NTNU
Norwegian University of
Science and Technology

Collision Avoidance for Multirotor Inspection

Vetle Andre Bjelland

Master of Science in Cybernetics and Robotics
Submission date: June 2018
Supervisor: Tor Arne Johansen, ITK
Co-supervisor: Kristian Klausen, ITK

Norwegian University of Science and Technology
Department of Engineering Cybernetics

NTNU
Norwegian University of
Science and Technology

Faculty of Information Technology
and Electrical Engineering
Department of Engineering Cybernetics

MSC THESIS DESCRIPTION SHEET

Name: Vetle André Bjelland
Department: Engineering Cybernetics
Thesis title: Collision avoidance for multirotor inspection

Thesis Description:

A system for automatic and safe inspection of tanks and other infrastructure with Unmanned Aerial Vehicles (UAVs) is under investigation at the department. Critical to the system is methods for collision and detection of obstacles, such as wires, pipes and walls.

The system is a part of an ongoing commercialization process in cooperation with NTNU TTO, and the student will sign a "Standardavtale" with NTNU TTO.

This thesis aims to investigate algorithms for obstacle avoidance and proximity keeping.

The following items should be considered:

1. Conduct a literature review on collision avoidance algorithms for 2- and 3D, with emphasis on 3D methods
2. Develop a simulation tool in Matlab or Python to test algorithms
3. Consider the sensory information generated by multiple on-board Radar-sensors.
4. Develop and test the most promising collision avoidance control algorithms investigated in 1. Divide in two categories; Algorithms requiring full environment knowledge and those with only local radar sensors.
5. Discuss how the global and reactive algorithms can be combined to create robust systems.
6. Develop and test behavioral algorithms (in connection with 4.) that enables easy safe pilot operations, such as enforcing a fixed distance to a wall to be traversed, override of pilot inputs in case of imminent collision, etc.
7. Demonstrate two main scenarios;
    a. Adequate stopping in front of detected obstacle even when pilot pushes forward
    b. When moving along wall, the system avoids object appearing to the side of the drone
8. Conclude findings in a report. Include Matlab/Python/C-code as digital appendices together with a user-guide.
9.

Start date: 2018-01-08
Due date: 2018-06-04

Thesis performed at: Department of Engineering Cybernetics, NTNU
Supervisor: Professor Thor Arne Johansen, Dept. of Eng. Cybernetics, NTNU
Co-Supervisor: Dr. Kristian Klausen, Dept. of Eng. Cybernetics, NTNU

Thank you for the music, Tim

Abstract

During the last couple of years, the goal of using unmanned aerial vehicles (UAV) for inspection purposes has become within reach. This thesis considers the different algorithms for implementing a high level controller for an anti-collision system for a multirotor inspection drone operating inside a shipping tank. The desired platform for this thesis is a quadcopter UAV with six on-board radar sensor, providing environmental knowledge. A literature review of existing collision avoidance algorithms is performed to consider which ones that are suited for tank inspection. The linearized dynamic quadcopter mathematical model is derived and simulated with low level PID-control and high level collision avoidance in a self-developed MATLAB simulator. The methodology of the algorithms implemented is explained and divided into two groups; those considering the environment to be known (global), and the ones continuously sensing the environment from the on-board radar sensors (local). The Null-Space Based (NSB) behavioral control algorithm is tested as a global approach. A self-developed reactive logic algorithm and the Velocity Obstacle (VO) algorithm are tested as local approaches for collision avoidance. The results yielded from the different algorithms' simulations show successful inspection with room for improvement on the implemented collision avoidance. The remarks of the simulation results are discussed and it is suggested to extend the self-developed reactive logic algorithm for future work.

Sammendrag

I løpet av de siste årene har målet om å utføre inspeksjoner ved bruk av ubemannede luftfartøy (UAV) blitt en realitet. Denne oppgaven tar for seg ulike algoritmer for implementasjon av høynivå kontrollere for antikollisjonssystemer på en multirotor-inspeksjonsdrone som opererer på innsiden av en skipstank. Den ønskede plattformen for dronen i denne oppgaven er en quadcopter-drone med seks radarsensorer montert på fartøyet for å registrere miljøet rundt seg. En litteraturstudie på eksisterende kollisjonsunngåelsesalgoritmer er utført for å vurdere hvilke algoritmer som egner seg for tankinspeksjon. Den lineariserte dynamiske modellen til quadcopteret er utledet, og det er utført simuleringer med lavnivås PID-regulator og høynivås antikollisjon på denne modellen i en egenkonstruert MATLAB-simulator. De implementerte algoritmenes metoder er forklart og delt i to kategorier; de som kjenner omgivelsene (globale), og de som kontinuerlig registrerer miljøet rundt seg ved hjelp av radar sensorene (lokale). Nullromsbasert-Oppførselskontrollalgoritmen (NSB) er testet som en global fremgangsmåte. En selvutviklet reaktiv logikksalgoritme og Fartshindringsalgoritmen (VO) er testet som lokale fremgangsmåter for kollisonsunngåelse. Resultatene fra de ulike algoritmenes simuleringer viser at inspeksjonen blir utført korrekt, men at forbedringer bør gjøres på anti-kollisjonen. Resultatene er diskutert og det blir foreslått å lage en utbedret versjon av den selvutviklede reaktive logikksalgoritmen.

Preface

This thesis is submitted in partial fulfillment of the requirements for the Master of Science degree at the Department of Engineering Cybernetics at the Norwegian University of Science and Technology (NTNU).

I want to thank my family and friends, who have supported me throughout these five years of university. I want to thank my supervisor, Professor Tor Arne Johansen for overall help and guidance for this thesis. A special thanks goes to my co-supervisor Dr. Kristian Klausen for detailed insight, guidance and feedback.

The equipment used for this thesis is the MATLAB software running on a Windows 10 computer provided by the institute. The MATLAB simulator used for algorithmic testing is based on the one used in the *MEAM620* robotics-class held by the University of Pennsylvania [1]. The dynamic quadcopter model and trajectory planner is provided in the original simulator, whereas the collision avoidance algorithmic implementations have been coded by the author.

Vetle Andre Bjelland
Trondheim, June 2018

Table of Contents

List of Tables

List of Algorithms

List of Figures

Abbreviations

Symbol	Definition
2D	Two Dimensions
3D	Three Dimensions
CEPA	Cushion Extended-Periphery Avoidance
CLIK	Closed Loop Inverse Kinematics
DWA	Dynamic Window Approach
GES	Global Exponential Stability
GPS	Global Positioning System
IMU	Inertial Measurement Unit
LIDAR	Light Imaging Detection And Ranging
RADAR	Radio Detection And Ranging
NED	North East Down
NEu	North East Up
NSB	Null Space Based Behaviour
PID	Proportional-Integral-Derivative
RRT	Rapidly-Exploring of Random Trees
SAR	Synthetic-Aperture Radar
SLAM	Simultaneous Localization And Mapping
UAV	Unmanned Aerial Vehicle
VFF	Virtual Force Field
VO	Velocity Obstacle

Chapter 1

Introduction

This chapter introduce the topic and structure of the thesis.

1.1 Background and Motivation

The ever increasing advancements in technology have made our society move towards an autonomous state where robots are able to do the dull, dirty and dangerous jobs humans have done before. Clever ways of using remotely controlled or fully autonomous vehicles have been a hot topic in the recent years. From pioneering in the military industry, different private sectors have come up with new ways of using autonomous drones for efficiency and risk reduction in their operations. Most people today have either seen or played around with the increasingly popular and most common multirotor platform, the *quadcopter*. Today, the quadcopter has become more than just a toy. Countless opportunities for the multirotor drone has been discovered for both military and private sectors. In order to optimize usage in a certain application, a dedicated system designed for the drone's purpose has to be constructed. One of the main challenges on UAVs, in general, is how to control the vehicle through an environment containing unknown obstacles in the vehicle's desired path. It is a critical task to maintain proximity of the planned path and stability in its attitude in order to avoid collisions that can hinder the objective and/or cause material or personal damage [2].
When a UAV is deployed to inspect the inside of a tank with unknown content, the only feedback information provided is usually a live video feed from a camera mounted to the vehicle. Even for skilled pilots, avoiding collisions in such environments is a de-

manding and difficult task. Thus, the need for robust automatic collision avoidance is critical and necessary to allow pilots to focus on higher-priority tasks such as inspecting the tank area for dangers and damage. This thesis will focus on the design of such a collision avoidance system, particularly in unknown environments where sensor data is provided by radar sensors.

1.1.1 Scenario

In order to further understand the challenges of multirotor inspection, a scenario for autonomous inspection is proposed. The main aspects to be considered are:

- UAV platform and dynamics
- Pilot input controls
- Obstacle avoidance and anti-collision for walls and valves
- Sensor data collection and processing
- Fail-safe methods and redundancy
- Obstacle avoidance and anti-collision
- Effects of the second order acceleration dynamics

The inside of a cubic shipping-tank is to be inspected autonomously by a multirotor UAV. The operators bring the inspection drone inside of the tank and execute the drone's starting procedures. The inspection mission is done by taking off inside the tank, fly to the top and follow the walls with a fixed distance. If abnormalities occur, the drone should execute counter maneuvers to avoid collisions but still continue its inspection mission. The tank to be inspected in this scenario is illustrated in figure 1.1. The multirotor inspection drone has six radar sensors mounted around its body chassis, resulting in a 360° sensor coverage for obstacle detection. Each sensor detects the distance to objects by the reflecting radio wave pulses. These measurements from the different sensors can be combined in order to understand the environment around the drone. Based on the active sensors and the relative distances between the drone and the obstacles, a modification of the drone's behaviour is done in the situation of a potential collision. Furthermore, by evaluating the provided sector possibilities, we can optimize the inspection path by steering the drone towards the optimal trajectory.

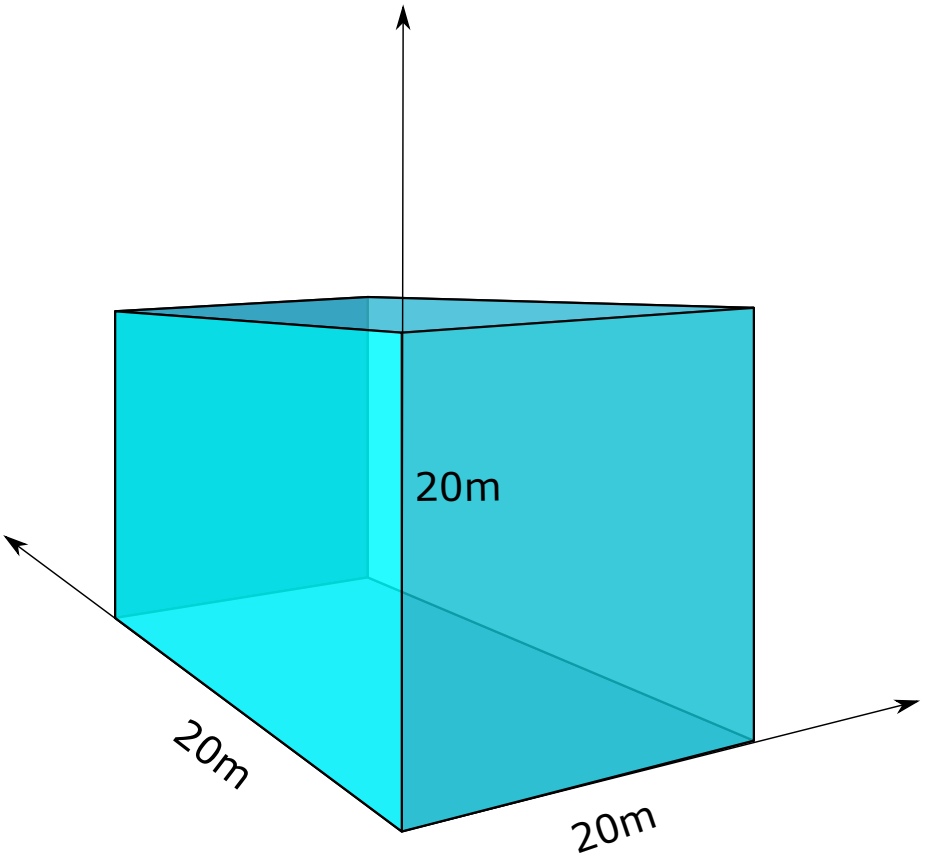


Figure 1.1: Transparent shipping tank

1.1.2 Mission Idea

Now that we have a given scenario - how can the inspection mission be realized?

- Which existing algorithms are suited for the scenario?
- What reactive logic needs to be designed?
- What are the challenges the drone can face during the inspection mission?
- How to receive and process the sensor data in an easy and efficient way?

The drone is to provide live video feed from the executed inspection path. The video can later be reviewed by operators, such that they are not exposed to hazards. By following a fixed distance from the container walls, the drone can get an overview of the entire tank. This thesis will mainly focus on the implementation of collision avoidance approaches, and this is demonstrated by simulated step- and square trajectories inside the tank.

1.1.3 Challenges

The drone needs to keep a safe distance during the collision avoidance maneuvers, but also keep proximity to its planned inspection path. This can be challenging in narrow places where valves occur, where the gap between the valve and a wall is limited. The challenge of avoiding collisions is a top priority, as the risk of harming operators or material damage needs to be avoided. The operator commands hence have to be overridden if a collision is implied.

1.1.4 Proposed Solution

For the problem described of multirotor inspection inside a shipping tank, a proposed solution is to simulate the drone navigation in a simple environment. An empty shipping tank of given dimensions is to be inspected, with the obstacles being the container's walls and valves located at unknown locations. The dimensions of the shipping container in meters are given in table 1.1:

	Container Dimensions
Length	20m
Width	20m
Height	20m

Table 1.1: Table of shipping tank dimensions

After bringing the drone into the desired inspection area and finishing its starting procedure, it takes off inside the shipping tank with an operator monitoring it. To ensure the drone has a safe distance from the container walls, a fixed distance from the wall is set. This safe distance needs to be kept at all times, and in the case of an obstacle, the UAV might need to take a longer path around the obstacle in order to not violate this constraint. By traversing alongside the walls, the inspection is finished when the drone reaches its start position again. The operator can now safely pick up the drone

again and review the collected inspection video. The inspection is to be simulated in a self-developed MATLAB simulator with the control structure designed by the author.

1.2 Previous Work

There exists a scope of papers and previous theses concerning collision avoidance. Some are targeted for robots, others for UAVs, but most methods can be designed for either one. Here is a selection of the most relevant, covering the algorithms that have been investigated further in this thesis.

In [2], a collision avoidance algorithm with applications for remotely-piloted UAVs is implemented. This is done to release the operators from any unnecessary workload and to allow them to focus on more important tasks during operations. The local, model-based stochastic algorithm uses the pilot's input and on-board sensors to predict if a collision will occur based on the UAV's dynamics. If a collision is imminent, the pilot's input is modified to avoid collisions.

In [3], behavioral control for multi-agent use is investigated and the use of NSB as a collision avoidance algorithm is implemented. The NSB algorithm is also used in [4] for coordination of multi-robot systems.

In [5], a method for robot motion planning in dynamic environment by utilizing the velocity space is investigated. The method used is the Velocity Obstacle algorithm, which is also used for aerial vehicles in [6].

1.3 Contribution and Scope of This Thesis

The goal of this thesis is to investigate the potential of different methods for ensuring collision-free inspection using a multirotor drone. This is mainly done by focusing on the algorithms available, their functionality and implementation. Testing of the different methods' response and robustness on the multirotor platform is done in a self-developed MATLAB simulator. The yielded results are discussed to illustrate the potential of the collision avoidance algorithms implemented for autonomous multirotor inspection. A conclusion of the findings and suggested future work on the topic of collision avoidance for multirotor inspection is then provided.

1.4 Organization of This Thesis

- A literature review of different collision avoidance algorithms is conducted in Chapter 2.
- Chapter 3 covers the model setup for the simulator used to test the most promising algorithms from chapter 2. A stability analysis of a simplified mass-spring-damper system is done, the dynamic model of the multirotor platform and how the dynamics are implemented to the MATLAB simulator is then presented.
- Chapter 4 covers a brief theoretical background on sensors as well as the methodology for the algorithms most suited from chapter 2.
- The yielded results from the different algorithms explained in chapter 4 are presented in chapter 5.
- Chapter 6 discusses the algorithms' suitability for the inspection application based on the results obtained in chapter 5.
- A conclusion of the results and suggested future work on the topic is presented in Chapter 7.

1.5 Note on Project Thesis

This master thesis is a continuation of the author's project thesis [7]. Some of the project thesis parts are reused in this thesis. This mainly applies to the literature review, some of the mathematical models as well as the NSB mythology, given in section 4.4.

1.6 Nomenclature and Notation

- All bold letters are column vectors $\boldsymbol{a} \in \mathbb{R}^{m \times 1}$
- All big bold letters are matrices $\boldsymbol{A} \in \mathbb{R}^{m \times n}$
- The transpose of a vector $\boldsymbol{a} \in \mathbb{R}^{m \times 1}$ is denoted $\boldsymbol{a}^T \in \mathbb{R}^{1 \times m}$
- The norm of a vector is denoted as $||\cdot||$ where $\cdot$ is some vector
- The frame of a certain parameter a is noted in the parameter's upper right, $a^{\mathcal{A}}$
- The frame of a certain vector $\boldsymbol{a}$ is noted in the vector's upper left, ${}^{\mathcal{A}}\boldsymbol{a}$

- A transformation vector ${}^{\mathcal{A}}\boldsymbol{t}_{\mathcal{B}}$ transforms from one frame to another, where $\mathcal{A}$ is the start frame and $\mathcal{B}$ is the end frame
- A rotation matrix ${}^{\mathcal{A}}\boldsymbol{R}_{\mathcal{B}}$ transforms from one frame to another, where $\mathcal{A}$ is the start frame and $\mathcal{B}$ is the end frame
- When rotations are about a certain point p, the rotation is denoted as ${}^{\mathcal{A}}\boldsymbol{R}_{p}^{\mathcal{B}}$, where $\boldsymbol{R}$ is the desired rotation, $\mathcal{A}$ is the start frame and $\mathcal{B}$ is the end frame
- The identity matrix is denoted $\boldsymbol{I}_{m\times m}$, where m is the proper dimension
- A function $\boldsymbol{f}(\boldsymbol{x})$ can be continuously differentiable of k-th degree. This is noted as $\mathbb{C}^k$, if $k = 0$, $\boldsymbol{f}(\boldsymbol{x})$ is said to be continuous
- The gradient ∇ of a 3D system $f(\boldsymbol{x})$ is given as:

$$\nabla f(\boldsymbol{x}) = \frac{\partial f(\boldsymbol{x})}{\partial x_1}\boldsymbol{i} + \frac{\partial f(\boldsymbol{x})}{\partial x_2}\boldsymbol{j} + \frac{\partial f(\boldsymbol{x})}{\partial x_3}\boldsymbol{k}$$

- The Jacobian $\boldsymbol{J}$ of a system $\boldsymbol{f}(\boldsymbol{x})$ is given as:

$$\boldsymbol{J}(\boldsymbol{f}(\boldsymbol{x})) = \begin{bmatrix} \dfrac{\partial f_1(\boldsymbol{x})}{\partial x_1} & \dfrac{\partial f_1(\boldsymbol{x})}{\partial x_2} & \cdots & \dfrac{\partial f_1(\boldsymbol{x})}{\partial x_n} \\ \vdots & & & \\ \dfrac{\partial f_2(\boldsymbol{x})}{\partial x_1} & \dfrac{\partial f_2(\boldsymbol{x})}{\partial x_2} & \cdots & \dfrac{\partial f_2(\boldsymbol{x})}{\partial x_n} \\ \dfrac{\partial f_m(\boldsymbol{x})}{\partial x_1} & \dfrac{\partial f_m(\boldsymbol{x})}{\partial x_2} & \cdots & \dfrac{\partial f_m(\boldsymbol{x})}{\partial x_n} \end{bmatrix}$$

Chapter 2

Literature Review

In order to get an overview of which collision avoidance approaches that exist and which ones that are best suited for autonomous inspection, a literature review is executed. Using scientific papers and previous master theses, different collision avoidance approaches are read up on and analyzed. This section quickly sums up the findings and which properties the different techniques obtain.

2.1 Trajectory Generation

In order to understand collision avoidance, we have to explain and distinguish the terms *trajectory generation* and *collision-free path*. Most inspection missions have an objective of moving from point A to point B while avoiding to crash into obstacles occurring on that path. Such an objective can be achieved by planning a collision-free path. A collision-free path is a path from the beginning till the end where the drone avoids obstacles in the planned path such that no collisions occur. This can be done either by planning the path on beforehand knowing the environment the drone is to fly in (trajectory generation) or by ensuring that the drone has an anti-collision system that changes the behaviour of the drone when obstacles are detected (collision avoidance). A mixture of both planning and sensing is the most robust way of ensuring collision avoidance. A collision-free path can be realized by path planning in global and local maps. When static obstacles are known, this problem has many solutions.

2.1.1 Global Planners

Global planners ensure convergence to the goal position, but assume a known environment. Not applicable for dynamic or unknown environments. A global approach generates a path from the beginning till the end by using the available information on obstacles in the area of operation. The downside of this is that it often process much information, which has an impact on the time it takes to generate the path. The processing time may vary from less than a second to several minutes, and a global path planning method might not be suited for a drone flying in a dynamic environment. A local correction to the path can be obtained by dynamic obstacle avoidance algorithms. These algorithms have a faster processing time that can handle the local dynamics the global map doesn't represent.

2.1.2 Local Planners

Local planners reduce computational cost by examining a shorter time window than the global ones. They can address obstacles that are unknown using sensor information. The main drawback is the lack of overall safety or convergence guarantee since the optimization only occurs for short time windows and the closest obstacles. As a consequence of this, the drone can be stuck in a local minimum. Reactive controllers, which are a type of local planners, generate trajectories directly based on the information of the environment provided by the sensors. The main drawback of the reactive controllers is that the vehicle thrust constraints may be violated, and higher derivatives may not be bounded, which can violate the drone's low level controller requirements.

2.2 Control Architectures

For us humans, it is in our nature to deal with a constantly changing reality and how to comprehend the different changes. We gather the information we get from our senses, in order to make a decision on which way to go, when to drink water and so on. Just as for humans, we need to implement a drone that can gather information from its surroundings in order to execute its purpose in its current environment. A drone can be programmed in infinitely many ways, but we commonly use three architectures of control approaches, namely the *Deliberative-*, the *Reactive-* and the *Hybrid* architecture [8].

2.2.1 The Deliberative Architecture

This architecture can be described as a ”think, then act”-control. The drone uses all the information provided by its different sensors and its initial knowledge in order to decide what action to take next. The organization of the control structure is often a functional decomposition of the decision making progress. This decomposition consists of a sensory processing module, a modeling module, a planning module, a value judgment module and an execution module [9]. The decomposition allows complex operations, but also strong sequential dependencies between the decision-making modules. Planning is the typical reasoning done in the deliberative systems and is known for being a computationally complex process. The sequential process of *sense-plan-act* has to be done for each iteration, and a map representation of the drone's environment is needed in order to allow the drone to look ahead into the future and predict the values for the various states. Hence, for a drone with computationally efficient hardware and up-to-date accurate maps, the architecture allows the drone to choose an optimal course of action for a given situation. However, a dynamic world is considered noisy and makes this approach close to impossible, and there are no droneic implementations that are purely deliberative.

2.2.2 The Reactive Architecture

By combining the impressions gathered from the sensory information only, the reactive approach can be considered as a ”don't think, react”-control. Inspired by a rule-based method involving minimum computation and no internal information about the global map, it is suited for real-time application drones such as the quadcopter. By embedding the drone's controller in a collection of reprogrammed, concurrent condition-action rules, it is well suited for dynamic worlds, and the fast computing time makes it ideal for rapidly changing environments. Limitations do however apply, such as the lack of storage of information and knowledge about the world. This makes it incapable to learn and improve over time [8]. Compared to the deliberative architecture, the reactive one trades of the complexity of reasoning for fast reaction time.

2.2.3 The Hybrid Architecture

As the name suggests, the hybrid architecture is a mixture of the best aspects from the deliberative- and the reactive architectures: the real-time response from reactivity and the rationality and optimality of the deliberative. Hence, the hybrid controller consists of two components. These two must interact in order to produce a coherent output. This interaction is challenging as the reactive component deals with the drone's im-

mediate needs, such as avoiding obstacles, meaning that it uses direct external sensor data on a short timescale. Whereas the deliberative component uses internal world data representation and operates on these data on a longer timescale, such as global path-planning. For the situation where the two components' output doesn't conflict, no further coordination is required. However, the two components need to interact in order to benefit from each other. Hence, the reactive system needs to override the deliberative one if some unexpected challenge is detected. At the same time, the deliberative system must inform the reactive one if more optimal paths towards the goal are detected. The interaction between the systems is typically the greatest challenge of hybrid systems. An intermediate component needs to be constructed in order to combine the two components, which reconciles the different representations and any conflicts between their outputs. The hybrid architecture is commonly referred to as the *Three-Layer Architecture* because of the layered structure consisting of the reactive- deliberative- and interaction part [8].

2.3 Collision Avoidance Approaches

2.3.1 A* Algorithm

An algorithm that uses a heuristic approach to find the cheapest path from the beginning till the end in a graph is the A* algorithm [10]. It is easy to implement and often used in path planning. It always finds a solution if one exists and is considered very efficient. An evaluation function consisting of the cost for traversing neighboring nodes in the graph, and a heuristic estimate for the cheapest path form the current node to the end is considered. The idea is to minimize this evaluation function in order to find the collision-free path.

2.3.2 Artificial Potential Fields

The potential fields can be created for a highly efficient path planning with obstacle avoidance but require that we know the environment on beforehand. For an inspection drone, the environment may have uncertainties and the algorithm is not optimal for inspection applications. The algorithm is however intuitive and easy to implement and can be used as a benchmark for an optimal path. The algorithm, introduced in [11] has the intuitive idea of obstacles creating artificial fields of forces that are set to act repulsively on the drone, whereas the goal sets up a similar field with forces acting attractively on the controllable drone. The combined forces, or the combined potential field, determines which direction the drone is pushed towards and thereby the drone's

velocity. Knowledge about the obstacle(s) in the environment is necessary in order to navigate. A slightly different version of the algorithm is the Virtual Force Field (VFF) algorithm from [12]. Since APF is a purely global approach, this thesis does not focus on the potential fields. A detailed description of the APF is done in the author's project thesis [7].

2.3.3 Curvature - Velocity Method

As a method for local obstacle avoidance, the Curvature-Velocity Method uses a set of constraints for navigation. By using the velocity space, it runs an optimization considering the physical constraints from the transnational and rotational velocities from the drone, as well as environmental constraints such as obstacles, provided by sensor data. The velocity that satisfies these constraints is then maximized in the objective function, which considers the elements of safety, speed, and goal-directness [13]. After the optimization problem is solved, the yielded velocity is commanded to the drone. For real-time performance, a piecewise constant function is used to approximate the curvature distance to obstacles. The velocity space is divided into a discrete number of regions from the approximation, each of which has a constant distance to impact. The method then finds the point in each region that maximizes the objective function and chooses the overall maximum point to command the drone.

2.3.4 Cushion Extended - Periphery Avoidance

CEPA is a reactive obstacle avoidance plugin described in [14]. A robust navigation solution requires quick response to obstacles in dynamic, tight environments with non-negligible disturbances - this is something CEPA provides. It addresses two main issues related to safe operations:

1. Guide the drone around obstacles towards destinations chosen by the high-level planner.

2. Apply additional control in emergency situations if the drone comes too close to an obstacle.

CEPA analytically inflates the proposed path in polar coordinates. As a result, the path can be verified for obstacles by a simple difference in the polar domain, which reduce computational load and algorithm latency.

The plugin consists of a steering algorithm and an emergency avoidance. The drone projects two cushions in its moving direction, where the outer cushion is seen as a

comfort zone that can be intruded, whereas the inner activates the emergency avoidance when intruded. A detailed explanation of the algorithm is done in the author's project thesis [7]. Even though CEPA is a good algorithm for collision avoidance for UAVs, the method is not well suited for radar applications.

2.3.5 D* Algorithm

Having very much the same behavior as the A* algorithm, the D* algorithm differentiate in the sense that the arc cost parameters can change during the problem-solving process [15]. The algorithm keeps the calculated information of the path in order to be able to re-plan the path when needed. Hence, it is a more dynamic version of the A* algorithm, as it is able to re-calculate a new path much faster.

2.3.6 Dynamic Window Approach

Using the dynamic model of the intended drone, the dynamic window approach (DWA) stands out from other methods as a model-based approach. The method reduces its velocity space by removing those that are unreachable within a short amount of time and the ones causing collisions. Hence, the reduced velocity space only consists of velocities for safe navigation where the drone can stop safely. An objective function is used to achieve this reduction and is formed by combining the drone's transnational speed and rotation rate, the progress towards the goal and the relative distance between obstacles and the drone. The result of maximizing the objective function is balanced velocity outputs considering both speed and safe avoidance maneuvers. Most drones have physical constraints for velocity and acceleration, and DWA is well suited for such constraints as it handles it naturally. Since the method is model-based though, a fully dynamic model of the drone is needed for usage [16].
DWA only considers admissible velocities. For a velocity pair to be considered admissible, the drone has to be able to stop before colliding with the closest obstacle along the projected trajectory defined by the velocity pair.

2.3.7 Null-Space Behavioral Control

The algorithm, described in [17] is a behavioral algorithm that focuses on task scheduling. Compared to other behavioral methods, the NSB algorithm provides a unified framework with the two main behavioral approaches:

- Competitive approach

- Cooperative approach

It utilizes the null-space of the task functions to provide a framework for task priority. NSB always fulfills the highest priority task. The lower priority ones are fulfilled only in a subspace where they do not conflict with the ones having higher priorities. This is much better than competitive and cooperative and will be further explained in section 4.4.

2.3.8 Rapidly Exploring Random Trees

In [18], Rapidly-Exploring Random Trees (RRT) is presented as a randomized path planning technique. It is a simple algorithm, that can take the dynamics of the drone into account. Note that since it is a randomized method, it is not a complete method. It works by generating a tree structure which grows from the initial node. After each iteration, a connection is attempted to be drawn between the tree and the drawn sample if the connection is feasible with regards to constraints and obstacles. The tree structure will this way explore a widened area of the configuration space for each iteration. In [18] it is stated that the RRT is biased towards places not yet visited. By increasing the probability of sampling states of specific areas, one can also bias the RRT growing towards these specific areas. This could be used to make the RRT focus the path more towards the goal.

2.3.9 Velocity Obstacle

The Velocity Obstacle (VO) is a concept used for collision avoidance on a local scale. A collision cone is constructed which consists of the collection of velocities that will cause the drone to collide [19]. The objects are transformed into static obstacles with a relative velocity to the drone. By mapping the obstacle into the drone's configuration space, the velocity obstacle is built by using feasible sets of velocities. In order to avoid collisions, one can now see that the end of the drone's velocity vector must be outside of the velocity obstacle. A set of avoidance maneuvers is done by vector operations to avoid collision with the obstacle and returns a trajectory correction. The dynamics of the drone is not directly considered as the dynamic constraints are set to constant on the velocity and turning angle, but still taken somewhat into account. The method is described further in section 4.6.

2.3.10 Virtual Force Field

By using sensor readings, the virtual force field algorithm generates a map. The map is represented as a 2D Cartesian histogram grid and contains probabilities of obstacles

being located in different places of the area. The probability values' magnitudes act as the repulsive forces affecting the drone. One limitation of this algorithm mentioned in [12] is that passage between close lying obstacles might not be granted. Oscillations may also develop as the drone flies through narrow corridors if the travel direction deviates from the corridor center. The algorithm can be described in two steps:

1. Calculate the position of the moving object from the distance and angle received from the sensors. This is useful to calculate the gradient of the potential function.

2. Check whether the distance received from the sensor is less than the avoidance distance. If it is less, calculate the gradient of the potential function. This acts as the control input for the drone.

Chapter 3

Model Setup for the Simulator

The desired platform for an inspection drone, its environment, and the drone's behaviour are to be simulated. It should consist of six radar sensors to ensure 360° coverage for obstacle detection and avoidance. A suitable platform for the application area is a *quadcopter*. A quadcopter consists, as the name suggests of a symmetric cross-shaped airframe carrying the payload in the cross' center. Four thrust giving propellers in each end ensure lift, as illustrated in figure 3.1. Because of its maneuverability, safe- and low-cost experimentation in mapping, navigation, and control strategies in three dimensions, the platform is the most popular in the research of aerial robotics [8].
The mission to be executed is to inspect a shipping tank for rust/corrosion or other damages to structures autonomously. The quadcopter offers a physical platform able of high precision maneuvers in cramped environments. This enables responsive commands where time response is an important factor. Because of its recent popularity, there are countless options for software and hardware packages for quadcopter flight operations. This chapter covers the stability analysis of a simplified mass-spring-damper system, the full dynamics of a quadcopter and how the dynamics are implemented to the MATLAB simulator.

3.1 Second Order Mass-Spring Damper

As a simplified version of a quadcopter, we can illustrate the quadcopter as a point mass with the classical mass-spring damper example. This allows an easier mathematical model for stability and behavioral analysis of a simplified system.

$$m\ddot{\boldsymbol{x}} + c\dot{\boldsymbol{x}} + k\boldsymbol{x} = \boldsymbol{u} \tag{3.1}$$

Where $\boldsymbol{x}$ is the position of the point mass, $\boldsymbol{u}$ is the force affecting the point mass, m is the mass constant, c is the damping constant and k is the spring constant. This second order system can be derived into two first order systems by introducing new state variables:

$$\boldsymbol{x}_1 = \boldsymbol{x} \tag{3.2}$$

$$\boldsymbol{x}_2 = \dot{\boldsymbol{x}} \tag{3.3}$$

Hence, we will have:

$$\dot{\boldsymbol{x}_1} = \boldsymbol{x}_2 \tag{3.4}$$

$$\dot{\boldsymbol{x}_2} = -\frac{k}{m}\boldsymbol{x}_1 - \frac{c}{m}\boldsymbol{x}_2 + \frac{\boldsymbol{u}}{m} \tag{3.5}$$

Which can be represented on a matrix form:

$$\dot{\boldsymbol{x}} = \boldsymbol{A}\boldsymbol{x} + \boldsymbol{b}\boldsymbol{u} \tag{3.6}$$

$$\dot{\boldsymbol{x}} = \begin{bmatrix} 0 & 1 \\ -\frac{k}{m} & -\frac{c}{m} \end{bmatrix} \begin{bmatrix} \boldsymbol{x}_1 \\ \boldsymbol{x}_2 \end{bmatrix} + \begin{bmatrix} 0 \\ \frac{1}{m} \end{bmatrix} \boldsymbol{u} \tag{3.7}$$

3.1.1 Eigenvalues

Eigenvalues allow to assess the dynamic behavior of the system in particular the stability of its equilibrium without the need to solve the state differential equation [20]
The eigenvalues of $\boldsymbol{A}$ are the n roots $\lambda_i, i = 1, ..., n$ of the characteristic polynomial:

$$\det(\lambda_i \boldsymbol{I} - \boldsymbol{A}) = 0 \iff \lambda_i \text{ is an eigenvalue of } \boldsymbol{A}$$

$\boldsymbol{A}$ does not necessarily have different unique eigenvalues. If an eigenvalue λ_i occurs k_i times, we say it has an algebraic multiplicity of k_i. It is important that the eigenvalues for our closed loop system have negative real values, as we want our system to be stable.

3.1.2 Error Dynamics

The mass-spring damper system described in (3.7) can be written out as

$$\dot{\boldsymbol{x}}_1 = \boldsymbol{x}_2 \tag{3.8}$$

$$\dot{\boldsymbol{x}}_2 = \frac{1}{m}\big(- k\boldsymbol{x}_1 - c\boldsymbol{x}_2 + u\big) \tag{3.9}$$

By introducing the error variables $\tilde{\boldsymbol{x}}_1 = \boldsymbol{x}_1 - \boldsymbol{x}_{des}$ and $\tilde{\boldsymbol{x}}_2 = \boldsymbol{x}_2 - \dot{\boldsymbol{x}}_{des}$, where $\boldsymbol{x}_{des}$ is the desired state of the point mass, the error dynamics can be described as:

$$\begin{aligned}\dot{\tilde{\boldsymbol{x}}}_1 &= \dot{\boldsymbol{x}}_1 - \dot{\boldsymbol{x}}_{des} = \boldsymbol{x}_2 - \dot{\boldsymbol{x}}_{des} = \tilde{\boldsymbol{x}}_2 \\ \dot{\tilde{\boldsymbol{x}}}_2 &= \dot{\boldsymbol{x}}_2 - \ddot{\boldsymbol{x}}_{des} = \frac{1}{m}\big(- k\boldsymbol{x}_1 - c\boldsymbol{x}_2 + u\big) - \ddot{\boldsymbol{x}}_{des}\end{aligned} \tag{3.10}$$

Now, by using $\boldsymbol{x}_1 = \tilde{\boldsymbol{x}}_1 + \boldsymbol{x}_{des}$ and $\boldsymbol{x}_2 = \tilde{\boldsymbol{x}}_2 + \dot{\boldsymbol{x}}_{des}$, we can rewrite (3.10) as:

$$\begin{aligned}\dot{\tilde{\boldsymbol{x}}}_1 &= \tilde{\boldsymbol{x}}_2 \\ \dot{\tilde{\boldsymbol{x}}}_2 &= \frac{1}{m}\big(- k(\tilde{\boldsymbol{x}}_1 + \boldsymbol{x}_{des}) - c(\tilde{\boldsymbol{x}}_2 + \dot{\boldsymbol{x}}_{des}) + \boldsymbol{u}\big) - \ddot{\boldsymbol{x}}_{des} \\ &= \frac{1}{m}\big((-k\tilde{\boldsymbol{x}}_1 - c\tilde{\boldsymbol{x}}_2 + \boldsymbol{u}) - \ddot{\boldsymbol{x}}_{des} - c\dot{\boldsymbol{x}}_{des} - k\boldsymbol{x}_{des}\big) \\ &= \frac{1}{m}\big((-k\tilde{\boldsymbol{x}}_1 - c\tilde{\boldsymbol{x}}_2 + \boldsymbol{u}) + \boldsymbol{d}(t)\big) \\ &= f(\tilde{\boldsymbol{x}}) + g(\tilde{\boldsymbol{x}})\boldsymbol{u}\end{aligned} \tag{3.11}$$

Here $f(\tilde{\boldsymbol{x}}) = \frac{1}{m}\big((-k\tilde{\boldsymbol{x}}_1 - c\tilde{\boldsymbol{x}}_2) + \boldsymbol{d}(t)\big)$, where $\boldsymbol{d}(t) = -\frac{1}{m}\big(\ddot{\boldsymbol{x}}_{des} + c\dot{\boldsymbol{x}}_{des} + k\boldsymbol{x}_{des}\big)$. Where $g(\tilde{\boldsymbol{x}}) = \frac{1}{m}$ is constant and known since the mass m is known. We can now do a stability analysis of the mass-spring-damper system, which is done in the next section.

3.1.3 Lyapunov Stability Analysis

A stability analysis of the simplified mass-spring-damper system is done by Lyapunov Stability analysis. The unforced error dynamics ($\boldsymbol{u} = 0$) can be written as:

$$\dot{\tilde{\boldsymbol{x}}} = \begin{bmatrix} 0 & 1 \\ -\frac{k}{m} & -\frac{c}{m} \end{bmatrix} \begin{bmatrix} \tilde{\boldsymbol{x}}_1 \\ \tilde{\boldsymbol{x}}_2 \end{bmatrix} + \begin{bmatrix} 0 \\ \boldsymbol{d}(t) \end{bmatrix} \tag{3.12}$$

The corresponding nominal system of (3.12) is:

$$\dot{\tilde{\boldsymbol{x}}} = \boldsymbol{A}\tilde{\boldsymbol{x}} = \begin{bmatrix} 0 & 1 \\ -\frac{k}{m} & -\frac{c}{m} \end{bmatrix} \begin{bmatrix} \tilde{\boldsymbol{x}}_1 \\ \tilde{\boldsymbol{x}}_2 \end{bmatrix} \tag{3.13}$$

The corresponding eigenvalues of $\boldsymbol{A}$ for the given values of $m = 2.5$, $c = 6$ and $k = 3$ yields $\lambda_{1,2} = -1.2 \pm \sqrt{0.96}$. Both the eigenvalues fulfill the criteria of $Re\{\lambda_i\} < 0$ and $\boldsymbol{A}$ is Hurwitz by theorem A.1.1.
The ratio $\frac{\lambda_{min}(\boldsymbol{Q})}{\lambda_{max}(\boldsymbol{P})}$ is to be maximized by setting $\boldsymbol{Q} = \boldsymbol{I}_{2\times 2}$. The Lyapunov function candidate $\boldsymbol{V} = \tilde{\boldsymbol{x}}^T \boldsymbol{P} \tilde{\boldsymbol{x}}$ satisfies:

$$\begin{aligned} \lambda_{min}(\boldsymbol{P})||\tilde{\boldsymbol{x}}||_2^2 \leq V \leq \lambda_{max}(\boldsymbol{Q})||\tilde{\boldsymbol{x}}||_2^2 \\ \frac{\partial \boldsymbol{V}}{\partial \tilde{\boldsymbol{x}}} \boldsymbol{A}\tilde{\boldsymbol{x}} = -\tilde{\boldsymbol{x}}^T \boldsymbol{Q} \tilde{\boldsymbol{x}} \leq \lambda_{min}(\boldsymbol{Q})||\tilde{\boldsymbol{x}}||_2^2 \\ \left|\left| \frac{\partial \boldsymbol{V}}{\partial \tilde{\boldsymbol{x}}} \right|\right| \leq 2\lambda_{max}(\boldsymbol{P})||\tilde{\boldsymbol{x}}||_2^2 \end{aligned} \tag{3.14}$$

Where $\boldsymbol{P}$ is found from the equation $\boldsymbol{PA} + \boldsymbol{A}^T\boldsymbol{P} = -\boldsymbol{Q}$ from theorem A.1.1. Based on the satisfaction of (3.14), the nominal system has a Globally Exponentially Stable (GES) origin since theorem A.1.2 is satisfied [21].

3.2 Frames and Relations

In order to model the quadrotor, a coordinate system and a free body diagram for the quadcopter has to be set up. Geographic reference frames are used for the consistency of reference. The reference frames used for our simulator and their relation is covered in this section.

3.2.1 NEU Frame $\mathcal{A}$

The North-East-Up (NEU) frame is denoted $\mathcal{A}$ and is a slightly modified version of the well known North-East-Down (NED) frame. For flat earth navigation, it is assumed inertial, such that Newton's laws still apply. It is defined by the triad consisting of $\boldsymbol{a}_1$, $\boldsymbol{a}_2$ and $\boldsymbol{a}_3$, where $\boldsymbol{a}_3$ points upwards, as illustrated in figure 3.1. As for the NED frame, it is an earth-fixed coordinate system with a defined origin location relative to the Earth's reference ellipsoid [22]. The $\boldsymbol{a}_1$-vector in the north direction, the $\boldsymbol{a}_2$-vector

in the east direction, and the $\boldsymbol{a}_3$-vector in the opposite direction of the earth center direction. The states in this frame, denoted ${}^{\mathcal{A}}\boldsymbol{x} \in \mathbb{R}^6$ are:

$$ {}^{\mathcal{A}}\boldsymbol{x} = \boldsymbol{\eta} = \begin{bmatrix} x^{\mathcal{A}} & y^{\mathcal{A}} & z^{\mathcal{A}} & \phi^{\mathcal{A}} & \theta^{\mathcal{A}} & \psi^{\mathcal{A}} \end{bmatrix}^T = \begin{bmatrix} x^{\mathcal{A}} & y^{\mathcal{A}} & z^{\mathcal{A}} & {}^{\mathcal{A}}\boldsymbol{\Theta}^T \end{bmatrix}^T \tag{3.15} $$

Where $x^{\mathcal{A}}$, $y^{\mathcal{A}}$, $z^{\mathcal{A}}$ are the respective NEU frame positions of the drone, and ${}^{\mathcal{A}}\boldsymbol{\Theta} = \begin{bmatrix} \phi^{\mathcal{A}} & \theta^{\mathcal{A}} & \psi^{\mathcal{A}} \end{bmatrix}^T$ are the Euler-angles as defined in the zyx-rotation sequence.

3.2.2 Body frame $\mathcal{B}$

The body frame, $\mathcal{B}$, attached to the quadcopter's center of mass consists of the triad $\boldsymbol{b}_1$, $\boldsymbol{b}_2$ and $\boldsymbol{b}_3$ as illustrated in figure 3.1. $\boldsymbol{b}_1$ coincides with the preferred forward direction of the drone, $\boldsymbol{b}_3$ is perpendicular to the plane created by $\boldsymbol{b}_1$ and $\boldsymbol{b}_2$ with the rotors pointing up. The body states are denoted as ${}^{\mathcal{B}}\boldsymbol{x} \in \mathbb{R}^6$.

$$ {}^{\mathcal{B}}\boldsymbol{x} = \begin{bmatrix} x^{\mathcal{B}} & y^{\mathcal{B}} & z^{\mathcal{B}} & \phi^{\mathcal{B}} & \theta^{\mathcal{B}} & \psi^{\mathcal{B}} \end{bmatrix}^T = \begin{bmatrix} x^{\mathcal{B}} & y^{\mathcal{B}} & z^{\mathcal{B}} & {}^{\mathcal{B}}\boldsymbol{\Theta}^T \end{bmatrix}^T \tag{3.16} $$

And its time derivative as:

$$ {}^{\mathcal{B}}\dot{\boldsymbol{x}} = \boldsymbol{\nu} = \begin{bmatrix} u^{\mathcal{B}} & v^{\mathcal{B}} & w^{\mathcal{B}} & p^{\mathcal{B}} & q^{\mathcal{B}} & r^{\mathcal{B}} \end{bmatrix}^T = \begin{bmatrix} u^{\mathcal{B}} & v^{\mathcal{B}} & w^{\mathcal{B}} & {}^{\mathcal{B}}\dot{\boldsymbol{\Theta}}^T \end{bmatrix}^T \tag{3.17} $$

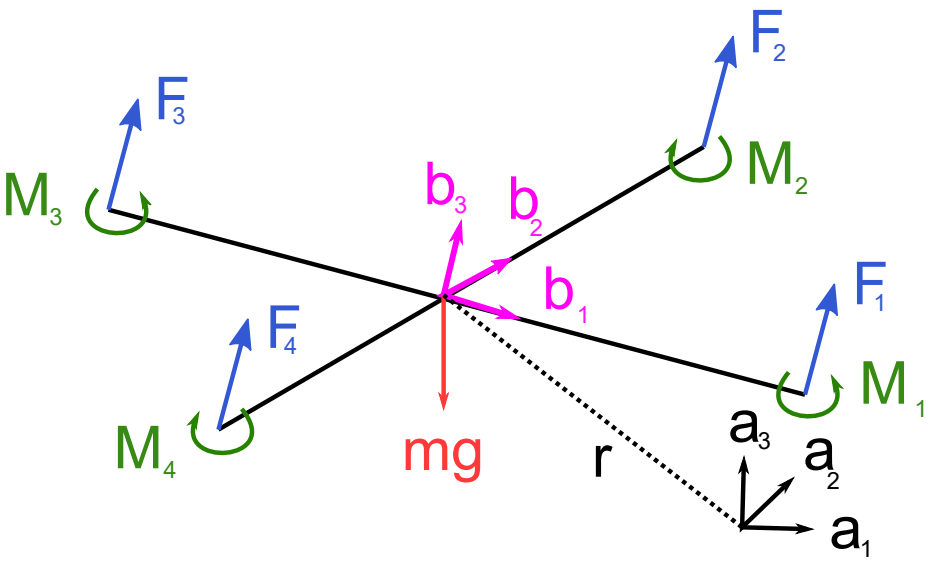


Figure 3.1: Illustration of the quadcopter orientation for both frames

3.2.3 Relation between NED and Body Reference Frames

The drone's yaw angle ψ plays a special role as we can choose it freely without affecting the drone's dynamics directly. For this reason, zxy-Euler angles are used to describe the transform from the body frame $\mathcal{B}$ to the inertial frame $\mathcal{A}$ [8]. A yaw rotation of ψ around the $\boldsymbol{a}_3$-axis is first preformed, followed by a roll rotation of ϕ around the rotated $\boldsymbol{a}_1$-axis, and lastly a pitch rotation of θ around the rotated $\boldsymbol{a}_2$-axis. This rotation matrix can be computed to be:

$$ {}^{\mathcal{A}}\boldsymbol{R}_{\mathcal{B}} = \begin{bmatrix} \cos\psi\cos\theta - \sin\phi\sin\psi\sin\theta & -\cos\phi\sin\psi & \cos\psi\sin\theta + \cos\theta\sin\phi\sin\psi \\ \cos\theta\sin\psi + \cos\psi\sin\phi\sin\theta & \cos\phi\cos\psi & \sin\psi\sin\theta - \cos\psi\cos\theta\sin\phi \\ -\cos\phi\sin\theta & \sin\phi & \cos\phi\cos\theta \end{bmatrix} \tag{3.18} $$

We also define the angular transformation matrix

$$ \boldsymbol{T}_{\boldsymbol{\Theta}} = \begin{bmatrix} 1 & \dfrac{\sin\phi\sin\theta}{\cos\theta} & \dfrac{\cos\phi\sin\theta}{\cos\theta} \\ 0 & \cos\theta & -\sin\phi \\ 0 & \dfrac{\sin\phi}{\cos\theta} & \dfrac{\cos\phi}{\cos\theta} \end{bmatrix} \quad \forall\theta \neq \frac{\pi}{2} + k\pi, \quad k \in \mathbb{Z} \tag{3.19} $$

Describing the angular relation:

$$ {}^{\mathcal{A}}\dot{\boldsymbol{\Theta}} = \boldsymbol{T}_{\boldsymbol{\Theta}}{}^{\mathcal{B}}\dot{\boldsymbol{\Theta}} \tag{3.20} $$

Where the angular rotation speed of an arbitrary frame $\mathcal{F}$ can be defined as:

$$ {}^{\mathcal{F}}\dot{\boldsymbol{\Theta}} = {}^{\mathcal{F}}\boldsymbol{\omega} \tag{3.21} $$

3.3 Full Dynamic Quadcopter Model

3.3.1 Kinematics

The kinematic velocities for the two frames can be related by the following equation:

$$ \dot{\boldsymbol{\eta}} = \boldsymbol{J}_{\boldsymbol{\Theta}}\boldsymbol{\nu} \tag{3.22} $$

Where

$$ \boldsymbol{J}_{\boldsymbol{\Theta}} = \begin{bmatrix} {}^{\mathcal{A}}\boldsymbol{R}_{\mathcal{B}} & \boldsymbol{0}_{3\times3} \\ \boldsymbol{0}_{3\times3} & \boldsymbol{T}_{\boldsymbol{\Theta}} \end{bmatrix} \tag{3.23} $$

This as defined in [22].

3.3.2 Kinetics

The kinetics of a rigid-body can be written as the external forces $\tau_1 \in \mathbb{R}^3$:

$$m(\begin{bmatrix} \dot{u} \\ \dot{v} \\ \dot{w} \end{bmatrix} + \begin{bmatrix} p \\ q \\ r \end{bmatrix} \times \begin{bmatrix} u \\ v \\ w \end{bmatrix}) = \tau_1 \tag{3.24}$$

And the external moments $\tau_2 \in \mathbb{R}^3$:

$$\boldsymbol{I}_{CM} \begin{bmatrix} \dot{p} \\ \dot{q} \\ \dot{r} \end{bmatrix} + \begin{bmatrix} p \\ q \\ r \end{bmatrix} \times (\boldsymbol{I}_{CM} \begin{bmatrix} p \\ q \\ r \end{bmatrix}) = \tau_2 \tag{3.25}$$

Where $\boldsymbol{I}_{CM}$ is the moment of inertia about the drone's center of mass and is defined by:

$$\boldsymbol{I}_{CM} = \begin{bmatrix} I_{xx} & I_{xy} & I_{xz} \\ I_{xy} & I_{yy} & I_{yz} \\ I_{xz} & I_{yz} & I_{zz} \end{bmatrix} \tag{3.26}$$

By assuming the center of mass is located in the origin of the body fixed reference frame $\mathcal{B}$, we can rewrite equations (3.24)-(3.25) as:

$$\boldsymbol{M} \begin{bmatrix} \dot{u} \\ \dot{v} \\ \dot{w} \\ \dot{p} \\ \dot{q} \\ \dot{r} \end{bmatrix} + \boldsymbol{C} \begin{bmatrix} u \\ v \\ w \\ p \\ q \\ r \end{bmatrix} = \boldsymbol{M}\dot{\boldsymbol{\nu}} + \boldsymbol{C}\boldsymbol{\nu} = \boldsymbol{\tau}_{RB} \tag{3.27}$$

Where:

$$\boldsymbol{M} = \begin{bmatrix} m\boldsymbol{I}_{3\times3} & \boldsymbol{0}_{3\times3} \\ \boldsymbol{0}_{3\times3} & \boldsymbol{I}_{CM} \end{bmatrix} \tag{3.28}$$

And:

$$\boldsymbol{C} = \begin{bmatrix} m\boldsymbol{S}({}^{\mathcal{B}}\dot{\boldsymbol{\Theta}}) & \boldsymbol{0}_{3\times3} \\ \boldsymbol{0}_{3\times3} & -\boldsymbol{S}(\boldsymbol{I}_{CM}{}^{\mathcal{B}}\dot{\boldsymbol{\Theta}}) \end{bmatrix} \tag{3.29}$$

Given that $\boldsymbol{S}(\boldsymbol{\lambda})$ is a generated skew symmetric matrix defined by:

$$\boldsymbol{S}(\boldsymbol{\lambda}) := \begin{bmatrix} 0 & -\lambda_3 & \lambda_2 \\ \lambda_3 & 0 & -\lambda_1 \\ -\lambda_2 & \lambda_1 & 0 \end{bmatrix} \tag{3.30}$$

Where $\boldsymbol{\lambda} \in \mathbb{R}^3$, $\boldsymbol{S}(\boldsymbol{\lambda})^T = -\boldsymbol{S}(\boldsymbol{\lambda})$ and $\boldsymbol{\lambda} \times \boldsymbol{\lambda} = \boldsymbol{S}(\boldsymbol{\lambda})\boldsymbol{\lambda}$ is satisfied for skew symmetry. The gravitational forces working on the drone's body can be described by:

$$ {}^{\mathcal{B}}\boldsymbol{g} = - \begin{bmatrix} ({}^{\mathcal{A}}\boldsymbol{R}_{\mathcal{B}})^T \begin{bmatrix} 0 \\ 0 \\ mg \end{bmatrix} \\ \mathbf{0}_{3\times 1} \end{bmatrix} \tag{3.31} $$

Hence, the resulting dynamic model of the quadcopter now becomes:

$$ \begin{aligned} \dot{\boldsymbol{\eta}} &= \boldsymbol{J}_{\boldsymbol{\Theta}}\boldsymbol{\nu} \\ \boldsymbol{M}\dot{\boldsymbol{\nu}} + \boldsymbol{C}\boldsymbol{\nu} + {}^{\mathcal{B}}\boldsymbol{g} &= \boldsymbol{\tau}_{SUM} \end{aligned} \tag{3.32} $$

Where $\boldsymbol{\tau}_{SUM}$ consists of all external forces that affects the drone, except gravity.

3.3.3 Dominant Dynamics for the Quadcopter

Due to the symmetry of the quadcopter and its ability of generating thrust in all directions by manipulation the roll- and pitch angle, we want derive the equations represented in the $\mathcal{A}$-frame. By rewriting (3.22) and (3.32), we can get:

$$ \begin{aligned} {}^{\mathcal{A}}\dot{\boldsymbol{p}} &= {}^{\mathcal{A}}\boldsymbol{v} \\ {}^{\mathcal{A}}\dot{\boldsymbol{\Theta}} &= \begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} = \boldsymbol{T}_{\boldsymbol{\Theta}}{}^{\mathcal{B}}\boldsymbol{\omega} \end{aligned} \tag{3.33} $$

Where ${}^{\mathcal{A}}\boldsymbol{p} = \begin{bmatrix} x^{\mathcal{A}} & y^{\mathcal{A}} & z^{\mathcal{A}} \end{bmatrix}^T \in \mathbb{R}^3$ is the NEU position of the quadcopter. Hence, we get the total dominant dynamics of:

$$ \begin{aligned} {}^{\mathcal{A}}\dot{\boldsymbol{p}} &= {}^{\mathcal{A}}\boldsymbol{v} \\ m{}^{\mathcal{A}}\dot{\boldsymbol{v}} &= m{}^{\mathcal{B}}\boldsymbol{g} + {}^{\mathcal{A}}\boldsymbol{R}_{\mathcal{B}} f \\ {}^{\mathcal{A}}\dot{\boldsymbol{\Theta}} &= \boldsymbol{T}_{\boldsymbol{\Theta}}{}^{\mathcal{B}}\boldsymbol{\omega} \\ \boldsymbol{I}^{\mathcal{B}}\dot{\boldsymbol{\omega}} &= \boldsymbol{S}(\boldsymbol{I}^{\mathcal{B}}\boldsymbol{\omega})^{\mathcal{B}}\boldsymbol{\omega} + \boldsymbol{M} \end{aligned} \tag{3.34} $$

Here, ${}^{\mathcal{A}}\boldsymbol{p}$ and ${}^{\mathcal{A}}\boldsymbol{v}$ are the quadcopter's position and linear velocity in the inertial frame $\mathcal{A}$ respectively. ${}^{\mathcal{A}}\boldsymbol{R}_{\mathcal{B}}$ is the rotation matrix from the inertial frame to the body-fixed frame, ${}^{\mathcal{B}}\boldsymbol{\omega}$ is the angular velocity of the quadcopter, represented in the body-fixed frame. $\boldsymbol{I}^{\mathcal{B}}$ is the body-fixed inertia tensor matrix, f is the vertical thrust in the body-aligned z-axis, and $\boldsymbol{M}$ is the applied torque about the quadcopter, generated by the motors.

3.4 Model Implementation and Control

In this thesis, the simulations are done on the full dynamic model of a quadcopter. This section explains the dynamics that is implemented in the simulator and the algorithms are tested on. This model, position- and attitude control is solely based on the same as in [23].

Symbol	Explanation
F_i	Force generated from rotor i
M_i	Momentum generated from rotor i
${}^{\mathcal{A}}\boldsymbol{r}$	Drone NEU position
$\boldsymbol{u}$	Input vector
L	Arm length from center of mass till rotor
m	Mass of the quadcopter
g	Gravity constant
${}^{\mathcal{B}}\boldsymbol{\omega}$	Angular velocities in the body-frame
Ω_i	Angular velocities of rotor i

Table 3.1: Table of the quadcopter model parameters

For a quadcopter with four propellers we have the general equations for forces and moments:

$$F_i = k_F \Omega_i^2 \tag{3.35}$$

$$M_i = k_M \Omega_i^2 \tag{3.36}$$

Where each rotor of the drone has an angular speed Ω_i and produces the vertical force F_i and a moment M_i. The constants k_F and k_M can be found from a fixed rotor at steady-state.

3.4.1 Newton-Euler Equations of Motion

We let ${}^{\mathcal{A}}\boldsymbol{r}$ denote the drone's center of gravity position in the inertial frame $\mathcal{A}$. The forces on the system in an undisturbed environment will then be the thrust forces from each rotor F_i working in the $\boldsymbol{b}_3$-direction and the gravity mg working in the $-\boldsymbol{a}_3$-direction. The equation governing the acceleration of the center of mass is

$$m{}^{\mathcal{A}}\ddot{\boldsymbol{r}} = \begin{bmatrix} 0 \\ 0 \\ -mg \end{bmatrix} + {}^{\mathcal{A}}\boldsymbol{R}_{\mathcal{B}} \begin{bmatrix} 0 \\ 0 \\ F_1 + F_2 + F_3 + F_4 \end{bmatrix} \tag{3.37}$$

The angular acceleration determined by the Euler equations is:

$$I^{\mathcal{B}}\dot{\boldsymbol{\omega}} = \begin{bmatrix} L(F_2 - F_4) \\ L(F_3 - F_1) \\ M_1 - M_2 + M_3 - M_4 \end{bmatrix} - {}^{\mathcal{B}}\boldsymbol{\omega} \times I^{\mathcal{B}}\boldsymbol{\omega} \tag{3.38}$$

From equation (3.37) we have the input:

$$u_1 = F_1 + F_2 + F_3 + F_4 \tag{3.39}$$

Which is the total trust applied to the quadcopter, and equation (3.38) yields the input:

$$\boldsymbol{u}_2 = \begin{bmatrix} L(F_2 - F_4) \\ L(F_3 - F_1) \\ M_1 - M_2 + M_3 - M_4 \end{bmatrix} \tag{3.40}$$

Which represents the moment about the x, y and z-axis respectively.
Using (3.35)-(3.36), we now introduce a relation parameter γ, which is the the relation between the moment and force.

$$\gamma = \frac{M_i}{F_i} = \frac{k_M}{k_F} \tag{3.41}$$

The body-frame Newton-Euler equation (3.38) can then be transformed to:

$$I^{\mathcal{B}}\dot{\boldsymbol{\omega}} = \begin{bmatrix} 0 & L & 0 & -L \\ -L & 0 & L & 0 \\ \gamma & -\gamma & \gamma & -\gamma \end{bmatrix} \begin{bmatrix} F_1 \\ F_2 \\ F_3 \\ F_4 \end{bmatrix} - {}^{\mathcal{B}}\boldsymbol{\omega} \times I^{\mathcal{B}}\boldsymbol{\omega} \tag{3.42}$$

Where:

$$\boldsymbol{u}_2 = \begin{bmatrix} L(F_2 - F_4) \\ L(F_3 - F_1) \\ M_1 - M_2 + M_3 - M_4 \end{bmatrix} = \begin{bmatrix} 0 & L & 0 & -L \\ -L & 0 & L & 0 \\ \gamma & -\gamma & \gamma & -\gamma \end{bmatrix} \begin{bmatrix} F_1 \\ F_2 \\ F_3 \\ F_4 \end{bmatrix} \tag{3.43}$$

Combined, the total input vector becomes:

$$\boldsymbol{u} = \begin{bmatrix} u_1 \\ \boldsymbol{u}_2 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 0 & L & 0 & -L \\ -L & 0 & L & 0 \\ \gamma & -\gamma & \gamma & -\gamma \end{bmatrix} \begin{bmatrix} F_1 \\ F_2 \\ F_3 \\ F_4 \end{bmatrix} \tag{3.44}$$

3.4.2 Nominal State Linearization

The controllers used in this thesis are derived by linearizing the equations of motion at an operating point corresponding to the hover state where the roll and pitch angles are small.

$$\begin{aligned} {}^{\mathcal{A}}\boldsymbol{r} &= {}^{\mathcal{A}}\boldsymbol{r}_0 \\ {}^{\mathcal{A}}\dot{\boldsymbol{r}} &= \mathbf{0}_{3\times 1} \\ \phi^{\mathcal{A}} &= \theta^{\mathcal{A}} = 0 \\ \psi^{\mathcal{A}} &= \psi_0^{\mathcal{A}} \\ \dot{\phi}^{\mathcal{A}} &= \dot{\theta}^{\mathcal{A}} = \dot{\psi}^{\mathcal{A}} = 0 \end{aligned} \tag{3.45}$$

The propeller force in this state is given by:

$$F_{i,0} = \frac{mg}{4} \tag{3.46}$$

The nominal input values for hover is given by:

$$\begin{aligned} u_{1,0} &= mg \\ \boldsymbol{u}_{2,0} &= \mathbf{0}_{3\times 1} \end{aligned} \tag{3.47}$$

By linearizing equation (3.37) and (3.38), we get:

$$\begin{aligned} \ddot{r}_1^{\mathcal{A}} &= g(\Delta\theta^{\mathcal{A}} \cos\psi_0^{\mathcal{A}} + \Delta\phi^{\mathcal{A}} \sin\psi_0^{\mathcal{A}}) \\ \ddot{r}_2^{\mathcal{A}} &= g(\Delta\theta^{\mathcal{A}} \sin\phi_0^{\mathcal{A}} - \Delta\phi^{\mathcal{A}} \cos\psi_0^{\mathcal{A}}) \\ \ddot{r}_3^{\mathcal{A}} &= \frac{u_1}{m} - g \end{aligned} \tag{3.48}$$

and

$${}^{\mathcal{B}}\boldsymbol{\omega} = \frac{1}{\boldsymbol{I}^{\mathcal{B}}} \begin{bmatrix} 0 & L & 0 & -L \\ -L & 0 & L & 0 \\ \gamma & -\gamma & \gamma & -\gamma \end{bmatrix} \begin{bmatrix} F_1 \\ F_2 \\ F_3 \\ F_4 \end{bmatrix} \tag{3.49}$$

Where $\Delta\phi^{\mathcal{A}} = \phi^{\mathcal{A}} - \phi_0^{\mathcal{A}}$ and $\Delta\theta^{\mathcal{A}} = \theta^{\mathcal{A}} - \theta_0^{\mathcal{A}}$. By assuming symmetric rotorcraft ($I_{xx} = I_{yy}$), we get the following angular accelerations:

$$\begin{aligned}
\omega_1^{\mathcal{B}} = \dot{p} &= \frac{u_{2,x}}{I_{xx}} = \frac{L}{I_{xx}}(F_2 - F_4) \\
\omega_2^{\mathcal{B}} = \dot{q} &= \frac{u_{2,y}}{I_{yy}} = \frac{L}{I_{yy}}(F_3 - F_1) \\
\omega_3^{\mathcal{B}} = \dot{r} &= \frac{u_{2,z}}{I_{zz}} = \frac{\gamma}{I_{zz}}(F_1 - F_2 + F_3 - F_4)
\end{aligned} \tag{3.50}$$

As each component of angular acceleration only depend on the appropriate component of $\boldsymbol{u}_2$, we now see that the equations of motion are decoupled in terms of angular acceleration.

3.4.3 Position and Attitude Control

In order to obtain hover or follow a desired trajectory $\boldsymbol{z}_{des}$, control of the input $\boldsymbol{u}$ is needed. By determining the error in the drone's position, a position controller can be derived from (3.48) to directly obtain u_1 as well as a desired orientation. Note that all parameters in this section is in the inertial NEU-frame as this is not noted in the notation.

Attitude Control

The attitude input $\boldsymbol{u}_2$ can be described by:

$$\boldsymbol{u}_2 = \boldsymbol{I} \begin{bmatrix} k_{p,\phi}(\phi_{des} - \phi) + k_{d,\phi}(p_{des} - p) \\ k_{p,\theta}(\theta des - \theta) + k_{d,\theta}(q_{des} - q) \\ k_{p,\psi}(\psi_{des} - \psi) + k_{d,\psi}(r_{des} - r) \end{bmatrix} \tag{3.51}$$

This represents an attitude PD-controller. Note that since the controller is based on linearized equations, the attitude has to operate close to the nominal hover state where roll and pitch angles are small.

Position Control

For position control, we derive two different controllers. One for hover, and one for trajectory following. For both methods, the position control will determine the desired roll (ϕ_{des}) and pitch (θ_{des}) angles used to compute the desired $\boldsymbol{u}_2$ in equation (3.51).

The desired trajectory can be defined as:

$$\boldsymbol{z}_{des} = \begin{bmatrix} \boldsymbol{r}_T(t) \\ \psi_T(t) \end{bmatrix} \tag{3.52}$$

Where $\boldsymbol{r}_T$ is the desired position vector and ψ_T is the specified yaw angle we are trying to track.

Position: Hover Control

For stationary position keeping at a desired position vector $\boldsymbol{r}_T(t) = \boldsymbol{r}_0$ and $\psi_T = \psi_0$. The desired acceleration commands $\ddot{r}_{des,i}$ are to be calculated from a new PD controller. The position error is defined as $\boldsymbol{e} = (\boldsymbol{r}_T - \boldsymbol{r})$. We want to guarantee that this error converges exponentially towards zero and require:

$$(\ddot{\boldsymbol{r}}_T - \ddot{\boldsymbol{r}}_{des}) + k_d(\dot{\boldsymbol{r}}_T - \dot{\boldsymbol{r}}_{des}) + k_p(\boldsymbol{r}_T - \boldsymbol{r}_{des}) = 0 \tag{3.53}$$

For hover, $\ddot{r}_{T,i} = \dot{r}_{T,i} = 0$. A relationship between desired acceleration and roll and pitch angles can be derived form (3.48) and the given values for ϕ_0 and θ_0.

$$\begin{aligned} \ddot{r}_{1,des} &= g(\theta_{des}\cos\psi_T + \phi_{des}\sin\psi_T) \\ \ddot{r}_{2,des} &= g(\theta_{des}\sin\phi_T - \phi_{des}\cos\psi_T) \\ \ddot{r}_{3,des} &= \frac{u_1}{m} - g \end{aligned} \tag{3.54}$$

Hence, the controlled input u_1 for hover becomes:

$$u_1 = mg + m\ddot{r}_{3,des} = mg - m(k_{d,3}\dot{r}_3 + k_{p,3}(r_3 - r_{3,0})) \tag{3.55}$$

Where as the desired roll and pitch for the attitude controller becomes:

$$\begin{aligned} \phi_{des} &= \frac{1}{g}(\ddot{r}_1\sin\psi_T - \ddot{r}_2\cos\psi_T) \\ \theta_{des} &= \frac{1}{g}(\ddot{r}_1\cos\psi_T + \ddot{r}_2\sin\psi_T) \end{aligned} \tag{3.56}$$

The desired roll- and pitch are both set to zero fro hover:

$$\begin{aligned} p_{des} &= 0 \\ q_{des} &= 0 \end{aligned} \tag{3.57}$$

The yaw angle $\psi_T(t)$ is independent and prescribed by the trajectory generator, and the yaw angle and angle rate become:

$$\begin{aligned} \psi_{des} &= \psi_T(t) \\ r_{des} &= \dot{\psi}_T(t) \end{aligned} \tag{3.58}$$

These equations provides the setpoints for the attitude control in (3.51) for hover, and hence the whole control input $\boldsymbol{u}$.

Position: Trajectory Control

Three-dimensional trajectories can be followed by using a trajectory controller assuming modest accelerations such that the hover assumptions hold. The controller can be derived with the same condition as in (3.53), but with $\ddot{r}_{T,i}$ and $\dot{r}_{T,i}$ not necessarily being zero, but obtained from the specification of the trajectory. For near-hover conditions and linear dynamics with no saturation on the inputs, a controller based on the condition (3.53) will guarantee to drive the error exponentially towards zero. However, there might occur errors in the model or limitations to the inputs that hinders the drone to follow its planned trajectory. Hence, a modification to the controller is proposed: Consider $\boldsymbol{r}_T$ the closest point on the desired trajectory to the current position $\boldsymbol{r}$, and the velocity and acceleration as $\dot{\boldsymbol{r}}_T$ and $\ddot{\boldsymbol{r}}_T$ respectively. The unit tangent vector of the trajectory, or the unit vector along $\dot{\boldsymbol{r}}_T$ is denoted as $\hat{\boldsymbol{t}}$. The unit normal to the trajectory, $\hat{\boldsymbol{n}}$ is derived by taking the time derivative of the tangent vector, and the unit binormal vector, $\hat{\boldsymbol{b}}$, is defined as the cross-product $\hat{\boldsymbol{t}} \times \hat{\boldsymbol{n}}$. The position and velocity errors are now redefined as follows:

$$\boldsymbol{e}_{pos} = \big((\boldsymbol{r}_T - \boldsymbol{r}) \cdot \hat{\boldsymbol{n}}\big)\hat{\boldsymbol{n}} + \big((\boldsymbol{r}_T - \boldsymbol{r}) \cdot \hat{\boldsymbol{b}}\big)\hat{\boldsymbol{b}} \tag{3.59}$$

$$\boldsymbol{e}_{vel} = \dot{\boldsymbol{r}}_T - \dot{\boldsymbol{r}} \tag{3.60}$$

Note that we only consider the position error in the plane that is normal to the curve at the closest point, hence ignoring the position error in the tangential direction. The commanded acceleration is calculated from the PD-feedback controller, which is redefined now as:

$$(\ddot{\boldsymbol{r}}_T - \ddot{\boldsymbol{r}}_{des}) + \boldsymbol{k}_d \boldsymbol{e}_{vel} + \boldsymbol{k}_d \boldsymbol{e}_{pos} = 0 \tag{3.61}$$

As for the hover controller, we now use equations (3.56)-(3.58) to compute the desired roll-, pitch- and yaw angles and the angular rates in order to feed the values to the attitude control.

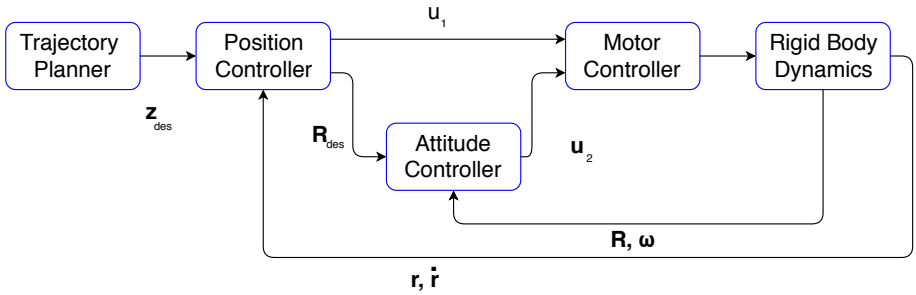


Figure 3.2: Position and attitude control loops

3.5 Collision Avoidance

In addition to the low level PID-controller for position and attitude, we have a high level controller for collision avoidance. Three methods are suggested for the high level control implemented in the simulator:

- The Null-Space Based Behavioral control (NSB) algorithm as a global version
- A self-developed reactive logic as a local version
- The Velocity Obstacle algorithm as a local version

The strategies proposed above are tested for safe navigation around obstacles such as walls and valves, and are all covered in chapter 4. To test our strategies, a simulator is built in MATLAB. The next section explains the basics of the simulator and what the simulation returns.

3.6 Simulator

The simulator created for this thesis is based on a simulator used in the subject MEAM620 at the university of Pennsylvania [1]. In order to simulate the desired autonomous inspection, a trajectory needs to be given to the simulator. Based on a set of given waypoints, the trajectory planner decides the drone's trajectory, which is calculated by the low level controller. The position controller runs at 20 Hz with a zero-order hold, while the attitude controller runs at 100 Hz. This is done in order to consider the practicalities of implementing the controllers for a real drone. When obstacles are detected,

the high level controller will make the drone deviate from its originally planned trajectory in order to avoid collisions. The drone has a $360°$ coverage of radar sensors, the simulator represents these as six sectors, covering $60°$ each with a range detection radius of 5 meters. Walls, pipes etc. are all displayed in the visual representation of the simulator in addition to the drone itself. The drone will either know the obstacles' locations based on a global map or sense them from the on board sensors. The obstacle information will be considered for the collision avoidance. An overall representation of the simulator architecture is illustrated in figure 3.3.

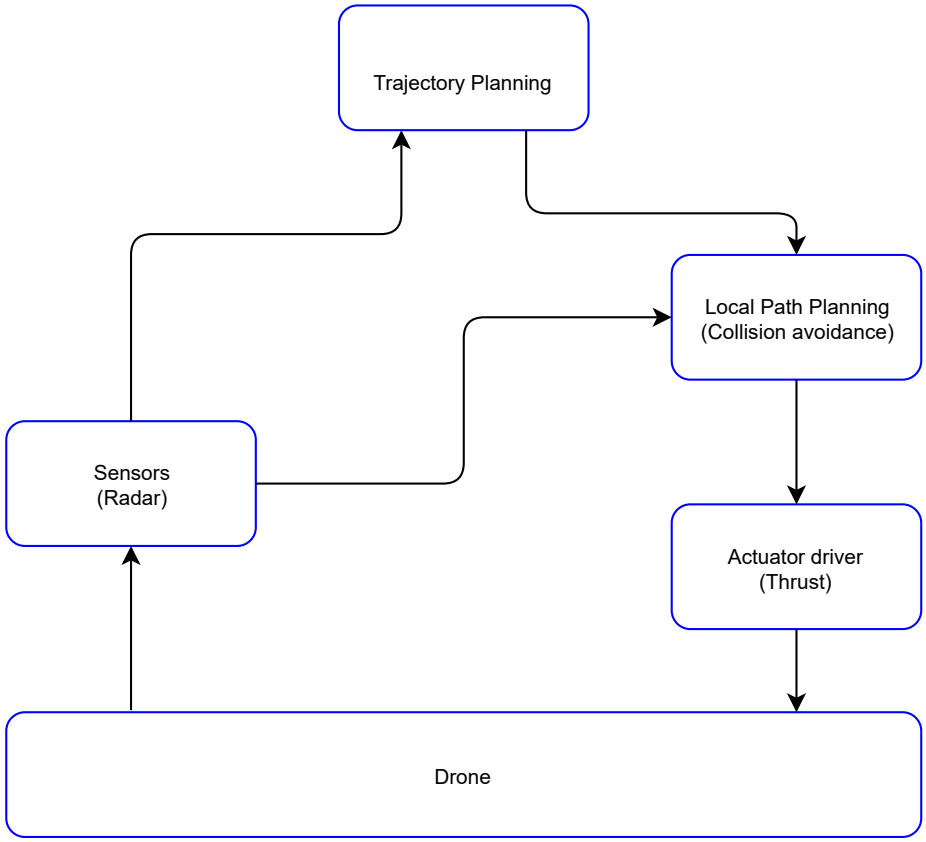


Figure 3.3: Simulator architecture

Chapter 4

Theory and Methods

This chapter covers the theory of radar sensors and explains the methodology of the different algorithms for collision avoidance. Available sensors are discussed and the functionality and properties of the radar sensor is explained. The global Null-Space Based Behavioral Control (NSB) algorithm is explained in detail, followed by the local approaches; the self-developed reactive logic algorithm and the Velocity Obstacle (VO) algorithm.

4.1 Sensors

In order to make our drone perceive its surrounding environment and current state, a set of sensors is used. Commonly, a quadcopter consists of at least an Inertial Measurement Unit (IMU) for attitude acquisition and a Global Positioning System (GPS) for position measure [8]. Since this thesis focuses on obstacle avoidance, the sensors considered are range sensors, determining distance and/or angle to obstacles. Note that since the inspection is done inside a relatively small tank, the GPS will most likely be unavailable and we need to use other perception methods. The GPS accuracy is typically at 5 meters [24], which is far from good enough for collision avoidance and proximity keeping in cluttered environments.
For inspection purposes, there exists a scope of different sensors that can be used. Different conditions of environment and visual constraints may influence the choice of which sensors that are ideal. Some of the commonly used sensors in robotics for range detection are LIDAR and cameras. These sensors have already been applied to a lot

of drone applications worldwide. As described in [25], LIDAR is a good choice for SLAM (Simultaneous Localization And Mapping) applications. Cameras are great for capturing visual data, and can in the right light settings yield a large amount of data that can be used for navigation and collision avoidance. This thesis, however, focuses on the much less applied usage of radar as the sensor for collision detection. Even though LIDAR is the most common choice in these types of applications, radar technology has some properties which are unique and might just be better suited for our specific application.
The main drawbacks of the LIDAR are the light emitted waves from the sensor will reflect if there exist particles in the air and that the LIDAR scan returns a 2D plane. The drawbacks of camera usage are the requirement of good lighting and clear view at all times, the distance measured from the stereo vision is not perfectly accurate and the data from the two cameras need to be mapped together, which requires sufficient computational power [26]. The next section covers the radar sensor's functionality and why we have chosen it as a range detection sensor for our application.

4.2 Radar

As the name suggests, radar (Radio Detection And Ranging) works on the principle of radio wave pulses being transmitted from the sensor and received back again. Traveling at the speed of light, the distance to detected objects can be measured by the simple equation:

$$d_{obj} = \frac{v_l t_{rw}}{2} \tag{4.1}$$

Where d_{obj} is the distance to the object, v_l is the speed of light and t_{rw} is the radio wave pulse's round-trip time, hence the division of 2. Radio waves are not reflected from particles in the air, so the distance measured will always be correct if no disturbances or noise. This is an advantage for autonomous inspection of hazardous environments with smoke etc. where humans are unavailable to operate. Because of its robustness and the information this sensor provides, radar is the desired sensor choice. Also, by the usage of SAR, we can create 3D images based on the radar data [27].

4.2.1 Radar Data Processing

A radar system generally consists of two main components: a signal processor for target detection, and a data processor for information retrieval [27]. When a target is detected, the signal will be transmitted to a data recording device, where characteristic parameters of the target are recorded. These recordings need to be processed further

in the data processor to predict the obtained measurement data, such as target distance and bearing angle.
The measurement data acquired from the radar sensors contains two types of errors. Resulting from the interior noise of the measurement system, the random error is the first error generated. This random error may vary from each measurement to another and can be reduced to a certain degree by increasing the measurement frequency and minimizing its variance by filtering. The second error results from antennas, servo systems and other non-calibration factors in the data correction process, and is called system error. It is complex, slowly varying and non-random, hence its viewed as an unknown variable in a relatively long period of time. When the ratio of system errors to random errors is greater or equal to 1, the system error must be corrected as the effect of distributed track fusion and centralized measurement fusion deteriorates markedly [27]. In the case of obviously abnormal values from the radar measurement data, an outlier rejection process is done to remove these abnormalities.

4.2.2 Radar Sectors

The drone consists of a specified set of six radar sensors on board, each covering a $60°$ sector around the drone as illustrated in figure 4.1. These sensors calculate the distance from the sensor to a detected object within the radar sector by measuring the time it takes for the radio pulse to reflect back. The sensor only returns a signal concluding something is in the sector, but doesn't know exactly where in this sector the obstacle is. The output of the processed radar signal will be Boolean; *true* if there is something in the sector, and *false* if not. In order to retrieve position of the sensed obstacle, a combination of the different radar sector readings can construct an estimate of the obstacle's exact position.

Time to Collision

In order to achieve successful collision avoidance, it is important to have an estimate the time to collision for the sensor's field of view. Given that ρ is the length vector between the drone and an obstacle, the time to collision is given by:

$$t_c = \frac{\rho}{\dot{\rho}} \tag{4.2}$$

This as defined in [28]. The time to collision can be used as a threshold to determine if an obstacle will cause a collision in a shorter time interval. Consider t_{safe} to be a time the drone can operate in any state with the ability to correct itself if an obstacle occurs. If $t_c \leq t_{safe}$, this implies that the integrity of the drone is threatened by the

obstacle, and collision avoidance maneuvers needs to be executed. Hence, we do not need to process any radar data where $t_c > t_{safe}$.

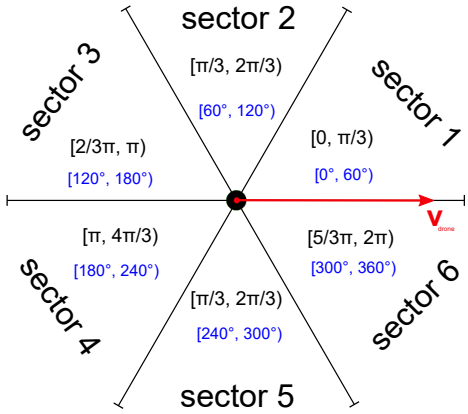


Figure 4.1: Radar sectors

4.3 Low Level Control

For our drone, the classical PID-controller is used as a low level controller. It is used to ensure stability of our simulated drone in its attitude and position.

4.3.1 PID Implementation

The PID is a closed loop control system trying to get the desired values by adjusting the input. PID controllers are used for a large scope of applications and the essential part is the adjustment of the tuning parameters for ultimate performance in different conditions. The parameters are tuned to satisfy the following three objectives:

1. Stability and stability robustness
2. Transient response
3. Steady state accuracy

Mathematically, the input acquired from the PID controller can be expressed as:

$$\boldsymbol{u} = K_p\tilde{\boldsymbol{x}} + K_i \int_0^t \tilde{\boldsymbol{x}} d\tau + K_d\dot{\tilde{\boldsymbol{x}}} \tag{4.3}$$

Where $\tilde{\boldsymbol{x}} = \boldsymbol{x}_d - \boldsymbol{x}$ represents the error between the drone's desired position $\boldsymbol{x}_d$ and current position $\boldsymbol{x}$, K_p the proportional gain, K_i the integral gain and K_d the derivative gain.

4.3.2 Reference Signal

The controller designed requires a smooth $\mathcal{C}^3$ trajectory. Given a sequence of waypoints (corners of the cube/ walls), the trajectory can be generated by feeding the waypoints through a reference model of sufficient order. This is resembled by a 4th order low pass filter on the following form:

$$\boldsymbol{x}^{(4)} + 4\zeta\omega_0\boldsymbol{x}^{(3)} + (2 + 4\zeta^2)\omega_0^2\ddot{\boldsymbol{x}} + 4\zeta\omega_0^3\dot{\boldsymbol{x}} + \omega_0^4\boldsymbol{x} = \omega_0^4\boldsymbol{x}_d \tag{4.4}$$

Here, $\boldsymbol{x} \in \mathbb{R}^3$ is the reference signal, $\boldsymbol{x}_d \in \mathbb{R}^3$ is the current waypoint and $\zeta, \omega_0 \in \mathbb{R}$ are the tuning parameters. Considering only the first dimension of $\boldsymbol{x}$, a reference position of $\boldsymbol{x}_{d_3} = 20$ will give the result in figure 4.2a. Because of the drone's constraints, this is an unsaturated and hence unfeasible result. The drone's dynamics, such as the maximum velocity and acceleration are violated and can be solved by introducing a cascaded controller system by rearranging (4.4) with respective saturation constraints.

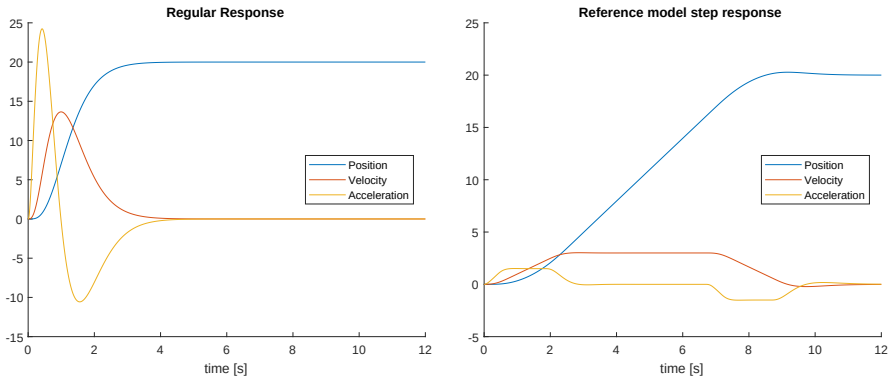
$$\boldsymbol{x}^{(4)} = \boldsymbol{u} \tag{4.5a}$$

$$\tau_1 = \text{sat}(k_1(\boldsymbol{x}_d - \boldsymbol{x}), v_{\max}) \tag{4.5b}$$
$$\tau_2 = \text{sat}(k_2(\tau_1 - \boldsymbol{x}^{(1)}), a_{\max}) \tag{4.5c}$$
$$\tau_3 = \text{sat}(k_3(\tau_2 - \boldsymbol{x}^{(2)}), j_{\max}) \tag{4.5d}$$
$$\boldsymbol{u} = k_4(\tau_3 - \boldsymbol{x}^{(3)}) \tag{4.5e}$$

Where v_{max}, a_{max} and j_{max} are the maximum velocity, acceleration and jerk respectively. By comparing (4.5) with (4.4), we can find the parameters k_1, k_2, k_3, k_4 to be.

$$k_4 = 4\zeta\omega_0 \tag{4.6a}$$
$$k_3 = \frac{(2 + 4\zeta^2)\omega_0^2}{k_4} \tag{4.6b}$$
$$k_2 = \frac{4\zeta\omega_0^3}{k_4 k_3} \tag{4.6c}$$
$$k_1 = \frac{\omega_0^4}{k_4 k_3 k_2} \tag{4.6d}$$

By setting the limited values for $v_{\max}$, $a_{\max}$ and $j_{\max}$, we are now able to construct a trajectory satisfying the drone's physical constraints. This naturally yields a slower response, but guarantees feasible, smooth $\mathcal{C}^3$ trajectories for the controller.



(a) Normal Response **(b)** Reference Filter

Figure 4.2: Responses with and without reference filter

4.4 High Level Control for Global Maps: NSB

For the high level control, the different methods focus on the information available. For the case where information about the environment the drone operates in is provided, there exist a scope of methods for solving the mission. In this thesis, we have focused on the Null-Space Based behavioral control algorithm (NSB) as this is a clever global

method for optimal collision-free trajectories. The NSB algorithm is a behavioral-based approach for control of autonomous robotic systems. The output of the single elementary behaviors are combined to compose a complex behavior.

The idea is to decompose the overall problem into several sub problems (tasks, functional modules, motor schemas, behaviors). The NSB algorithm is presented as a unified framework in [4], consisting of two main behavioral approaches.

4.4.1 Competitive Approach

Can be seen as a competition between behaviors. The strategy can be implemented in many ways, the layered control system proposed in [29] is an example. Each behaviour is connected to an asynchronous layer module represented in an augmented finite state machine. Based on the sensor data, each layer provides the independently calculated desired output, and is chosen by priorities of each layer module. The coordination can be viewed as a competition among behaviours; only one behaviour wins and its response only is sent to the drone for execution

4.4.2 Cooperative Approach

Is an alternative to the competitive one. A fusion of behaviours gives the ability to concurrently use the collection of several behaviour outputs. A sum of all motion commands is given to the actuating force, where each behavior can be weighted by a supervisor. A common cooperative method for behavioural approach is the motor schema control. A motor schema is a basic unit of behavior specification for the navigation of a mobile robot. The schemas are multiple concurrent processes that operate in conjunction with associated perceptual schemas and contribute independently to the overall concerted action of the vehicle. The motivation behind the use of schemas for this domain is drawn from neuroscientific, psychological, and robotic sources [30]. A variant of the potential field algorithm is used to produce the appropriate velocity and steering commands for the robot. Simulation results and actual mobile robot experiments demonstrate the feasibility of this approach. The cooperative method's main difference from the competitive is the realization of a linear combination of the outputs elaborated for each task. Hence, no task is completely achieved, but a compromised solution is found.

4.4.3 Behavioural Control

Behaviors are expressed through a function of the drone configuration that measures the degree of fulfillment of a task. When in a static environment, such a task is achieved when its output is constant at a value that minimizes the task function. If for instance the task output is the velocity of the drone, then in order to reach the desired goal a distance-from-goal task function can be considered. The velocity command will then be generated to reduce the distance between the drone and the goal, and it will be zero when the goal position is reached.

If avoidance of an obstacle also needs to be considered, then another velocity command needs to be generated to increase the distance between the drone and the obstacle. This velocity command is zero once the drone is out of reach from the obstacle. When the obstacle is somewhere in the line of sight as the drone moves towards the goal, the two behaviors come in conflict and the two velocity commands will counteract each other. The drone can either approach the goal position or escape the obstacle.

This is where the feature of coordination comes in handy. Handling multiple elementary tasks of behavioral approach such that all tasks can be achieved simultaneously. For competitive methods, only one task is selected at each time instant and the control algorithm tries to solve this specific task. For the cooperative method, a supervisor elaborates each elementary task as if it is alone and calculates an overall solution as a weighted sum of all the motion commands resulting from the elementary tasks. The supervisor can also dynamically change the relative importance of the tasks by changing the vector of weight gains.

4.4.4 NSB Applied

The theory in this section is based on [17].
For a task variable $\boldsymbol{\sigma} \in \mathbb{R}^m$ and a system configuration $\boldsymbol{x} \in \mathbb{R}^n$ we have the following relationship:

$$\boldsymbol{\sigma} = \boldsymbol{f}(\boldsymbol{x}) \tag{4.7}$$

And a corresponding differential relationship:

$$\dot{\boldsymbol{\sigma}} = \frac{\partial \boldsymbol{f}(\boldsymbol{x})}{\partial \boldsymbol{x}} \boldsymbol{v} = \boldsymbol{J}(\boldsymbol{x})\boldsymbol{v} \tag{4.8}$$

Here, $\boldsymbol{J} \in \mathbb{R}^{m \times n}$ is the configuration-dependent task Jacobian matrix and $\boldsymbol{v} \in \mathbb{R}^n$ is the system velocity.

The term system configuration refers to the drone's position or orientation. One widely used method to generate motion references $\boldsymbol{x}_d$ for the drone starting at desired values of the task function $\boldsymbol{\sigma_d}$ is to act at the differential level by inverting the (locally linear) mapping. A typical requirement is to pursue minimum-norm velocity, leading to the least-square solution:

$$\boldsymbol{v}_d = \boldsymbol{J}^{\dagger}\dot{\boldsymbol{\sigma}}_d \tag{4.9}$$

Where $\boldsymbol{J}^{\dagger}$ is defined as:

$$\boldsymbol{J}^{\dagger} := \boldsymbol{J}^T\left(\boldsymbol{J}\boldsymbol{J}^T\right)^{-1} \tag{4.10}$$

A reference position trajectory besides the velocity is now needed for the drone, which can be obtained by time integration of $\boldsymbol{v}_d$. The discrete time integration will however result in a numerical drift of the drone's reconstructed position. This drift can be counteracted by a closed loop inverse kinematics (CLIK) version of the algorithm [17].

$$\boldsymbol{v}_d = \boldsymbol{J}^{\dagger}\left(\dot{\boldsymbol{\sigma}}_d + \boldsymbol{\Lambda}\tilde{\boldsymbol{\sigma}}\right) \tag{4.11}$$

Here, $\boldsymbol{\Lambda}$ is a suitable constant positive definite matrix of gains and $\tilde{\boldsymbol{\sigma}}$ is the task error defined as $\tilde{\boldsymbol{\sigma}} = \boldsymbol{\sigma}_d - \boldsymbol{\sigma}$.

The velocity for each single task i then becomes:

$$\boldsymbol{v}_i = \boldsymbol{J}_i{}^{\dagger}\left(\dot{\boldsymbol{\sigma}}_{i\,d} + \boldsymbol{\Lambda}_i\tilde{\boldsymbol{\sigma}}_i\right) \tag{4.12}$$

The NSB control always fulfills the highest priority task, and the lower priority tasks are fulfilled only in a subspace where they do not conflict with the higher priority ones. This is an advantage compared to the competitive approach, where one single task can be achieved at once, and to the cooperative approach, where the linear combination of each task's output results in a set of partly fulfilled tasks.

4.4.5 Reach Goal with Obstacle Avoidance

For our drone to reach the desired position, the mission is decomposed in two tasks with the lowest number being the highest priority:

- Task 1: Obstacle avoidance
- Task 2: Reach goal

To ensure that the highest priority task of obstacle avoidance is fulfilled, it is of most importance to ensure integrity of the drone. The task aims to keep a safe distance

from an obstacle detected in the advancing direction of the drone. Hence, its output is a velocity ensuring that the drone keeps a safe distance from the obstacle when approaching. The task of obstacle avoidance (σ_1) can be described as:

$$\begin{aligned} \sigma_1 &= ||\boldsymbol{x} - \boldsymbol{x}_o|| \in \mathbb{R} \\ \sigma_{1d} &= d \\ \boldsymbol{J}_1 &= \hat{\boldsymbol{r}}^T \in \mathbb{R}^{1\times 2} \end{aligned} \tag{4.13}$$

Here, $\boldsymbol{x}_o$ is the obstacle's position, d is the distance to the goal and

$$\hat{\boldsymbol{r}} = \frac{\boldsymbol{x} - \boldsymbol{x_o}}{||\boldsymbol{x} - \boldsymbol{x_o}||} \tag{4.14}$$

is the unit vector aligned with the obstacle-to-drone direction. By equation (4.12), the primary task velocity is:

$$\boldsymbol{v}_1 = {\boldsymbol{J}_1}^{\dagger}\lambda_1\big(d - ||\boldsymbol{x} - \boldsymbol{x}_o||\big) \tag{4.15}$$

For the NSB approach, the obstacle avoidance also elaborates to the null-space direction. This null-space contribution is expressed as:

$$\boldsymbol{N}(\boldsymbol{J}_1) = \boldsymbol{I} - {\boldsymbol{J}_1}^{\dagger}\boldsymbol{J_1} = \boldsymbol{I} - \hat{\boldsymbol{r}}\hat{\boldsymbol{r}}^T \tag{4.16}$$

Where $\boldsymbol{I}$ is the identity matrix of the proper dimension. The obstacle avoidance task is only active when required. For instance when the drone is closer than a set threshold value to the obstacle and the output velocity of the lower priority tasks is in the direction of the obstacle.

The reach goal task ($\boldsymbol{\sigma_2}$) sets its output velocity proportional to the distance from the goal $\boldsymbol{x}_d$, and hence, the task is described as:

$$\begin{aligned} \boldsymbol{\sigma}_2 &= \boldsymbol{x} \in \mathbb{R}^2 \\ \boldsymbol{\sigma}_{2d} &= \boldsymbol{x}_d \in \mathbb{R}^2 \\ \boldsymbol{J}_2 &= \boldsymbol{I} \in \mathbb{R}^{2\times 2} \end{aligned} \tag{4.17}$$

By equation (4.12), the secondary task velocity is:

$$\boldsymbol{v}_2 = \boldsymbol{\Lambda}_2\big(\boldsymbol{x}_d - \boldsymbol{x}\big) \tag{4.18}$$

A saturation can be set for the resulting velocity in order to ensure limitations in the drone's input signals. An illustration of the task scheduler is shown in figure 4.3.

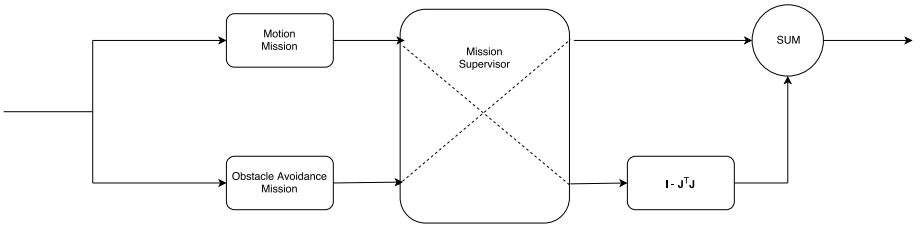


Figure 4.3: The NSB task scheduler

4.4.6 Task Activation

As mentioned in [4], there are practical drawbacks to the obstacle avoidance task. These drawbacks are discussed here and solved with task activation.

When the obstacle avoidance task has been active, it forces the drone to follow the threshold-circle all the way around. This conflicts with the second task as the obstacle avoidance task has the higher priority. This can be avoided by setting some rules for task activation and apply these rules to the scheduler. The obstacle avoidance task doesn't need to be active unless a collision is about to happen. By checking if the current velocity vector interfere with the obstacle, we can reschedule the tasks. By calculating the angle between the velocity vector and the obstacle center, as shown in figure 4.4, we can calculate d_o. The obstacle avoidance task is only active if:

$$\hat{d}_o < d_o \qquad \text{and} \qquad |\theta| \in [0, \frac{\pi}{2}) \tag{4.19}$$

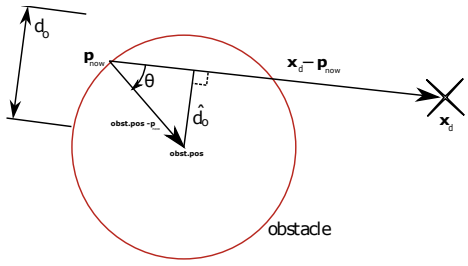


Figure 4.4: The NSB angle problem, adapted from [3]

4.5 High Level Control for Local Maps: Reactive Logic

Unlike for the global maps, local maps are continuously updated from the sensor readings. We need to evaluate the drone's states and how it will interact with the sensed environment at each sensor scan. An intuitive logic for designing a high level controller on such restricted information is to use a sense-act logic. This section describes a self-developed reactive logic for collision avoidance based on the radar-readings described in section 4.2.

4.5.1 Self-developed Reactive Logic

Consider the drone with radar sectors as illustrated in figure 4.1. The drone's heading will always face such that sector 1 and 6 will face towards the desired path. For an obstacle in sector 1, a rotation of the velocity vector of $45°$ in the clockwise direction will be a way of steering the drone away from the obstacle. The same way goes for an obstacle in sector 6, but now we rotate the velocity vector counter-clockwise.
For sector 2 and 5, a rotation of $90°$ can be done in the same way as for sector 1 and 6, whereas sector 3 and 4 faces towards the back of the drone, and are only important after the drone has surpassed the obstacles to keep track of the drone's surroundings. Hence, no velocity vector rotation is done for obstacles detected in sector 3 and 4. After each radar reading, the velocity vector is multiplied by the rotation matrix $\boldsymbol{R}(sector)$. The newly rotated vectors, based on which sector the obstacle is detected are displayed in figure 4.5. The reactive logic described above can be composed as an algorithm,

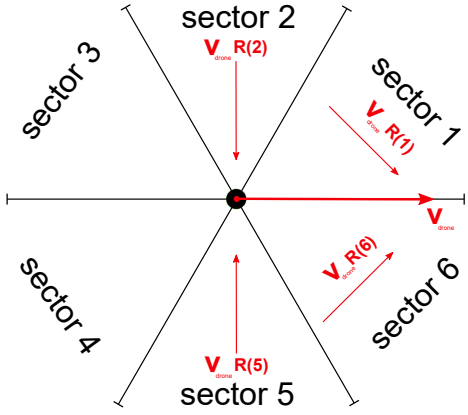


Figure 4.5: Rotated vectors based on sectors

represented in algorithm 1. This algorithm only consider the delicate situation where

Algorithm 1 Reactive algorithm for one obstacle

```
1: function REACTIVE(v_ref, sensor_scan)
2:     for all sectors return sensor_scan do
3:         if sector then
4:             v_new ← v_ref · R(sector)          ▷ Rotates the velocity vector
5:         end if
6:     end for
7:     return v_new                                ▷ New velocity is returned
8: end function
```

only one obstacle is sensed. This is not always the reality, and in order to make a more robust reactive logic, a new algorithm that considers multiple obstacles can be developed.

4.5.2 Extended Reactive Logic for Multiple Obstacles

If algorithm 1 is used and multiple sectors sense obstacles, the algorithm only chooses to rotate the velocity vector based on the highest numbered *active* sector. A sector being active, is a sector where obstacles are detected. In order to get a contribution from all active sectors, a modified algorithm is used. This algorithm considers how many obstacles that are sensed and gives one supplement from each to the total resulting velocity vector and is described in algorithm 2.

Algorithm 2 Reactive algorithm for multiple obstacles

```
1: function REACTIVE(v_ref, sensor_scan)
2:     for all sectors in sensor_scan do
3:         if sector then
4:             v_sector ← v_ref · R(sector)       ▷ Rotates the velocity vector
5:         end if
6:         v_middle ← v_middle + v_sector         ▷ Sum up all velocity contributions
7:         α ← α + 1                              ▷ Obstacle counter
8:     end for
9:     v_new ← v_middle / α                       ▷ Divide by number of obstacles
10:    return v_new                               ▷ New velocity is returned
11: end function
```

4.6 High Level Control for Local Maps: VO

The velocity obstacle algorithm demonstrated in this thesis is based on the one described in [5]. This method considers both static and dynamic obstacles, whereas this thesis only concerns the matter of static obstacles, sensed by the radars. The concept of the velocity obstacle method is quite intuitive. It computes avoidance maneuvers based on the current position and velocity of the drone and the obstacles. This is a first-order method since it yields positions as a function of time without integrating the velocities. It is chosen to be investigated for our application as it is a robust way of ensuring collision avoidance and a well-explored method.

4.6.1 Assumptions

For simplicity, we assume the drone to be a circle and the obstacles to be cylindrical. Thus, this analysis considers a planar problem with no rotations. Since the walls can be represented as a series of cylindrical objects, this is not a severe limitation. We also assume that all obstacles are static and that their position is measurable from the sensors.
Consider now a planar problem with the drone represented as a circular object A with the respective velocity $\boldsymbol{v}_A$, and a circular obstacle B that is placed in the nearby area of the drone as illustrated in figure 4.6. In order to compute the Velocity Obstacle, the obstacle B is mapped to the configuration space of the drone A. This is done by reducing the circular drone A to a point $\hat{A}$ and enlarging the radius of B by the radius of A, resulting in a new configured obstacle $\hat{B}$. Each object is now represented with a position and velocity vector attached in the center.

4.6.2 Collision Cone

The collision cone is defined as the set of colliding relative velocities between $\hat{A}$ and $\hat{B}$ and is denoted as:

$$CC_{A,B} = \{\boldsymbol{v}_{A,B} | \lambda_{A,B} \cap \hat{B} \neq \emptyset\} \tag{4.20}$$

Where $\boldsymbol{v}_{A,B} = \boldsymbol{v}_A - \boldsymbol{v}_B$ is the relative velocity of $\hat{A}$ with respect to $\hat{B}$, and $\lambda_{A,B}$ is the line of $\boldsymbol{v}_{A,B}$. Note that for static obstacles $\boldsymbol{v}_B = 0$ and hence, $\boldsymbol{v}_{A,B} = \boldsymbol{v}_A$. The cone $CC_{A,B}$ represents the planar sector with apex in $\hat{A}$ which restricted velocities are in. The sector is bounded by the two tangents λ_f and λ_r from $\hat{A}$ to $\hat{B}$ as illustrated in figure 4.7 with the cyan colored area representing the CC. Hence, if the velocities within the tangent lines is chosen, collisions will occur. Since we now know which velocities not to chose, a choice of a velocity vector outside $CC_{A,B}$ will guarantee a

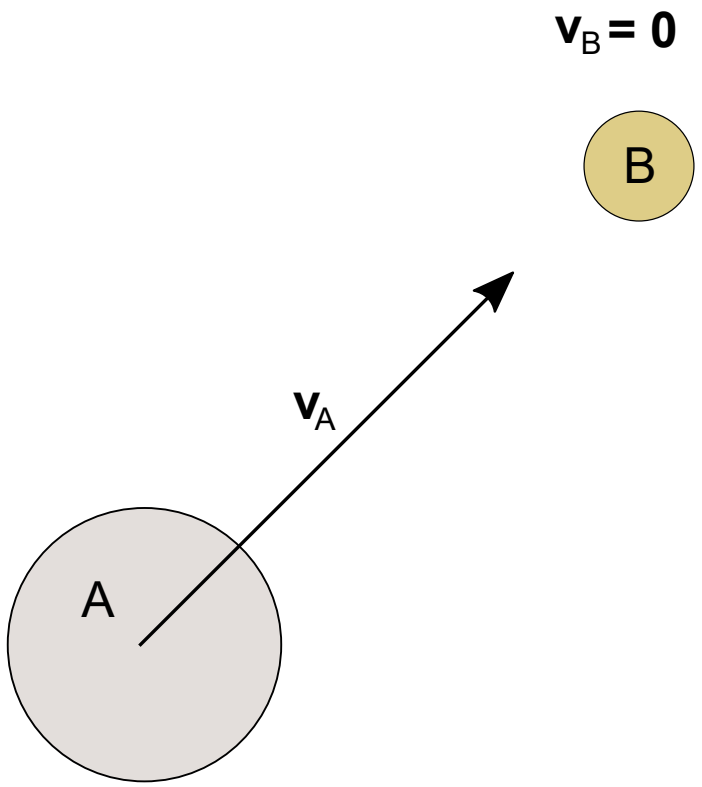


Figure 4.6: Drone and obstacle illustrated as circles

collision free trajectory.
An equivalent condition on the absolute velocities of A is established by simply adding $\boldsymbol{v}_B$ to each velocity in $CC_{A,B}$. This is equivalent to translating CC by $\boldsymbol{v}_B$. The Velocity Obstacle hence becomes:

$$VO = CC_{A,B} \oplus \boldsymbol{v}_B \tag{4.21}$$

Where $\oplus$ is the Minkowski vector sum operator (see appendix A.2). The velocity obstacle in equation (4.21) partitions the absolute velocities of A into avoiding and colliding velocities. By selecting a $\boldsymbol{v}_A$ outside the VO will result in collision avoidance with B, described mathematically by:

$$A \cap B = \emptyset \qquad \text{if} \qquad \boldsymbol{v}_A \notin VO \tag{4.22}$$

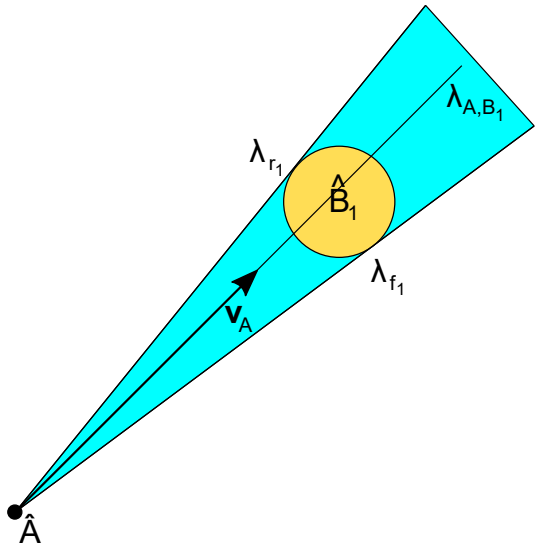


Figure 4.7: Collision Cone illustration

Any velocity on the boundary of VO will result in A and B grazing. Note that for our case of static obstacles, all obstacle velocities will be zero ($\boldsymbol{v}_B = 0$), and hence, the relative velocity cone will always be the velocity obstacle and equation (4.21) is reduced to:

$$VO = CC_{A,B} \tag{4.23}$$

The collision cone described considers only one obstacle with respect o the drone, and the next section will explain how we easily can extend the cone for consideration of more obstacles.

4.6.3 Multiple Obstacles

In order to avoid multiple obstacles, we simply consider the VO as the union of each obstacle's VO:

$$VO = \cup_{i=1}^{n} VO_{b_i} \tag{4.24}$$

Where n is the number of obstacles sensed. The avoidance velocities are now the $\boldsymbol{v}_A$s outside the VO as shown in figure 4.8

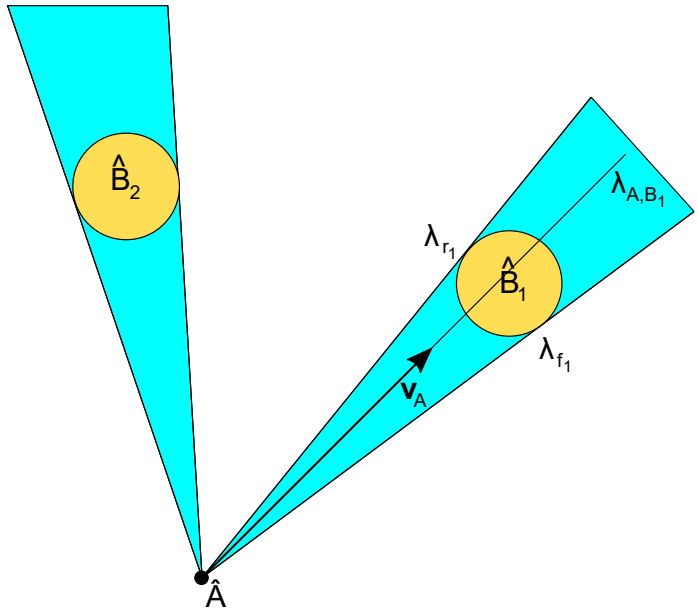


Figure 4.8: Velocity Obstacle with multiple obstacles illustration

4.6.4 Imminent Collisions

In the case of a cluttered environment, one might prioritize the closer obstacles over the ones with longer time to collision. Since our sensor has a certain sense range, this is partly done by the sensing constraints, but can also be modified by classifying *imminent* collisions. A collision is imminent if the collision happens some time $t < T_h$, where T_h is a suitable time horizon [5]. A modified VO can be constructed by subtracting VO_h from the current VO, where VO_h is defined as:

$$VO_h = \{\boldsymbol{v}_A | \boldsymbol{v}_A \in VO, \qquad ||\boldsymbol{v}_{A,B}|| \leq \frac{d_m}{T_h}\} \tag{4.25}$$

Where d_m is the shortest relative distance between the drone and the obstacle. VO_h is the set of velocities that would cause a collision occurring beyond the time horizon T_h.

4.6.5 Avoidance Maneuver

The avoidance maneuver consists of a one-step change in the velocity in order to avoid a future collision within a given time horizon [5]. The set of reachable velocities is constrained by the drone's dynamics. Given a time interval Δt, the velocities reachable by the drone A at a given state are found by mapping the actuator constraints to the acceleration constraints. A set of feasible accelerations at time t is defined as:

$$FA(t) = \{\ddot{\boldsymbol{x}} | \ddot{\boldsymbol{x}} = f(\boldsymbol{x}, \dot{\boldsymbol{x}}, \boldsymbol{u}), \boldsymbol{u} \in U\} \tag{4.26}$$

Where $\boldsymbol{x}$ is the position vector of the drone, $f(\boldsymbol{x}, \dot{\boldsymbol{x}}, \boldsymbol{u})$ is the dynamics of the drone, $\boldsymbol{u}$ is the actuation efforts, and U is the admissible controls.
By using the feasible accelerations, we can now find the reachable velocities for the time interval to come $(t + \Delta t)$, this defined as:

$$RV(t + \Delta t) = \{\boldsymbol{v} | \boldsymbol{v} = \boldsymbol{v}_A(t) \oplus \Delta t \cdot FA(t)\} \tag{4.27}$$

The velocities we are interested in in order to avoid collisions is the reachable avoidance velocities (RAV), these can be found by the difference between the set of reachable velocities, and the velocity obstacle set:

$$RAV(t + \Delta t) = RV(t + \Delta t) \ominus VO(t) \tag{4.28}$$

Where $\ominus$ is the Minkowski difference (see appendix A.2). We can now choose any velocity in the RAV set in order to avoid collisions. In the case of multiple obstacles, the RAV set consists of multiple disjoint closed subsets.

The algorithms described in this chapter are now to be implemented on the self-developed MATLAB-simulator. The results obtained are presented in the next chapter.

Chapter 5

Simulation Results

By implementing the algorithms described in chapter 4, the yielded results obtained from the different simulations executed in the MATLAB simulator are presented here.

5.1 Representation of the Results

The findings in this thesis are obtained from the created simulator described in chapter 3. We mainly look at the trajectories of the quadcopter operating within the inspection area, which are generated in 3D. An example of the 3D illustration is shown figure 5.1. Although 3D simulations are obtained, the drone's change of behaviour mainly affects the xy-plane. Hence, most of the illustrations are displayed from above as this presents the findings in a better way. Table 5.1 explain the appearance of the simulated objects and trajectories.

Object	Explanation
Drone	Cross-frame and 4 circular rotors
Container	Black edges
Obstacles	Cyan colored cylinders with magenta edges
Actual trajectory	Red
Desired trajectory	Blue

Table 5.1: Table of object appearance in the simulation display

In addition to the simulation trajectory display, two additional figures cover the recorded position and velocities of the drone at the given time t for the respective simulation. An example of the position and velocity plot is illustrated in figure 5.2b and 5.2c respectively. The blue line illustrates the desired state, whereas the red line illustrates the actual state. Note that the velocities are saturated at $3m/s$ as this is the set velocity restriction for all directions in this thesis.

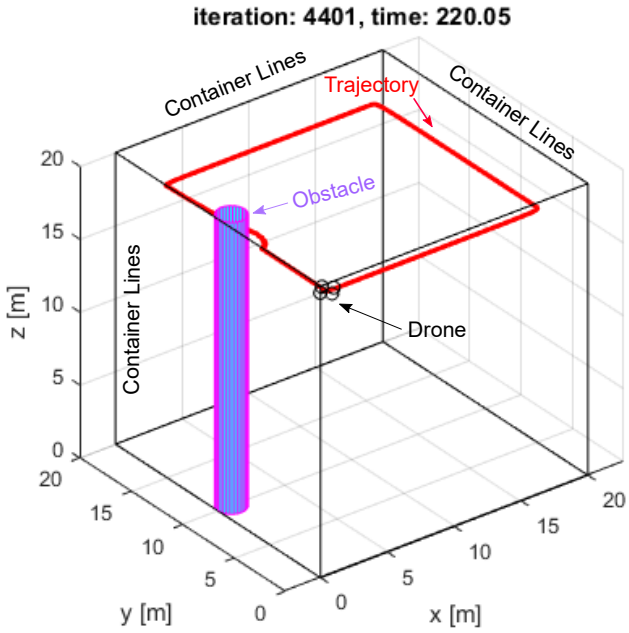


Figure 5.1: Illustrative figure for the simulator display

5.2 PID-control of a Step Trajectory

For the tuning of PID-controller, step trajectories for each of the x,y and z direction were tested. After the execution of several simulations of step trajectories, the tuning

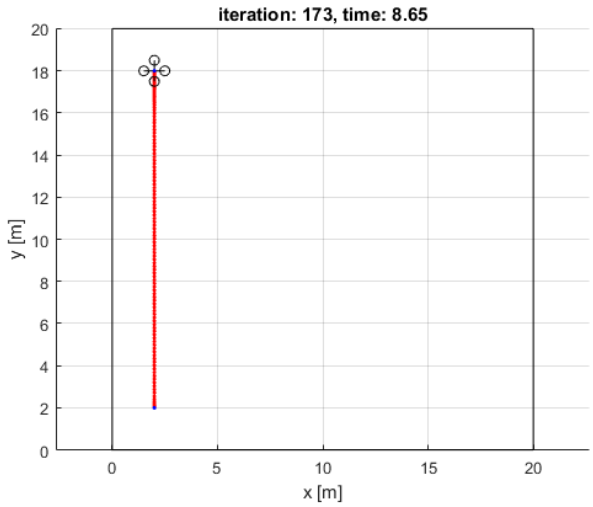
constants for the position PD-controller obtained for critical damping became:

$$\begin{aligned} K_p &= \begin{bmatrix} K_{p_x} \\ K_{p_y} \\ K_{p_z} \end{bmatrix} = \begin{bmatrix} 15 \\ 15 \\ 30 \end{bmatrix} \\ K_d &= \begin{bmatrix} K_{d_x} \\ K_{d_y} \\ K_{d_z} \end{bmatrix} = \begin{bmatrix} 17 \\ 12 \\ 10 \end{bmatrix} \end{aligned} \tag{5.1}$$

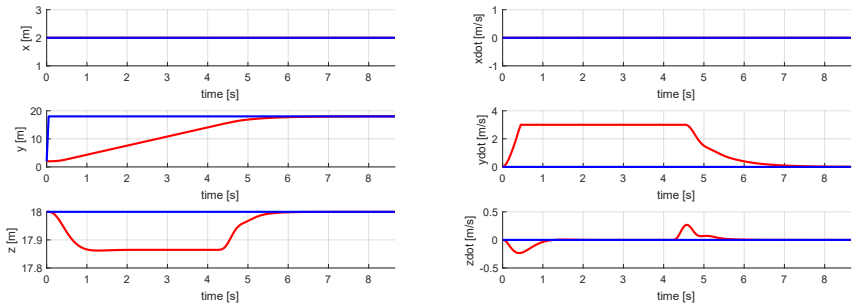
Whereas the tuned attitude PD-controller yielded the constants:

$$\begin{aligned} K_{p_m} &= \begin{bmatrix} 3000 \\ 3000 \\ 3000 \end{bmatrix} \\ K_{d_m} &= \begin{bmatrix} 300 \\ 300 \\ 300 \end{bmatrix} \end{aligned} \tag{5.2}$$

Note that we do not tune the integration parameters as the integration part of the PID-controller is done with MATLAB's ode45 solver. A simulation with the tuning gains from equations (5.1)-(5.2) yields the response illustrated in figure 5.2. Figure 5.2a illustrate the drone's trajectory from $\boldsymbol{p}_{start} = \begin{bmatrix} 2 & 2 & 18 \end{bmatrix}^T$ to $\boldsymbol{p}_{end} = \begin{bmatrix} 2 & 18 & 18 \end{bmatrix}^T$ displayed in the xy-plane. The simulated time it takes to reach the end position is $t_{sim} = 8.65s$ and by looking at figure 5.2b, we clearly see a critically damped response for the y-position as well as a saturated y-velocity response, seen in figure 5.2c.



(a) The trajectory of a PID-controller of a y-step



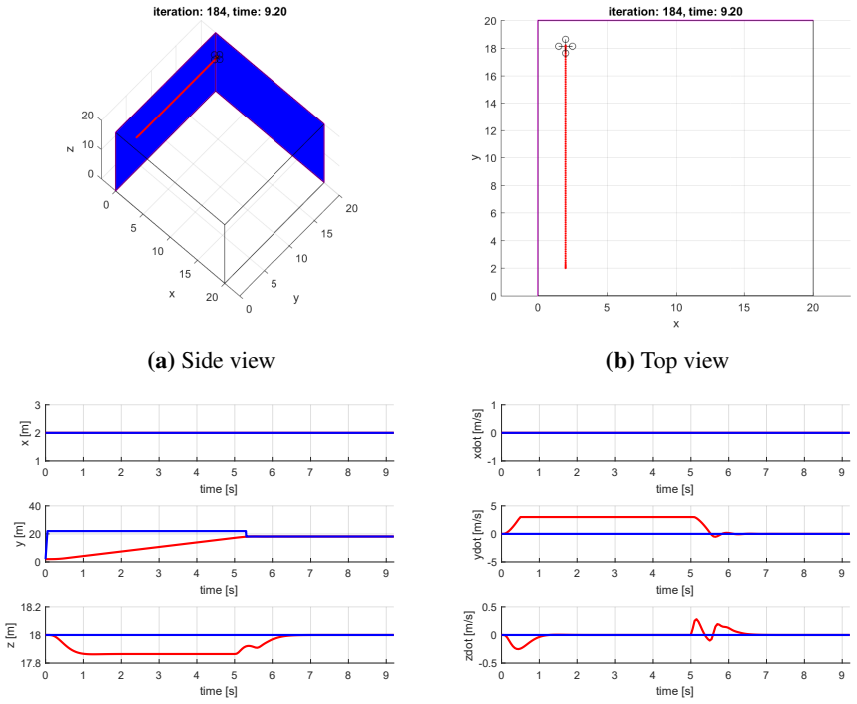
(b) The positions of the PID-control of a y-step **(c)** The velocities of the PID-control of a y-step

Figure 5.2: Simulator display for the PID-control of a y-step

5.2.1 Wall Avoidance

In order to assure that the quadcopter doesn't fly into the container walls, a safety constraint of $d_{safe} = 2$ meters is set. If a pilot pushes the drone towards a destination beyond the walls or tries to violate the set safety constraint, the desired end position $\boldsymbol{p}_{end}$ is changed. The scenario is illustrated in figure 5.3 below with the active walls

colored in blue.



(a) Side view

(b) Top view

(c) Position

(d) Velocity

Figure 5.3: Override of pilot command for wall avoidance

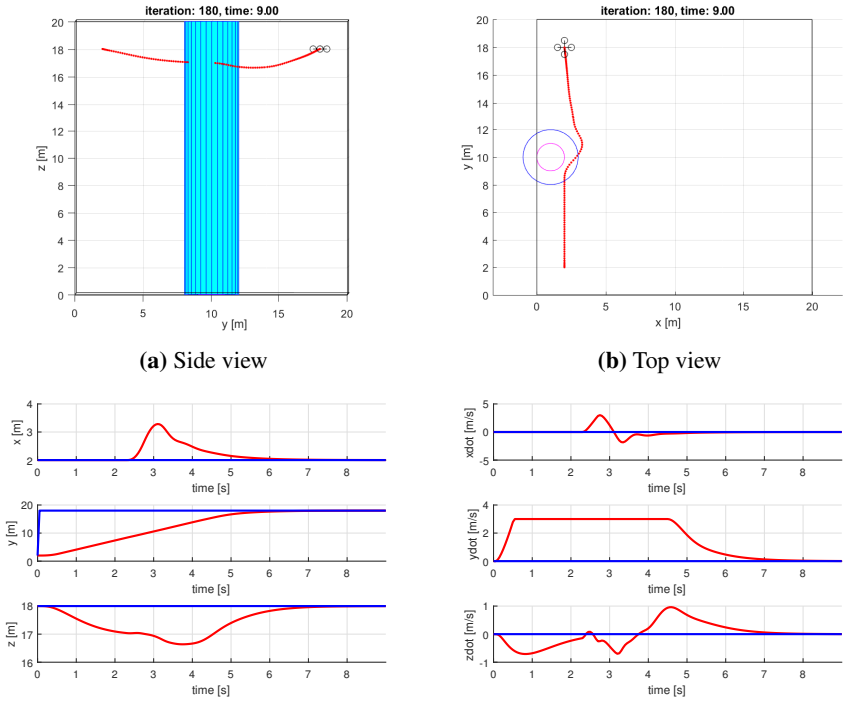
The behaviour is as desired when the pilot pushes the drone towards the wall, and this ensures that collisions with the container walls are avoided in a simulation time of $t_{sim} = 9.20s$. Note the change of desired y-position around $t = 5.3s$ To make sure the drone also considers other static obstacles that might occur in the inspection area, high level control needs to be implemented. This is done both globally and locally in the following sections.

5.3 Global Approach for Step Trajectory (NSB)

For the simplest case of obstacle avoidance, we simulate the same trajectory inside the tank we want to inspect. There is an obstacle in the drone's planned trajectory and the NSB-algorithm described in section 4.4 is used to avoid the obstacle. The assumptions of the simulated flight are as follows:

- The obstacle's location is known (global algorithm)
- The drone's initial states are set to be at a hovering state at $\boldsymbol{p}_{start} = \begin{bmatrix} 2 & 2 & 18 \end{bmatrix}^T$
- The drone's final and desired position is set to be at $\boldsymbol{p}_{end} = \begin{bmatrix} 2 & 18 & 18 \end{bmatrix}^T$
- An obstacle of radius $R_{obst} = 1$ is centered at $\boldsymbol{p}_{obst} = \begin{bmatrix} 1 & 10 & z \end{bmatrix}^T$

From the simulation results of the NSB-algorithm without task activation and the tuning parameters set to $\lambda_1 = 6$, $\boldsymbol{\Lambda}_2 = 2 \cdot \boldsymbol{I}_{3\times 3}$, a simulation time of $t_{sim} = 9.00s$ and the results illustrated in figure 5.4 are obtained:



(a) Side view

(b) Top view

(c) Position

(d) Velocity

Figure 5.4: The NSB algorithm for a step response without tuning and task activation

Note the threshold of where the scheduler switches between tasks, illustrated as the obstacle's blue outer circle and set to $R_{thresh} = 2$. We can clearly see that the quadcopter dynamics interfere with the response of the collision avoidance. The deacceleration of the already built up speed starts when the obstacle avoidance task is activated and the wished behaviour of proximity within the threshold circle is not obtained. One also see the lack of angle task activation, as the algorithm doesn't release the obstacle avoidance task once the obstacle has been avoided and the path towards the goal is collision free. This as described in section 4.4.6. Hence, a tuning of the algorithm is needed for optimal behaviour as well as the implementation of the angle task activation.

5.3.1 NSB with Appropriate Tuning

As discussed in section 4.4, we can tune the NSB behaviour by adjusting the lambdas. An increase in λ_1 will result in a greater contribution to the obstacle avoidance as well as a greater $\mathbf{\Lambda}_2$ will result in a contribution increase to the reach goal task, and vice-versa. First, by increasing λ_1 drastically to 80, we see that the trajectory changes, as illustrated in figure 5.5. This is expected as the focus of collision avoidance is now overly focused, and this directly affects the velocity response. As we want to optimally avoid the obstacle and proximity keeping within the threshold, this is not a desirable result.

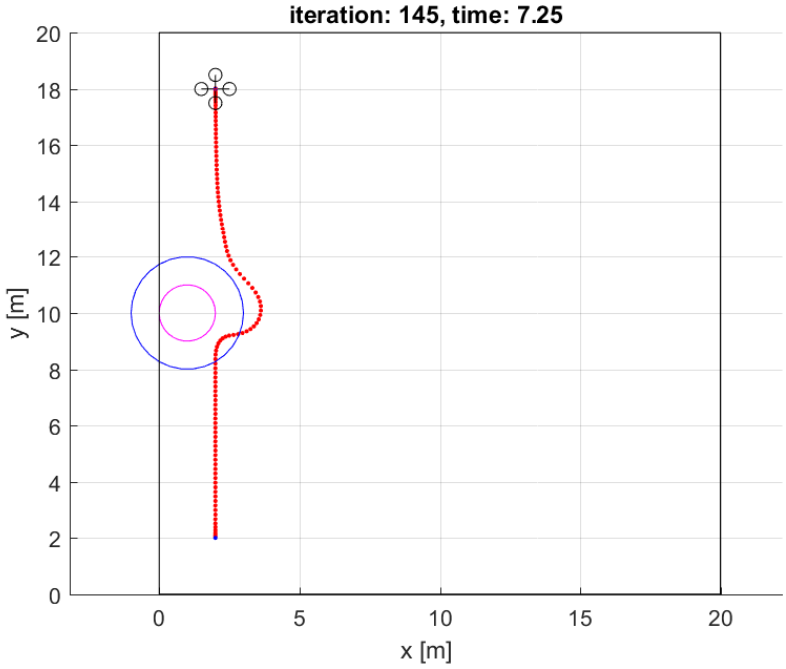


Figure 5.5: The NSB algorithm for a step response with increased $\lambda_1 = 80$

By tuning the lambdas to $\lambda_1 = 6$ and $\mathbf{\Lambda}_2 = 0.2 \cdot \boldsymbol{I}_{3\times3}$, we can see that we get a much more desirable trajectory where the drone keeps its path within the threshold as seen in figure 5.6. However, the drone gets stuck when traversed around the obstacle and doesn't release its obstacle avoidance task when its path towards the goal is clear.

This ”lock” is due to the null-space contribution being zero. In order to avoid this lock, we implement the angle task activation. The oscillations that occur when moving around the obstacle is caused by the quadcopter's dynamics as the algorithm switches between the two tasks. Note that the radius of the obstacle and threshold is increased to $R_{obst} = 2$ and $R_{thresh} = 3$ respectively. This is done for more detailed illustrations.

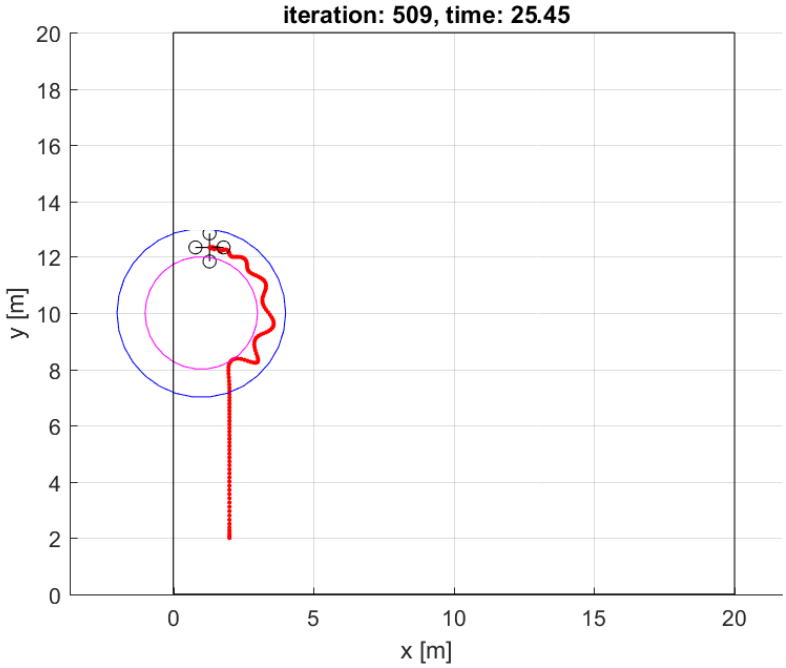


Figure 5.6: The NSB algorithm for a step response with tuning

5.3.2 NSB with Angle Task Activation

After implementing the angle task activation described in section 4.4.6 with the same tuning parameters as before, a collision-free step trajectory without ”lock” is obtained with the simulation time of $t_{sim} = 36.65s$.

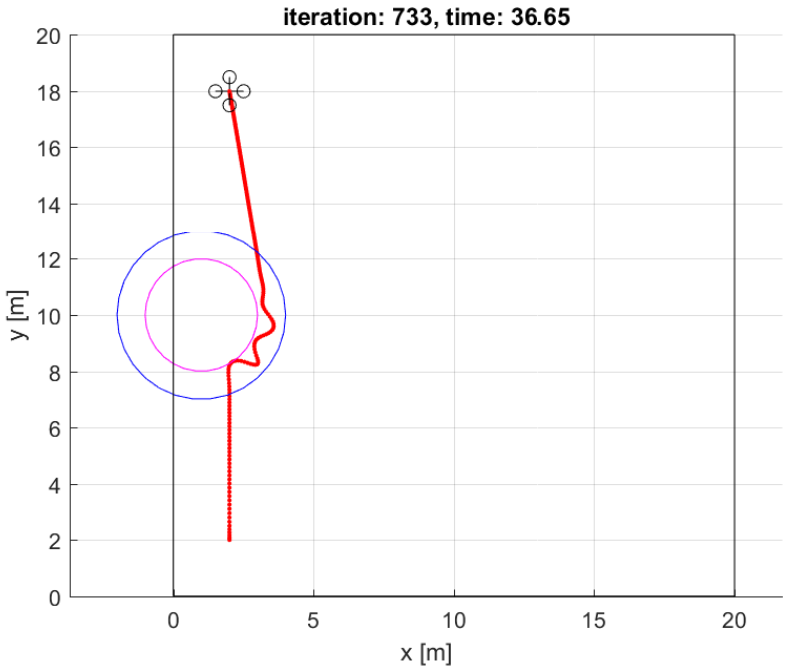
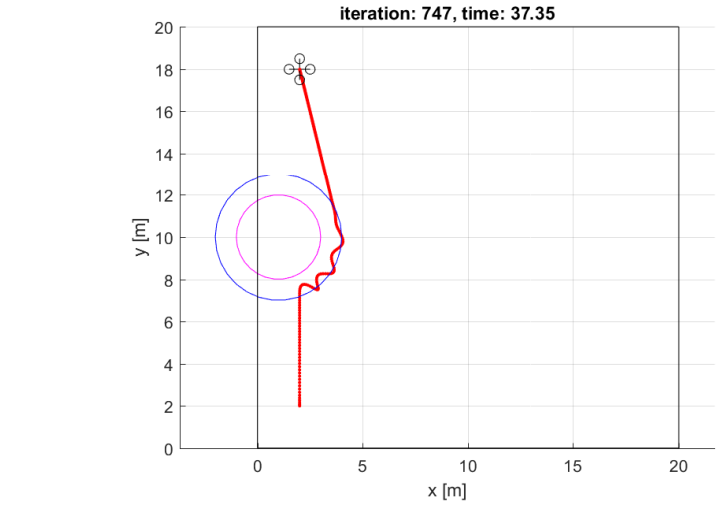
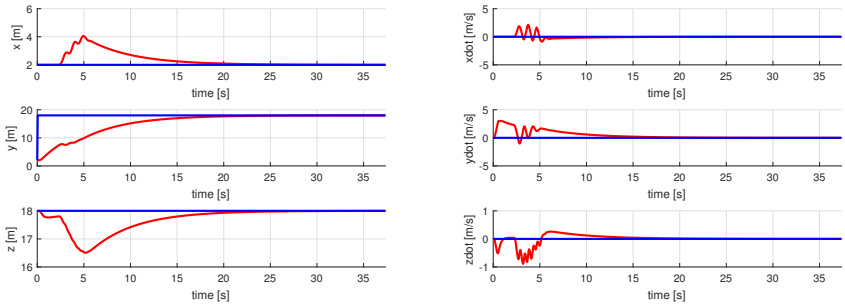


Figure 5.7: The NSB algorithm for a step response with tuning and task activation

As illustrated in figure 5.7, the dynamic slowness of the drone threatens the drone's safety and might cause collision with the obstacle. Hence, an increased threshold is set to $R_{thresh} = 4$ for safety reasons.



(a) Top view



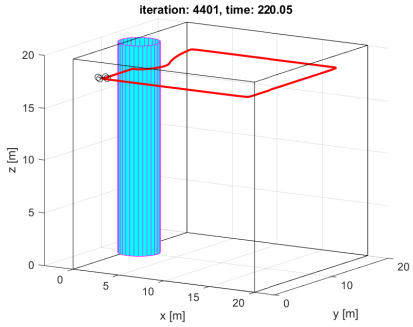
(b) Position **(c)** Velocity

Figure 5.8: NSB for a step response with tuning, task activation and extended threshold

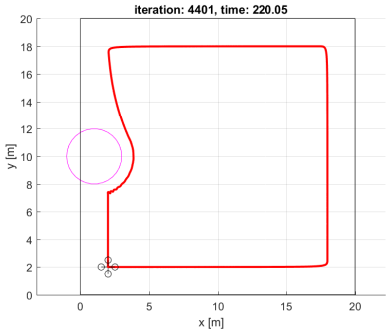
As illustrated in figure 5.8a, we now obtain a safe collision free trajectory within the threshold. Note that the dynamics cause oscillations when the tasks are switching. Another remark is that the simulation time is $t_{sim} = 37.35s$, which is drastically longer than for the original trajectory without obstacle consideration.

5.3.3 NSB for Wall Following

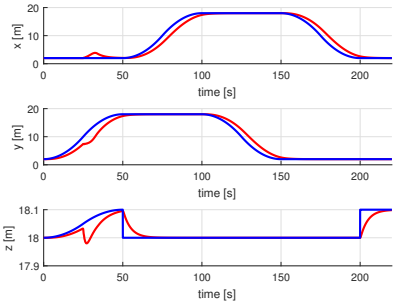
In order to follow the walls of the container, we plan a path consisting of multiple step-like trajectories. The drone's initial position is set to the top of the container at $\boldsymbol{p}_{start} = \begin{bmatrix} 2 & 2 & 18 \end{bmatrix}^T$ and is to traverse four step trajectories with the container corners as waypoints before it reaches its start position again. This as illustrated in figure 5.9.
An extension for takeoff and landing can be done by staring the drone on the ground at $\boldsymbol{p}_{start} = \begin{bmatrix} 2 & 2 & 0 \end{bmatrix}^T$, fly upwards, execute the square-trajectory before it descends to its start position again, as illustrated in figure 5.10.



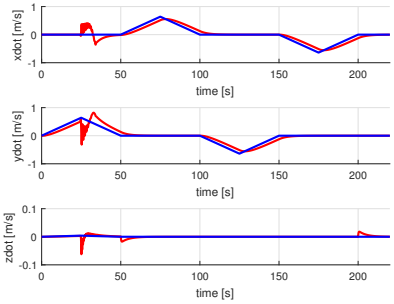
(a) Side view



(b) Top view

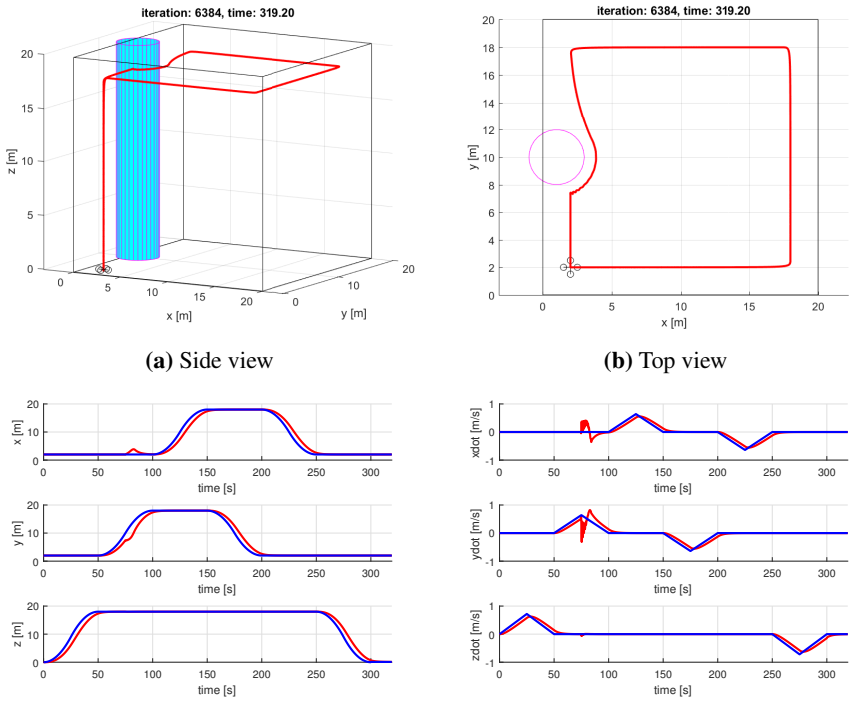


(c) Position



(d) Velocity

Figure 5.9: NSB wall follow upper square



(a) Side view

(b) Top view

(c) Position

(d) Velocity

Figure 5.10: NSB wall follow with takeoff and landing

We notice that both simulations for wall following successfully complete the inspection mission. The obstacle is avoided with a safe distance and the desired inspection path is followed. Note the simulation times of $t_{sim} = 220.05$ for the upper square trajectory and $t_{sim} = 319.20$ for the extended trajectory with takeoff and landing. They are much longer than the multiplicity of step trajectories they execute but also yield a less oscillating collision avoidance.

5.4 Local Approaches

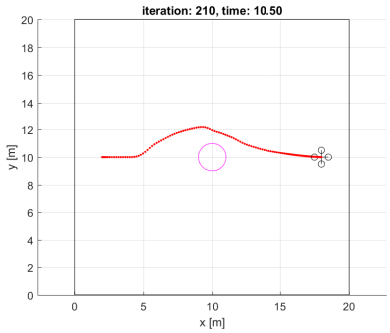
Up until now, the simulations executed have assumed the current environment to be known. For local approaches, this assumption doesn't hold and we need to obtain information about the environment from the sensor data as well as the drone's initial conditions. The local approaches described in chapter 4 are tested in this section. Simulations of the self-developed reactive logic are executed first and the Velocity Obstacle simulations are done afterwards.

5.4.1 Self-developed Reactive Logic

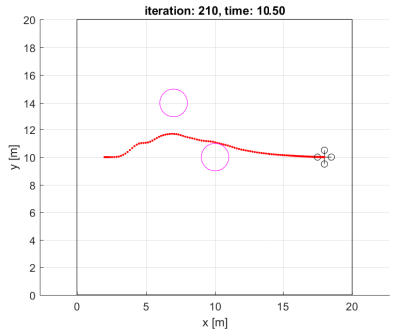
The reactive logic described in section 4.5.1 is to be tested on our simulator. Firstly, algorithm 1 is implemented for the step trajectory with an obstacle occurring in the middle of its originally planned path. As seen in figure 5.11a, the algorithm successfully navigates the drone in a collision-free path with a simulation time of $t_{sim} = 10.50s$.
For the consideration of two obstacles, algorithm 2 is implemented for collision avoidance. However, the trajectory response is a bit modest as illustrated in figure 5.11b. We see that the compromised velocity ensures that both obstacles are considered, but it doesn't prioritize the relative distances. This threatens the drone's safety when avoiding collisions. From figure 5.11c, we see that only sector 6 is considered and this pushes the velocity vector $45°$ counter-clockwise. For figure 5.11d however, the compromised velocity vector based on the sensed obstacles in sector 2 and 6 will actually be rotated $45°$ clockwise, which is directed towards the obstacle relative to the drone's current position. This is due to the summation of a $90°$ clockwise rotation from the active sector 2, and a $45°$ counter-clockwise rotation from the active sector 6 and is a highly undesirable behaviour that needs to be solved for a robust collision avoidance system.

5.4.2 Velocity Obstacle

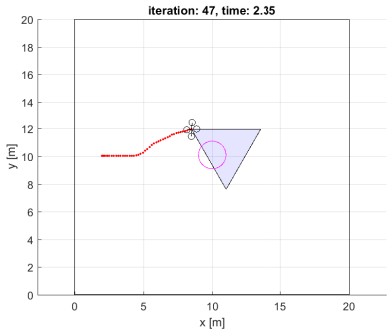
Unlike the self-developed reactive algorithm, the velocity obstacle (VO) algorithm only considers velocity vectors that guarantees collision-free navigation.



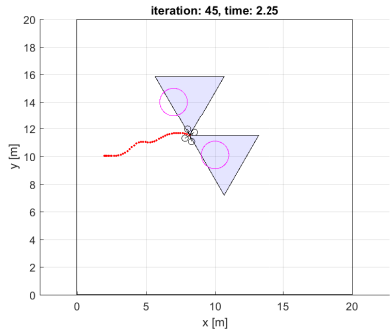
(a) One obstacle



(b) Two obstacles



(c) Sector activation for one obstacle

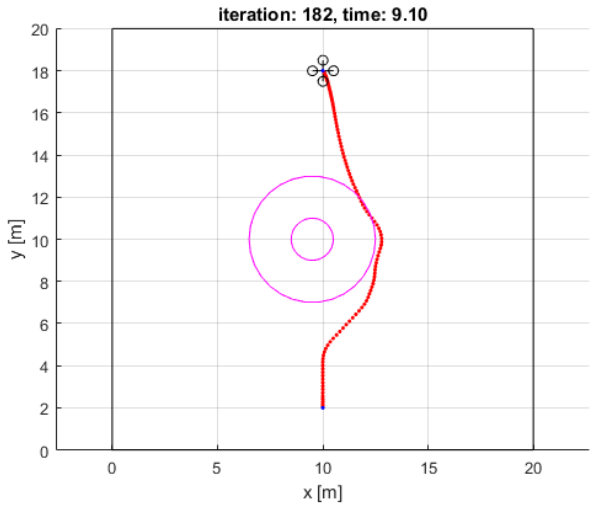


(d) Two sectors activated

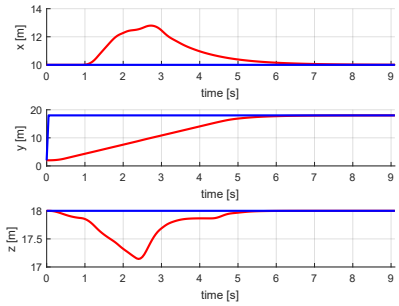
Figure 5.11: Reactive algorithm

One Obstacle for VO

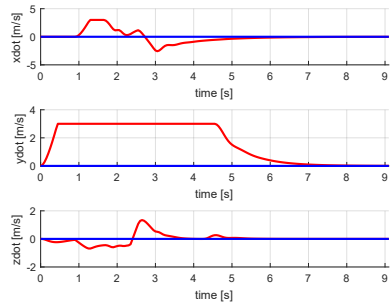
The results obtained for one obstacle are illustrated in figure 5.12. The simulation yields a simulation time of $t_{sim} = 9.10s$ and a decent safety margin determined by the threshold.



(a) Top view



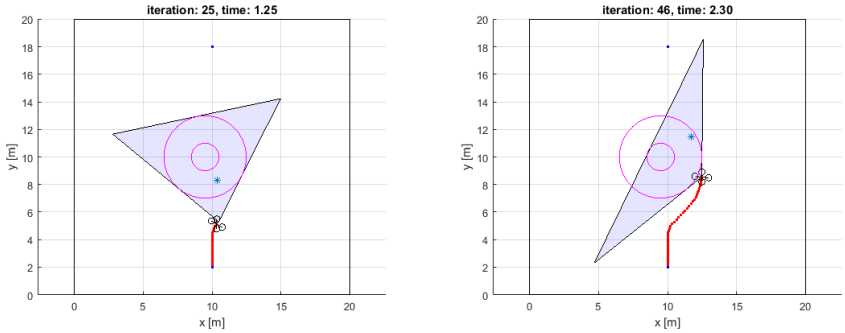
(b) Position



(c) Velocity

Figure 5.12: The VO algorithm for one obstacle

To further explain the results, the VO cone that is created when obstacles are sensed is illustrated. As explained in section 4.6, the velocity obstacle algorithm creates a cone of restricted velocities when obstacles are within the sensor range, this cone is illustrated at different time steps in figure 5.13.



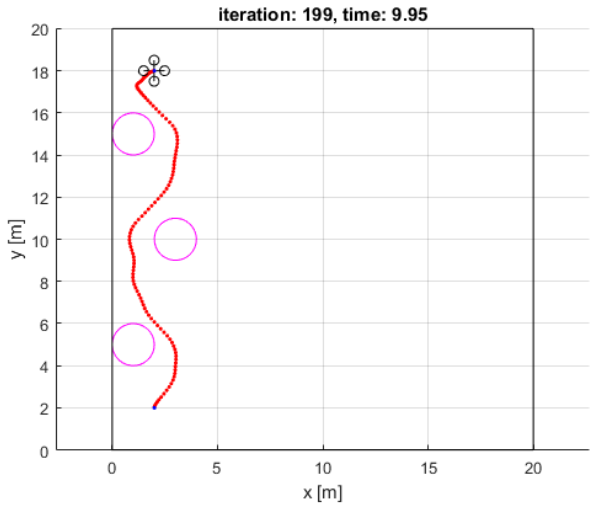
(a) The VO cone at $t = 1.25$ **(b)** The VO cone at $t = 2.30$

Figure 5.13: The VO cones at different simulation times

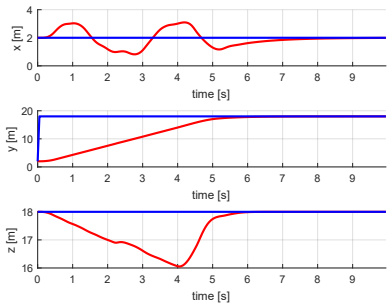
The transparent blue cone illustrates the set of restricted velocities, whereas the blue star illustrates the drone's desired position for the next time step $t + \Delta t$. A velocity vector outside this cone is considered at each time step, and the vector that points the most towards the drone's desired position will be selected.

Multiple Obstacles for VO

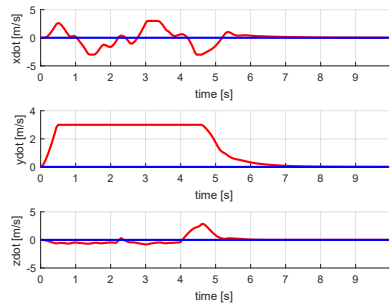
In the case of multiple obstacles, the velocity obstacle algorithm still ensures collision free trajectories, as illustrated in figure 5.14a. The simulation of collision avoidance with three obstacles yields a simulation time of $t_{sim} = 9.95s$. One interesting behaviour is the x-movement towards the end of the path. Instead of going directly to the goal after the last obstacle is avoided, it slightly overshoots towards left. This clearly is due to the drone's dynamics as the x-velocity drastically changes at $t_{sim} = 4.60s$, illustrated in figure 5.14c.



(a) Top view



(b) Position

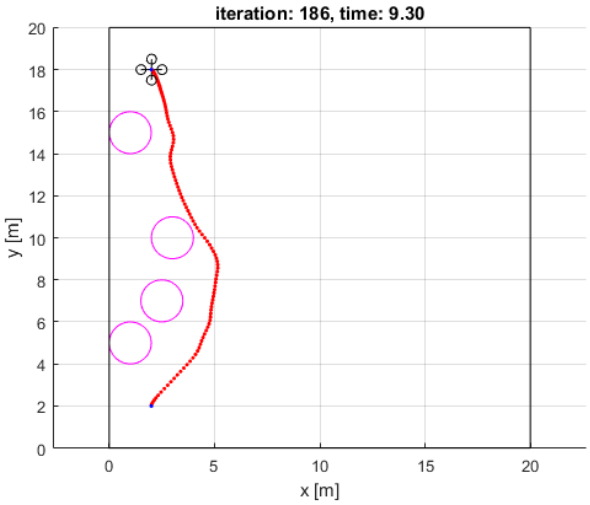


(c) Velocity

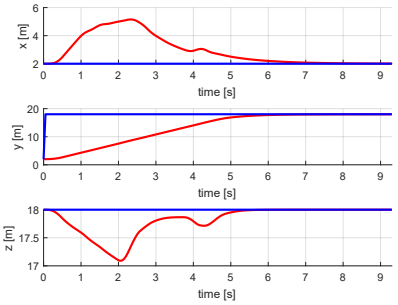
Figure 5.14: The VO algorithm with 3 obstacles

By intentionally pushing the drone further away from the desired trajectory by introducing yet another obstacle, we see that the drone's safety margin gets threatened when surpassing the third obstacle. The simulation of collision avoidance with four obstacles yields a simulation time of $t_{sim} = 9.30s$, which is faster than for the simulation with three obstacles. The reason for the close encounter with the third obstacle, illustrated in figure 5.15a, is due to the selected velocity towards the goal not being in the VO. This

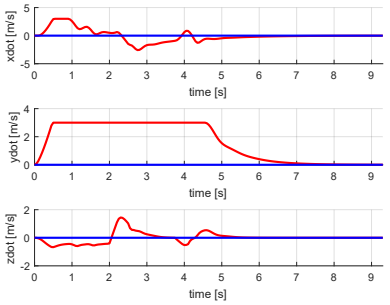
causes the drone to interfere with the obstacle's threshold margin and is undesirable.



(a) Top view



(b) Position



(c) Velocity

Figure 5.15: The VO algorithm with 4 obstacles

This chapter has presented the simulations for the different algorithms of collision avoidance. Different behaviours and simulation times have been noted for the further investigation of the algorithms. Next chapter will discuss the findings of the executed simulations.

Chapter 6

Discussion

The remarks noted from the obtained results in chapter 5 are discussed in this chapter. Suggestions for optimality and robustness are mentioned. Table 6.1 tells each approach's configuration and simulation time.

Method	**Obstacles**	**Time**$[s]$	**Tuned**	**Trajectory**	**Comment**
PID	None	8.65	Yes	Step	Smooth Trajectory
PID	Wall	9.20	Yes	Step	Pilot Override
NSB	1	9.00	No	Step	$\lambda_1 = 6$, $\mathbf{\Lambda}_2 = 2 \cdot \boldsymbol{I}_{3\times3}$
NSB	1	7.25	No	Step	$\lambda_1 = 80$, $\mathbf{\Lambda}_2 = 2 \cdot \boldsymbol{I}_{3\times3}$
NSB	1	36.65	Yes	Step	Task Activation, $R_{thresh} = 2$
NSB	1	37.35	Yes	Step	Task Activation, $R_{thresh} = 3$
NSB	1	220.05	Yes	Square	Successful Inspection
NSB	1	319.20	Yes	Full	Successful Inspection
Reactive	1	10.05	-	Step	Successful Inspection
Reactive	2	10.05	-	Step	Safety Threatened
VO	1	9.10	-	Step	Successful Inspection
VO	3	9.95	-	Step	Overshoot at the end
VO	4	9.30	-	Step	Safety Threatened

Table 6.1: Table of simulation times

6.1 The NSB Algorithm for Autonomous Inspection

The Null-Space Behavior Control Algorithm was investigated as a possible candidate for collision avoidance. The algorithm provides a simple framework of task scheduling according to priorities, in order to navigate around obstacles. One of the drawbacks of this method, as discussed in section 4.4.6 is that once an obstacle is confronted, the task scheduler will guide the drone all the way around the obstacle, even when the drone has a collision-free path in an earlier stage about halfway around the obstacle. Hence, the scheduler was modified with angle task activation to ensure direct flight towards the goal once the obstacle is avoided. The task activation optimizes the travel time and eliminates the corner case where the drone gets stuck if the null-space contribution becomes zero.
It should be pointed out that the NSB algorithm is a global approach, which considers a known environment at all times. A reactive implantation could be implemented for inspection purposes, based on sensor data provided by the drone's sensors. However, this would require updating of the global map continuously as the inspection is executed, which would be computationally expensive. This also conflicts with the scheduling structure, as the prioritized tasks might suddenly change their prioritization as new obstacles are discovered. The simulation time of the tuned NSB approach is also considerably long compared to the other methods. Then again, slower movements might provide better video recordings than for fast maneuverable flights, as well as our main priority is to avoid collisions, not to optimize inspection time.

6.2 Self-designed Reactive Logic for Autonomous Inspection

A reactive logic was designed as a high level controller for navigation in unknown environments. As the obtained results illustrate, the self-developed logic described in algorithm 1 successfully guide the drone around the sensed obstacle. Based on the simplistic sector feedback from the radar sensors, the rotations of the velocity vector is an intuitive collision avoidance strategy. However, conflicts occur when multiple sectors are active, and a proposed solution was suggested by an extended algorithm that calculates a compromised rotation of the velocity vector. Algorithm 2 seems to solve the main issue that algorithm 1 had but still doesn't consider the relative distances between the drone and the different obstacles. Due to the lack of distance consideration, the drone's safety margin can be threatened, and yet another extension needs to be implemented to the algorithm. However, the simplistic sensor feedback only tells us if the sector senses an obstacle but not how far away it is. We only know that an obstacle

is in the sector, but not if it is at the end of the sensor range, or right next to the drone. Hence, the system needs to extract distance information from the radar sensors. A fusion of the different radar sensor readings can be done for more accurate distance measurements.

6.3 The Velocity Obstacle Algorithm for Autonomous Inspection

Due to the lack of robustness for the self-developed logic, the Velocity Obstacle algorithm was implemented for the simulator. Based on previous work on Velocity Obstacles, it was expected to be a more robust solution for reactive collision avoidance. As the results obtained illustrate, the VO algorithm ensures collision-free trajectories, but the path executed is not optimal. Even though the tuned NSB with task activation and extended threshold has a simulation time of $t_{sim} = 37.35s$ and the VO for one obstacle only uses $t_{sim} = 9.10s$, the path traversed is definitely longer. This has to do with the NSB tuning of the collision avoidance task vs. the reach goal task.
A notable behaviour happens when the velocity obstacle cone is out of the drone's heading direction. When no obstacle occurs in front of the drone, it instantly tries to move towards the initial reference position, which yields a fast turn towards the obstacle's threshold. As a consequence of this behaviour, the drone actually overshoots when it reaches its desired x-position due to the acceleration dynamics. One way of dealing with this problem might be to introduce a delay in which the normal velocities are affecting the drone again. Another way to overcome the acceleration dynamics would be to saturate the velocity even more.

6.4 Combined Approach for Optimality and Robustness

Now that we have seen the different behaviours of the global and local collision avoidance approaches, how can we create an optimal and robust algorithm for multirotor inspection? We want to obtain the accurate proximity keeping that the global method applies, but the unknown environment restricts us for such accurate calculations with the current data provided by the sensors. Can we create a compromised solution that extracts the best of both approaches?
By processing the data collected from the radar sensors, a local map can be created for each sample given by the sensors. Using this local map, a re-planned path can be

generated by using the NSB algorithm. This might be a suitable solution for optimal trajectories and robustness of collision avoidance. However, it also requires the NSB tasks to be rescheduled for every given sample. This frequent rescheduling demands computational power from a computer larger than what a quadcopter usually carry. A solution to the limited computational power might be to process the data elsewhere by sending it to an external computing unit with sufficient computing power. Sufficient bandwidth for the telemetry is then required in order to send all the gathered data. The problem of the acceleration dynamics could be solved by using the nonlinear models but the best solution would be to have the quadcopter to operate within the linearized equilibrium points to avoid high complexity.

Chapter 7

Conclusion and Future Work

Based on the results obtained in chapter 5, a robust collision avoidance algorithm for fully autonomous inspection using multirotor drones is partly developed. We clearly see the advantages of the quadcopter platform for proximity keeping, but still have the challenge of the acceleration dynamics interfering with the desired behaviour. Also, the reactive methods need to gather much more information from the sensors than whats considered in this thesis. The robustness and accuracy of the collision avoidance algorithms could be significantly improved by processing the distance and bearing angle yielded from the radar scans.
The NSB algorithm has proved to be a great way of achieving collision-free trajectories when the global maps are provided, but we also see that we need to consider the corner case of the null-space contribution deviating towards zero, as well as an extended threshold around obstacles because of the drone's acceleration dynamics. The NSB algorithm's performance can be used to set a benchmark for the reactive algorithms' maximum potential. For the self-developed reactive logic algorithm, it is shown that we can design a simplistic high level controller for obstacle avoidance based solely on rotating the velocity vector. Even though this result in collision-free trajectories for some configurations, it is not a robust solution. This is mainly because we only control the velocity directly based only on the active sectors. For future work on the self-developed algorithm, an extension of algorithm 2 can be realized, given that the relative distances between the drone and obstacles are obtained from the sensors. For the Velocity Obstacle algorithm, we have seen that it yields robust collision avoidance. It does however require that we know the dimensions of the obstacles, provided by the

sensor, and an extension of our self designed algorithm might be better suited for our application.

7.1 Suggested Future Work

As the work on this thesis is time limited, the author has not considered every aspect of the topic completely. A suggestion for future work on the topic is to consider more complex information provided by the radar sensors. If SAR can be used for 3D imaging of the sensor readings, a more robust collision avoidance could be designed based on the environmental information. Even though VO is a much used algorithm for collision avoidance, it is most commonly used for applications where several agents are moving in the same environment and by adjusting all agents' velocity vectors. Hence, a self-developed algorithm will be more suited for the drone inspection application. It is therefore suggested to develop an extended algorithm based on bearing angle, distance and sectors provided from the radar sensors. Algorithm 3 is a proposed algorithm considering distance.

Algorithm 3 Reactive algorithm with distance consideration

function REACTIVE($\boldsymbol{v}_{ref}, sensor\_scan$)
  for all $sectors$ **in** $sensor\_scan$ **do**
    if $sector$ **then**
      $current\_distance \leftarrow getDistance(sensor\_scan)$
      if $current\_distance \leq min\_distance$ **then** ▷ Rotates
        $min\_distance \leftarrow current\_distance$ ▷ and scales
        $\boldsymbol{v}_{sector} \leftarrow 2 \cdot \boldsymbol{v}_{ref} \cdot \boldsymbol{R}(sector)$ ▷ the velocity vector
      else
        $\boldsymbol{v}_{sector} \leftarrow \boldsymbol{v}_{ref} \cdot \boldsymbol{R}(sector)$ ▷ Rotates the velocity vector
      end if
    end if
    $\boldsymbol{v}_{middle} \leftarrow \boldsymbol{v}_{middle} + \boldsymbol{v}_{sector}$ ▷ Sum up all velocity contributions
    $\alpha \leftarrow \alpha + 1$ ▷ Obstacle counter
  end for
  $\boldsymbol{v}_{new} \leftarrow \frac{\boldsymbol{v}_{middle}}{\alpha}$ ▷ Divide by number of obstacles
  return $\boldsymbol{v}_{new}$ ▷ New velocity is returned
end function

Bibliography

[1] University of Pennsylvania. *MEAM620: Robotics*. 2018. URL: https://alliance.seas.upenn.edu/~meam620/wiki/ (visited on 01/19/2018).

[2] Daman Bareiss, Joseph R. Bourne, and Kam K. Leang. "On-board model-based automatic collision avoidance: application in remotely-piloted unmanned aerial vehicles". In: *Autonomous Robots* 41.7 (2017), pp. 1539–1554. ISSN: 15737527. DOI: 10.1007/s10514-017-9614-4.

[3] Kristian Klausen. "Cooperative Behavioural Control for Omni-Wheeled Robots". In: June (2013).

[4] Filippo Arrichiello. "Coordination control of multiple mobile robots". In: November (2006), p. 141. URL: http://www.scuoladottoratoingegneria.unicas.it/Tesi/Ciclo%20XIX/Tesi%20Arrichiello.pdf.

[5] P Fiorini and Z Shiller. "Motion planning in dynamic environments using velocity obstacles". In: *Int. Journal of Robotics Research* 17.7 (1998), pp. 760–772.

[6] Javier Alonso-Mora et al. "Collision avoidance for aerial vehicles in multi-agent scenarios". In: *Autonomous Robots* 39.1 (2015), pp. 101–121. ISSN: 15737527. DOI: 10.1007/s10514-015-9429-0.

[7] Vetle Andre Bjelland. "Investigation of Collision Avoidance Algorithms Targeted for an Inspection Drone". In: (2017).

[8] Bruno Siciliano and Oussama Khatib. *Springer Handbook of Robotics*. 2008. ISBN: 978-3-540-23957-4. DOI: 10.1007/978-3-540-30301-5. arXiv: arXiv:1011.1669v3. URL: http://www.mendeley.com/research/force-tactile-sensors/.

[9] James S. Albus. "Outline for a theory of intelligence". In: *IEEE Transactions on Systems, Man and Cybernetics* 21.3 (1991), pp. 473–509. ISSN: 00189472. DOI: 10.1109/21.97471.

[10] Peter E. Hart, Nils J. Nilsson, and Bertram Raphael. "Formal Basis for the Heuristic Determination eijj ," in: *Systems Science and Cybernetics* 2 (1968), pp. 100–107.

[11] Oussama Khatib. *Real-Time Obstacle Avoidance for Manipulators and Mobile Robots*. 1986. DOI: 10.1177/027836498600500106. arXiv: arXiv:1011.1669v3. URL: http://journals.sagepub.com/doi/10.1177/027836498600500106.

[12] Johann Borenstein and Yoram Koren. "The Vector Field Histogram: Fast Obstacle Avoidance for Mobile Robots". In: *IEEE Transactions on Robotics and Automation* 7.3 (1991), pp. 278–288. ISSN: 1042296X. DOI: 10.1109/70.88137.

[13] R. Simmons. "The curvature-velocity method for local obstacle avoidance". In: *Proceedings of IEEE International Conference on Robotics and Automation* 4.April (1996), pp. 3375–3382. ISSN: 0-89791-362-0. DOI: 10.1109/ROBOT.1996.511023. URL: http://ieeexplore.ieee.org/document/511023/.

[14] James Jackson, David Wheeler, and Tim McLain. "Cushioned extended-periphery avoidance: A reactive obstacle avoidance plugin". In: *2016 International Conference on Unmanned Aircraft Systems, ICUAS 2016* (2016), pp. 399–405. DOI: 10.1109/ICUAS.2016.7502597.

[15] Anthony Stentz. "Optimal and Efficient Path Planning for Partially Known Environments". In: *Intelligent Unmanned Ground Vehicles* (1997), pp. 203–220. ISSN: 03405354. DOI: 10.1007/978-1-4615-6325-9_11. URL: http://link.springer.com/10.1007/978-1-4615-6325-9%7B%5C_%7D11.

[16] Cyrille Berger et al. "Evaluation of reactive obstacle avoidance algorithms for a quadcopter". In: *2016 14th International Conference on Control, Automation, Robotics and Vision, ICARCV 2016* 2016.November (2016), pp. 13–15. DOI: 10.1109/ICARCV.2016.7838803.

[17] Gianluca Antonelli, Filippo Arrichiello, and Stefano Chiaverini. “The null-space-based behavioral control for autonomous robotic systems”. In: *Intelligent Service Robotics* 1.1 (2008), pp. 27–39. ISSN: 18612776. DOI: 10.1007/s11370-007-0002-3.

[18] S M LaValle. “Rapidly-Exploring Random Trees: A New Tool for Path Planning”. In: *In* 129 (1998), pp. 98–11. ISSN: 1098-6596. DOI: 10.1.1.35.1853. arXiv: arXiv:1011.1669v3. URL: http://scholar.google.com/scholar?hl=en%7B%5C&%7DbtnG=Search%7B%5C&%7Dq=intitle:Rapidly-exploring+random+trees:+A+new+tool+for+path+planning%7B%5C#%7D0.

[19] Paolo Fiorini and Zvi Shiller. “Motion planning in dynamic environments using the relative velocity paradigm”. In: *1993 IEEE International Conference on Robotics and Automation* 1 (1993), pp. 560–566. ISSN: 10504729. DOI: 10.1109/ROBOT.1993.292038. URL: http://www.scopus.com/inward/record.url?eid=2-s2.0-0027264303%7B%5C&%7DpartnerID=tZOtx3y1.

[20] New Brunswick. “Advanced Control”. In: June (2013).

[21] Hassan K Khalil. “Nonlinear systems”. eng. In: (2015).

[22] Thor I. Fossen. *Handbook of Marine Craft Hydrodynamics and Motion Control*. 2011. ISBN: 9781119991496. DOI: 10.1002/9781119994138.

[23] Vijay Kumar and Nathan Michael. “Opportunities and challenges with autonomous micro aerial vehicles”. In: *The International Journal of Robotics Research* 31.11 (2012), pp. 1279–1291. ISSN: 0278-3649. DOI: 10.1177/0278364912455954. URL: http://journals.sagepub.com/doi/10.1177/0278364912455954.

[24] MIT Open Courseware. *Principles of the Global Positioning System*. 2012. URL: https://ocw.mit.edu/courses/earth-atmospheric-and-planetary-sciences/12-540-principles-of-the-global-positioning-system-spring-2012/ (visited on 04/13/2018).

[25] Brage Gerdsønn Eikanger. “The path to Autonomous Inspection using an Unmanned Aerial Vehicle Brage Gerds{ø}nn Eikanger”. In: June (2017).

[26] Stefan Hrabar. “3D path planning and stereo-based obstacle avoidance for rotorcraft UAVs”. In: *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS* (2008), pp. 807–814. ISSN: 978-1-4244-2057-5. DOI: 10.1109/IROS.2008.4650775.

[27] You He. “Radar Data Processing with Applications (1)”. eng. In: (Aug. 2016). URL: http://site.ebrary.com/id/11244272.

[28] Randal W. Beard and Timothy W. McLain. “Small unmanned aircraft theory and practice”. In: (2012).

[29] Rodney A. Brooks. “A Robust Layered Control System For A Mobile Robot”. In: *IEEE Journal on Robotics and Automation* 2.1 (1986), pp. 14–23. ISSN: 08824967. DOI: 10.1109/JRA.1986.1087032. arXiv: 1010.0034.

[30] R. C. Arkin. “Motor Schema – Based Mobile Robot Navigation”. In: *The International Journal of Robotics Research* 8 (1989), pp. 92–112. ISSN: 0278-3649. DOI: 10.1177/027836498900800406.

Appendix A

Appendices

A.1 Theorems

A.1.1 Hurwitz

A matrix $\boldsymbol{A}$ is Hurwitz; that is, $Re\{\lambda_i\} < 0$ for all eigenvalues of $\boldsymbol{A}$, if and only if for any given positive definite symmetric matrix $\boldsymbol{Q}$ there exists a positive definite symmetric matrix $\boldsymbol{P}$ that satisfies the Lyapunov equation $\boldsymbol{PA} + \boldsymbol{A}^T\boldsymbol{P} = -\boldsymbol{Q}$. Moreover, if $\boldsymbol{A}$ is Hurwitz, then P is the unique solution of $\boldsymbol{PA} + \boldsymbol{A}^T\boldsymbol{P} = -\boldsymbol{Q}$. This as proved in theorem 4.6 in [21]

A.1.2 Exponential Stability

Let $x = 0$ be an equilibrium point for $\dot{x} = f(t, x)$ and $D \subset R^n$ be a domain containing $x = 0$. Let $V : [0, \infty) \times D \to R$ be a continuous differientable function such that:

$$\begin{aligned} k_1||x||^a \leq V(t,x) \leq k_2||x||^a \\ \frac{\partial V}{\partial t} + \frac{\partial V}{\partial x} f(t,x) \leq -k_3||x||^a \end{aligned} \tag{A.1}$$

$\forall t \geq 0$ and $\forall x \in D$, where k_1, k_2, k_3 and a are positive constants. Then, $x = 0$ is exponentially stable. If the assumption holds globally, then $x = 0$ is globally exponentially stable (GES). This as proved in theorem 4.10 in [21]

A.2 Minkowski Addition

A.2.1 Minkowski Sum

In geometry, the Minkowski sum of two sets of position vectors S_A and S_B in Euclidean space is formed by adding each vector in S_A to each vector in S_B, for example, the set

$$S_A \oplus S_B = \{\boldsymbol{a} + \boldsymbol{b} \mid \boldsymbol{a} \in S_A,\ \boldsymbol{b} \in S_B\} \tag{A.2}$$

A.2.2 Minkowski Difference

In geometry, the Minkowski difference of two sets of position vectors S_A and S_B in Euclidean space is formed by subtracting each vector in S_B from each vector in S_A, for example, the set

$$S_A \ominus S_B = \{\boldsymbol{a} - \boldsymbol{b} \mid \boldsymbol{a} \in S_A,\ \boldsymbol{b} \in S_B\} \tag{A.3}$$