated in batches put in cascade to remove the unlikely patches at an early stage, lowering the computational requirements. Because the cascade of detectors reports several matching patches in the neighborhood of an object (instead of a single report per object), a non-maximum suppression step is applied to the output of the cascade of detectors. The non-maximum suppression step removes a matching patch if it has another matching patch with a higher matching score (i.e., matches more detectors) in its neighbourhood. Second, a visual tracker uses the detections from the cascade of detectors and generates time coherent tracks by learning on-the-fly the shape of the aircraft. If a region of the image consistently generates detections, a track is generated, and the visual tracker learns the appearance of the quadrotor from the image patches where a quadrotor was found (provided by the cascade of detectors). At the next image, the visual tracker will find the image patch that matches best the learned appearance of the track. Both algorithm provide the position on the image and the width of the patches that are the most likely to contain a quadrotor.

Color Blob Detection

There exist numerous computer vision algorithms to detect objects, but only a few of them have been adapted for the detection of a generic aircraft and it was not possible to find a single implementation with a low enough computational complexity to be run on board of small quadrotors. In order to relax the computational requirements of an on-board solution based on computer vision and at the same time make the effort relevant for future enhanced

embedded computational capabilities, we decided to simplify the detection task by adding an easy-to-detect marker on the vehicles in the form of a colored sphere. While the detection task is now substantially simplified, all the other difficulties are maintained: estimation of the position and size based on monocular vision, sensing affected by noise (clutter and misdetections). Color segmentation is used to detect the 15 cm in diameter colored marker (see Figure 2.1). The algorithm returns the position of the color blob's center as well as the number of pixels it covers. The color segmentation is performed on the raw image obtained by the camera. No distortion correction or smoothing is performed in order to save CPU time. As all markers are identical, there is no possible identification of the quadrotors. Marker-based detection of other UAV are used in [47].

2.2.4 Simulation

In the case where the validation does not require to be very faithful to reality, a point-mass microscopic simulation is implemented in Matlab. Unless mentioned, all simulations are performed with Matlab and use perfect kinematic equation and noiseless sensing. This allows us to clearly study the behavior of a specific algorithm without any interference from external (and typically uncontrollable) factors.

When the performance of the SAA system has to be evaluated and the effect of unmodeled dynamics or sensing noise can not be ignored, the simulations are performed using Webots [48], a realistic sub-microscopic simulator. A screenshot of the simulation environment can be seen in Figure 2.7. In Webots, each object is composed of one or several modules. Each module usually implements a simple characteristic of the object, such as its physical shape or a sensor that the robot is endowed with. Each robot in Webots has its own controller, also organized in a dedicated module. The simulator is able to interface with ROS, allowing it to run the same code as the one implemented on our quadrotors. Some ROS nodes used on the quadrotors are replaced with nodes that can interface with the Webots API. First, the MCS and the associated MCS ROS node are replaced by one implemented at the supervisor level of Webots, resulting in a node which gathers the poses of the simulated quadrotors and publishes them on the ROS topics related to the MCS. In this case, there is only one Webots supervisor node for the whole simulation. Second, the AscTec MAV framework ROS node functionality is taken over by the controller module of a Webots robot, which takes care of translating the commands provided by the main control ROS node into motor speeds used by the Webots simulator. To automatize the process of performing several simulations with different scenarios, two more ROS nodes were added. The resulting dataflow for two quadrotors is shown in Figure 2.8. The two additional nodes are:

Flight Manager This node emits high level commands like takeoff or operate.

Scenario Manager This node synchronizes the operations between the quadrotors, for instance for making sure that the quadrotors takeoff and start navigating at the same time. It essentially communicates with the Flight Manager of each quadrotor.

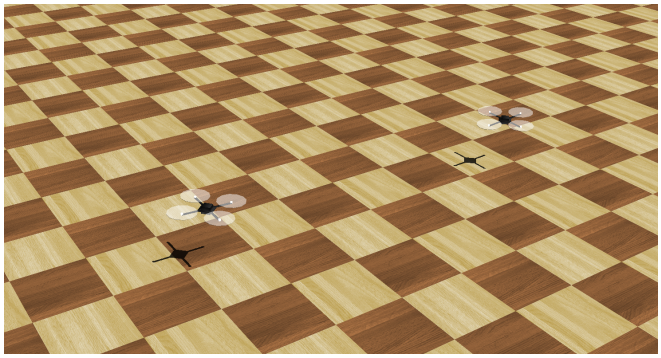


Figure 2.7 – Rendering of a Webots simulation with two quadrotors.

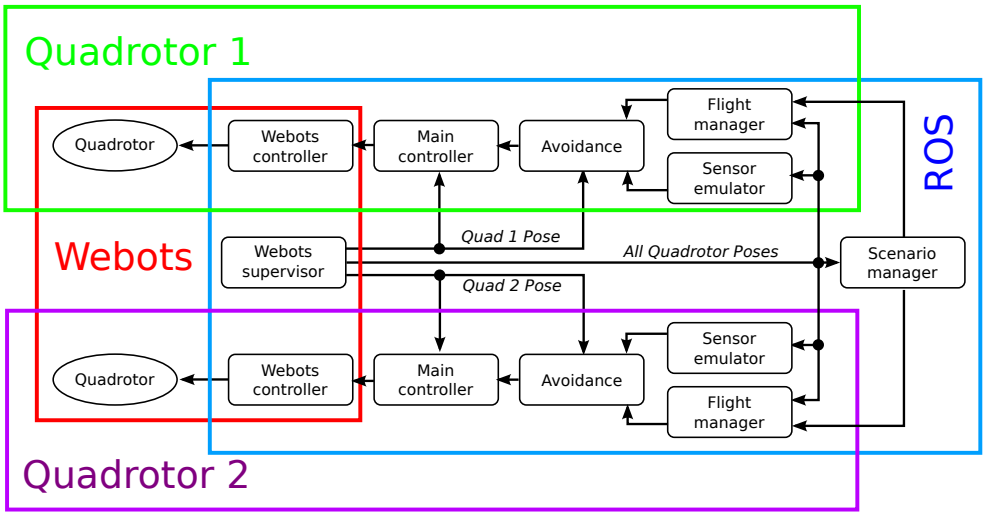


Figure 2.8 – Interaction between the different ROS nodes for two quadrotors in the simulation. The simulation framework uses the same avoidance and sensor emulation code that is implemented on the quadrotors.

To obtain simulation results comparable to reality, the simulation was calibrated. Physical parameters such as weight are directly measured on the quadrotor. Because the internal structure of the low-level control (responsible to control the motor speeds to bring the quadrotor to the correct attitude) is unknown to us, its parameters needed also to be estimated. The unknown control scheme is assumed to be a PID controller and to have the structure described by Equations 2.7, 2.8 and 2.9:

$$M_x = K_a(\phi_t - \phi) - K_{da}\Omega_\phi \tag{2.7}$$

$$M_y = K_a(\theta_t - \theta) - K_{da}\Omega_\theta \tag{2.8}$$

$$M_z = K_{dy}(\Omega_{\psi,t} - \Omega_\psi) - K_{ddy}\frac{d\Omega_\psi}{dt} - K_{Iy}\int \Omega_\psi, \tag{2.9}$$

with ϕ, θ and ψ being roll, pitch and yaw respectively, and Ω_ϕ, Ω_θ and Ω_ψ their respective angular rates. ϕ_t and θ_t are target roll and pitch respectively. $\Omega_{\psi,t}$ is a target yaw rate. M_x, M_y

and M_z and the desired torques to apply along the x, y and z axis in order to get the quadrotor to the desired state. K_a, K_{da}, K_{dy}, K_{ddy} and K_{Iy} are the control parameters that need to be calibrated.

The five control parameters given in Equations 2.7 to 2.9 are optimized to minimize the difference between trajectories obtained from simulation and real experiments. The optimization was carried out using Particle Swarm Optimization (PSO), where each particle is a vector containing the five control parameters. Their fitnesses are defined as the RMS difference between the average trajectory obtained through real experiments and the average trajectory generated in simulation. The real experiment data set is composed of a quadrotor trying to follow a predefined trajectory eleven times. The simulation trajectory is obtained by simulating the quadrotor following the same predefined trajectory. The simulations are performed four times to average out the timing variability of the ROS framework (i.e., messages do not arrive with a deterministic timing).

While performing the experiments, it was observed that the quadrotors were affected by the airflow they were generating, resulting in a not perfectly steady flight. To replicate this disturbance in simulation, we added a first order Gauss-Markov process on the thrust and torques in roll, pitch and yaw. The disturbances for each time step are described as:

$$T_d[n+1] = 0.995T_d[n] + 0.005(W(0,2)) \tag{2.10}$$

$$M_{x,d}[n+1] = 0.995M_{x,d}[n] + 0.005(W(0,0.12)) \tag{2.11}$$

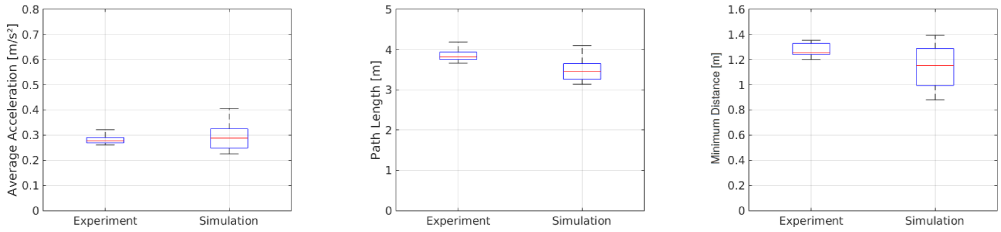
$$M_{y,d}[n+1] = 0.995M_{y,d}[n] + 0.005(W(0,0.12)) \tag{2.12}$$

$$M_{z,d}[n+1] = 0.995M_{z,d}[n] + 0.005(W(0,0.02)), \tag{2.13}$$

with $W(\mu,\sigma)$ a Gaussian process of mean μ and standard deviation σ. The time step is 10 *ms*. The parameters have been set empirically so that the average acceleration between experimental and simulated data is similar.

To validate the calibration of the simulator, trajectories of two quadrotors performing collision avoidance maneuvers obtained with simulation and real experiments are compared. Comparing trajectories of collision avoidance maneuvers is preferable over comparing trajectories that are obtained from predefined trajectories as the former ensures that the simulator behaves faithfully for the maneuvers of interest (i.e. collision avoidance maneuvers). The collision avoidance algorithm used for the comparison is the one described in Chapter 13. The sensor for this experiment was emulated using the method described in Chapter 16. The sensor was emulated with no lag, a FOV of 200°, a range of 3 *m* and a Gaussian noise level of 0.01 *m*. The scenario used was a head-on configuration where the quadrotors start 3.2 *m* apart. For both simulation and real experiments, 20 runs have been gathered. Three metrics are used to evaluate the trajectories. The first metric is the average acceleration during the collision avoidance:

$$\mathbf{a} = \frac{1}{n_f - n_i} \sum_{n_i<n<n_f} \frac{\|\mathbf{p}[n+1] - 2\mathbf{p}[n] + \mathbf{p}[n-1]\|}{\Delta t^2}, \tag{2.14}$$



(a) Average acceleration (b) Path length (c) Minimum distance

Figure 2.9 – Comparison between simulation and real experiments.

with $\mathbf{p}[n]$ the position of a quadrotor at time step n, n_i the time step when the quadrotor leaves its initial position and n_f the time step the quadrotor arrives at its destination. Δt is the time between two time steps. The second metric is the path length, i.e. the length of a quadrotor's trajectory from its initial point to the quadrotor's destination.

$$\mathbf{d} = \sum_{n_i < n < n_f} \|\mathbf{p}[n] - \mathbf{p}[n-1]\|. \tag{2.15}$$

Finally, the minimum distance between the two quadrotors during the collision avoidance maneuver is also recorded:

$$\mathbf{e} = \min_{n_i < n < n_f} \|\mathbf{p_i}[n] - \mathbf{p_j}[n].\| \tag{2.16}$$

It can be seen in Figure 2.9 that for all the metrics considered (acceleration, path length, and minimum distance between the quadrotors) the simulated data assume smaller values than those gathered through experiments. This is probably due to the airflow generated by a quadrotor that tends to push away the other one, an effect that is, only roughly approximated in simulation (i.e. the approximation does not simulate that the airflow push the quadrotors away from each other).

2.3 Discussion

No experimental setup is perfect, and this one is no exception. Although this setup has satisfying performance, reliability and fidelity to carry out our experiments, there is room for improvement.

On the hardware side, some issues were encountered. First, the on-board computational unit was barely powerful enough to run all the algorithms on its single core. To fully leverage the powerful concept of nodes in ROS (i.e., separating the different processes), multiple cores would be an advantage. With multiple cores, slow processes (e.g., computer vision) could run along with processes that need tight timing (e.g., position control) with as little unwanted inter-process interaction as possible. Second, by using a closed-source platform, it was not possible to fully understand the dynamics of the quadrotors, and the use of gray-box

estimation was needed to calibrate the simulation tools. As a consequence, this eventually in a larger simulation-to-reality gap than what could have been achieved if all the software components of the quadrotors were designed by us.

On the software side, the simulation of the quadrotor could be improved by also simulating the airflow interactions between two quadrotors, as this airflow has significant effect on the quadrotors at close range. Furthermore, the low-level control could be improved, either by using better control gains for the roll and pitch command angles (computed with, for example, LQR), base the control scheme on the differential flatness of the quadrotor [49], or by directly commanding the motors' speeds, which allows to use more advanced techniques such as model predictive control [50], [51], which would allow for more aggressive and precise control of the quadrotor's state (i.e., position or velocity).

Summary

This chapter presents the tools and setups used to validate the work of this thesis. From an hardware perspective, the chosen UAV platform is an off-the-shelf quadrotor with an ARM-based computer unit attached to it. The poses of the quadrotors are tracked in a controlled environment using a motion capture system. Our algorithms leverage the Robotic Operating System (ROS) framework to communicate between each other. Our simulation environment is leveraging Webots, a sub-microscopic, physic-based simulator that also interfaces with the ROS framework, allowing the same algorithms to run both in simulation and on the real quadrotors. Some algorithms, such as the computer vision algorithms used, are also presented in this chapter.

Sensing & Tracking Part II

3 Introduction to Part II

THIS part will focus on the tracking algorithms used to estimate the position and velocities of other UAVs. The tracking algorithms presented in Chapters 5 to 7 have several characteristics in common. First, they use one or two-cameras with a large FOV (i.e., fish-eye lenses). Second, they all rely on one of the two computer vision algorithms presented in Chapter 2, which are the color-blob detection and the shape-based detection. Both computer vision algorithms return similar types of measurements, which are the position on the image of the detected object of interest (horizontal and vertical), and some measurement of the apparent size of the detection (either the number of pixels for the color-blob detection or the pixel width for the shape-based detection). Third, all tracking algorithms rely on the same core Gaussian Mixture Probability Hypothesis Density (GM-PHD) theory, although each algorithm was adapted to the characteristics of the camera setup and the computer vision algorithm. The similarities and differences between the three tracking algorithms presented in Chapters 5 to 7 are summarized in Table 3.1.

This part starts by an introductory chapter on the topic of multi-target tracking, and will provide a technical explanation about the GM-PHD filter. Chapter 4 will also present the different implementations of the GM-PHD filter, based on the Extended Kalman Filter (EKF) or the Unscented Kalman Filter (UKF).

Chapter 5 presents our first implementation of a tracking algorithm. It is based on a two cameras setup and uses the color-blob detection algorithm. It is implemented as an EKF-GM-PHD filter.

Chapter 6 presents our efforts to implement a tracking algorithm that uses the shape-based detection algorithm as computer vision algorithm. It uses a single camera. It is implemented as an EKF-GM-PHD filter.

Chapter 7 presents our implementation of a tracking algorithm that uses a single camera with a FOV large enough to satisfy the requirements given in Chapter 10. It uses measurements from the color-blob detection algorithm and is implemented using an UKF-GM-PHD filter.

Table 3.1 – Characteristics of the different sensing systems.

Chapter	Vision algorithm	# of cameras	Camera FOV	Update method
5	Color blob detection	Two	$< 180°$	EKF
6	Shape-based detection	One	$< 180°$	EKF
7	Color blob detection	One	$220°$	UKF

4 Tracking: Related work

In this work, the focus is not on the visual detection of aircraft, but rather on how those detections can be fused into track estimates, filtering out clutter and noise. A good tracking of other aircraft is essential to enable collision avoidance algorithms to produce safe paths for the concerned vehicles. Because multiple objects that have to be tracked can be present in the scene simultaneously, the tracking algorithm has to not only estimate the state of each object reported by the sensor, but also the correct number of objects. Furthermore, the sensor might report a measurement where there is no object (called a false positive, or clutter), or might not a report an existing object (called a false negative, or a misdetection). The difficulty of multi-target tracking resides in the data association, that is, associating the right measurement to the right track. Furthermore, the multi-target tracking algorithm should also manage tracks, generating new tracks from non-associated measurements and delete tracks that do not represent an object.

Several multi-target tracking algorithms have been designed over the years and can be separated in four major categories: Nearest Neighbour (NN)[52], Joint Probabilistic Data Association (JPDA) [53], Multiple Hypothesis Tracking (MHT) [54] and Random Finite Set (RFS) trackers [55].

The data association in NN-based algorithms is solved by associating a measurement to its closest track. This inexpensive solution is effective as long as the objects tracked are well separated and there is almost no clutter or miss detection. The result can be improved by computing the data association that minimizes the overall distance to track. This method improves the results compared to a simple NN approach, but still fails in presence of clutter. Furthermore, this approach does not provide a way to manage tracks, and has to be implemented using some heuristic.

The JPDA approach improves over the NN-based approach by computing the joint probability of associating measurements and tracks [56]. This allows for a better rejection of clutter. Computing all possible data associations is a combinatorial problem, which grows super-exponentially with the number of tracks. Any reasonable implementation of a JPDA is an

approximation, such as only computing the probabilities of the most likely associations [52]. The JPDA main drawbacks are that it assumes a known number of tracks and performs poorly when the tracks are close together. Furthermore, like the NN approach, it does not provide a way to manage tracks.

The MHT approach propagates in time several sets of tracks, each one having its own data association history (hypotheses). The probability of each set of tracks is computed and the best one is provided as output of the algorithm. This method provides with means to manage tracks. Although this method provides a rigorous framework for multi-target tracking and is able to potentially solve any multi-target tracking problem, the algorithm is quickly intractable due to the super-exponential increase of the number of hypotheses, because each hypothesis generates as many new hypotheses as there are possible data associations. Most MHT algorithms are thus approximations in order to become tractable. In [57], each data association hypothesis is contained in a particle, and at each new measurement the particles are updated with random data associations. This method was found to be too computationally demanding for our application.

The RFS theory [55] extends the notion of probability density function to finite sets. This theory allows not only to rigorously describe a multi-target Bayes filter, but also handles track management by design. As a result, parameters relating to track management have a more physical meaning compared to, for example, MHT-based algorithms. However, similarly to MHT-based algorithms, RFS based algorithms are intractable if not approximated. But the RFS theory does provide the tools to generate correct approximations based on the assumptions about the problem.

4.1 Metrics for Multi-Target Tracking

The data association problem in multi-target tracking does not only concern tracks with measurements, but is also present when assessing the performance of the tracking algorithm. Indeed, the tracking algorithm might not only have errors in estimating the states of each track, but also the number of tracks. As a result, it is not possible to simply use the root mean square error as metric of the accuracy of the tracking algorithm. Consider the two scenarios in Figure 4.1, in which it is not clear which is the best solution. Does one favor accurate estimation of the states of each target, or is the estimation of the number of targets more important? Arguably, one can see how each one can be better, depending on the requirements. Furthermore, it might not be clear which estimated track belongs to which ground truth, in particular if the number of tracks differ from the real number of objects tracked.

A solution comes in the form of Optimal Sub-Pattern Assignment (OSPA) [58]. The concept behind OSPA is to find the best possible assignment between the estimates and the ground truth, and add some predefined distance value for every estimate (or ground truth) that is not assigned. The OSPA is then defined as the average of the assigned and the non-assigned distances combined. Consider $d^{(c)}(\mathbf{x}, \mathbf{y})$ the distance between some state vectors $\mathbf{x}$ and $\mathbf{y}$ with a

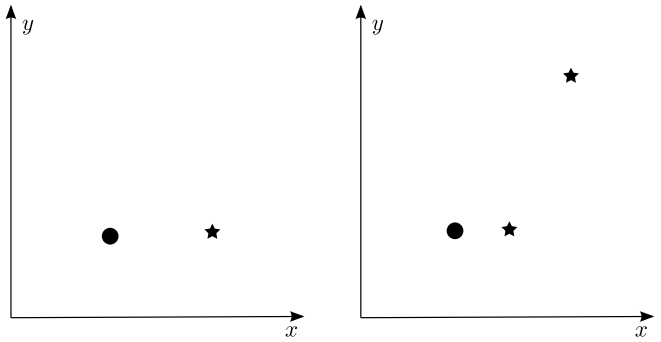


Figure 4.1 – Two possible output of a multi-target tracking algorithm. The black circles represent the ground-truth, the starts show the estimated tracks.

cutoff distance $c > 0$. The cutoff distance defines the maximum distance at which an estimate and a ground truth can be associated. If an estimate and a ground truth are separated by a distance larger than c, then $d^{(c)}(\mathbf{x},\mathbf{y}) = c$. The OSPA $d_p^{(c)}(\mathbf{X},\mathbf{Y})$ between the set of m estimated tracks $\mathbf{X} = \{\mathbf{x}_1,...,\mathbf{x}_m\}$ and the set of n ground truth tracks $\mathbf{Y} = \{\mathbf{y}_1,...,\mathbf{y}_n\}$ is given by:

$$d_p^{(c)}(\mathbf{X},\mathbf{Y}) = \left(\frac{1}{n}\left(\min_{\pi\in\Pi_n}\sum_{i=1}^{m} d^{(c)}(\mathbf{x},\mathbf{y}_{\pi(i)})^p + c^p(n-m)\right)\right)^{1/p}, \tag{4.1}$$

if $m \leq n$, and $d_p^{(c)}(\mathbf{X},\mathbf{Y}) = d_p^{(c)}(\mathbf{Y},\mathbf{X})$ if $m > n$. Π_n is the set of all possible permutations of the vectors of $\mathbf{Y}$. p is a parameter that defines the type of metric, and is set to 2 in this work (similar to the norm). Note that the maximum value of the OSPA is defined by its cutoff distance c.

4.2 Random Finite Set Theory

RFS theory is a source of new and effective tracking techniques with a wide range of possible applications, such as in automotive [59][60], tracking of people [61] or traffic [62]. A brief introduction on RFS will now be presented. For more in depth theoretical explanations, please refer to [55] and [63].

RFS extends the notion of random variables to sets. Those sets come from considering that at time k there are $M(k)$ targets present in the scene. Their states $\mathbf{x}_{k,1},...,\mathbf{x}_{k,M(k)} \in \mathscr{X}$ become $\mathbf{x}_{k+1,1},...,\mathbf{x}_{k+1,M(k+1)}$ at time $k+1$, with $\mathscr{X}$ being the state space. At time k the sensor generates $N(k)$ measurements $\mathbf{z}_{k,1},...,\mathbf{z}_{k,N(k)} \in \mathscr{Z}$ with $\mathscr{Z}$ the measurement space. The number of targets and measurements may not be the same due to misdetections and clutter. The random sets are defined as:

$$\mathbf{X}_k = \{\mathbf{x}_{k,1},...,\mathbf{x}_{k,M(k)}\} \in \mathscr{F}(\mathscr{X}) \tag{4.2}$$

$$\mathbf{Z}_k = \{\mathbf{z}_{k,1},...,\mathbf{z}_{k,N(k)}\} \in \mathscr{F}(\mathscr{Z}), \tag{4.3}$$

with $\mathscr{F}(\mathscr{X})$ and $\mathscr{F}(\mathscr{Z})$ being the collections of all finite subsets of $\mathscr{X}$ and $\mathscr{Z}$, respectively. One can write the targets' time evolution using RFS. If $\mathbf{X}_{k-1}$ is the multi-target state at time $k-1$, each single target $\mathbf{x}_{k-1} \in \mathbf{X}_{k-1}$ either continues to exist at time k with the survival probability $p_{S,k}(\mathbf{x}_{k-1})$ or disappears with probability $1 - p_{S,k}(\mathbf{x}_{k-1})$. If the target survives, it will transition from state $\mathbf{x}_{k-1}$ to $\mathbf{x}_k$ with probability density $f_{k|k-1}(\mathbf{x}_k|\mathbf{x}_{k-1})$. Using those definitions, one can build the set $\mathbf{S}_{k|k-1}(\mathbf{x}_{k-1})$ that returns $\{\mathbf{x}_k\}$ with probability $p_{S,k}(\mathbf{x}_{k-1})$ and $\varnothing$ with probability $1 - p_{S,k}(\mathbf{x}_{k-1})$. Targets can also appear between time $k-1$ and k, which is described with the RFS Γ_k. In this work we ignore targets birth from other targets. The time evolution of $\mathbf{X}_k$ is given by:

$$\mathbf{X}_k = \left[\bigcup_{\zeta \in \mathbf{X}_{k-1}} \mathbf{S}_{k|k-1}(\zeta) \right] \cup \Gamma_k. \tag{4.4}$$

In a similar way, a RFS model can be written for the measurement set. At time k, a target $\mathbf{x}_k \in \mathbf{X}_k$ can either generate a measurement $\mathbf{z}_k$ with probability $p_{D,k}(\mathbf{x}_k)$ or generate none with probability $1 - p_{D,k}(\mathbf{x}_k)$. Each target $\mathbf{x}_k$ generates a RFS $\Theta_k(\mathbf{x}_k)$ that either returns a measurement $\{\mathbf{z}_k\}$ according to $g_k(\mathbf{z}_k|\mathbf{x}_k)$ if it is detected or $\varnothing$ if not. The sensor might also be corrupted with clutter, which is described by the RFS $\mathbf{K}_k$. The RFS $\mathbf{Z}_k$ given $\mathbf{X}_k$ is:

$$\mathbf{Z}_k = \left[\bigcup_{\mathbf{x} \in \mathbf{X}_k} \Theta_k(\mathbf{x}) \right] \cup \mathbf{K}_k. \tag{4.5}$$

Similar to the single target case, the multi-target transition and observation randomness can be captured with a multi-target transition density $f_{k|k-1}(\mathbf{X}_k|\mathbf{X}_{k-1})$ and a multi-target likelihood $g_k(\mathbf{Z}_k|\mathbf{X}_k)$.

If $p_k(\cdot|\mathbf{Z}_{1:k})$ denotes the multi-target posterior density then the optimal multi-target Bayes filter is given by:

$$p_{k|k-1}(\mathbf{X}_k|\mathbf{Z}_{1:k-1}) = \int f_{k|k-1}(\mathbf{X}_k|\mathbf{X})\, p_{k-1}(\mathbf{X}|\mathbf{Z}_{1:k-1}) \mu_s(d\mathbf{X}), \tag{4.6}$$

$$p_k(\mathbf{X}_k|\mathbf{Z}_{1:k}) = \frac{g_k(\mathbf{Z}_k|\mathbf{X}_k)\, p_{k|k-1}(\mathbf{X}_k|\mathbf{Z}_{1:k-1})}{\int g_k(\mathbf{Z}_k|\mathbf{X})\, p_{k|k-1}(\mathbf{X}|\mathbf{Z}_{1:k-1}) \mu_s(d\mathbf{X})}. \tag{4.7}$$

Computing Equations 4.6 and 4.7 requires the evaluation of several integrals on $\mathscr{F}(\mathscr{X})$, which is intractable. Further approximations have to be carried out, such as a first moment approximation of the probability distribution, presented in the next section.

4.3 GM-PHD Filter Implementation

To have a tractable algorithm, it is possible to propagate a first order statistical moment instead of the full multi-target posterior density. Such filter is called a Probability Hypothesis Density (PHD) filter. The first order moment of a RFS $\mathbf{X}$ with probability distribution P is a

non-negative function v on $\mathscr{X}$ called *intensity* such that for each *region* $\mathbf{S} \subseteq \mathscr{X}$,

$$\int |\mathbf{X} \cap \mathbf{S}| P(d\mathbf{X}) = \int_{\mathbf{S}} v(\mathbf{x}) d\mathbf{x}. \tag{4.8}$$

Using Equation 4.8, Equations 4.6 and 4.7 can be approximated using an intensity measure, leading to the following equations:

$$v_{k|k-1}(\mathbf{x}) = \int p_{S,k}(\zeta) f_{k|k-1}(\mathbf{x}|\zeta) v_{k-1}(\zeta) d\zeta + \gamma_k(\mathbf{x}), \tag{4.9}$$

$$v_k(\mathbf{x}) = [1 - p_{D,k}(\mathbf{x})] v_{k|k-1}(\mathbf{x}) + \sum_{\mathbf{z} \in \mathbf{Z}_k} \frac{p_{D,k}(\mathbf{x}) g_k(\mathbf{z}|\mathbf{x}) v_{k|k-1}(\mathbf{x})}{\kappa(\mathbf{z}) + \int p_{D,k}(\zeta) g_k(\mathbf{z}|\zeta) v_{k|k-1}(\zeta) d\zeta}, \tag{4.10}$$

with $\gamma_k(\mathbf{x})$ the intensity of targets appearing and $\kappa(\mathbf{z})$ the intensity of clutter in the measurements. Note that in this formulation, the notion of separate tracks for each target is lost, as they are all merged in a single state space. Instead, each region of the state space has now a certain density of targets. As a result, it is not possible to predict the motion of a target from its state history, and the prediction can be performed with the latest state estimate only. The PHD filter is enough for reactive collision avoidance that only needs the immediate state of tracked objects, but would not be enough for algorithms that generate avoidance trajectories based on the state history of tracked objects. Even if this formulation reduces the computational cost, Equations 4.9 and 4.10 do not admit a general closed form solution. By approximating the intensity as the sum of Gaussians, the Gaussian Mixture Probability Hypothesis Density (GM-PHD) allows for a closed form solution in a similar form to the Kalman filter for single target. The intensity of the GM-PHD filter is of the form:

$$v_k(\mathbf{x}) = \sum_{i=1}^{J_k} w_k^{(i)} \mathcal{N}(\mathbf{x}; \mathbf{m}_k^{(i)}, P_k^{(i)}), \tag{4.11}$$

with J_k the number of Gaussians in the intensity at time k. $\mathbf{m}_k^{(i)}$ is i Gaussian's mean (in this case a vector composed of the target's position $\mathbf{p}_k^{(i)} = [x_k^{(i)}, y_k^{(i)}, z_k^{(i)}]^T$ and velocity $\mathbf{v}_k^{(i)}$) at time k and $P_k^{(i)}$ is its covariance. $w_k^{(i)}$ is the weight associated with the Gaussian.

4.4 EKF Implementation of the GM-PHD Filter

Because the probability density function of the RFS is composed of Gaussians, it is possible to use the well known Kalman approach to update them. Because the camera model is non-linear, the update step is performed using the Extended Kalman Filter (EKF) approximation.

The GM-PHD filter has four fundamental steps. First, a prediction step is done by applying the motion model on the previous intensity. Then the intensity is updated with the newly acquired measurements. To keep the algorithm tractable, the Gaussians are merged and pruned in the third step. Finally, the targets are inferred from the intensity map during the extraction step. Each step is performed at the time one of the cameras returns a measurement. Each stage will

now be described individually as it is implemented, rather than the general approach which can be found in [63].

4.4.1 Prediction

During the prediction step, the intensity is computed as:

$$v_{k|k-1}(\mathbf{x}) = \sum_{i=1}^{J_k} p_{S,k}(\mathbf{m}_{k|k-1}^{(i)})\, w_k^{(i)} \mathcal{N}(\mathbf{x}; \mathbf{m}_{k|k-1}^{(i)}, P_{k|k-1}^{(i)}) + \gamma_k(\mathbf{x}). \tag{4.12}$$

The birth model $\gamma_k(\mathbf{x})$ is defined to cover the quadrotor's sensing region. Thus, the birth model is given as a Gaussian mixture, with the position of the Gaussians defined in the quadrotor's body frame: $\mathbf{m}_{\gamma,k}^{(i)} = [\mathbf{p}_{\gamma,k}^{(i)}, 0, 0, 0]^T$ with $\mathbf{p}_{\gamma,k}^{(i)} = R_b \mathbf{p}_{\gamma,b}^{(i)} + \mathbf{p}$. R_b is the rotation matrix to go from the body frame to the world frame. $\mathbf{p}$ is the position of the camera in the world. The predicted mean and covariance are defined as:

$$\mathbf{m}_{k|k-1}^{(i)} = F_{k-1} \mathbf{m}_{k-1}^{(i)} \tag{4.13}$$

$$P_{k|k-1}^{(i)} = Q_{k-1} + F_{k-1} P_{k-1}^{(i)} F_{k-1}^T, \tag{4.14}$$

with F_{k-1} the transition model and Q_{k-1} motion noise model. Note that Equation 4.12 differs from [63] in that the survival probability $p_{S,k}$ is not constant but is a function of the Gaussian mean. This is an approximation necessary to cope with the limited FOV of the sensors as the targets might accumulate in non-sensed areas if care is not taken.

4.4.2 Update

The update step for a set of measurements $\mathbf{Z}_k$ is given by:

$$\begin{aligned} v_k(\mathbf{x}) = &\sum_{i=1}^{J_k} (1 - p_{D,k}(\mathbf{m}_{k|k-1}^{(i)}))\, w_{k|k-1}^{(i)} \mathcal{N}(\mathbf{x}; \mathbf{m}_{k|k-1}^{(i)}, P_{k|k-1}^{(i)}) \\ &+ \sum_{\mathbf{z} \in \mathbf{Z}_k} \sum_{i=1}^{J_k} w_k^{(i)}(\mathbf{z}) \mathcal{N}(\mathbf{x}; \mathbf{m}_{k|k}^{(i)}(\mathbf{z}), P_{k|k}^{(i)}), \end{aligned} \tag{4.15}$$

with

$$w_k^{(i)}(\mathbf{z}) = \frac{p_{D,k}(\mathbf{m}_{k|k-1}^{(i)})\, w_{k|k-1}^{(i)} l_k^{(i)}(\mathbf{z})}{\kappa(z) + \sum_{i=1}^{J_k} p_{D,k}(\mathbf{m}_{k|k-1}^{(i)})\, w_{k|k-1}^{(i)} l_k^{(i)}(\mathbf{z})}, \tag{4.16}$$

$$l_k^{(i)}(\mathbf{z}) = \mathcal{N}(\mathbf{z}; H_k^{(i)} \mathbf{m}_{k|k-1}^{(i)}, R_k + H_k^{(i)} P_{k|k-1}^{(i)} [H_k^{(i)}]^T), \tag{4.17}$$

$$\mathbf{m}_{k|k}^{(i)}(\mathbf{z}) = \mathbf{m}_{k|k-1}^{(i)} + K_k^{(i)}(\mathbf{z} - H_k^{(i)} \mathbf{m}_{k|k-1}^{(i)}), \tag{4.18}$$

$$P_{k|k}^{(i)} = \left[I - K_k^{(i)} H_k^{(i)} \right] P_{k|k-1}^{(i)}, \tag{4.19}$$

$$K_k^{(i)} = P_{k|k-1}^{(i)} [H_k^{(i)}]^T (H_k^{(i)} P_{k|k-1}^{(i)} [H_k^{(i)}]^T + R_k)^{-1}. \tag{4.20}$$

with $\kappa(\mathbf{z})$ the clutter level and R_k the sensor noise covariance. $H_k^{(i)}$ is the linearized sensor model:

$$H_k^{(i)} = \left.\frac{\partial g_k(\mathbf{x})}{\partial \mathbf{x}}\right|_{\mathbf{x}=\mathbf{m}_{k|k-1}^{(i)}}. \tag{4.21}$$

4.4.3 Selection

To keep the problem tractable, the number of Gaussians needs to be limited. This involves two steps: pruning and merging. First, only the Gaussians with a weight more than a certain value are selected: $I = \{i = 1, ..., J_k | w_k^{(i)} > T\}$. The threshold T is set low enough to allow new targets to appear, but higher enough to keep the computational power required low. A commonly used value for T is 10^{-6}. Then the Gaussians are merged as follows. First the largest Gaussian is selected, that is $j = \text{argmax}_{i \in I} w_k^{(i)}$. Then all Gaussians that are close enough are selected:

$$\mathbf{L} := \left\{ i \in I | (\mathbf{m}_k^{(i)} - \mathbf{m}_k^{(j)})^T (P_k^{(i)})^{-1} (\mathbf{m}_k^{(i)} - \mathbf{m}_k^{(j)}) \leq U \right\}, \tag{4.22}$$

with U the normalized merging radius, typically set between 1 and 2. The selected Gaussians are merged into a unique Gaussian as follows:

$$\widetilde{w}_k^{(l)} = \sum_{i \in I} w_k^{(i)} \tag{4.23}$$

$$\widetilde{\mathbf{m}}_k^{(l)} = \frac{1}{\widetilde{w}_k^{(l)}} \sum_{i \in I} w_k^{(i)} \mathbf{m}_k^{(i)} \tag{4.24}$$

$$\widetilde{P}_k^{(l)} = \frac{1}{\widetilde{w}_k^{(l)}} \sum_{i \in I} w_k^{(i)} \left(P_k^{(i)} + (\widetilde{\mathbf{m}}_k^{(l)} - \mathbf{m}_k^{(i)})(\widetilde{\mathbf{m}}_k^{(l)} - \mathbf{m}_k^{(i)})^T \right). \tag{4.25}$$

4.4.4 Extraction

Extracting the target from the RFS intensity is performed by selection of the largest maximas of the intensity function. For the GM-PHD filter, this reduces to deciding on the Gaussians with weight larger than 0.5, as explained in [63].

4.5 UKF Implementation of the GM-PHD Filter

When the non-linearity of the sensor model is too severe, or the analytic derivation of the Jacobian needed for the EKF is too complex, the Unscented Kalman Filter (UKF)[64] is preferable. The UKF is an approach based on the unscented transform that approximates the non-linearity of the motion or sensor model. The UKF uses the so called sigma-points that are propagated through the function to estimate the best linear approximation at several locations instead of only a single point like for the EKF. The other advantage of the UKF over the EKF, besides a better approximation, is that the Jacobian of the function is not needed, which can

be difficult to compute depending on the function. This usually comes at the expense of more computational cost.

The UKF will now be introduced, based on [64]. The UKF is based on the unscented transform, which is done as follows: consider a random variable $\mathbf{x}$ (of dimension L) with mean $\bar{\mathbf{x}}$ and covariance $P_{\mathbf{x}}$. To calculate the statistics of $\mathbf{x}$ after going through a non-linear function $\mathbf{z} = g(\mathbf{x})$, a set of sigma points are computed:

$$X_0 = \bar{\mathbf{x}} \tag{4.26}$$

$$X_i = \bar{\mathbf{x}} + \left(\sqrt{(L+\lambda)P_{\mathbf{x}}}\right)_i \quad i = 1,...,L \tag{4.27}$$

$$X_i = \bar{\mathbf{x}} - \left(\sqrt{(L+\lambda)P_{\mathbf{x}}}\right)_i \quad i = L+1,...,2L, \tag{4.28}$$

where $\lambda = \alpha^2(L+\delta) - L$ is a primary *scaling* parameter. The *spread factor* α is usually kept small (in this work $\alpha = 0.1$), and δ is a secondary scaling parameter, which is set to $3 - L$, as advised in [65]. $\left(\sqrt{(L+\lambda)P_{\mathbf{x}}}\right)_i$ is the ith column of the matrix square root. The sigma points are then put through the non-linear function:

$$Y_i = g(X_i) \quad i = 0,...,2L. \tag{4.29}$$

Using the transformed sigma points, the mean and covariance of $\mathbf{y}$ can be approximated:

$$\bar{\mathbf{y}} \approx \sum_{i=0}^{2L} w_i^{(m)} Y_i \tag{4.30}$$

$$P_{\mathbf{y}} \approx \sum_{i=0}^{2L} w_i^{(c)} (Y_i - \bar{\mathbf{y}})(Y_i - \bar{\mathbf{y}}), \tag{4.31}$$

with the weights given by:

$$w_0^{(m)} = \frac{\lambda}{L+\lambda}, \tag{4.32}$$

$$w_0^{(c)} = \frac{\lambda}{L+\lambda} + 1 - \alpha^2 + \beta, \tag{4.33}$$

$$w_i^{(m)} = w_i^{(c)} = \frac{1}{2(L+\lambda)} \quad i = 1,...,2L, \tag{4.34}$$

with β used to incorporate prior knowledge of the distribution (e.g., $\beta = 2$ for Gaussian distributions). $w_i^{(m)}$ is the weight for the mean and $w_i^{(c)}$ is the weight for the covariance.

The UKF-GM-PHD filter implementation only differs from the EKF implementation (Section 4.4) in the update step, the other three steps (prediction, selection and extraction) being the same. This is a design choice, and a UKF prediction step also be used if the motion model was highly non-linear. The UKF update step is performed as follows: consider the sensor

model $\mathbf{y^{(i)}} = g(\mathbf{x^{(i)}})$, then the update step is given by:

$$v_k(\mathbf{x}) = \sum_{i=1}^{J_k} (1 - p_{D,k}(\mathbf{m}_{k|k-1}^{(i)}))\, w_{k|k-1}^{(i)} \mathcal{N}(\mathbf{x}; \mathbf{m}_{k|k-1}^{(i)}, P_{k|k-1}^{(i)}) + \sum_{\mathbf{z}\in\mathbf{Z}_k} \sum_{i=1}^{J_k} w_k^{(i)}(\mathbf{z}) \mathcal{N}(\mathbf{x}; \mathbf{m}_{k|k}^{(i)}(\mathbf{z}), P_{k|k}^{(i)}) \tag{4.35}$$

with

$$w_k^{(i)}(\mathbf{z}) = \frac{p_{D,k}(\mathbf{m}_{k|k-1}^{(i)})\, w_{k|k-1}^{(i)} l_k^{(i)}(\mathbf{z})}{\kappa(\mathbf{z}) + \sum_{i=1}^{J_k} p_{D,k}(\mathbf{m}_{k|k-1}^{(i)})\, w_{k|k-1}^{(i)} l_k^{(i)}(\mathbf{z})} \tag{4.36}$$

$$l_k^{(i)}(\mathbf{z}) = \mathcal{N}(\mathbf{z}, \bar{\mathbf{y}}^{(i)}, S_k^{(i)}) \tag{4.37}$$

$$\mathbf{m}_{k|k}^{(i)}(\mathbf{z}) = \mathbf{m}_{k|k-1}^{(i)} + K_k^{(i)}(\mathbf{z} - \bar{\mathbf{y}}^{(i)}) \tag{4.38}$$

$$P_{k|k}^{(i)} = P_{k|k-1}^{(i)} - K_k^{(i)}(C_k^{(i)})^T \tag{4.39}$$

$$K_k^{(i)} = C_k^{(i)}(S_k^{(i)})^{-1} \tag{4.40}$$

$$S_k^{(i)} = R_k + \sum_{j=0}^{2L} w_j^{(c)}(Y_j - \bar{\mathbf{y}})(Y_j - \bar{\mathbf{y}}) \tag{4.41}$$

$$C_k^{(i)} = \sum_{j=0}^{2L} w_j^{(c)}(X_j - \mathbf{m}_{k|k-1}^{(i)})(Y_j - \bar{\mathbf{y}}). \tag{4.42}$$

Summary

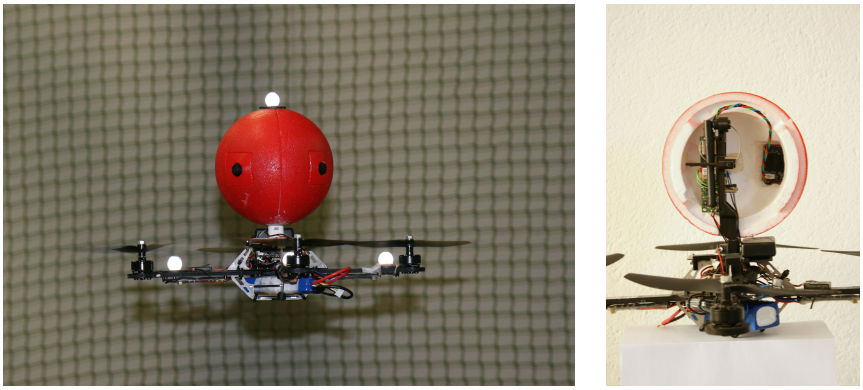
In this chapter, the topic of multi-target is introduced and existing multi-target tracking algorithms are reviewed. Then, the theory for the tracking algorithms used in this work is presented, namely the Random Finite Set and the PHD filter. Finally, two implementations of the PHD filter, based on the EKF and the UKF, are presented.

5 Marker-Based Tracking with Two Cameras

THE first iteration of the sensing part of the SAA was designed to test the first collision avoidance algorithm designed for limited FOV sensing, which is described in Chapter 13. Because the shape detection computer vision algorithm was not available at the time of designing this filter, the simpler color blob detection algorithm was used. The color blob detection algorithm also has the advantage to run reasonably well on the limited computational power available on the quadrotor. The work in this chapter was previously published in the 2015 proceedings of the International Conference on Intelligent Robots and Systems [35].

5.1 Hardware

The underlying requirements of this system is that the sensing modality is exclusively vision-based, that the FOV of the vision system is of 220°, and that all the related computation is performed on board. To achieve the FOV of 220°, two cameras with fisheye lenses of 185° FOV are used. The two cameras are positioned horizontally, pointing with an 45° angle away from the front of the quadrotor (c.f., Figure 5.2). By only using the central portion of the image, the distortion is small enough to use the standard distortion model with reasonable results. The two cameras are connected to the on-board computational unit with USB cables. As the resolution needs to be small to run on the on-board embedded computer, and the FOV of the lenses are large, only large objects can be seen by the cameras. As a result, the markers attached on the quadrotors are 15 *cm* in diameter. The marker contains most of all the additional hardware that the quadrotor carries (i.e., embedded computer, cameras with fish-eye lens). It is composed of a hollow sphere of expanded polystyrene, painted in red. Built in the polystyrene sphere are the two cameras and the computation unit. The sphere is hold internally by a thin plastic frame that also holds the computation unit. A quadrotor flying as well as the inside of a marker is shown in Figure 5.1.



a) b)

Figure 5.1 – The quadrotor equipped with a marker. a) the quadrotor flying. b) The inside of the marker. The central tower holds the computation unit. The camera is embedded inside the polystyrene sphere.

5.2 Filter Implementation

The two camera setup uses the EKF-GM-PHD filter to track other quadrotors. For each tracked quadrotor, its position and velocity are estimated; the state space of the tracks is $\mathbf{m} = [x, y, z, v_x, v_y, v_z]^T$.

5.2.1 Motion Model

Based on Equations 4.12 and 4.13, the motion model of the GM-PHD filter can be written as:

$$v_{k|k-1}(\mathbf{x}) = \sum_{i=1}^{J_k} p_{S,k}(F_{k-1}\mathbf{m}_{k-1}^{(i)})\, w_k^{(i)} \mathcal{N}(\mathbf{x}; F_{k-1}\mathbf{m}_{k-1}^{(i)}, Q_{k-1} + F_{k-1}P_{k-1}^{(i)}F_{k-1}^T) + \gamma_k(\mathbf{x}). \quad (5.1)$$

Because it is not possible to precisely know the type of dynamics the tracked aircraft are subject to, it is preferable to choose a simple motion model that remains correct for a large range of motions. For this reason, the constant velocity kinematic motion model is chosen, which is defined by the following matrices:

$$F_k = \begin{bmatrix} I_3 & \Delta t I_3 \\ 0_3 & I_3 \end{bmatrix} \quad (5.2)$$

$$Q_k = \begin{bmatrix} \frac{\Delta t^4 \sigma_m}{4} I_3 & \frac{\Delta t^3 \sigma_m}{2} I_3 \\ \frac{\Delta t^3 \sigma_m}{2} I_3 & \Delta t^2 \sigma_m I_3 \end{bmatrix}, \quad (5.3)$$

with Δt the difference in time between two prediction iterations, and $\sigma_m = 3m/s^2$ the noise on the motion. I_3 is the identity matrix of size 3 and 0_3 is the zero matrix of size 3.

A target needs to be removed from the tracker as soon as it goes out of the FOV of both cameras. This is necessary to avoid tracks diverging from the truth (because no measurement can

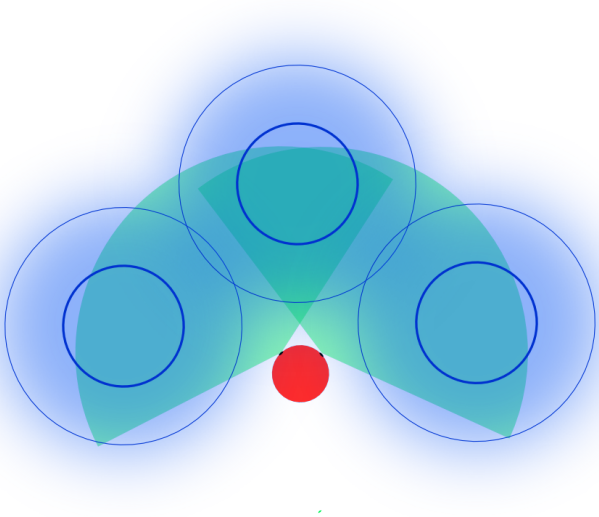


Figure 5.2 – Illustration of the positions and orientations of the camera FOVs (in green) and the Gaussians forming the birth intensity γ (blue). The head of the quadrotor is shown as a red disk. The components are not to scale.

update the state) or the increase in amount of targets (as there are no measurements to lower their weight). To remove the targets, the survival probability is modified to match the FOV of the two cameras combined:

$$p_{S,k}(\mathbf{m}_{k|k-1}^{(i)})) = \begin{cases} 0.9 & \text{if } |\psi_i| < 110° \\ 0.2 & \text{otherwise,} \end{cases} \tag{5.4}$$

with ψ_j the bearing angle of the position defined by $\mathbf{m}_{k|k-1}^{(i)}$ in the body frame of the camera. The probability of survival is set to 0.2 so that targets that go out of the FOV for a short moment of time can quickly be tracked again.

The birth model is composed of three Gaussians, one in the front of the quadrotor at $\mathbf{m}_\gamma^{(1)} = [0, 1.2, 0, 0, 0, 0]m$, and two on the sides of the quadrotor at $\mathbf{m}_\gamma^{(2)} = [-1, 0.2, 0, 0, 0, 0]m$ and $\mathbf{m}_\gamma^{(3)} = [1, 0.2, 0, 0, 0, 0]m$. The full birth model is given by:

$$\gamma_k(\mathbf{x}) = \sum_{j=1}^{3} 0.01 \cdot \mathcal{N}(\mathbf{x}; R_{b,l}\mathbf{m}_\gamma^{(j)} + \mathbf{p}, P_\gamma^{(j)}), \tag{5.5}$$

with $R_{b,l}$ the rotation matrix from the body frame to the local frame and $\mathbf{p}$ the position of the observing quadrotor. The initial covariance matrix is set to:

$$P_\gamma^{(j)} = \begin{bmatrix} 32I_3 & 0_3 \\ 0_3 & 4I_3 \end{bmatrix}. \tag{5.6}$$

An illustration of the birth model is shown in Figure 5.2.

Table 5.1 – Typical camera related parameter values for the two camera setup.

F_x	F_y	X_0	Y_0	K_1	K_2	P_1	P_2	S_t	K_S
111.3	147.7	157.5	118.7	-0.29	0.04	0.001	0.001	0.14	0.09

5.2.2 Camera Model

The measurements from the cameras are composed of the position of the target on the image $[x_p^{(i)}, y_p^{(i)}]^T$ and the size of the target as appearing on the image $s_p^{(i)}$. A measurement vector is given by $\mathbf{z}_k^{(i)} = [x_p^{(i)}, y_p^{(i)}, s_p^{(i)}]^T$. The camera model is based on the standard OpenCV model and is defined as follows. First, the location of target i is put in camera's body frame:

$$\mathbf{p}_{b,k}^{(i)} = R_{c,l}\left(\mathbf{p}_k^{(i)} - \mathbf{p}\right). \tag{5.7}$$

Then the position on the image $[x_p^{(i)}, y_p^{(i)}]^T$ is computed, including camera's distortion:

$$x_p^{(i)} = -F_x x_d^{(i)} + X_0 \tag{5.8}$$
$$y_p^{(i)} = -F_y y_d^{(i)} + Y_0 \tag{5.9}$$
$$r_y = \frac{y_{b,k}^{(i)}}{x_{b,k}^{(i)}} \tag{5.10}$$
$$r_z = \frac{z_{b,k}^{(i)}}{x_{b,k}^{(i)}} \tag{5.11}$$
$$x_d^{(i)} = \left[k_r r_y + 2P_1 r_y r_z + P_2\left(s_r + r_y^2\right)\right] \tag{5.12}$$
$$y_d^{(i)} = \left[k_r r_z + 2P_2 r_y r_z + P_1\left(s_r + r_z^2\right)\right] \tag{5.13}$$
$$s_r^{(i)} = r_y^2 + r_z^2 \tag{5.14}$$
$$k_r^{(i)} = 1 + K_1 s_r + K_2 s_r^2, \tag{5.15}$$

with K_1, K_2, P_1, P_2, F_x, F_y, X_0 and Y_0 representing parameters of the camera. The sensor model for target's pixel size $w_i^{(i)}$ has been derived empirically:

$$s_i^{(i)} = \frac{\pi F_x^2 S_t^2}{4\left(\|\mathbf{p}_{b,k}^{(i)}\|^2 + K_S\|\mathbf{p}_{b,k}^{(i)}\|^4\right)}, \tag{5.16}$$

with S_t the marker's size and K_S a distortion parameter. The intrinsic parameters of the cameras are either measured using a dedicated ROS calibration package [66], or, in the case of K_S, determined empirically. Typical values for the parameters are given in Table 5.1.

The covariance of the sensor noise is estimated to be:

$$R_k = \begin{bmatrix} 16 & 0 & 0 \\ 0 & 16 & 0 \\ 0 & 0 & 64 \end{bmatrix}. \tag{5.17}$$

The clutter model has two values, depending on the measured apparent size, as it was observed that false positive detection are usually only a few pixels in size:

$$\kappa(z) = \begin{cases} \frac{1}{320*240*200} & \text{if } s_p^{(i)} < 10 \\ \frac{0.01}{320*240*200} & \text{otherwise} \end{cases} \tag{5.18}$$

Because the targets are only detectable when in the FOV of the cameras, the detection probability $p_{D,k}$ of camera $l \in 1,2$ is not constant and instead is equal to:

$$p_{D_l,k}(\mathbf{m}_{k|k-1}^{(i)}) = \begin{cases} 0.95 & \text{if } \psi_{c,l} - \frac{\alpha_{d_l}}{2} < \psi_j < \psi_{c,l} + \frac{\alpha_{d_l}}{2} \\ 0 & \text{otherwise} \end{cases} \tag{5.19}$$

with $\psi_{c,l}$ is the heading of camera l in the world frame and $\alpha_{d_l} = 127°$ the FOV of a single camera. This approach is similar as the one found in [67].

5.3 Results

To validate our tracking algorithm, the filter was run on one quadrotor sitting still in our flying arena, and another quadrotor was carried around the recording quadrotor. The position of both quadrotors were recorded using the MCS. The estimated position of the tracked quadrotor was compared with the ground-truth position recorded by the MCS. The obtained error in tracking is shown in Figure 5.3. We found that the error in position was around 26 *cm* in the FOV of interest (i.e. used by the collision avoidance algorithm). Each camera had a frame rate of around 8 *Hz*.

5.4 Discussion

Because the calibration of the camera models are at their worst at the edge of the image, the small regions where two or more cameras overlap see an oscillation of the target, resulting in a noisy estimation of the target's state. This is a problem for velocity estimation in particular. A possible improvement of this system is to stitch the images from the multiple cameras to get a unified picture of the scene. Another solution is to use a single camera with a very wide fish-eye lens (> 200°), which can be useful if image stitching is too computationally expensive.

An error of 26*cm* is roughly twice the size of the marker used. This error is concentrated radially (i.e., on the camera-target axis). This is because the tracking algorithm relies on only a few pixels (around 5 at the maximum range) to estimate the distance between the cameras

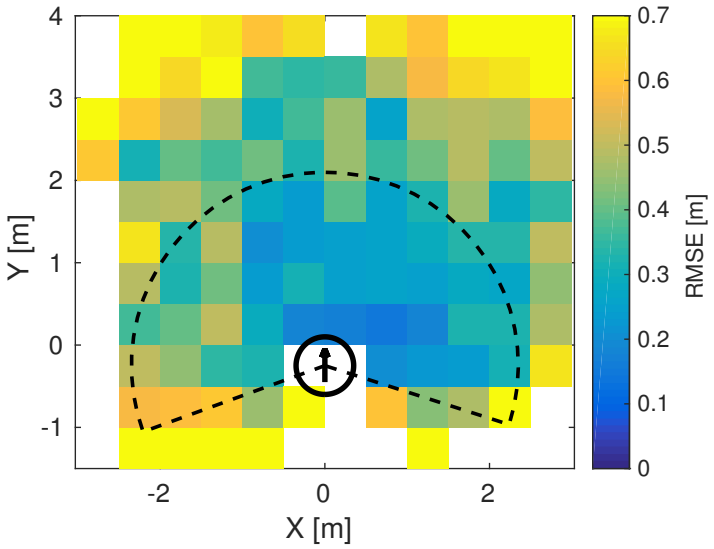


Figure 5.3 – Tracking error of a single target obtained by systematically moving a quadrotor in front of another quadrotor with a fix position. The black circle shows the quadrotor position, the black arrow is the heading and the dashed lines the region usable for avoidance.

and the object. Any pixel difference between the model and reality results in a large distance difference. Furthermore, the pixels on the edge of the color blob are often flickering. For long distances, the color blob is mostly composed of pixels that are on the edge. The signal to noise ratio is thus close to one when the target is far away. One solution to this would be to increase the resolution of the camera, to the expense of higher computational power required.

This tracking algorithm uses measurements given by the color-blob detector algorithm. This computer vision algorithm relies on markers, and is not suited to be used outdoor, where any color can be found in the environment. In next chapter, this tracking algorithm will be adapted to be used with a computer vision algorithm that detects other UAVs based on their shape.

Summary

In this chapter the implementation of the first iteration of the tracking system for the SAA is presented. The detection method is based on the color blob detection computer vision algorithm. The tracking system uses two fisheye cameras to get to the required FOV, and the EKF-GM-PHD filter as tracking algorithm. Results show an average accuracy of 26cm in the FOV of interest.

6 Marker-Less Tracking Based on Shape Detection

In real world applications, and in particular for SAA systems for UAVs, the use of markers for visual tracking is impossible, especially if they are as large relative to the aircraft as what is presented in Chapter 5. Other visual tracking methods need to be used that do not require modifications the structure or aspect of the robot. A natural way to visually detect a robot is by detecting its shape. In [36], in collaboration with the CVLab, we presented our work on tracking other UAVs based on their shape, without the use of markers. In this chapter, we will focus exclusively on the tracking algorithm that leverages the measurements obtained with the shape-detection algorithms presented in Section 2.2.3. The quadrotor shape-detection algorithms described in Section 2.2.3 returns the bounding boxes of detected objects. Due to the fact that width and height of these bounding boxes are heavily correlated, the measurements of a quadrotor are defined as $\mathbf{z} = \left[x_c, y_c, w_c\right]^T$ with the components being the detection center in horizontal and vertical coordinate on the image, and the width of the bounding box, respectively. The shape-detection computer vision algorithm comes in two flavors: with and without the visual tracker. The first version of the shape-detection algorithm solely uses the cascade of detectors. The second version, the detections from the cascade of detectors go through the visual tracker, which learns the appearance of the quadrotor and tracks it on the image directly.

6.1 Filter Implementation

The implementation of the camera sensor model is similar to the algorithm presented in Chapter 5, which uses the standard camera with distortion model, with the exception of the measurement model for the apparent width $w_{c,k}^{(i)}$ of target i at time k which is given by:

$$w_c^{(i)} = \frac{F_x S_t \left(1 + K_1 s_r\right)}{\left(\|\mathbf{p}_{b,k}^{(i)}\| + K_w \|\mathbf{p}_{b,k}^{(i)}\|^2\right)}. \tag{6.1}$$

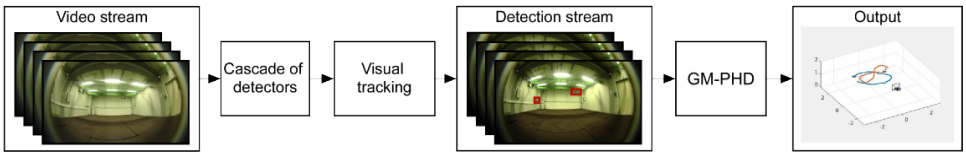


Figure 6.1 – The workflow of the image processing and and tracking. Courtesy of Artem Rosantev.

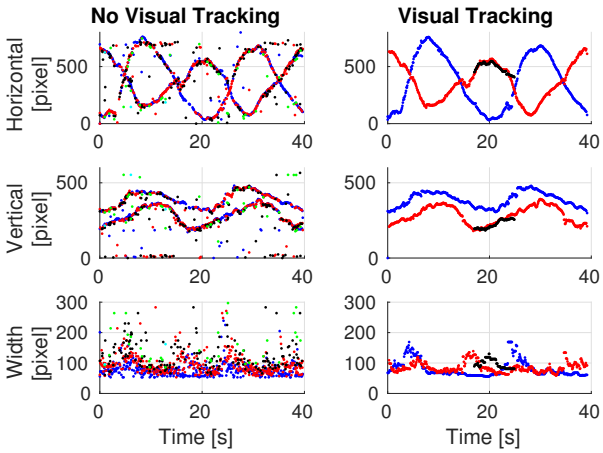


Figure 6.2 – Position estimates obtained from the computer vision algorithm. Different colors correspond to different measurements for the same time-step with this order: blue, red, black, green, cyan. The colors might not be correlated in time nor do they show any association with ground truth.

Similarly to the notation in Chapter 5, $\mathbf{p}_{b,k}^{(i)} = \left[x_{b,k}^{(i)}, y_{b,k}^{(i)}, z_{b,k}^{(i)}\right]^T$ is the position of the target relative to the camera in the camera frame, K_1 and F_x are some standard intrinsic camera parameters and were determined using the ROS calibration package [66], S_t is the size of the target, and K_w is a parameter related to the change of the apparent width as function of the target's distance to the camera and has been determined empirically. As the camera sensor model is not linear, the GM-PHD filter has been implemented using the EKF-GM-PHD filter of Chapter 4. This setup uses a single camera. The camera model is the same as the ones used in Chapter 5. The camera was in a front-facing orientation. Because the shape-based computer vision algorithm only works with high-resolution images, the resolution used was of 960 by 540 pixels (cropped from a 1920 by 1080 pixels' image), instead of the 320 by 240 pixels used in Chapter 5. Consequently, the camera model parameters change as well. Typical values for the parameters are given in Table 6.1. The covariance of the sensor noise for the computer vision without visual tracking is estimated to be:

$$R_k = \begin{bmatrix} 200 & 0 & 0 \\ 0 & 200 & 0 \\ 0 & 0 & 3200 \end{bmatrix}, \tag{6.2}$$

Table 6.1 – Typical camera related parameter values for the shape tracking setup.

F_x	F_y	X_0	Y_0	K_1	K_2	P_1	P_2	S_t	K_S
500.5	500.5	1000.5	539.7	-0.1539	0.0267	0	0	0.32	-0.04

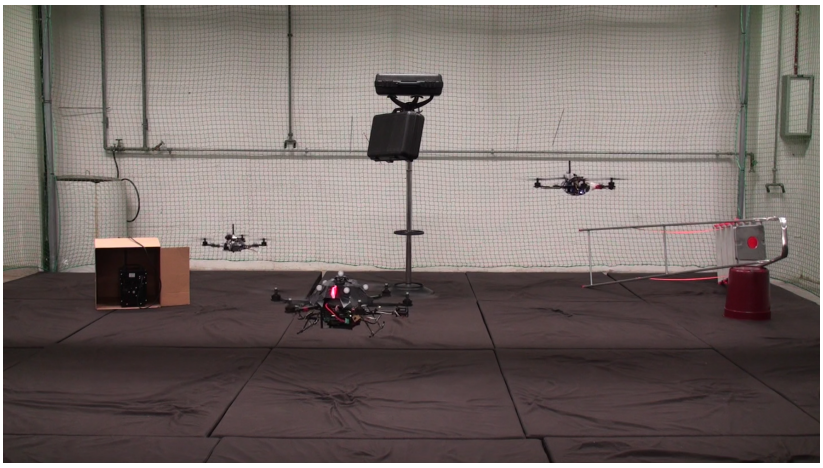


Figure 6.3 – Experimental setup for the shape tracking algorithm. A camera on the AscTec Firefly hexacopter records the scene. Two Hummingbird quadrotors are flying in front of the Firefly, and are the ones that are tracked. Objects were added to the scene to make it more realistic.

and for the computer vision with visual tracking, the covariance is set to:

$$R_k = \begin{bmatrix} 200 & 0 & 0 \\ 0 & 200 & 0 \\ 0 & 0 & 1600 \end{bmatrix}. \tag{6.3}$$

The covariance for the size is set lower because the visual tracker filters the noise on the size measurement. The clutter level was set to $\kappa(\mathbf{z}) = 0.5/(4 \cdot 10^6)$ for the version with visual tracking and $\kappa(\mathbf{z}) = 4/(4 \cdot 10^6)$ for the version without. Unlike what is presented in Chapter 5, the clutter level does not vary as function of the measurement size.

6.2 Results

The experimental setup is composed of an AscTec Firefly hexacopter carrying a camera and two AscTec Hummingbird quadrotors (our standard experimental platform described in Chapter 2) flying in front as targets of the tracking framework. The camera is equipped with a fish-eye lens with a field of view of 185°. All quadrotors flew autonomously using localization data acquired by the MCS and sent through WiFi to the vehicles. The trajectories of the Hummingbird quadrotors were set as follows: for one, an ellipsoid of 3×2 *m* and for the other one an ∞-shape of 2×1 *m*. The quadrotors performed their loop in 20 *s*. The hexacopter was mostly in static flight, moving to another position from time to time. A picture of the experimental setup is shown in Figure 6.3. All the data was recorded on the hexacopter to be

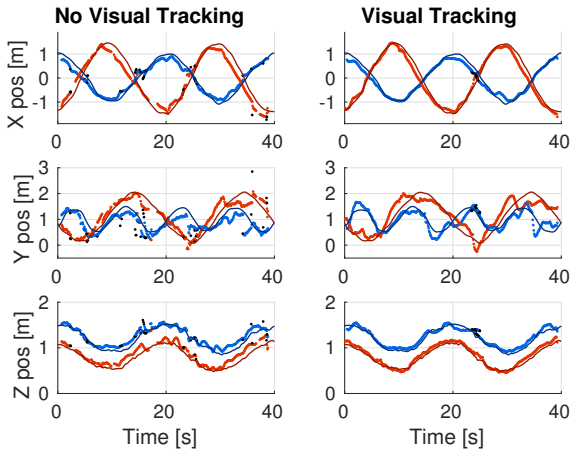


Figure 6.4 – Position estimates obtained in our experiment with two quadrotors (blue and red track, respectively) in a cluttered environment. The solid, thinner lines are the true trajectories of the quadrotors acquired by the MCS. The dots are the estimated positions from the EKF-GM-PHD filter. The red and blue colored dots are the ones closest to the red and blue curves, respectively. The data association is obtained during the computation of the OSPA metric and not as a result of the EKF-GM-PHD filter. The black dots are the estimates that were not associated with a quadrotor.

post-processed by leveraging the rosbag functionality of ROS.

In the post-processing phase, the data was played back in real time and fed to the computer vision and tracking algorithms. The computation was performed on a desktop computer with an Intel Core i7, a hardware comparable to the high-end computation modules for quadrotors (e.g., AscTec Mastermind). Although both versions of the shape-detection algorithm operate on the same video stream, the parameters were set differently for fair comparison. One difference resides in the neighborhood size of the non-maximum suppression step, which is set to 3 for the case with visual-tracking and 1 in case without. This is because visual tracking is more sensitive to false positives, as opposed to the multi-target state tracking which is more sensitive to missed detections. Furthermore, when only the cascade of detectors are used in the shape-detection algorithm, all overlapping detections are averaged together. This is because the cascade of detectors reports the same detection at different sizes, despite the non-maximum suppression step. This step is not applied when the visual tracker step is used in the shape-detection algorithm. A sample of the measurements obtained by both computer vision flavors is shown in Figure 6.2.

We compared the two computer vision algorithms versions: with and without visual tracking. The obtained trajectories for both cases are shown in Figure 6.4. Figure 6.5 shows estimated velocities for both cases. The trajectories are obtained after post-processing the results, by associating estimated targets with the closest ground truth position. Both figures show the estimates as dots and the ground-truth as solid lines. The dots are colored according to the ground-truth they are closest to. On the one hand, the trajectories obtained without visual tracking are not only less accurate than those with visual tracking, but they do also suffer of

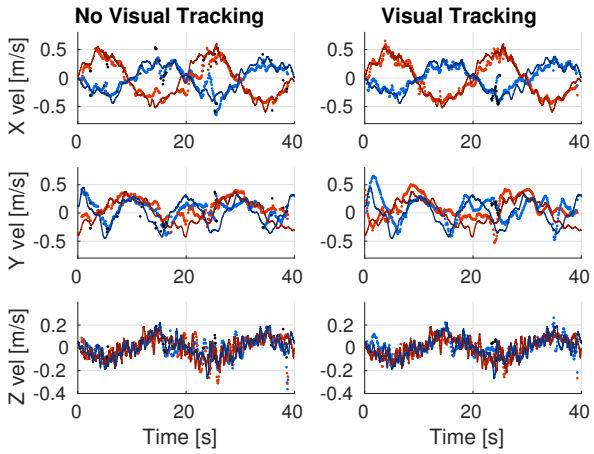


Figure 6.5 – Velocity estimates obtained in our experiment with two quadrotors (blue and red track, respectively) in a cluttered environment. The solid, thinner lines are the true trajectories of the quadrotors acquired by the MCS. The dots are the estimated velocities from the EKF-GM-PHD filter. The red and blue colored dots are the ones closest to the red and blue curves, respectively. The data association is obtained during the computation of the OSPA metric and not as a result of the EKF-GM-PHD filter. The black dots are the estimates that were not associated with a quadrotor.

target disappearing. This is due to the cascade of detectors not reliably detecting a target, and when one is not detected, the EKF-GM-PHD filter immediately reacts by lowering the weight of the estimate below the target existence threshold. On the other hand, the visual tracking will search the image for features that it learned previously. As a result, the algorithm returns a measurement for each quadrotor at each frame. This consistency allows the EKF-GM-PHD filter to correctly track both quadrotors and get a smooth estimate of their velocity.

To assess the improvement that visual-based tracking brings to the tracking framework, the OSPA metric is used with a cutoff threshold $c = 2m$. The OSPA errors are shown in Figure 6.6 for both in the case where visual tracking is enabled in the shape-detection algorithm, and in case it is disabled. Overall, adding visual tracking improves the result, mostly due to the two targets being consistently tracked. The OSPA metric goes suddenly up around 24 and 27 s of the experiment, which is due to a cardinality error in the estimation outputted by the EKF-GM-PHD filter (i.e., the EKF-GM-PHD filter does not report the right number of targets). The reason for the error in cardinality is that the the visual tracker of the shape-detection algorithm is keeping two separate tracks for a single target. This is then interpreted as a two distinct targets by the EKF-GM-PHD filter. Without visual tracking in the shape-detection algorithm, the target's appearance flickers as they are not reliably detected solely with the cascade of detectors, resulting in large and sudden variations of the OSPA metric.

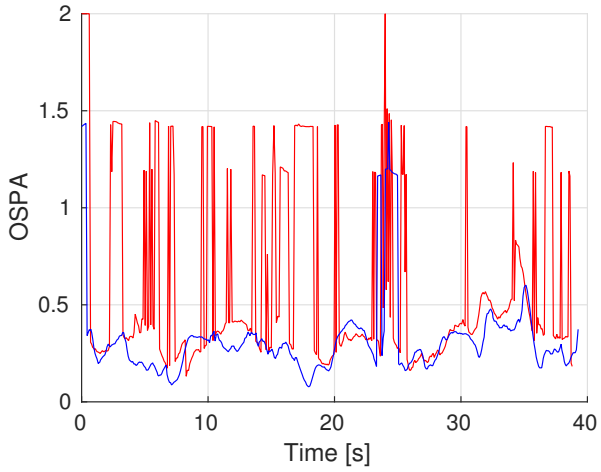


Figure 6.6 – Obtained OSPA [58] measurement for tracked positions of the quadrotors. Blue curve is the OSPA for the case with visual-tracking enabled, red curve is for the case where it is disabled. Lower is better. The error is 2 at maximum.

6.3 Discussion

The shape-based tracking algorithm has two different cascaded trackers (one tracking on the image and another one tracking in 3D), which is not only redundant, but can also lead to wrong tracks persisting for a significant amount of time. A solution would to combine the visual tracking with the state estimation into a single algorithm. The idea would be to use a Labeled Multi-Bernoulli (LMB) tracker [61] that tracks the robots in 3D, then projects the positions of the robots on the image space to generate the pseudo-measurements and update the tracks. If implemented as a particle filter, then even the cascade of detectors (used to extract the image patches that look like a robot) can be merged into the algorithm, as each particle projected on the image would give the image patch to be evaluated by the cascade of detectors. Then the binary response (detected or not) would be used to update the weight of the particle. Preliminary results based on a first merged implementation appear promising, but falls out of the scope of this work.

In this work, the size of the robot is assumed to be accurately known to be able to estimate the distance between the robot and the camera. In more realistic conditions, such information might be missing. For example, several types of robots might be present in the same environment, requiring a prior identification before such assumption can be made. Furthermore, the apparent size of the robots might change as a function of their orientation (which is not the case when rotational symmetry of the spherical markers are used, as in Chapter 5). In that case, the heading needs to be estimated as well as the size, but the heading of a robot is hard to estimate visually therefore leading often to unreliable results. A possible solution would be to add another sensor, such as a range sensor, or a radar, and perform dedicated sensor fusion. The tracking algorithm would become significantly more complex, in particular due to the data association between the sensors, sensor management, and the different sensing modalities

(i.e., single-target measurements for laser range finder versus multi-target measurements for vision).

In this work, the combination of the camera sensor and the lens resulted in a FOV that was smaller than 180°. In the next work, we will present our implementation of the GM-PHD filter to track other quadrotors using a single lens with a FOV of 220°.

Summary

In this chapter, a markerless tracking system based on a shape-detection algorithm is presented, along with its implementation details. Results from real experimental footage are provided for two different versions of the shape-detection algorithm, with and without subsequent visual tracking, and their tracking performances are compared.

7 Marker-Based Tracking with One Camera

To overcome the lack of vertical FOV and improve the velocity estimation of the tracking system presented in Chapter 5, a single camera tracking solution was set up. The same color blob detection algorithm leveraged in Chapter 5 and detailed in Section 2.2.3 is implemented, but using a single camera with a very wide fish-eye lens that requires to significantly change the camera model.

7.1 Hardware

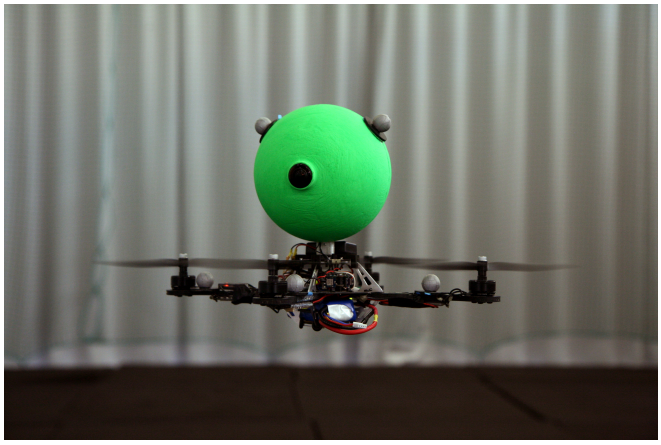


Figure 7.1 – A quadrotor in flight equipped with the green marker. The black disk is the lens of the camera.

To achieve the 220° FOV needed for the collision avoidance algorithms, the BF16M220D lens is mounted on a off-the-shelf USB camera. The camera is mounted inside a hollow polystyrene sphere similar to the hardware presented in Chapter 5. In this case, the markers were painted green due to the presence of red structures in the scene (the experiment using this new marker were carried out in the larger Jordils arena mentioned in Section 2.1.2). Figure 7.1 shows a quadrotor equipped with a green marker. The camera is connected to the on-board

computational unit of the quadrotor and runs the visual detection and tracking algorithms. The camera resolution used is 640 by 480 pixels.

7.2 Camera Model

Because the distortion created is severe, both the camera model and the underlying tracking algorithm had to be modified. The pinhole camera model was changed in favor for the omnidirectional camera model described in [68]. Furthermore, the EKF-GM-PHD filter was modified into a UK-GM-PHD filter to accommodate the strong non-linearity of the model as well as ease the integration of complex camera models.

Computing the sensor model $\mathbf{z} = g(\mathbf{x})$, with $\mathbf{x} = [x, y, z, v_x, v_y, v_z]^T$ is done in multiple steps. First, the estimated position of the target $\mathbf{p} = [x, y, z]^T$ is placed in the camera frame:

$$\mathbf{p}_{c,k}^{(i)} = R_{c,l}\left(\mathbf{p}_k^{(i)} - \mathbf{p}_k^{(c)}\right) \tag{7.1}$$

with $\mathbf{p}_k^{(c)}$ the position of the camera and $R_{c,l}$ the rotation matrix from the local frame to the image frame (z being the direction of the camera). Second, the position of the target is projected on the camera using the method described in [68]. We use a modified version of the method from [69] to compute an approximation with a single iteration. The approximation is based on a polynomial of order seven that has been fitted to match the true solution as well as possible. The projection function $\mathbf{q} = \gamma(\mathbf{p})$ is given by the following equations:

$$\mathbf{q}_p^{(i)} = \begin{bmatrix} c\frac{x\rho}{r} + d\frac{y\rho}{r} + x_0 \\ e\frac{x\rho}{r} + \frac{y\rho}{r} + y_0 \end{bmatrix}, \tag{7.2}$$

$$\rho = \sum_{j=0}^{7} a_j \theta^j, \tag{7.3}$$

$$\theta = \arctan\left(\frac{z}{r}\right), \tag{7.4}$$

$$r = \sqrt{x^2 + y^2}, \tag{7.5}$$

with c, d and e scaling parameters and x_0 and y_0 parameters related to the center of the image. Typical values for those parameters are given in Table 7.1. Care is taken to put r to a small value (e.g. 10^{-12}) if r becomes too small. Typical values for a_j are given in Table 7.2.

To compute the size of the target on the image, four edges of the color blob are computed. To

obtain those edges, first the two vectors **u** and **v** which serve as basis are computed:

$$\mathbf{u} = \frac{\mathbf{p}_{c,k}^{(i)} \times \mathbf{z}}{||\mathbf{p}_{c,k}^{(i)} \times \mathbf{z}||} \tag{7.6}$$

$$\mathbf{v} = \frac{\mathbf{u} \times \mathbf{p}_{c,k}^{(i)}}{||\mathbf{u} \times \mathbf{p}_{c,k}^{(i)}||}, \tag{7.7}$$

with $\mathbf{z} = [0,0,1]^T$. Then the target's edges are computed in 3D:

$$\mathbf{e}_{+\mathbf{u}} = \mathbf{p}_{c,k}^{(i)} + S_t\mathbf{u} \tag{7.8}$$

$$\mathbf{e}_{-\mathbf{u}} = \mathbf{p}_{c,k}^{(i)} - S_t\mathbf{u} \tag{7.9}$$

$$\mathbf{e}_{+\mathbf{v}} = \mathbf{p}_{c,k}^{(i)} + S_t\mathbf{v} \tag{7.10}$$

$$\mathbf{e}_{-\mathbf{v}} = \mathbf{p}_{c,k}^{(i)} - S_t\mathbf{v}, \tag{7.11}$$

with S_t the size of the target. The positions of the edges of the target are projected on the camera image and the size of the target as appearing on the camera is obtained as follows:

$$s_p^{(i)} = \rho_d \sqrt{\frac{\pi}{4} ||\gamma(\mathbf{e}_{+\mathbf{u}}) - \gamma(\mathbf{e}_{-\mathbf{u}})|| \cdot ||\gamma(\mathbf{e}_{+\mathbf{v}}) - \gamma(\mathbf{e}_{-\mathbf{v}})||}. \tag{7.12}$$

ρ_d is a polynomial that adapts the size of the target with the distance. This term was added because of the deviation of the model that was observed when the target is far from the camera. The cause of the change of scale is not clear. ρ_d is given by:

$$\rho_d = \sum_{j=0}^{4} b_j \left(\frac{||\mathbf{p}_{c,k}^{(i)}||}{S_t} \right)^j, \tag{7.13}$$

with b_j the coefficients of the polynomial obtained by calibration. Typical values for b_j are given in Table 7.3. The distance is divided by the size of the target to obtain a unit-less quantity. The covariance of the sensor noise is estimated to be:

$$R_k = \begin{bmatrix} 9 & 0 & 0 \\ 0 & 9 & 0 \\ 0 & 0 & 4 \end{bmatrix}. \tag{7.14}$$

The clutter model has two values, depending on the size measurement, as it was observed that false positive detection are usually only a few pixels in size:

$$\kappa(z) = \begin{cases} \frac{1}{320*240*200} & \text{if } s_p^{(i)} < 10 \\ \frac{0.01}{320*240*200} & \text{otherwise} \end{cases} \tag{7.15}$$

The calibration of the camera follows a two-step process. First, the camera parameters are

Table 7.1 – Typical camera related parameter values for the single camera tracking setup.

c	d	e	x_0	y_0	S_t
1.0596	2.69e-3	-13.68e-3	289.84	244.27	0.0581

Table 7.2 – Typical camera related parameter values for the radial distortion polynomial given by Equation 7.3.

a_0	a_1	a_2	a_3	a_4	a_5	a_6	a_7
199.09	115.08	-20.12	-14.91	-3.2154	944.48e-3	-1.056	-789.21e-3

estimated using the OCamCalib library [69]. This calibration step is done by taking images of a checkerboard with the camera. Second, the model parameters are optimized in order to minimize the difference between data acquired from the camera and the ground truth obtained with the MCS. The dataset is obtained using a fixed camera and a moving marker (the same that are used on the quadrotors), both being tracked using the MCS. The camera model is applied to the ground-truth positions to generate pseudo-measurements (what the measurement should look like). The images from the camera are processed using the color blob tracking algorithm that is used to track the other quadrotors. The detections from the camera are compared to the pseudo-measurements obtained with the MCS, and the error is used as input to the *fminsearch* optimization function of Matlab.

7.3 Results

To validate the single-camera tracking system, a marker was moved in front of a camera recording the position of any green blob on the image. The positions of the camera and the marker were also recorded using the MCS. The trajectory of the marker during the experiment can be seen in Figure 7.2. The dataset was then used to feed the tracking algorithm of this chapter, and the OSPA error for each estimate is computed. The result is shown in Figure 7.2.

The mean error in position is around 0.45 m in the FOV used by the avoidance algorithms, and quickly degrades as soon as the marker goes out of range. Furthermore, the velocity error is on average 0.33 m/s. The marker is correctly tracked most of the time, with moments where the tracker is unable to start a track for the marker. The reason why the tracker fails to track occasionally is for now unknown.

7.4 Discussion

The lens used in this chapter has enough FOV so that only one camera is necessary to obtain enough sensory coverage for the collision avoidance algorithms. But such a large FOV comes with an extreme distortion of the image. This is not a problem for the marker-based computer vision algorithm, as the marker is of a uniform color, and the distortion only affects the size of marker on the image (i.e., the marker is green, independently of the distortion). This is not

Table 7.3 – Typical camera related parameter values for the distance distortion polynomial given by Equation 7.13.

b_0	b_1	b_2	b_3	b_4
1	-4.66e-3	-62.11e-6	1.05e-6	7.025e-12

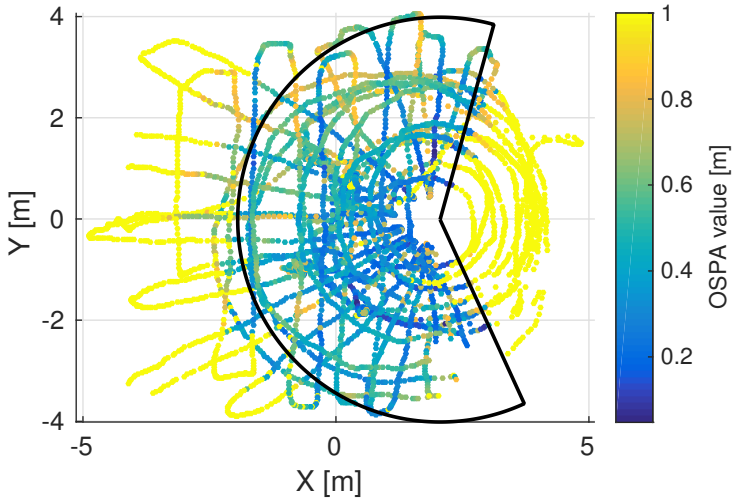


Figure 7.2 – Tracking error of a single target obtained by systematically moving a marker in front of a fixed camera. Each dot represents the OSPA error in position for a measurement update. The position of the dot is given by ground-truth. Only the update measurements for which the ground-truth is valid are shown. The black circle sector shows the FOV used for the collision avoidance algorithms.

true for the shape-based computer vision algorithm, which relies on the appearance of the aircraft on the image. In this case, the distorted shape of a quadrotor needs to be learned, increasing the variability of the data-set. Even if the distortion does not prevent the use of the shape-based detection algorithms, the effect of the distortion on its performance is still unknown at this point.

Summary

This chapter presents the improvements done over the marker-based tracking system of Chapter 5. A single camera with a 220° FOV lens is used. The camera model is modified to adjust for the correspondingly higher image distortion. The tracking of other quadrotors is done using the UK-GM-PHD filter instead of the EK-GM-PHD filter. Implementation details as well as performance evaluation using a camera and a single marker are given.

Collision Avoidance Part III

8 Introduction to Part III

THIS part presents our contributions to the field of collision avoidance algorithms. All novel algorithms presented in this part share common features. First and foremost, they all guarantee collision avoidance with limited FOV sensing. Second, our collision avoidance algorithms generate the same type of output: a desired velocity and heading. Finally, all algorithms also require at least a position estimate of neighboring UAVs, which is provided by one of the tracking algorithms presented in Part II.

Most collision avoidance algorithms we developed were designed for vehicles that move on a two dimensional plane. In this case, two types of vehicles were studied: the holonomic and the unicycle. The holonomic vehicle is able to move in all directions, independently of its heading. The vehicle's heading is decoupled from the velocity of the vehicle. A unicycle type vehicle is only able to move forward along the direction defined by the vehicle's heading. The velocity and heading of the vehicle are coupled, and the vehicle is unable to move sideways, and is thus non-holonomic. The unicycle kinematic model is used to study differential-wheeled robots and serves as a simplified model for fixed-wing aircraft. In this work, unicycle, differential-wheeled, and fixed-wing describe the same type of motion, although the actuation constraints might differ. One algorithm in this thesis (Chapter 14) was designed to take advantage of the third dimension. The motion model for this three dimensional collision avoidance algorithm is an extension of the unicycle model.

Different collision avoidance frameworks were used to design our algorithms, namely Rule-Based (RB), Potential Field (PF), and Velocity Obstacle (VO). Although different collision avoidance frameworks require a position estimate of the neighboring UAVs, the VO approach also needs a velocity estimate of other UAVs.

This part will continue, followed with Chapter 9 providing a literature review on collision avoidance algorithms. The concept of collision avoidance with limited FOV sensing is presented in Chapter 10. The different collision avoidance algorithms are presented in Chapters 11-15, and their characteristics are described in Table 8.1. Finally, Chapters 16 and 17 present our efforts to compare the performance of different collision avoidance algorithms.

Table 8.1 – Characteristics of the different collision avoidance algorithms.

Chapter	Kinematics	Avoidance method	Uses velocity estimates
11	Unicycle	RB	No
12	Holonomic	PF	No
13	Unicycle	PF	No
14	3D fixed-wing	PF	No
15	Holonomic	VO	Yes

9 Avoidance: Related Work

AVOIDANCE, and collision avoidance in particular, is a well studied subject, with many avoidance paradigms created over the years. This work focuses on the topic of collision avoidance, that is, robots or autonomous vehicles avoiding each other. This is different from obstacle avoidance, in which robots avoid static objects. Of course, collision avoidance and obstacle avoidance are not incompatible and can be solved together with a single algorithm.

In this work, several collision avoidance paradigms are used, such as Rule-Based (RB), Potential Field (PF), and Velocity Obstacle (VO). Other paradigms are possible, such as, for example, geometry-based [70], [71], dynamic window [72], generation of an optimal trajectory [73]–[75], Markov decision process [76] or collision cones [77]–[79]. The simplest approach to collision avoidance is probably RB, where the robot has distinct behaviors and switches between them based on the sensor measurements. Some work apply directly a RB as a control law, as in [80] where the authors extend the VFR for autonomous separation of aircraft. However, RB is usually not implemented alone, but is built on top to extend the capabilities of another algorithm. For example, in [81] a VO algorithm is extended by a set of rules.

Another popular collision avoidance paradigm we use in this work is based on PF, where the robot follows the negative gradient of a virtual PF function. The PF is tailored to adapt the behavior of the robot to the application. The scientific literature is rich in different PF algorithms and PF have been adapted to a large variety of systems such as cars [82], [83], telescopes [84], aircraft [85][86], differential wheeled robots [87] or quadrotors [88].

Finally, in this work, we also use the VO paradigm for collision avoidance. VO-based algorithms compute the optimal command (with regard to some function, usually the minimum deviation from some desired velocity) from the set of allowed velocities. The set of allowed velocities is constructed by removing all sets of velocities that will lead to a collision. A set of velocities that will lead to a collision with an obstacle is called a VO. Several improvements to VO-based collision avoidance have been proposed over the years, such as sharing the responsibility of the collision avoidance maneuver between the robots [89] or making it computationally

faster [90]. The notion of VO can also be extended to be computed in the command input space instead of being simply defined in the velocity space. This allows the VO approach to be used on a large set of vehicle types while guaranteeing that the result is always feasible [91]. VO-based algorithms have been applied to many type of vehicles, such as cars [92], boats [93], or quadrotors [94]. It has also been adapted to follow the VFR [81].

If limitations in the actuation (e.g., maximum speed or non-holomonicity) are usually considered when a collision avoidance algorithm is designed, limitations in the sensing (might it be accuracy, range, field of view) are usually an afterthought, and the sensors are chosen with enough safety margin to guarantee collision avoidance. Only some work consider the case where the robot's sensing system has limitations. Some work, such as [95], [96] or [97], consider that the sensors have a limited range, but not a limited FOV. Other work consider the case where sensory data has some uncertainties, such as [90]. Some contributions have a notion of a limited FOV (i.e., not all directions are sensed). In [98], the authors present an obstacle avoidance algorithm with camera FOV constraints. But because the nature of the problem between obstacle avoidance and collision avoidance differ, the results cannot be reused as is. For example, in [99] the authors present a collision avoidance for limited FOV sensing and validated in simulation, although no analytic proof of collision avoidance is presented. In fact, in this work it will be proven that the FOV used in [99] is not sufficient to guarantee collision free navigation. Some other work, such as [96] or [100] have algorithms that incorporate a limited FOV, but none of those work give an explicit, analytic guarantee of collision avoidance under limited FOV sensing, which we will do in the next chapter.

Summary

In this chapter, the literature on collision avoidance algorithms is reviewed, with a focus on sensor limitations, and in particular for limited FOV sensing. The literature review shows that only a few studies investigated the effect of a limited FOV sensing on the collision avoidance algorithm, and that no work gives guarantee of collision-free navigation with limited FOV sensing.

10 Collision Avoidance for Limited Field of View Sensing

A ROBOT, just like animals, relies on its sensors to observe and understand its environment, to interact with objects of interest, and avoid potential threats. Depending on the sensory type and the environment, the volume observed can vary greatly, ranging from very short (e.g., touch), to long range (e.g., vision). The volume observed by the sensor has to be measured and adapted to the system, as high-performance high-resolution sensors come at an expense, both physical and computational. This is the case for vision in Nature, animals' FOV depends on their use thereof. A few animals have a full spherical FOV, such as the stalk-eyed fly [101], and comes at the cost of having large eyes that are placed in vulnerable locations. When it comes to birds, their FOV varies depending whether they are preys or predators. Preys have large FOV to easily detect a predator, wherever it can attack from. For example the American woodcock can observe the whole hemisphere above it [102]. Predators, such as the barn owl (with 150° FOV), have more front-facing eyes, reducing their FOV, but augmenting their binocular vision [103], probably used to evaluate distances to preys, and because they are less threaten by predator attacks (as they are on top of the food chain). Similarly, for robots, the FOV of a camera has to be adjusted based on the application, and there is a trade-off between the size of the FOV required for the task and other requirements (resolution, distortion, weight). In this chapter, we will provide the minimum FOV size to guarantee the success of the task, which is collision avoidance.

10.1 Field of View

The FOV of a robot can have different shapes, depending on the configuration of the sensor. The FOV can be formed by a combination of several sensors, but it is then assumed that they are fused into a single FOV without discontinuity. The results presented in this chapter assume a disk sector shaped FOV defined in two dimensions, but it can easily be extended to the third dimension, as presented in Chapter 14. The FOV set in two dimensions $\mathbf{F}_i$ of robot i is defined

a) b) c)

Figure 10.1 – Illustration of animals with different FOV: a) the stalk-eyed fly, b) the American woodcock c) the barn owl. [1].

as:

$$\mathbf{F}_i = \{\mathbf{p} \mid ||\mathbf{p} - \mathbf{p}_i|| < r_i^s, |\angle(\mathbf{d}_i, \mathbf{p} - \mathbf{p}_i)| \leq \alpha_i^s\}, \tag{10.1}$$

with $\mathbf{p}_i$ the position of the robot i, $\mathbf{d}_i$ the heading of the robot and $\pi/2 < \alpha_i^s < \pi$ the sensor's FOV half-angle. Without loss of generality, the sensor range r_i^s is also included in this definition by convenience. In this work, the sensor's range is assumed to be constant for all bearing angles in the FOV (but can be extended to arbitrary range functions). Unless stated otherwise, robot i can detect robot j if and only if the position $\mathbf{p}_j$ of robot j, defined as its geometrical center, is in robot i's FOV: $\mathbf{p}_j \in \mathbf{F}_i$. The FOV set only defines where another robot is detected, and not how its state is measured; the sensor could, for example, return full position and velocity measurements (such as in Chapter 15), or return a binary value (such as in Chapter 11).

The main issue with navigation under a limited FOV, as described by Equation 10.1, is the risk of collision arising from the ignorance of the area that is not sensed (the blind angle). Most of the time, the limited FOV poses little threat as the robot moves forward facing, leaving its ignorance behind (assuming its FOV is centered at the front of the robot). In that case, the only threat comes from other, faster robots chasing it; this possibility is out of the scope of this thesis. Prominently, the risk of collision due to ignorance resides when aggressive maneuvers are required, because, for example, the robot changes course (as shown in Figure 10.2), or has to perform a collision avoidance maneuver (as shown in Figure 10.3). The motion of the robot has thus to be limited in order to avoid potential collision with unknown threats.

Two scenarios were designed to show that collision due to the ignorance caused by the FOV is possible. The first scenario involves two robots that start back-to-back. Both robots have their destination on the left side of the horizontal axis. One slower robot has the right orientation and simply has to go forward to reach its destination. The other, faster, robot faces away from its destination. The fastest trajectory is for the robot to go in a straight line, which leads to a collision if using an algorithm that does not take into account the FOV constraints of the

[1] a) Image from Wikipedia (CC BY-SA 2.0 DE): https://en.wikipedia.org/wiki/Stalk-eyed_fly
b) Image from Fyn Kynd (CC BY 2.0): https://www.flickr.com/photos/79452129@N02/16885281018
c) Image from Wikipedia (CC BY 2.0): https://en.wikipedia.org/wiki/Barn_owl

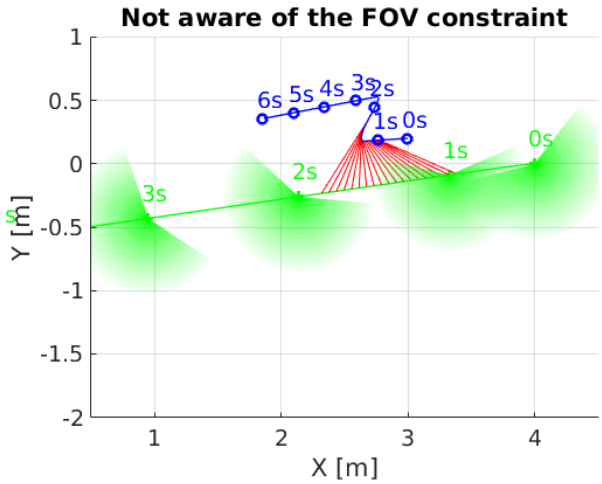


Figure 10.2 – Illustration of the risk of changing course with limited FOV sensing. The green robot changes course at time 0 *s* while slowly rotating its sensor towards its new course. It does not see the slow moving blue robot and collides with it, represented as red solid lines. The green and blue solid lines are the trajectories of the green and blue robots, respectively. The green circle sector represents the FOV of the green robot. All robots use the algorithm given in Chapter 15.

avoided , as shown in Figure 10.2.

The second scenario involves four robots. Two slow robots avoid each other and thus get close. Just when the two slow robots successfully avoid the collision and go out of each other's FOV, each one sees a fast robot approaching from the other side they performed the avoidance. If the algorithm is not respecting the constraints imposed by the sensor, the slow robots will perform an aggressive maneuver towards each other to avoid the faster robots. That maneuver will lead them to collide (Figure 10.3).

We use the scenarios shown in Figures 10.2 and 10.3 to do the simulations in Chapters 12, 13 and 15, both to show that our algorithms solve the FOV ignorance issue, and have common scenarios to compare trajectories. Because the collisions due to limited FOV only happen in specific configurations (e.g., initial positions, algorithm parameters, etc.), and because the configuration that leads to a collision also depends on the type of algorithm, the parameters used for the simulations are slightly different depending on the algorithm evaluated.

10.2 Sensor-Constrained Set

The main concept of collision avoidance under FOV sensing constraints is to limit the motion of the robots in order to guarantee that they cannot collide with each other because of their intrinsic blind zones. In this work we propose that the velocity remains in the following set, called the *sensor-constrained set*:

$$\mathbf{S}_i = \left\{ \mathbf{v} \;\middle|\; |\angle(\mathbf{v}, \mathbf{d}_i)| \leq \alpha_i^s - \frac{\pi}{2} \right\}. \tag{10.2}$$

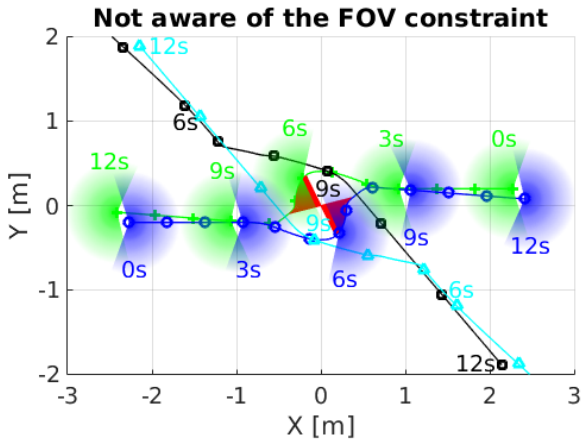


Figure 10.3 – Illustration of the risk of avoiding collisions with limited FOV sensing. Between time 3 *s* and 6 *s*, the green and blue robots avoid each other, and end up leaving each other FOV. At that moment, each one detects another robot, black for the green robot and cyan for the blue robot. The blue and green robots will attempt to avoid collision by moving toward each other (without knowing, as they are not in each other's FOV) and will collide, which is represented as red solid lines. The green, blue, cyan and black solid lines are the trajectories of the green, blue, cyan and black robots, respectively. The green and blue circle sector represents the FOV of the green and blue robots, respectively. All robots use the algorithm given in Chapter 15.

We will now prove that constraining the velocity within the set defined by Equation 10.2 will be sufficient to ensure safe maneuvering. First, we show that robots that do not see each other will not get closer if they choose their velocity in their respective sensor-constrained sets.

Lemma 1. A robot i that chooses $\mathbf{v}_i \in \mathbf{S}_i$ will not go towards the position of another robot j that robot i is unable to detect.

Proof. First note that because robot i does not detect robot j we have that $|\angle(\mathbf{d}_i, \mathbf{p}_j - \mathbf{p}_i)| > \alpha_i^s$. Define a line $\Gamma \perp (\mathbf{p}_j - \mathbf{p}_j)$ that passes somewhere between robots i and j, with a distance of at least r_i and r_j from robot i and j, respectively. Because $\mathbf{v}_i \in \mathbf{S}_i$, it is possible to find a lower bound for the angle between $\mathbf{v}_i$ and the vector $\mathbf{p}_j - \mathbf{p}_i$:

$$|\angle(\mathbf{v}_i, \mathbf{p}_j - \mathbf{p}_i)| = |\angle(\mathbf{d}_i, \mathbf{p}_j - \mathbf{p}_i) + \angle(\mathbf{v}_i, \mathbf{d}_i)| \geq \tag{10.3}$$
$$|\angle(\mathbf{d}_i, \mathbf{p}_j - \mathbf{p}_i)| - |\angle(\mathbf{v}_i, \mathbf{d}_i)| > \alpha_i^s - (\alpha_i^s - \frac{\pi}{2}) = \frac{\pi}{2}.$$

As a result the velocity $\mathbf{v}_i$ points away from the line Γ and moves away from the position of robot j. ■

Using Lemma 1, we will now prove there is no collision possible between two robots that do not see each other. Figure 10.4 summarizes the principle behind the following proposition.

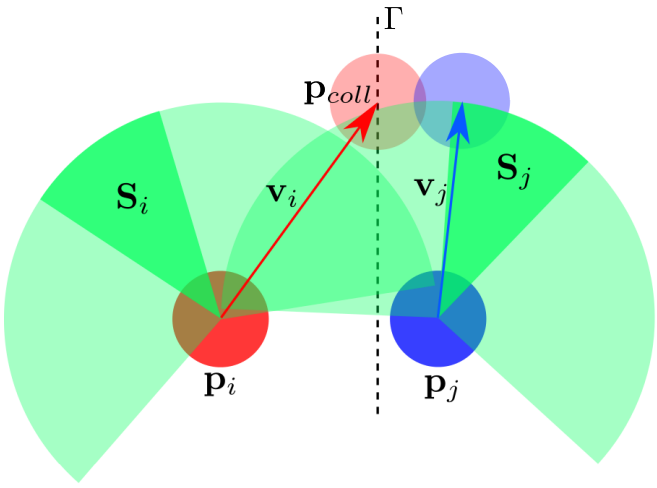


Figure 10.4 – Illustration of a situation for which choosing $\mathbf{v}_i \notin \mathbf{S}_i$ leads to a collision. Robot i is the dark red disk, robot j is the dark blue disk. The FOV of the robots are shown as light green circular sectors. The dark green circular sectors are the sensor-constrained sets for each robot. The light red and light blue disks represent position of robot i and j, respectively, that collided due to robot i choosing a velocity outside of $\mathbf{S}_i$.

Proposition 1. Two robots i and j that do not detect each other, and do not have prior knowledge of each other, are guaranteed to be collision free if and only if $\mathbf{v}_i \in \mathbf{S}_i$ and $\mathbf{v}_j \in \mathbf{S}_j$.

Proof. The proof of necessity goes as follows: suppose that there is a velocity $\mathbf{v}_i \notin \mathbf{S}_i$ that does not lead to a collision with a robot j with any possible $\mathbf{p}_j$, $\mathbf{d}_j$ and $\mathbf{v}_j \in \mathbf{S}_j$. Choose $\mathbf{p}_j$ and a small ϵ such that $|\angle(\mathbf{d}_i, \mathbf{p}_j - \mathbf{p}_i)| = \alpha_i^s + \epsilon$. It follows that $\mathbf{p}_j \notin \mathbf{F}_i$ and that robot i does not see robot j. Now take, without loss of generality, a velocity $\mathbf{v}_i$ such that $|\angle(\mathbf{v}_i, \mathbf{d}_i)| > \alpha_i^s - \pi/2 + \epsilon$, and that $\angle(\mathbf{v}_i, \mathbf{d}_i)$ has the same sign as $\angle(\mathbf{d}_i, \mathbf{p}_j - \mathbf{p}_i)$. Note that by definition $\mathbf{v}_i \notin \mathbf{S}_i$. It follows that $|\angle(\mathbf{v}_i, \mathbf{p}_j - \mathbf{p}_i)| < \pi/2$. Define the line $\Gamma \perp (\mathbf{p}_j - \mathbf{p}_i)$ that goes between the two robots at a distance of $r_j + r_i/2$ from robot j. Because $\angle(\mathbf{v}_i, \mathbf{p}_j - \mathbf{p}_i) < \pi/2$, the center of robot i will cross Γ at point $\mathbf{p}_{coll}$ in some time in the future Δt_{coll}. It is now possible to choose $\mathbf{v}_j$ and $\mathbf{d}_j$ such that $\mathbf{v}_j \in \mathbf{S}_j$ and that the distance between the robot j and the line Γ is less than $r_j + r_i$ when robot j is closest to $\mathbf{p}_{coll}$. It is thus possible to find a triplet $\mathbf{p}_j$, $\mathbf{d}_j$ and $\mathbf{v}_j \in \mathbf{S}_j$ that will lead to a collision between robots i and j. It is thus not possible to have $\mathbf{v}_i \notin \mathbf{S}_i$ without risking a collision with j.

The proof that $\mathbf{v}_i \in \mathbf{S}_i$ is a sufficient condition for no collision is a direct consequence of Lemma 1. If both robots do not see each other, then they both move away from the line Γ that divides the space between the two robots. They thus move away from each other. ■

Remark 1. The result remains true when more than two robots are involved because a single collision is essentially a two-robot problem. More than two robots leads to multiple collisions, each one involving only two robots.

Remark 2. Note that Proposition 1 is valid in both two and three dimensions (in the case of three dimensions, Γ is a plane and not a line in Proposition 1).

Remark 3. The angle α_s has to be larger than $\pi/2$ for the sensor-constrained set to exist. If $\alpha_s < \pi/2$, then $\mathbf{S} = \emptyset$ and no guarantee about collision can be given. The risk of collision when $\alpha_s < \pi/2$ is studied in [104]. As a result, the work in [99], based on the withdrawn standard [105], is not safe considering the FOV used. A possible collision scenario is, for example, two aircraft exactly above one another going in the same direction at the same speed. If the one below slowly climbs or the one above slowly goes down, they will collide without seeing each-other.

The sensor-constrained set only constraints the kinematics of a robot. If the dynamics of a robot cannot be approximated using exclusively kinematic laws (i.e., the robot has significant inertia), the dynamics has to be controlled such that the robot's velocity remains in the sensor-constrained set.

If the robot follows a trajectory in open loop over a time window, then at any point on the trajectory, the remaining of the trajectory has to stay in the sensor-constrained set defined at that point. A trajectory can be planned outside of the sensor-constrained set as long as both a sensor update and a trajectory replanning is performed before leaving the sensor-constrained set. Same goes for limited range: the trajectory should be updated before the robot leaves the area covered by its sensors at the time the trajectory was previously updated. As a result, high sensory frame-rate and frequent trajectory update allows for more aggressive trajectories, despite the constraints imposed by the limited FOV.

At the time of writing this thesis, it is not yet known whether it is possible to design a navigation algorithm that is guaranteed to be collision free with FOV $\alpha_s < \pi/2$ in the case where:

- Each robot knows the existence of every other robot or moving object in the environment.
- Each robot has some knowledge of the state of the other robots. This could be obtained by prediction when another robot is not in FOV or by observing the state of other robots.
- Each robot knows the behavior of the other robots.

Such assumptions make sense in applications where the robots keep some cohesion (such as formation control). Furthermore, if a model of the mobility constraints is known, it is possible to use motion primitive based approaches [75], which admit a formal verification of collision avoidance. There might also be systems for which the motion guarantees that all threats can be detected despite a FOV smaller than $\pi/2$, for example in case of train systems. But in the general case of collision avoidance, in particular for SAA applications, none of the assumptions above are fulfilled, and a FOV of $\alpha_s > \pi/2$ is required.

10.3 Guidelines for Avoidance under Limited FOV

Below a FOV of $\pi/2$, no guarantee of collision avoidance can be given. Furthermore, for FOV that are close to this required minimum, the reduction in maneuverability can be severe and affect the performance of some algorithms (more results are presented in Chapter 16). The specific requirements of using a limited FOV sensor in collision avoidance can be formulated with the following guidelines:

1. **The actuation shall be constrained to be in the sensor-constrained set defined by Equation 10.2.** The reasons for this constraint are explained above.

2. **The vehicle shall be able to stop before colliding.** Due to the reduced maneuverability enforced by sensor-constrained set, it is possible that a robot faces situations where the required collision avoidance maneuver involve velocities outside of the sensor-constrained set. Leaving to all robots the possibility to come to a full stop guarantees at least one safe solution for all robots.

3. **The sensor should be aligned with the vehicle's motion.** In order to keep enough maneuverability at all time, the sensor's heading should follow the velocity vector of the robot. In case where the sensor's FOV is not enough to clearly avoid collision, the robot should stop and turn its sensor until it can see a clear path.

4. **There should be a smooth transition from navigation to avoidance (and vice versa) at the edge of the sensor-constrained set.** This proposition cannot be applied to all algorithms, but helps to avoid oscillations when a robot sees another robot at the edge of its FOV. This rule is mostly used for PF-based algorithms.

Following these guidelines when designing a collision avoidance algorithm should give guarantee of collision-free navigation, as long as the other limitations (i.e., sensor imperfection and actuation dynamics) are taken into account. Illustration of the use of these guidelines on well-known collision avoidance paradigms are shown in Chapters 11, 12, 13, 14 and 15. The guidelines are formulated in the context of collision avoidance, but are also applicable to other, more general navigation problems. Some of the guidelines, such as sensor alignment, could be modified to better track some regions of interest.

With the exception of the first guideline, the guidelines given in this chapter are not confirmed by theory but were established solely on empirical findings. Future work is required to find out if they are indeed necessary conditions or if it is possible to make collision avoidance algorithms that have good performance without following those last three guidelines.

Summary

To guarantee collision-free navigation with a sensory system that has a limited FOV, the actuation has to be constrained to the so-called sensor-constrained set. In this chapter the notion of FOV and sensor-constrained set are defined. We mathematically have shown that the motion of the robot has to lie within the sensor-constrained set to remain collision-free. Finally, guidelines are given on how to adapt a collision avoidance algorithm in order to guarantee its safety even with limited FOV sensing.

11 Rule-Based Avoidance

THE guidelines described in Chapter 10 can be applied to many different types of collision avoidance concepts, including very simple ones. In this chapter, we aim to design a collision avoidance algorithm as simple as possible but that respects the constraints imposed by the limited FOV. This is done in the form of a Rule-Based (RB) algorithm.

Consider the problem of N differential-wheel robots with the following kinematic equation:

$$\dot{\mathbf{q}} = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} c\cos(\theta) \\ c\sin(\theta) \\ \omega \end{bmatrix}, \tag{11.1}$$

with c the command in speed and ω the turning rate command. Their task is to navigate in some space and avoiding collisions with the environment or other robots. The robots have a proximity sensor that allows them to detect any object that comes in close range, as long as the object is in its FOV. The sensor does not return any distance or bearing measurement. The measurement z returned by the sensor is given by:

$$z = \begin{cases} 1, & \text{if } \mathbf{O} \cap \mathbf{F} \neq \emptyset \\ 0, & \text{otherwise,} \end{cases} \tag{11.2}$$

with $\mathbf{O}$ the set of all positions covered by obstacles and $\mathbf{F}$ the FOV. Note that this definition differs from what is given in Chapter 10 in that even the edge of an obstacle is enough to be detected by the sensor, and not necessarily the center. Such sensor can be implemented with a large variety of sensor modality, such as vision, but also a laser range finder, proximity sensors (if their FOV overlap), or even a tactile border around the robot (if collision avoidance means "not colliding too hard").

A simple way to navigate the space while avoiding collision that respects the three first guideline described in Chapter 10 is to move forward while no obstacle is detected, and to turn on

the spot when an obstacle is detected. This rule-based control law is given by:

$$\begin{bmatrix} c \\ \omega \end{bmatrix} = \begin{cases} [c^{max}, 0]^T, & \text{if } z = 0 \\ [0, \omega^{max}]^T, & \text{if } z = 1, \end{cases} \tag{11.3}$$

with $c^{max} > 0$ the maximum speed of the robot, and $\omega^{max} \neq 0$ its maximum turning rate.

Proposition 2. Two robots with kinematic properties given by Equation 11.1, and the control law given by Equation 11.3, will not collide.

Proof. Consider two robots i and j that are at a small enough distance from each other such that, if they are facing the other robot, they would detect it. Now consider the case of robot j from the point of view of robot i. If $\mathbf{p}_j \notin \mathbf{F}_i$, then, according to Proposition 1, robot i will not be the cause of the collision with robot j (i.e., either it stops or moves away from robot j). If $\mathbf{p}_j \in \mathbf{F}_i$, then robot i is not moving and is not responsible for the collision. The same reasoning can be made for robot j. Thus, both robots will not have a motion that can lead to a collision and as a result no collision is possible between robots i and j. ■

Remark 4. Proposition 3 also applies to more than two robots, because either all the other robots are out of the FOV of some robot i and thus robot i moves away from the other robots, or at least one robot is in the FOV of robot i and robot i is not moving, thus not participating in the collision. Because

$$\left(\bigcup_j \mathbf{O}_j\right) \cap \mathbf{F}_i = \bigcup_j \left(\mathbf{O}_j \cap \mathbf{F}_i\right) \neq \emptyset, \tag{11.4}$$

with $\mathbf{O}_j$ the obstacle formed by some robot j, robot i will be stopped whenever there is one or multiple robots in its FOV.

In fact, the RB algorithm takes advantage of the inherent dynamics of the robot (i.e., a differential wheeled robot) to satisfy the constraints imposed by the limited FOV. In general, if a robot is only able to move forwards, then it will not violate the sensor-constrained set imposed by the limited FOV.

11.1 Results

We validated our RB-based approach in simulation, in which 8 robots navigate a square space of six by six meters. The simulations was implemented in Matlab as a microscopic point simulation (no physical simulation) with perfect kinematics and noiseless sensing. The simulation was carried out with a time step of 0.05 *s*. The robots' shape is a disk of radius $r = 0.3$ *m*. The maximum speed c^{max} of each robot is different and ranges from 0.4 to 0.9 *m/s*; this variability in speed makes possible that a robot can hit another robot from behind. For all robots $\omega^{max} = 1$ *rad/s*. The sensor range is $r^s = r + 1.8\Delta t$ from the center of the robot. This sensor range is the minimum sensor range to leave the robot one time step Δt to react and

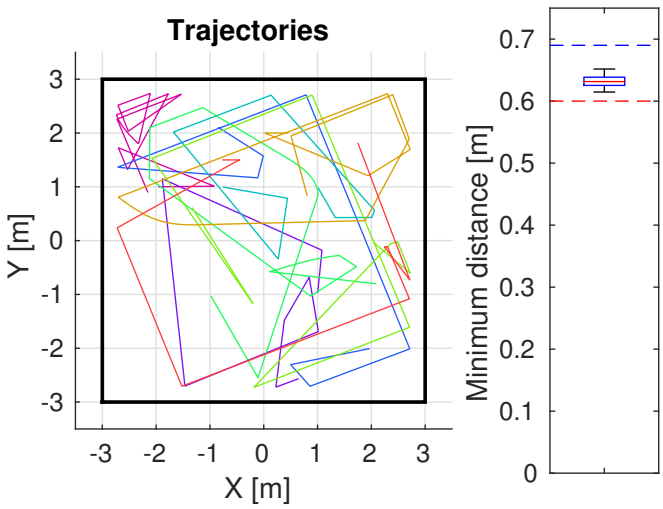


Figure 11.1 – Simulation of the RB approach. Left: trajectories obtained over 800 time steps. Right: Boxplot of the minimum distance between the robots over 100'000 time steps. The red dashed line shows the distance at which two robots are considered colliding, and the blue dashed line is the sensor range.

stop in order to avoid the collision. The FOV half angle of the sensor is $\alpha^s = 110°$. The result of the simulation is shown in Figure 11.1. Over the course of 100000 simulation iterations, the robots moved in their environment without colliding with each-other.

11.2 Discussion

Other simple algorithms are possible, such as the Braitenberg algorithm [106] applied to differential wheeled robots, as long as the weights used in the Braitenberg are set in such a way that, for any sensors values and a limited FOV facing forwards, the robot never moves backwards. It is not yet known if there is a set of weights that not only avoid collisions, but also guarantee that the robot never moves backwards.

It is possible to generate a smoother trajectory with the RB approach by slowly accelerating the robot to its cruise speed after it was stopped. It is not yet known if a smooth deceleration is also possible, as it requires the robot to keep enough distance for it to decelerate in time.

Although this RB approach does guarantee collision avoidance, it does not guide a robot towards some desired destination. To navigate the environment, this RB algorithm has to be extended to steer the robot towards its destination. A RB approach for navigation might be too aggressive to navigate the robot, and another method, such as PF, might be more suited for the task. In that case, the robot would navigate the environment using the PF if the path is clear, and switch to a collision avoidance behavior if an obstacle is detected.

Summary

In this chapter, we show that a simple rule-based control law can follow the guidelines presented in Chapter 10. The rule-based control law is presented, as well as proof of collision avoidance and simulations.

12 Potential-Field-Based Avoidance for Holonomic Vehicles

One of the popular paradigms for collision avoidance is based on the concept of artificial Potential Fields (PFs). Like their natural counterparts, artificial PFs are scalar functions that embody in their negative gradient the motion of the objects they affect (e.g., electrons in nature, robots for artificial potential fields). With artificial PFs the negative gradient can be replaced by another local operator. By carefully designing the PF, it is possible to obtain a wide variety of robotic behaviors. Some PFs show good behavior for navigation, and are called navigation functions. A PF V with a destination set $\mathbf{X}^d$ has the following properties [107]:

- $V(x) = 0 \forall\, x \in \mathbf{X}^d$
- $V(x) = \infty$ if and only if $\mathbf{X}^d$ is not reachable from x.
- for every reachable state $x \in \mathbf{X}/\mathbf{X}^d$, following the local operator will produce a new state x' such that $V(x') < V(x)$

12.1 Problem Statement

In this work, the PFs are implemented without necessarily strictly following the definition above. First, our local operator is always the negative gradient of the PF. Second, the PF used in this work is the sum of two different PFs:

$$V^t = V^n + V^a, \tag{12.1}$$

with V^n the PF responsible for navigating the robot to its destination, and V^a the PF to avoid other robots or obstacles. This allows to design each behavior separately with relative ease and reduces possible unexpected behaviors. But it is likely that when the robot reaches its destination and $V^n = 0$, that $V^a \neq 0$, and as a result $V^t > 0$. But this is a small issue, as the author of [107] states that the first point is more for convenience. But it is important that both V^n and V^a are as low as possible at the destination $\mathbf{X}^d$ so that the destination lies in

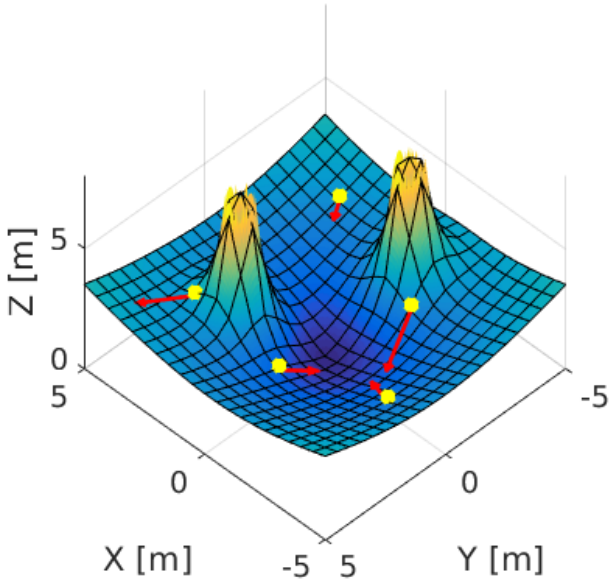


Figure 12.1 – Illustration of the shape of the PF with two obstacles. The yellow dots with red arrows represent possible robot positions and the negative gradient associated to those positions.

a global minimum of the PF. Furthermore, depending on the configuration of the obstacles and other robots, and the shape of V^n, it is possible that local minimums appear. This can only be avoided by carefully choosing the shapes of the PFs. In this chapter, we will show how some generic PF can be made compliant with the requirements stemming from a limited FOV sensing.

PFs have been adapted to a large variety of dynamical systems. In this chapter, a holonomic robot controlled in speed is chosen because it illustrates how the motion of the robot needs to be limited due to the limited FOV. The robot's equation of motion is given by:

$$\dot{\mathbf{q}} = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} v_x \\ v_y \\ \omega \end{bmatrix}, \tag{12.2}$$

with $\mathbf{p} = [x, y]^T$ and $\mathbf{v} = [v_x, v_y]^T$ the position and the velocity of the robot, respectively. Many PF functions are possible; inspired from [82], V^n and V^a are chosen to be:

$$V^n = K^n \frac{\|\mathbf{p}_i - \mathbf{p}_i^d\|^2}{\|\mathbf{p}_i - \mathbf{p}_i^d\| + r^d} \tag{12.3}$$

$$V^a = \sum_j \frac{K^a}{\gamma + \|\mathbf{p}_i - \mathbf{p}_j\|^2 - (r_i + r_j)^2}, \tag{12.4}$$

with $\gamma > 0$ a parameter to make the avoidance component less aggressive. $\mathbf{p}_i^d$ is the position

of the destination for robot i, and $r^d \geq 0$ is a term to slow down the robot when it reaches its destination. K^n and K^a are constant gains that set the intensity of the navigation and avoidance components of the PF, respectively. The rationale behind the choice of V^n is that the norm of its gradient close to K^n in most places and tends to 0 as the vehicle gets close to its destination. As a result, the desired cruise speed (when not avoiding) of the robot is defined by $c^d = K^n$. The PF formed by V^n and V^a is illustrated in Figure 12.1. The negative gradient of the total PF is then used to steer the vehicle to its destination:

$$\mathbf{v} = -\nabla V^t, \tag{12.5}$$

with

$$-\nabla V^t = -K^n(\mathbf{p}_i - \mathbf{p}_i^d)\frac{\|\mathbf{p}_i - \mathbf{p}_i^d\| + 2r^d}{[\|\mathbf{p}_i - \mathbf{p}_i^d\| + r^d]^2} + \sum_j \frac{2K^a(\mathbf{p}_i - \mathbf{p}_j)}{[\gamma + \|\mathbf{p}_i - \mathbf{p}_j\|^2 - (r_i + r_j)^2]^2}. \tag{12.6}$$

This algorithm is named PFOS in Chapter 16. To ensure collision avoidance, the gains of the PF have to be carefully chosen so that under a certain small range ϵ the gradient of the PF moves the robots apart, even in the worst conditions which are when the goal is far away ($\|\mathbf{p}_i - \mathbf{p}_i^d\| >> r^d$). That is, for $\|\mathbf{p}_i - \mathbf{p}_j\| < \epsilon$, the following relation has to hold true:

$$\left\|-\nabla V^n\right\| < \left\|-\nabla V^a\right\|. \tag{12.7}$$

Considering the single robot case, and taking the limit on the distance to the destination to infinity, Equation 12.7 reduces to:

$$K^n < \frac{2K^a\epsilon}{\left[\gamma + \epsilon^2 + (r_i + r_j)^2\right]^2}. \tag{12.8}$$

12.2 FOV-aware PF

Up to this point, the PF does not comply with the FOV guidelines presented in Chapter 10. To ensure no collision due to the limited FOV, the negative gradient of the PF has to be modified so that the velocity remains in the sensor-constrained set, i.e. $[v_x, v_y]^T \in \mathbf{S}$. As a result, the PF V^t would need to be modified to comply to the constraints enumerated in Chapter 10. This is challenging for several reasons. First, because the vehicle is only allowed to move in its sensor-constrained set, the navigation is only well defined in that set (by definition, the PF requires that $V(q) = \infty, \forall q \notin \mathbf{S}$). Second, the goal might be outside the sensor-constrained set, which is invalidating the requirement that $V(q) = 0$ at destination. Third, because the PF at the edge of the sensor-constrained set needs to go to infinity, and because the sensor-constrained set is usually narrow, the gradient can see dramatic changes over a small distance, which can lead to instability. This is especially important when considering sensors that are noisy, as the estimated position can significantly change between two time steps.

Instead of searching for a PF that contains the limited FOV constraint, we will use a function

Ψ that maps the PF's gradient into the sensor-constrained set:

$$\begin{bmatrix} v_x \\ v_y \end{bmatrix} = \Psi(-\nabla V^t, \theta) \in \mathbf{S}. \tag{12.9}$$

Note that Ψ is dependent on the robot's heading θ, because the sensor-constrained set is aligned with the robot's heading. Furthermore, function Ψ should follow the guidance of the PF as much as possible, while preventing forbidden motions. A possible solution goes as follows: first define two vectors $\mathbf{c}^{\parallel}, \mathbf{c}^{\perp}$ parallel and perpendicular to the sensor heading. Then project the gradient of the PF $\mathbf{f}^t = -\nabla V^t$ on those two axes:

$$\mathbf{f}^{\parallel} = \mathbf{c}^{\parallel} \frac{\mathbf{c}^{\parallel} \cdot \mathbf{f}^t}{\|\mathbf{c}^{\parallel}\|^2} \tag{12.10}$$

$$\mathbf{f}^{\perp} = \mathbf{c}^{\perp} \frac{\mathbf{c}^{\perp} \cdot \mathbf{f}^t}{\|\mathbf{c}^{\perp}\|^2}. \tag{12.11}$$

Finally, $\mathbf{f}^{\perp}$ is scaled so that when combined with the projection on $\mathbf{f}^{\parallel}$, the result remains in $\mathbf{S}$. The Ψ function is thus defined as follows:

$$\Psi = \begin{cases} -\nabla V^t, & \text{if } -\nabla V^t \in \mathbf{S}_{\alpha^a} \\ \mathbf{0}, & \text{if } |\angle(-\nabla V^t, \mathbf{c}^{\parallel})| > \frac{\pi}{2} \\ \mathbf{f}^{\parallel} + \text{sgn}(\mathbf{f}^t \cdot \mathbf{c}^{\perp}) M^{\alpha} \mathbf{f}^{\parallel}, & \text{otherwise,} \end{cases} \tag{12.12}$$

with $\mathbf{S}_{\alpha^a}$ the sensor constrain set with FOV angle of α^a, with α^a an angle such that $\pi/2 < \alpha^a < \alpha^s$ (this angle is used to smooth the collision avoidance further on), and

$$M^{\alpha} = \sin\left(\alpha^a - \frac{\pi}{2}\right) \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix}. \tag{12.13}$$

An illustration of the vectors used in the mapping function is shown iun Figure 12.2. Notice how that if $-\nabla V^t$ is such that it would move the robot backward, the mapping function has to return $\mathbf{0}$ to prevent violation of the limited FOV constraint. This conflict between the need to quickly change course to avoid collision and the risk of collision due to the lack of knowledge due to the limited FOV can only be solved by a change of heading of the sensor, such as stated by the third guideline of Chapter 10. Other mapping functions Ψ are possible, such as projecting the gradient of the PF on the closest edge of the sensor-constrained set.

Directly using the PF described by Equation 12.6 into the mapping function given by Equation 12.12 will guarantee collision avoidance, but depending on the configurations of the different robots, it might not result in smooth trajectories due to the robots appearing and disappearing from each other FOVs. This is due to the non-respect of the fourth guideline proposed in Chapter 10. To avoid oscillations due to the FOV, the collision avoidance PF V^a

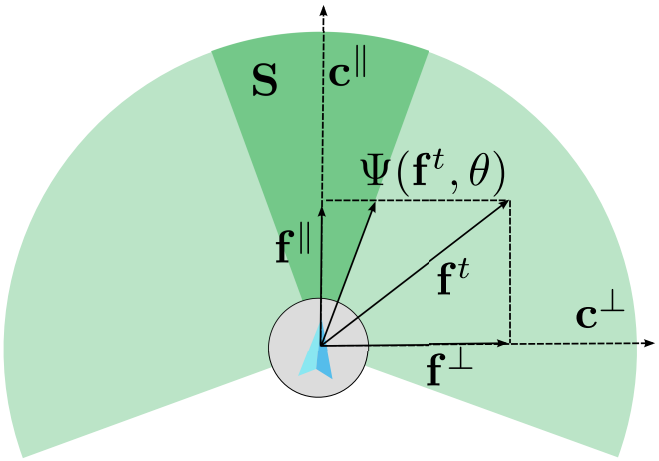


Figure 12.2 – Illustration of the vectors used in the mapping function Ψ. The light green sector represents the robot's FOV and the the dark sector represents the sensor-constrained set.

can be modified to give the following equation:

$$-\nabla V^a = \sum_j \frac{2K^a \beta^a \beta^d (\mathbf{p}_i - \mathbf{p}_j)}{[\gamma + \|\mathbf{p}_i - \mathbf{p}_j\|^2 - (r_i + r_j)^2]^2}, \tag{12.14}$$

with

$$\beta^d = \beta\left(\frac{r^s - d_{ij}}{r^s - r^a}\right) \tag{12.15}$$

$$\beta^a = \beta\left(\frac{\alpha^s - |\theta_{ij}|}{\alpha^s - \alpha^a}\right), \tag{12.16}$$

and d_{ij} the distance between robots i and j and θ_{ij} the bearing of robot j with respect to robot i. To simplify the computation, we take the approximation that both d_{ij} and θ_{ij} are constant and do not depend on the positions of the robots. r^a is a parameter related to the smoothness of the transition with $0 < r^a < r^s$. β is a smooth monotonic function from 0 to 1, in this work it is implemented as:

$$\beta(a) = \begin{cases} 0 & \text{if } a < 0 \\ 3a^2 - 2a^3 & \text{if } 0 \le a < 1 \\ 1 & \text{otherwise} \end{cases} \tag{12.17}$$

The β function are used to smoothly ignore other robots that are on the border of the sensed area, might it be due to long distance or large bearing angle. The shape of the combined β functions is shown in Figure 12.3. Equation 12.14 is the reason to use α^a instead of α^s in Equation 12.13: it allows for a smooth transition near the edge of the FOV. But having a large difference $\alpha^s - \alpha^a$ reduces the "normal operation" mode of the PF (i.e., when $-\nabla V^t \in \mathbf{S}_{\alpha^a}$). Furthermore, the size of $\mathbf{S}_{\alpha^a}$ also defines how much controlability there is for the heading control (i.e. harder to trade-off between aggressive and sluggish heading control). There is

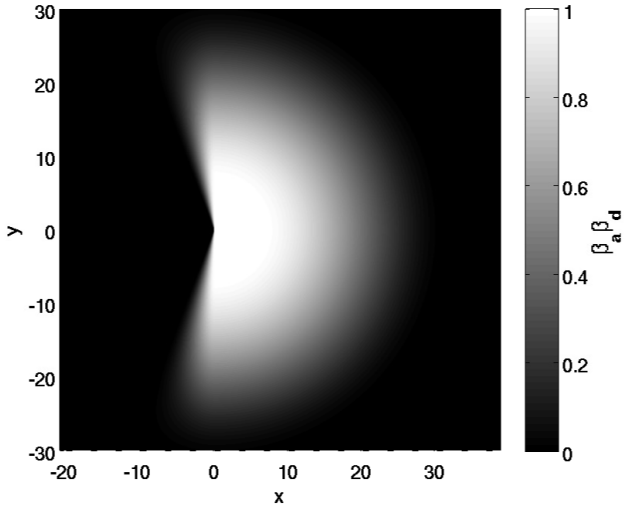


Figure 12.3 – Shape of the function $\beta_{a_j}\beta_{d_j}$. The function equals zero in the back of the agent, ignoring any other agents in that area as it is unable to sense them.

thus a trade-off between maneuverability and smoothness of the trajectories. This algorithm, which is sensor-constrained, is named SAPFOV in Chapter 16.

We are now able to formulate the following proof:

Proposition 3. Two robots with dynamics given by Equation 12.2, and the PF given by Equations 12.1, 12.3 and 12.14, transformed by the mapping given by Equation 12.12, and under the condition given by Equation 12.8 will not collide.

Proof. Consider two robots i and j at a distance $d_{ij} < \min(\epsilon, r_a)$. We will show that each robot will not move closer to the other robot.

First consider the situation from the point of view of robot i. Robot j can be at different positions around robot i. First, robot j can be at a location such that $|\theta_{ij}| > \alpha^a$. In that case, according to Proposition 1 and because $\mathbf{v}_i \in \mathbf{S}$, robot i will either move away from robot j, or be stopped. If robot j is in the FOV of robot i such that $|\theta_{ij}| < \alpha^a$, then we have that $\beta^a\beta^d = 1$ because $d_{ij} < r_a$. The avoidance component has thus full strength in this configuration. Furthermore, because of condition given by Equation 12.8, we know that the avoidance component in the PF gradient is larger than the navigation component. As a consequence, the negative gradient of the total PF for robot i will point away of robot j (and thus robot i will have a velocity of $\mathbf{0}$, as per Equation 12.12). As a result, for any position of robot j such as $d_{ij} < \min(\epsilon, r_a)$, robot i will either move away from robot j or not move, and will not be responsible for the collision.

The same reasoning can be applied to robot j. Because both robots either move away from each other or do not move at all. Hence collision is impossible. ■

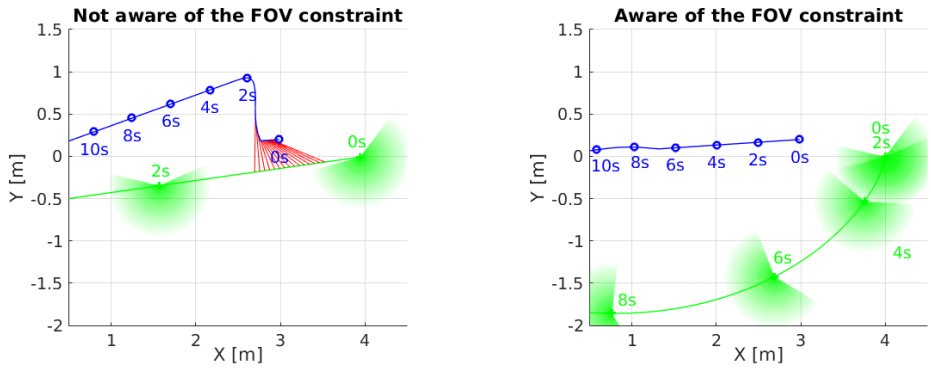


Figure 12.4 – Trajectories obtained with the first scenario. Left: the algorithm is not aware of the FOV. Right: The PF algorithm respects the FOV constraints. The solid blue and green lines are robots' trajectories. The markers and numbers give an indication of the timing of the trajectories. The FOV of some robots at the timings indicated by numbers are represented by circular sectors. The color of the circular sectors match the color of the trajectories. The FOV range is not to scale. The red solid lines are distances between two robots that are less than the sum of their radii.

Remark 5. Proposition 3 only applies to two robots. In case more than two robots are involved in the collision avoidance, the avoidance forces can cancel out, leaving the navigation strong enough to push a robot into a collision. A solution could be to reduce the weight of the navigation component while avoiding collisions. But this solution increases the risk of deadlocks.

According to the third guideline proposed in Chapter 10, the heading of the robot should be aligned with robot's velocity. A special rule has to be implemented to avoid deadlock: according to Equation 12.12, in the case where the negative gradient of the PF is in the opposite direction of the sensor's direction, the mapped PF's negative gradient is zero, and aligning the sensor with robot's velocity is not well defined. To avoid the deadlock, when the command velocity of the robot given by the mapped negative gradient, the robot rotates with $\dot{\omega}^{max}$ until the deadlock is solved. As a result, the heading control law is given by:

$$\omega = \begin{cases} -K^{\omega} \angle(\mathbf{d}_i, \mathbf{v}_i), & \text{if } \mathbf{v}_i \neq \mathbf{0} \\ \dot{\omega}^{max}, & \text{otherwise,} \end{cases} \tag{12.18}$$

with K^{ω} a constant gain.

12.3 Results

This algorithm was validated in simulation. The simulations was implemented in Matlab as a microscopic point simulation (no physical simulation) with perfect kinematics and noiseless sensing. The PF approach was simulated using the two scenarios presented in Chapter 10. The resulting trajectories are presented in Figures 12.4 and 12.5. Both scenarios are tuned in order

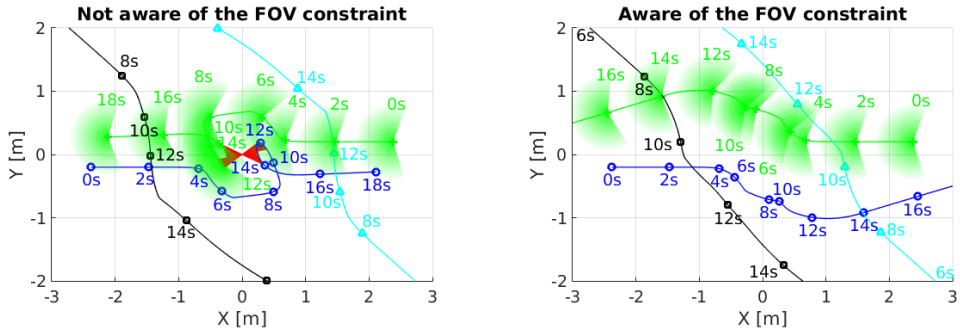


Figure 12.5 – Trajectories obtained with the second scenario. Left: the algorithm is not aware of the FOV. Right: The PF algorithm respects the FOV constraints. The solid blue, black, green and cyan lines are robots' trajectories. The markers and numbers give an indication of the timing of the trajectories. The FOV of some robots at the timings indicated by numbers are represented by circular sectors. The color of the circular sectors match the color of the trajectories. The FOV range is not to scale. The red solid lines are distances between two robots that are less than the sum of their radii.

for the non FOV-aware PF algorithm generates trajectories that create collisions. For both scenarios, and with the same parameters, the PF version that respects the FOV constraints generates trajectories that do not collide.

In general, the deadlock problems are partially due to the inability of PFs to anticipate the required collision avoidance maneuver because they only take as input the position of other robots and not the velocity. Another reason is that for FOV that are only marginally larger than $\pi/2$, the change of heading is slow for reasonable values of K^{ω}. Next chapter presents a solution in the form of heading control instead of velocity control, which partially addresses these issues.

12.4 Discussion

This chapter presents a general methodology to make PFs compatible with the requirements imposed by limited FOV sensing. The algorithms presented here are more for the sake of illustration, and many changes can be made to improve the performance of this class of algorithms. First, the PFs used in this chapter could be changed to something that are less deadlock-prone, especially in the case of head-on collisions. Furthermore, the way the negative gradient of the PF is mapped into the sensor-constrained set can be tuned to a specific application. Another improvement would be to modify the heading control to be more aggressive when the PF's negative gradient goes significantly out of the sensor-constrained set, with the hope to limit the amount of time spent in deadlocks. Finally, switching behaviors (e.g., getting away from other robots before going back into navigation mode) might also mitigate deadlock problems.

Summary

In this chapter, a generic method to adapt PFs to the requirements imposed by limited FOV sensing is presented. First, the concept of PF is presented, followed by a specific implementation of PF. Then the methodology of adapting the PF to make it compatible with the FOV is discussed. Finally simulations of the obtained collision avoidance algorithm are presented.

13 Potential Field-Based Avoidance for Fixed-Wing Vehicles

INSTEAD of artificially constraining the velocity to the sensor-constrained set, it is possible to find a control law that, similarly to the RB-based approach of Section 11, inherently satisfies the constraints stemming from limited FOV sensing due to the dynamics of the system. Furthermore, such control law can be applied to systems which are not capable to move sideways, such as fixed-wing aircraft. This algorithm was first presented in [37], and was originally not designed with Theorem 1. This chapter presents the algorithm as first presented in [37] and subsequently deployed on real robots in [35], while incorporating more recent knowledge on the topic.

13.1 Problem Statement

In this chapter, we focus on fixed-wing type of aircraft to design the algorithm. The fixed-wing kinematic model is approximated by the unicycle robot equation:

$$\begin{bmatrix} \dot{x}_i \\ \dot{y}_i \\ \dot{\theta}_i \end{bmatrix} = \begin{bmatrix} c_i \cos\theta_i \\ c_i \sin\theta_i \\ \omega_i \end{bmatrix}, \tag{13.1}$$

with ω_i the turn rate command and $0 \leq c^{min} \leq c_i \leq c^{max}$ the forward speed command of robot i. Note that this notation allows for constant speed robots with $c^{min} = c^{max}$, although it is not desirable as we will show in Section 13.2. $\mathbf{p}_i = \begin{bmatrix} x_i, y_i \end{bmatrix}^T$ is robot's position and θ_i is its heading. Together, $\mathbf{p}_i$ and θ_i define the pose vector. In this chapter all angles are defined on the interval $[-\pi, \pi]$ and all operations with angles return a value in that interval. A robot i has a collision radius r_i. Two robots i and j should not come closer than a radius $r_i + r_j$ to each other, and below that distance, they are considered to have collided. Each robot i can sense a robot j which is in its FOV $\mathbf{p}_j \in \mathbf{F}_i$ (with r^s the radius and α^s the half FOV angle), such as defined in Chapter 10. The measurement returned by the sensor is a range $d_{ij} = \|\mathbf{p}_j - \mathbf{p}_i\| - r_i - r_j$ and a bearing angle $\phi_{ij} = \text{atan2}(y_j - y_i, x_j - x_i) - \theta_i$. The angles and distances are illustrated in Figure 13.1.

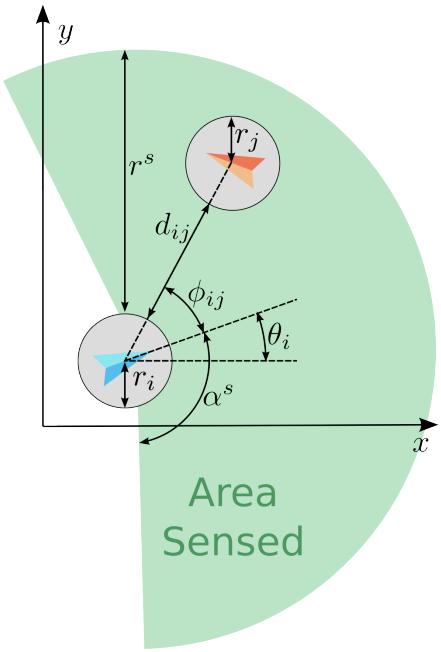


Figure 13.1 – Illustration of the angles and distances used in this chapter.

Because the range of speed allowed by the robot i can be limited (e.g. in case of fixed-wing aircraft), the maneuvering is mostly done by steering its heading in a desired direction θ_i^t. Furthermore, as the minimum velocity c^{min} is not necessarily 0, collision avoidance is also done by changing the heading. In this case, the collision avoidance is done through a simple turning behavior, with increasing rate as the threat gets closer. The navigation and collision avoidance behaviors are linked together by the shape of the FOV, similarly to Chapter 12. The turn rate command u^1 is given by:

$$\omega_i = K^n \left(1 - \max_j \left(\beta_j^a \beta_j^d\right)\right)\left(\theta_i^t - \theta_i\right) + \sum_j \beta_j^a \beta_j^d \frac{-2\pi c^{max}}{d_{ij}}, \tag{13.2}$$

with

$$\beta_j^d = \beta\left(\frac{r^s - d_{ij}}{r^s - r^a}\right) \tag{13.3}$$

$$\beta_j^a = \beta\left(\frac{\alpha^s - |\phi_{ij}|}{\alpha^s - \pi/2}\right), \tag{13.4}$$

with β the function given by Equation 12.17. r^a is the radius for which, in case $d_{ij} < r^a$, a robot will fully switch to collision avoidance. Note that this algorithm is not suited for 360° FOV sensing, as it will keep spinning around if another robot is close by and never leave its collision avoidance behavior.

[1] The yaw control described by Equation 13.2 is slightly different from the work in [37] and [35]. This is to correct for a mistake in [37] where the turn rate is only half what would be needed to prove avoidance. The proof in [37] is also correct if the avoidance term in the yaw control is multiplied by two.

There are many possible ways to compute the desired heading θ_i^t. A solution is to use PFs; inspired by [86], the following PF was used:

$$V_i(\mathbf{p}_i) = \|\mathbf{p}_i - \mathbf{p}_i^d\|^2 + K^p \beta\left(\frac{\|\mathbf{p}_i - \mathbf{p}_i^d\|}{r^p}\right) e_i^2 \tag{13.5}$$

$$e_i = \frac{\|(\mathbf{p}_i^o - \mathbf{p}_i^d) \times (\mathbf{p}_i - \mathbf{p}_i^d)\|}{\|\mathbf{p}_i^o - \mathbf{p}_i^d\|}, \tag{13.6}$$

with $\mathbf{p}_i^d$ the position of the destination and $\mathbf{p}_i^o$ the robot's initial position. K^p is a constant gain defining how much the robot will deviate from the line between $\mathbf{p}_i^o$ and $\mathbf{p}_i^d$. r^p defines a distance at which the robot should slow down when getting close to its destination. The negative gradient of this PF is not directly used as input. Rather, it is used to compute the desired heading $\theta_i^t = \text{atan2}(y_t, x_t)$ with $\left[x_t, y_t\right]^T = -\nabla V_i(\mathbf{p}_i)$.

Most robots, including fixed-wing aircraft, have some speed range $\Delta c = c^{max} - c^{min}$ that they can use to improve the collision avoidance performance. To improve the separation capability of the robots (explained in Section 13.2), the speed of the robot is set as:

$$c_i = c^{max} - \max_j\left(\beta_j^a \beta_j^d\right)\Delta c, \tag{13.7}$$

with β_j^a and β_j^d given by Equations 13.43 and Equations 13.42, respectively. This algorithm is named SAPFUV in Chapters 16 and 17. Using the system described by the equations above, the following proposition can be formulated.

Proposition 4. A robot with a kinematic model described by Equation 13.1 and with a controller described in Equations 13.2 and 13.7 will not move in a way that will lead to collision with another robot.

Proof. Suppose that a collision occurs at time t^c between two robots and consider them earlier when they were apart from each other by a small distance $2r^c$. As they can go at a maximum speed of c^{max}, there is a small time interval $\Delta t = \frac{r^c}{c^{max}}$ for which they will not collide. Δt is thus a lower bound of the time required for them to collide. We want to prove by contradiction that *if a robot is moving towards another robot, there will not be a frontal collision between the two robots.* That is, a robot on a collision course will turn enough to not face the other robot and move away from the collision point in the time Δt, contradicting with the fact that t^c is the time of collision.

As they are on a collision course, at least one robot, for instance robot i, is moving towards the other during the time $t \in [t^c - \Delta t, t^c]$. Because it is moving towards the collision, robot i is sensing another robot j in front of it, meaning that $|\phi_{ij}| \leq \frac{\pi}{2}$. Thus according to Equation 13.4, $\beta_j^a = 1$. Moreover, the distance between the robots is small, meaning that $d_{ij} \leq r^a$ thus $\beta_j^d = 1$ (from Equation 13.3). As a result we have $\left(1 - \max_j \beta_j^a \beta_j^d\right) = 0$ and the control law is reduced

to

$$\omega_i = \sum_j \beta_j^a \beta_j^d \frac{-2\pi c^{max}}{d_{ij}} < \frac{-2\pi c^{max}}{d_{ij}} < 0. \tag{13.8}$$

The change of heading during time Δt of robot i can be computed by integrating it:

$$|\Delta\theta_i| = \left| \int_{t^c-\Delta t}^{t^c} \omega_i \, dt \right| \geq \left| \int_{t^c-\Delta t}^{t^c} \frac{-2\pi c^{max}}{d_{ij}} \, dt \right|. \tag{13.9}$$

As the robot is going to collide, it will always move towards the other one during the time $t \in [t^c - \Delta t, t^c]$, the distance d_{ij} is continually decreasing from the distance r^c. Equation 13.9 can be rewritten as:

$$|\Delta\theta_i| \geq \left| \int_{t^c-\Delta t}^{t^c} \frac{-2\pi c^{max}}{d_{ij}} \, dt \right| \geq \left| \int_{t^c-\Delta t}^{t^c} \frac{-2\pi c^{max}}{2r^c} \, dt \right| = \left| \frac{-2\pi c^{max}}{2r^c} \Delta t \right| = \pi. \tag{13.10}$$

Thus the robot does more that a π turn during the time $t \in [t^c - \Delta t, t^c]$ contradicting the affirmation that the robot collides with the other robot in front of it. It will thus never reach the collision point because it will move away from that point before t^c. ■

During the whole proof, the case of robot j was not considered. Only mattered the fact that robot i will not travel further than a distance r^c towards robot j. We will now discuss the possible behaviors of robot j. Robot j can have four different behaviors depending on its initial heading. In the first case, the robot j is also moving towards the collision point. In that case, its behavior will be the same as robot i; it will perform a collision avoidance maneuver to the right, and no collision is possible. The second possible case is when robot j is moving away from robot i with an angle $|\phi_{ij}| > \alpha^s + \epsilon$ with ϵ such that

$$|\Delta\theta_j| = \left| \int_{t^c-\Delta t}^{t^c} \omega_j \, ds \right| = \int_{t^c-\Delta t}^{t^c} |K^n \left(\theta_j^t - \theta_j\right) \, ds| = \epsilon << \pi - \alpha^s. \tag{13.11}$$

In this case, $\omega_j = K^n \left(\theta_j^t - \theta_j\right)$ because the robot i is out of the field of view of robot j and thus $\beta_{a_i}\beta_{d_i} = 0$. The robot j will always have $|\phi_{ji}| > \alpha^s$ and will keep moving away from robot i while robot i is performing its collision avoidance maneuver. Thus no collision is possible. The third case is defined by $-\frac{\pi}{2} > \phi_{ji} \geq -\alpha^s - \epsilon$. Then the robot j will not rotate in a way that will lead to $\phi_{ji} > -\frac{\pi}{2}$ because at $\phi_{ji} = -\frac{\pi}{2}$ the avoidance term will bring it back in the region of $-\frac{\pi}{2} > \phi_{ji} \geq -\alpha^s - \epsilon$. Thus no collision is possible in this case either. The last possible case happens when $\frac{\pi}{2} < \phi_{ji} \leq \alpha^s + \epsilon$. Two outcomes are then possible. On the one hand, $\phi_{ji} > \frac{\pi}{2}$ during the time robot i performs its avoidance maneuver, robot j will move away from the collision during time Δt. On the other hand, $\phi_{ji}(t) < \frac{\pi}{2}$ during $t \in [t^c - \Delta t, t^c]$ in which case one can choose a new $\Delta t' \in]0, \Delta t[$ and reuse the proof proposed above with robot j instead of robot i. Note that now the situation will always be classifiable as one of the first three scenarios because to get to the fourth one robot i needs to cross the region with $|\phi_{ij}| > \alpha^s$ for which the rotation is very small during time $t \in [t^c - \Delta t', t^c]$.

This collision avoidance algorithm also works for more than two robots. If multiple robots are surrounding a given robot, that robot will spin until it finds a passage through which it can escape. This spinning effect can be seen as an emulation of a full stop as the robot remains in a small area, which is alike the second guideline of Chapter 10: consider the case where, due to inertia, the turn rate $\dot{\theta}_i$ is approximately constant. Then the heading of the robot can be approximated as $\theta_i = ut$. As a result, the trajectory of the robot is approximately a circle with a radius of:

$$r^c = \|\mathbf{p}_i(t)\| \approx \left\| \int c \begin{bmatrix} \cos(\omega t) \\ \sin(\omega t) \end{bmatrix} dt \right\| = \left\| \frac{c}{\omega} \begin{bmatrix} \sin(\omega t) \\ -\cos(\omega t) \end{bmatrix} \right\| = \frac{c}{|\omega|}. \tag{13.12}$$

It is assumed here, without loss of generality, that the circle is centered at the origin. Approximating the average turn rate $\dot{\theta} = \frac{-\pi c^{max}}{d_{ij}}$ with d_{ij} constant, we obtain the expected result: As the distance d_{ij} between the robots decreases, the turn rate command u increases and the radius of the circles decreases:

$$\lim_{d_{ij}\to 0} r^c = \lim_{d_{ij}\to 0} \frac{c}{|\omega|} = \lim_{d_{ij}\to 0} \frac{c d_{ij}}{2\pi c^{max}} = 0. \tag{13.13}$$

Despite its non-zero velocity, the robot's trajectory is contained is an increasingly smaller region as the distance to another robot decreases, effectively stopping its motion. But this behavior is not reasonable as it requires from the robot extremely high turn rate capabilities.

13.2 Avoiding Local Minimum

In this section we show why the change in speed is necessary. The result will be discussed afterwards. Only the case of two robots interacting is considered.

In the context of this work, a local minimum means that two robots do not turn anymore although at least one of them is not following its desired direction. An example of this situation is shown in Figure 13.2. We will show that this configuration is stable. It is not a mathematical proof but rather an approximation to help understand what is happening. This deadlock happens when one robot is moving in the rear of another one and the avoidance turning rate equals the goal tracking turning rate, or differently stated:

$$\dot{\theta}_i = K^n \left(1 - \beta_j^a \beta_j^d\right)\left(\theta_i^t - \theta_i\right) + \beta_j^a \beta_j^d \frac{-\pi c^{max}}{d_{ij}} = 0. \tag{13.14}$$

A first condition to have a local minimum can be derived directly from Equation 13.14. Indeed, writing it differently we obtain:

$$\left(\theta_i^t - \theta_i\right) = \frac{1}{K^n\left(1 - \beta_j^a \beta_j^d\right)} \beta_j^a \beta_j^d \frac{\pi c^{max}}{d_{ij}} > 0. \tag{13.15}$$

As $K^n > 0$ and $0 \le \beta_j^a \beta_j^d \le 1$ the value of $\left(\theta_i^t - \theta_i\right)$ is always positive. Thus the local minimum

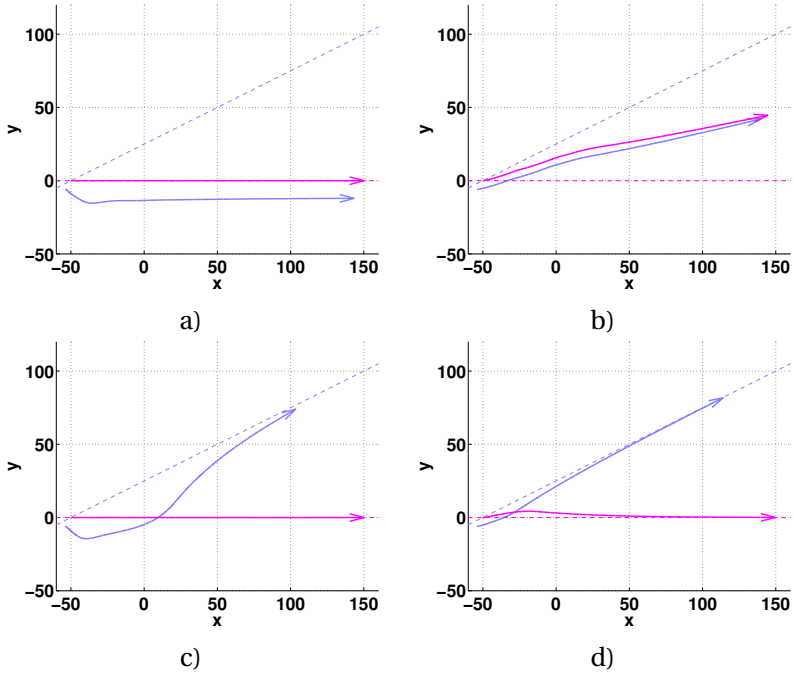


Figure 13.2 – Illustration of a scenario where two robots move in the same direction even if they do not have the same destination. The arrow heads show the direction of motion. The dashed lines show the desired courses. a) The proposed algorithm in the case the forward speed is constant. b) The same scenario for the algorithm presented in [96]. c) Same algorithm as in a) but the robots can decrease their speed. d) Same algorithm as in b) but the robots can decrease their speed.

forces the robot to deviate from its desired direction towards the right.

To show that the state resulting from Equation 13.14 is a local minimum, consider the second time derivative of θ_i,

$$\ddot{\theta}_i = \frac{\mathrm{d}}{\mathrm{d}t}\left[K^n\left(1-\beta_j^a\beta_j^d\right)\left(\theta_i^t-\theta_i\right)+\beta_j^a\beta_j^d\frac{-\pi c^{max}}{d_{ij}}\right]. \tag{13.16}$$

To be able to get the results, the following assumptions were adopted:

Assumption 1: Both robots move at the same constant speed $c_i = c_j = c^{max}$. This assumption will be relaxed later in this section when the means to avoid this local minimum will be discussed. The control law given by Equation 13.7 uses the result from this section.

Assumption 2: The desired heading θ^t does not vary in time. This is reasonable if the change in target heading is much slower than the time an robot takes to converge to the local minimum.

Assumption 3: The two robots start with similar headings so that $\Delta\theta = \theta_i - \theta_j$ is small and that the approximation $\Delta\theta^2 \approx 0$ holds.

Assumption 4: The robot j is not influenced by robot i because robot i is not in the FOV of

robot j. As consequence $\dot{\theta}_j = 0$ and $\ddot{\theta}_j = 0$.

The derivation of the local minimum will go as follows. We will first get the time derivative of Equation 13.14. It will be followed by the time derivative of the barrier functions and the sensing data. Then the pieces will be put together to get a second order differential equation. By approximating the obtained differential equation by a linear system we will discuss the convergence of the local minimum depending on the spatial configuration of the two robots. Let's first consider the first term of Equation 13.16:

$$\frac{\mathrm{d}}{\mathrm{d}t}\left[K^n\left(1-\beta_j^a\beta_j^d\right)\left(\theta^t-\theta_i\right)\right] = -K^n\left[\frac{\mathrm{d}}{\mathrm{d}t}\left[\beta_j^a\beta_j^d\right]\left(\theta^t-\theta_i\right)+\left(1-\beta_j^a\beta_j^d\right)\dot{\theta}_i\right]. \tag{13.17}$$

The second term of Equation 13.16 is:

$$\frac{\mathrm{d}}{\mathrm{d}t}\beta_j^a\beta_j^d\frac{-\pi c^{max}}{d_{ij}} = \frac{-\pi c^{max}}{d_{ij}}\left[\frac{\mathrm{d}}{\mathrm{d}t}\left[\beta_j^a\beta_j^d\right]-\beta_j^a\beta_j^d\frac{\dot{d}_{ij}}{d_{ij}}\right]. \tag{13.18}$$

The time derivative of the barrier function is obtained by using partial derivatives.

$$\frac{\mathrm{d}}{\mathrm{d}t}\beta_j^a\beta_j^d = \beta_j^a\frac{\partial\beta_j^d}{\partial d_{ij}}\dot{d}_{ij}+\beta_j^d\frac{\partial\beta_j^a}{\partial\phi_{ij}}\dot{\phi}_{ij}+\frac{\partial\beta_j^a\beta_j^d}{\partial t}. \tag{13.19}$$

Note that $\frac{\partial\beta_j^a\beta_j^d}{\partial t} = 0$ because both functions are time invariant. Because β is monotonic increasing we have:

$$\frac{\partial\beta_j^a}{\partial\phi_{ij}}\begin{cases} = 0 & \text{if } |\phi_{ij}| \le \frac{\pi}{2} \\ < 0 & \text{if } \frac{\pi}{2} < \phi_{ij} \le \alpha^s \\ > 0 & \text{if } -\frac{\pi}{2} > \phi_{ij} \ge -\alpha^s \\ = 0 & \text{otherwise} \end{cases} \tag{13.20}$$

$$\frac{\partial\beta_j^d}{\partial d_{ij}}\begin{cases} = 0 & \text{if } d_{ij} \le r^a \\ < 0 & \text{if } r^a < d_{ij} \le r^s \\ = 0 & \text{otherwise} \end{cases}. \tag{13.21}$$

The time derivatives of the sensor values $\dot{d}_{ij}$ and $\dot{\phi}_{ij}$ needs to be further developed. The time derivative $\dot{d}$ is computed as follows :

$$\dot{d}_{ij} = \frac{1}{\|\Delta\mathbf{p}\|}\left(\Delta x\Delta\dot{x}+\Delta y\Delta\dot{y}\right), \tag{13.22}$$

with $\Delta\mathbf{p} = \mathbf{p}_j - \mathbf{p}_i$. Using Assumption 1 the position difference's change rate $\Delta\dot{\mathbf{p}} = [\Delta\dot{x}, \Delta\dot{y}]^T$

becomes:

$$\Delta\dot{x} = c^{max}\left(\cos\theta_j - \cos\theta_i\right) = -2c^{max}\sin\left(\tfrac{\theta_j+\theta_i}{2}\right)\sin\left(\tfrac{\theta_j-\theta_i}{2}\right) \tag{13.23}$$

$$\Delta\dot{y} = c^{max}\left(\sin\theta_j - \sin\theta_i\right) = 2c^{max}\cos\left(\tfrac{\theta_j+\theta_i}{2}\right)\sin\left(\tfrac{\theta_j-\theta_i}{2}\right). \tag{13.24}$$

Equations 13.23 and 13.24 can be approximated with $2\sin\left(\frac{\theta_j-\theta_i}{2}\right) \approx -\Delta\theta$ using Assumption 3. Equation 13.22 can now be approximated by:

$$\dot{d}_{ij} \approx -\frac{1}{\|\Delta\mathbf{q}\|}c^{max}\Delta\theta\left(-\Delta x\sin\bar{\theta} + \Delta y\cos\bar{\theta}\right), \tag{13.25}$$

with $\bar{\theta} = \frac{\theta_i+\theta_j}{2}$. Because the measured bearing angle of robot i in the world frame is $\phi_{ij}+\theta_i$ we have that $\Delta\mathbf{p} = \|\Delta\mathbf{p}\|[\cos\left(\phi_{ij}+\theta_i\right), \sin\left(\phi_{ij}+\theta_i\right)]^T$ and thus:

$$\dot{d}_{ij} \approx c^{max}\begin{bmatrix}\sin\bar{\theta}\\ -\cos\bar{\theta}\end{bmatrix}^T\begin{bmatrix}\cos\left(\phi_{ij}+\theta_i\right)\\ \sin\left(\phi_{ij}+\theta_i\right)\end{bmatrix}\Delta\theta. \tag{13.26}$$

$\dot{\phi}_{ij}$ is computed in a similar way:

$$\begin{aligned}\dot{\phi}_{ij} = &\quad \frac{\mathrm{d}}{\mathrm{d}t}\mathrm{atan2}\left(\Delta y, \Delta x\right) = \frac{-\Delta y\Delta\dot{x} + \Delta x\Delta\dot{y}}{\Delta x^2 + \Delta y^2}\\ \approx &\quad -\frac{c^{max}\left(\Delta x\cos\bar{\theta} + \Delta y\sin\bar{\theta}\right)}{\|\Delta\mathbf{p}\|^2}\Delta\theta\\ = &\quad -\frac{c^{max}}{\|\Delta\mathbf{p}\|}\begin{bmatrix}\cos\bar{\theta}\\ \sin\bar{\theta}\end{bmatrix}^T\begin{bmatrix}\cos\left(\phi_{ij}+\theta_i\right)\\ \sin\left(\phi_{ij}+\theta_i\right)\end{bmatrix}\Delta\theta.\end{aligned} \tag{13.27}$$

Using the results above the terms in Equations 13.17 and 13.18 can be rewritten as:

$$-\beta_j^a\beta_j^d\frac{\dot{d}_{ij}}{d_{ij}} \approx A_1\Delta\theta \tag{13.28}$$

$$\beta_j^a\frac{\partial\beta_j^d}{\partial d_{ij}}\dot{d}_{ij} \approx A_2\Delta\theta \tag{13.29}$$

$$\beta_j^d\frac{\partial\beta_j^a}{\partial\phi_{ij}}\dot{\theta}_{d_{ij}} \approx A_3\Delta\theta, \tag{13.30}$$

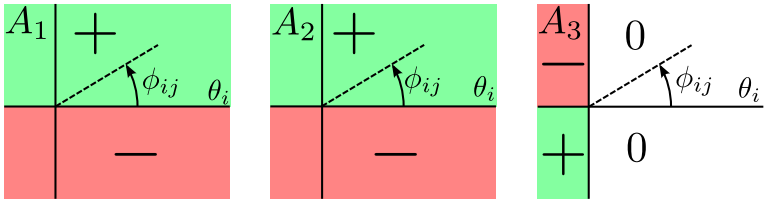


Figure 13.3 – Signs of A_1, A_2 and A_3 as function of ϕ_{ij} angles if $\theta_i = \theta_j$.

with A_1, A_2 and A_3 defined as:

$$A_1 = -\frac{c^{max}}{d_{ij}} \beta_j^a \beta_j^d \begin{bmatrix} \sin\bar{\theta} \\ -\cos\bar{\theta} \end{bmatrix}^T \begin{bmatrix} \cos(\phi_{ij} + \theta_i) \\ \sin(\phi_{ij} + \theta_i) \end{bmatrix} \tag{13.31}$$

$$A_2 = c^{max} \beta_j^a \frac{\partial \beta_j^d}{\partial d_{ij}} \begin{bmatrix} \sin\bar{\theta} \\ -\cos\bar{\theta} \end{bmatrix}^T \begin{bmatrix} \cos(\phi_{ij} + \theta_i) \\ \sin(\phi_{ij} + \theta_i) \end{bmatrix} \tag{13.32}$$

$$A_3 = -\frac{c^{max}}{\|\Delta\mathbf{p}\|} \beta_j^d \frac{\partial \beta_j^a}{\partial \phi_{ij}} \begin{bmatrix} \cos\bar{\theta} \\ \sin\bar{\theta} \end{bmatrix}^T \begin{bmatrix} \cos(\phi_{ij} + \theta_i) \\ \sin(\phi_{ij} + \theta_i) \end{bmatrix}. \tag{13.33}$$

The signs of A_1, A_2 and A_3 are determined by the result of the vector multiplication. As the scalar product can be related to the cosine of the angles between those vectors, the sign of A_1, A_2 and A_3 is determined by the angle $\phi_{ij} + \theta_i - \bar{\theta} \approx \phi_{ij}$. This is illustrated in Figure 13.3. Using Equations 13.32 and 13.33, Equation 13.17 can be rewritten as:

$$\frac{\mathrm{d}}{\mathrm{d}t}\left[K^n\left(1-\beta_j^a\beta_j^d\right)\left(\theta_i^t-\theta_i\right)\right] = -K^n\left[\left(1-\beta_j^a\beta_j^d\right)\dot{\theta}_i + (A_2+A_3)\left(\theta_i^t-\theta_i\right)\Delta\theta\right]. \tag{13.34}$$

Because θ_j is constant the term $\left(\theta_i^t - \theta_i\right)$ of Equation 13.34 can be rewritten as $\left(\theta_i^t - \theta_j + \theta_j - \theta_i\right) = \left(\theta_i^t - \theta_j - \Delta\theta\right)$. Using Assumption 3 as in Equation 13.25, the term $(A_2 + A_3)\left(\theta_i^t - \theta_i\right)\Delta\theta$ of Equation 13.34 can be written as:

$$\begin{aligned} &(A_2 + A_3)\left(\theta_i^t - \theta_i\right)\Delta\theta \\ =\ &(A_2 + A_3)\left(\theta_i^t - \theta_j - \Delta\theta\right)\Delta\theta \\ =\ &(A_2 + A_3)\left(\left(\theta_i^t - \theta_j\right)\Delta\theta - \Delta\theta^2\right) \\ \approx\ &(A_2 + A_3)\left(\theta_i^t - \theta_j\right)\Delta\theta, \end{aligned} \tag{13.35}$$

because $\Delta\theta^2$ is small. We can now rewrite Equation 13.16 using the Assumption 4 that $\dot{\theta}_j = 0$ leading to $\Delta\dot{\theta} = \dot{\theta}$ and $\Delta\ddot{\theta} = \ddot{\theta}$:

$$\Delta\ddot{\theta} + a_1\Delta\dot{\theta} + a_2\Delta\theta \approx 0, \tag{13.36}$$

with a_1 and a_2 defined as:

$$a_1 = \quad K^n \left(1 - \beta_j^a \beta_j^d\right) \tag{13.37}$$

$$a_2 = \quad \left[(A_2 + A_3)\, BK^n + (A_1 + A_2 + A_3)\, \frac{2\pi c^{max}}{d_{ij}}\right] \tag{13.38}$$

$$B = \quad \theta_i^t - \theta_j, \tag{13.39}$$

which has a stable solution if $\frac{-a_1 \pm \sqrt{a_1^2 - 4a_2}}{2} < 0$ which is true if both $a_1 > 0$ and $a_2 > 0$. The system is unstable otherwise. As a_1 is always positive, the stability of local minimum comes essentially from a_2. The condition $a_2 > 0$ happens when $A_1, A_2 > 0$, $A_3 = 0$ and $B > 0$ which corresponds to the case where robot i sees the other robot on its front left and robot j is moving to the right with respect to the goal. Note that such analysis is not possible for robot j (in the front) because the sign of A_3 is different from A_1 and A_2. This local minimum results from the combination of a limited field of view and both robots moving at the same speed. Indeed when near the local minimum, robot i has only little correction to perform to align with its desired heading. But to align, robot i needs to get closer to robot j, which would increase its avoidance reaction. As both robots have the same speed, the distance stays constant. robot i is thus in a deadlock. The limited field of view leads to robot j (front one) to ignore robot i (in the rear of robot j). The solution will thus not come from robot j either. In the case of [96] the local minimum arises from their swirling effect reduction diminishing the radius of avoidance area at the rear of the aircraft. This is analogue to our algorithm where the front aircraft ignores the rear one, although for different reasons. This local minimum problem is solved by adding a change in speed. As the robot behind is slower, the distance between the two will increase leading to their separation. For example a decrease in speed of 20% allows for a quick escape out of the local minimum as shown in Figure 13.2. The same decrease in speed can also be used with the algorithm presented in [96] to avoid the same problem. If both are exactly side-by-side and have their heading in the same direction, they will slow down equally. But because one robot is always on the left side of the other one (thus sees the later on its right), at least one of them is in an unstable situation ($A_1, A_2 \leq 0$ and $A_3 = 0$) and will separate itself if the sensors are noisy.

13.3 Results

13.3.1 Simulations

The PF approach was simulated using the two scenarios presented in Chapter 10. The simulations was implemented in Matlab as a microscopic point simulation (no physical simulation) with perfect kinematics and noiseless sensing. The resulting trajectories are presented in Figures 13.4 and 13.5. In this case only the FOV aware version of the algorithm is presented because it is not possible to design a version of the algorithm that does not respect the sensor-

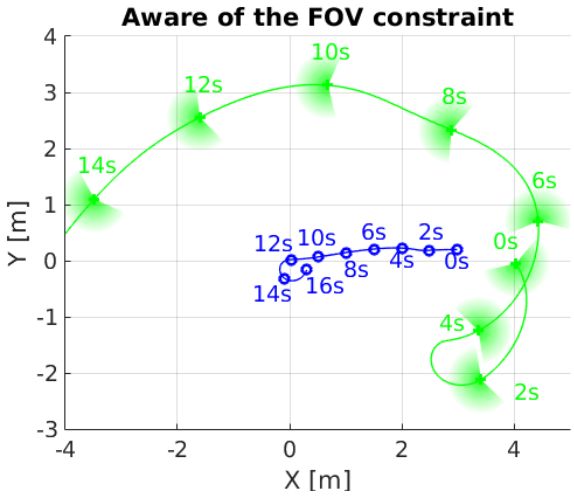


Figure 13.4 – Trajectories obtained with the first scenario. The solid blue and green lines are robots' trajectories. The markers and numbers give an indication of the timing of the trajectories. The FOV of some robots at the timings indicated by numbers are represented by circular sectors. The color of the circular sectors match the color of the trajectories. The FOV range is not to scale.

constrained set.

Significant differences between this heading-based algorithm and the pure PF-based algorithm from Chapter 12 can be observed. First, the heading-based algorithm performs a large detour in the first scenario, as shown in Figure 13.4. This is because the avoidance behavior only avoids on one side, which is in this case the opposite side of the other robot. This propensity of making detours can lead to inefficient, and even hazardous trajectories on real systems, such as rapid, poorly controlled spinning behaviors. In the second scenario too (Figure 13.5), the heading-based algorithm tend to do detours compared to the algorithm presented in Chapter 12. But this is compensated by an almost constant speed of the robots, with consequence that the heading-based algorithm is faster to reach its goal compared to the purely PF-based. Furthermore, an oscillation behavior can be observed for the two middle robots in Figure 13.5, also caused by the way the collision avoidance is handled, and partially a consequence of the limited FOV (as other robots appear and disappear from the FOV as the robot rotates).

13.3.2 Real-World Experiments

This algorithm has been validated with real robot experiments, using our quadrotor platform described in Chapter 2 with the visual sensing system described in Chapter 5. The full system as well as the obtained results were published in [35].

A few modifications had to be made to the algorithm to be validated experimentally. First, the heading-based algorithm assumes a unicycle-like robot, which is significantly different from the quasi-holonomicity of quadrotors. But because of the high maneuverability of the

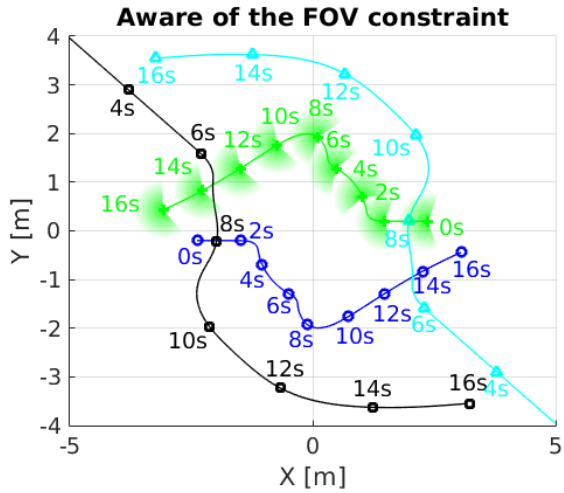


Figure 13.5 – Trajectories obtained with the second scenario. The solid blue, black, green and cyan lines are robots' trajectories. The markers and numbers give an indication of the timing of the trajectories. The FOV of some robots at the timings indicated by numbers are represented by circular sectors. The color of the circular sectors match the color of the trajectories. The FOV range is not to scale.

quadrotor, it is possible to emulate a unicycle trajectory using the following equation that links the control law output and the desired velocity of the quadrotor:

$$\begin{bmatrix} v_{x_i} \\ v_{y_i} \end{bmatrix} = \begin{bmatrix} c\cos(\theta_i) \\ c\sin(\theta_i) \end{bmatrix}, \tag{13.40}$$

with c the desired speed given by the algorithm. To assure a as faithful as possible emulation of the unicycle dynamics, the inertia of the quadrotor has to be compensated by adding a tangential acceleration:

$$\mathbf{a}_t = u_i c_i \begin{bmatrix} -\sin\theta_i \\ \cos\theta_i \\ 0 \end{bmatrix}. \tag{13.41}$$

This tangential acceleration is then added to the overall control acceleration in the main controller. Second, the β_j^d function is modified on the sides in order to reduce the oscillations:

$$\beta_j^d = \beta\left(\frac{r^s - 0.4(r^s - r^a)\frac{|\phi_{ij}|}{\alpha^s} - d_{ij}}{r^s - 0.4(r^s - r^a)\frac{|\phi_{ij}|}{\alpha^s} - r^a}\right), \tag{13.42}$$

$$\beta_j^a = \beta\left(\frac{\alpha^s - |\phi_{ij}|}{\alpha^s - \frac{\pi}{2}}\right). \tag{13.43}$$

Nonetheless, the guarantee of collision avoidance remains as the proof of it is based on the region where $\beta_j^d \beta_j^a = 1$, which remains the same as in [37].

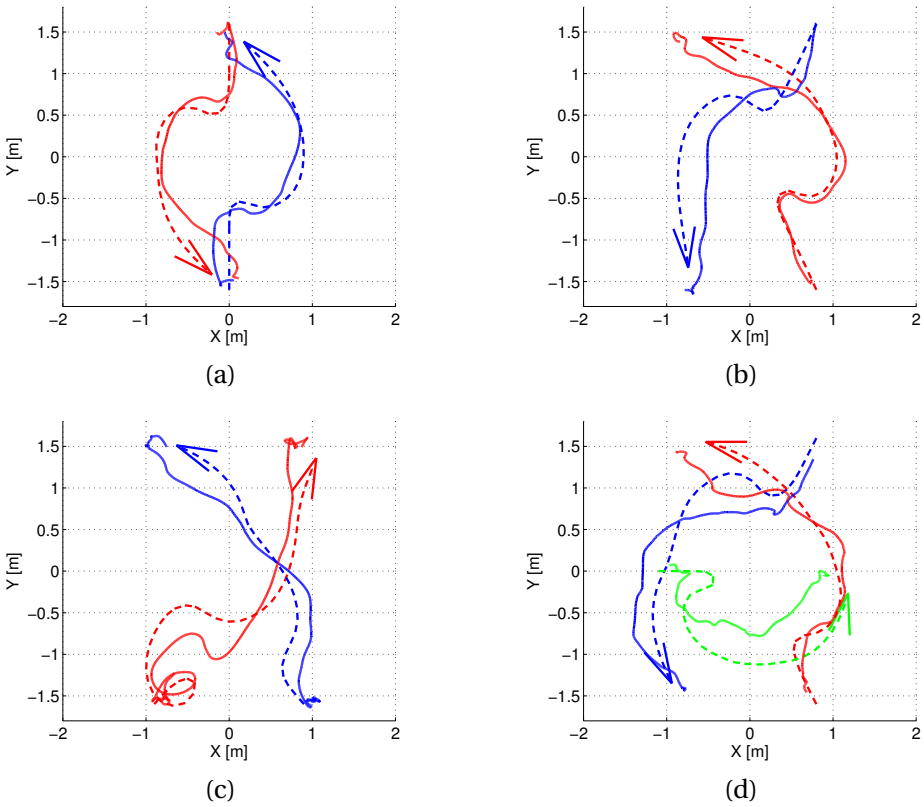


Figure 13.6 – Plots (a), (b), (c) and (d) show the results obtained during scenarios “Head-on", “Cross", “Side" and “Three" respectively. All solid lines corresponds to real data and all dashed lines to simulated data. The trajectories of each quadrotor are depicted in a different color (blue, red, and green for the first, second, and third quadrotor, respectively).

Four scenarios were designed to validate the proposed system. They are shown in Figure 13.6. In the first scenario, called “Head-on", a quadrotor starts at $[0,-1.6,1.2]m$ and aims to go to $[0,1.6,1.2]m$, and vice-versa for the other quadrotor. The second scenario, called “Cross", is a crossing scenario where one quadrotor starts at $[0.8,1.6,1.2]m$ and with the target position at $[-0.8,-1.6,1.2]m$. The other quadrotor starts at $[0.8,-1.6,1.2]m$ and aims to go to $[-0.8,1.6,1.2]m$. In the third scenario, called “Side", one quadrotor starts at $[0.9,-1.6,1.2]m$ and aims to go to $[-0.9,1.6,1.2]m$. The other quadrotor starts at $[-0.9,-1.6,1.2]m$ and aims to go to $[0.9,1.6,1.2]m$. The final scenario, named “Three", involves three quadrotors to show the capability of our system in handling multiple encounters simultaneously. Two quadrotors have the same start and goal positions as in the second scenario, and the third quadrotor starts at $[-1,0,1.2]m$ and the goal at $[1,0,1.2]m$.

Representative trajectories for the four scenarios are shown in Figure 13.6 (solid lines). Fifty collision course runs (using real quadrotors) have been performed for each scenario. The quadrotors never collided during the 200 runs. Figure 13.6 also shows, overlapped to real robot data, trajectories obtained with a point-mass, microscopic simulator implemented in

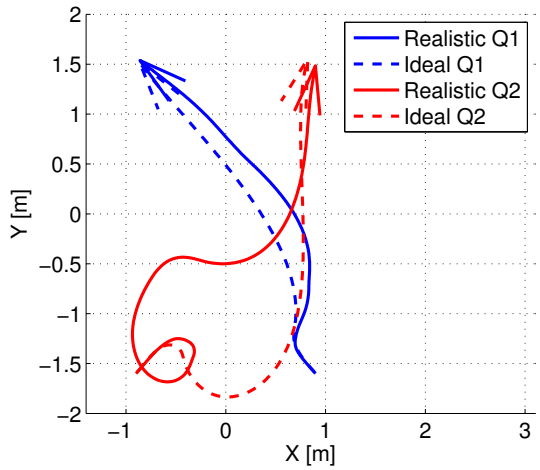


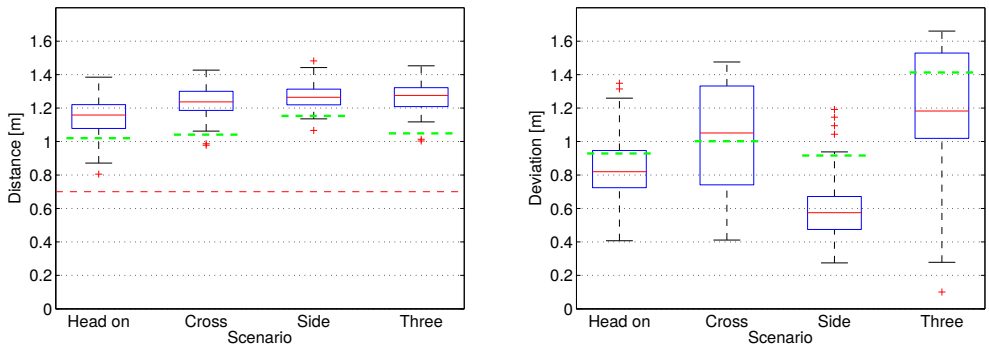
Figure 13.7 – Illustration of the trajectories of the noise-free and lag-free model (in dashed lines) and the realistic model that includes noise, lag, inertia and saturation (solid lines) for the "Side" scenario.

Matlab and already leveraged in [37]. In order to shed further lights on the various source of errors, in contrast to our previous work, we have extended our simulator to include real world effects in terms of sensing, actuation, and computation. In particular, the simulation has been adapted to include a Gaussian noise on the sensing of 0.25 meters RMS, a sensing lag of 200 milliseconds and a control lag of 50 milliseconds. Furthermore, the model of the unicycle in the simulation has been adapted to include inertia. The aircraft in the simulator has thus the dynamical equation:

$$\begin{bmatrix} \dot{x}_t \\ \dot{y}_t \\ \dot{\psi}_t \\ \dot{\omega}_t \end{bmatrix} = \begin{bmatrix} c\cos\psi \\ c\sin\psi \\ \omega_t \\ -3(\omega_t - u) \end{bmatrix}, \tag{13.44}$$

instead of Equation 13.1. A saturation on yaw acceleration of $3\frac{rad}{s^2}$ was also implemented. The obtained trajectories are less smooth than what was presented in [37] for two reasons. First, the sensing range r^s is smaller compared to the vehicle size than in [37] which results in sharper trajectories. Second, our system is now affected by real-world imperfections (i.e., noise, lag). This can drastically change the trajectory of the quadrotors as for the "Side" scenario the left quadrotor prefers to do a full 360° turn to avoid the other quadrotor. The interplay of those real-world effects can drastically modify obtained trajectories, as illustrated in Figure 13.7. It appears to be mostly due to the lag (both in sensing and control) as the 360° turn behavior does not appear if the sensing noise is applied without any lag.

This sensibility to lag is inherent to the collision avoidance algorithm. Indeed, when another quadrotor enters the collision avoidance zone of the host quadrotor, the later will turn until there is an equilibrium between the avoidance and the navigation component. This equilib-



(a) (b)

Figure 13.8 – Plot (a) shows the boxplots (red line is the median, the blue box represents first and last quartiles) of the minimal distance between real quadrotors for 50 experiments for all four scenarios. The red dashed line represents the distance for which quadrotors would be considered to have collided. The green dashed line represents the average performance of the simulation (realistic model) for 50 experiments. Plot (b) shows the boxplots of the maximum deviation from desired trajectory for 50 experiments using real quadrotors for all four experiments. The green dashed line represents the average performance of the simulation (realistic model) for 50 experiments.

rium happens mostly because the angular transition function β_j^a tends to 0 (as the bearing angle changes faster than the distance). But the region in which β_j^a ranges between 0 and 1 corresponds only to 20° in bearing. Since there is lag in sensing and control, the quadrotor overshoots the yaw angle. At overshooting, the quadrotor will switch behavior and go back to navigation. In the "Side" scenario case, the overshoot brings the quadrotor to a state where following the navigation function leads to a 360° turn.

The minimum horizontal distance between the aircraft during the runs are shown in Figure 13.8. The average minimum distance obtained in simulation is lower by approximately 20 centimeters for all scenarios. This might be due to unmodeled phenomenons, such as the airflow generated laterally by the quadrotors that tends to push them away from each other. The distances are well above the distance for which the quadrotors will be considered to have collided. As a second metric, the maximum deviation of the quadrotors from their desired course (virtual line between start point and goal) over a run has also been recorded (Figure 13.8b). Even if the simulations are quite close to the real trajectory for three out of four scenarios, there is a significant difference for the "Side" scenario resulting on a larger delta between the two mean results on the metric of Figure 13.8b. This is because, in both simulation and reality, the two behaviors illustrated in Figure 13.7 are present but not with the same amount. In reality, the most common case is where the quadrotor does a 360° turn. In simulation, the smoother trajectory is more common. The switching behavior is hard to capture, and has significant effect on the metric.

13.4 Discussion

One main drawback of this algorithm is that it always turn right. This is implemented to follow the “right of way” rule used in aeronautics. But the implementation is too simplistic, and leads to inefficient and hazardous behaviors such as spinning and detours. To solve those issues, a more explicit and deliberate implementation of the “right of way” rule is needed. This may be achieved by asymmetrically shaping the β function or adding a RB layer on top (i.e., if a spinning behavior is detected, switch to another behavior).

UAVs navigate in a three dimensional space. Depending on the configuration of the UAVs, it might be more effective to change altitude instead of doing a detour in the horizontal plane. Up to now, our collision avoidance algorithms do not take advantage of this fact. In next chapter, we will present a collision avoidance algorithm that uses all three dimensions.

Summary

In this chapter, a heading-based collision avoidance algorithm is presented. First, the dynamics of the system is described, followed by the proposed control law. Then a proof of collision avoidance is given, as well as a discussion on the possible local minima and means to avoid them. Finally, simulation and experimental validation of the collision avoidance algorithm is presented.

14 Potential-Fields-Based Avoidance in Three Dimensions

FOR UAVs, it is possible to take advantage of the vertical dimension to avoid collisions. Some aircraft, such as fixed-wing airplanes, are more maneuverable vertically than in the horizontal plane. The downside of vertical collision avoidance is that there is no simple "right has priority" rule as there is for the horizontal plane. There is thus a risk that, without communication, the two robots perform the same maneuver (e.g., both decide to go up), making the situation worse. Some collision avoidance systems (such as the TCAS used in aircraft) rely on communication to solve the issue. As we assume no communication between the aircraft in this work, we will present an algorithm that need to consider all three dimensions concurrently (i.e., combined horizontal and vertical maneuver), to guarantee avoidance.

This work build on top of the algorithm presented in Chapter 13, notwithstanding the issues that have been unraveled since. As a consequence, the algorithm presented in this chapter has many of the shortcomings of the algorithm of Chapter 13. Nonetheless, this chapter illustrates well the need of limiting the actuation based on the FOV, as well as showing that the notions presented in Chapter 10 can be easily applied to the third dimension as well. This work was published in [38].

14.1 Problem Statement

For this algorithm, the unicycle model is extended to the third dimension by adding a vertical component z that is controlled in velocity:

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} c\cos\theta \\ c\sin\theta \\ v_z \\ \omega \end{bmatrix}, \tag{14.1}$$

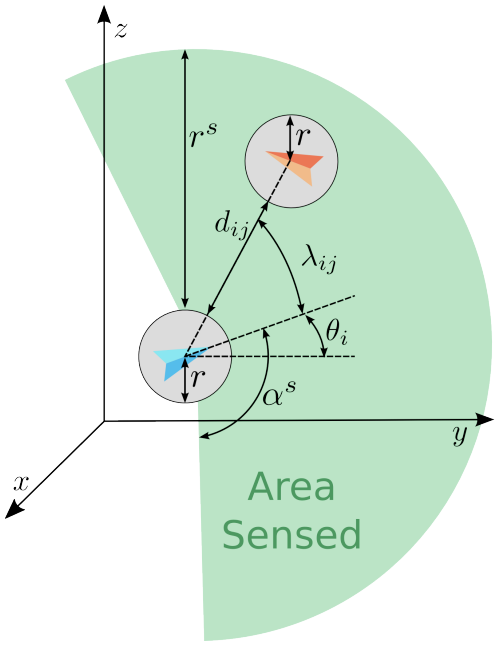


Figure 14.1 – Illustration of the angles and distances used in this chapter. The green area represents the area where aircraft j (in orange) is sensed by aircraft i (in blue). Note that the angles θ_i and λ_{ij} might not be in the same plane. The FOV is defined by α^s in all direction (thus both horizontally and vertically).

with z robot's height and v_z the vertical speed of the robot. Furthermore, $\mathbf{p}_i = [x_i, y_i, z_i]$ is the position of robot i and θ_i its heading (defined on the horizontal plane). The heading and any other angle considered in this chapter are always defined between $[-\pi;\pi]$ and any function returns angles in that interval. The aircraft can be controlled with the three input variables ω, v_z, and c that are the yaw turning rate, the vertical speed, and the forward speed, respectively. The horizontal speed c is constrained by:

$$0 < c^- \leq c \leq c^+, \tag{14.2}$$

with c^+ the maximum horizontal speed and $c^- = c^+ - \Delta c$ the minimum horizontal speed of the aircraft.

The notion of FOV is extended to the third dimension by rotating the 2D FOV around the camera axis, obtaining a sphere sector. Consider a range and bearing sensor that stays horizontal but with the heading aligned with the robot:

$$\begin{bmatrix} d_{ij} \\ \phi_{ij} \\ \lambda_{ij} \end{bmatrix} = \begin{bmatrix} \|\mathbf{p}_i - \mathbf{p}_j\| - 2r \\ \arctan\left(\frac{[0,0,1]^T \cdot (\mathbf{p}_i - \mathbf{p}_j)}{[\sin\theta_i, -\cos\theta_i, 0]^T \cdot (\mathbf{p}_i - \mathbf{p}_j)}\right) \\ \arccos\left(\frac{[\cos\theta_i, \sin\theta_i, 0]^T \cdot (\mathbf{p}_i - \mathbf{p}_j)}{\|\mathbf{p}_i - \mathbf{p}_j\|}\right) \end{bmatrix}, \tag{14.3}$$

then the FOV in three dimensions can be written as:

$$\mathbf{F}_i = \left\{ \mathbf{p} \;\middle|\; ||\mathbf{p} - \mathbf{p}_i|| < r_i^s, |\lambda_{ij}| \leq \alpha_i^s \right\}. \tag{14.4}$$

The angles are illustrated in Figure 14.1. Equations 14.4 is also possible for other definitions of λ_{ij}, such as not being horizontally aligned. In general, Equations 10.1 and 10.2 from Chapter 10 are still valid when used with the third dimension.

The proposed control scheme is based on PF theory but adapted to the FOV constraints of our system. The proposed PF is a combination of a function V_i^n that steers aircraft i to its destination, and another function V_{ij}^a that steers aircraft i away from another aircraft j. The function that steers aircraft i to destination is defined as:

$$V_i^n = K^n \|\mathbf{p}_i - \mathbf{p}_i^d\|^2, \tag{14.5}$$

but other functions are possible [96]. The collision avoidance function is defined as:

$$V_{ij}^a = \frac{K^a}{\|\mathbf{p}_i - \mathbf{p}_j\| - 2r}. \tag{14.6}$$

Both K^a and K^n are gains. Like for the other algorithms such as those presented in Chapters 12 and 13, the aircraft i is directed by the weighted sum of the negative gradient of the PFs described in Equations 14.5 and 14.6 in the following manner:

$$\nabla V_i^t = \left(1 - \max_j \left(\beta_j^a \beta_j^d\right)\right) \nabla V_i^n + \sum_j \beta_j^a \beta_j^d \nabla V_{ij}^a. \tag{14.7}$$

The weighting function β_j^a and β_j^d encode aircraft's sensors limitations, and are defined as:

$$\beta_j^d = \beta\left(\frac{r^s - d_{ij}}{r^s - r^a}\right), \tag{14.8}$$

$$\beta_j^a = \beta\left(\frac{\alpha^s - |\lambda_{ij}|}{\alpha^s - \frac{\pi}{2}}\right), \tag{14.9}$$

with r^a the distance at which an aircraft should only be avoiding, and the β function defined by Equation 12.17 in Chapter 12. From Equation 14.7 we also define the desired direction for aircraft i as:

$$-\nabla V_i^t = [x_i^t, y_i^t, z_i^t]^T. \tag{14.10}$$

As explained in Chapter 13, a change in forward speed c_i is required to mitigate deadlocks. Even if having two aircraft at exactly the same speed and height is unlikely, we reuse the same

control law to improve system's separation capability.

$$c_i = c^+ - \max_j\left(\beta_j^a \beta_j^d\right)\Delta c. \tag{14.11}$$

As the height dynamics is decoupled from the other state variables, it would be possible to directly apply the negative PF's gradient $-\nabla V_i^t$ to the vertical speed v_z. But due to FOV constraints, a saturation l^s on the vertical speed is required (see Proposition 5).

$$0 < l^s = c\sin\left(\alpha^s - \frac{\pi}{2} - \rho\right) < c\sin\left(\alpha^s - \frac{\pi}{2}\right), \tag{14.12}$$

and the maximum vertical speed is :

$$l^m = c^+\sin\left(\alpha^s - \frac{\pi}{2} - \rho\right), \tag{14.13}$$

with $0 < \rho < \alpha^s - \frac{\pi}{2}$ a margin angle. The absolute maximum speed is $c^{max} = \sqrt{(c^+)^2 + (l^m)^2}$. The saturation is applied to obtain the vertical speed:

$$v_z = \min\left(\max\left(K^z z_i^t, -l^s\right), l^s\right), \tag{14.14}$$

with K^z a strictly positive gain. The yaw control[1] is responsible of steering the aircraft away from threat on the horizontal plane. It switches between following the PF and a simple collision avoidance behavior.

$$\omega = K^\omega\left(1 - \max_j\left(\beta_j^a \beta_j^d\right)\right)\left(\theta^t - \theta_i\right) + \sum_j \beta_j^a \beta_j^d \frac{-2\pi c^+}{d_{ij}}. \tag{14.15}$$

with the target heading $\theta^t = \text{atan2}(y_i^t, x_i^t)$, and K^ω a gain.

It is now possible to give a proof of guaranteed collision avoidance. Proposition 5 is essentially the same proof as Proposition 1 of Chapter 10. For the sack of completeness, we present the earlier, less general variant of Proposition 1.

Proposition 5. An aircraft with FOV described by Equation 14.4, kinematics described by Equation 14.1 and the control law described by Equations 14.11, 14.13 and 14.15 will not move along a trajectory that would lead to a collision with another aircraft without them being able to sense each other. That is, there is no possible collision without at least one of

[1] The yaw control described by Equation 14.15 is slightly different from the work in [37]. This is to correct for a mistake in [37] where the turn rate is only half what would be needed to prove avoidance. The proof in [37] is also correct if the avoidance term in the yaw control is multiplied by two.

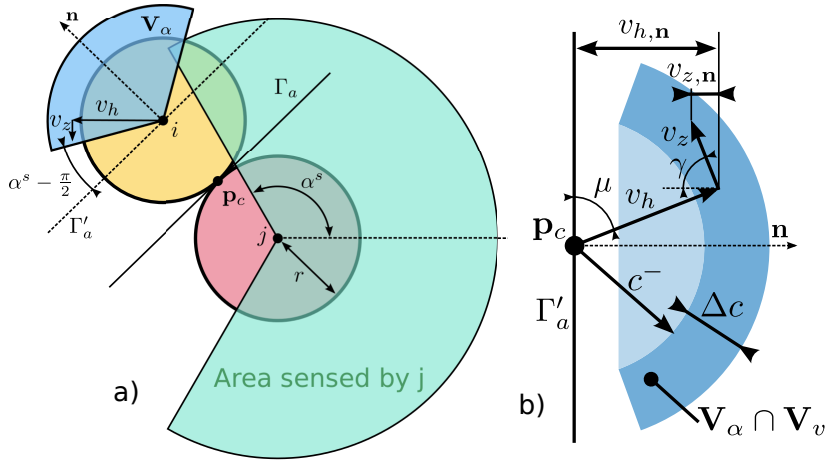


Figure 14.2 – Illustration of the variables involved in Proposition 5. a) Two aircraft about to collide, defining Γ_a and $\mathbf{V}_\alpha$. b) Computation of the projections of $\mathbf{v}_h$ and $\mathbf{v}_z$ on $\mathbf{n}$.

the two aircraft sensing the other aircraft.

Proof. Consider two UAVs i and j which are about to collide at point $\mathbf{p}_c$ while none of the two sensing the other one. Consider also the plane Γ_a that goes through $\mathbf{p}_c$ and is tangent to the two collision spheres of aircraft i and j. To collide, at least one of the two aircraft needs to move towards the plane Γ_a. We will prove that with the proposed control law it is impossible for any of the UAVs to move towards Γ_a without sensing the other UAV.

Without loss of generality, the proof can be done while fixing the frame of one of the aircraft, for example aircraft j as shown in Figure 14.2a. As none of the two can see each other, the point $\mathbf{p}_c$ is on the sphere sector of j that is not in the FOV of aircraft j. Placing $\mathbf{p}_c$ also defines the position of aircraft i relatively to aircraft j. Define $\mathbf{V}_\alpha$ as the cone formed by all camera directions for which aircraft i does not see aircraft j. Define also Γ'_a the plane parallel to Γ_a located at the center of i. Note that aircraft i not crossing Γ_a is equivalent to the center of i moving away from aircraft j. Because it is solely defined by the geometry of the FOV of j, $\mathbf{V}_\alpha$ is always the same relative to Γ'_a: it is a right circular cone with its apex centered on i, its axis normal to Γ'_a and an aperture of $\pi - 2\left(\alpha^s - \frac{\pi}{2}\right)$.

Because the horizontal velocity $\mathrm{v}_h = c_i\,[\cos\theta_i, \sin\theta_i, 0]^T$ is aligned with the camera, we have that $\mathrm{v}_h \in \mathbf{V}_\alpha \cap \mathbf{V}_v$ with $\mathbf{V}_v$ the spherical shell formed by the velocities between $[c^-, c^+]$. The shape of $\mathbf{V}_\alpha \cap \mathbf{V}_v$ is shown in Figure 14.2b. Because $\alpha^s > \pi/2$ the velocity v_h alone will never lead to a collision when both aircraft do not see each other and can only be due to the vertical velocity $\mathrm{v}_z = [0, 0, v_z]^T$. Define $\mathbf{n}$ the vector normal to Γ'_a and directed away from aircraft j, and project v_h and v_z to obtain $v_{h,\mathbf{n}}$ and $v_{z,\mathbf{n}}$ respectively. We want to show that:

$$\|v_{h,\mathbf{n}}\| > \|v_{z,\mathbf{n}}\| \quad \forall v_{h,\mathbf{n}} \in \mathbf{V}_\alpha \cap \mathbf{V}_v, v_z. \tag{14.16}$$

First, define the angle μ between the plane Γ'_a and the vector v_h. Note that $\pi > \mu \geq \alpha^s - \pi/2$. The left hand of Equation 14.16 is:

$$\|v_{h,\mathbf{n}}\| = \|\mathrm{v}_h\| \sin(\mu) = v_i \sin(\mu) > v_i \sin(\alpha^s - \pi/2). \tag{14.17}$$

Similarly, by defining the angle γ between $v_{z,\mathbf{n}}$ and $\mathbf{n}$, the right hand of Equation 14.16 becomes:

$$\|v_{z,\mathbf{n}}\| = \|\mathrm{v}_z\| \cos(\gamma) < \|\mathrm{v}_z\| \leq l^s = v_i \sin(\alpha^s - \frac{\pi}{2} - \rho), \tag{14.18}$$

with $\pi > \gamma \geq \alpha^s - \pi/2$. Combining Equations 14.17 and 14.18 the final result is obtained:

$$\|v_{h,\mathbf{n}}\| > v_i \sin(\alpha^s - \frac{\pi}{2}) > v_i \sin(\alpha^s - \frac{\pi}{2} - \rho) > \|v_{z,\mathbf{n}}\|. \tag{14.19}$$

Thus aircraft i subject to the control laws described by Equations 14.11, 14.13 and 14.15 will move away from aircraft j if unable to sense aircraft j and as a result will not be responsible for colliding. The same reasoning can be done for aircraft j and as a result both aircraft are unable to move towards the other aircraft while not sensing it.

■

Using Proposition 5 it will now be shown that two aircraft are unable to collide if they use the control law proposed in this chapter.

Proposition 6. Two aircraft with the kinematics given by Equation 14.1 and the control laws given by Equations 14.11, 14.13 and 14.15 will not collide.

Proof.

The proof is done in a similar way as proposed in [37], but care has to be taken because of the vertical component of aircraft's velocity. Consider two aircraft i and j about to collide at point $\mathbf{p}_c$ at time t_c. We will prove that with the proposed control law there is no trajectory that leads to point $\mathbf{p}_c$, that is, both the horizontal and vertical velocities will drive the aircraft away from point $\mathbf{p}_c$ before it is reached.

First, notice that at least one of the two aircraft has to move towards the collision point. Let's assume, without loss of generality, it is aircraft i. Two cases have to be studied: first, the case where the camera is pointing towards the collision point $\mathbf{p}_c$. As a result $\beta_j^a \beta_j^d = 1$. A second case is when $0 < \beta_j^a \beta_j^d < 1$.

For the first case, when $\beta_j^a \beta_j^d = 1$, the horizontal velocity is directed towards the collision point. We will prove that u_i will change the heading of the aircraft early enough so that the aircraft moves away from point $\mathbf{p}_c$ at time t_c. Consider the situation a short moment before the two aircraft collide and are separated by a small distance $2r^c$. Because the maximum

speed of the aircraft is c^+, the minimum time Δt before the two aircraft collide is given by:

$$\Delta t = \frac{r^c}{c^+}. \tag{14.20}$$

Because of $\beta_j^a \beta_j^d = 1$, the turning rate is bounded above by the interaction between aircraft i and j:

$$u_i = \sum_j \beta_j^a \beta_j^d \frac{-\pi c^+}{d_{ij}} < \frac{-\pi c^+}{d_{ij}} < 0. \tag{14.21}$$

The change in heading between time $t \in [t_c - \Delta t, t_c]$ is as a result also bounded:

$$\begin{aligned} |\Delta\theta_i| = & \left| \int_{t_c-\Delta t}^{t_c} u_i \, dt \right| && (14.22) \\ \geq & \left| \int_{t_c-\Delta t}^{t_c} \frac{-2\pi c^+}{d_{ij}} \, dt \right| \\ \geq & \left| \int_{t_c-\Delta t}^{t_c} \frac{-2\pi c^+}{2r^c} \, dt \right| \\ = & \left| \frac{-\pi c^+}{r^c} \Delta t \right| = \pi. && (14.23) \end{aligned}$$

As a result aircraft i will do more than half a turn, thus moving away from the collision point. Thus they could only collide if the two aircraft come closer on the vertical axis. But because $\beta_j^a \beta_j^d = 1$, the vertical velocity for aircraft i can only make it move away from the collision point as only the component ∇V_{ij}^a remains in ∇V_{ij}^t. This can be verified by computing the sign of the scalar product between the vertical velocity $v_z \mathbf{z}$ and the vector between the two aircraft $\mathbf{p}_i - \mathbf{p}_j$. Because v_z is monotonic and odd as function of $-\nabla V_i^t \mathbf{z}$, it is possible to ignore the saturation as it does not influence the sign of the scalar product. The following result is obtained:

$$\begin{aligned} & \text{sign}(v_z \mathbf{z} \cdot (\mathbf{p}_i - \mathbf{p}_j)) && (14.24) \\ & = \text{sign}((-K^z \nabla V_i^t \cdot \mathbf{z})\mathbf{z} \cdot (\mathbf{p}_i - \mathbf{p}_j)) \\ & = \text{sign}(-\nabla V_i^t \cdot (\mathbf{p}_i - \mathbf{p}_j)) \\ & = \text{sign}(-\nabla V_{ij}^a \cdot (\mathbf{p}_i - \mathbf{p}_j)) \\ & = \text{sign}\left(\frac{K^a}{(\|\mathbf{p}_i - \mathbf{p}_j\| - 2r)^2} (\mathbf{p}_i - \mathbf{p}_j) \cdot (\mathbf{p}_i - \mathbf{p}_j) \right) > 0. \end{aligned}$$

The vertical speed vector is in the same direction as the vector that goes from aircraft j (encounter) to aircraft i. This shows that the vertical velocity drives aircraft i away from aircraft j.

For the second case where $0 < \beta_j^a \beta_j^d < 1$, the horizontal velocity is already moving the aircraft away from the collision point. As for the vertical velocity v_z, one has to consider an r^c small enough so that:

$$|\beta_j^a \beta_j^d \nabla V_{ij}^a \mathbf{z}| \gg \left| \left(1 - \max_j \left(\beta_j^a \beta_j^d\right)\right) \nabla V_i^n \mathbf{z} \right|. \tag{14.25}$$

The reasoning is then the same as for Equation 14.24. As a result, the vertical speed component v_z will try to drive the aircraft away from the collision point. There exists such small r^c that Equation 14.25 is satisfied, with two corner cases that require further attention. One corner case is if the two aircraft are at the same height, in which case $\nabla V_{ij}^a \cdot \mathbf{z} \approx 0$. Define ϕ the angle between the vector ∇V_{ij}^a and the vector z so that:

$$\begin{aligned}
-\nabla V_{ij}^a \cdot \mathbf{z} = & \quad \| -\nabla V_{ij}^a \| \|\mathbf{z}\| \cos\phi & (14.26)\\
= & \quad \left\| \frac{K^a K^z}{(\|\mathbf{p}_i - \mathbf{p}_j\| - 2r)^2} \frac{\mathbf{p}_i - \mathbf{p}_j}{\|\mathbf{p}_i - \mathbf{p}_j\|} \right\| \cos\phi & (14.27)\\
= & \quad \frac{K^a K^z}{(\|\mathbf{p}_i - \mathbf{p}_j\| - 2r)^2} \left\| \frac{\mathbf{p}_i - \mathbf{p}_j}{\|\mathbf{p}_i - \mathbf{p}_j\|} \right\| \sin\left(\tfrac{\pi}{2} - \phi\right) & (14.28)\\
= & \quad K^a K^z \frac{\sin\left(\frac{\pi}{2} - \phi\right)}{(\|\mathbf{p}_i - \mathbf{p}_j\| - 2r)^2} & (14.29)\\
= & \quad K^a K^z \frac{\sin\left(\frac{\pi}{2} - \phi\right)}{(r^c)^2}. & (14.30)
\end{aligned}$$

By choosing $r^c = \frac{\pi}{2} - \phi$ and taking he limit $\phi \to \pi/2$ we obtain:

$$\lim_{\phi \to \pi/2} \left| K^a K^z \frac{\sin\left(\frac{\pi}{2} - \phi\right)}{\left(\frac{\pi}{2} - \phi\right)^2} \right|, = \infty \tag{14.31}$$

thus the avoidance component ∇V_{ij}^a grows faster than its decrease due to being almost perpendicular with the vertical axis. And because of Equation 14.24 we know that the vertical component will always drive aircraft i away. If $\nabla V_{ij}^a \cdot \mathbf{z} = 0$, the case degenerates to the 2D case treated in [37] and collision avoidance is also guaranteed. Another corner case is if $\beta_j^a = 0$, because $\beta_j^a \beta_j^d \nabla V_{ij}^a = 0$. But this case is treated in Proposition 5 with the margin angle $\rho > 0$.

■

14.2 Results

In this section we will present simulations to show the effectiveness of the presented algorithm in comparison to the 2D algorithm presented in Chapter 13. The scenarios presented in Chapter 10 are not used as they are designed for cases in the plane only. We simulated the encounter of two aircraft in three different scenarios and the obtained trajectories are shown

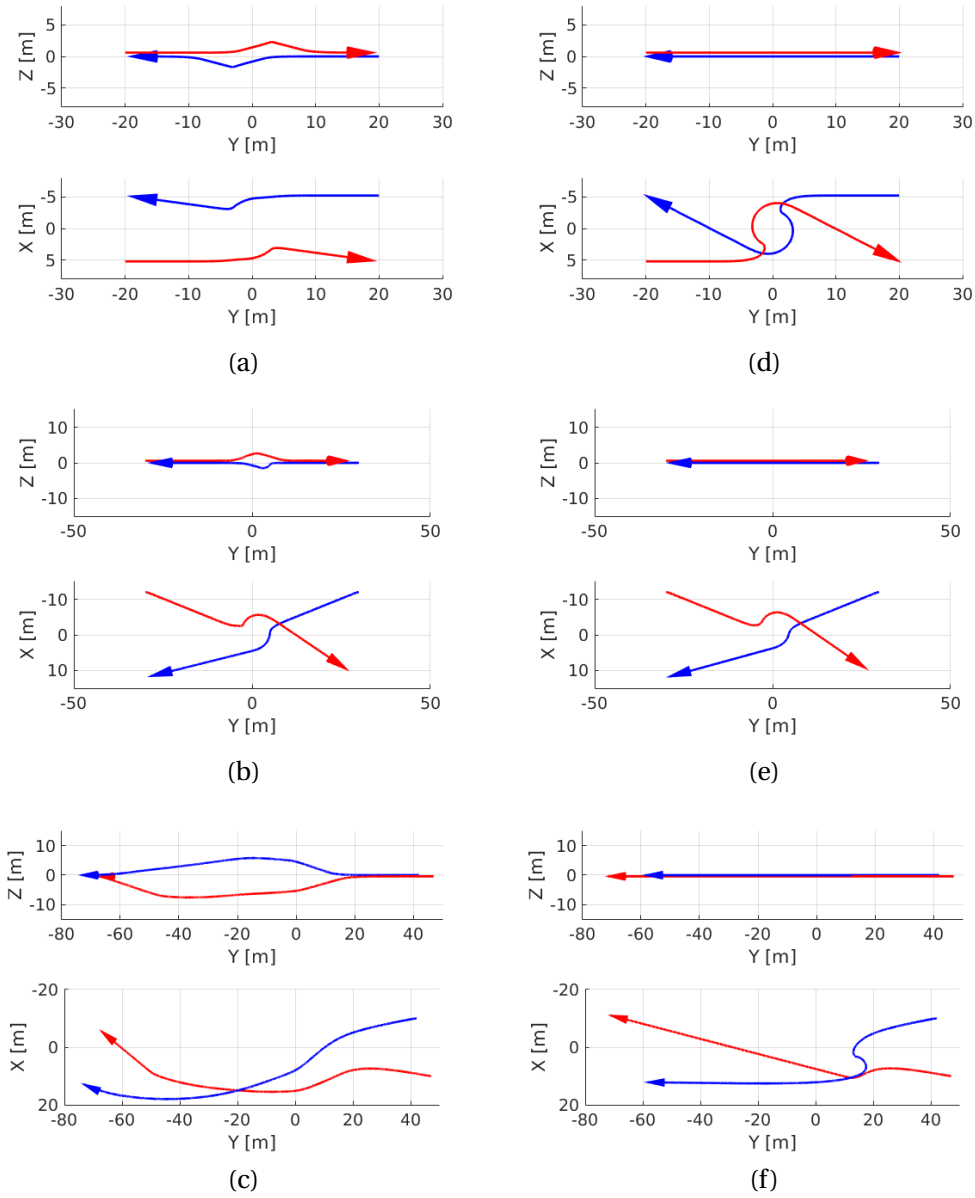


Figure 14.3 – Left column (a,b,c): three sets of trajectories obtained with presented 3D collision avoidance algorithm. Right column (d,e,f): three sets of trajectories obtained with the algorithm from Chapter 13 for the same initial conditions.

in Figure 14.3. The simulations was implemented in Matlab as a microscopic point simulation (no physical simulation) with perfect kinematics and noiseless sensing. The parameters used are: $c^+ = 0.5$, $c^- = 0.4$, $\alpha^s = 110°$, $r = 1$, $r^s = 4$, $r^s = 20$, $K^\omega = 1.0$, $K^z = 1.0$, $K^n = 0.2$, $K^a = 80$, $\rho = 5°$. The simulations were performed with a time step of 0.05 seconds.

Both algorithms show different behaviors: generally speaking, taking advantage of the third

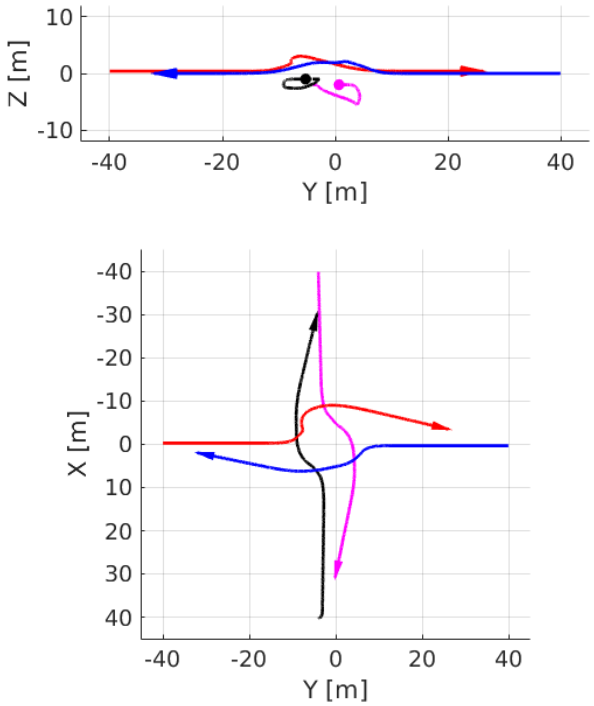


Figure 14.4 – Example of trajectories obtained with 4 aircraft.

dimension enhances the trajectories by making them smoother in most cases (compare Figures 14.3a and 14.3c with Figures 14.3d and 14.3f). Since the PF of the 3D collision avoidance algorithm has been modified to add V_{ij}^{a}, the turning rate command will tend to react strongly around the obstacle. As a result, the 3D collision avoidance algorithm does not always produce the smoothest trajectories, as for example shown in Figure 14.3b. Our algorithm is also capable of avoiding several aircraft. An example is shown in Figure 14.4.

Another scenario was designed to show that the presented algorithm reduces further the average turning rate when compared to the horizontal algorithm from Chapter 13. The turning rate command is studied in depth as it is the only input that is not bounded in both algorithms. Although both algorithms rely on the fact that it can be infinite to prove collision avoidance, it usually remains bounded and depends on the parameters used. Keeping it low is thus of great importance. The scenario is based on three parameters: the distance between tracks e, the angle between tracks ψ (both shown on Figure 14.5) and the height between tracks h. The aircraft aim to stay at their initial height and to follow their track. The parameters are chosen randomly with uniform distribution $\mathscr{U}$ as follows: $e \sim 12(\mathscr{U}-0.5)$, $\phi \sim 1.6\pi(\mathscr{U}-0.5)$ and $h \sim 8r(\mathscr{U}-0.5)$. The simulations were performed 1000 times. For each run, the root mean squared turning rate command over the whole simulation run was computed. The result are presented as boxplots in Figure 14.6. The novel 3D algorithm does perform better than the 2D version from Chapter 13 in terms of the average turning rate with a median value three times

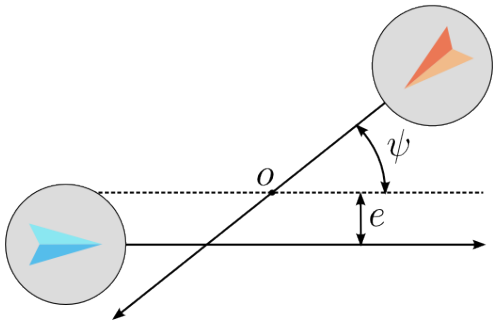


Figure 14.5 – Comparison of the average turning rate for the 2D and 3D implementations for our case scenario.

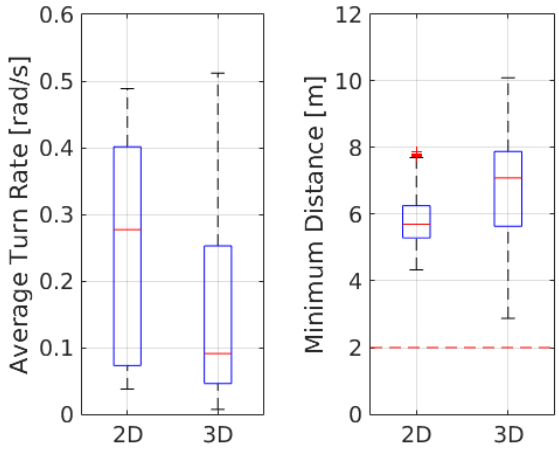


Figure 14.6 – Left boxplot: Comparison of the average turning rate command input for the 2D and 3D implementations. Right boxplot: Minimum distances between two aircraft during the simulations. The red dashed line is the distance for which the two aircraft are considered colliding.

smaller. This shows that the algorithm trades vertical movement for smoother horizontal curves. However, the novel algorithm does sometimes have a larger average turning rate, due to the addition of the avoidance PF resulting in more aggressive maneuvers. The minimum distance between aircraft during the whole simulation was also recorded and for none of the simulations the minimum distance went below the collision distance of $2r$ (see Figure 14.6).

14.3 Discussion

The presented algorithm combines a horizontal and vertical collision maneuvers, improving the performance over a horizontal-only algorithm. But it is not always possible to define a horizontal and a vertical plane, for example in space. In that case, a better approach would be to start with a fully 3 dimensional PF, and adapt it similarly to what is presented in Chapter 10.

The RB and PF algorithms only use the position estimates of other UAVs to navigate and avoid collisions. But by only using the position as input, those algorithms are not able to infer where the other UAVs will be in the future. They maneuver away from a potential collision based the current positions of the UAVs, not based on the positions of the other UAVs when the collision would happen. As a result, our RB and PF algorithms tend to do detours and slow down when near collision because they did not anticipate the motion of the other UAVs. The collision avoidance algorithm in the next chapter uses the velocity estimate provided by the tracking algorithm to add a predictive element and generate more efficient trajectories.

Summary

In this chapter, a three dimensional collision avoidance algorithm is presented. First the definition of FOV is extended to the third dimension, and the dynamical system model is presented. Then the collision avoidance algorithm is presented, followed by the proof of collision avoidance. The chapter concludes with simulations, including a comparison between this algorithm and the one presented in Chapter 13.

15 Velocity-Obstacle-Based Avoidance

ONLY relying on the current configuration of the robots to solve collision avoidance can result in inefficient trajectories. In most cases, knowing the intent of other robots can help solving avoidance conflicts faster and in a safer manner. If the intent cannot be directly known because the robots are not communicating, the intent can nonetheless be inferred from the motion of the other robots as long as perceived. The class of collision avoidance algorithms based on the concept of Velocity Obstacle (VO) use the estimated velocities of other robots to compute the best own velocity to avoid collision. Because the approach is based on both position and velocity, VO-based algorithms are able to anticipate the future position of obstacles (or robots), allowing them to be more efficient (in a sense that will be defined later) than other classes of algorithms (such as PFs). The work presented in this chapter was published in [39].

15.1 Problem Statement

For this algorithm, the robots are assumed to be holonomic and controlled in velocity. The robots also have a heading, which can be controlled independently, and is relevant solely for the orientation of the sensor. The overall kinematic model of the vehicles is described by the following equation:

$$\dot{\mathbf{q}} = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} v_x \\ v_y \\ \omega \end{bmatrix}. \tag{15.1}$$

with $\mathbf{p} = [x, y]^T$ and $\mathbf{v} = [v_x, v_y]^T$ the position and the velocity of the robot, respectively. Furthermore, the robots have the following constraints: first, they have a maximum velocity $c^{max} \geq ||\mathbf{v}||$ and a maximum velocity change Δc^{max} that the robot can perform during a time step Δt. The algorithm also requires that the robot can come to a full stop (i.e., no airplane motion model can be considered). The set of allowed velocities can be summarized by the

following actuation set in the velocity space:

$$\mathbf{U} = \mathbf{D}(\mathbf{v}, \Delta c^{max}) \cap \mathbf{D}(0, c^{max}), \tag{15.2}$$

with $\mathbf{D}(\mathbf{c}, r)$ the closed disk of radius r centered at $\mathbf{c}$. Finally, the robot also has a maximum turning rate $|\omega| \leq \omega^{max}$.

15.2 Collision Avoidance Algorithm

In this section, the Sensor-Aware Velocity Obstacle (SAVO) algorithm is presented. The approach for this algorithm starts from the classical definition of VO; a VO of robot j from the point of view of robot i is defined as:

$$\mathbf{VO}^{\tau}_{i|j} = \left\{\mathbf{v} \,|\, \exists t \in [0, \tau] :: t\mathbf{v} \in \mathbf{D}(\mathbf{p}_j - \mathbf{p}_i, r_j + r_i + \sigma_r) \oplus \Sigma_p\right\}, \tag{15.3}$$

with Σ_p a set of possible position error and σ_r the error on the estimated radius. Care is taken to choose a value of τ large enough for the robot i to go from $||\mathbf{v}|| = c_i^{max}$ to $||\mathbf{v}|| = 0$ in less than time τ. In a similar fashion, the maximum velocity should remain low enough in order for the robot to come at a full stop in less than half its sensor range. Those conditions can be summarized as follows:

$$\begin{cases} \tau > \frac{c_i^{max}}{\Delta c^{max}} \Delta t \\ c_i^{max} < \sqrt{\frac{\Delta c^{max} r_i^s}{\Delta t}}, \end{cases} \tag{15.4}$$

with Δt the time step between two computations.

As it is not always possible to know the behavior of other robots due to the limited FOV, we take the conservative approach to build the set of velocities that will lead to a collision with robot j, named $\mathbf{C}^{\tau}_{i|j}$, by taking into account all the possible future velocities of robot j. To take into account a possible uncertainty on the estimated velocity of the other robot j, the actuation set $\mathbf{U}_j$ is augmented by performing the Minkowski sum with the set of velocity uncertainty Σ_v. Furthermore, to guarantee that all robots have as solution the possibility to stop, the zero velocity point is added to the actuation set of robot the other robots. To keep a convex set, a convex hull operation is performed after the addition of the zero velocity, resulting in the augmented actuation set:

$$\mathbf{U}^0_j = \mathrm{Conv}\Big(\left\{\mathbf{U}_j \oplus \Sigma_v, (0,0)\right\}\Big). \tag{15.5}$$

From the sets $\mathbf{VO}^{\tau}_{i|j}$ and $\mathbf{U}^0_j$ it is now possible to build $\mathbf{C}^{\tau}_{i|j}$, the set containing all velocities of i that will lead to a collision with j in the next time τ, as follows:

$$\mathbf{C}^{\tau}_{i|j} = \mathbf{VO}^{\tau}_{i|j} \oplus \mathbf{U}^0_j. \tag{15.6}$$

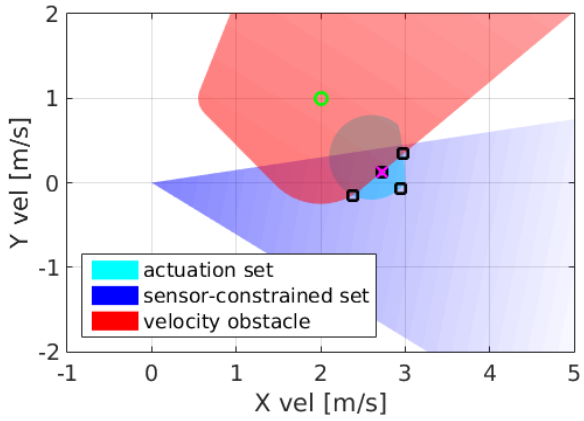


Figure 15.1 – Example of the velocity sets used in this algorithm. The black squares are possible solutions. The magenta cross inside a box shows the optimal solution for the desired velocity illustrated by a green circle.

The set of velocities that avoids collision with all detected robots and respects the constraints imposed by both the sensing and the actuation of robot i is given by:

$$\mathbf{CA}_i^\tau = \left\{ \mathbf{v} \,|\, \mathbf{v} \notin \bigcup_{j \neq i} \mathbf{C}_{i|j}^\tau \wedge \mathbf{v} \in \mathbf{S}_i \cap \mathbf{U}_i \right\}. \tag{15.7}$$

This algorithm is named SAVO in Chapters 16 and 17. An example of the sets involved the $\mathbf{CA}_i^\tau$ are illustrated in Figure 15.1. The new velocity $\mathbf{v}_i^{new} \in \mathbf{CA}_i^\tau$ to apply is then chosen to be the closest to some desired velocity $\mathbf{v}_i^{pref}$. If we choose a quadratic function as optimization function, the search for a solution is limited to the border of the collision avoidance set $\partial\mathbf{CA}_i^\tau$. Note that due to the conservative approach, it is possible that $\mathbf{CA}_i^\tau = \emptyset$. If it is the case, the robot chooses $\mathbf{v}_i^{new} = [0,0]^T$ as a last resort avoidance maneuver:

$$\mathbf{v}_i^{new} = \begin{cases} \underset{\partial\mathbf{CA}_i^\tau}{\arg\min} ||\mathbf{v} - \mathbf{v}_i^{pref}||, & \text{if } \mathbf{CA}_i^\tau \neq \emptyset \\ [0,0]^T, & \text{otherwise.} \end{cases} \tag{15.8}$$

Proposition 7. *N robots with their velocities defined with Equation 15.8 will not collide.*

Proof. If Equation 15.8 has a solution, then by the construction of the velocity obstacle and because of Proposition 7, each robot will choose a velocity that will not collide with any other robot.

If no solution can be found (i.e. $\mathbf{CA}_i^\tau = \emptyset$), a robot i will try to stop as quickly as possible (i.e. as fast as $\mathbf{U}_i$ allows for). The other robots fall into two categories. Either a robot j does not

detect robot i, in which case it is of no threat to robot i because of Proposition 1. Or robot j does detect robot i, in which case, if robot j has a solution, it will avoid robot i with the consideration that robot j will leave the possibility to robot i to stop.

Finally, if several robots are not able to find a solution, consider the moment where they first are not able to find a solution and start to brake. This can only happen if the sensors of the robots detect of robots that were not in the FOV before. A robot j can enter the FOV of another robot in two ways. First, the robot j either comes from the rear, and was not seen by some robot i because $|\angle(\mathbf{d}_i, \mathbf{p}_j - \mathbf{p}_i)| > \alpha_i^s$. In this case, robot j is moving towards robot i (otherwise there would be no collision). As a consequence, robot j was able to detect robot i at previous time step and to find a solution to avoid i (as this is the first time step that robot j does not find a solution). If robot j could find a solution previously, the solution includes the possibility that robot i would suddenly brake and stop, even if robot j has to perform an emergency brake too because of the condition described by Equation 15.4. The second possibility is that robot i could not sense other robots previously because they were out of range. In which case all robots have, according to the conditions of Equation 15.4, enough time to come to a full stop before colliding with the other robots. ■

In fact, it is possible to derive Equation 10.2 directly from the VO framework. The argument goes as follows: consider two robots i and j that do not detect each other and are about to collide. As robot j is not responsible for the collision, the worst case scenario is that robot j stays stopped. Because the two robots are about to collide, their distance is exactly $r_i + r_j$. Robot j can be anywhere around robot i as long as $\angle(\mathbf{d}_i, \mathbf{p}_j - \mathbf{p}_i) > \alpha_i^s$. As a result, the VO of robot j that robot i has to avoid is a half plane at the collision point between robots i and j and tangent to robot j at that point. It follows that the set of velocities that can potentially lead robot i to collide with robot j because they cannot detect each other is:

$$\mathbf{I} = \left\{ \mathbf{v} \,|\, \left(\angle(\mathbf{v}_i, \mathbf{d}_i) > \alpha_i^s - \frac{\pi}{2} \right) \vee \left(\angle(\mathbf{v}_i, \mathbf{d}_i) < -\alpha_i^s + \frac{\pi}{2} \right) \right\}. \tag{15.9}$$

To guarantee no collision, robot i has to pick a velocity in the set $\bar{\mathbf{I}} = \mathbf{S}_i$.

15.3 Heading Control

The control of the robot's heading, and thus the sensor's FOV, is essentially what allows SAVO to reach any point in the plane (as long as it is reachable) and not just the points in $\mathbf{S}_i$. We propose to steer the sensor heading based on the robot's velocity outputted by the collision avoidance algorithm, as prescribed by the third guideline of Chapter10. To avoid a deadlock due to no robot finding a solution, we also propose that when a robot uses its last resort avoidance maneuver, it spins in a defined direction at maximum turn rate:

$$u = \begin{cases} -K^a \angle(\mathbf{d}_i, \mathbf{v}_i), & \text{if } \mathbf{CA}_i^\tau \neq \emptyset \\ \dot{\theta}_i^{max}, & \text{otherwise.} \end{cases} \tag{15.10}$$

To have a smooth transition between deadlock and normal maneuvering on one of the sides, and to use the full range of turning rate, K^a should be set as follows:

$$K^a = \frac{\dot{\theta}_i^{max}}{\alpha_i^s - \frac{\pi}{2}}. \tag{15.11}$$

If the gain K^a is found to be too aggressive, it is possible to use a smaller value for $\dot{\theta}_i^{max}$.

15.4 Reciprocity

Due to the limited FOV, it is not always possible for a robot to know if it is seen by an other robot, especially when the sensors are noisy. As a consequence, a robot is not able to rely on the fact another robot will cooperate, which is an important assumption for algorithms such as ORCA [90] or HRVO [89]. We propose to mitigate the oscillations common to non-reciprocal VO as follows. The robot checks if only giving half of the change of velocity is also collision free:

$$\mathbf{v}_i^{half} = \frac{\mathbf{v}_i^{new} + \mathbf{v}_i^{prev}}{2} \notin \bigcup_{j \neq i} C_{i|j}^{\tau}, \tag{15.12}$$

with $\mathbf{v}_i^{prev}$ the velocity computed at previous time step. The velocity $\mathbf{v}_i^{prev}$ is preferred over the actual velocity $\mathbf{v}_i$ because experiments with our quadrotors showed that the use of $\mathbf{v}_i$ amplifies the noise on the actuation. If $\mathbf{v}_i^{half}$ is collision free, it is used instead of $\mathbf{v}_i^{new}$, resulting in a maneuver aggressiveness similar to what is obtained with reciprocal algorithms. When robots meet, the second collision avoidance computation is usually enough to perform the avoidance with $\mathbf{v}_i^{half}$ instead of $\mathbf{v}_i^{new}$. Note that $\mathbf{v}_i^{half} \in \mathbf{S}_i \cap \mathbf{U}_i$ because both sets are convex and both $\mathbf{v}_i^{new}$ and $\mathbf{v}_i^{prev}$ are in those two sets.

15.5 Implementation

Due to the complex shapes that this method can create, it is not possible to apply the commonly used "triangle" approach for the computation on the velocity sets (e.g., [89]). It is not possible to simplify the problem into half-planes as in [90] due to the possible lack of reciprocity. To stay as close as possible to the mathematical formulation, the sets are implemented as convex polytopes. This has the drawback to be more computationally heavy and in principle scale with $\mathcal{O}(n^2)$, n being the number of robots observed. However, the computational cost is physically limited by the sensor's range. This method is thus well suited for robots with a small sensor range or that navigate in sparsely populated environments.

For both simulations and real experiments, the robots had a goal position $\mathbf{p}^d$ from which the

preferred velocity was derived:

$$\mathbf{v}_i^{pref} = c_i^{pref} \frac{\mathbf{p}_i^d - \mathbf{p}_i}{||\mathbf{p}_i^d - \mathbf{p}_i|| + r^d}, \tag{15.13}$$

with c_i^{pref} the preferred speed of robot i and r^d a term that will slow down the robot as it reaches its goal position.

15.6 Results

15.6.1 Simulations

To illustrate the importance of FOV constraints in collision avoidance, we leveraged the two scenarios described in Chapter 10 and compared the performances of both the canonical non FOV-aware VO algorithm with those of the SAVO algorithm. The simulations was implemented in Matlab as a microscopic point simulation (no physical simulation) with perfect kinematics and noiseless sensing. The canonical VO implementation differs from the SAVO algorithm in only two points. First, it does not constrain the velocity of a robot i to stay in $\mathbf{S}_i$. Second, it does not include the zero velocity in what would be $\mathbf{U}_j^0$ for the SAVO algorithm. It uses, instead of $\mathbf{U}_j^0$, the set $\mathbf{U}_j^\Sigma = \mathbf{U}_j \oplus \Sigma_v$. This canonical version of the VO algorithm is named VOOS in Chapter 16. This conservative implementation of VO allows us to demonstrate that the problem is indeed due to violating the constraints imposed by the limited FOV of the sensor. The parameters of our simulations are $r = 0.4$ *m*, $r^s = 2.5$ *m*, $\alpha^s = 110°$, $\Delta c^{max} = 0.12$ *m/s*, $c^{max} = 4$ *m/s*, $\dot{\theta}^{max} = 0.5$ *rad/s* and $\tau = 1$ *s*. The simulations are performed with a time step $\Delta t = 0.05$ *s*. The same parameters are used for both SAVO and the canonical VO implementations. If one algorithm gives a command that the robot is not able to follow, the actual actuation saturate.

As in Chapter 12, the first scenario involves two robots that start back-to-back. Both robots have their destination on the left side of the horizontal axis. One slower robot has the right orientation and simply has to go forward to reach its destination. The other, faster robot faces away from its destination. The fastest trajectory is for the faster robot to go in a straight line, which leads to a collision in the case the algorithm is not aware of the each other constraints, as shown in the left plot of Figure 15.2. When the SAVO algorithm is used, the faster robot stays at the same position until it has turned enough for the sensor to see a clear path towards the destination, resulting in the trajectories shown in the right plot of Figure 15.2.

The second scenario involves four robots. Two slow robots avoid each other and thus get close. Just when the two slow robots successfully avoid the collision and go out of each other's detection range, each one sees a fast robot approaching from the other side they performed the avoidance. In case where the algorithm is not respecting the constraints imposed by the sensor, the slow robots will perform an aggressive maneuver towards each other to avoid the faster robots. That maneuver will lead them to collide (left plot of Figure 15.3). With the SAVO

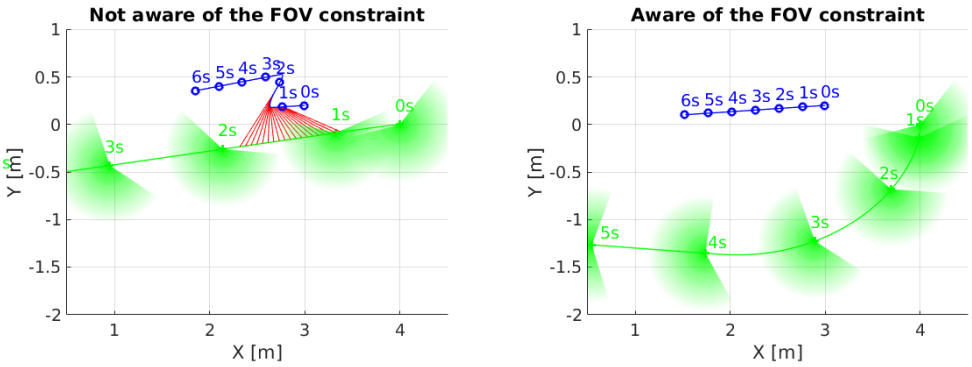


Figure 15.2 – Trajectories obtained with the first scenario. Left: the algorithm is not aware of the FOV. Right: The SAVO algorithm. The solid blue and green lines are robots' trajectories. The markers and numbers give an indication of the timing of the trajectories. The FOV of some robots at the timings indicated by numbers are represented by circular sectors. The color of the circular sectors match the color of the trajectories. The FOV range is not to scale. The red solid lines are distances between two robots that are less than the sum of their radii.

algorithm, the two slow robots will stop instead and remain safe (right plot of Figure 15.3). Notice how SAVO, although more constrained, generates smoother and faster trajectories compared than the normal VO implementation in both scenarios.

15.6.2 Real-World Experiments

To validate our approach, we performed experiments with our real quadrotors presented in Chapter 2. Although they do move in a three dimensional space, avoidance with quadrotors should be performed horizontally as it is hazardous to maneuver in the down-wash of another quadrotor. Furthermore, quadrotors typically are more agile at translational compared to yaw rotation, making it an ideal case study for the SAVO algorithm. The sensor is emulated using the MCS ground-truth data, providing position and velocity data with little noise. Limited range and FOV are also emulated.

Because the motion of a quadrotor is disturbed by non-deterministic air flows, small accelerations performed by the quadrotor have little effect on its motion. As a result the quadrotor is not able to properly accelerate and maneuver if Δc^{max} is too small. As a result, we chose $\Delta c^{max} = 0.3$ *m/s*. The desired speed was set to $c^{pref} = 0.6$ *m/s*. The other parameters have the same values as those used in simulation.

To test the SAVO algorithm, two scenarios were designed. Those scenarios are to demonstrate that the algorithm is able to run on a real platform rather than to show the consequences of a limited FOV. The first scenario involves two quadrotors in a head-on configuration. The first quadrotor starts at the position $[3.5, 0]$ *m* and has its goal position at $[-3.5, 0]$ *m*. The other robot does the opposite: it goes from the position $[-3.5, 0]$ *m* and tries to reach position $[3.5, 0]$ *m*. The second scenario illustrates a crossing scenario. The first quadrotor starts at

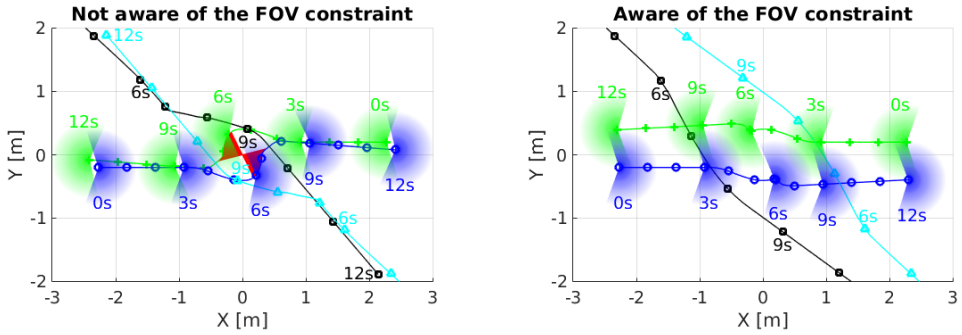


Figure 15.3 – Trajectories obtained with the second scenario. Left: the algorithm is not aware of the FOV. Right: The SAVO algorithm. The solid blue, black, green and cyan lines are robots' trajectories. The markers and numbers give an indication of the timing of the trajectories. The FOV of some robots at the timings indicated by numbers are represented by circular sectors. The color of the circular sectors match the color of the trajectories. The FOV range is not to scale. The red solid lines are distances between two robots that are less than the sum of their radii.

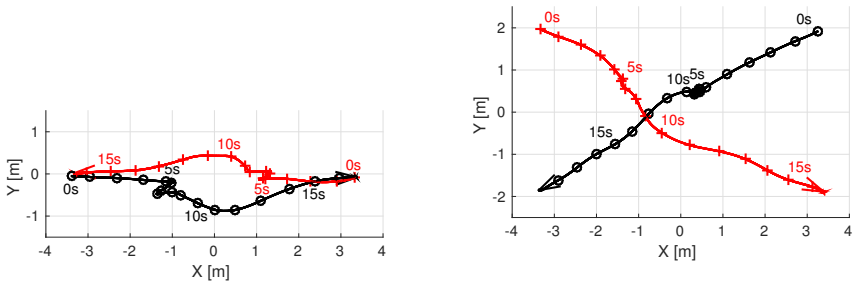


Figure 15.4 – Trajectories obtained for the experiments with quadrotors. Left: Head on collision. Right: Cross intersection. The black and red solid lines represent the trajectory of each quadrotor. The arrow ends show the direction of motion. The markers and numbers give an indication of the timing of the trajectory.

$[-3.5, 2]$ m and has its goal at $[3.5, -2]$ m. The second robot starts at $[3.5, 2]$ m and has its goal at $[-3.5, -2]$ m. In both scenarios, the robots start oriented facing the goal. A sample of the resulting trajectories are shown in Figure 15.4. Notice that the robot move back at some points of the trajectory. This is an overshoot of the braking maneuver and is inherent to the propulsion method of the quadrotor. It can be accounted for by increasing the radius of the quadrotors in the SAVO algorithm. This issue should not be present for ground robots because when they brake, their velocity goes to zero with respect to the ground. For each scenario, the minimum distance between the robots during the avoidance maneuver was recorded and is shown in Figure 15.5. Each scenario was run at least 19 times and at no moment did the distance between the robots drop below 0.8 m, the distance at which they would collide. In fact, the robots stayed at a safe distance. This is a consequence of the conservative approach required by the limited FOV of the robots.

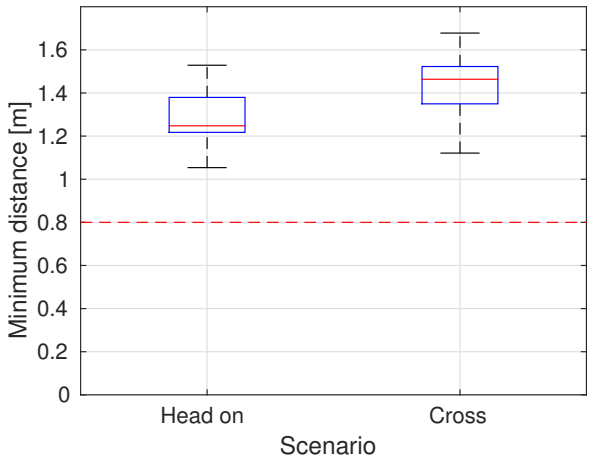


Figure 15.5 – Boxplot of the minimum distance between the quadrotors during the avoidance maneuvers. The red dashed line represents the distance at which the quadrotors would collide.

15.7 Discussion

In this algorithm, the collision avoidance is mainly performed by a lateral motion of the robot (i.e., a change in direction in the velocity). The change in heading is only a consequence of the change in velocity. As a consequence, the heading of the sensor will change slowly, preventing the robot to do significant changes in direction. It will instead slow down to avoid the collision, and as a result the collision avoidance with a FOV close to 180° will significantly slow down the robot. A look ahead strategy for the sensor heading could be investigated as a possible solution.

This algorithm can be computationally expensive for a large number of obstacles, as the problem complexity grows quadratically with the number of obstacles. Furthermore, the search space is not convex, which makes it more complex the search for a solution. A possible solution could be inspired from [90] where the VO are transformed into half planes, greatly simplifying the problem. But the work in [90] assumes reciprocity during the collision avoidance, and it is not certain that the algorithm can function without.

As mentioned in Section 15.4, a limited FOV makes it uncertain to know if two robots can mutually see each other, and relying on reciprocity can be hazardous. But there are situations where two robots are confident enough that both see the other one (e.g., head on collision), and take advantage of that knowledge to perform less aggressive avoidance maneuvers.

Another possible improvement would be to extend the algorithm to the third dimension. Although the theory will not change significantly, the implementation will differ significantly from the two-dimensional version. In particular, because the algorithm is implemented using polytopes, the current implementation might be too expensive for the current on-board computational resources.

Summary

In this chapter, a VO-based collision avoidance algorithm that guarantees collision-free navigation with limited FOV sensing is presented. First, the robot kinematic model is presented and the notion of VO is introduced. Then the necessary changes to the standard velocity obstacle algorithm are presented, as well as a proof of collision-free navigation. Finally, mass-point microscopic simulations and real-world experiments validate the approach.

16 The Effect of Sensor Limitations on Avoidance

SEVERAL collision avoidance paradigms have been presented in the scientific literature, and some were used in this work. Each collision avoidance algorithm has its unique properties, and as a result a given algorithm might be better suited to a certain application than another. But with the large selection of possible algorithms to choose from, it might be hard to know which collision avoidance algorithm to choose for a specific application, especially in presence of realistic sensory limitations. In this chapter, an attempt to address this problem is presented by studying the relation between sensory limitations and collision avoidance performance.

16.1 Sensory Limitations and How to Mitigate Them

In this section, we present the limitations usually present in vision, namely noise (i.e. stochastic error), lag, limited range and limited FOV. Our sensor model is illustrated in Figure 16.1. This generic sensor model was chosen as it allows to independently change the parameters of the filter in order to provide a clear link between a specific sensor limitation and the performance of an algorithm. Furthermore, it only requires a ground-truth position data to emulate the sensor, which enable us to use this model not only in a high-fidelity simulation, but also for real experiments (with the ground-truth given by a MCS).

For fairness, and when possible, the collision avoidance algorithms are made robust to the sensor inaccuracies. Not implementing the means to handle sensor's inaccuracies would favor algorithms that by design avoid with additional safety margin (i.e. distance) than necessary. Care is taken to ensure that the implementation of the mitigation systems are as similar as possible across the algorithms.

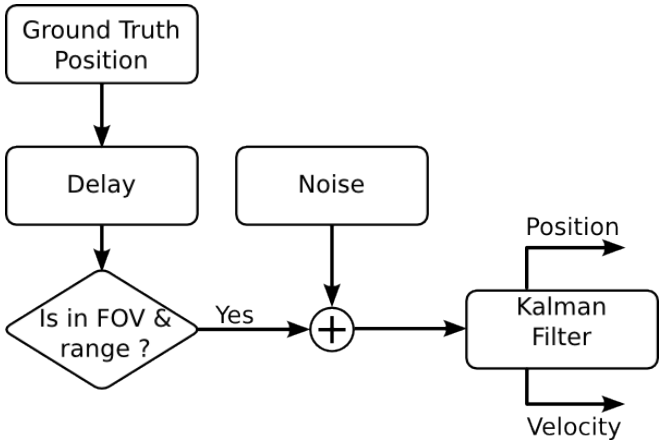


Figure 16.1 – Overview of the sensor's model.

16.1.1 Noise

To allow for fair comparison between the collision avoidance algorithms, the noise level needs to be equivalent, despite the fact that the algorithms use different inputs. All algorithms use position, but only VO-based algorithms use velocity. To remain fair, the errors in position and velocity need to be linked to a single parameter. This is done by considering a sensor that only returns a position measurement that is affected by some Gaussian noise. The velocity is derived from the position using a Kalman filter. This way, all algorithms have access to the same information (filtered position information) but may not use all of the information available (PF-based algorithms do not use the velocity information that is contained in the position measurements).

All algorithms presented in Section 16.2 are made noise-resistant using the same methodology, which was presented in [40]. For all algorithms, the shape of encountered robots O have been increased by the set of position error Σ_p (or an approximation of it). This can be performed by using the Minkowski addition: $O \oplus \Sigma_p$. The uncertainty on the shape Σ_s (e.g., size) can be considered as well: $O \oplus \Sigma_p \oplus \Sigma_s$. If both the shape of the robots and the shape of the noise are circles, both radii (robot and noise) can be added. In this work, the noise originates from an isotropic Gaussian, so the radius of the other robots is simply augmented by the one standard deviation of the noise of the position. One standard deviation was chosen based on existing literature [40][89] but a higher safety margin can be used if the application requires it. Regarding noisy velocity estimates, only the VO-based algorithms have to be made resistant, as the PF-based do not use the velocity estimate. In this case, the VO is dilated in a similar way as done for the position: $VO \oplus \Sigma_v$ with Σ_v the shape of the noise on the velocity. Contrary to [40] where only half of the standard deviation of the velocity error was used, in this work the velocity margin added to the velocity obstacle is equal to one standard deviation of the velocity error. This change is made for consistency, not only between the different VO algorithms, but also between velocity and position estimates.

16.1.2 Lag

Time delay, or lag, is a well studied problem with sensors. However, due to the nature of the problem there is no real mitigation technique when it comes to collision avoidance, besides adding extra safety margin and be careful that the algorithms do not maneuver to aggressively in order to avoid oscillations. In this work, no sensor lag mitigation technique will be proposed and will be bounded to the study of the effect of lag on collision avoidance performance for different algorithms.

16.1.3 Range

Sensors have an effective range under which they can reliably detect objects and other robots. The sensory range can be limited by many factors, such as lack of resolution (in case of vision for example), signal strength (for lasers), etc. Some collision avoidance algorithms have been designed for sensors with a limited range [96][95][37]. A limited range is usually not a problem, provided that it is sufficient to give the robot enough time to decide and perform the collision avoidance maneuver. In this work, no sensor range mitigation technique will be proposed and will be bounded to the study of the effect of lag on collision avoidance performance for different algorithms. A robot i can detect an other robot j as long as the position of the latter is in range $p_j \in \mathbf{R}_i$, with:

$$\mathbf{R}_i = \left\{ \mathbf{p} \;\middle|\; ||\mathbf{p} - \mathbf{p}_i|| < r^s \right\}, \tag{16.1}$$

with with $\mathbf{p}_i$ the position of the robot i and r^s the sensor's range. An illustration of the parameter r^s is given in Figure 16.2.

16.1.4 Field of View

The consequences of a limited FOV have been already detailed in Chapter 10. In this chapter, the effect of the amount of FOV on the collision avoidance algorithms is studied.

16.2 Considered Collision Avoidance Algorithms

Six collision avoidance algorithms are evaluated. Their characteristics are compared in Tables 16.1 and 16.2. Two algorithms (SAVO and SAPFHV) have non-FOV aware variants (VOOS and PFOS, respectively). The non FOV-aware variants are described in the same chapters than their FOV-aware counterparts. The ORCA algorithm, which was not described yet, is presented in this section.

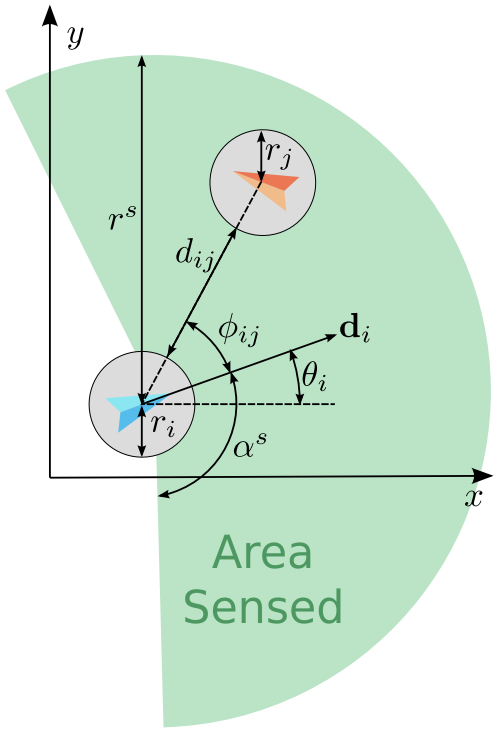


Figure 16.2 – Illustration of a robot i facing direction $\mathbf{d}_i$ with a sensor that has a limited range r^s and FOV defined by α^s. Robot i detects robot j with a distance d_{ij} and with a bearing angle ϕ_{ij}

.

16.2.1 Optimal Reciprocal Collision Avoidance

The Optimal Reciprocal Collision Avoidance (ORCA) [90] is an improvement on VO. It is also based on VO, but the procedure to find a solution is different. Instead of searching a suitable velocity on the edge of all VOs, ORCA first computes for each other robot j the minimum amount of change in velocity $\mathbf{u}_a$ to avoid that robot. It then defines the $ORCA^{\tau}_{i,j}$ half space in which the robot i can freely choose its velocity as:

$$ORCA^{\tau}_{i,j} = \left\{ \mathbf{v} | (\mathbf{v} - (\mathbf{v}_i + \frac{\mathbf{u}_a}{2})) \cdot \frac{\mathbf{u}_a}{\|\mathbf{u}_a\|} \geq 0 \right\}. \tag{16.2}$$

Because the problem is described by half-planes, the problem is now convex, which is computationally more efficient compared to the standard approach. Furthermore, this approach implements the concept of reciprocity, where both robots only do half of the required maneuver to avoid the collision because they assume that the other robot will perform a collision avoidance as well in the same way. This results in more efficient collision avoidance trajectories. In this work, ORCA is implemented using the library available at [108].

Table 16.1 – A summary of the different algorithms used in this work, where they are described and what their acronym is.

Name	Chapter	Acronym
VO for Omnidirectional Sensor	15	VOOS
Sensor-Aware VO	15	SAVO
Optimal Reciprocal Collision Avoidance	16	ORCA
PF for Omnidirectional Sensor	12	PFOS
Sensor-Aware PF for Holonomic vehicles	12	SAPFHV
Sensor-Aware PF for Unicycle vehicles	13	SAPFUV

Table 16.2 – A summary of the different algorithms used in this work, what type they are, if they guarantee collision avoidance with limited FOV sensing and for which experiment they are used.

Acronym	Type	Motion Model	FOV aware	Limitation studied
VOOS	VO	Holonomic	No	Noise, lag and range
SAVO	VO	Holonomic	Yes	FOV
ORCA	VO	Holonomic	No	Noise, lag and range
PFOS	PF	Holonomic	No	Noise, lag and range
SAPFHV	PF	Holonomic	Yes	FOV
SAPFUV	PF	Unicycle	Yes	FOV

16.3 Scenario

Our scenario involves two quadrotors that start at a predetermined initial position and have to reach a predefined destination. The systematic experiments are carried out in the calibrated, high-fidelity simulator Webots (see Section 2.2.4 for additional details). The scenario is built as follows: first, define the center of the scenario as $[0,0]^T$ and put a first quadrotor at position $[-4,e]^T$ with e an offset; second, place the second quadrotor at the position $[4\cos(\psi),4\sin(\psi)]^T$ with ψ some angle; third, set the destinations of the quadrotors as $[4,e]^T$ for the first quadrotor and $[-4\cos(\psi),-4\sin(\psi)]^T$ for the second one; finally we run the simulation until both quadrotors arrive at 0.5 or less m from their respective destination. An illustration of the scenario is shown in Figure 16.3. We perform 20 collision avoidance simulations for each combination of offset $e \in \{-0.9,-0.6,-0.3,0,0.3,0.6,0.9\}$ and angle $\psi \in \{180°,150°,120°,90°,60°\}$, which results in 700 collision avoidance attempts for each data-point for Figures 16.6 to 16.9. If some parameter is not studied in particular (i.e. not changed across simulations), it is set to the following default value: 0.1 m for noise, 0 s for lag, 4 m for sensing range and a 360° FOV (omnidirectional) for the non FOV-aware algorithms (i.e., PFOS, VOOS, ORCA). The SOVA, SAPFHV and SAPFUV algorithms are only used in the scenario where the effect of the FOV is studied. The SOVA and SAPFHV are essential limited-FOV resistant variants of the VOOS and PFOS algorithms, respectively. For all simulations, the target speed is 0.3 m/s.

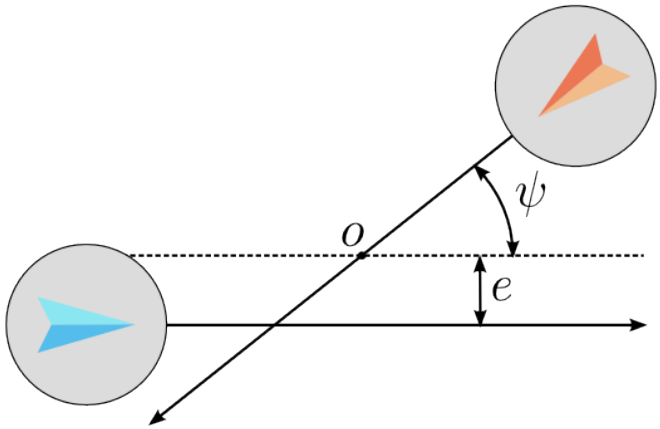


Figure 16.3 – Illustration of the scenario.

16.4 Metrics

To analyze the collision avoidance trajectories obtained from the simulations, four metrics are used:

- **Distance traveled:** The distance traveled by a quadrotor for the collision avoidance maneuver k is defined as:

$$D_k = \sum_{n_k^i < n < n_k^f} \|\mathbf{p}[n] - \mathbf{p}[n-1]\|, \tag{16.3}$$

with n_k^i being the first simulation step n during the collision maneuver k at which both quadrotors are at a distance equal or superior to 0.5 m from their initial positions, and n_k^f being the first simulation step n during the collision maneuver k at which both quadrotors are at a distance equal or less to 0.5 m from their destinations. The average distance traveled during the simulation is then

$$D = \frac{\sum_{k=0}^{700} D_k}{700}. \tag{16.4}$$

In the Figures 16.6 to 16.9, the thick solid line in plots (a) represents the median traveled distance over 700 runs, and the colored patches show the upper and lower quartiles of the same data.

- **Time to completion:** The time it took both quadrotors to complete their collision avoidance maneuver, averaged over all collision avoidance maneuvers, is defined as:

$$T = \frac{\sum_{k=0}^{700} (t_k^f - t_k^i)}{700}, \tag{16.5}$$

with t_k^i the time at step n_k^i and t_k^f the time at step n_k^f. In the Figures 16.6 to 16.9, the thick solid line in plots (b) represents the median time to completion over 700 runs, and the colored patches show the upper and lower quartiles of the same data.

- **Average acceleration:** To obtain the average acceleration, the average velocity has to be computed first:

$$\mathbf{v}[n] = \frac{\mathbf{p}[n] - \mathbf{p}[n-1]}{t_n - t_{n-1}}, \tag{16.6}$$

with t_n the time at step n. Because the velocity had significant noise (probably due to timing issues), the velocity was first filtered using a moving average with a window of 5, followed by a median filter with a window of 7. The acceleration at time n is then computed as:

$$\mathbf{a}[n] = \frac{\mathbf{v}[n] - \mathbf{v}[n-1]}{t_n - t_{n-1}}. \tag{16.7}$$

Because the acceleration is noisy, it is filtered using a median filter with a window of 5. The overall average acceleration over the simulation is defined as:

$$A = \frac{\sum_{k=0}^{700} \sum_{n_k^i < n < n_k^f} \|\mathbf{a}[n]\|}{700}, \tag{16.8}$$

In the Figures 16.6 to 16.9, the thick solid line in plots (c) represents the median average acceleration over 700 runs, and the colored patches show the upper and lower quartiles of the same data.

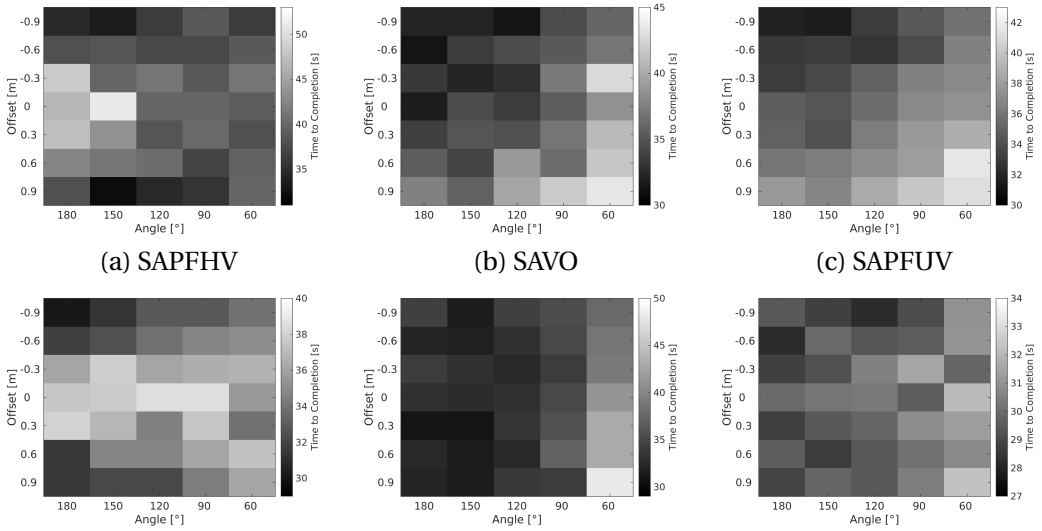
- **Proportion of collisions:** The proportion of collisions over the whole simulation is defined as:

$$C = \frac{\sum_{k=0}^{700} \bigvee_{n_k^i < n_k < n_k^f} 1_c(n)}{700}, \tag{16.9}$$

with

$$1_c(n) = \begin{cases} 1, & \text{if } \|\mathbf{p}_i[n] - \mathbf{p}_j[n]\| < 0.7 \\ 0, & \text{otherwise.} \end{cases} \tag{16.10}$$

In the Figures 16.6 to 16.9, the thick solid line in plots (d) represents the proportion of collision over 700 runs, and the colored patches show the 95% interval of confidence, computed using the Clopper-Pearson method.



(a) SAPFHV (b) SAVO (c) SAPFUV

(d) PFOS (e) VOOS (f) ORCA

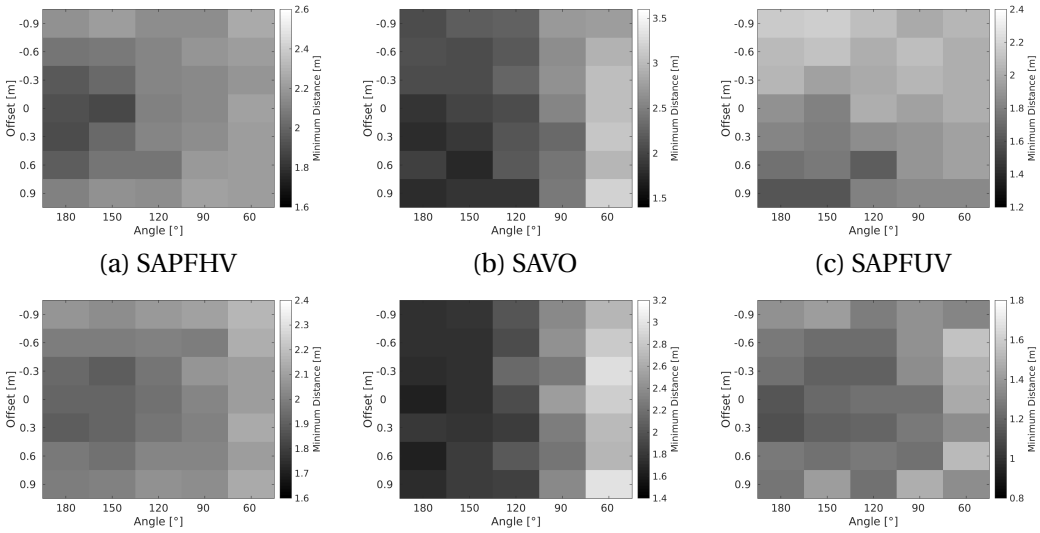
Figure 16.4 – Maps of the minimum distance achieved during the avoidance maneuver for the robots as function of the scenario's offset and angle.

16.5 Results

To understand which scenario configuration is the most challenging for a certain collision avoidance algorithm, the time of completion and the minimum distance averaged across the runs have been plotted as function of the offset and the angle (see Figures 16.4 and 16.5). Some algorithms, such as the PFOS and the SAPFHV algorithms, have more difficulties when the scenario configuration is head-on with no offset, as it is where it takes them most time to avoid and with the smallest distance margin. VO-based algorithms seem to behave similarly, where they are the most time efficient in head on collision, but with the least distance margin. On the contrary, in side-by-side configurations (with an angle of 60°) the VO-based algorithms take more time to avoid and do so with more margin. Noticeably, the ORCA performs better than the SAVO and VOOS algorithm. Finally, the features of the SAPFUV algorithm (avoiding always on the right), result in asymmetric maps, both in time to completion and minimum distance; additionally both metrics get worse as the offset increases. The consequence is that it avoids well on one side, but poorly on the other. This also shows the necessity of varying the offset in our scenario.

The different noise levels used for the simulations are presented in Table 16.3, along with their corresponding errors obtained after Kalman filtering and which are used by the collision avoidance algorithms to compute the safety margins as explained in Section 16.1.1.

Figure 16.6 shows the performance of the algorithms as function of the sensor's noise. A first observation: all three algorithms see a similar increase in the distance traveled as the noise level increases. This is especially dramatic for the VOOS algorithm. The same conclusion can



(a) SAPFHV (b) SAVO (c) SAPFUV

(d) PFOS (e) VOOS (f) ORCA

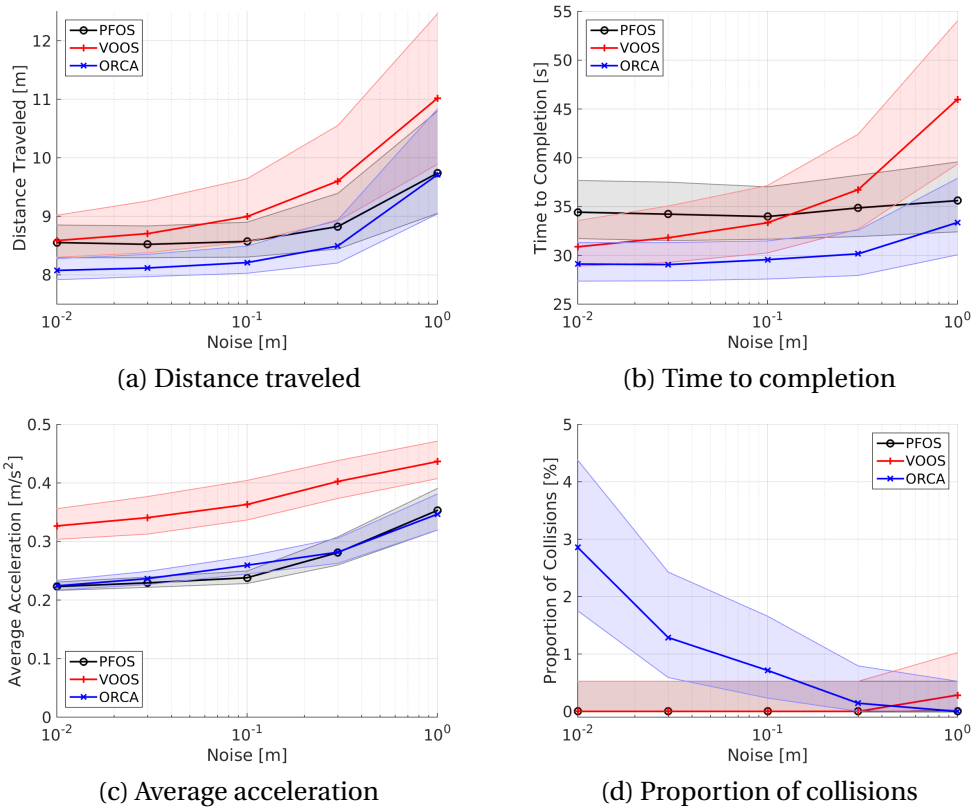
Figure 16.5 – Maps of the time to go from initial position to destination for the robots as function of the scenario's offset and angle.

Table 16.3 – Noise levels expressed with their standard deviations, and their corresponding errors in position and velocity obtained after the Kalman filter.

Noise level [m]	0.01	0.03	0.1	0.3	1
Position RMSE [m]	0.0071	0.017	0.044	0.10	0.26
Velocity RMSE [m/s]	0.077	0.11	0.15	0.19	0.26

be made for the average acceleration metric. The VO-based algorithms also show an increase in the time to complete the collision avoidance, where the PF-based algorithm does not appear to be significantly influenced in that metric. The decrease in distance performance is partially due to the increase in the margin added to the radius of the quadrotors (added to mitigate the risk due to the uncertainty). A second cause for the increase in distance traveled is that as the sensor's noise level increases, showing that noisy sensing results in poor trajectories because the quadrotors hesitate on how to perform the collision avoidance. Finally, the probability of collision for the ORCA decreases as the noise level increases. This is not directly a consequence of more noise on the sensor, but rather that this implementation of ORCA does not consider the noise on the actuation. For low sensor noise levels, the ORCA algorithm avoids with a small distance, and a collision due to some disturbance on the actuation is more likely. For high sensor noise levels, the margin added due to that sensor noise is significantly larger than what the noise on the actuation would require, lowering the probability of a collision.

Simulations have been performed for the following sensor delays: 0 *s*, 0.2 *s*, 0.4 *s*, 0.6 *s*, 0.8 *s*. The results are shown in Figure 16.7. Lag seems to have little effect on all collision avoidance algorithms. Only the ORCA algorithm sees a significant increase in its proportion of collisions, probably because the lag increases the occurrence of so called reciprocal dance of the quadro-



(a) Distance traveled

(b) Time to completion

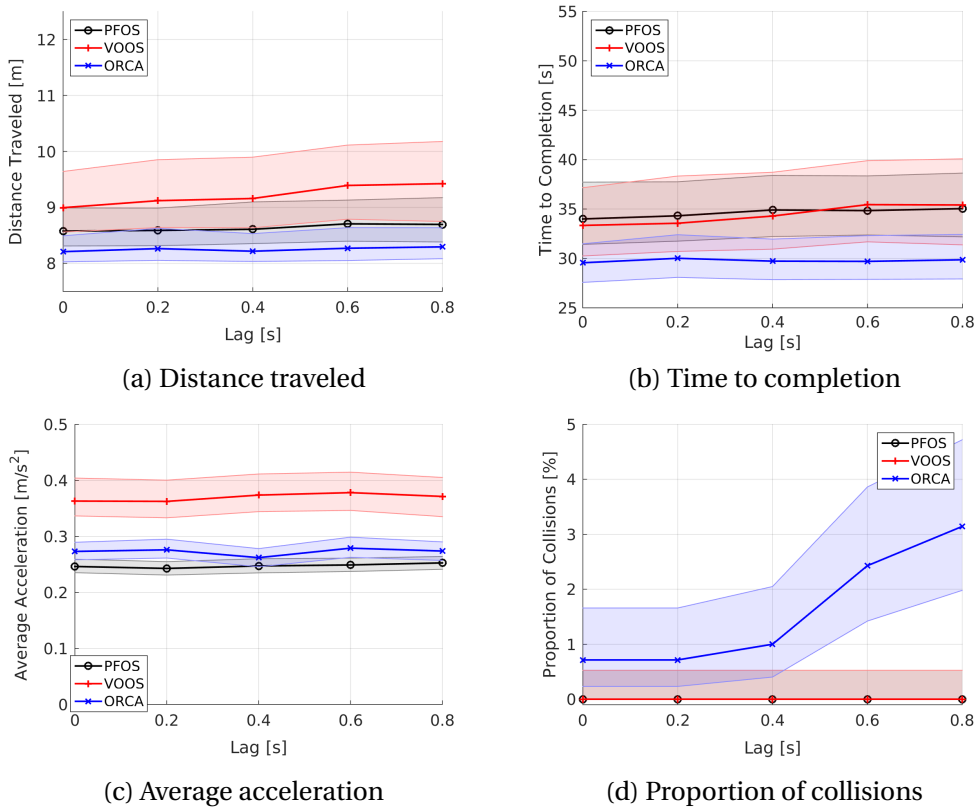
(c) Average acceleration

(d) Proportion of collisions

Figure 16.6 – The performance of three collision avoidance algorithms as function of the sensor's noise level for four metrics: (a) distance traveled, (b) time to completion, (c) average acceleration and (d) proportion of collisions.

tors [89]. The VOOS algorithm, for which the reciprocity is applied if the resulting velocity does not fall inside a velocity obstacle, seems more robust to lag. The reason why lag has such a little impact on the performance is partially due to the sensor's range being larger than the product of the lag with the speed difference (which is up to 0.6 *m/s*). As a result, even with a lag of 0.8 *s* the quadrotors have enough time to compute a reasonable collision avoidance maneuver, which results in a trajectory close to what they would produce if there was no delay on the sensor.

Simulations have been performed for the following sensor ranges: 2 *m*, 2.5 *m*, 3 *m*, 3.5 *m*, 4 *m*. The results are shown in Figure 16.8. There are several points to observe in Figure 16.8. First, the probability of a collision decreases as sensor's range increases, in particular for the ORCA and VOOS algorithm. This shows that VO-based algorithms need enough time before the potential collision point to find the right maneuver to avoid the collision. This might be a consequence of the reciprocity concept natively designed in those algorithms: because they only perform half the maneuver required (they assume the other robot will do the other half) and because there is no designated side to do the avoidance, those algorithms need several iterations before both converge to the right avoidance maneuver. A small sensor range does



(a) Distance traveled

(b) Time to completion

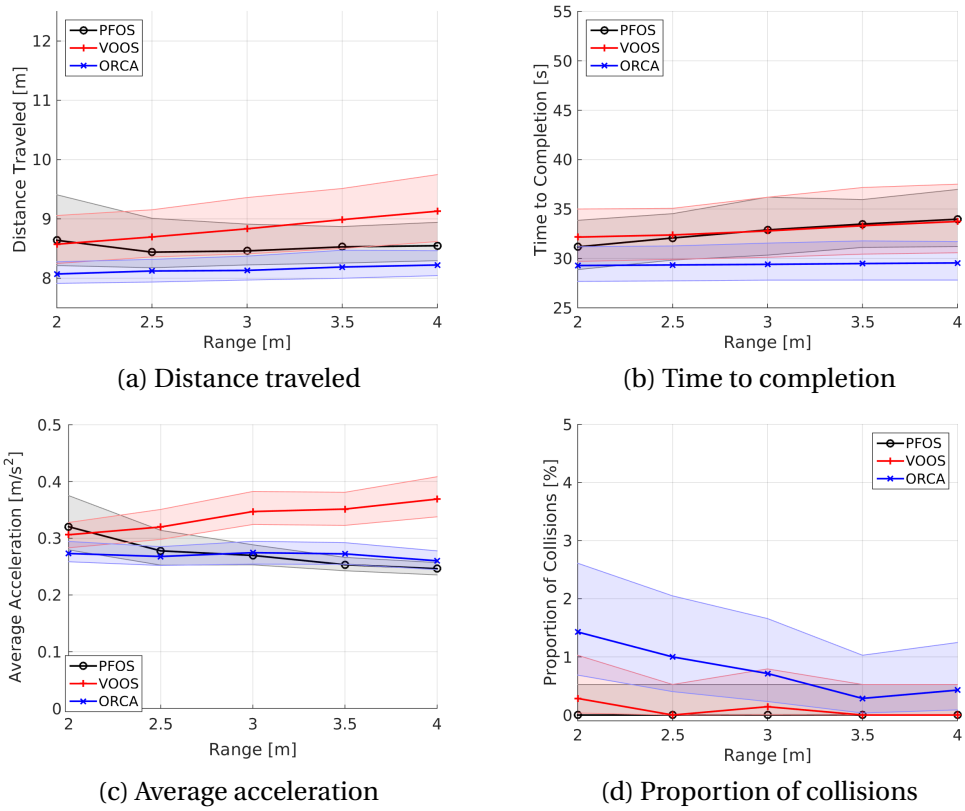
(c) Average acceleration

(d) Proportion of collisions

Figure 16.7 – The performance of three collision avoidance algorithms as function of the sensor's lag level for four metrics: (a) distance traveled, (b) time to completion, (c) average acceleration and (d) proportion of collisions.

not leave enough time to those algorithms to find the solution. Second, both the distance traveled and the time to completion metrics increase with sensor's range for all algorithms, which shows that all algorithms take advantage of the increased range to add some margin when avoiding. Finally, the average acceleration decreases as the sensor's range increases for the ORCA and PFOS, showing that a longer range allows for smoother trajectories during collision avoidance. On the contrary, the acceleration increases for the VOOS algorithm as the range increases, which is probably related to the phenomenon of so-called reciprocal dance, in which the two quadrotors have oscillating velocities because they overreact to the other robot's change in velocity [89]. The longer the range, the more time the two quadrotors have to do the reciprocal dance.

Simulations have been performed for the following sensor FOV: 210°, 220°, 230°, 240° and 250°. The results are shown in Figure 16.9. Note that only the algorithms that guarantee collision avoidance with limited FOV sensing are tested, because the scenario is not designed to test cases where a limited FOV can be challenging. As a result, algorithms that do not restrict themself to be keep their velocity in the sensor-constrained set have an unfair advantage. The three algorithms studied here are SAPFHV and SAVO (derived from the PFOS and VOOS algorithms,



(a) Distance traveled

(b) Time to completion

(c) Average acceleration

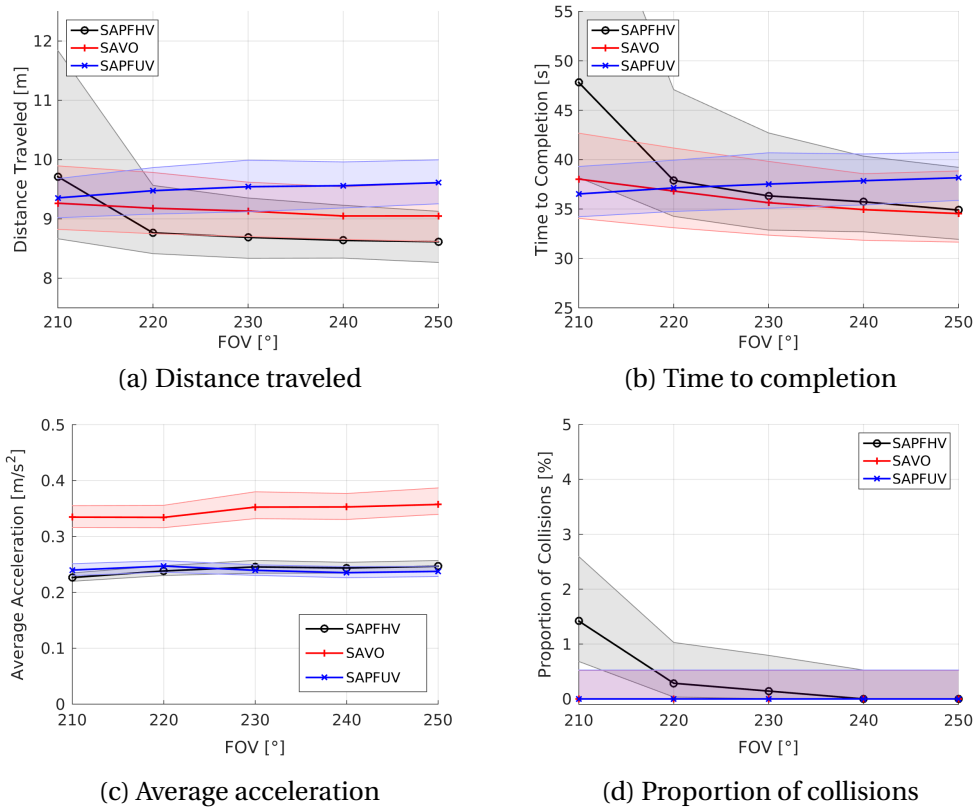
(d) Proportion of collisions

Figure 16.8 – The performance of three collision avoidance algorithms as function of the sensor's range level for four metrics: (a) distance traveled, (b) time to completion, (c) average acceleration and (d) proportion of collisions.

respectively), and the SAPFUV algorithm. The three algorithms behave differently. First, as the FOV increases, so does the distance traveled and the time to complete metrics for the SAPFUV algorithm. On the contrary, the SAPFHV and SAVO algorithms see those metrics go down as the FOV increases, showing they take advantage of the extra maneuverability to be more optimal, which is not possible for the SAPFUV algorithm because it assume a unicycle kinematics of the vehicle.. Second, all three algorithms see an increase in their average acceleration as the FOV becomes larger, showing that all algorithms show an increase in maneuverability. Finally, both the SAPFUV and the SAVO algorithms are collision-free for all FOV configurations, as the SAPFHV algorithm had collisions for a FOV lower than 230°. This indicated that the SAPFHV algorithm needs a minimum amount of maneuverability to be effective.

16.6 Discussion

Comparing algorithms is a difficult task due to the number of variables. Not only is there a large number of collision avoidance algorithms to choose from, but also the type of platform, scenario or metrics used can greatly influence the outcome. Although great care has been



(a) Distance traveled

(b) Time to completion

(c) Average acceleration

(d) Proportion of collisions

Figure 16.9 – The performance of three collision avoidance algorithms as function of the sensor's FOV level for four metrics: (a) distance traveled, (b) time to completion, (c) average acceleration and (d) proportion of collisions.

taken to be as fair as possible for all algorithms, it cannot be excluded that this work might bias some algorithms performance in one way or another because of the particular platform or scenarios used. One could extend the variability in platforms and scenarios, but the results presented in this chapter were already obtained with over 100 hours (wall-clock time) of high-fidelity simulation on a Intel i7 processor. The simulations were running with a speed up factor of around 14 times the reality. Adding a significant amount of variability to the simulations would make the simulation time required quickly unpractical. Overall, it might be impossible to be truly fair when undertaking such study, and one has to keep in mind that the results might be only valid for the specific platform and scenario combination.

Another way to reduce the bias in the comparison is to frame the question differently. This work is designed to answer the question "For the same parameters, how does an algorithm's performance evolve as function of the sensing limitation severity?". Another possible question that could be studied is "How do algorithms compare as function of sensor's limitation if one metric (e.g., proportion of collisions) is kept constant". In such study, some parameters would be changed until some performance metric is achieved, and the other metrics are compared between the different algorithms. Framing the question in this way allows a better comparison

between the algorithms, as some parameters that are only present in some algorithms are tied to some performance levels (e.g., safety requirement). This approach, although more difficult (the correct values for the parameters have first to be found, which requires much more experimental runs), would at least partially answer the fairness problem. But the fairness of the scenarios and metrics would not be solved by this approach.

This work was done in simulation, which always raises the question of the fidelity of the simulator. Replicating the simulations with real experiments would need more than 1400 hours of experiments (the simulations had a speed up of 14 times). A better solution would be to validate a few data points with real experiments in order to assess the faithfulness of the obtained results.

Summary

In this chapter, a study of the effect of sensor limitations on collision avoidance algorithms is carried out using two quadrotors in a high-fidelity, calibrated simulator. The sensor limitations are Gaussian noise, lag, limited range and FOV. The algorithms studied are of two kinds: PF-based which only use position estimation of other robots, and VO-based that also use the velocity estimate of the other robots. Overall, we show that although the VO-based algorithms are efficient in terms of avoidance overhead generated (distance and time), they are also more affected by sensor limitations than PF-based algorithms.

17 Comparison of Collision Avoidance Algorithms using Real Robots

ULTIMATELY, only real robotic experiments allow for a full validation of the algorithmic development and an assessment of algorithmic performance under realistic conditions. In this chapter, different collision avoidance algorithms are compared using the quadrotor platform equipped with the sensing system presented in Chapter 7. The algorithms compared are the SAPFUV (described in Chapter 13) and the SAVO (described in Chapter 15), both FOV-aware but belonging to the PF and to the VO class, respectively.

The collision avoidance algorithms have been made noise-resistant as described in Chapter 16. The margin in position was set to 0.45 m and the velocity margin was set to 0.2 m. The velocity margin was set to a lower value than the average velocity error measured in Chapter 7 to leave some maneuverability margin to the SAVO algorithm.

To compare the algorithms, four metrics are used: distance traveled, time to completion, average acceleration, and minimum distance. The three first metrics, distance traveled, time to completion and average acceleration are similar to the ones described in Chapter 16, with the difference that they are not averaged over the runs. The fourth metric, minimum distance between the quadrotors during the collision avoidance maneuver is preferred over the proportion of collisions as there were no collision due to a failure of the collision avoidance algorithm. The minimum distance metric is described by Equation 2.16 in Chapter 2.

Two scenarios were used for the experiments, called "head-on" and "cross". In the head-on scenario, one quadrotor starts at the position $[3.2, 0, 1.0]\,m$ and has to go to the position $[-1.8, 0, 1.0]\,m$, and vice-versa for the other quadrotor. For the cross scenario, one quadrotor starts at $[3.2, 1.0, 1.0]\,m$ and has to go to $[-1.8, -1.0, 1.0]\,m$, while the other quadrotor starts at $[-1.8, 1.0, 1.0]\,m$ and has to go to $[3.2, -1.0, 1.0]\,m$. For each scenario, 10 collision avoidance maneuvers were performed for each algorithm.

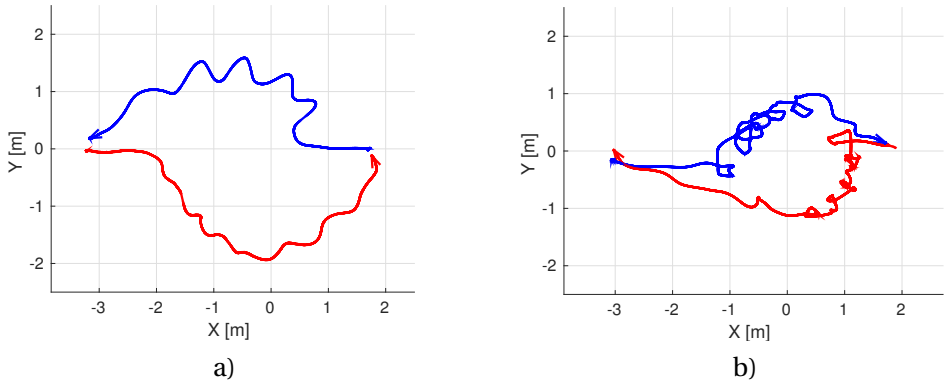
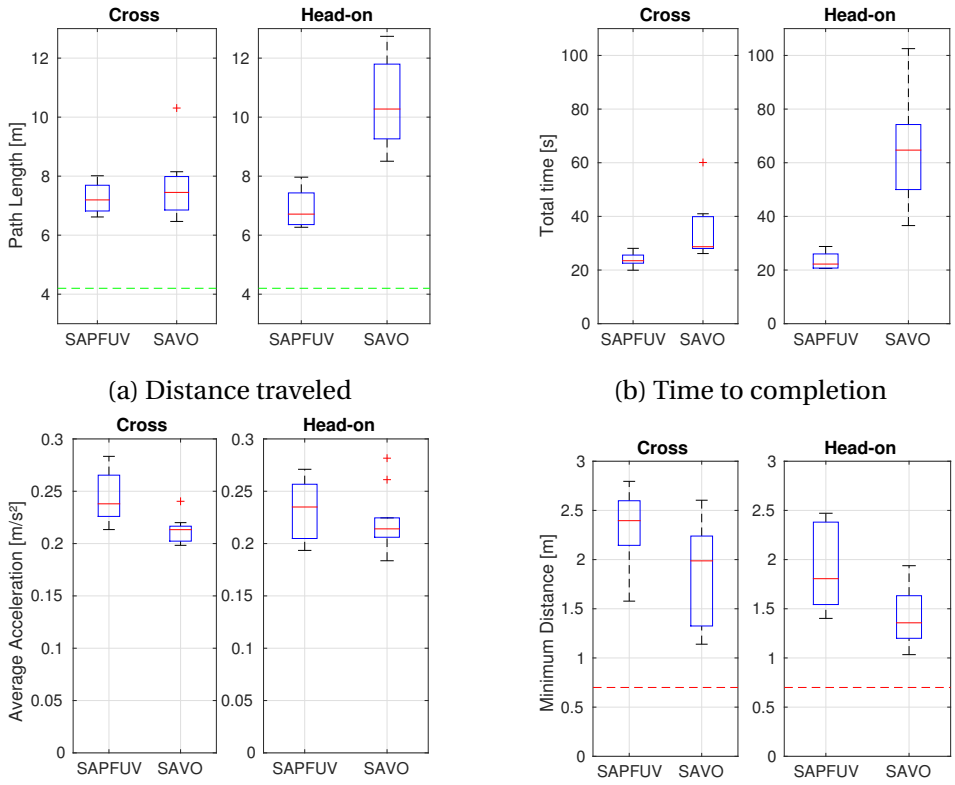


Figure 17.1 – Example of trajectories obtained with the head-on scenario. a) The SAPFUV algorithm. b) The SAVO algorithm. The arrow heads on the trajectories show the direction of motion.

17.1 Results

Figure 17.1 shows the trajectories generated by the collision avoidance algorithms for the head-on scenario. The SAPFUV algorithm performs similarly to the experiments presented in Chapter 13, although there is an additional oscillatory behavior. The trajectory of the SAVO algorithm does not perform as well as the SAPFUV algorithm, and shows a spinning behavior. This spinning is due to the large velocity uncertainty of the other quadrotor. Consequently, the search space for the SAVO algorithm is almost entirely filled with the VO of the other quadrotor; therefore, a feasible solution is almost never found, and the quadrotor switches to its fallback spinning behavior. In this case, the lag is beneficial because the quadrotors can advance a little bit while the other quadrotor is not detected. But although this lag does help to bring the quadrotors out of their local minima, it raises questions on the reliability of such system.

Figure 17.2 presents the performance of the algorithms for the four different metrics. First observation, we can notice that the minimum distance between the two quadrotors during a collision avoidance maneuver is larger for the SAPFUV algorithm, than that obtained by the SAVO algorithm. This is because the SOVA algorithm tries to avoid as optimally as possible and will take the shortest path that does not enter the VO of the other quadrotor. Thus, the SAVO algorithm avoids while leaving the minimum distance allowed (i.e., the combined radii of the quadrotor plus the safety margin). The SAPFUV algorithm on the hand, will avoid as soon as the turning command from the avoidance behavior overcomes the turning command from the navigation component, which happens at a larger distance than the minimum distance allowed. Despite leaving more distance between the quadrotors, the SAPFUV algorithm does have shorter paths, in particular for the head-on scenario. This is because the SAPFUV algorithm has a preferred side on which it will avoid, and the head-on scenario does not present any ambiguity on how the collision avoidance has to be performed. On the contrary, the SAVO algorithm does not have a preferred side, and the two quadrotors have first to agree on which side to avoid. Combined with the highly noisy sensing, the quadrotors will oscillate



(a) Distance traveled

(b) Time to completion

(c) Average acceleration

(d) Minimum distance

Figure 17.2 – The performance of the collision avoidance algorithms. The metrics are: a) distance traveled, b) time to completion, c) average acceleration and d) minimum distance between quadrotors. The green dashed line for the distance traveled represents the distance the quadrotors would take if they would travel in a straight line. The dashed red line for the minimum distance plot represents the distance at which the quadrotors would have collided.

between which side to avoid on. Only the fallback spinning behavior provides a preferred side on which to avoid, but this method is inefficient, resulting in more distance traveled. Directly correlated to the distance traveled is the time to completion. In this case, the performance is even worse for the SAVO algorithm compared to the SAPFUV algorithm. This shows that not only the quadrotors have a longer path to travel with the SAVO algorithm, they also do so significantly slower. This is because the SAVO algorithm is often in its fallback mode, in which it is stopped, while the SAPFUV algorithm always moves forward by design. Finally, the SAVO algorithm has a lower average acceleration compared to the SAPFUV, which is probably due to the slower average speed of the SAVO algorithm during the collision avoidance maneuver.

17.2 Discussion

The experiments presented in this chapter confirm the observation made in Chapter 16; the performance of a VO-based algorithm degrades significantly more than an algorithm based

on position data only when the noise on the sensor increases. In this case, the sensor noise is comparable, if not exceeding, the highest noise levels used in the simulations of Chapter 16. This is the main reason why the SAPFUV algorithm outperforms the SAVO algorithm.

The experiments presented in this chapter test the robustness of the different collision avoidance algorithms when constrained to the FOV of the sensing part, rather than validate them with scenarios where FOV constrained motion is essential to avoid collisions(such as the scenarios presented in Chapter 10).

Summary

In this chapter, a set of experiments are presented that compare different collision avoidance algorithms performance using the sensing system presented in Chapter 7. The PF-based SAPFUV algorithm has a better performance compared to the VO-based SAVO algorithm in this setup where the sensor has a high noise and lag.

Conclusion Part IV

18 Conclusion

MOST works for SAA systems clearly separate sensing and avoidance tasks, and design one without properly addressing the other, and indirectly assume that once combined, there will be no interaction issues. This work shows that a SAA system has to be designed with a holistic approach, where the limitations of one task translate into constraints for the other. This holistic approach is illustrated in the use of a limited FOV sensing. On the avoidance algorithm side, the maneuvering is constrained to the FOV of the sensor, while the sensor is required to have a FOV larger than 180° to allow the avoidance algorithm to be collision-free.

18.1 Summary

In this thesis, we developed a vision-based SAA system. Its characteristics are to be solely vision based, and uses cameras with large FOV obtained through fish-eye lenses. To detect neighboring robots, two different computer vision algorithms were used. The first relies on markers attached to the robots, and is lightweight enough to be used for experiments where the full SAA system runs on-board the robotic platforms. The second algorithm (shape detection) is able to detect the shape of the robot on the image without the need of a marker, and was developed to be used in real conditions. While this algorithm was too computationally heavy to be used on-board the robotic platforms, good results are presented for footage gathered in indoor conditions similar to the ones where markers were used.

In relation with the vision-based sensing system, we developed a methodology to perform safe collision avoidance maneuvers despite a sensing system that has a limited FOV. To our knowledge, this is the first work that explicitly studies the relation between limited FOV sensing and collision avoidance algorithms. A mathematical proof shows that the motion of a robot has to be restricted according to the FOV of the robot. We present guidelines to ensure that, if followed, a collision avoidance algorithm will be safe and produce smooth trajectories (i.e. little to no oscillatory behavior) when used with limited FOV. We designed several collision avoidance algorithms that follow the proposed guidelines, and show how to adapt common

types of collision avoidance algorithms to be compatible with limited FOV sensing. We also show the cost in efficiency and maneuverability by using a sensory system with a limited FOV for applications that require collision avoidance. The reduction in maneuverability depends not only on the sensing system's FOV but also on the kinematics of the robot itself.

Although collision avoidance with a sensing system that has a limited FOV is the main focus of this thesis, we investigated the effect of other sensor limitations and imperfections such as noise, lag and limited range. To our knowledge, this is one of the few works that studies the relations between sensing limitations in general and collision avoidance. When possible, a solution was proposed to help mitigate the problem.

18.2 Discussion

The detections from the computer vision algorithms are processed using a GM-PHD filter, a multi-target filter based on the recent RFS theory. The GM-PHD filter proved itself to be an effective and lightweight solution to estimate the position and velocity of other robots while removing false positive detections. One of the drawbacks of this filter is that it does not provide tracks (i.e. coherent trajectory over time for each tracked object) but only an estimated state for each observed robot. The lack of tracked trajectories potentially prevents it from being used with other, more predictive collision avoidance algorithms. Another drawback is that the GM-PHD filter often does not report an object when it is not detected by the camera. This can result in aggressive behaviors generated by the collision avoidance algorithm.

This work intentionally did not focus on obtaining the best sensory system possible, and instead looked into how to mitigate the sensor's imperfections with the collision avoidance algorithm. While being interesting from an academic perspective, this approach is not possible for every day and outdoor use. From our results, it seems that the proposed sensing system, which only relies on vision, will not give the required level of reliability to ensure safe navigation in common airspace. Not only should the camera run at higher resolution (the range is a few meters for a resolution of 640 by 480 pixels), but other sensors should also be used, in particular sensors that can accurately estimate the distance, as the camera gives a poor distance estimate. A distance sensor, such as a radar or a LIDAR, would also be able to reduce the ambiguity between the size of an object and the distance from it (in this work, the size is assumed to be known).

This thesis focuses on the limited FOV as sensor limitation. We have shown the danger of ignoring the limited FOV and the constraints of using a sensory system with limited FOV. But in practice, the occurrence of scenarios where the limited FOV causes a collision is small, in particular when considering that most vehicles are designed to move in the direction their sensors are headed. Furthermore, limited FOV is only one of the problems related to sensing. Other sensor limitations, such as range or frame-rate, which can also seriously affect the performance of a SAA system, as shown in Chapter 16.

As the possibilities to modify an algorithm to make it compatible with limited FOV sensing are potentially endless, this thesis does not claim to present the most efficient algorithms from this perspective. There is room for improvement, in particular concerning the PF-based algorithms. Nonetheless, this thesis opens (or at the very least expands) a new field of investigation that studies the interactions between sensors and actuators in collision avoidance applications.

The scope of this thesis is collision avoidance, and although the results can extend to other fields, such as navigation and obstacle avoidance, they are not general enough to be applied to other problems. In particular, in situations where all participants in a scene are known and the behavior of each one can be predicted, the constraints stemming from the limited FOV may be relaxed. Nonetheless, as soon as an agent has to move in an environment where it should avoid others, unknown agents, its motion should be constrained by the sensor-constrained set.

In this work, some algorithms did perform poorly during the experiments carried out with quadrotors. This is partially a result of the combination of a highly mobile platform (i.e., a quadrotor) and a slow and inaccurate sensor (i.e., a fisheye camera). In fact, actuation and sensing mutually constrain each other, and the navigation algorithms are the link that resolve the constraints between the two, usually at the disadvantage of the actuation. Ideally, actuation and sensing should be designed together in order for their respective performances to match. Retrospectively, a differential wheeled robot (such as the Khepera IV) would have been more suited for the vision system used in this thesis, as the dynamics of the robot would match with the constraints of the sensing (as in Chapter 11). Furthermore, the collision avoidance algorithms have been designed taking exclusively into account the kinematic laws of the vehicles. On the other hand, dynamic laws play a major role in platforms such as quadrotors. This unfaithfulness to the vehicle reality is one of the reasons why some algorithms performed poorly when validated with real experiments, such as those of Chapter 17.

While many experiments have been performed with real quadrotors running all the algorithms on-board, no experiment has been done outside, in real conditions. The main reason this were not achieved was because the computer vision algorithms were not suited for outdoor experiments; the marker-based approach requires that the tracked color is not present in the scene, and the shape-detection based algorithm was not fast enough to run onboard of the quadrotors.

18.3 Outlook

Improvements on the sensing part of this work would be mostly hardware-related, and would include the use of higher resolution cameras and the addition of other sensor modalities. Some future work on the sensors should consider to add a radar or a LIDAR. Because those sensors usually have a FOV smaller than 180° for sizes that can fit on a UAV, a dual sensor system with a static fish-eye lens camera and a LIDAR on a pan-tilt system could be an interesting solution. The fish-eye camera would continuously monitor the FOV of the robot and the pan-

tilt LIDAR would provide a better measurement of the tracked objects. By adding a camera with a small FOV on the pan-tilt, it is even possible to perform the object identification on an image with higher quality (less distortion, higher resolution) than what can be obtained with a fish-eye lens, and use a computationally fast algorithm that needs a small number of pixels to do the detection (e.g., morphological computer vision). This combination of different sensors is probably the only way to achieve the performance necessary to fly fully autonomous UAVs outdoor. But it is yet to be known if current technology is capable of enough computational power and camera resolution to fulfill the requirements of flying in common airspace. Furthermore, a pan-tilt system adds on complexity, weight and fragility to the system, trading in sensing reliability for sensing accuracy and thus reliability on the collision avoidance.

The methodology proposed in this thesis can be applied to virtually any collision avoidance algorithm, and several algorithm adaptations have been presented. But the effectiveness of each algorithm is heavily affected by the size of the FOV. An important question that remains to be addressed is what are the conditions for which an algorithm will work well with limited FOV and in which situations does the algorithm perform poorly. A partial answer is given by the observation that differential-wheeled robots do see little to no effect on their trajectory when using a limited FOV sensor.

In this work, the robots do not cooperate or even assume a possible cooperation. They only assume the implicit reciprocity that every robot will avoid collision (if such threat is detected). Adding cooperation can potentially enhance a SAA system in several ways. First, robots can extend their FOV by either sharing the sensor measurements or the tracks of other objects. By increasing the FOV, the maneuverability of the robots can be greatly improved, in particular for near-holonomic robots such as quadrotors. Additionally, by communicating their ego-state, the robots can accurately and reliably track each other, and the avoidance will become smoother and safer as a consequence. Another possibility for cooperation is in the collision avoidance algorithm. Applying concepts such as reciprocity, commonly found in VO-based approaches, can reduce the amount of maneuvering required by each robot while keeping the same level of safety.

Bibliography

[1] L. Zongjian, "UAV for mapping—low altitude photogrammetric survey", *International Archives of Photogrammetry and Remote Sensing, Beijing, China*, vol. 37, pp. 1183–1186, 2008.

[2] I. Sa and P. Corke, "Vertical infrastructure inspection using a quadcopter and shared autonomy control", in *Field and Service Robotics*, Springer, 2014, pp. 219–232.

[3] C. Deng, S. Wang, Z. Huang, Z. Tan, and J. Liu, "Unmanned aerial vehicles for power line inspection: a cooperative way in platforms and communications", *J. Commun*, vol. 9, no. 9, pp. 687–692, 2014.

[4] M. A. Goodrich, B. S. Morse, D. Gerhardt, J. L. Cooper, M. Quigley, J. A. Adams, and C. Humphrey, "Supporting wilderness search and rescue using a camera-equipped mini UAV", *Journal of Field Robotics*, vol. 25, no. 1-2, pp. 89–110, 2008.

[5] 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*, vol. 19, no. 3, pp. 46–56, 2012.

[6] L. Johnson, S. Herwitz, S. Dunagan, B. Lobitz, D. Sullivan, and R. Slye, "Collection of ultra high spatial and spectral resolution image data over california vineyards with a small UAV", in *Proceedings of the 30th International Symposium on Remote Sensing of Environment*, vol. 20, 2003.

[7] W. C. Evans, D. Dias, S. Roelofsen, and A. Martinoli, "Environmental field estimation with hybrid-mobility sensor networks", in *IEEE International Conference on Robotics and Automation*, 2016, pp. 5301–5308.

[8] D. Perez, I. Maza, F. Caballero, D. Scarlatti, E. Casado, and A. Ollero, "A ground control station for a multi-UAV surveillance system", *Journal of Intelligent & Robotic Systems*, vol. 69, no. 1-4, pp. 119–130, 2013.

[9] E. A. Cappo, A. Desai, and N. Michael, "Robust coordinated aerial deployments for theatrical applications given online user interaction via behavior composition", in *International Symposium on Distributed Autonomous Robotic Systems; Springer Proceedings in Advanced Robotics (2018)*, 2016.

[10] K. Miyoshi, R. Konomura, and K. Hori, "Entertainment multi-rotor robot that realises direct and multimodal interaction", in *Proceedings of the 28th International BCS Human Computer Interaction Conference on HCI 2014-Sand, Sea and Sky-Holiday HCI*, BCS, 2014, pp. 218–221.

[11] K. L. Koster, *Delivery platform for unmanned aerial vehicles*, US Patent App. 14/560,821, 2014.

[12] C. A. Thiels, J. M. Aho, S. P. Zietlow, and D. H. Jenkins, "Use of unmanned aerial vehicles for medical product transport", *Air Medical Journal*, vol. 34, no. 2, pp. 104–108, 2015.

[13] R. P. Spraying, "Commercial applications of UAV's in japanese agriculture", *Technical Conference and Workshop on Unmanned Aerospace Vehicles*, 2002.

[14] M. Hochstenbach, C. Notteboom, B. Theys, and J. De Schutter, "Design and control of an unmanned aerial vehicle for autonomous parcel delivery with transition from vertical take-off to forward flight–vertikul, a quadcopter tailsitter", *International Journal of Micro Air Vehicles*, vol. 7, no. 4, pp. 395–405, 2015.

[15] H. Zhang and J. P. Ostrowski, "Visual servoing with dynamics: control of an unmanned blimp", in *Robotics and Automation, 1999. Proceedings.*, vol. 1, 1999, pp. 618–623.

[16] A. Elfes, S. S. Bueno, M. Bergerman, and J. G. Ramos, "A semi-autonomous robotic airship for environmental monitoring missions", in *IEEE International Conference on Robotics and Automation*, vol. 4, 1998, pp. 3449–3455.

[17] H. Xiang and L. Tian, "Development of a low-cost agricultural remote sensing system based on an autonomous unmanned aerial vehicle (uav)", *Biosystems engineering*, vol. 108, no. 2, pp. 174–190, 2011.

[18] A. Rango, A. Laliberte, C. Steele, J. E. Herrick, B. Bestelmeyer, T. Schmugge, A. Roanhorse, and V. Jenkins, "Using unmanned aerial vehicles for rangelands: current applications and future potentials", *Environmental Practice*, vol. 8, no. 3, pp. 159–168, 2006.

[19] X. Yu and Y. Zhang, "Sense and avoid technologies with applications to unmanned aircraft systems: review and prospects", *Progress in Aerospace Sciences*, vol. 74, pp. 152–166, 2015.

[20] C. Geyer, S. Singh, and L. J. Chamberlain, "Avoiding collisions between aircraft: state of the art and requirements for UAVs operating in civilian airspace", *Technical report, Robotics Institute, Carnegie Mellon University. CMU-RI-TR-08-03.*, 2008.

[21] C. G. Santel, P. Gerber, S. Mehringskoetter, V. Schochlow, J. Vogt, and U. Klingauf, "How glider pilots misread the flarm collision alerting display: a laboratory study.", *Aviation Psychology and Applied Human Factors*, vol. 4, no. 2, p. 86, 2014.

[22] P. Cornic, P. Garrec, S. Kemkemian, and L. Ratton, "Sense and avoid radar using data fusion with other sensors", in *IEEE International Aerospace Conference*, 2011, pp. 1–14.

[23] A. Moses, M. J. Rutherford, and K. P. Valavanis, "Radar-based detection and identification for miniature air vehicles", in *IEEE International Conference on Control Applications*, 2011, pp. 933–940.

[24] B. C. Karhoff, J. I. Limb, S. W. Oravsky, and A. D. Shephard, "Eyes in the domestic sky: an assessment of sense and avoid technology for the army's"warrior"unmanned aerial vehicle", in *IEEE Systems and Information Engineering Design Symposium*, 2006, pp. 36–42.

[25] R. Sabatini, A. Gardi, and M. Richardson, "Lidar obstacle warning and avoidance system for unmanned aircraft", *International Journal of Mechanical, Aerospace, Industrial and Mechatronics Engineering*, vol. 8, no. 4, pp. 718–729, 2014.

[26] S. Ramasamy, R. Sabatini, A. Gardi, and J. Liu, "Lidar obstacle warning and avoidance system for unmanned aerial vehicle sense-and-avoid", *Aerospace Science and Technology*, vol. 55, pp. 344–358, 2016.

[27] A. Finn and S. Franklin, "Acoustic sense & avoid for UAV's", in *IEEE International Conference on Intelligent Sensors, Sensor Networks and Information Processing*, 2011, pp. 586–589.

[28] C. Geyer, D. Dey, and S. Singh, "Prototype sense-and-avoid system for UAVs", *Technical report, Robotics Institute, Carnegie Mellon University. CMU-RI-TR-09-09.*, 2009.

[29] D. J. Wing and W. B. Cotton, "For spacious skies: self-separation with 'autonomous flight rules' in us domestic airspace", in *11th AIAA Aviation Technology, Integration, and Operations Conference*, 2011, pp. 20–22.

[30] B. Stark, B. Stevenson, and Y. Chen, "Ads-b for small unmanned aerial systems: case study and regulatory practices", in *International Conference on Unmanned Aircraft Systems*, 2013, pp. 152–159.

[31] X. Prats, L. Delgado, J. Ramirez, P. Royo, and E. Pastor, "Requirements, issues, and challenges for sense and avoid in unmanned aircraft systems", *Journal of aircraft*, vol. 49, no. 3, pp. 677–687, 2012.

[32] A. Mujumdar and R. Padhi, "Evolving philosophies on autonomous obstacle/collision avoidance of unmanned aerial vehicles.", *Journal of Aerospace Computing, Information, and Communication*, vol. 8, no. 2, pp. 17–41, 2011.

[33] C. Goerzen, Z. Kong, and B. Mettler, "A survey of motion planning algorithms from the perspective of autonomous UAV guidance", *Journal of Intelligent and Robotic Systems*, vol. 57, no. 1-4, p. 65, 2010.

[34] B. Albaker and N. Rahim, "A survey of collision avoidance approaches for unmanned aerial vehicles", in *International Conference for Technical Postgraduates*, 2009, pp. 1–7.

[35] S. Roelofsen, D. Gillet, and A. Martinoli, "Reciprocal collision avoidance for quadrotors using on-board visual detection", in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2015, pp. 4810–4817.

[36] K. R. Sapkota, S. Roelofsen, A. Rozantsev, V. Lepetit, D. Gillet, P. Fua, and A. Martinoli, "Vision-based unmanned aerial vehicle detection and tracking for sense and avoid systems", in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2016, pp. 1556–1561.

[37] S. Roelofsen, A. Martinoli, and D. Gillet, "Distributed deconfliction algorithm for unmanned aerial vehicles with limited range and field of view sensors", in *American Control Conference*, 2015, pp. 4356–4361.

[38] S. Roelofsen, A. Martinoli, and D. Gillet, "3D collision avoidance algorithm for unmanned aerial vehicles with limited field of view constraints", in *Conference on Decision and Control*, 2016, pp. 2555–2560.

[39] S. Roelofsen, D. Gillet, and A. Martinoli, "Collision avoidance with limited field of view sensing: a velocity obstacle approach", in *IEEE International Conference on Robotics and Automation*, 2017, pp. 1922–1927.

[40] S. Roelofsen, D. Gillet, and A. Martinoli, "A comparative study of collision avoidance algorithms for unmanned aerial vehicles: performance and robustness to noise", in *International Symposium on Distributed Autonomous Robotic Systems; Springer Proceedings in Advanced Robotics (2018)*, 2016.

[41] S. Roelofsen, D. Gillet, and A. Martinoli, "The effects of sensory limitations on avoidance: a comparative study of collision avoidance algorithms for unmanned aerial vehicles", in *Autonomous Robots, to be revised and resubmitted*, 2018.

[42] *Asctec_hl_framework ros package*, http:http://wiki.ros.org/asctec_mav_framework, 2015-02-04.

[43] G. Fasano, D. Accardo, A. E. Tirri, A. Moccia, and E. De Lellis, "Morphological filtering and target tracking for vision-based uas sense and avoid", in *International Conference on Unmanned Aircraft Systems*, 2014, pp. 430–440.

[44] R. Carnie, R. Walker, and P. Corke, "Image processing algorithms for UAV "sense and avoid"", in *IEEE International Conference on Robotics and Automation*, 2006, pp. 2848–2853.

[45] T. Zsedrovits, A. Zarandy, B. Vanek, T. Peni, J. Bokor, and T. Roska, "Visual detection and implementation aspects of a UAV see and avoid system", in *IEEE/RSJ International Conference on Circuit Theory and Design*, 2011, pp. 472–475.

[46] L. Mejias, S. McNamara, J. Lai, and J. Ford, "Vision-based detection and tracking of aerial targets for UAV collision avoidance", in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2010, pp. 87–92.

[47] D. Dias, R. Ventura, P. Lima, and A. Martinoli, "On-board vision-based 3d relative localization system for multiple quadrotors", in *IEEE International Conference on Robotics and Automation*, 2016, pp. 1181–1187.

[48] O. Michel, "Cyberbotics ltd. webots™: professional mobile robot simulation", *International Journal of Advanced Robotic Systems*, vol. 1, no. 1, p. 5, 2004.

[49] D. Mellinger and V. Kumar, "Minimum snap trajectory generation and control for quadrotors", in *IEEE International Conference on Robotics and Automation*, 2011, pp. 2520–2525.

[50] M. Abdolhosseini, Y. Zhang, and C. A. Rabbath, "An efficient model predictive control scheme for an unmanned quadrotor helicopter", *Journal of Intelligent & Robotic Systems*, pp. 1–12, 2013.

[51] Y. Pu, M. N. Zeilinger, and C. N. Jones, "Fast alternating minimization algorithm for model predictive control", *IFAC Proceedings Volumes*, vol. 47, no. 3, pp. 11 980–11 986, 2014.

[52] Y. Bar-Shalom and X.-R. Li, *Multitarget-multisensor tracking: principles and techniques*. YBs London, UK: 1995, vol. 19.

[53] Y. Bar-Shalom, T. Fortmann, and M. Scheffe, "Joint probabilistic data association for multiple targets in clutter", in *Proc. Conf. on Information Sciences and Systems*, 1980, pp. 404–409.

[54] S. S. Blackman, "Multiple hypothesis tracking for multiple target tracking", *Aerospace and Electronic Systems Magazine*, vol. 19, no. 1, pp. 5–18, 2004.

[55] R. P. Mahler, *Statistical multisource-multitarget information fusion*. Artech House, Inc., 2007.

[56] Y. Bar-Shalom, F. Daum, and J. Huang, "The probabilistic data association filter", *IEEE Control Systems*, vol. 29, no. 6, 2009.

[57] S. Särkkä, A. Vehtari, and J. Lampinen, "Rao-blackwellized monte carlo data association for multiple target tracking", in *Proceedings of the International Conference on Information Fusion*, vol. 1, 2004, pp. 583–590.

[58] D. Schuhmacher, B.-T. Vo, and B.-N. Vo, "A consistent metric for performance evaluation of multi-object filters", *IEEE Transactions on Signal Processing*, vol. 56, no. 8, pp. 3447–3457, 2008.

[59] M. B. Guldogan, D. Lindgren, F. Gustafsson, H. Habberstad, and U. Orguner, "Multi-target tracking with PHD filter using doppler-only measurements", *Digital Signal Processing*, vol. 27, pp. 1–11, 2014.

[60] M. Vasic, D. Mansolino, and A. Martinoli, "A system implementation and evaluation of a cooperative fusion and tracking algorithm based on a Gaussian mixture PHD filter", in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2016, pp. 4172–4179.

[61] R. Hoseinnezhad, B.-N. Vo, B.-T. Vo, and D. Suter, "Visual tracking of numerous targets via multi-bernoulli filtering of image data", *Pattern Recognition*, vol. 45, no. 10, pp. 3625–3635, 2012.

[62] E. Pollard, A. Plyer, B. Pannetier, F. Champagnat, and G. Le Besnerais, "GM-PHD filters for multi-object tracking in uncalibrated aerial videos", in *Information Fusion, 2009. FUSION'09. 12th International Conference on*, 2009, pp. 1171–1178.

[63] B.-N. Vo and W.-K. Ma, "The gaussian mixture probability hypothesis density filter", *IEEE Trans. on Signal Processing*, vol. 54, no. 11, pp. 4091–4104, 2006.

[64] E. A. Wan and R. Van Der Merwe, "The unscented kalman filter for nonlinear estimation", in *Adaptive Systems for Signal Processing, Communications, and Control Symposium 2000. AS-SPCC. The IEEE 2000*, 2000, pp. 153–158.

[65] S. J. Julier, J. K. Uhlmann, and H. F. Durrant-Whyte, "A new approach for filtering nonlinear systems", in *American Control Conference, Proceedings of the 1995*, vol. 3, 1995, pp. 1628–1632.

[66] *Camera_calibration*, http://wiki.ros.org/camera_calibration, 2015-02-25.

[67] K. Granstrom, S. Reuter, D. Meissner, and A. Scheel, "A multiple model PHD approach to tracking of cars under an assumed rectangular shape", in *IEEE 17th International Conference on Information Fusion*, 2014, pp. 1–8.

[68] D. Scaramuzza, "Omnidirectional vision: from calibration to robot motion estimation", PhD thesis, Citeseer, 2008.

[69] D. Scaramuzza. (2005). Ocamcalib, [Online]. Available: https://sites.google.com/site/scarabotix/ocamcalib-toolbox (visited on 06/27/2017).

[70] J.-W. Park, H.-D. Oh, and M.-J. Tahk, "UAV collision avoidance based on geometric approach", in *SICE Annual Conference, 2008*, 2008, pp. 2122–2126.

[71] B. A. White, H.-S. Shin, and A. Tsourdos, "UAV obstacle avoidance using differential geometry concepts", *IFAC Proceedings Volumes*, vol. 44, no. 1, pp. 6325–6330, 2011.

[72] D. Fox, W. Burgard, and S. Thrun, "The dynamic window approach to collision avoidance", *IEEE Robotics & Automation Magazine*, vol. 4, no. 1, pp. 23–33, 1997.

[73] C.-K. Lai, M. Lone, P. Thomas, J. Whidborne, and A. Cooke, "On-board trajectory generation for collision avoidance in unmanned aerial vehicles", in *IEEE International Aerospace Conference*, 2011, pp. 1–14.

[74] X. Wang, V. Yadav, and S. Balakrishnan, "Cooperative UAV formation flying with obstacle/collision avoidance", *IEEE Transactions on Control Systems Technology*, vol. 15, no. 4, pp. 672–679, 2007.

[75] D. Heß, M. Althoff, and T. Sattel, "Formal verification of maneuver automata for parameterized motion primitives", in *Intelligent Robots and Systems (IROS 2014), 2014 IEEE/RSJ International Conference on*, IEEE, 2014, pp. 1474–1481.

[76] S. Temizer, M. J. Kochenderfer, L. P. Kaelbling, T. Lozano-Pérez, and J. K. Kuchar, "Collision avoidance for unmanned aircraft using markov decision processes", in *AIAA Guidance, Navigation, and Control Conference*, 2010.

[77] M. Coppola, K. N. McGuire, K. Y. Scheper, and G. C. de Croon, "On-board communication-based relative localization for collision avoidance in micro air vehicle teams", *SemanticScholar*, 2017.

[78] E. Lalish, K. A. Morgansen, and T. Tsukamaki, "Decentralized reactive collision avoidance for multiple unicycle-type vehicles", in *American Control Conference*, 2008, pp. 5055–5061.

[79] A. Chakravarthy and D. Ghose, "Obstacle avoidance in a dynamic environment: a collision cone approach", *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans*, vol. 28, no. 5, pp. 562–574, 1998.

[80] V. Duong, E. Hoffman, J.-P. Nicolaon, L. Floc'hic, and A. Bossu, "Extended flight rules to apply to the resolution of encounters in autonomous airborne separation", *Rapport technique, Eurocontrol*, p. 18, 1996.

[81] Y. I. Jenie, E.-J. v. Kampen, C. C. de Visser, J. Ellerbroek, and J. M. Hoekstra, "Selective velocity obstacle method for deconflicting maneuvers applied to unmanned aerial vehicles", *Journal of Guidance, Control, and Dynamics*, vol. 38, no. 6, pp. 1140–1146, 2015.

[82] L. Makarem and D. Gillet, "Decentralized coordination of autonomous vehicles at intersections", in *IFAC World Congress*, vol. 18, 2011, pp. 13 046–13 051.

[83] M. T. Wolf and J. W. Burdick, "Artificial potential functions for highway driving with collision avoidance", in *IEEE International Conference on Robotics and Automation*, 2008, pp. 3731–3736.

[84] L. Makarem, J.-P. Kneib, D. Gillet, H. Bleuler, M. Bouri, L. Jenni, F. Prada, and J. Sanchez, "Collision avoidance in next-generation fiber positioner robotic systems for large survey spectrographs", *Astronomy & Astrophysics*, vol. 566, A84, 2014.

[85] K. Sigurd and J. How, "UAV trajectory design using total field collision avoidance", in *AIAA Guidance, Navigation, and Control Conference and Exhibit*, 2003, p. 5728.

[86] P. Panyakeow and M. Mesbahi, "Deconfliction algorithms for a pair of constant speed unmanned aerial vehicles", *IEEE Trans. on Aerospace and Electronic Systems*, vol. 50, no. 1, pp. 456–476, 2014.

[87] S. Mastellone, D. M. Stipanović, C. R. Graunke, K. A. Intlekofer, and M. W. Spong, "Formation control and collision avoidance for multi-agent non-holonomic systems: theory and experiments", *The International Journal of Robotics Research*, vol. 27, no. 1, pp. 107–126, 2008.

[88] A. Budiyanto, A. Cahyadi, T. B. Adji, and O. Wahyunggoro, "UAV obstacle avoidance using potential field under dynamic environment", in *International Conference on Control, Electronics, Renewable Energy and Communications*, 2015, pp. 187–192.

[89] J. Snape, J. van den Berg, S. J. Guy, and D. Manocha, "The hybrid reciprocal velocity obstacle", *IEEE Transactions on Robotics*, vol. 27, no. 4, pp. 696–706, 2011.

[90] J. Van Den Berg, S. J. Guy, M. Lin, and D. Manocha, "Reciprocal n-body collision avoidance", in *Robotics research; Springer Tracts in Advanced Robotics (2011)*, Springer, 2011, pp. 3–19.

[91] D. Bareiss and J. van den Berg, "Generalized reciprocal collision avoidance", *The International Journal of Robotics Research*, vol. 34, no. 12, pp. 1501–1514, 2015.

[92] J. Alonso-Mora, A. Breitenmoser, P. Beardsley, and R. Siegwart, "Reciprocal collision avoidance for multiple car-like robots", in *IEEE International Conference on Robotics and Automation*, 2012, pp. 360–366.

[93] Y. Kuwata, M. T. Wolf, D. Zarzhitsky, and T. L. Huntsberger, "Safe maritime navigation with colregs using velocity obstacles", in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2011, pp. 4728–4734.

[94] P. Conroy, D. Bareiss, M. Beall, and J. van den Berg, "3-D reciprocal collision avoidance on physical quadrotor helicopters with on-board sensing for relative positioning", *ArXiv preprint arXiv:1411.3794*, 2014.

[95] D. V. Dimarogonas and K. J. Kyriakopoulos, "Decentralized navigation functions for multiple robotic agents with limited sensing capabilities", *Journal of Intelligent & Robotic Systems*, vol. 48, no. 3, pp. 411–433, 2007.

[96] P. Panyakeow and M. Mesbahi, "Decentralized deconfliction algorithms for unicycle UAVs", in *American Control Conference*, 2010, pp. 794–799.

[97] D. E. Chang and J. E. Marsden, "Gyroscopic forces and collision avoidance with convex obstacles", in *New Trends in Nonlinear Dynamics and Control and Their Applications*, 2003, pp. 145–159.

[98] J. Saunders and R. Beard, "Reactive vision based obstacle avoidance with camera field of view constraints", in *AIAA Guidance, Navigation, and Control Conference*, 2008, pp. 1–15.

[99] C. Carbone, U. Ciniglio, F. Corraro, and S. Luongo, "A novel 3D geometric algorithm for aircraft autonomous collision avoidance", in *IEEE Conference on Decision and Control*, 2006, pp. 1580–1585.

[100] A. Mcfadyen, L. Mejias, P. Corke, and C. Pradalier, "Aircraft collision avoidance using spherical visual predictive control and single point features", in *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2013, pp. 50–56.

[101] D. Burkhardt and I. de la Motte, "How stalk-eyed flies eye stalk-eyed flies: observations and measurements of the eyes of cyrtodiopsis whitei (diopsidae, diptera)", *Journal of Comparative Physiology*, vol. 151, no. 4, pp. 407–421, 1983.

[102] J. A. Waldvogel, "The bird's eye view", *American Scientist*, vol. 78, no. 4, pp. 342–353, 1990.

[103] M. P. Jones, K. E. P. Jr, and D. Ward, "Avian vision: a review of form and function with special consideration to birds of prey", *Journal of Exotic Pet Medicine*, vol. 16, no. 2, pp. 69 –87, 2007, Ophthalmology.

[104] Q. Zhang, G. Leng, and V. Govindaraju, "Duration of collision-free motion of unmanned vehicles in a confined area", *Robotica*, vol. 34, no. 02, pp. 347–360, 2016.

[105] ASTM, *F2411-07 standard specification for design and performance of an airborne sense-and-avoid system (withdrawn 2014)*, 2014.

[106] V. Braitenberg, *Vehicles: Experiments in synthetic psychology*. MIT press, 1986.

[107] S. M. Lavalle. (2012). Planning algorithms, [Online]. Available: http://planning.cs.uiuc.edu/node369.html (visited on 05/03/2017).

[108] J. van den Berg, S. J. Guy, J. Snape, M. C. Lin, and D. Manocha, *RVO2 Library: Reciprocal Collision Avoidance for Real-Time Multi-Agent Simulation*, http://gamma.cs.unc.edu/RVO2/, 2008–2015.

Curriculum Vitae

Steven Roelofsen

Education

2012-2017	PhD in Robotics, Control and Intelligent Systems *Ecole Polytechnique Fédérale de Lausanne (EPFL), Switzerland*
2007-2012	B.S. and M.S. in Micro-Engineering *Ecole Polytechnique Fédérale de Lausanne (EPFL), Switzerland*

Experience

2010	Research Assistant, *EPFL, Switzerland*
2009-2012	Teaching Assistant, *EPFL, Switzerland*

Publications

Refereed Conference Proceedings

1. S. Roelofsen, A. Martinoli, and D. Gillet, "Distributed deconfliction algorithm for unmanned aerial vehicles with limited range and field of view sensors", in *American Control Conference*, 2015, pp. 4356–4361

2. S. Roelofsen, D. Gillet, and A. Martinoli, "Reciprocal collision avoidance for quadrotors using on-board visual detection", in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2015, pp. 4810–4817

3. W. C. Evans, D. Dias, S. Roelofsen, and A. Martinoli, "Environmental field estimation with hybrid-mobility sensor networks", in *IEEE International Conference on Robotics and Automation*, 2016, pp. 5301–5308

4. K. R. Sapkota, S. Roelofsen, A. Rozantsev, V. Lepetit, D. Gillet, P. Fua, and A. Martinoli, “Vision-based unmanned aerial vehicle detection and tracking for sense and avoid systems”, in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2016, pp. 1556–1561

5. S. Roelofsen, A. Martinoli, and D. Gillet, “3D collision avoidance algorithm for unmanned aerial vehicles with limited field of view constraints”, in *Conference on Decision and Control*, 2016, pp. 2555–2560

6. S. Roelofsen, D. Gillet, and A. Martinoli, “A comparative study of collision avoidance algorithms for unmanned aerial vehicles: performance and robustness to noise”, in *International Symposium on Distributed Autonomous Robotic Systems; Springer Proceedings in Advanced Robotics (2018)*, 2016

7. S. Roelofsen, D. Gillet, and A. Martinoli, “Collision avoidance with limited field of view sensing: a velocity obstacle approach”, in *IEEE International Conference on Robotics and Automation*, 2017, pp. 1922–1927

Languages

French	native
English	fluent
German	limited proficiency
Dutch	limited proficiency