Quality assurance with drones: Relative navigation for targeted inspection of large structures using drones

Martin Schörner

Angaben zur Veröffentlichung / Publication details:

Schörner, Martin. 2026. “Quality assurance with drones: Relative navigation for targeted inspection of large structures using drones.” Augsburg: Universität Augsburg.

Quality Assurance with Drones

Relative Navigation for Targeted Inspection of Large Structures Using Drones

Martin Schörner

Dissertation
zur Erlangung des akademischen Grades
Doktor der Naturwissenschaften (Dr. rer. nat.)

Fakultät für Angewandte Informatik

16. Mai 2025

Quality Assurance with Drones

Erstgutachter:	Prof. Dr. Wolfgang Reif
Zweitgutachter:	Prof. Dr. Bernhard Bauer
Tag der mündlichen Prüfung:	25. Juli 2025

"Airplanes are beautiful dreams,
engineers turn dreams into reality."

— Hayao Miyazaki

Acknowledgements

This work would not have been possible without the support I received from various people. First, I want to thank my supervisor, Professor Dr. Wolfgang Reif. He provided an environment where I could concentrate on my studies, gave me the freedom to realize my ideas, and guided me throughout this dissertation.

I also want to thank all my colleagues who gave feedback on my ideas, participated in discussions about the project, and provided support in difficult times. I would especially like to thank Professor Dr. Constantin Wanninger, who first got me into a career in science, and without whose mentoring, I would not have finished the dissertation. Special thanks also go to current and former members of the ISSE robotics department and other close colleagues, namely Dr. Alexander Poeppel, Dr. Matthias Stüben, Dr. Julian Hanke, Dr. Christian Eymüller, Daniel Bermuth, Michael Filipenko, our technician Stefan Wolff, and Dr. Hella Ponsar.

I would also like to thank all students who worked with me during the past years. It was a great privilege to work with such a loyal team and watch every one of you develop and learn over the past five years. Special thanks go to some of my long-term student assistants, Raphael Katschinsky, Simon Hornung, Raban Poppall, and Jan Dräger, who made a significant effort to help me bring the QuAD project into a state where it is not just an assortment of tossed-together prototypes but a proper modular framework.

I am also thankful for the help provided by our secretarial team, consisting of Melina Buchschuster, Bettina Grabmann, and Rositta Bingger. They took most of the tedious administrative work off my hands and provided a protective shield against the frustrating world of the university's administrative apparatus, allowing me to focus on my studies.

Thanks are also due to Kevin Dittel, Christian Adorian, Stefan Salzburger, Halil Sinmaz, and Ismail Teoman from Airbus, who played a crucial part in developing the idea for QuAD. They provided invaluable insights into Airbus's production processes and feedback about the required functionalities and the various assemblies as well as structures needed for our tests.

Last, I want to thank my friends and family for their continuous support and understanding throughout the years.

Martin Schörner

Abstract

Due to advances in technology, the use of autonomous flying robots for inspecting large structures has become increasingly popular in recent years. However, most applications focus on performing outdoor inspections. In the manufacturing sector, automated inspections of large structures performed by autonomous flying robots are rare due to the additional challenges, such as the complex environment and the unavailability of GPS signals. Nevertheless, the global economy's demand for efficient manufacturing requires the automation of inspection processes in this sector.

This work introduces a modular relative path planning, perception and navigation system for flying robots performing inspections that relies only on onboard sensors and is based on model tracking. The system supports safe autonomous flight while avoiding collisions in complex and highly dynamic environments. The system includes path and trajectory planning algorithms based on either the targeted inspection of specific Points of Interest on the assembly or coverage of the entire surface of the model which can be used interchangeably depending on the specific inspection task. The architecture is designed to allow for easy integration of existing, ground-based inspection and error detection tools. It's functionality is demonstrated through a number of prototypes based on case studies focusing on inspecting aircraft fuselage parts and a wind turbine. The prototypes are used to perform simulated and real-world proofs of concept and evaluations. The results demonstrate the system's ability to plan efficient inspection paths, precisely navigate relative to the inspected structures, and detect and avoid obstacles in real time, thus validating its effectiveness in the complex environments encountered inside manufacturing facilities.

Contents

Summary. This chapter presents the potential of automated inspection using UAVs and two case studies of applications that can benefit from UAV-based inspections. Afterward, existing solutions and concepts for automated inspection are presented, and the objectives of this work are defined.

1 Introduction

Quality assurance is a critical part of modern manufacturing. A global economy has resulted in more competition and thus demands a more efficient and cost-effective production. This often requires the automation of labor-intensive processes to reduce manpower [170]. As a result, a need for the automation of quality assurance in the form of inspections during the manufacturing process has developed in recent years. Frequent inspections during the manufacturing process are a means to ensure a high-quality product and prevent the waste of resources through performing further work an already damaged or defective part. Especially during the manufacture of large components, performing manual inspections can be difficult, as it often requires additional infrastructure like scaffolding, lifting platforms, or industrial climbers. This work is often dangerous, and current safety standards can prevent workers from inspecting certain areas of an assembly [241]. The repetitive nature of structures like aircraft fuselages can often confuse human workers and lead to errors during inspections. While several solutions exist to facilitate manual inspections through tools like Augmented Reality (AR) [100, 286], the inspection process still must be performed by a human worker in most cases. The use of uncrewed aerial vehicles (UAVs) for performing inspections seems advantageous, as they are able to fly to inspection points that cannot be easily reached by humans and do not require additional support structure around the assemblies. UAVs are already used to perform various inspections on infrastructure like bridges [125, 260] and energy infrastructure [58, 123, 173, 180, 302] as well as to monitor the status of construction sites [155] and crops [301]. However, the use of UAVs in manufacturing proves more difficult, as signals from global navigation satellite systems like GPS often do not penetrate into buildings. Additionally, the dynamic, more complex, and often crowded environment poses more challenges for UAVs to navigate due to the increased potential for collisions with obstacles. This results in the use of UAVs being less common in manufacturing, which is often performed in enclosed facilities [171].

This work focuses on developing concepts for the use of UAVs for inspections performed in such environments and aims at finding solutions for the problems mentioned above. Throughout the following chapters, concepts are developed to plan efficient inspection paths, estimate the UAV's position relative to a structure and navigate alongside it while simultaneously avoiding obstacles. Based on these concepts, the QuAD (Quality Assurance using Drones) System was developed. QuAD is a platform for performing inspections on large structures during assembly and in other scenarios using multirotor UAVs.

The required basic techniques, developed concepts, architecture, and prototypes, as well as the performed proofs of concept and evaluations, are detailed in the following chapters. The following paragraphs of this chapter focus on providing a motivation for UAV-based inspections in manufacturing, introducing two separate case studies and summarizing the contributions of this work.

1.1 Motivation

Historically, the inspections performed for quality assurance in manufacturing were a human-centered process [166]. The inspector would compare the product to a blueprint, or a reference part. Based on these references, the inspected product would either pass or fail the inspection. Since these types of inspections are labor-intensive, they were not performed after every manufacturing step, with some products only being inspected after completion. This led to high scrap rates, wasted resources on manufacturing steps on already defective parts, and thus increased production costs. More complex products and smaller tolerances required for more modern products made this process increasingly less effective, leading to the introduction of computer vision-based systems aiding in the inspection process [166]. Increased competition through a globalized market increased pressure on manufacturers to produce their products in a more cost-efficient way [18]. Additionally, more complex products and production processes increased the demands on human workers, which increased the likelihood of errors in inspection tasks and resulted in the need for a more subjective and reliable way to perform inspections [18]. Especially in the manufacturing of large products like ships, wind turbines or airplanes, inspections are still performed mostly manually due to the size and complexity of the structures preventing the use of stationary inspection systems intended for smaller products or systems utilizing ground-based robots, which may not be able to reach all required inspection points. Manual inspections of these structures often require the use of additional structures or scaffolding and can be dangerous to the personnel, to the point that local regulations may prohibit certain types of inspections due to safety reasons. Often, support structures used to manufacture the assembly are used to perform the subsequent inspection. This, however, blocks the support structure, which may be needed for manufacturing the next part. This problem is especially present in the manufacturing of aircraft fuselage parts, where derivatives of decades-old aircraft designs are still manufactured. During design time, the high levels of competition present in today's markets were not taken into consideration, and thus, little effort was made to make the inspection processes efficient. This results in inspections of assemblies only being made after several assembly steps, which can result in situations where additional parts are fitted to assemblies that were rendered defective during an unnoticed error a few steps prior. The repetitive nature of these assemblies also makes manual inspection prone to human error. These factors result in high error rates, time expenditure, costs for the inspection process, and a need for a better solution. An automated inspection performed by UAVs no longer requires the inspection process to be carried out by human inspectors. Therefore, inspections can be performed outside the production stations and independently of scaffolding, freeing these facilities for the manufacturing of the next assembly. Humans are also no longer required to work in potentially dangerous areas, increasing worker safety and enabling inspections in areas that are impossible with current work safety regulations. Automatic conformity checking and defect detection allow for a more objective and consistent inspection process, eliminating variations in acceptance criteria between different human inspectors. Automating this process also allows for easy documentation of the production process and traceability of defects that occur later in the product life cycle.

Advantages of Selective Inspections

Existing UAV-based inspection solutions focus on capturing the state of the whole assembly. During in-service inspections of objects like aircraft, bridges, and wind turbines, this makes sense as the types of defects that the inspections try to detect, like cracks, delaminations in composite parts, or lightning strikes, can occur anywhere on the inspected object. However, covering the entire surface of an object is time-consuming (which is a problem due to the limited flight time of UAVs) and inefficient in many manufacturing scenarios, especially during assembly, where altered areas of an assembly can easily be determined based on the performed manufacturing steps. Selective inspection of areas that were modified during a specific manufacturing step can therefore result in a more time and cost-efficient inspection process. This can be combined with less frequent inspections covering the entire surface of the assembly to check for damage introduced to different parts of the assembly while performing the preceding manufacturing steps.

Advantages of Relative Positioning and Navigation

The inability of global navigation satellite signals to penetrate into buildings, combined with their low accuracy, makes them unusable in most manufacturing scenarios [133]. External positioning solutions like camera tracking systems or ultra-wideband require the installation of external sensors in each area where inspections are performed, making them less practical. External camera tracking systems have the additional disadvantage that they require visual line of sight to the tracked target, which complicates measuring a UAV's position once it enters a confined space inside an assembly. Therefore, using only the UAV's internal sensors is advantageous. Development in Augmented Reality applications in recent years has made model tracking (see: Section 2.2.4) a promising possibility of establishing the position of the UAV's camera relative to the inspected structure. This approach does not require the installation of markers on the inspected assembly, which might not be desirable due to aesthetic reasons and additional cost introduced by the precise installation and subsequent removal of these markers.

When tracking the inspected assembly, model tracking adds the benefit that position estimation and navigation are performed relative to the assembly, in contrast to external systems, which provide positions relative to a fixed origin, like the geographic North Pole or an arbitrary origin set in the camera tracking system. This allows for the specification of inspection points relative to the assembly, independent of its absolute position, enabling an inspection to be performed with the assembly located at an arbitrary position. This is especially beneficial when inspecting multiple identical products, like in a buffer storage facility, where assemblies are often kept between two manufacturing steps. Since the UAV performs all movements relative to the currently tracked object, inspection points and paths can be reused for identical assemblies in arbitrary positions, changing only the tracked target between inspections. This concept is even beneficial in other scenarios, like the inspection of wind turbines or power towers, where numerous identical objects are inspected.

Inspection Techniques in Manufacturing

During the manufacturing of a product, a plethora of different inspections need to be performed. In assembly, detecting the presence of parts, their correct location, and orientation is one of the most frequently used checks, and products that partially automate these checks exist in the industry [100, 286]. The automatic detection of scratches, dents, and other defects caused during production is another common type of inspection, for which commercial solutions exist [1]. Other common inspection types include the photogrammetric documentation of the state of objects for later manual analysis [51] and thermographic inspections, which are frequently used in composite manufacturing to detect subsurface defects in the material [46]. However, the number of different metrics and scenarios that can be checked for during an inspection is endless, ranging from detecting the protrusion of rivets [165] or the correct installation of cables inside a cable clamp over the proper application of sealant in a joint to the detection of microscopic cracks using methods like dye penetrant testing [277]. Each inspection type requires different approaches for gathering, capturing, and processing inspection data. However, the data for most of these inspections can be obtained using a camera capturing either visible light or thermographic images, and most trajectories that perform these inspections are either based on capturing images of specific points of the object or covering its entire surface. Therefore, developing a system that can perform every possible type of inspection that might be required in a manufacturing scenario is impossible. Instead, providing a system that is able to perform flights along trajectories covering the entire surface of an object or gather data from specific predefined points is sufficient to perform most types of inspections. As long as the gathered data and additional information like the camera position are provided, the functionality of external inspection solutions for the specific use-case can be extended to perform UAV-based inspections.

Therefore, this work aims to develop concepts and an exemplary implementation of a system for UAV-based, relative inspection of large assemblies in manufacturing. This includes a relative positioning system that does not rely on external sensor data like GPS, markers, or camera tracking and navigates relative to the inspected objects. While navigating through complex environments often found in manufacturing facilities, the system has to avoid previously known and unknown obstacles that might be stationary or moving throughout the inspection flight. The UAV has to be small enough to perform inspections inside structures like airplane fuselage parts, which limits the type of sensors that can be used for obstacle avoidance and the inspection. Finally, the system has to be agnostic to the used inspection technique, providing the ability to plan and execute trajectories covering either the entire surface of the inspected object or selectively inspect specific parts. Before introducing the proposed system, the following sections describe two distinct case studies to provide a deeper understanding of inspections in manufacturing and in-service inspections, as well as the current state of the art in this field.

1.2 Case Studies

The inspection of aircraft fuselage parts during manufacturing is used as a primary case study to illustrate the concepts and techniques developed throughout this work. The following paragraphs will explain why most inspections are still performed manually in this domain and the disadvantages of this approach. Additionally, in-service wind turbine inspections are introduced as a secondary case study used later in this work to demonstrate the advantages of several more advanced concepts when navigating along structures with multiple, independently moving sections.

1.2.1 CS1: Inspection of Aircraft Fuselages

Due to the high safety standards in the aviation industry, inspections are essential when building and operating aircraft, which must undergo frequent inspections to maintain its airworthiness certification during operation. In the US, these inspections are required by the Federal Aviation Administration (FAA) every 12 months or every 100 hours of operation, depending on the type of aircraft [284]. These inspections include checks of the entire fuselage for cracks, deformations, and deterioration. While these checks are only a few of many that have to be performed around the entire aircraft, the size of modern commercial airliners requires lifting platforms or scaffolding to be put up so that humans can perform these checks directly. Due to the costs induced by these inspections manually, many companies have developed solutions for performing them by UAV [105], [61], [173].

While various solutions are available for UAV-based in-service aircraft inspections, quality assurance during production is still primarily performed manually, and few automated solutions for inspecting large aircraft fuselage parts during production exist, covering only a small set of inspection tasks [170, 177]. Due to the high safety standards for aircraft manufacturing, an inspector must check every manufacturing step. Additionally, the high cost of development and certification often causes decades-old aircraft designs to remain in production, or the original models are simply updated to accommodate more passengers or improve fuel efficiency. This results in aircraft like the Airbus A320 *Single-Aisle* series, originally designed in the late 1970s and early 1980s. When the A320 was developed, the number of passengers carried by air transport annually was around 650 million [20]. This number rose to 4.46 billion in 2019, which required an unforeseen number of airplanes to cover this demand. While having received several upgrades and variants over the years, variations of this approximately 40-year-old design have been built over 12000 times with over 20000 total orders as of January 2025, leaving a little under 8000 of these planes to be built [258]. The extent of the demand for this model was not clear during the design phase, so manufacturing processes were not optimized for the high output required today. This was amplified by the fact that robotics and automation in production were in their infancy at this time and had not yet been widely adopted by aircraft manufacturers. Also, the complex geometries of the fuselage parts did not allow for a high degree of automation for a long time [31].

While this example only covers one type of aircraft, the Single-Aisle series makes up 81 % of Airbus's total orders, and 89 % of open orders are Single-Aisle aircraft [258]. Airbus, being the largest airplane manufacturer by market capitalization in 2024 [34], highlights the significance of this problem.

In recent years, automation was slowly introduced into the actual manufacturing steps, like drilling [259] or painting [31], but the inspection process is still performed with a low degree of automation. The only notable advances in this sector have been Augmented Reality tools and frameworks like MiRA [100], DELMIA [269] and VisionLib [286] that aid human inspectors with verifying the correct installation of parts on an assembly as well as the Iris GVI [63] UAV developed by Donecle [61] that is able to capture images of the entire surface of the assembly and perform some limited processing of this data like the extraction of serial numbers or highlighting the difference between the part in two subsequent inspection flights.

Structure and Terminology of Aircraft Fuselage Parts

The manufacturing of an airplane consists of a variety of different production steps. This work mainly focuses on the types of inspections required during the assembly of fuselage parts. These structures are built by attaching large amounts of parts using a combination of drilling, deburring, gluing, and riveting operations. These parts are mostly stamped and bent sheet metal brackets or machined aluminum parts for more complex geometries. Figure 1.1 shows the general structure of an airplane fuselage. It consists of a formed sheet metal skin supported by frames supporting the circumference of the tubular fuselage section and stringers providing support in the lengthwise direction [184]. This structure serves as a basis for installing various struts and brackets that either reinforce the fuselage's structure or serve as mounting points for other hardware, such as insulation, wiring, or paneling.

State of the Art of Manual Inspection

As mentioned before, most of the fuselage's structural parts are installed through a combination of gluing and riveting. Airbus offers its customers a high level of customization through features like optional winglets, a short airfield package, different cabin layouts, and comfort features for passengers. This results in a high variability in the resulting airframes, making the automation of bracket installation difficult. Therefore, a significant number of brackets are still installed manually by human workers. The repetitive nature of the structures that are worked on (see: Figure 1.1) introduces a high potential for human error during installation. The same is true for the inspection process itself. Additionally, this complex task is not only cost-intensive but also requires a great deal of manual labor, which can cause problems as it may require hiring new inspection workers when the required production output varies. The quality control during fuselage manufacturing has evolved from manual matching of the physical part to the manufacturing drawings and manual dimensional inspections to a computer-assisted approach using tools like MiRA [100] and DELMIA [269]. These tools allow workers to augment the camera image of an inspection tablet with an AR overlay provided by the

Computer Aided Design (CAD) model to spot mistakes more efficiently or even perform checks for the presence as well as the correct position and orientation of parts automatically. While augmenting the camera image reduces human errors during inspection, these tools still require manual positioning of the inspection camera at the part being inspected. When working with large assemblies, this necessitates additional scaffolding to be in place to get the camera to all required positions. Often, the same structures used to perform a particular production step are used to perform the corresponding inspection. However, this means that a supporting structure which could already be used to manufacture the next part is still occupied to perform an inspection. With a demand for high production output, inspections are often performed only every few production steps. While this procedure still guarantees the inspection of every installed part, it enables a scenario where an unfixable error stays undetected for longer, wasting time and resources on further work on a defective part.

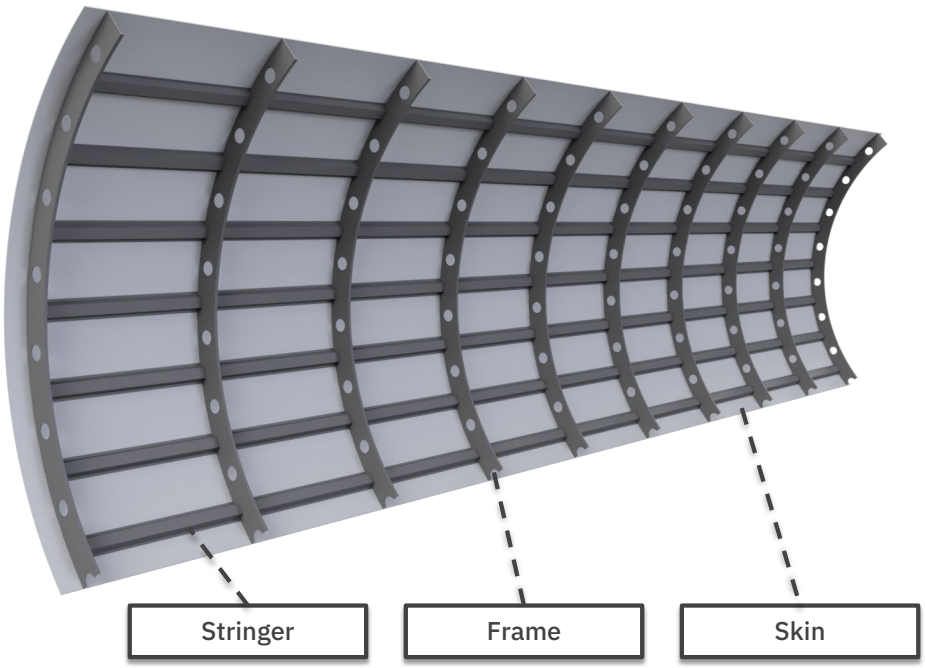


Figure 1.1. Model of an aircraft fuselage part consisting of skin, stringers, and frames.

1.2.2 CS2: Inspection of Wind Farms

Regular in-service inspection of infrastructure is required for a variety of different sectors like power lines [123], bridges [260], or wind farms [285]. These inspections can document the structure's state to conform to regulations, be performed as a damage report after an accident, or detect age-related damage and wear. This work uses the inspection of a wind turbine as a secondary case study that highlights the benefits of relative positioning and navigation during an inspection process.

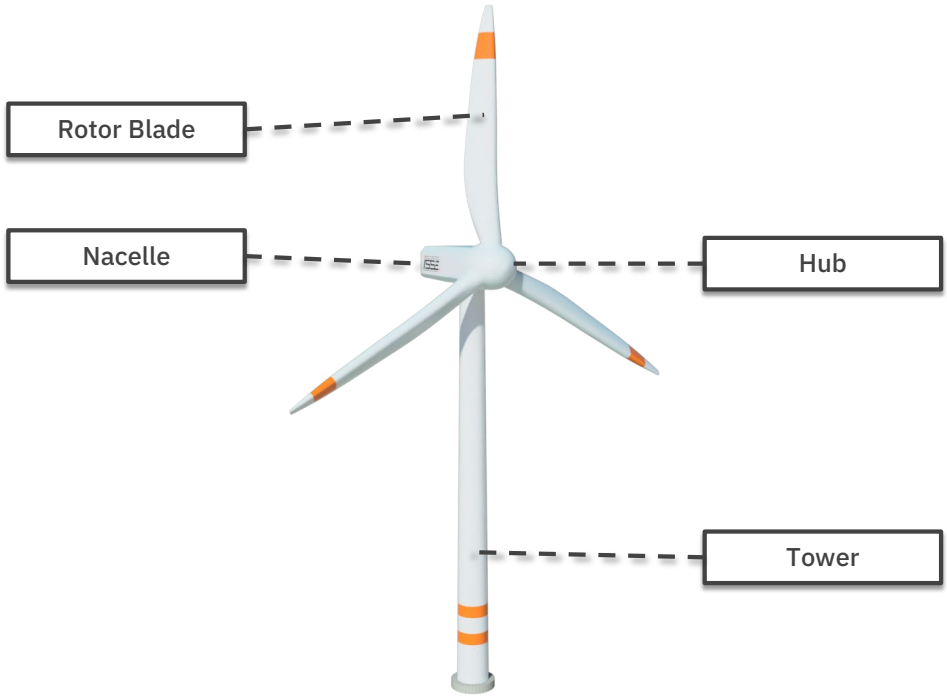


Figure 1.2. Basic terminology of a wind turbine consisting of a tower acting as a base for the nacelle. The rotor hub and blades are mounted to the nacelle.

Figure 1.2 shows the terminology of the individual parts of a wind turbine that might need to be inspected. The tower is mounted on a foundation, providing a mounting point for all other components at the correct altitude [276]. The nacelle is mounted to the tower and can swivel to point the turbine toward the wind. The nacelle also houses the generator, gearbox, brakes, and all other drive train parts required for power production. The hub is attached to the nacelle and transfers the rotational energy of the turbine blades into the drive train through a shaft. The rotor blades can change their pitch through a rotational joint at the hub's mounting point. This allows the rotor to be reconfigured for ideal efficiency at different wind speeds.

The inspection tasks on a wind turbine system are quite diverse. They range from checking the systems of the power train like the gearbox, bearings, and generator [241], to checking the condition of secondary features like the lightning protection system or drainage holes [5]. While the gearbox, bearings, and generator may be checked from within the turbine's nacelle, many tasks require a visual inspection from the outside. Some of these tasks are checks for:

- leading edge erosion
- cracks and pinholes
- clogged drainage holes
- damages on the lightning protection system
- damage through bird strike
- delamination and separation of the leading and trailing wing edge

as well as the documentation of the overall state of the structure [5], [241].

The structure of wind turbines introduces some challenges when performing the these inspections manually. Their size often makes it difficult for humans to reach all relevant areas of the structure. This can require the erection of scaffolding, lifts, cranes [10] or, in some cases, the use of helicopters [200]. Some types of inspections can only be performed with the help of industrial climbers [10]. While all these methods work and are in use today, they are inherently expensive and potentially endanger the inspection personnel. Hence, efforts have been made to find other ways to perform these inspections in the past. This includes performing the inspection with UAVs. Automated wind turbine inspections use enterprise-grade GPS-guided UAVs like the DJI M300 [256] to fly preplanned missions and capture high-resolution images of the entire exterior surface of the turbine. An inspector manually checks these images after the flight and sends a report to the turbine's owner. The companies offering these inspections advertise increased efficiency and reduced turbine downtime compared to conventional inspections. A byproduct of the process is the capture of high-resolution images of the turbine at angles that cannot be achieved by climbers or cranes while taking significantly less time. All this leads to an overall reduced inspection cost of $300-$500 per turbine vs. $1200-$2500 for the price of a manual inspection [241]. Also, an inspection by UAV is significantly safer for the inspection workers as the entire inspection task can be performed while sitting at a desk.

While several commercial solutions for inspecting wind turbines exist, the UAV is often piloted manually, or the inspection points it has to visit are defined manually, increasing the complexity of the setup for each inspection. Reusing inspection trajectories, especially on a large wind farm with over 100 turbines, can significantly reduce the inspection setup time and thus the cost. Also, conventional inspection UAVs cannot navigate around moving turbine blades. This requires the turbines to be shut down for inspection, indirectly increasing the inspection cost.

1.3 State of the Art of UAV-Based Inspections

UAV-based inspection has become increasingly popular in various fields in recent years [198]. The industries where these kinds of inspections are performed nowadays include:

- Progress monitoring at archaeological sites [77]
- Precision agriculture and crop monitoring [301]
- Oil and gas pipeline monitoring [191, 267]
- Construction [155]
- Mining [151]
- Rail infrastructure [60]
- Energy infrastructure like power lines [58, 123, 180, 302], power towers [291], wind turbines [5, 192, 241, 265, 285] and solar parks [305]
- In-service aircraft inspections [7, 8, 17, 190]
- Transport infrastructure inspection [195, 260]
- Airport inspections [141]

This work focuses on the inspection of large assemblies in a production context. Compared to the previously mentioned applications, the requirements for path planning and the inspection tasks that need to be performed differ in this application. However, many similarities to different types of UAV inspections still exist. Therefore, the following sections give an overview of relevant related works and the general state-of-the-art of UAV inspections. A special focus will be on the inspection of aircraft and wind turbines as they are the subject of the case studies presented in Section 1.2. While UAVs are already used for in-service aircraft inspections, they are not abundantly used during manufacturing, as discussed in CS1. Therefore, an overview of the available work related to UAV-based inspections for in-service inspections is given as well.

1.3.1 In-Service Inspection of Aircraft

The use of UAVs to perform in-service inspections of aircraft is becoming more popular in recent years [7, 8]. Bardis et al. [21] give an overview of non-invasive testing techniques used for automated UAV inspections of aircraft. The use of UAVs in this field is motivated by an increase in efficiency and thus a reduction in operating and maintenance costs. The data acquisition process of an in-service inspection can be reduced from a few hours to a flight lasting only 10 to 30 minutes, with the whole inspection finished after about 3 hours, compared to a day using conventional methods [7, 8]. Environmental and operational stressors can cause a variety of defects on the aircraft. For metallic parts, these can be fatigue cracks, corrosion, or mechanical deformation caused by foreign objects striking the fuselage. Composite parts can suffer from delamination of individual layers, cracks, and impact damage. The methods used for in-service inspections include thermography and 3D-scanning, but most of the inspections required by most authorities are visual inspections [208]. Since most UAVs are already equipped with an onboard camera, they are well-suited to perform these kinds of inspections.

Commercial Solutions

As the biggest airplane manufacturer by market capitalization [34], Airbus has already made multiple efforts to automate the in-service inspection of its airplanes. Early systems [7] feature a GPS-guided UAV using an onboard camera to capture images of the upper part of an airplane and perform a photogrammetric reconstruction to generate a textured 3D-model of the inspected aircraft. The flight is performed automatically, but a human supervisor is still necessary. Data acquisition for this particular type of inspection can be performed in a 10 to 15-minute flight, compared to two hours of manual labor when capturing the images using a worker on a lifting platform.

A later system developed by the Airbus subsidiary Testia [8] uses an octacopter in an X-configuration (see: Rodríguez et al. [234]) equipped with a camera and laser obstacle sensors to perform a similar type of inspection. The main difference with this system is that the laser sensors are used for positioning and obstacle avoidance, allowing for inspection flights in GPS-denied environments like hangars. The UAV appears to be specifically designed for these inspections and is equipped with protective shrouds around the propellers to limit the amount of damage in case of a crash. The system can capture the required images for the specific type of inspection it was developed for during a single 30-minute flight, which allows the entire inspection process to be finished within 3 hours. Conventional inspections of the same type can last for up to one day. The time savings cause reduced downtime of the affected airplane and thus can result in immense cost savings.

The Toulouse-based company Donecle [50, 61] also develops UAVs for in-service aircraft inspection. Donecle's inspection UAV, the Iris GVI [63] is approved by the FAA and EASA and listed in the aircraft maintenance manuals for Boeing and Airbus. They also offer the use of multiple UAVs in parallel to speed up inspections. The UAVs can also be equipped with a specialized *dentCHECK* sensor that increases the detection speed when checking for dents in the fuselage. Donecle also offers automated component inspections using this UAV [62], and lists the inspection of landing gear and engines as possible applications. The system helps with identifying differences on an assembly between two inspections and is able to automatically extract serial numbers from the captured images. However, more advanced automated inspection techniques like detecting correct installation of parts are not mentioned.

Other notable commercial providers of UAV-based in-service inspections of aircraft include Mainblades [173], Autaza [16, 55, 207] and Luftronix [168, 211].

Scientific Literature

In scientific literature, Rodríguez et al. [234] give an overview of the UAV-based inspection of airports and aircraft by analyzing 76 relevant papers. They conclude that the most common inspection method is visual inspection, which creates a record of the entire state of the aircraft by capturing images and reconstructing a 3D model using image or LiDAR scan data (see: Section 2.4).

Hrúz et al. [120] developed a system for detecting failures in a small, two-seater aircraft. The inspection flights used a DJI Mavic 2 Enterprise Dual [251] UAV. A combination of a visual and a thermographic inspection was used to detect defects. Their approach uses MATLAB's *Deep Learning and Machine Learning for Computer Vision* library to detect failures on the acquired data. They use a segmentation process to isolate Points of Interest like rivets and screws. This allows for a simple presence detection by counting the number of detected rivets and screws and comparing them to the number of rivets that should be installed.

Ruiqian et al. [238] use a DJI M210 quadrotor [255] with a modular camera system to inspect a Xian Y-7-100 short-range turboprop airplane. The inspection is performed by navigating along a predefined series of waypoints that seem to be created manually. The UAV takes a series of images from different orientations at different altitudes with an overlap of approximately 60 %. The Software Pix4Dmapper [214] is used to create a photogrammetric reconstruction of the aircraft. To improve positional accuracy of the UAV for better reconstruction results, ArUco markers on the aircraft tracked in conjunction with GPS measurements. How the markers are referenced in relation to the GPS data is not described. Actual error detection still has to be performed manually in this approach.

Aleshin et al. [14] propose a combination of an autonomous ground vehicle (AGV) and a UAV to visually inspect an aircraft's outer surface while parked. In this approach, the UAV is initially positioned on the ground vehicle, which drives the UAV to the aircraft. The UAV is connected to the ground vehicle using a tether that supplies it with power from the AGV. The AGV automatically controls the length of the tether, so the UAV can perform its inspection flight without being hindered by its connection to the AGV. During the actual inspection, the UAV and AGV move together on their respective trajectories around the aircraft while the UAV captures images of the airplane using its gimbal-mounted camera. The paper focuses on modeling the tether mechanism intended to prevent collisions between the UAV and the aircraft in high wind conditions.

Miranda et al. [190] use a Donecle Iris UAV [61] to capture images of the exterior of an airplane and automatically detect defective screws using computer vision techniques. The UAV uses laser sensors to determine its position and navigate relative to the aircraft [50, 59]. A convolutional neural network is then used to extract and classify parts of the captured images that contain exposed exterior screws and other defects. When detecting screws, the authors achieve a recall and precision of more than 95 %. The model is also able to detect a variety of different screw patterns, which further facilitates the detection of defective screws.

Papa and Ponte [208] propose a system for detecting lightning strikes and hail damage on aluminum aircraft fuselage parts. They use a UAV equipped with a Raspberry Pi 2 Model B single board computer [93] and a Pi camera [92] forming the *Image acquisition subsystem*. Their work focuses on this image-capturing and error-detection system. The camera is placed at a fixed distance from the aircraft panel using a series of ultrasonic distance sensors. The captured images are then processed on the onboard computer, which automatically highlights detected errors like dents and lightning strikes before sending the captured images to the PC-based ground station. This ground station is

used for data collection and processing. The system's effectiveness was demonstrated in laboratory tests using damaged aluminum panels.

While the solutions presented in this section are able to perform inspections of aircraft to a certain degree, they all lack one or several features that allow them to perform these inspections in a production context. Due to their focus on in-service inspection, many solutions rely on GPS for establishing their position, do not support automatic path planning or only perform trajectories covering the entire surface of the assembly. While navigating independent of GPS signals, the size of the Donecle Iris might prevent it from maneuvering in complex manufacturing facilities or inside fuselage parts. Also, works like Papa and Ponte or Miranda et al. focus on the detection of specific errors rather than the positioning and navigation of the UAV performing the inspection.

1.3.2 In-Service Inspection of Infrastructure and Wind Turbines

Due to the disadvantages of manual inspections mentioned in Section 1.2.2, a variety of commercial solutions and scientific works exist that focus on automating the inspection of wind turbines. While a few specialized robotic inspection solutions exist in this field [6], the following sections will only cover UAV-based systems.

In industry, the company Aerones [5] offers various services concerning wind turbine maintenance and inspection. This includes automated cleaning and repairs using robots, applying anti-ice coating to rotor blades, and performing several types of inspections. In addition to UAV-based visual inspections, specialized robotic systems allow for inspecting the inside of turbine blades, the lightning protection system, and drainage holes. The company Flyability [241] also offers commercial wind turbine inspections using UAVs. Their services include visual inspections for cracks, leading edge erosion, delamination, lightning, and bird strikes. One of their specialties is the internal inspection of turbine blades using their Elios 3 UAV [239]. For legal reasons, industrial climbers can only inspect the first 28 m of the blades when working on their inside [241]. This causes problems, especially with larger turbines, where a significant part of the internal blade cannot be inspected manually. The Elios 3 UAV claims to fix this problem by being able to perform inspection flights inside the blade. It can do this by using an onboard LiDAR scanner and stereoscopic cameras that not allow for collision-free navigation in narrow spaces. Additionally, the *collision-resilient* design of the UAV, consisting of a lightweight protective cage structure that covers the entire UAV, aims to reduce damage to the UAV and its environment in the event of a collision. The sensors used for navigation also capture inspection data in the form of LiDAR measurements and RGBD images, which can be combined to create textured 3D models of the inspected structure.

In scientific literature, Moolan-Feroze et al. [192, 193] focus on the relative localization of a UAV performing an inspection of a wind turbine. The localization is performed visually using images taken by the onboard camera. The camera's location is estimated using a convolutional neural network and a 3D model of the wind turbine. This information is used in conjunction with data from the inertial measurement unit (IMU) and GPS sensor of the UAV which all serve as input for a pose-graph-optimizer, which combines

the three measurements. This approach was tested in simulation and reality using a wind turbine, and showed an increase in position accuracy compared to GPS alone.

Stokkeland et al. [265] also focus on determining the position of an inspection UAV relative to a wind turbine. Initially, the UAV navigates to a predefined, absolute target using its GPS sensor. This serves to position the UAV at an ideal position to start with the proposed visual positioning system, where the relevant parts of the wind turbine are visible in the onboard camera. This system determines the position of the UAV relative to the wind turbine. It uses machine vision and a Kalman Filter to calculate this position. The positioning works using Canny edge detection [33] and Hough line transform algorithms on the camera image to recognize the tower, hub, and turbine blades. Upon successful detection, the state of the Kalman Filter is updated, and the position is used in the position controller of the UAV. If the detection fails, the prediction of the Kalman Filter is used as position data instead. When no position can be detected in too many consecutive images, the UAV uses GPS to return to its starting point to reestablish a relative visual position from this position.

Like with the in-service inspection of aircraft, UAV-based wind-turbine inspections largely rely on the availability of GPS signals to establish the UAV's position. Approaches for visual positioning are highly specialized to the shape of wind turbines making their adaptation to other domains complicated.

1.3.3 Automated Inspection in Production

The UAV-based automated inspection in the production of large components is beneficial in a variety of different sectors. Since this work mainly focuses on the inspection of aircraft fuselage parts, this section presents the state of the art in this area. As mentioned in Section 1.2.1, inspections are an essential part of manufacturing for the aviation industry. However, various factors have caused the overall state of automation of these inspections to be comparably low. Regarding the use of UAVs for inspection in manufacturing, Maghazei and Netland [170] state that they are hardly used. Jordan et al. [128] do not even list production or manufacturing as a sector in a report about the state of the art of UAV inspections. In a review of aircraft structures diagnostics by Bardis et al. [21], in-service inspections are the primary subject compared to inspection in manufacturing due to the amount of available information about both topics. When UAVs are used in inspections inside factories, most UAVs are manually piloted, with automated UAVs still being in the research stage [171]. Malandrakis et al. [177] claim that, despite research on these topics, no studies have been reported in scientific literature yet. Nevertheless, some works exist using ground-based robots for inspections [153, 189, 244]. However, their mode of operation often prevents them from reaching places located above them, making them impractical for the inspection of large components. For this reason, UAVs dominate the field of inspection of larger structures as demonstrated throughout the previous sections. Their rare use in manufacturing might originate from additional problems in navigation, avoiding obstacles, and operating in proximity to human workers [140]. Also, the absence of GPS signals in most production facilities requires an alternative means for positioning the UAV compared to almost all previously

mentioned projects. Companies like Donecle [59, 61] use LiDAR sensors mounted on the UAV to determine its position relative to the aircraft. In addition, while inspection trajectories covering the entire surface of the inspected parts are also necessary in some cases, in production, the areas of an assembly that were modified and thus need to be inspected can be clearly defined. This allows for a more targeted approach when inspecting, resulting in the need for path and trajectory planning algorithms that selectively inspect the modified areas to save time. The number of errors that need to be detected is more diverse as well. Some types of defects that are checked in in-service inspections of aircraft also need to be monitored during production. This includes dents, scratches, and other paint defects, the presence and correct installation of rivets or screws [120], correctly sealed panel gaps [120], or delaminations in composite materials [46]. However, a large number of additional, often selective inspection types are required to check the numerous manufacturing techniques and steps used in the production of aircraft. The types of checks that need to be performed include the simple visual capture of the state of the assembly, visual presence detection of parts, and the verification of their correct installation. This verification can consist of simply checking the correct position and orientation of the part to more complex inspections like detecting, whether a cable or hydraulic line was routed through a certain cable clamp or if a washer was installed underneath a screw.

Malandrakis [177], Tzitzilonis [277] et al. present a system for inspecting aircraft wing panels using UAVs. The inspection performed is a visual inspection of the wing panel that has been treated with liquid penetrant prior to the inspection. In combination with a developer powder, this process reveals cracks and corrosion before anodising the panel. The UAV featured in this work has a digitally stabilized onboard wide-angle camera and a UV light source. The approach generates a grid of overlapping inspection-waypoints at a fixed distance from the surface to cover the entire area of the inspected Airbus A320 wing. The trajectory is then flown by horizontally moving through all inspection-waypoints line by line. The machine-learning-based error detection algorithm was trained on only 25 images containing defects, and the approach could achieve an overall accuracy of 97 % on the validation data. While the detection algorithm seems to yield good results, the trajectory planning seems to work only for simple, flat surfaces, and the rigidly mounted camera and UV light require the inspected part to be oriented in a specific way. Inspections from different camera angles seem impossible with the system's current state.

Deane et al. [56] focus on adapting state-of-the-art thermographic imaging inspection solutions, which are heavily used for inspecting the increasing number of composite materials used in modern-day aircraft. It focuses on pulsed thermography and vibrothermography, two types of non-destructive testing (NDT) methods in thermography. Both methods work by thermally exciting the tested part and measuring the resulting heat propagation through the material. This allows for the detection of problems with layer adhesion, either caused by manufacturing or mechanical damage, air inclusions, and other types of defects that cause a change in heat propagation inside the material. The main difference between the two methods is that pulsed thermography uses light pulses to excite the tested material from a distance, while vibrothermography uses

ultrasonic transducers in contact with the material. While the two types of inspections were successfully tested in a lab environment using a stationary thermal camera, the authors mention that the system yet has to be tested on a UAV. They mention difficulties obtaining a GPS signal inside the testing facility, combining multiple captured images, and creating a lightweight excitation source as potential problems. Automatic path planning and execution do not seem to be the scope of the proposed system.

Yasuda et al. [298] provide a systematic literature review of visual inspection processes for aircraft. The focus of the work is on differences in inspection processes in manufacturing and maintenance, methods for visual inspection and the autonomous navigation in this context. While the authors found a variety of different methods for visual inspections that sometimes can even be automated using computer vision techniques, using cameras for different spectrums like RGB, infrared, or X-ray, as well as 3D data obtained from Laser or ultrasonic scanners. The authors list complete surface coverage as the only method for trajectory planning, suggesting that selective inspection of components using UAVs has not been extensively covered in the reviewed literature. The literature focuses on detecting assembly errors, manufacturing defects, and damages inflicted by the use and wear of the assemblies and airplanes. The findings of the paper include a big potential for the automation of aircraft inspection using computer vision techniques, while emphasizing that the lack of requirement specifications for such inspection systems is one of the major factors hindering their adoption. The authors also state that finding a completely automated inspection system covering the entire inspection process was difficult. They emphasize the necessity for a modular system that is able to integrate with other systems.

1.4 Summary of Contributions

The contributions of this work were made while developing a platform for the autonomous inspection of large structures like aircraft fuselage parts. The system includes a path-planning solution specialized to the targeted inspection of individual parts of an assembly, as well as a means to generate efficient trajectories for inspections where the surface of the entire assembly must be covered. Additionally, a novel relative positioning and vision-based navigation approach that solely relies on internal sensors available on consumer UAV's was developed to determine its position relative to the inspected parts. This allows performing inspections in GPS-denied environments like inside manufacturing plants. Since various solutions for inspections using camera-based handheld inspection devices exist [100, 269, 286], this work does not focus on developing ways to perform conformity checking based on images provided by a camera. Instead, it is centered around the challenges that occur when trying to automatically move the sensor to the points that need to be inspected using a UAV while offering an interface for integrating arbitrary 3rd party inspection tools. For this purpose, a system for detecting obstacles using only limited available onboard sensor data, as well as their storage for later trajectory-planning, was developed. Using this information, two motion planning algorithms were adapted to allow obstacle-avoiding relative navigation in selective inspection scenarios, and their performance was compared. Also, user interfaces were developed for the specification of inspection points and the visualization of inspection results.

The efficacy of the individual components of the system were later verified with a number of assemblies and different UAVs in several proofs of concept and evaluations. Each proof of concept and evaluation was designed to verify the applicability of the system to at least one of the previously discussed case studies. A summary of the contributions resulting from the development of this system is listed in the following paragraphs.

- The concept, implementation, and evaluation of an inspection-centered path planning system is detailed in Sections 3.2, 4.1 and 5.2.1.
  - The concept of *Points of Interest*, *Viewareas*, and *Viewpoints* is introduced as a way to specify locations that must be inspected as well as areas or points from which the UAV can perform the inspection.
  - An efficient path planning algorithm for inspecting a series of inspection points is developed
  - This algorithm enables the generation of alternate inspection points which can be used for further optimization of the path or as alternative points to perform an inspection from in case the inspection from the first point fails.
  - An optimization strategy for reducing the number of inspected Points of Interest to reduce inspection path length and time is also detailed.
  - A further optimization strategy presented uses information gathered over the course of multiple similar inspections to optimize the inspection path through learning which points allow for a successful inspection.

  - Additionally, a user interface for specifying the inspected parts and generating an inspection path for Point of Interest-based inspections is developed.
- For determining the position of the UAV relative to the assembly, a novel approach that leverages model tracking and sensor fusion algorithms is presented in Sections 3.3, 4.2.1 and 5.2.4.
  - The relative positioning system uses model tracking to calculate a position relative to the assembly using only the onboard camera of the UAV.
  - The quality of this measurement is further improved by using it in conjunction with data obtained from the UAV's inertial measurement system in a Kalman Filter to calculate a more robust position.
  - The approach allows for the specification of Points of Interest relative to the inspected assembly, making the inspection independent of the absolute location of the assembly itself.
  - An extension of the system for the use with assemblies featuring several independently moving parts is also discussed in these sections.
- A system for detecting and persisting the state of obstacles using the UAV's onboard sensors is described in Sections 3.4.3, 4.2.2 and 5.2.5.
  - The proposed system is specially developed to work with low-quality obstacle sensor data provided by consumer UAVs.
  - Data of detected obstacles is stored in a space-efficient octree data structure creating a more detailed representation of the obstacles in the area.
  - Moving obstacles are handled by actively clearing areas that are no longer detected as occupied by the sensors using a ray-tracing algorithm.
  - Previously known obstacles such as the inspected assembly, fixtures or parts of the manufacturing facility can be inserted into the obstacle system in advance to optimize trajectory planning.
- Several collision-avoiding trajectory planning algorithms that leverage the developed obstacle detection systems are presented in Sections 3.4, 3.6.1, 4.2.3, 4.2.6, 5.2.5 and 5.2.6.
  - For selectively inspecting a set of Points of Interest on the surface of an assembly, the artificial potential field and Anytime Dynamic A* motion planning algorithms were adapted for the navigation of a UAV in a selective inspection scenario.
  - The Heat Equation Driven Area Coverage-Algorithm (HEDAC-Algorithm) is also used in its original form to precalculate trajectories covering the entire surface of the assembly for photogrammetric documentation of its state.
  - A modified version of the HEDAC algorithm is proposed with the intention of providing execution times fast enough for real-time execution in conjunction with feedback provided from the photogrammetric reconstruction process.

- For inspecting Points of Interest on the assembly, an interface for the integration of ground based inspection tools is proposed in Section 3.5.2. Additionally, a rudimentary system for visual presence detection of parts is proposed in Section 3.5.1 and two different approaches to the photogrammetric documentation of an entire assembly are described in Sections 4.2.6 and 5.2.6.
  - To enable the integration of the multitude of available manually operated visual inspection tools into the system, an interface for the action of inspecting a Point of Interest is proposed, and its functionality is demonstrated by the integration of the inspection tool VisionLib [286] for presence detection on assemblies.
  - Additionally, a concept for rudimentary presence detection based on pattern matching using renders of the inspected part is developed and tested using images captured during the inspection flights.
  - The images captured by inspection flights using the coverage-based inspection flights is computed into a textured 3D model of the assembly. For this task, a classic photogrammetric approach for reconstructing the geometry and texture of the model as well as a workflow, that uses the provided geometry of the CAD model is proposed, implemented and tested.

The remainder of this work is structured as follows: Chapter 2 details the fundamental techniques used throughout this work to develop the proposed system. The concepts that are based on the requirements of the inspections detailed in the two case studies are presented in Chapter 3. The implementation of these concepts and the resulting modular architecture of the inspection platform are then discussed in Chapter 4. Chapter 5 lists the Hardware and Simulation environment developed for the proofs of concept and evaluations that follow later in this section. These are used to validate the concepts presented in Chapter 3. Lastly, Chapter 6 summarizes the discussed topics and the research results obtained from Chapter 5. Furthermore, potential improvements to the system and future work are discussed in this chapter.

Summary. This chapter covers the technical foundation of the concepts presented in Chapter 3. The covered topics include planning methods, positioning and navigation techniques as well as collision avoidance.

2

Techniques for Visual Inspection

The automated inspection of assemblies using UAVs is a complex topic that requires knowledge in a variety of different areas. The following chapter will cover the technological foundation on which the concepts and implementation developed in this work are based. This includes an overview of path planning methods with a focus on path planning using the Ant Colony Optimization (ACO) algorithm. Afterward, the required techniques for a position estimation in GPS-denied environments are discussed by first defining coordinate frames and transformations and basic UAV positioning. Furthermore, position estimation using odometry and model tracking, as well as how these individual measurements can be combined using a Kalman Filter, are presented. Additionally, an overview of the required navigation strategies for performing an inspection flight is given. This includes Viewpoint-based algorithms for inspection of specific sections of the assembly and the coverage strategy used to perform a photogrammetric reconstruction of the assembly. To provide the necessary information for the obstacle-avoiding navigation strategies in Chapter 3, state-of-the-art sensors for collision avoidance and the storage of collision sensor data in the form of OctoMaps are discussed as well.

2.1 Planning Methods

In order to calculate an efficient inspection path during a selective inspection, the points visited by the UAV must be ordered in a way that results in a reasonably short path. This is required to reduce inspection time, which is important for the efficiency of the production process and also required to perform inspections with numerous parts as the UAV's flight time is limited.

The problem of visiting geometric points in a roundtrip while minimizing traveled distance can be viewed as a TSP (Traveling Salesperson / Salesman Problem) [23]. According to Biggs et al. [23] the TSP is an optimization problem in which a traveling salesman must visit a set of cities exactly once and return to the starting point while minimizing the distance traveled. Each city can be modeled as a node in a graph, with the edges between the nodes representing the connections between the cities and their weights corresponding to the distance between them. The problem is NP-hard [266], [98], which makes finding an efficient algorithm for an increasing number of cities difficult. While numerous approaches exist for solving the TSP, this work relies on the Ant Colony System (ACS) Algorithm because its potential for parallelization allows calculating a reasonably good approximation of the ideal path quickly [39].

2.1.1 Path Planning with the Ant Colony System

Ant Colony System (ACS) [97], [266] is a derivative of the Ant Colony Optimization (ACO) which are both is inspired by the behavior of ants trying to find the shortest possible paths from their nest to a source of food. In order to achieve this, Ants leave pheromones on their path when they have found a food source. Other ants follow this pheromone trail while leaving their pheromones. The probability of an ant following a specific path on a fork in the path is proportional to the intensity of the pheromone trail on the divergent paths. Over time, more and more ants follow the strongest/optimal pheromone trail while further increasing its intensity. Since more pheromones can be deposited on a shorter path than on a longer one in the same time interval the intensity of the pheromone trail increases most on the shorter path. In turn, the intensity of the pheromone trail decreases over time on less-used paths due to the natural decay of the pheromones. Compared to Ant Colony Optimization, ACS allows tuning the ants' behavior to either explore new paths or stay on known good paths. In addition, only the ant with the best path deposits its pheromone on the edges of its tour.

To apply this behavior to a mathematical model, all points that need to be visited are modeled as individual nodes in a graph [97]. The edges of this graph are populated with the intensity of the pheromone trail between the points. This pheromone level is defined for every edge c_ij in a graph $G = (V, E)$ using the pheromone update Equation (2.1).

$$\tau_{ij} = (1 - \rho) \cdot \tau_{ij} + \rho \cdot \Delta\tau_{ij}^{\text{best}} \tag{2.1}$$

With:

- τ_{ij} being the pheromone level on edge (i, j).
- ρ being the pheromone evaporation rate.
- $\Delta\tau_{ij}^{\text{best}}$ being the pheromone deposit from the best path found.

The pheromone evaporation rate is defined using Equation (2.2).

$$\Delta\tau_{ij}^{k} = \begin{cases} \frac{Q}{L_k} & \text{if } c_{ij} \text{ is on the path of ant } k \\ 0 & \text{otherwise} \end{cases} \tag{2.2}$$

With:

- Q being a constant related to the total pheromone deposited.
- L_k being the length of the path traveled by ant k.

Additionally, a local pheromone update system makes it less likely for other ants to visit previously visited edges.

$$\tau_{ij} = (1 - \rho) \cdot \tau_{ij} + \rho \cdot \tau_0 \tag{2.3}$$

With:

- τ_{ij} being the pheromone level on edge (i, j).
- ρ being the pheromone evaporation rate.

In the ACO algorithm, the probability of an ant taking the transition over an edge from node i to j would now be defined solely by an equation similar to Equation (2.5). ACS implements an additional decision rule that lets the ant choose an edge with a high pheromone intensity and a short length or use the probability-based selection strategy. Equation (2.4) defines the rules for this decision process.

$$j = \begin{cases} \underset{c_{ij} \in N(j^p)}{\arg\max} \{\tau_{ij} \cdot \eta_{ij}^{\beta}\} & \text{if } q \leq q_0 \\ J & \text{otherwise} \end{cases} \tag{2.4}$$

With:

- j being the next node to visit.
- τ_{ij} being the pheromone level on edge (i, j).
- η_{ij} being the heuristic information for edge (i, j).
- β being the parameter controlling the influence of the heuristic information.
- q being a random number used to decide exploration or exploitation.

- q_0 being the threshold for choosing exploration over exploitation.
- J being a randomly chosen node if exploration occurs.

The parameter q_0 decides whether to take the probabilistic approach or the best edge in terms of path length and pheromone level. It defines the probability of selecting the best edge. Upon selecting the probabilistic approach, p_{ij}^k defines the probability that an edge leading from node i to j is taken by ant k and is used to select node J.

$$p_{ij}^k = \begin{cases} \frac{\tau_{ij} \cdot \eta_{ij}^\beta}{\sum_{c_{il} \in N(j^p)} \tau_{il} \cdot \eta_{il}^\beta} & \text{if } c_{ij} \in N(j^p) \\ 0 & \text{otherwise} \end{cases} \tag{2.5}$$

With:

- τ_{ij} being the pheromone level on edge (i, j).
- η_{ij} being the heuristic information for edge (i, j).
- β being the parameter controlling the influence of the heuristic information.
- $N(j^p)$ being the set of candidate neighbors of the current node j^p.

During execution, the ants traverse the graph for a predefined number of iterations and ants according to the rules defined in the equations above. The choice of ants and iterations also influences the quality of the resulting path, and a trade-off between both parameters must be made as both increase the execution time. The directions the ants take eventually converge on a single short (but not necessarily optimal) path.

2.2 Positioning

A UAV must know its precise position to navigate the inspection trajectory when performing an inspection. Since GPS signals do not penetrate most buildings well enough, to enable accurate positioning and even in an outdoor scenario where GPS positioning is available, its accuracy might not be sufficient for the performed inspection task [133]. Hence, more advanced solutions are required, especially for inspections in a production scenario. Therefore, the following paragraphs introduce the basics necessary to establish the position of a UAV without external positioning systems. This includes coordinate frames, basic UAV control system theory, model tracking, and odometry to measure position, speed, and orientation, and the sensor fusion of these values. Lastly, kinematic models are introduced in preparation for their use in improving position estimation with respect to moving assemblies.

2.2.1 Coordinate Frames and Transformations

To position the UAV in 3D space, a base coordinate frame that defines the origin of the UAV as a discrete point and orientation is required. Since coordinate frames are used throughout this work to describe the position and orientation of various objects in 3D space, the following paragraphs first define coordinate frames before explaining the two most important frames on the UAV. Afterward, the cascaded control loops for position, velocity, attitude, and rate control inside a multirotor are described to generate a basic understanding of their onboard position control system.

Coordinate Frames

A Cartesian coordinate frame or Cartesian reference frame is mathematically defined by an origin point and a set of mutually perpendicular basis vectors [169]. The origin point defines a fixed point in space, $\mathbf{o}$, which is a reference for the coordinate system. The basis vectors define the coordinate axes of the frame. Mathematically, the origin $\mathbf{o}$ is defined as a point in $\mathbb{R}^n$. As this work exclusively works in $\mathbb{R}^3$, $n = 3$ is assumed for the rest of the definition. This results in $\mathbf{o}$ being defined as a three-dimensional vector $\mathbf{o} = \begin{pmatrix} o_x & o_y & o_z \end{pmatrix}^T$. The set of basis vectors $\mathbf{e}_1, \mathbf{e}_2, ..., \mathbf{e}_n$ therefore are comprised of $\mathbf{e}_1$, $\mathbf{e}_2$ and $\mathbf{e}_3$ or alternatively $\mathbf{e}_x$, $\mathbf{e}_y$ and $\mathbf{e}_z$ with $n = 3$. They are unit vectors (which means their length is normalized to 1), and orthogonal (perpendicular) to each other, which, in combination with their normalized length, makes them form an orthonormal basis for $\mathbb{R}^3$.

The Cartesian coordinate frame for $n = 3$ can be defined according to Equation (2.6).

$$F = \mathbf{o}, (\mathbf{e}_x, \mathbf{e}_y, \mathbf{e}_z) \tag{2.6}$$

Rotation Matrices

Combining the three basis vectors $\mathbf{e}_x$, $\mathbf{e}_y$ and $\mathbf{e}_z$ into a single matrix using Equation (2.7) results in a rotation matrix $\mathbf{R}$ that facilitates transforming points between coordinate frames [29].

$$\mathbf{R} = \begin{pmatrix} \mathbf{e}_x & \mathbf{e}_y & \mathbf{e}_z \end{pmatrix} \in \mathbb{R}^{3\times3} \tag{2.7}$$

Rotating a vector $\mathbf{v}_A$ defined relative to frame B into frame A is done using the rotation matrix $\mathbf{R}_B^A$.

$$\mathbf{v}_A = \mathbf{R}_B^A \cdot \mathbf{v}_B \tag{2.8}$$

In robotics, this principle can be applied to rotate vectors containing a position, velocity or acceleration between two base coordinate frames.

Rotation matrices can also be combined to calculate a rotation between frames for which no direct rotation exists. The rotation matrix $\mathbf{R}_C^A$ can be calculated using $\mathbf{R}_B^A$ and $\mathbf{R}_C^B$ according to Equation (2.9) [29].

$$\mathbf{R}_A^C = \mathbf{R}_B^A \cdot \mathbf{R}_C^B \tag{2.9}$$

Transformation Matrices

To describe the offset and rotation from a specific frame F_B to F_A, a transformation matrix $\mathbf{T}_B^A$ can be defined according to Equation (2.10) using the rotation matrix and a translation (or offset) between the two coordinate frames [169].

$$\mathbf{T}_B^A = \begin{pmatrix} \mathbf{R}_B^A & \mathbf{t}_B^A \\ 0 & 1 \end{pmatrix} \tag{2.10}$$

With:

- $\mathbf{R}_B^A \in \mathbb{R}^{3\times3}$ being the orientation of F_B relative to F_A.
- $\mathbf{t}_B^A \in \mathbb{R}^{3\times1}$ being the translation vector representing the position of F_A's origin in the frame F_B.

An arbitrary point $\mathbf{p}_B$ representing a position relative to frame B can be transformed into frame A using the transformation $\mathbf{T}_B^A$ as described in Equation (2.11) [169].

$$\mathbf{p}_A = \mathbf{T}_B^A \cdot \mathbf{p}_B \tag{2.11}$$

The same principle applies to derivatives of positions like velocities and accelerations.

If a direct Transformation from a frame D to another frame A is not available, a point $\mathbf{p}_D$ can be transformed to frame A through a chain of transformations between different coordinate frames that ends at the right reference frame like demonstrated in Equation (2.12).

$$\mathbf{p}_A = \mathbf{T}_B^A \cdot \mathbf{T}_C^B \cdot \mathbf{T}_D^C \cdot \mathbf{p}_D \tag{2.12}$$

Inverting a Transformation Matrix
Sometimes, it may be necessary to transform a point from F_B to F_A, but only the transformation $\mathbf{T}_A^B$ is known [169]. In this case, the inverse transformation $\mathbf{T}_B^A$ can be calculated based on $\mathbf{T}_A^B$ using Equation (2.13).

$$\mathbf{T}_B^A = \begin{bmatrix} (\mathbf{R}_A^B)^T & -(\mathbf{R}_A^B)^T \mathbf{t}_A^B \\ 0 & 1 \end{bmatrix} \tag{2.13}$$

With:

- $\mathbf{R}_A^B$ being the rotation matrix of $\mathbf{T}_A^B$.
- $\mathbf{t}_A^B$ being the translation vector of $\mathbf{T}_A^B$.

These techniques allow tools like the tf2-package of ROS [84] to build up a graph using a set of available transformations and use it to offer users the ability to find transformations between arbitrary frames as long as they are connected in the graph.

2.2.2 Kinematic Modeling of Assemblies

To navigate relative to structures with individual parts that move in relation to each other (like a wind turbine with a stationary base and rotating blades), a means of modeling this relative movement is required. Section 3.3.5 describes, how the Denavit-Hartenberg (D-H) convention [57] can be used to achieve this goal. The D-H convention is intended to model the kinematics of a robot by systematically assigning coordinate frames to the links and joints of a kinematic system. The convention standardizes the mathematical modeling of a kinematic chain, which makes it possible to describe the relationship between two adjacent joints as a set of four parameters.

Convention for Assigning Frames to Kinematic Links
The D-H convention describes the kinematics of a system defined by a set of links that are connected through joints. A series of interconnected links and joints is called a kinematic chain. The D-H convention standardizes the placement of reference frames, making it possible to derive the forward kinematics for the chain using a standardized procedure [169]. Each joint of the robot is assigned a coordinate frame, which is defined based on the frame of the previous joint. The Z-axis of each frame aligns with the axis of the joint's motion (rotation for revolute joints, translation for prismatic or linear joints). The frame's origin and the X-axis's orientation are determined depending on the relationship of the previous and current axis. If they intersect, the origin of the new joint is set at the intersection point, and the X-axis is set to be orthogonal to both the prior and current Z-axis. With non-intersecting, non-parallel axes, a common normal is determined for both Z-axes, which forms the new X-axis. The origin is defined as the intersection of the new Z-axis and X-axis. When the old and the new Z-axis are parallel to each other, the normal between both Z-axis that intersects the old origin forms the new X-axis with the new frame again being the intersection of new Z- and X-axis. Lastly, the y-axis is added so that the resulting frame forms a right-handed base.

D-H Parameters
When adhering to the rules for coordinate frames defined in the previous section, the relationship between two links of the robot described by frames F_{i-1} and F_i can be defined using the four D-H parameters:

- θ defines the joint angle as the rotation around the Z-axis.
- d defines the link offset as the translation along the Z-axis.
- a defines the link length as the translation along the X-axis.
- α defines the link twist as the rotation around the X-axis.

While a and α are usually fixed values based on the geometry of the link, θ or d may depend on the current joint angle in the case of a rotational or linear joint.

The transformation matrix $\mathbf{T}_{i-1}^{i}$ for Joint J_i from F_{i-1} to F_i can be defined using the D-H parameters as two separate transformations $\mathbf{T}_{i-1}^{i'}$ and $\mathbf{T}_{i'}^{i}$. These two transformations are defined by the rotation θ and translation d for $\mathbf{T}_{i-1}^{i'}$ as well as the rotation of α and translation a for $\mathbf{T}_{i'}^{i}$ according to Equations (2.14) to (2.16).

$$\mathbf{T}_{i-1}^{i'} = \begin{pmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & d \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{2.14}$$

$$\mathbf{T}_{i'}^{i} = \begin{pmatrix} 1 & 0 & 0 & a \\ 0 & \cos\alpha & -\sin\alpha & 0 \\ 0 & \sin\alpha & \cos\alpha & d \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{2.15}$$

$$\mathbf{T}_{i-1}^{i} = \mathbf{T}_{i'}^{i} \cdot \mathbf{T}_{i-1}^{i'} \tag{2.16}$$

After determining the individual transformations for each joint, the combined transformation of a kinematic chain $\mathbf{T}_0^n$ can be calculated using Equation (2.17).

$$\mathbf{T}_0^n = \prod_{i=n}^{1} \mathbf{T}_{i-1}^{i} = \mathbf{T}_{n-1}^{n} \cdot \dots \cdot \mathbf{T}_1^2 \cdot \mathbf{T}_0^1 \tag{2.17}$$

The techniques described in this section serve as a basis to the concept for modeling complex assemblies as kinematic chains to calculate their velocities through numeric differentiation of their positions described in Section 3.3.5.

2.2.3 UAV Positioning

While the term *UAV* commonly refers to all types of uncrewed or unmanned aerial vehicles or systems (UAVs / UASs) [301], this work mostly refers to multirotors or multicopters and, more specifically, quadrotors or quadcopters [234] when using the term. Compared to other UAV types, like fixed-wing aircraft, their ability to hover in place without movement and not require a minimum movement speed to generate sufficient lift for flight is essential for many inspection tasks.

The devices are usually powered by a lithium-polymer battery and have several motors attached to arms on which downward-acting propellers are mounted [115]. They are controlled by a radio remote control or fly autonomously [106]. A flight controller controls the position and attitude of the copter in the air by sending control signals to the motors' electronic speed controllers (ESC), which, in turn, drive the motors. It also processes the signals from the remote control and can navigate to preset waypoints autonomously.

To perform stable flight, it uses sensors like GPS modules, barometers, and external tracking systems [159] to determine its position and location. An IMU (Inertial Measuring Unit) uses the values of a gyroscope, a magnetometer, and an acceleration sensor to determine linear acceleration, rotational speeds, and orientation. By controlling position and attitude, multicopters can fly to and hover at any point in three-dimensional space.

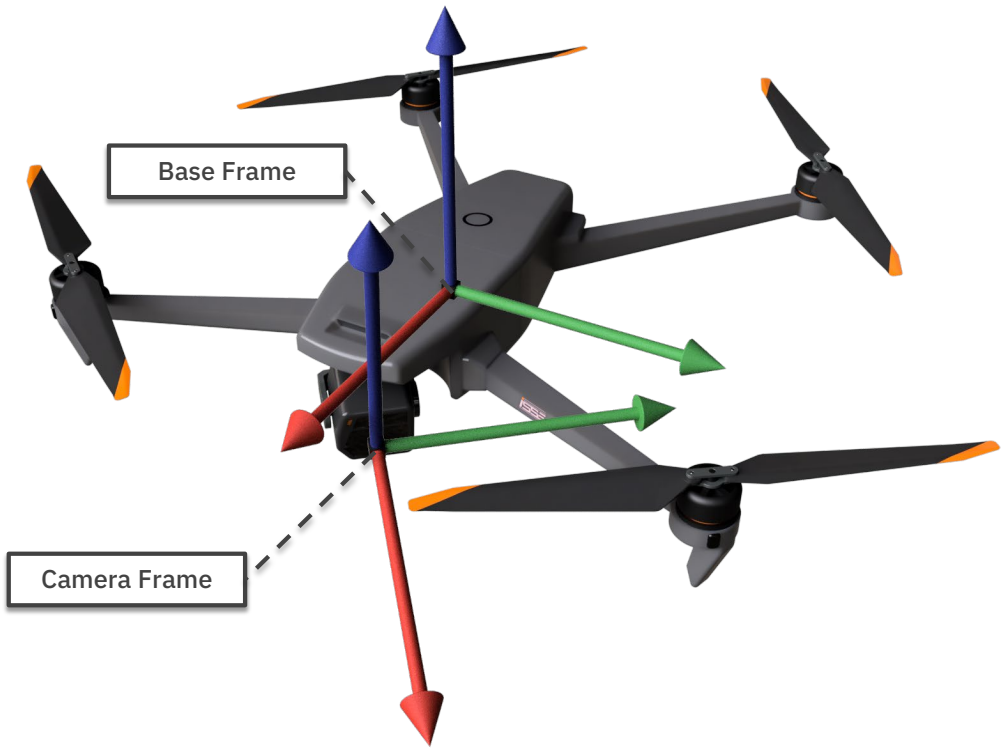


Figure 2.1. Coordinate frames and axes of a quadrotor. The red arrows correspond to the x-axis, green to the y-axis, and blue to the z-axis.

A UAV is either controlled manually by a remote control signal or automatically. Automatic control can be performed on an onboard flight computer or via a ground control station (GCS) that sends commands to the UAV via a radio transceiver and a protocol like MAVLink [219]. Additional mixed forms of control exist where only parts of the flight are performed autonomously.

In the following concepts, two frames located on the UAV are important for position control. Figure 2.1 shows the location and orientation of these frames. The base frame is located at the UAV's center of mass and fixed to its body. It indicates the UAV's position and is used for attitude and position control. The camera frame describes the camera's position mounted to the UAV gimbal. Its position is defined by the transformation $\mathbf{T}_{\text{uav}}^{\text{cam}}$ (see: Equation (3.9) in Section 3.3.2). These orientations are used internally by the developed system. In aviation and when working with UAVs in an outdoor environment, when positioning happens in global coordinates, the NED-convention [202] is used which results in the z-axis of the coordinate frame pointing down towards the ground and the y-axis of the base frame being flipped as a result. While this convention is useful for navigation with global positions, using the coordinate frame orientation depicted in Figure 2.1 proved less complicated. As a result, the driver implementation for the UAVs used in this project contains logic that converts the control commands from the frame orientation in Figure 2.1 to the NED-convention expected by the UAVs' flight controller.

In aviation, the x, y- and z-axis of the base frame of the aircraft are called the roll, pitch, and yaw axis. This work uses both conventions alternately, depending on which is more suitable for the current explanation. Additionally, ϕ, θ, and ψ describe the rotation angles around the x, y, and z axes, while the letters x, y, and z define a position.

Multirotor Control

The inherently unstable nature of a multirotor necessitates the use of a flight controller in order to control it by commanding positions or velocities [159]. In contrast, on an inherently stable fixed-wing aircraft, inputs on control surfaces more or less directly translate to its movements, with the individual control surfaces controlling one movement axis with minimal interaction. The rudder angle controls the yaw rate, elevator controls the pitch, engine controls the forward speed, etc.

Up until the development of electronically managed control surfaces and actuators for fly-by-wire systems, commercial aircraft were flown with their control surfaces and engine throttle linked directly (either mechanically or hydraulically) to the pilot's control inputs. This limited airframes to inherently stable designs. Work performed by NASA in form of the development of computer assisted fly-by-wire system for the Lunar-Landing Research Vehicle [26, 126] and the F8 fighter jet [67], enabled inherently unstable aircraft designs to perform computer-assisted stable flight. A notable example are jet fighters like the F14 series [233], which can only achieve stable flight throughout their entire operational speed ranging from relatively slow landing maneuvers to supersonic flight through the use of a sophisticated flight computer. Computer controlled fly-by-wire also enabled a stable atmospheric reentry and landing of the Space Shuttle despite its glide performance, which is often described by its pilots as similar to a *flying brick* [54].

Advances in computer assisted fly-by-wire systems also allowed software-defined stability control to be added to regular passenger airliners with the Airbus A320 being the first mass-produced aircraft of this type to be equipped with a digital fly-by-wire system [74]. This contributed to a reduced fatal accident rate caused by loss of control of Airbus

generation 4 aircraft (the first generation to be equipped with fly-by-wire technology) compared to previous generations. This type of accident, on average, occurred only 0.00 to 0.02 times per million flights in generation 4 aircraft throughout the last twenty years compared to a value between 0.06 and 0.09 of the previous generation in the same time-frame [9].

Software defined control and fly-by-wire do not just allow for more advanced and safe fixed-wing airplanes. It, can also be used to control arbitrary aircraft designs like the Lunar-Landing Research Vehicle [126] or multirotors. The only actuators on a multirotor are the individual motors that do not map directly to change in position or speed of a particular axis. Instead, the ratio of the rotation speed and, therefore, the thrust and torque produced by the individual motors in relation to each other cause a movement of the UAV. For the sake of simplicity, the following explanations focus on a quadrotor, but the foundational concept is similar for multirotors with different numbers of rotors. On a quadrotor, four rotors attached to four motors produce thrust in the downward direction. Changing the speed of all propellers to the same amount changes overall thrust and, therefore, a change in velocity in the z-axis [115]. Speed in the x- and y-axis requires the UAV to tilt in the pitch and roll axis, which requires a change in rotation speed, which can be achieved by creating a difference in rotation speeds of the motors on the opposite side of the relevant axis. Therefore, pitching forwards can be achieved by increasing the speed of the rear motors while decreasing the speed of the front motors by the same amount to keep the overall thrust the same. A change in the pitch also results in a forward velocity of the craft due to the forward component of the resulting thrust vector. However, the reduced upward component of the vector must be compensated for, through an overall higher thrust output, if no change in altitude is desired. Rotating the UAV around its yaw-axis can be achieved by creating a difference in torque in the motors. A quadrotor usually has two motors turning in the clockwise direction and two turning in the counter-clockwise direction. The motor pairs with the same turning direction are mounted to diagonally opposite arms. When all motors turn at the same speed, the individual torques they generate cancel each other out. Increasing the rotation speed of the motors turning in one direction while simultaneously reducing the speed of the other motors results in the sum of torques becoming positive or negative, which results in a rotation around the UAVs' yaw axis [115]. Combining all these constraints for motor speeds in a way that, for example, a yaw-rate input does not affect the UAV's pitch, requires a flight controller that generates motor signals based on the desired control inputs and performs higher-level tasks like position control. The following paragraphs give a simplified explanation of the cascaded control loops for rotational rate, rotation, velocity, and position as implemented in many flight controllers. This forms the core control loop that allows for position control using sensor feedback on a UAV [115].

Position and Velocity Control

When operating in position-control mode, the UAV receives an external position setpoint from either a waypoint that must be reached or through moving a target position through a remote control. Also, the current position can be fed to the controller to prevent the UAV from drifting over time through external influences like wind. The

position controller uses the difference between this setpoint and the current position to calculate an error value used to create an appropriate control output.

This output is a velocity setpoint for the underlying velocity controller (see: Figure 2.2). Like the position controller, the velocity controller uses feedback from a position sensor like a GPS receiver or an optical tracking system. In some cases, this data is combined with measurements from the IMU to calculate the current velocity. The current and target velocity difference is formed similarly to the position control loop to create an error input for the velocity controller. The output of the velocity controller serves as a setpoint for the attitude controller (see: Figure 2.3).

Attitude and Rate Control

The attitude controller uses the orientation calculated by the IMU to calculate an error based on the current attitude setpoint of the UAV [232]. Based on this error, a setpoint for the rate controller is processed [28]. The rate controller calculates target thrusts for each of the UAV's motors. Since the movement of the x, y, z and yaw position of the UAV is only loosely coupled [185], their individual thrust target values can be calculated in individual PID controllers for each axis and combined in a mixer node which passes the final control signal to the electronic speed controllers of the motors [270]. It uses

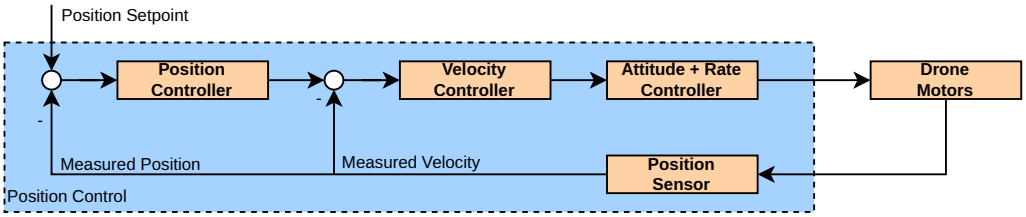


Figure 2.2. Position controller of a multirotor. The position error is fed into the position controller, which calculates a velocity setpoint. This velocity is used in conjunction with the measured velocity to calculate the velocity error which in turn is used by the velocity controller to calculate an attitude setpoint necessary to reach the target velocity.

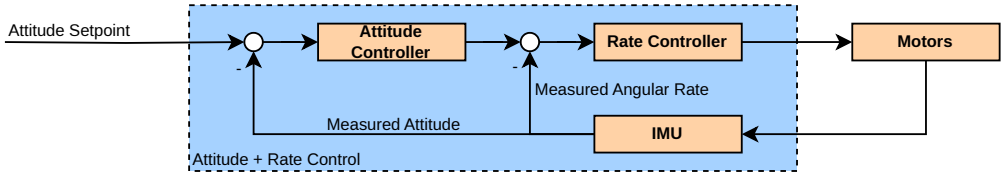


Figure 2.3. Attitude controller of a multirotor. The attitude setpoint is used in conjunction with the measured attitude to calculate the attitude error which is fed into the attitude controller. The attitude controller creates n angular rate setpoint which is used together with the measured rate to calculate the rate error for the rate controller. The rate controller calculates individual motor speeds needed to achieve the required angular rates.

the rotational rates measured by the gyroscope of the IMU and a setpoint provided by the attitude controller to calculate an error for each axis and adjust motor speeds accordingly.

While numerous solutions exist for implementing the individual controllers, PID controllers, which generate an output signal as a combination of a proportional, integral, and differential part of the error, are used in most cases [160]. The simplicity of their implementation and tuning make them suitable for this application. Note that while the preceding paragraphs only give a generalized overview of the mode of operation of a multirotor flight controller, they contain the necessary background information required for the concepts for positioning and navigation proposed in Section 3.3 and Section 3.4.

Camera Gimbal

Most modern camera drones use a gimbal to stabilize the camera image. The only exception to this are specialized FPV-drones (First Person View drones), which feature rigidly mounted cameras and steer camera motion through coordinated flight paths of the drone. However, for an inspection scenario, this limits the possible camera angles and trajectories unnecessarily, which is why this work focuses on UAVs featuring a gimbal-mounted camera. A gimbal mechanically counteracts the UAV's movement on a particular axis by rotating the camera by the same amount but in the opposing direction. Figure 2.4 depicts the design for a three-axis camera gimbal that is often used in consumer UAVs. Each of the individual axes of the camera can be controlled individually using a motor located inside the joints of the gimbal assembly. This results

Figure 2.4. Front mounted camera gimbal on a modern consumer UAV.

in a stabilization of the camera image to a point that the change in orientation of the UAV caused by its movement is unnoticeable. Using a software controller, each axis (pitch, roll, and yaw) can be set individually to a fixed angle either relative to the UAV or the global world frame. A typical configuration is to lock the horizon of the camera image by setting the gimbal to stabilize the roll angle of the camera to 0°relative to the world frame. The pitch is often set by the operator's remote to a desired angle relative to the world frame to be able to *look* up or down independently of the UAV's orientation. The yaw axis is often locked to the yaw orientation of the UAV, so it can be controlled by turning the UAV around its yaw-axis. In some cases, this coupling of the UAV and gimbal yaw is not fixed, but rather, the yaw-axis of the gimbal slowly follows the rotation of the UAV to smooth out the movement of the UAV, which is, in many cases, too abrupt for smooth video recordings.

2.2.4 Model Tracking

The term model tracking describes a variation of vision-based camera tracking that uses the video stream of a camera to calculate the camera's position relative to an object. To achieve this, model tracking tries to match the current camera frame to an orientation of a 3D model of the observed object. This process is mainly used in Augmented Reality applications where the goal is to overlay a live camera image taken by a camera with virtual objects like 3D models or additional information [144]. In contrast to marker or image-based tracking, where specially designed images called markers are tracked with the camera, model tracking relies solely on the geometry of the tracked object and tries to match it to a tracking model. This is done by extracting features like lines and points in the camera image and matching them to corresponding features in the model [144], which allows for calculating the camera position relative to the model. Using the basic principle of matching features to a 3D model to extract the camera's position relative to an object allows Augmented Reality applications to render an overlay over the physical object (see: Figure 2.5). Using the transformation between camera and object, the overlay can be rendered from the correct perspective to give the user the impression that the overlay is fixed to the physical object. These overlays often provide additional information relating to the physical object, depending on the context of the application. The applications of these techniques are vast and include production, remote maintenance [196], medicine [199], education [235], and marketing [22]. When used in the domain of assembly, these overlays can show workers where to install additional parts or how the assembly should look after installation [286].

The reason for choosing this approach over other visual tracking procedures like marker tracking is that the assembly itself does not need to be modified to work with the tracking system. Adding markers to the assembly that may need to be removed after inspection is an additional step that must be performed and slows down production. The positioning of the markers also needs to be sufficiently precise, so the manual installation of markers introduces the potential for errors. In addition, a marker-based approach would require the installation of multiple markers to track the model from all perspectives. When the UAV position changes or the camera moves around the object, an individual markers may not be visible anymore, and tracking is lost [262]. With

model tracking, the entire object functions as a marker. As long as enough geometric features of the object are in the camera view, it can be tracked. This makes this approach particularly attractive for a UAV-based inspection scenario with the inspected item used as the model target. This is because the camera views the assembly during the entire inspection, so tracking is possible throughout the whole flight. The need for a 3D model of the tracked part, which is sometimes seen as a disadvantage of model tracking, is not an actual problem in this use case because it can easily be derived from the CAD model of the part that is inspected.

2.2.5 Odometry

Another important technique for determining the position of a robot is odometry. The term odometry originates from the Greek words *hodos* (travel) and *metron* (measure) [76]. Therefore, it describes the estimation of a robot's pose through measuring the traveled distance. While there are numerous types of odometry, like wheel odometry, where the speed, position, and orientation of a robot is calculated using the rotation of its wheels, this work focuses on inertial odometry to help with navigation. For inertial odometry, an inertial navigation system determines a position relative to a starting point using an inertial measurement unit (IMU) as a sensor. The IMU is a system of sensors consisting of an accelerometer, a rate gyroscope, and sometimes a magnetometer (compass). The odometry data is generated by a system consisting of the IMU and computer or microcontroller that continuously calculates the estimated position and orientation relative to the robot's starting point using the sensor data [15]. This is done by integrating the accelerometer sensor values to calculate linear velocities and

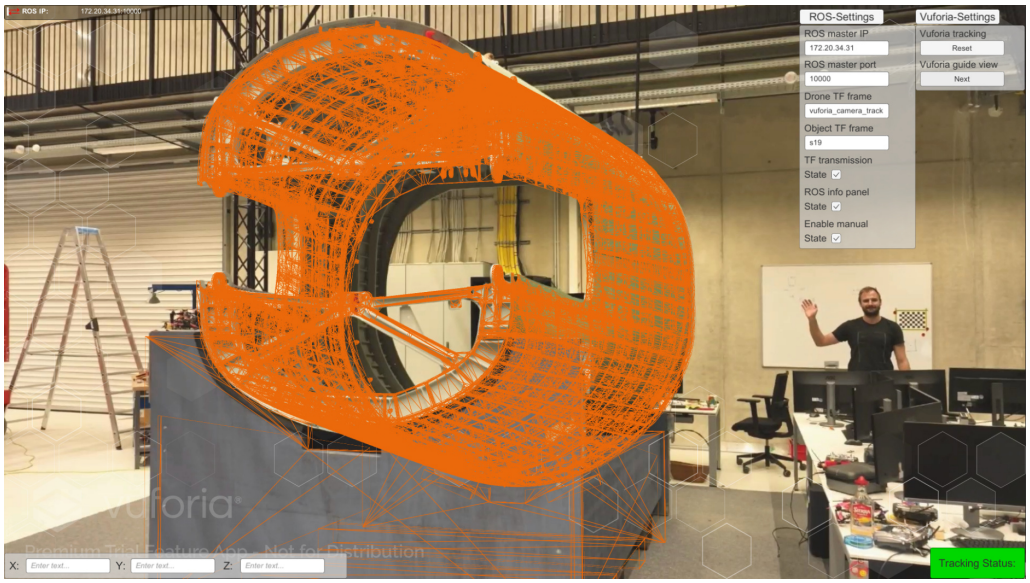


Figure 2.5. Screenshot of the overlay of the Augmented Reality application developed in this work using Vuforia Engine.

positions or integrating gyro rate data for angular positions. Since the simple numeric integration of this data is prone to accumulative errors [236], additional processing is done to estimate position and orientation better. An approach often chosen for the same purpose is filtering the sensor data using, for example, an Extended Kalman Filter [75]. The filtering allows the compensation of the drift introduced by integration of the gyroscope data by using the values of the accelerometer that can be used to determine the direction of the ground as a second input. To compensate for drift in the yaw-axis that the accelerometer cannot compensate, a magnetometer can be added to this system to reference this axis against the earth's magnetic field.

Inertial odometry is essential to achieving stable flight when using an inherently unstable quadrotor [48]. The measured orientation, angular rates, and velocities are important inputs for the rotation and rotational rate controller of the flight controller. This allows UAVs in FPV racing tournaments to perform agile and precise maneuvers using only a gyroscope and accelerometer as sensor inputs. However, with this type of UAV, the pilot manually controls velocity and position. Additional sensors like GPS are needed to compensate for the drift in linear velocity and position introduced by the IMU to achieve automatic position control. Still, the linear accelerations measured by the IMU can help improve the accuracy of the overall position estimate again using a Kalman Filter for the fusion of both measurements.

2.2.6 Sensor Fusion Using a Kalman Filter

After measuring estimated positions, orientations, and velocities from the UAV using various sensors, combining these different types of sensor data into a single estimated state of the system is necessary. The Kalman Filter [132] is often used as a sensor fusion algorithm to perform this task. It uses an internal model of the UAV's state to predict its expected behavior. This prediction is then used to rate the quality of the incoming sensor data. The quality of the sensor data and the internal prediction are used to determine the extent to which each updated sensor measurement influences the updated internal state vector. The Kalman Filter does this by calculating and constantly updating a covariance matrix of the incoming sensor data and its internal model and determining the weight with which sensor values and the internal prediction influence the final state estimation based on these covariances. During operation, predictions are performed periodically and updated using incoming sensor data. The following paragraphs describe how this process works in general, while Section 3.3.4 will go into further detail on how a Kalman Filter is used to estimate the UAV's position in the QuAD system.

Prediction of the State Estimate

As previously mentioned, the periodically performed prediction step predicts the system's behavior. To do this, it requires a state vector $\hat{\mathbf{x}}$ that contains all relevant parameters of the system. When estimating the position of a UAV, all linear and angular positions and velocities might be used to form a model of the system.

A state transition matrix $\mathbf{F}$ is necessary to predict the state vector's future value from a time step k to $k+1$. It is multiplied by the state vector to generate a predicted state

estimate and, therefore, defines how the existing state influences the next one. The state transition matrix is structured in a way that the next predicted position in all axes (linear and angular) is the sum of the old position and the numeric integration of the velocity (see: Equation (3.12)).

Optionally, the control inputs of the system can be considered by providing a control input vector $\mathbf{u}$ containing all control inputs for the system. It is multiplied by the control transition matrix, which defines how the elements of the input vector influence the system state. The result is added to the predicted state vector.

Through this process, it is possible to predict the state of the entire system using Equation (2.18).

$$\hat{\mathbf{x}}_{k+1}^{-} = \mathbf{F} \cdot \hat{\mathbf{x}}_k + \mathbf{B} \cdot \mathbf{u} \tag{2.18}$$

With:

- $\hat{\mathbf{x}}_{k+1}^{-}$ being the predicted state estimate at time $k+1$.
- $\hat{\mathbf{x}}_k$ being the state estimate at time k.
- $\mathbf{F}$ being the state transition matrix.
- $\mathbf{u}$ being the control input vector containing all input commands for the system.
- $\mathbf{B}$ being the control transition matrix.

This prediction is then updated with measurements of the individual sensors to calculate an updated state estimate $\hat{\mathbf{x}}_{k+1}$ during the update step. The weight in which both the predicted state and the measurement influence the updated state estimate is determined by the Kalman gain K_{k+1}, which is, in turn, calculated for each measurement using their corresponding covariance matrices. To calculate the covariance of the prediction $\mathbf{P}_{k+1}^{-}$, Equation (2.19) is used.

$$\mathbf{P}_{k+1}^{-} = \mathbf{F} \cdot \mathbf{P}_k \cdot \mathbf{F}^T \tag{2.19}$$

With:

- $\mathbf{P}_{k+1}^{-}$ being the predicted state error covariance matrix at time $k+1$.
- $\mathbf{P}_k$ being the state error covariance matrix at time k.

Update of the State Estimate

Every time a sensor measurement is obtained, the state estimate of the Kalman Filter is updated. To perform this update, first, a prediction of the expected measurement $\hat{\mathbf{z}}_{k+1}$ is calculated based on the state prediction $\hat{\mathbf{x}}_{k+1}^{-}$ calculated in Equation (2.18).

$$\hat{\mathbf{z}}_{k+1} = \mathbf{H} \cdot \hat{\mathbf{x}}_{k+1}^{-} \tag{2.20}$$

With:

- $\mathbf{H}$ being the measurement model that relates the state space to the measurement space.
- $\hat{\mathbf{z}}_{k+1}$ being the predicted measurement at step $k+1$.

It is also required to provide or calculate the measurement noise covariance matrix $\mathbf{R}$ for all measurements to determine their quality. This matrix is often provided by the sensor system itself. Since the Kalman Filter works on the assumption of a normal probability distribution of $\mathbf{R}$ (see: Equation (2.21)), it can often also be determined empirically by performing measurements of the sensor at rest.

$$p(v) \sim \mathcal{N}(0, \mathbf{R}) \tag{2.21}$$

The last value needed for performing the update step using a new measurement is the Kalman gain. It describes the ratio of the covariance of the internal state $\mathbf{P}_{k+1}^{-}$ over the sum of $\mathbf{P}_{k+1}^{-}$ and the measurement noise covariance $\mathbf{R}$. In other words, it indicates the proportion of the state variance in the total variance. Its exact calculation is described in Equation (2.22).

$$K_{k+1} = \mathbf{P}_{k+1}^{-} \cdot \mathbf{H}^{T} \cdot \left(\mathbf{H} \cdot \mathbf{P}_{k+1}^{-} \cdot \mathbf{H}^{T} + \mathbf{R}\right)^{-1} \tag{2.22}$$

With:

- K_{k+1} being the Kalman gain at time $k+1$.
- $\mathbf{H}$ being the observation model.
- $\mathbf{R}$ being the measurement noise covariance matrix.

Subsequently, Equation (2.23) is used to correct the prediction of the internal state with the measurement based on the Kalman gain.

$$\hat{\mathbf{x}}_{k+1} = \hat{\mathbf{x}}_{k+1}^{-} + K_{k+1} \cdot \left(\mathbf{z}_{k+1} - \mathbf{H} \cdot \hat{\mathbf{x}}_{k+1}^{-}\right) \tag{2.23}$$

With:

- $\hat{\mathbf{x}}_{k+1}$ being the updated state estimate at time $k+1$.
- $\mathbf{z}_{k+1}$ being the measurement vector at time $k+1$.

Lastly, the state covariance can be updated to represent the updated state using Equation (2.24).

$$\mathbf{P}_{k+1} = (\mathbf{I} - K_{k+1} \cdot \mathbf{H}) \cdot \mathbf{P}_{k+1}^{-} \tag{2.24}$$

With:

- $\mathbf{P}_{k+1}$ being the updated error covariance matrix at time $k+1$.
- $\mathbf{I}$ being the identity matrix.

Both the prediction and update steps can be performed at arbitrary times when the time interval since the last update or prediction is taken into account in the state transition matrix $\mathbf{F}$ and observation model $\mathbf{H}$, respectively. Also, updates of different sensors that measure various system parameters can be implemented by defining separate observation models and measurement noise covariance matrices $\mathbf{R}$ for each sensor. This allows for the Kalman Filter to predict an updated internal state and provide an estimation even if no updates are performed for a longer duration of time.

This behavior can be experienced when driving through a tunnel in a car while using a GPS-based navigation system, which often uses a Kalman Filter internally for position and velocity estimation. After entering the tunnel, position updates from the GPS sensors stop, and the position on the display is derived solely from predictions performed by the Kalman Filter based on the velocity the car was traveling before entering the tunnel. During this long phase without updates, the covariance of the state estimate continuously increases, representing an increasing uncertainty about the internal state. When exiting the tunnel, an update with a new GPS position measurement is performed. The high covariance of the internal state combined with a comparatively low covariance of the GPS-measurement results in a high Kalman gain and, therefore, a strong influence on the measurement on the updated state $\hat{\mathbf{x}}_{k+1}$. This results in the jumps in position that can often be observed when exiting a tunnel because the first GPS measurement drastically corrects the state estimation that has considerably drifted when only prediction was possible. Modern navigation systems or smartphones use data from their inertial measurement unit (IMU) in addition to the GPS signal to provide acceleration measurements. In this case, the acceleration is used to update the cars' velocity to provide a better position estimate and improve position accuracy when driving in situations without regular GPS updates. This is an example of using a Kalman Filter that uses multiple types of sensors to provide an improved state estimate.

Since the introduction of the Kalman Filter, several improved versions like the Extended Kalman Filter (EKF) [182], [66] and Unscented Kalman Filter (UKF) [129] have been developed. The main benefit of these variants is the support of nonlinear systems. While the EKF works well for systems with only a slight nonlinearity through linearization around the operation point using first order Taylor-expansion, the UKF uses the Unscented transform to work with highly nonlinear systems. Since both the EKF and the UKF are more complex to implement as well as more computationally intensive and the system presented in this work is only linear, the regular Kalman Filter was deemed sufficient to perform the sensor fusion.

2.3 Navigation Strategies

After a reliable position estimate of the UAV is calculated, a means of navigating towards a precalculated goal is necessary. For selective inspections of components, this work implements relative navigation strategies based on two different algorithms: Potential field navigation and Anytime Dynamic A*. In preparation for their adaptation to relative navigation in Chapter 3, the following paragraph details the Anytime Dynamic A* algorithm (the necessary fundamentals for the potential field method are introduced in Section 3.4.1). Afterward, the HEDAC algorithm [124, 125] is discussed in detail, as it is used in this work as a navigation strategy that allows for full surface coverage inspections.

2.3.1 Dynamic Path Planning Using Anytime Dynamic A*

The Anytime Dynamic A* (AD*) algorithm [158] is a heuristic, graph-based path planning and replanning method that quickly produces initial, possibly suboptimal solutions. AD* incrementally improves the solution through continuous optimization and efficiently repairs it when the environment changes (e.g., new obstacles are detected). These algorithms can be used for path planning in robotic applications. They sample discrete points in the environment and find the shortest path through a graph. As the name implies, the algorithm falls under the category of anytime algorithms. This means it can generate a (suboptimal) initial solution rather quickly and send it to the robot, which can start the execution before the final result is calculated. The algorithm continues to improve the solution during execution, constantly updating the robot with a better path until the ideal solution is found. The dynamic property allows the algorithm to react to changes in the environment and, therefore, handle newly detected obstacles without having to replan the entire path. This property makes it suitable for path planning in the QuAD Project. AD* combines the principles of the D* Lite algorithm [143], which allows for efficient incremental replanning, and the Anytime Repairing A* (ARA*) algorithm [157], which quickly finds an initial path that may not be perfect, and then improves it over time. AD* uses an inflation factor $\varepsilon > 1$ to find a fast, suboptimal solution initially. Over time, the calculation is repeated while continuously decreasing ε, thus improving the solution quality until $\varepsilon = 1$, which yields an optimal path.

Each node in the search graph is assigned a key, which determines when it should be expanded (visited by the algorithm) and how it should be prioritized compared to other nodes in the priority queue. Nodes with smaller keys are explored first, guiding the search effectively toward the goal. This key includes a weighted heuristic function $h(s, s_{\text{goal}})$, which is essential for the algorithm's efficiency and anytime behavior. The heuristic function estimates the cost from a given state s to the goal. In AD*, the heuristic is multiplied by an inflation factor $\varepsilon > 1$, which biases the search toward the goal and accelerates the planning process. A larger ε value allows AD* to find an initial path more quickly by expanding fewer nodes. This is achieved by trusting the heuristic more heavily. An approximation like the Euclidean or Manhattan (Taxicab) distance [147] can be used as the heuristic function $h(s_1, s_2)$. The algorithm begins with a high ε and decreases it over time to refine the path quality. The algorithm converges to an optimal

solution as $\varepsilon \to 1$. This allows for the continuous refinement of the solution. The search for the shortest path between a node s_{start} and s_{goal} is performed as a backward search starting from the goal.

The algorithm uses three lists to store nodes during processing (OPEN, INCONS and CLOSED). The OPEN list contains all nodes that still need to be expanded and is initialized with s_{goal}. During computation of a path, neighbors of the currently expanded node are added to the list, and it determines, which node is expanded next. Each node in the graph is assigned a key, using the function $key(s)$, defined in Equation (2.25), which determines the order in which the nodes are removed from the open list.

$$key(s) = [k_1(s), k_2(s)] = \begin{cases} [rhs(s) + \varepsilon \cdot h(s_{\text{start}}, s), rhs(s)] & \text{if } g(s) > rhs(s) \\ [g(s) + h(s_{\text{start}}, s), g(s)] & \text{otherwise} \end{cases} \tag{2.25}$$

With:

- s being the current node (or state) in the graph.
- s_{start} being the start node of the search.
- $g(s)$ being the current cost of the shortest path found from s_{start} to s.
- $rhs(s)$ being the one-step look-ahead value (minimum expected cost to reach s).
- $\varepsilon \geq 1$ being the inflation factor used to regulate the quality of the solution.
- $h(s_{\text{start}}, s)$ being the heuristic estimate of the cost to reach s from s_{start}.

The key is used to define a lexicographic order for the OPEN list according to Equation (2.26), which is in turn used to order the open list.

$$key(s) < key(s') \Leftrightarrow k_1(s) < k_1(s') \vee (k_1(s) = k_1(s') \wedge k_2(s) < k_2(s')) \tag{2.26}$$

The second list is the INCONS (inconsistent) list, which stores nodes that have been expanded but whose g-values have been improved after expansion. These nodes are inconsistent because their recorded g-values are now better than what was used during their expansion, meaning their current state may lead to a better solution than previously thought. The CLOSED list tracks nodes that have already been expanded in the current iteration of the search.

Calculation of the Initial Path

Initially, both $g(s)$ and $rhs(s)$ are set to $\inf$ for each node s except for the value of $rhs(s_{\text{goal}})$, which is set to 0. s_{goal} is then added to the INCONS list as well as the OPEN list and therefore expanded first. For each node in the open list, the one-step look-ahead value $rhs(s)$ is calculated using Equation (2.27).

$$rhs(s) = \begin{cases} 0 & \text{if } s = s_{\text{goal}}, \\ \min_{s' \in Succ(s)} \left(c(s, s') + g(s') \right) & \text{otherwise}, \end{cases} \tag{2.27}$$

With:

- $Succ(s)$ being the set of successor states of s.
- $c(s, s')$ being the edge cost from s to a neighboring node s'.
- $g(s')$ being the value function at s'.

The one-step look-ahead value $rhs(s)$ is then compared to $g(s)$ (current cost of the shortest path found). If $g(s) > rhs(s)$, $g(s)$ is set to $rhs(s)$ (updating the cost of the shortest path), the node is closed and the state of all neighboring nodes is updated. Otherwise, $g(s)$ is set to ∞ and its state is updated along with all neighboring nodes. Updating the state of a node updates its rhs value according to Equation (2.27) and removes it from the OPEN list (if it is in the OPEN list). If $g(s) \neq rhs(s) \wedge s \notin CLOSED$, it is added to the OPEN list again. Otherwise, it is inserted into the INCONS list. This procedure then expands nodes incrementally, always using the node of the OPEN list with the smallest key. The algorithm continues this behavior until $rhs(s_{\text{goal}}) = g(s_{\text{goal}})$ or the smallest key value in the open list is smaller than $key(s_{\text{start}})$, which indicates that a path from s_{start} to s_{goal} was found.

Refining the Path and Replanning Around Obstacles

The robot or UAV can then start navigation using this plan. To refine the path, ε is decreased, and the search is repeated. The smaller ε causes fewer nodes to be skipped due to a lower trust in the heuristic metric, so the resulting path quality improves. Eventually ε becomes 1, which results in the calculation of the optimal path. When changes in the environment occur, only affected states are updated. The algorithm incrementally repairs the plan by modifying inconsistent states. This avoids full replanning from scratch, resulting in better computational performance compared to similar approaches. In the algorithm, this is done by setting all edge costs to the node blocked by the obstacle to ∞ and invalidating all nodes whose cost depends on the affected node. Updating only the affected parts of the graph is done by adding only these nodes back to the OPEN list and expanding them again. If changes in the area are significant, ε is temporarily increased to encourage faster replanning.

Application to Obstacle-Avoiding Navigation

To apply AD* to UAV path planning in dynamic environments, a graph on which the search can be performed has to be created first. Since the number of possible positions in

3D space is infinite, the UAV can theoretically occupy any point (x, y, z) in $\mathbb{R}^3$. However, graph search algorithms like AD* (and A*, D*, etc.) operate on discrete graphs with a finite number of nodes. Hence, a way to approximate the continuous world with a finite, discrete representation is needed. This is usually achieved through spatial discretization in a voxel grid, where the 3D environment is divided into a grid of voxels of fixed size r. The center of each voxel becomes a graph node, and the UAV is assumed to be able to move between adjacent voxels. This gives a finite number of states for the search. Depending on the application, a set of neighboring nodes to which the UAV can move is defined as the neighborhood. Common neighborhoods are 6, 18, or 26 neighbors in 3D space. The neighborhood is chosen based on the vehicle's movement constraints and the available computing power, as an increasing number of neighbors increases computation time. Obstacles can also prevent the UAV from reaching particular neighbors. This set of neighbors for all nodes forms the edges of the graph. If needed, the UAV's velocity and orientation can also be included in the state space. Obstacles can be represented using a 3D occupancy grid obtained from a LiDAR map or OctoMap. This allows for voxels to be marked as free or occupied. Pruning transitions from and to occupied nodes prevents the algorithm from planning paths through areas occupied by obstacles.

2.3.2 Strategies for Surface Coverage

Compared to a selective inspection where only modified components or other Points of Interest are visited, some applications may require inspecting and capturing the state of the entire assembly. This can be necessary every few production steps to ensure existing parts of the assembly were not damaged while mounting new parts or before an assembly leaves a factory to document its state. The process creates evidence of the part's state if it gets damaged during transport or further down the manufacturing chain. This type of inspection is also required for in-service inspections of products like wind turbines or aircraft. Compared to an inspection context where the location of the Point of Interest can be easily determined, in-service inspections need to look for damage like cracks, bird-strike, or delaminations on the entire product surface. This requires a navigation strategy that provides full surface coverage of the inspected product to capture data from all angles.

The HEDAC Algorithm

One of the algorithms suitable for generating a trajectory covering the entire surface of a part is the Heat Equation Driven Area Coverage-Algorithm (HEDAC-Algorithm) [124, 125]. The approach allows for coordinated 3D visual inspection of complex geometries by multiple UAVs. The algorithm is not based on waypoints but instead uses potential fields to lead one or multiple UAVs to the areas of the assembly with the least amount of coverage. Repelling potentials are used as an obstacle avoidance strategy to repel the UAV from the assembly and other nearby UAVs. This allows the inspection to occur at a predefined distance from the object's surface. Coverage is modeled using a potential field based on a Gaussian function around the inspected obstacle based on the desired inspection distance and around each UAV's trajectory to represent already covered areas.

The cumulative effect over time results in a coverage density field $\rho(x, t)$, representing how well the space has been observed. The algorithm's goal is to commandeer the UAVs in a way that minimizes the discrepancy between the desired target inspection density $\mu_0(x)$, which is the strongest near the structure, and the actual coverage $\rho(x, t)$. Based on the difference between $\mu_0(x)$ and $\rho(x, t)$, a scalar potential field ψ representing the areas with insufficient coverage is computed. The movement of each UAV is dictated by the direction of the gradient of the generated potential field.

$$\mathbf{v} = -\nabla\psi(x) \tag{2.28}$$

With $\mathbf{v}$ being the target velocity vector for the UAV. This ensures coverage of under-inspected areas in a smooth and coordinated fashion. To avoid collisions and boundary violations, each UAV solves a local trust-region optimization problem that slightly adjusts the velocity vector to avoid collisions with other UAVs or obstacles while simultaneously staying inside the specified inspection area. For calculating camera orientations, the distance field containing the Euclidean distance to the nearest point on the structure's surface is used. The gradient of this field is used as a direction vector to determine the camera orientation. This ensures effective visual coverage of the structure's surface.

The amount of surface coverage is also evaluated to determine the progress of the inspection. For this, the covered surface area of each UAV's camera is modeled as a cone with the tip located at the camera's position. The taper of the cone is determined by the camera's field of view. Based on this cone, ray tracing is used to determine which mesh nodes of the component's surface are visible during each time step. A coverage counter tracks the number of views per surface node. The algorithm provides a unified trajectory planning and coverage control solution and handles detailed, non-convex geometry. The resulting trajectories ensure a smooth, adaptive motion with natural collision avoidance. The authors claim that the algorithm in its current form is not designed for real-time onboard use due to its computational complexity.

2.4 Collision Avoidance

Due to the complex environments a UAV can encounter in an industrial inspection context, robust obstacle avoidance is necessary to perform an inspection safely. The dynamic nature of a production facility makes it infeasible to consider and model all possible obstacles the UAV can encounter in advance. Therefore, to avoid obstacles, it must be equipped with onboard sensors to detect and avoid them. The following paragraphs give an overview of the types of obstacle sensors used on UAVs as well as the use of OctoMaps [201], [116] to store information about detected obstacles, while the strategies used for collision avoidance are detailed further in Chapter 3.

2.4.1 Sensors for Collision Avoidance

Several sensor types exist for detecting obstacles with a UAV. The three types that are commonly used in UAVs today are ultrasonic distance sensors, LiDAR sensors, and stereoscopic cameras.

LiDAR (light imaging, detection, and ranging) sensors emit laser light and measure the time it takes for the light to return to the sensor [297]. They are used in robotic applications for obstacle detection and mapping. They calculate the distance to an object through the time of flight (ToF) principle, which emits a laser pulse and measures the time it takes for the light to reflect back from an obstacle. The distance between sensor and obstacle d can then be calculated using Equation (2.29).

$$d = \frac{c \cdot t}{2} \tag{2.29}$$

With:

- c being the speed of light.
- t being the time between sending the pulse and detecting the reflection.

The division by two is needed because the light must get to the obstacle and then back again to the sensor, so it travels twice the distance between sensor and obstacle. In 2D- or 3D LiDAR Sensors, the sensor often scans multiple angles by deflecting the signal through a mirror that can rotate and/or tilt [229]. This allows for obstacle measurements in one (for 2D LiDAR sensors) or multiple (for 3D LiDAR sensors) planes.

While they offer measurements with great accuracy and frequency, the mechanically rotating mirror assembly often makes them heavy and expensive compared to other sensors, preventing their use in lightweight or inexpensive consumer UAVs.

Sonar Sensors
Sonar sensors apply the same ToF principle of LiDAR sensors but use ultrasonic sound pulses instead of light to estimate a distance [297]. Similarly to ToF Laser Sensors, the distance to an object d can be calculated using Equation (2.30).

$$d = \frac{v \cdot t}{2} \tag{2.30}$$

With:

- v being the speed of sound.
- t being the time between sending the pulse and detecting the echo.

However, they lack the ability to precisely reflect the sound impulse in different directions as LiDAR sensors do with the mirror assembly, limiting their use to one direction in most cases. The emitted sound waves also move away from the sensor in a wide cone compared to the focused beam of laser light. This causes a wider detection angle, which can be beneficial in cases where this behavior is desired (e.g. in park distance control sensors in cars). The significantly slower speed of sound can also limit the measurement frequency because all echoes of the previous pulse need to decay before a new pulse can be emitted without detecting an echo from the last pulse.

Stereoscopic Cameras
Reduction in size and cost for cameras and the hardware needed to process stereoscopic images have caused stereoscopic cameras to be widely spread for detecting obstacles in UAVs. Their main advantage is their low weight and energy consumption as well as their compact size [297]. However, in contrast to LiDAR sensors, they are highly dependent on ambient light, good weather conditions and require high levels of processing power. However, recent advances in embedded microprocessor technology allow even mid-level consumer UAVs like the DJI Mini 4 Pro [257] to feature multiple stereoscopic cameras that provide omnidirectional obstacle detection at a takeoff weight of less than 249 g. The technology works by placing two cameras at a fixed distance from each other to mimic human binocular vision. Capturing two images from slightly different positions allows for depth perception in the images. Closer objects shift more between the two images, while farther objects shift less. This effect, called parallax, is used to calculate depth. After capturing both images and correcting effects caused by lens distortion, various algorithms can be used to compare both images and find the same features in both. Examples of algorithms used for this task are Sum of Absolute Differences (SAD) [107], Normalized Cross-Correlation (NCC) [303], and Semi-Global Matching (SGM) [113]. Afterward, the disparity d, being the shift of a feature from left to right, is calculated. Using

the disparity, the depth information of this feature can be calculated using Equation (2.31).

$$Z = \frac{f \cdot B}{d} \tag{2.31}$$

With:

- Z being the depth (distance to object).
- f being the focal length of the camera.
- B being the baseline distance (distance between the two cameras).

2.4.2 OctoMaps

After detecting relevant obstacles, a means of storing their position is required to create a representation of free and occupied space around the UAV. OctoMaps [201], [116] are a 3D occupancy mapping framework based on octrees [183] that fulfill this task.

An octree is a hierarchical geometric data structure that divides space dynamically only where required which reduces memory usage compared to a dense grid. An octree is structured in a tree data structure with each node having either eight or zero child nodes. Each node occupies a cubic volume of space called a voxel. A voxel can either be occupied (if an obstacle was detected inside), free (if known to be empty) or unknown if it was not explored yet. Since real-world sensor data can often be unreliable, it is also possible to assign a probability for the occupancy based on the quality of the sensor measurement. The resolution of the map depends on how deeply the octree is divided.

Octrees work by dividing the 3D space in increasingly smaller sections. While the root node represents the entire 3D space, if needed, it is split up into eight child nodes refining the occupancy mapping where necessary. A node may be recursively subdivided into eight child nodes at sections of the map where a more detailed distinction between occupied, free and unknown space must be made [116]. This increases efficiency, because large chunks of identical space can be stored at a coarse resolution, while complex areas are represented in more detail.

The octree can be updated through new obstacle sensor data. To update the tree, the sensor measurement is translated into a position in 3D space by using the position of the UAV and casting a ray of the length of the distance measurement in the direction the measurement was taken at. The voxel at the end of the ray is marked as occupied while all other voxels the ray intersects are marked free. All other areas remain unchanged.

Summary. This chapter presents the conceptual basis for UAV-based inspection in the production environment. This task requires path-planning options to reach inspection points efficiently. In addition, flying in production halls poses challenges when determining the UAV's position. Navigation strategies must also be found that enable collision-free navigation relative to the assembly.

3

Concept for Relative Inspection

This chapter covers the concepts underlying the relative inspection system developed throughout this work. After a general overview of the concept, the underlying terminology needed to understand the following detailed concepts is defined. The first is calculating an efficient path, allowing the UAV to inspect all relevant parts of the assembly. This process is done entirely offline (meaning before the inspection flight), which allows for extensive optimizations, which are also discussed in detail. After offline planning, the concept of all aspects that are needed to perform the actual inspection flight are discussed. First, a way to establish the UAV's position relative to the assembly is needed. Once the position is established, navigation strategies for navigating relative to the assembly while avoiding obstacles are developed. To do the latter, a method for detecting and persisting knowledge about obstacles is presented that allows for them to be used in the navigation strategies, even if they are not currently detected by any of the UAV's sensors. All these concepts are designed to work in conjunction with relative localization of the UAV. Lastly, techniques for performing actual inspection tasks using only the sensors available on lightweight consumer UAVs are presented.

3.1 General Overview of the Concept

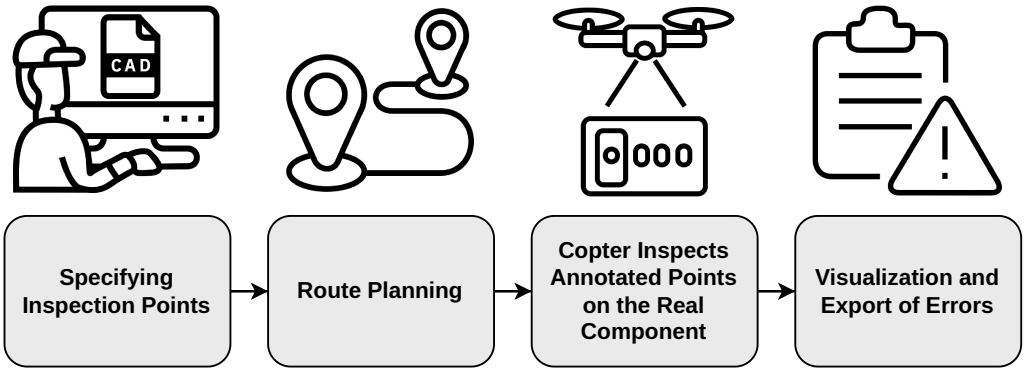


Figure 3.1. Overview of the system architecture.

Figure 3.1 shows a general overview of the concept developed in this work. The focus of this work is set on trajectory planning and relative navigation as well as performing relative inspection with UAVs. This includes a means of specifying the parts to be inspected and the analysis of the inspection results. Therefore, the depicted workflow starts with an engineer using the CAD file of the assembly to specify points that need to be inspected after a certain production step. Having a CAD file to work from has become standard procedure in modern manufacturing [121], so no additional data is needed apart from what's already required for production. The specification of inspection points happens in a tool that allows the export of the added information in a format that the path planning algorithm can process. This algorithm uses this information to generate areas from which the inspection points can be seen and, therefore, areas in 3D space that the UAV must visit to perform the inspection successfully. Afterward, distinct points inside these areas are chosen for the UAV to visit. Alternative points are also generated if the UAV can not reach one of the primary points. All points contain a position for the UAV to reach and an orientation for the UAV's camera. These orientations are calculated in a way that ensures that the camera points directly at the object that must be inspected. In the last step of the planning phase, all points that the UAV must reach are ordered in a way that results in the shortest possible inspection path. During the inspection, the UAV navigates to the points specified by the path planner in the predefined order. It does this while determining its position relative to the assembly that must be inspected and while avoiding obstacles in its way. A live video feed is streamed to the inspector, and high-resolution pictures and videos are taken during the inspection. The UAV remains at a certain inspection point until all inspection tools confirm that the current part was inspected successfully. After the inspection, the captured image and video data is automatically retrieved from the UAV and stored together with the results of the automated inspections performed during the flight. The inspector can then review this data with the help of another tool that uses the geometry derived from the CAD model to help find recorded video segments, images, or inspection results of specific components. The following sections will cover all parts of this process in more detail.

3.2 Viewpoint-Dependent Path Planning

For a UAV to perform an autonomous selective inspection of an assembly, an efficient path that allows it to inspect all necessary components has to be calculated first. To do this, the following sections first define essential terminology for the concepts that follow before covering the developed algorithms for determining an efficient inspection path. In addition, further optimization strategies for this path are discussed, and an overview of the current state of the art is given.

3.2.1 Points of Interest, Viewareas, and Viewpoints

The following paragraphs define several terms specific to the path planning system presented in later sections of this chapter. The definition is consistent with previous work [246–248, 293]. Figure 3.2 gives an overview of these terms. In this figure, one-half of a tubular aircraft fuselage section is depicted. Various parts like brackets, stringers or bolts are installed on the assembly during manufacture.

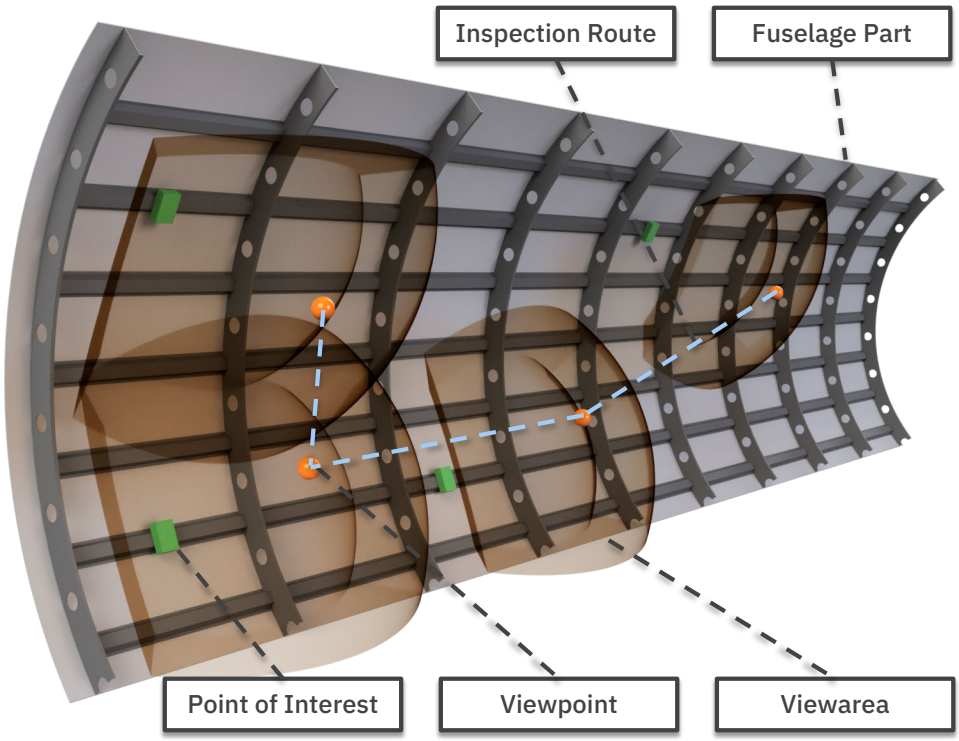


Figure 3.2. Overview of the terminology used throughout this work. Viewareas specifying areas from which Points of Interest can be inspected. Viewpoints as discrete points inside the Viewarea and a path connecting multiple Viewpoints.

Points of Interest (POIs)
After each production step, the correct installation of the added parts must be verified. Each point at which this verification must be performed is defined as a *Point of Interest* (POI). Examples of a Point of Interest are brackets that were installed during manufacturing, rivets (either a single rivet or a whole set), welds that must be inspected or a screw connection. The physical object the Point of Interest describes is arbitrary. It is only defined by its coordinates in 3D space x, y, z, its orientation as a quaternion q_x, q_y, q_z, q_w, its dimensions w, h and the maximum deviation angle φ_{max}. Figure 3.3 helps with

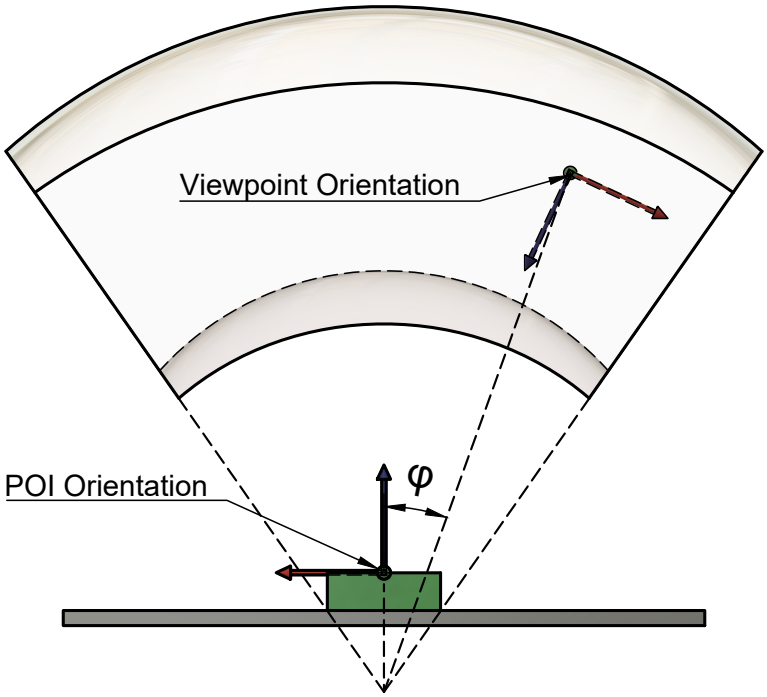


Figure 3.3. Schematic two-dimensional depiction of the deviation angle φ between the camera axis and the normal of the Point of Interest.

the definition of the deviation angle φ. It describes the angle between the normal vector of the Point of Interest and the optical axis of the camera. In other words, it indicates the angle from which the camera views the Point of Interest. The maximum deviation angle describes the maximum acceptable lateral angle for the inspection. Using a small maximum deviation angle allows the inspector to restrict inspection of a part to only a few degrees off the specified orientation, while a larger φ_{max} allows inspection from a more lateral orientation as well. This allows for more possible UAV positions for the inspection. In addition to the previously mentioned parameters, an ID is specified together with a semantic annotation to identify the Point of Interest.

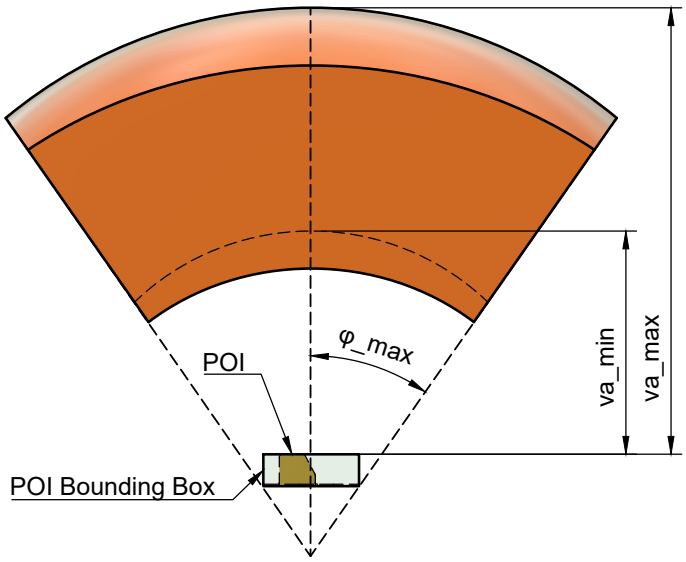


Figure 3.4. Example of a Viewarea for an arbitrary Point of Interest defined by its minimum and maximum distance from the Point of Interest and the maximum viewing angle.

Viewareas

To inspect a Point of Interest, a specification of the area it can be inspected from is needed. Therefore, the concept of *Viewareas* (VAs) is introduced here. It defines a volume in space from which inspection can be performed. Each point within this volume is assigned an orientation for the UAV and camera, which ensures alignment with the Point of Interest. It is described through a minimum and maximum radius va_{min} and va_{max} as well as the size and the maximum deviation angle φ_{max} of the corresponding Point of Interest (see: Figure 3.4). The maximum radius is defined as a constant distance from which the camera can realistically inspect the part while va_{min} is the maximum of the minimum distance the UAV must keep from the part for safety reasons and the minimum distance that is required for the part to be seen by the camera. The latter is calculated from the field of view of the camera and the dimensions specified in the Point of Interest. Its general shape is defined by subtracting the volume of two spheres, with radii of va_{min} and va_{max} resulting in a hollow sphere. The center of these spheres is located at a point behind the Point of Interest that is determined by the point at which four planes that each intersect one of the four outside edges of the bounding box of the Point of Interest rotated by the maximum deviation angle intersect (see: Figure 3.5). These four planes also define the outer boundary of the Viewarea in the horizontal and vertical direction. The Viewarea is created by discarding the area of the hollow sphere that is not enclosed by these planes. The bounding box is defined by the rectangle with the dimensions w and h, with the Point of Interest position and orientation as the center. While one edge of the bounding box is used as a vector for describing each plane, the other axis is a rotation $+$ or $-$ half of the camera's horizontal

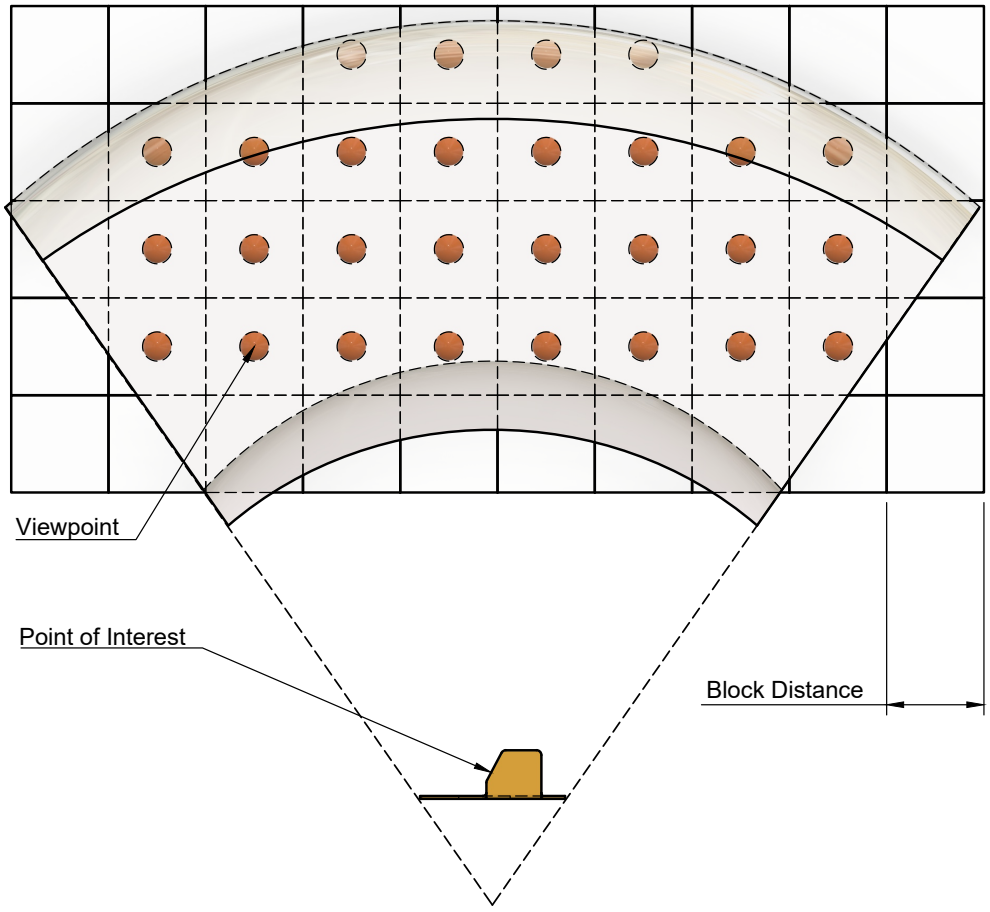


Figure 3.5. Sampling of discrete Viewpoints inside a Viewarea. The block distance specifies the sampling density.

or vertical field of view starting from the normal of the Point of Interest. This results in a rectangular cutout of a hollow sphere, which can be seen in the example in Figure 3.4. Each point inside a given Viewarea is a *Viewpoint* (VP). It can be used as a target for the inspection UAV to navigate to. In addition to the position of the point, each Viewpoint also contains an orientation for the UAV and its camera that allows it to look directly at the corresponding Point of Interest. This is achieved through trigonometric calculation of the roll, pitch, and yaw angle from the position of the given Viewpoint to the Point of Interest. Since the volume inside a Viewarea defines an infinite amount of Viewpoints, a specific Viewpoint must be selected for each Viewarea for the UAV to fly to during inspection. If the goal is to inspect the part with a minimum deviation angle (in other words, looking directly at it) and as close as possible, the point that is located at a distance of va_{min} along the normal vector of the Point of Interest would be an ideal choice. A different method for selecting a suitable Viewpoint using knowledge gathered

from previous flights is discussed later in this chapter. However, if the UAV cannot reach this point or the Point of Interest cannot be seen from this point because it is obstructed by an obstacle, the availability of alternative Viewpoints can allow the UAV to try inspection from these points. To select a set of Viewpoints the infinite number of points inside the Viewarea, a sampling algorithm was developed and described in previous work [246]. This approach generates Viewpoints that are equidistantly spaced at a predefined distance. It works by filling the area in and around the Viewarea with a grid of cubes of the size of the desired spacing of the Viewpoints. After discarding all cubes not located entirely inside the Viewarea, the centers of the remaining cubes are used as positions for the set of selected Viewpoints. Figure 3.5 shows a simplified 2D version of this sampling process. Afterward, the required camera orientation is calculated for each position, and IDs are generated for each Point of Interest. As a last step, the Viewpoint with the smallest combination of deviation angle and distance to the Point of Interest is chosen as a *Key Viewpoint*. The Key Viewpoint is used for this Viewarea in the path planning phase. While the UAV can still select other Viewpoints from the same Viewarea if inspection from the Key Viewpoint is not possible, it always tries to inspect from the Key Viewpoint first. This concept results in the domain diagram

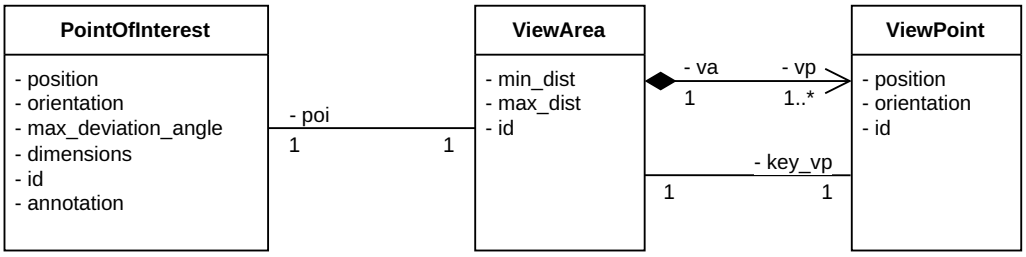


Figure 3.6. Domain model showing the relationships and properties of the concepts Point of Interest, Viewarea and Viewpoint.

shown in Figure 3.6. Each Point of Interest is described by its parameters position, orientation, max deviation angle, dimensions, ID, and annotation. Each Point of Interest corresponds to precisely one Viewarea described by a minimum and maximum distance as well as an ID. The required parameters of the camera for calculating the minimum distance are defined globally. Each Viewarea corresponds to precisely one Point of Interest and contains a set of Viewpoints, of which one is a Key Viewpoint. A Viewpoint is defined by its position, the orientation needed to inspect its Point of Interest, and an ID.

3.2.2 Calculation of an Efficient Inspection Sequence

After selecting a Key Viewpoint for each Viewarea and, therefore, for each Point of Interest, an order in which the Viewpoints should be visited has to be determined. Due to the inspection having to be performed by a UAV, generating a sequence of Key Viewpoints that results in a short Trajectory and, therefore, in a fast inspection flight

is crucial due to the limited flight time of UAVs. Also, a quicker inspection decreases the overall production time, which is also desirable. As previously mentioned, the approach chosen to determine an order in which the Viewpoints need to be visited is the Ant Colony System Algorithm (ACS). As the details of the basic algorithm itself were explained in detail in Section 2.1.1, the following paragraphs describe its adaptation for the calculation of a short inspection path that visits all Key Viewpoints.

A solution for finding a short permutation for a set of Key Viewpoints using the Ant Colony System algorithm was demonstrated in previous work [293], but is detailed here again for the sake of completeness. In this approach, Key Viewpoints represent the cities or nodes, and the distance between cities is modeled as the Euclidean distance between the positions of the two Viewpoints.

To apply the Ant Colony System algorithm to the domain of ordering Key Viewpoints, several changes had to be made to the algorithm.

Initial Pheromone Level
Equation (3.1) defines the initial pheromone level for each node In turn, the initial pheromone value τ_0 is calculated using Equation (3.1).

$$\tau_0 = \frac{1}{n \cdot L_{\text{nn}}} \tag{3.1}$$

With:

- τ_0 being the initial pheromone value.
- n being the number of nodes in the problem.
- L_{nn} being the length of the nearest-neighbor tour.

The nearest neighbor heuristic [218] is used to approximate the round trip length. While this heuristic is not ideal for finding the shortest path for a round trip, it is good at providing a rough estimation in a very short time. Combined with the number of Key Viewpoints n, it helps distribute an initial level of pheromones to each node that works well in conjunction with the pheromone update function.

Heuristic
The heuristic information η_{ij} provides an estimate of the distance the UAV must travel from Key Viewpoint i to j. Since the UAV can travel in a direct line in 3D space, the Euclidean distance is used to approximate the distance between two Key Viewpoints d_{ij}. The heuristic itself is calculated using Equation (3.2) as the inverse of the distance between the current node and the node connected to the edge d_{ij}.

$$\eta_{il} = \frac{1}{d_{ij}} \tag{3.2}$$

Parameterization
Minor adjustments were performed to the parameterization of the algorithm since its publication [246]. The parameter q_0, used to set the balance between exploitation and exploration, was increased from 0.5 to 0.6 to slightly decrease the encouragement of new paths. The evaporation rate ρ was adjusted from 0.5 to 0.3 to facilitate the establishment of new paths with only a few ants. The factor β controlling the influence of the heuristic information was left at 5 as it produced good results in empirical tests.

Number of Ants and Iterations
Determining the correct number of ants and iterations is one of the main challenges that need to be solved when using the ACS algorithm. Increasing the number of ants and iterations increases the quality of the solution, but returns diminish at a certain point [293]. More iterations and ants also increase the execution time of the algorithm. While the calculations needed for each ant can be parallelized through a multithreaded implementation, this works only until the number of ants reaches the number of CPU cores available. Since performing the calculation for an arbitrary iteration requires the result of the preceding iteration, calculating multiple iterations in parallel is not possible.

Therefore, a trade-off between the number of ants, iterations, and execution time has to be made. Empirical tests (see: Section 5.2.1) have shown that the definition of the number of ants m proportional to the number of Key Viewpoints N according to Equation (3.3) yields good results.

$$m = 0.3 \cdot N \tag{3.3}$$

The evaluation performed in Section 5.2.1 showed that increasing the number of iterations above $2000/(m/2)$ does not yield better results.

3.2.3 Optimization Potential Through Overlapping Viewareas

During the testing of the path planning algorithm and the subsequent generation of Viewareas, the observation was made that a significant overlap was present between two or more distinct Viewareas in many cases. This effect was amplified when testing a hypothetical inspection scenario of verifying correct mounting of brackets installed inside a longer concave aircraft fuselage section that was cut in half lengthwise (see: Figure 3.2). This is caused by the curvature of the part, causing the majority of the Viewareas to accumulate around the central axis of the tube. Through this, reducing the number of Viewareas that need to be visited by the UAV is possible by intersecting two or more Viewareas resulting in a new, smaller Viewarea and discarding the original Viewareas. A simplified, 2D visualization of the described intersction of two Viewareas can be seen in Figure 3.7. The optimization is possible, because if a Point of Interest can be seen from any point inside its original Viewarea, it also can be seen from any point inside the intersected Viewarea. However, the intersection process also ensures

that all other Points of Interest belonging to the other Viewareas can be seen from within the intersected Viewarea. So, when the UAV is located inside the intersected Viewarea, a simple change of its orientation allows it to inspect all Points of Interest whose Viewareas have been used to create the intersected Viewarea. This train of thought assumes that no obstructions are present between the location of the UAV and any of the Points of Interest, but since the intersected Viewarea still describes a volume with an infinite number of Viewpoints, alternative Viewpoints inside the Viewarea can be used to circumvent problems resulting from visual obstructions, should they occur. Also, in order not to reduce the amount of available alternative Viewpoints too much, the replacement of the original Viewareas is only performed if the intersected Viewarea is not significantly smaller than the original ones.

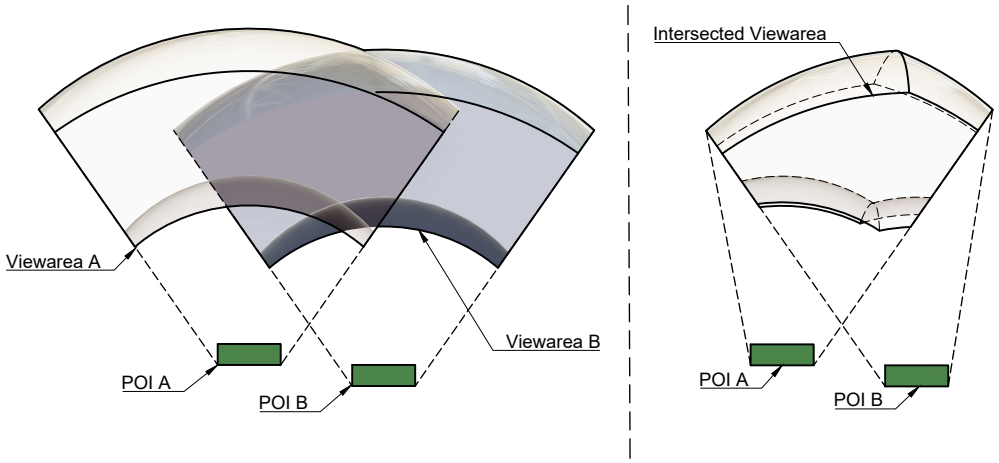


Figure 3.7. Reducing the number of Viewareas to shorten overall inspection length by combining intersecting Viewareas.

This strategy creates two main benefits: Firstly, the resulting length of the path and the number of accelerations and decelerations needed to perform the inspection are reduced. In previous work, it was demonstrated that a 36.7 % reduction in path length in a fuselage inspection scenario was possible while reducing the number of Key Viewpoints from 49 to 21 [246]. This results in shorter inspection times and the ability to inspect more parts per inspection flight due to decreased energy losses through accelerating and braking. Secondly, reducing the amount of Key Viewpoints resulting from the optimization process results in fewer points that need to be considered for calculating the inspection path. Since the TSP is NP-hard, the calculation time of the shortest inspection path increases significantly with the number of Viewpoints that need to be considered. Even when only calculating a reasonably short path using the proposed solution using the ACO algorithm, it is paramount to keep the number of Viewpoints as low as possible to reduce the time needed to calculate the path. The proposed optimization approach is, therefore, an essential step towards being able to carry out longer inspections with many Points of Interest.

3.2.4 Optimization of Key Viewpoints Across Inspections

All concepts presented so far only consider a single inspection performed once on an arbitrary part. Excluding the manufacture of highly individual parts or one-off prototypes, a factory usually manufactures multiple similar or identical parts. As this requires similar or identical inspections and, therefore, inspection trajectories, using information gained from previous inspections can be used to optimize future flights. In particular, problems with reaching a Viewpoint or being unable to inspect a part from a particular position due to a static obstacle may be avoided by choosing a Key Viewpoint for subsequent flights which is known to work, which reduces overall inspection times. Problems with reaching the Key Viewpoint or performing an inspection at the Key Viewpoint result in a significantly longer inspection time as the UAV tries to reach the Viewpoint or inspect the Point of Interest until a timeout occurs as it has no certain way of detecting that it cannot reach the Viewpoint or that it can not do the inspection from its current position. As previously hinted, it picks a different Viewpoint from the same Viewarea if one of these timeouts occur and tries to perform the inspection from there. When repeating the same or a similar inspection, the same process repeats, and the same delays caused by timeouts occur again. The proposed

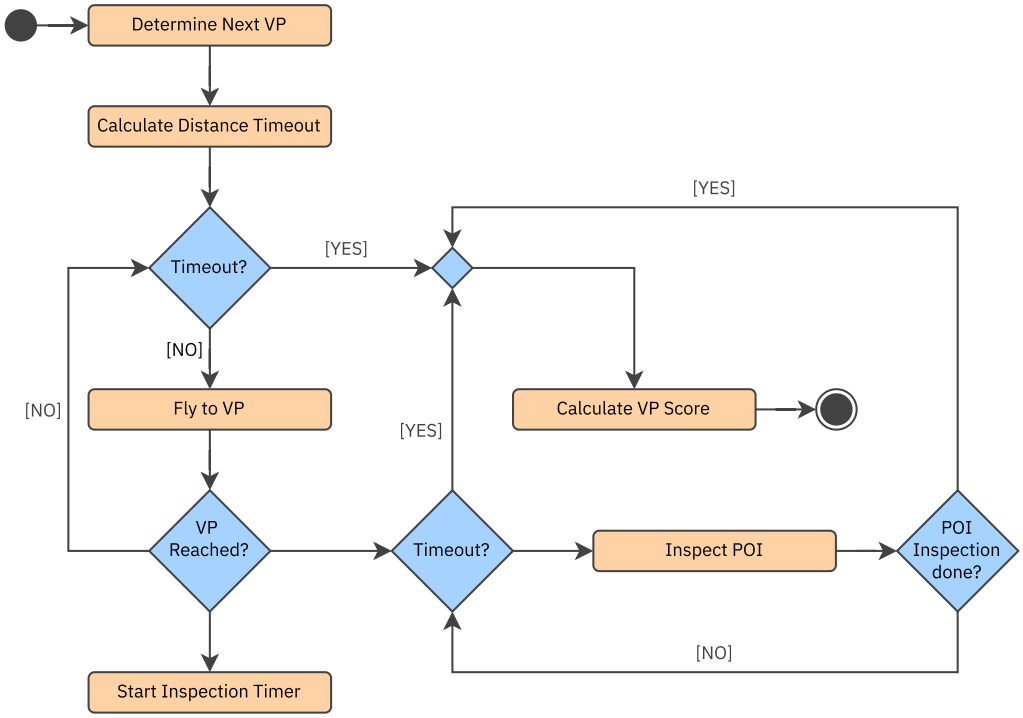


Figure 3.8. Activity Diagram of Viewpoint processing and scoring logic.

solution for this problem is acquiring knowledge about the quality of Key Viewpoints throughout multiple flights [247]. This is done by assigning a score value to each of the sampled Viewpoints of a given Viewarea. When initially calculating an inspection path,

it is set to 0 upon initialization, the previously described strategy for selecting a Key Viewpoint is used, and its score is set to a higher number K. The procedure followed when trying to inspect an arbitrary Point of Interest during an inspection is depicted in Figure 3.8. Initially, the UAV tries to fly to the Viewpoint with the highest score, which is the Viewpoint with a score of K. Before trying to reach the Viewpoint, a distance timeout is calculated based on the Euclidean distance from the UAV's current position and the goal. If the UAV can not reach its goal within this timeout, the score of this Viewpoint is decreased by a fixed amount $A > K$. Afterward, the process is repeated, and a new Key Viewpoint is selected from the Viewpoints with the highest score. If a Viewpoint is reached, but a successful inspection cannot be performed after a certain duration, the Viewpoint's score is also decreased by an amount of $B > K$, and again, a new Key Viewpoint is selected from the Points of Interest with the highest score. If multiple contenders for a new Key Viewpoint are available, the one with the largest Euclidean distance to the current Viewpoint is selected. This is done to maximize the chance of avoiding potential obstacles blocking the UAV from reaching the Viewpoint or viewing the Point of Interest. Upon successful inspection of a Point of Interest from a given Viewpoint, its score is increased by the constant D to reward a successful inspection. In addition, every Viewpoint that was reached successfully gets its score updated using the following formulas:

$$\text{trajectory\_efficiency} = \frac{d_e \cdot 1.1}{d_r} \tag{3.4}$$

$$score = score + C \cdot \text{trajectory\_efficiency} - 1 \tag{3.5}$$

First, the efficiency of the flown trajectory is calculated using the Euclidean distance d_e between start and goal and the actually traveled distance d_r. This efficiency is then used in conjunction with a constant C to update the Viewpoint's score. This operation aims to penalize Viewpoints that result in complicated and long trajectories due to the UAV having to fly around obstacles.

The scores calculated during a flight are then persisted and used for the next flight. Also, the ACO-algorithm can be used again to calculate a new shortest path for the updated Key Viewpoints. The overall idea of the score system is to reward known good Viewpoints and penalize Viewpoints that do not work for an inspection. Throughout multiple inspections, the system converges on a selection of known good Viewpoints, and avoids repeatedly flying to Viewpoints that do not work.

Since the scores are specific to a Point of Interest and only loosely to an inspection path, it is possible to reuse the scores on similar inspections that may occur when some variance in the produced assemblies is present. While the algorithm presented here is intended to improve inspection time throughout multiple real inspections, it is also possible to run it for multiple iterations in simulation to provide an already optimized set of Key Viewpoints for the actual inspection. To do this, the actual flight must be performed in simulation with the assembly and the environment simulated as precisely as possible. When all possible obstacles are placed in the simulation, a

good approximation of the accessibility of Viewpoints and the trajectory length can be determined. To verify that a particular Point of Interest is visible from a Viewpoint, ray tracing can be used to check for obstructions between the UAV's camera and the Point of Interest. Simulating a few flights and updating the Viewpoint scores before performing the real-world inspection can reduce inspection time right at the first inspection. This also provides the benefits of this optimization strategy for one-off inspection flights that would otherwise not be able to benefit from this approach.

3.2.5 Related Work and State of the Art

Many types of UAV-based inspections require covering the entire surface of the inspected part, like the inspection of wind turbines or aircraft while in service. Various solutions exist for planning a path that covers the entire surface of an object using coverage path planning (CPP) [95]. While in these cases the entire structure must be checked for defects, in a manufacturing context, the modified sections of the assembly that need to be checked are known in advance. Therefore, this section will focus on path planning for viewpoint-based inspections. However, some concepts from CPP algorithms, like the use of Viewpoints, overlap with the concepts used for selective path planning presented in this work. Therefore, relevant solutions for CPP will be discussed as well.

Coverage Planning

According to Bircher et al. [24] an approach to computing a coverage path often consists of two consecutive steps. First, the Viewpoints that are required for the inspection are computed. In a second step, a path or trajectory connecting all inspection points is calculated. For computation of the required Viewpoints, coverage path planning can be seen as an Art Gallery problem [103, 206]. The Art Gallery problem is a problem in computational geometry that covers the visibility of a polygon. It is analogous to an art gallery that has to place security guards in a way that each part of the gallery is visible to at least one guard. Applied to CPP, the problem is finding camera positions in a way that each part of the assembly is visible in at least one Viewpoint. Danner and Kavraki [52] also use an art gallery approach to generate inspection points and connect them to an inspection path. Since the problem of placing the guards in the gallery is NP-hard [266], [98], they place them randomly in unguarded sections. Bircher et al., in contrast, model the component surface as a triangle mesh and define a Viewpoint for each triangle. Each Viewpoint is defined in a way that the corresponding triangle is completely visible. While this first step is not of much use for calculating a Viewpoint-based inspection path, the second step, planning a path through all Viewpoints, has many similarities to the solutions found in CPP. One common approach is to model the path planning as a traveling salesman problem, for which various well-studied and performant solutions exist [53, 162].

Englot and Hover [71] also use this two-step approach to perform CPP for the inspection of ship hulls using an autonomous underwater vehicle. The path planning takes place relative to the ship hull. The Viewpoints are sampled randomly using the Redundant Roadmap Algorithm until sufficient surface coverage is achieved. An iterative combina-

tion of computing the shortest path between all nodes and using the Rapidly Exploring Random Tree (RRT) [150] algorithm is then used to calculate a short inspection path.

Viewpoint-Dependent Path Planning

Papaioannou et al. [209] introduce a receding horizon control approach for autonomous UAVs to inspect a finite number of feature points on 3D structures. The inspection task is formulated as a constrained optimal control problem, which is solved using mixed-integer programming. This results in efficient and targeted inspections.

Wang et al. [291] propose a specialized system for the inspection of electricity pylons. They also use the concepts of Viewpoints to define inspection positions and use optimizations based on predicted overlapping captured images. The Viewpoints take the camera orientation into consideration, and an option to specify Viewpoints in a way that ensures a correct lighting based on the position of the sun is proposed. However, the approach does not use the concept of Viewareas and alternative Viewpoints and is therefore dependent on the only Viewpoint working. In the domain of power line inspection, automated detection of specific points of interest during flight is possible, allowing for less manual planning and specification requirements before a flight.

Ramon-Soria et al. [230] describe a planning system for the autonomous inspection of infrastructure by UAVs. In the publication, a complete system to automate the UAV inspection of arbitrary civil infrastructure is presented. This system includes a planning algorithm for computing ideal inspection points and trajectories from one point to the next. The planning algorithm seems to be based on the concepts of Points of Interest and Viewpoints (called *approaching points* in the publication) described in the previous sections. Approaching points are determined using a set of constraints. The UAV has to be located in front of the target surface and remain within a fixed safety distance from the surface. Additionally, the points are created in a way that the UAV stays visible to the ground station and the operator. The points that match these constraints form an equivalent of the concept of a Viewarea. An ideal approaching point is selected by minimizing a cost function that includes the distance to the Point of Interest, ground station and angle to the normal vector of the inspected surface. The path between Viewpoints is planned using an RRT*-based algorithm [135]. How or if an ideal order for visiting the Viewpoints is determined is not mentioned in the paper, but the provided source code [72] suggests that the problem is modeled as a TSP, which is solved by calculating the path length of all possible permutations and returning the path with the minimum length. While this approach works, it seems to have a computational complexity of $O(n \cdot n!)$, which results in long planning times with an increasing number of Viewpoints.

Maini et al. [175] present a set of algorithms that allow to find the minimum length path for an UAV visually inspecting a set of Points of Interest. They take the field of view of the UAV into consideration by developing a constant-factor approximation algorithm to solve the problem of visually monitoring points of interest on a 2.5D terrain. Therefore, the approach is limited to 2.5D environments with the drone camera fixed in an orientation looking straight down at the terrain. The authors also use an approach

using a set of Points of Interest that need to be inspected. Camera parameters and the surrounding terrain are used to calculate visibility regions from where each Point of Interest is visible, similar to the concept of Viewareas. However, these regions are two-dimensional regions at a fixed altitude. The authors then propose two different approaches for planning a path through all visibility regions. The first approach uses a constant-factor approximation algorithm that executes in polynomial time and works without discretization of the visibility regions. The second approach models the problem as a generalized TSP where discrete points are sampled within each visibility region, and a short path has to be calculated through exactly one of the points of each visibility region. This is approach is implemented using a GLNS-solver [264]. Both algorithms are benchmarked and compared, and the authors conclude that the GLNS-based approach outperforms the approximation algorithm. The feasibility of the approach is demonstrated in a proof of concept consisting of a number of real-world flights.

Camera Orientation

According to Alarcon-Herrera et al. [12], an ideal Viewpoint has its normal axis aligned with the camera's optical axis. The optimal Viewpoint can be found by using the point located on the optical axis of the Point of Interest at a so-called standoff distance. It is calculated using the desired size and resolution of the Point of Interest in the captured camera image. This can, in turn, be calculated using the dimensions of the camera sensor and the camera's focal length. Using this approach, the Point of Interest is always centered in the image and directly viewed at. This prevents distortions that may happen on the corners of wide-angle lenses or by looking at the Point of Interest from the side. While this approach can calculate the ideal Viewpoint for each scenario, it also fails to provide an alternative Viewpoint if the inspection fails from the initial Viewpoint.

3.3 Relative Positioning

For any autonomous flight with a UAV, its position must be established to provide feedback for the position controller. One of the significant differences that separates the system presented in this work from existing solutions is the complete reliance on relative positioning instead of using absolute coordinates. This allows inspections of an assembly independent of where it is currently placed since all positions (including the UAV position) are defined relative to the assembly. This not only brings flexibility on where to perform the inspection for a particular assembly, when inspecting multiple similar targets like wind turbines in a wind farm (see: Section 1.2.2), the same inspection can be performed on all turbines without making alterations.

To meet the requirements of an inspection within a production hall, the position must be determined using only the sensors available in the UAV. Global Navigation Satellite Systems (GNSS) such as GPS do not work inside enclosed buildings [102], and the cost and setup complexity of other external positioning systems such as optical trackers or Ultra-Wideband (UWB)-based systems [43] also make them unsuitable. Optical tracking systems such as the Vicon system [187], which determine the UAV's position using markers attached to it, also have the problem that the tracking markers attached to the UAV are obstructed once it enters the interior of a structure. In addition, these systems limit the flexibility otherwise gained through the relative positioning approach by constraining the inspection area to the space covered by the tracking system. Avoiding external positioning systems limits the choice of sensors for position estimation to onboard sensors. The following sections provide a general overview of the proposed system for position estimation using only onboard sensors. Afterward, the concept for estimating the UAV position using model tracking, along with the required pre-processing of the sensor data and its fusion with other sensor data, is explained in detail. Figure 3.9 depicts the general structure of the basic positioning system. The UAV measures its position using two different sensors. Firstly, it calculates a geometric position relative to the starting point using the onboard inertial measurement unit (IMU), consisting of an accelerometer, gyroscope, and compass, together with data obtained from a downward-facing optical flow sensor. Secondly, the camera image captured by the UAV is fed into a software that tries to match features of a 3D model of the part to the camera image and uses this to calculate the orientation of the camera. This type of optical tracking has recently found use in Augmented Reality frameworks like Vuforia Engine [226], where 3D models need to be overlaid on a live camera image.

While both types of measurements estimate the UAV's position, they are not sufficient to perform navigation tasks on their own. The position calculated by the UAV odometry is, in large parts, obtained by integrating the acceleration and rotation speed values obtained from the IMU twice. The numeric integration of these values introduces a small error that sums up over time [236]. While this effect can be mitigated if the UAV is equipped with an optical flow sensor and a downward-facing distance sensor like a LIDAR, the position measurements still drift over time.

The model tracking, on the other hand, not only relies on being able to see the object to be inspected at all times, the matching of the camera image to the CAD model may

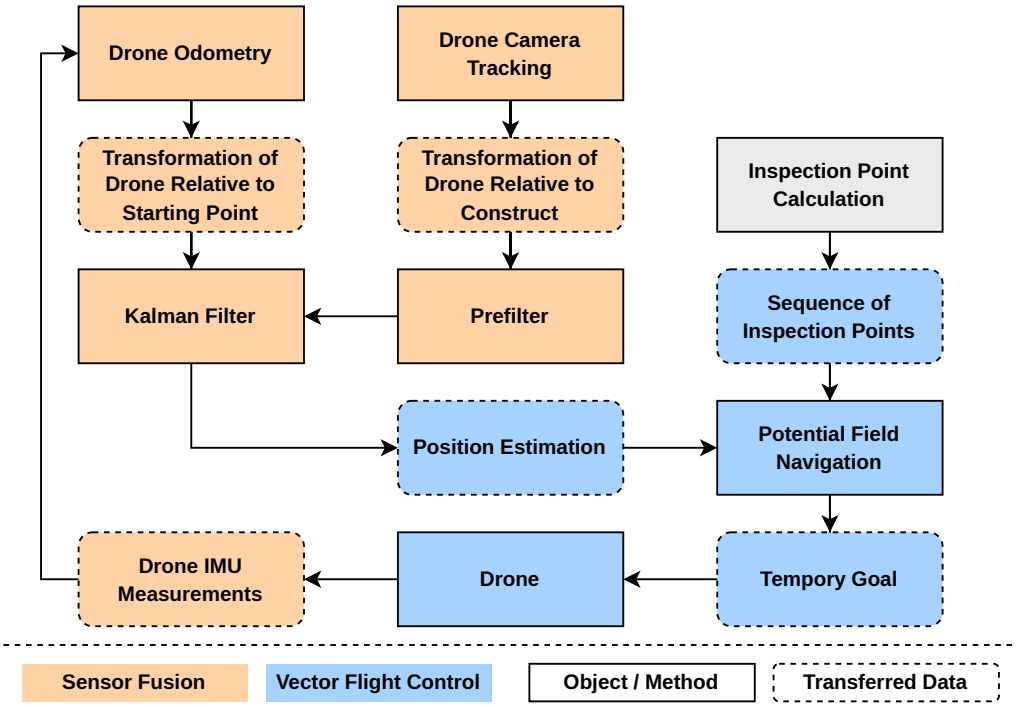


Figure 3.9. Structure of the proposed visual navigation stack.

also fail on some occasions during flight, which can result in missing position data. Another way this system can fail is if the algorithm matches the camera image and model in a wrong orientation or at the wrong position. This often happens for only a few frames until the model is tracked in the correct position again. These erroneous position measurements often result in a big difference between the UAV's target and its perceived position, resulting in significant position errors in the position controller and, therefore, extreme correction maneuvers that move the UAV into a seemingly random position at high speeds. This can result in crashes or collisions with its environment.

The proposed approach, therefore, uses a pre-filter for the model tracking measurements to filter values that would require the UAV to move faster than it is physically capable of. Additionally, sensor fusion with a Kalman Filter in conjunction with the pre-filter is used to create a more robust position estimation using both position measurements and a prediction of the UAV's state based on an internal model. This position is then used alongside the previously calculated Viewpoints as input for the navigation algorithm, which in turn provides control commands to the UAV to make it perform the inspection.

3.3.1 Position Estimation Through Model Tracking and Onboard Sensors

While the basic working principle of model tracking was covered in detail in Section 2.2.4, the following paragraphs focus on its use as a position sensor for relative navigation. Model tracking has been used in literature to perform tasks like determining the pose from camera to an object in order to perform tasks like grasping [44]. However, its use for navigation is relatively rare. This might be due to most robot applications requiring an absolute position either inside a map or in the form of GPS coordinates for navigation tasks. Knowledge of the robot's position relative to an object is only useful if the robot wants to interact with the object or if the object's position inside the map is known. Therefore, an absolute position on the map can be derived from a model tracking measurement of the object. In comparison, the relative navigation proposed in this work is mainly performed in the reference system of the object that is inspected, making a model tracking measurement equivalent to an absolute measurement inside the UAV's map. This allows for all Points of Interest, Viewpoints, and other important poses to be defined relative to the inspected object, independent of their absolute positions.

Since the proposed system does not place any special requirements on the model tracking that would require it to be modified, a preexisting commercial solution can be used during implementation. This is why the model tracking itself in this work is treated as a black box with a video stream and tracking model as an input and a pose estimate as output. The focus is filtering and integrating the model tracking data as part of the position estimation system and combining its data with other sensors to obtain an improved position estimate.

3.3.2 Processing of Model Tracking Measurements

As previously described, using only the UAVs' onboard IMU is not feasible due to the drift introduced into the position measurement over time. When limiting the options to only the onboard sensors of the UAV, using a model tracking algorithm to obtain a second type of position measurement seems logical. The system needs the CAD model of the part to be inspected anyway for path planning, and its camera naturally faces the assembly most of the time for inspection purposes. As long as the tracking is stable, the model tracking gives a stable position relative to the assembly. Should tracking problems occur, however, measurements may drop out, or incorrect measurements may occur.

In order to mitigate these problems, a two-step filtering approach is used to first sanitize the tracking data and then fuse them with the odometry measurements and combine them with a predicted UAV position using the internal system model of a Kalman Filter. In the first step, the model tracking data is pre-filtered based on the physical capabilities of the UAV. Once a new tracking result enters the system, it is compared to the last good measurement that was kept in memory. This is done by first calculating the Euclidean distance d between the last and current position like shown in Equation (3.6).

$$d = \sqrt{(x_{\text{current}} - x_{\text{old}})^2 + (y_{\text{current}} - y_{\text{old}})^2 + (z_{\text{current}} - z_{\text{old}})^2} \tag{3.6}$$

With x_{current}, y_{current} and z_{current} being the relative coordinates of the current model tracking measurement and x_{old}, y_{old} and z_{old} being the coordinates of the last good measurement. The system then proceeds with calculating the maximum distance the UAV could have realistically traveled since the last valid position measurement using Equation (3.7).

$$d_{\text{max}} = (v_{\text{uav-max}} + v_{\text{comp-max}}) \cdot \Delta t \cdot C_n \tag{3.7}$$

With d_{max} being the maximum distance that the UAV could theoretically have travelled when flying at maximum speed with the component moving in the opposite direction at its maximum speed, $v_{\text{uav-max}}$ is the maximum speed of the UAV, $v_{\text{comp-max}}$ is the maximum speed of the component and Δt the time that has passed since the last valid tracking measurement. The maximum component velocity must be considered when calculating this upper limit for the distance the UAV could have travelled because components can also be inspected while moving. This could be done while moving from one manufacturing station to another with a ceiling crane. Performing inspection during this time when no other work can be done on it is one of the factors contributing to the improved production output expected from the system. This, however, means that the maximum theoretical distance the part could have traveled must be considered when calculating the upper bound d_{max}. Lastly, the constant $C_n > 1$ adds a margin of error for the calculations. Its exact value has to be determined empirically, but a value between 1 and 2 seems to be appropriate, with smaller values increasing the chance of discarding valid measurements and bigger values resulting in more invalid measurements being accepted as valid.

Finally, Equation (3.8) is used to determine if a measurement is valid by comparing if the traveled distance d is smaller than the maximum distance that is physically possible d_{max} (see Figure 3.9).

$$valid = d < d_{\text{max}} \tag{3.8}$$

If this check is successful, the measurements are passed on to the Kalman Filter. As demonstrated in previous work [248], this results in cleaner input for the Kalman Filter and, therefore, better position estimates. Due to the nature of the Kalman Filter, it can continue to operate and provide position estimates even when operating without or with only some of its sensor values for a while.

Transformation from Camera to UAV

Since the model tracking works using the camera image of the UAV, the model tracking approach provides the estimated pose between the assembly and the camera. However, for sensor fusion and position control, the pose of the UAV's base frame located in the center of the UAV is needed. On most UAVs, the Camera is mounted at a fixed position in the lower front of the main body. It is often stabilized in two or three axes of rotation using a camera gimbal that compensates for the rotational movement of the UAV when navigating and provides a stable camera image.

Therefore, the measurement of the model tracking must be transformed into the base frame of the UAV. This is done using the transformation matrix from the UAV's base frame to the camera $\mathbf{T}_u^c$ defined in Equation (3.9).

$$\mathbf{T}_u^c = \begin{bmatrix} \mathbf{R}_u^c(\hat{\phi}_u^c, \hat{\theta}_u^c, \hat{\psi}_u^c) & \begin{bmatrix} x_u^c \\ y_u^c \\ z_u^c \end{bmatrix} \\ 0 \quad 0 \quad 0 & 1 \end{bmatrix} \tag{3.9}$$

With:

- $\mathbf{R}_u^c(\hat{\phi}_u^c, \hat{\theta}_u^c, \hat{\psi}_u^c)$ being the rotation matrix of the camera frame relative to the UAV base frame.
- x_u^c, y_u^c, z_u^c being the position of the camera frame relative to the UAV base frame.

While the position of the UAV base frame relative to the camera gimbal frame remains constant, the rotation matrix must be regularly updated using the current camera gimbal positions. These angles need to be obtained as sensor readings from the UAV. The implementation developed in this work locks the yaw axis of the gimbal to a fixed orientation relative to the UAV, so the camera always faces straight forward. This also means that the yaw orientation at a particular viewpoint is achieved by rotating the entire UAV into the desired orientation. The roll axis is set to counteract the UAV roll so that the horizon of the camera image stays locked. The pitch can be influenced by the inspection mission. This makes it possible for the UAV to look down or up at Points of Interest at predefined angles. Since the pitch is defined relative to the global horizon and the UAV's pitch varies in relation to that, a measurement of the actual gimbal position in the yaw axis is needed to calculate the rotation matrix correctly.

After calculating $\mathbf{T}_u^c$, the pose of the UAV in the frame of the assembly is defined by the transformation from the assembly origin to the UAV's base frame $\mathbf{T}_a^u$ according to Equation (3.10).

$$\mathbf{T}_a^u = \mathbf{T}_c^u \cdot \mathbf{T}_a^c = (\mathbf{T}_u^c)^T \cdot \begin{bmatrix} \mathbf{R}_a^t(\hat{\phi}_a^t, \hat{\theta}_a^t, \hat{\psi}_a^t) & \begin{bmatrix} \hat{x}_a^t \\ \hat{y}_a^t \\ \hat{z}_a^t \end{bmatrix} \\ 0 \quad 0 \quad 0 & 1 \end{bmatrix} \tag{3.10}$$

With:

- $\mathbf{R}_a^t(\hat{\phi}_a^t, \hat{\theta}_a^t, \hat{\psi}_a^t)$ being the rotation matrix of the estimated Euler angles $\hat{\phi}_a^t, \hat{\theta}_a^t, \hat{\psi}_a^t$ from the model tracking.
- $\hat{x}_a^t, \hat{y}_a^t, \hat{z}_a^t$ being the estimated position from the model tracking.

3.3.3 Related Work on Relative Position Determination

With the concepts for position estimation presented in the previous sections, the following paragraphs compare them to the current state of the art in industry and scientific literature. The focus is on using model tracking and other sensors for determining a flying robot's position.

Model Tracking for Determining Robot Positions

In the domain of visual inspection, especially in the aviation industry, two notable examples for the use of model tracking are the tools DELMIA (formerly known as DIOTA) [269] and MiRA [100]. While MiRA is a tool developed by an airbus subsidiary for visual inspection of their aircraft parts, DELMIA is offered as a product by Dassault Systèmes. Both offer handheld inspection solutions using a camera mounted to a tablet. The camera image is displayed on the tablet with an augmented model of the inspected part. The software allows to perform both automated and manual inspection tasks, storing the inspection results using a connection to the PLM system.

Augmented Reality Frameworks

Regarding Augmented Reality frameworks, notable examples are Vuforia Engine by PLC [222] and VisionLib [245, 286] by Visometry GmbH. Both offer a framework for the development of Augmented Reality applications and, as a result, offer their own implementation of a model tracking system. While VisionLib focuses on model tracking, Vuforia Engine also offers marker-based tracking through flat or cylindrical images [221]. Also, it provides support for objects equipped with multiple markers [227]. In a patent [245], VisionLib also describes a system for detecting the presence and orientation of a part on an assembly which will be discussed in more detail in Section 3.5.3.

Scientific Literature

In scientific literature, model tracking was first introduced in the 1990s with approaches that calculate the camera's pose and tracked object using feature extraction matching between camera and 3D model [144, 278]. Ababsa and Mallem [3] also present a similar approach for camera pose estimation using point and line tracking on the 3D model. They also extend their system by using the calculated camera positions as input for an Extended Kalman Filter [2] to filter the output of the tracking to achieve better tracking precision. Gall et al. use a different approach to increase accuracy by adding texture data to the 3D model and extracting and tracking features not only from the 3D geometry but also from the texture of the object [96]. While this is done with a relatively simple model explicitly manufactured to match the texture of the 3D model, the widespread availability of photogrammetry applications could make this approach feasible for arbitrary objects that can be scanned using these. Another approach by Jurado-Rodríguez et al. [130] combines model tracking with fiducial markers to derive a system that outperforms conventional approaches based on model tracking. Choi et al. apply 3D model tracking for grasping tasks using a robot arm [44]. Their approach also

supports meshed 3D models. Most other approaches use CAD models since it is easier to extract relevant data, such as points and edges, from these kinds of models.

Computer Vision for Determining Robot Positions

Due to the availability of cameras on most modern UAVs, many systems that leverage this type of sensor exist [212]. Weiss et al. [295] propose a computer-vision algorithm for a stream of images obtained from a downward facing camera on a UAV and process the output in an Extended Kalman Filter to implement an optical-flow sensor for measuring movements of the UAV.

Leishman [152] uses a stereoscopic RGB-D camera, a sonar altimeter and an IMU as input for a visual graph-SLAM algorithm to determine the position of an UAV. A Multiplicative Extended Kalman Filter (MEKF) is used to filter the calculated position. Similar approaches exist in literature with Liu et al. [164], Pestana et al. [212], Cazzato et al. [35] and Hardy et al. [109] using computer-vision based object tracking algorithms and a nonlinear Kalman Filter.

While these generic approaches work with a wide variety of tracking targets or do not need the explicit specification of a tracking target, in an inspection scenario, using the available geometric structure of the inspected object can improve positioning accuracy and facilitate the position determination of inspection points relative to the UAV. Moolan-Feroze et al. [192, 193] use vision-based algorithms to determine the relative position of a UAV inspecting a wind turbine. Their approach uses a pose graph optimizer to combine the GPS data and position estimates calculated by a novel algorithm processing the camera images. The vision-based algorithm works by comparing the images of the UAV's camera with a 3D-model of the wind turbine. A specially trained Convolutional Neural Network (CNN) is used to extract the position of the wind turbine's tower and turbine blades from the images and match them to the positions on the model. While the approach works well and was tested in real-world inspection flights, it is questionable, if the approach can easily be adapted to more complex geometries.

Some approaches track visual markers to determine the position of the UAV relative to the marker [25, 27, 185]. Lange et al. [149] use a camera mounted on the UAV to track a visual marker placed on the ground for precise landing. Kalinov et al. [131] use a combination of 2D-LiDAR sensors, vision-based algorithms and ultrasonic sensors to determine the position of a UAV relative to a mobile ground robot. The system is designed for a collaborative robot pair operating in a warehouse environment. Contrary to most other approaches, the camera is mounted not to the drone but to the ground robot which is supposed to eliminate errors introduced by pitch and roll of the UAV while navigating. The UAV is instead equipped with a set of infrared markers indicating its position. The relative position measurement is used to land the UAV on the ground-based robot and is able to do so while achieving a positional accuracy of 1.25 cm. Potena et al. [217] focuses on navigating along optimal trajectories while flying towards an optical marker. The authors propose an optimization-based visual navigation method for multirotor UAVs to reach a goal while maintaining line of sight with a target object. During flight, the onboard camera tracks an AprilTag [204] and continuously adjusts the

orientation of the vehicle to ensure optimal tracking of the marker. Walter et al. [290] use a specialized marker emitting a pattern of UV-light on a leading UAV to provide position estimation for following UAVs in a formation flight. Each follower UAV is equipped with a camera which detects the marker pattern and uses it to determine its relative position to the leader. This enables the multi-robot-system to follow the leader in formation flights. While markers offer an easy way to make visual tracking approaches more robust, their use in the inspection of large assemblies also require them to be attached to the inspected part prior to installation adding a step that must be performed before the inspection can take place. Because their presence is undesirable in many scenarios, they also might need to be removed after the inspection. Additionally, to achieve maximum tracking accuracy, they need to be placed precisely on the inspected object. These factors often make them undesirable for inspection tasks.

Laser Sensors for Determining Robot Positions

Another common type of system for position estimation is the use of laser-based distance sensors (see: Section 2.4). LiDAR sensors with their ability for simultaneous localization and mapping (SLAM) are most popular in this field. However, localization relative to an inspected object usually requires an additional means of determining the objects position inside the generated map.

In industry, many companies make use of LiDAR sensors as they provide accurate localization combined with obstacle avoidance capabilities with examples being Donecle [59, 63] and Mainblades [174] both offering inspection of large aircraft. However, the drones used in these scenarios are designed to work either outside or in large hangars where they have plenty of space to operate. The heavy LiDAR sensor requires the UAVs to be comparatively large to lift the added weight limiting their ability to navigate through tight spaces. An exception to this phenomenon is the Elios 3 by Flyability [239] with dimensions of only 38 cm(height) by 48 cm (width). The UAV is specially built for flight in areas with a great deal of enclosed spaces with the propellers guarded in fan ducts and an anti-collision cage surrounding the entire drone. Still, the LiDAR Sensor makes up 450 g of the total weight of 2350 g causing a reduction of a reduction in flight time from 12,5 min to only 9 min.

Tenorio et al. [275] present a system that uses an approach similar to model tracking to determine the position of a robot inside a room that a 3D camera has previously scanned. In the preparation phase, the textured 3D model of this room is used to generate 2D renders from different perspectives that are then matched to the camera's image at runtime to determine the position. The drawback of this approach is that a textured model of the object or environment is needed, and the additional preprocessing needed to create the renders adds additional complexity to the setup. In scientific literature, Khalil et al. [139] use a similar system for inspecting the inside of heating, ventilation, and air conditioning (HVAC) ducts. Like the Elios 3, the custom-built UAV uses a cage-structure to prevent collisions between the UAV and the surrounding ducts. However, instead of a quadrotor with ducted propellers, the authors use two motors per arm pointing in opposite directions to build an octacopter using identically sized propellers.

The weight of the drone including an onboard companion computer and thermal camera is 3 kg and the maximum flight time is stated as 4 minutes. The authors compensate for this low flight time by implementing an additional means of movement by rolling the drone along the ground using the spherical cage. Miranda et al. [190] perform their inspection of exterior screws on an airplane fuselage using a Donecle UAV [59, 63], but focus on the processing of the acquired inspection data.

3.3.4 Sensor Data Fusion

As previously described, both means of measuring the UAV's position using onboard sensors have inherent flaws that make them unsuitable as a stand-alone means for establishing its position on their own. Therefore, as demonstrated in previous work [248], a discrete Kalman Filter [132] is used to combine the two measurements to achieve a better position estimation. The following paragraphs describe how the Kalman Filter, as described in Section 2.2.6, must be designed to work in the context of relative positioning. The parameterization of the Kalman Filter for both its prediction and update step is discussed in the following paragraphs. The concept of sensor fusion in this section only considers positioning relative to a stationary object. An advanced approach that works even if the assembly is in motion or different parts of the assembly move relative to each other is discussed in later sections.

Prediction of the State Estimate

In order to describe the state of the UAV for navigation tasks, its position and velocity in x, y, and z-direction relative to the assembly are needed, together with the orientation in the roll, pitch, and yaw axes, and the corresponding rotational velocities. This results in the state vector $\hat{\mathbf{x}}$ described in Equation (3.11).

$$\hat{\mathbf{x}} = \begin{pmatrix} x_a^u & \dot{x}_a^u & y_a^u & \dot{y}_a^u & z_a^u & \dot{z}_a^u & \phi_a^u & \dot{\phi}_a^u & \theta_a^u & \dot{\theta}_a^u & \psi_a^u & \dot{\psi}_a^u \end{pmatrix}^T \tag{3.11}$$

With:

- x_a^u, y_a^u and z_a^u being the current x-, y- and z-position.
- $\dot{x}_a^u$, $\dot{y}_a^u$ and $\dot{z}_a^u$ being the current x-, y- and z-velocity.
- ϕ_a^u, θ_a^u and ψ_a^u being the orientation on the roll-, pitch- and yaw-axes.
- $\dot{\phi}_a^u$, $\dot{\theta}_a^u$ and $\dot{\psi}_a^u$ being the rotation speeds around the roll-, pitch- and yaw-axes.

Note that unless stated differently, all positions, velocities, and rotations are described relative to the base link frame or origin of the assembly F_a.

The state transition matrix of the filter is structured in a way that the next predicted position in all axes (linear and angular) is the sum of the old position and the numeric integration of the velocity estimate, as described in Equation (3.12).

$$\bar{u_{k+1}} = u_k + \dot{u}_k \cdot \Delta t \tag{3.12}$$

With:

- $\bar{u_{k+1}}$ being the prediction of the position of an arbitrary axis of the UAV.
- u_k being the current state estimate of this axis.
- $\dot{u}_k$ being the velocity estimate corresponding to the same axis.
- Δt being the time difference from step k to $k+1$.

Note that the superscript $-$ indicates that this value is a prediction that has not been corrected with an update by a sensor value. Velocities are not updated in the prediction. To achieve this behavior, the state transition matrix $\mathbf{F}$ shown in Equation (3.13) is needed.

$$\mathbf{F} = \begin{pmatrix} 1 & \Delta t & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & \Delta t & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & \Delta t & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & \Delta t & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & \Delta t & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & \Delta t \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \tag{3.13}$$

Upon start, due to the absence of information about the position and velocity of the UAV, $\hat{\mathbf{x}}_0$ is initialized as $\mathbf{0}_{12} \in \mathbb{R}^{12}$. Because this state is most likely not accurate, a high initial value for the error covariance is used (see: Equation (3.14)).

$$\mathbf{P}_0 = 100 \cdot \mathbf{I}_{12} \tag{3.14}$$

Since $\mathbf{P}$ is recalculated after each update with new sensor data, the exact value for the initial matrix is not necessary as long as it is significantly higher than the covariance of the sensor data. This is required to ensure that the initial measurement has a high influence on the first update step to achieve a fast correction of the erroneous data in $\hat{\mathbf{x}}_0$.

The control input for the UAV is a linear and angular velocity, resulting in the following control input vector $\mathbf{u}$ and control input matrix $\mathbf{B}$.

$$\mathbf{u} = \begin{pmatrix} \dot{x}_a^u & \dot{y}_a^u & \dot{z}_a^u & \dot{\phi}_a^u & \dot{\theta}_a^u & \dot{\psi}_a^u \end{pmatrix}^T \tag{3.15}$$

$$\mathbf{B} = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \tag{3.16}$$

This provides a prediction for all velocities of the state vector, which would otherwise not be updated.

Update of the State Estimate

Every time a sensor measurement of either the IMU or the model tracking is performed, it is used to update the state estimate of the Kalman Filter. This is done for both the IMU $\hat{\mathbf{z}}_{i|k+1}$ and model tracking measurement $\hat{\mathbf{z}}_{t|k+1}$ respectively. While the general equation for calculating the predicted measurement (see: Equation (2.20)) is the same for both IMU- and model tracking measurements, the values for $\mathbf{H}$ differ due to the different structure of the measurement $\mathbf{z}$ for both types of measurements. Therefore, they are separately discussed in the following paragraph.

Tracking Measurements
The transformation matrix $\mathbf{T}_a^u$ calculated by the model tracking prefilter according to Equation (3.10) provides the filtered position and orientation of the UAV's base frame relative to the assembly based on the model tracking measurement. The individual positions and orientations can be obtained from the translation vector and rotation matrix of this transformation and are used to form the measurement vector $\mathbf{z}_t$ for the model tracking update.

$$\mathbf{z}_t = \begin{pmatrix} x_a^t & y_a^t & z_a^t & \phi_a^t & \theta_a^t & \psi_a^t \end{pmatrix}^T \tag{3.17}$$

With:

- x_a^t, y_a^t and z_a^t being the x, y, and z position of the UAV's base frame in relation to the assembly extracted from the translation vector of $\mathbf{T}_a^u$.
- ϕ_a^t, θ_a^t and ψ_a^t being the roll, pitch, and yaw orientation of the UAV's base frame in relation to the assembly extracted from the rotation matrix of $\mathbf{T}_a^u$.

With the corresponding observation model $\mathbf{H}_t$ needed for the state vector defined in Equation (3.11).

$$\mathbf{H}_t = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix} \tag{3.18}$$

IMU Measurements
The IMU measures linear velocities and the rotation of the UAV in the roll, pitch, and yaw axes.

With the UAV used in the implementation, the position and orientation are provided relative to a global reference system, with the UAV's world origin as a base (in contrast to most other IMUs that use the UAV base frame). Therefore, they must be transformed into the reference system of the assembly. For the velocity $\mathbf{v}_a^u$, this is achieved using Equation (3.19).

$$\mathbf{R}_w^a = (\mathbf{R}_a^u)^T \cdot \mathbf{R}_w^u \tag{3.19}$$

$$\mathbf{v}_a^u = \begin{pmatrix} \dot{x}_a^u \\ \dot{y}_a^u \\ \dot{z}_a^u \end{pmatrix} = \mathbf{R}_w^a \cdot \mathbf{v}_w^u \tag{3.20}$$

$$\mathbf{R}_a^u = \mathbf{R}_w^a \cdot \mathbf{R}_w^u \tag{3.21}$$

With:

- $\mathbf{v}_a^u$ being the linear IMU velocity in the frame of the assembly.
- $\mathbf{v}_w^u$ being the linear IMU velocity in the world frame.

- $\mathbf{R}_w^a$ being the rotation matrix from world to assembly.
- $\mathbf{R}_a^u$ being the rotation matrix from the assembly frame to the UAV. This value is derived from the filtered model tracking measurement.
- $\mathbf{R}_w^u$ being the rotation matrix of the world frame to the UAV. This value is derived directly from the IMU.

To rotate the orientation measured by the IMU from the world frame to frame of the assembly, it is represented as a rotation matrix $\mathbf{R}_w^u$ and rotated to the reference frame of the assembly using Equations (3.19) and (3.21). The rotational roll, pitch, and yaw angles required for the Kalman Filter update can then be extracted from $\mathbf{R}_a^u$.

$$\begin{pmatrix} \phi_a^i \\ \theta_a^i \\ \psi_a^i \end{pmatrix} = \begin{pmatrix} \text{atan2}(r_{23}, r_{33}) \\ -\text{asin}(r_{13}) \\ \text{atan2}(r_{12}, r_{11}) \end{pmatrix} \tag{3.22}$$

With:

- r_{ij} being the scalar value of the rotation matrix $\mathbf{R}_a^u$ at position i, j.

This leads to the following vector $\mathbf{z}_i$ for the measurement.

$$\mathbf{z}_i = \begin{pmatrix} \dot{x}_a^i & \dot{y}_a^i & \dot{z}_a^i & \phi_a^i & \theta_a^i & \psi_a^i \end{pmatrix}^T \tag{3.23}$$

With:

- $\dot{x}_a^i$, $\dot{y}_a^i$ and $\dot{z}_a^i$ being the x, y, and z velocities of the UAVs's base frame as measured by the IMU rotated into the reference frame of the assembly.
- ϕ_a^i, θ_a^i and ψ_a^i being the roll, pitch, and yaw orientation of the UAV's base frame as measured by the IMU in relation to the assembly.

This also results in a different observation model for the IMU data to associate the velocity measurements with the corresponding elements of the state vector.

$$\mathbf{H}_i = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix} \tag{3.24}$$

When used in conjunction with Equation (2.20), these observation models can be used to calculate the predicted measurements for both $\hat{\mathbf{z}}_{i|k+1}$ and $\hat{\mathbf{z}}_{t|k+1}$.

Both kinds of measurements also require the calculation of their respective measurement noise covariance matrix $\mathbf{R}$ in order to determine their quality. Both $\mathbf{R}_i$ and $\mathbf{R}_t$ are initialized based on the measured covariance of each sensor at rest. While there are ways to update the measurement noise covariance throughout the execution of an inspection,

these values have been found sufficient for achieving a stable position estimate with the system under consideration. Upon arrival of a new measurement, according to the procedure described in Section 2.2.6, Equation (2.22) is used to calculate the Kalman Gain. Afterward, Equation (2.23) is used to correct the prediction of the internal state with the measurement performed by either the IMU or model tracking. Finally, Equation (2.24) is used to update the state covariance to represent the corrected state. Note that both calculating the Kalman gain and updating the prediction is done with the respective measurement noise covariance matrix $\mathbf{R}$, observation model $\mathbf{H}$ and measurement $\mathbf{z}_{k+1}$ of either the IMU- or the model tracking measurement.

While the process was described as strict predict, then update cycle, measurements arrive at different time intervals, or individual measurements may drop out due to pre-filtering (see: Section 3.3.2). In reality, prediction is performed in fixed 30Hz cycles, while sensor value updates are performed whenever new sensor data arrives. As mentioned in Section 2.2.6, the demonstrated concept for sensor fusion is resistant to missing measurements and variable update intervals. As a result, the resulting state prediction is expected to be more precise than the individual sensor measurements on their own.

3.3.5 Position Estimation with Moving or Changing Reference Systems

While the sensor fusion concept proposed in Section 3.3.4 works for stationary or very slow-moving subjects, the accuracy of the estimated position decreases with the movement speed of the assembly. This is caused by the fact that the position measurement of the model tracking is performed relative to the assembly, while the onboard IMU measures absolute velocities (relative to the ground). This causes problems if the assembly that must be inspected moves, i.e., while being transported via a ceiling crane or on a conveyor belt. The IMU of a stationary UAV reports a velocity of 0 $\frac{m}{s}$, while the model tracking interprets an assembly moving away or towards the UAV as a relative velocity. This leads to discrepancies when combining both measurements with a Kalman Filter.

To solve this problem, the movement of the assembly relative to the world is used as an input for the Kalman Filter. To calculate the velocity of an assembly moving on a crane or conveyor belt, the assembly and its means of transportation are modeled using a kinematic chain. This allows the calculation of the speed and orientation of the assembly relative to the world frame, which can be used as an offset for the IMU measurements, so they are used internally as if they were performed relative to the assembly.

Kinematic Modeling of Assemblies

In automated transportation systems like automatic cranes or assembly lines, the crane's position or the conveyor belt's speed is known. Even conventional machines like cranes can easily be retrofitted with sensors relatively easily to give feedback on their current position. A kinematic model of the machine can then be used with its joint positions to calculate the position and speed of all components of the assembly. This is comparable

to the direct kinematics of a robot arm, with the assembly being in the place of the end effector.

Therefore, the Denavit-Hartenberg convention (see: Section 2.2.2) is used to describe these kinematic models as a multitude of individual links connected through rotary or translational joints. These links and joints are connected in alternating order, forming a kinematic chain.

Kinematic Model of an Assembly Moving on a Crane

To provide feedback about the position and velocity of an assembly hanging from an overhead crane (see: Figure 3.10), it is modeled as a kinematic chain with the base frame F_0 located on the runway rail and the assembly itself forming the end of the chain similar to an end effector on a robot system. A linear joint J_1 connects the base frame of the runway rail to the bridge segment with its origin at F_1. The bridge is connected to the trolley's origin F_2 with another linear joint J_2. The connection from the trolley to the hook is defined as a linear joint J_3 from F_2 to F_3. Lastly, the rotary axis of the hook is modeled through joint J_4 from F_3 to F_4, which is located on the assembly but not depicted in Figure 3.10.

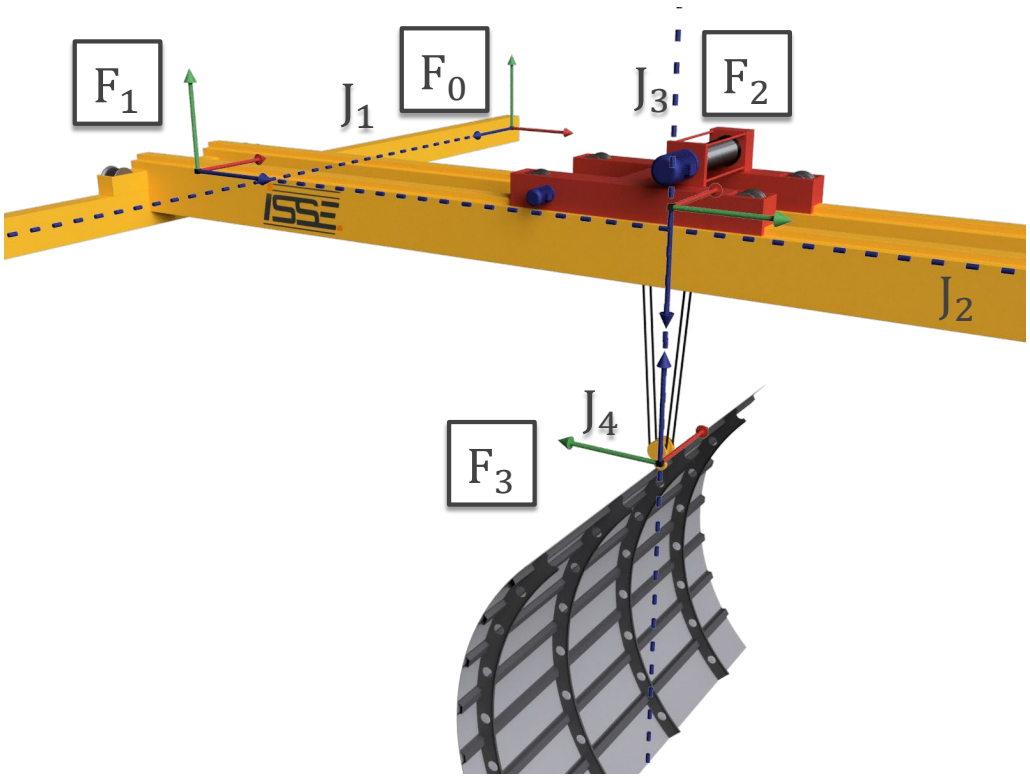


Figure 3.10. Coordinate frames resulting from kinematic modeling of an aircraft fuselage suspended from a crane. The red arrows correspond to the x-axis, green to the y-axis, and blue to the z-axis.

Kinematic Model of a Wind Turbine

A wind turbine can be modeled similarly to the previous example, with the main difference being that the inspection takes place on multiple links. While the assembly hanging from the crane may move relative to the world frame, the transformations between its individual parts stay constant during the inspection. When inspecting a wind turbine, its rotor may be oriented in an arbitrary position relative to the nacelle, making the specification of Points of Interest and navigation using a single reference system difficult. Therefore, Points of Interest are defined relative to the link they are mounted on, and the UAV switches its reference frame for navigation when moving to a different link of the assembly. Figure 3.13 shows the resulting frames of a wind turbine derived from the kinematic modeling technique. The base frame of the tower F_0 is located at the turbine's foundation. The rotatory joint J_1 connects it to F_1 inside the nacelle, allowing the latter to swivel. Joint J_2 is (like all other joints in this example) also rotational. It describes the rotation of the rotor hub on the nacelle and connects to the frame of the hub F_2. The Rotor hub is again connected to each rotor blade (F_3, F_4, and F_5) through joints J_3, J_4, and J_5. For clarity, only J_3 is shown in Figure 3.12. The necessary DH-parameters α_i, a_i, d_i, and θ_i are either derived from the physical structure of the wind turbine or provided through actual joint positions measured by a sensor. The frames do not form a single kinematic chain because the three rotor blades are connected to the same hub. Therefore, F_3, F_4, and F_5 are defined in relation to F_2 with the respective transformations $\mathbf{T}_2^3$, $\mathbf{T}_2^4$, and $\mathbf{T}_2^5$.

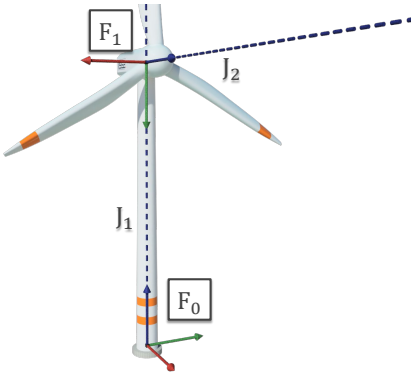


Figure 3.11

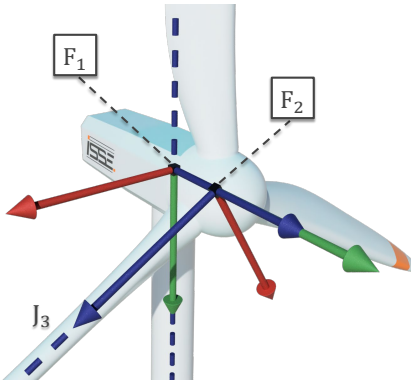


Figure 3.12

Figure 3.13. Coordinate frames resulting from kinematic modeling of a wind turbine. The red arrow corresponds to the x-axis, green to the y-axis, and blue to the z-axis.

State Vector and State Transition Matrix

The state vector $\hat{\mathbf{x}}$, state transition matrix $\mathbf{F}$, initial error covariance matrix $\mathbf{P}_0$ and process noise $\mathbf{Q}_0$ as well as the performed calculations of the prediction step are identical to the original concept.

Odometry Measurement Processing

Compared to a fixed assembly, the rotation of the IMU velocity into the frame of the assembly is more complicated. The IMU's velocity is defined relative to the world origin of the UAV, while the position of the assembly must be constantly recalculated based on the orientation of the base of its kinematic.

Since the kinematic chain's base frame is not necessarily aligned with the UAV's world frame, an additional fixed rotation $\mathbf{R}_w^b$ is introduced between the world frame w and the kinematic base frame b. This value accounts for any constant orientation offset. Since only the IMU velocities and no absolute position measurements need to be transformed, the linear offset between the world frame and the base frame of the kinematic is not relevant.

$$\mathbf{R}_w^b = \mathbf{R}_a^b \cdot (\mathbf{R}_a^u)^T \cdot \mathbf{R}_w^u \tag{3.25}$$

With:

- $\mathbf{R}_w^b$ being the rotation matrix from world to the kinematic base frame.
- $\mathbf{R}_a^b = (\mathbf{R}_b^a)^T$ being the rotation from the assembly frame to the kinematic base frame and the inverse of the rotation of the kinematic chain.
- $\mathbf{R}_a^u$ being the rotation matrix from the assembly frame to the UAV. This value is derived from the filtered model tracking measurement.
- $\mathbf{R}_w^u$ being the rotation matrix from the world frame to the UAV. This value is derived directly from the IMU.

It is assumed that the base of the kinematic frame is fixed in relation to the IMU's world frame. Therefore, $\mathbf{R}_w^b$ is constant and only calculated once as soon as model tracking, IMU, and kinematic data are available.

To express the IMU rotation and velocity in the assembly frame, the data must be first rotated into the fixed base frame of the kinematic (using $\mathbf{R}_w^b$) and then to the frame of the assembly itself. However, $\mathbf{R}_b^a$ is based on the joint states of the kinematic and must be determined on each sensor measurement update by calculating the kinematic transformation using Equation (2.17) and extracting the rotation matrix. The rotation between the IMU velocity and orientation of the IMU can be rotated into the assembly frame using Equations (3.26) and (3.27).

$$\mathbf{v}_a^u = \mathbf{R}_b^a \cdot \mathbf{R}_w^b \cdot \mathbf{v}_w^u \tag{3.26}$$

$$\mathbf{R}_a^u = \mathbf{R}_b^a \cdot \mathbf{R}_w^b \cdot \mathbf{R}_w^u \tag{3.27}$$

With:

- $\mathbf{v}_a^u$ being the linear IMU velocity in the frame of the assembly.
- $\mathbf{v}_w^u$ being the linear IMU velocity in the world frame.

To compensate for the velocity of the assembly, it has to be subtracted from $\mathbf{v}_a^u$.

$$\hat{\mathbf{v}}_a^u = \begin{pmatrix} \dot{xc}_a^u \\ \dot{yc}_a^u \\ \dot{zc}_a^u \end{pmatrix} = \mathbf{v}_a^u - \mathbf{v}_a^a \tag{3.28}$$

With:

- $\dot{xc}_a^u$, $\dot{yc}_a^u$, $\dot{zc}_a^u$ being the velocity vector of the IMU corrected by the component speed vector.
- $\hat{\mathbf{v}}_a^u$ being the corrected velocity vector.
- $\mathbf{v}_a^a$ being the velocity of the assembly relative to the assembly frame.

Therefore, the velocity of the assembly itself is needed. It can be calculated through numeric differentiation of the assembly position relative to its base frame according to Equation (3.29) and is subsequently rotated into the assembly frame using Equation (3.30) to be compatible with Equation (3.28).

$$\mathbf{v}_b^a = \left(\begin{pmatrix} x(k)_b^a \\ y(k)_b^a \\ z(k)_b^a \end{pmatrix} - \begin{pmatrix} x(k-1)_b^a \\ y(k-1)_b^a \\ z(k-1)_b^a \end{pmatrix} \right) \frac{1}{\Delta t} \tag{3.29}$$

With:

- $\mathbf{v}_b^a$ being the assembly velocity in the frame of the kinematic chain's base link.
- $x(k)_b^a, y(k)_b^a, z(k)_b^a$ being the joint position at time step k.
- $x(k-1)_b^a, y(k-1)_b^a, z(k-1)_b^a$ being the joint position at the previous time step $k-1$.
- k being an update step of the IMU data in the Kalman Filter.
- Δt being the time interval between k and (k-1).

$$\mathbf{v}_a^a = \mathbf{R}_b^a \cdot \mathbf{v}_b^a \tag{3.30}$$

The measurement vector $\mathbf{z}_i$ for IMU measurements contains the individual elements of the velocities and the roll, pitch and yaw angles (extracted from $\mathbf{R}_a^u$) of the UAV and is defined in Equation (3.31).

$$\mathbf{z}_i = \begin{bmatrix} \dot{xc}_a^u & \dot{yc}_a^u & \dot{zc}_a^u & \phi_a^i & \theta_a^i & \psi_a^i \end{bmatrix}^T \tag{3.31}$$

The observation model $\mathbf{H}_i$ remains unchanged and Equations (2.20) to (2.24) can be used to update the state and state covariance of the Kalman Filter.

Note that with this implementation, the velocity of the assembly is calculated directly from the kinematic model. Depending on the available sensors for measuring the joint state of the assembly, the structure of the kinematic and the dynamics of the movement, it might be necessary to add the state of the assembly ($\mathbf{v}_b^a$, the position $\mathbf{p}_b^a$ and the roll,

pitch and yaw angles derived from $\mathbf{R}_b^a$) to the state vector of the Kalman Filter, update the state transition matrix accordingly and add update steps for the sensors providing these data to get a filtered state estimate for these values. However, since the assembly's movement during an assembly is most likely not highly dynamic, this approach was not included in the concept.

Model Tracking Measurement Update

The updates performed with the model tracking measurements are performed similarly to the previous sensor update steps. The reference frame of the positions is already correct, as the model tracking must switch the tracked model to the currently inspected link.

The observation model z_t remains unchanged. Subsequently, the Kalman Gain K^t is calculated, and the state vector $\hat{\mathbf{x}}$ as well as the state covariance matrix $\mathbf{P}$ are updated using Equations (2.20) to (2.24).

Control Input Vector

Similar to the measurements of the IMU, the control inputs of the UAV have to be compensated for the assembly velocity before being used in the prediction step. While the navigation algorithms already rotate the control input vector into the correct reference frame, the offset velocity of the assembly still must be subtracted, similar to the compensation of the IMU-velocity described in Equation (3.28).

Individual Filters per Kinematic Link

Usually, the position estimate provided by a sensor fusion algorithm is defined relative to a single reference frame. However, in certain scenarios, an inspection might occur on an assembly with multiple parts that can move relative to each other (like the inspection of a wind turbine). When modeling the entire assembly as a kinematic chain, relative inspection allows the specification of inspection points relative to the link it is located on, which removes the necessity to bring the joints of the assembly into a position that matches the state of the CAD model (e.g. bringing the wind turbine into a parking position). This also allows inspections on multiple links of an assembly that move in relation to each other, using the kinematic model to calculate the required transformations and velocities for the link, the currently inspected Point of Interest is attached to (the currently active link). This also requires tracking this link with the model tracking algorithm, as only this link is guaranteed to be in the camera frame during the inspection of this Point of Interest.

Therefore, to provide a position estimation while navigating across multiple moving links of an assembly, a Kalman Filter is implemented for each link, as described in the previous section. Figure 3.14 shows the architecture of the proposed system. Model tracking data from the images captured by the onboard camera is pre-filtered according to Section 3.3.2 and, together with the IMU data, provided to $N + 1$ instances of the previously described Kalman Filter, with N being the number of joints in the kinematic

model. The $N + 1$th filter provides a position estimate in the global reference frame. Since tracking updates ($\mathbf{T}_n^u$) can only be provided for the currently active link n, a substitute for the tracking measurement is calculated for all other links. This is done by multiplying the $\mathbf{T}_n^u$ to the kinematic transform from link n and m $\mathbf{T}_m^n$ and subsequently extracting the translation vector and roll, pitch and yaw angles. This ensures that all filters receive tracking updates, even if they are not currently tracked. While the extra transformations through the kinematic chain might slightly reduce the quality of the substitution measurements, they still provide a better result than using only IMU measurements when the link is not actively tracked. This helps to prevent big jumps in the position estimate, when the link is tracked again (see: navigation system in tunnel example in Section 2.2.6). Lastly, the IMU updates also need to be rotated to the frame of the current link using $\mathbf{R}_w^m = \mathbf{R}_b^w \cdot \mathbf{R}_w^b$ instead of $\mathbf{R}_w^a$.

The output of all filters is fed into a multiplexer (MUX) that forwards the position estimate that selects the correct value for the reference system the UAV is currently navigating in and provides it to the navigation algorithm, which in turn controls the UAV.

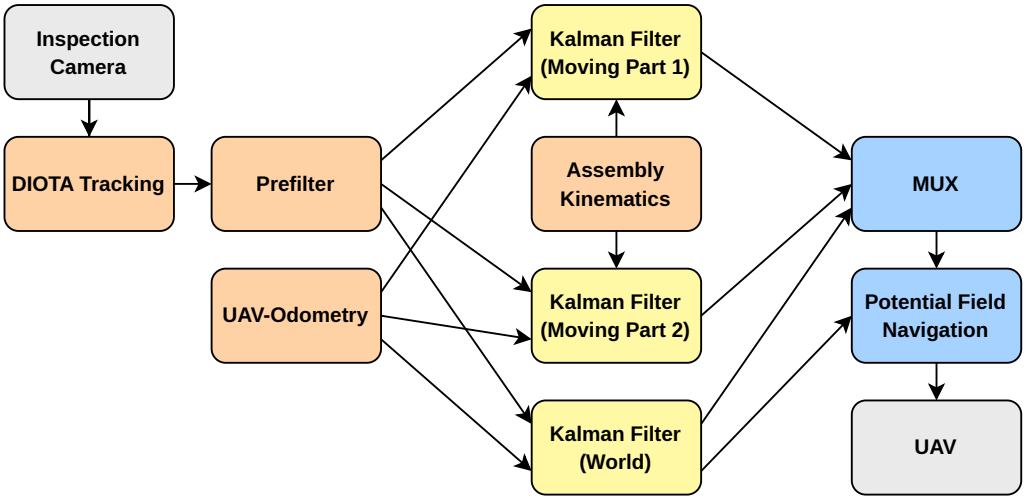


Figure 3.14. Architecture of the advanced fusion model with multiple filters.

When the UAV finishes its inspection on an arbitrary link l_{old} of the assembly and moves to a different link l_{new}, upon switching to a new goal (which is defined in the reference frame of l_{new}), the MUX also switches its output to the filter that provides the position estimate relative to j_{new}. This allows the system to perform even more complex inspection tasks, such as inspecting multiple sections of a wind turbine in operation. This approach allows for specifying all Points of Interest, Viewareas, and Viewpoints in the reference frame of the link they are rigidly connected to.

3.3.6 Related Work on Sensor Fusion for Position Estimation

Fusion of multiple sensor data values to obtain a state estimate of the UAV is a common problem for UAV control for which a variety of solutions are used in both industry and scientific literature. The company Luftronix, which specializes in automated inspections and scanning using UAVs in industries like communication, aviation, oil and gas holds a patent for the fusion of optical flow, inertial, vision and depth-sensor data for the purpose of autonomous flight in complex environments [211]. Their approach combines optical flow data as well as optical and thermal images from the UAV augmented with depth data from distance sensors. The camera images are analyzed for *points of greatest contrast*, which are tracked over time. The distance to these points of contrast is then measured using either laser- or sound waves (most likely using a LiDAR Sensor). The patent describes that the distance to these points is used to adjust the UAV's path to avoid collisions but does not go into detail on how a position is determined using these measurements.

In scientific literature, Marković et al. [179] present a system for UAV localization in indoor environments, where GPS signals are not available, and magnetometer measurements cannot be used due to metal structures in proximity distorting the earth's magnetic field. Their approach uses an Error State Extended Kalman Filter (ES-EKF) to fuse sensor signals of a LiDAR sensor, visual odometry data generated by a stereoscopic camera and an ultra-wideband (UWB) localization system. The LiDAR sensor uses a SLAM algorithm [111] to determine the UAV's position. The UWB works similar to a Global Navigation Satellite System (GNSS) like GPS but on a smaller scale by trilateration of radio signals transmitted between the tracked object and a number of base stations. The developed system was validated in a laboratory tests and was able to compensate for drift in the LiDAR-based SLAM localization and combine the strengths of the individual sensors.

Karaked et al. [134] use a similar approach to combine LiDAR SLAM and Visual SLAM provided by a stereoscopic camera to estimate the position of an UAV with an Extended Kalman Filter. Their general idea is to combine the strengths of both used sensor types and eliminate their weaknesses. Stereoscopic cameras need sufficient light to function while LiDAR sensors often fail in geometrically extremely complex or repetitive regions (like a long hallway). The authors use an Extended Kalman Filter provided by the ROS `robot_localization`-package [82] to estimate a 2D position as well as the yaw-orientation of the UAV. They validate their approach with different test scenarios in which one of the used sensors is not able to provide an accurate position measurement. The state estimate of the filter is able to provide a correct position measurement even if one of the used sensors is producing invalid measurements achieving a positional accuracy of 0.06 m to 0.56 m.

Santana et al. [242] estimate the position of a UAV using a Kalman Filter which fuses inertial and landmark-based visual position data. The state vector of the filter contains the position and velocity of the UAV in x, y, z and ψ and the state transition matrix is structured analogously to Equation (3.13). Sensor data is also rotated into the correct reference frame using a technique similar to the one presented in this work. While

acknowledging that an Extended Kalman Filter is the most commonly used approach for estimating a UAV position in this case, they state that they were able to calculate good predictions for the system state using a normal Kalman Filter. The system was evaluated together with the also proposed position controller by flying different trajectories. During the flights, the landmark for visual positioning was not visible in parts of the trajectory, which means that only the inertial sensor data and state prediction was used in these sections. The unfortunately no direct evaluation of the positioning accuracy was performed, but in test flights, the UAV was able to follow the specified trajectory with a root-mean-square error of 0.04 m to 0.08 m per axis with the authors stating that the error can be traced back to insufficient parameterization of the control system suggesting that the accuracy of the position estimation might be higher than these numbers.

Zolotukhin et al. [304] use an Extended Kalman Filter for flying trajectories using a *Parrot AR.Drone* UAV equipped with a camera. The filtering algorithm combines the sensor data of the onboard IMU and camera-based odometry. The Extended Kalman Filter was used due to nonlinearities between the UAV orientation and acceleration measurements by the IMU. The authors also demonstrated a proof of concept of their approach and were able to fly trajectories with a positional error of 0.1 m to 0.2 m using the developed system. A similar system using two cameras and an IMU was developed by Liu et al. [164]. Shen et al. [261] propose a system for vision based flight with a UAV using two cameras and an IMU. The approach works by processing both camera images as a stereoscopic image on an on-board computer to estimate an orientation and position. These values are then used together with the IMU data in an Unscented Kalman Filter to estimate the final position. Unique to this approach is the use of different updates of both cameras due to the low processing power of the onboard-computer. While the primary camera image is processed at a rate of 20 Hz, the stereoscopic depth information is only processed at a rate of 1 Hz on each update of the secondary camera. The created system was able to fly multiple trajectories with speeds of up to 4 $\frac{m}{s}$ and pitch angles of up to 20°.

When working with more complex systems, a single filter might not be enough for sufficient state estimation. In literature, Multiple Model Adaptive Estimation (MMAE) [65] or Interacting Multiple Model (IMM) [11] algorithms are often used in these cases. In both methods, multiple filtering algorithms (like a Kalman Filter) create a state estimate and the extent to which of the models is influencing the final estimate is based on how well the individual filter's prediction matches the actual measurement.

While the majority of work presented in this section uses either Extended Kalman Filters (EKFs) for sensor fusion, it is necessary to mention why this work (as well as the work by Santana et al. [242]) are able to use the simpler Kalman Filter for this task. While a regular Kalman Filter is used for linear systems, an EKF is (to a certain extent) able to work with nonlinear systems by linearizing the system around the operating point [66, 263]. The two main nonlinearities involved in position estimation of a UAV arise from the need to rotate or transform sensor values obtained in the wrong reference frame and the nonlinear relationship between the angle of the UAV and its acceleration as well as the processing of orientation information based on raw IMU sensor values.

The approach presented in this work handles the conversion of the rotation and transformation of sensor measurements into the correct frame before they are being used in the update step of the filter. While also practical because the middleware used in the implementation provides functionality to do this, this approach also does not require changes to the filter parameterization itself if sensors are changed or relocated allowing for higher reusability.

The nonlinear relationship between UAV orientation and acceleration is not relevant to the system proposed in this approach, as the IMU of the used drones provides a preprocessed orientation and velocity measurement (presumably calculated using an EKF), which provide sufficient data for a measurement update. When working with a less sophisticated IMU a manual linearization around a fixed operating point might be sufficient for the low pitch and roll angles involved in a relatively slow inspection trajectory.

3.4 Relative Navigation Along Assemblies

This section discusses the navigation strategies developed to navigate relative to a structure. In contrast to most conventional approaches, the Viewpoints used as Waypoints are defined relative to the assembly and do not use absolute coordinates. This allows for inspecting a component independent of its absolute position or reusing an inspection path for multiple identical objects. Therefore, an assembly does not need to be positioned at an exact predefined point in a production facility, which allows for flexibility on when the inspection is performed (i.e., in a manufacturing station, on a storage site, or while hanging from a crane). Also, when performing the inspection of a wind farm like described in Section 1.2.2, only a single inspection path is needed when performing inspections for an arbitrary number of identical wind turbines. This allows more time to be spent on computing an efficient inspection path because the path only needs to be calculated once.

In the context of this work, *navigation* describes the process of flying the UAV from its current position a to an arbitrary point b. These points can be Viewpoints, start- and landing points, or support points introduced into the path to facilitate navigation. During the cause of an inspection, the UAV navigates to each Viewpoint in the order calculated by the algorithm described in Section 3.2.2. While performing the navigation, the UAV must avoid collisions with obstacles of any kind. Therefore, a system was developed for detecting obstacles and retaining their position over time (see: Section 3.4.3). Also, due to limitations on flight time, the navigation algorithm for the UAV must minimize the time necessary to reach a waypoint but not at the cost of spending an excessive amount of energy for fast acceleration and breaking maneuvers. An additional requirement unique to an inspection scenario is the consideration of the camera orientation. In order to perform a successful inspection, the camera has to be pointed at the current Point of Interest at all times to offer inspection tools that have the best chance of detecting faults at the Point of Interest.

This section describes the two navigation algorithms that were developed and tested with the system. First, the adaptation of a more traditional approach for obstacle-avoiding navigation using the potential field algorithm is presented. Subsequently, a navigation strategy using Anytime Dynamic A*, a more recent search algorithm, and the potential benefits of this solution over the potential field algorithm are discussed.

3.4.1 Potential Field Navigation

The first navigation strategy that was adapted to the inspection system is the potential field navigation [145]. When adapted to the inspection use case, it uses an attractive potential to pull the UAV towards its goal while pushing it away from all obstacles using a repellant potential. While the general idea has been already published in previous work [248], this section covers the topic in more detail. The navigation process can be imagined as the UAV following the gradient in the potential field created by the goal and all obstacles. A more detailed explanation of its function was provided in Section 2.3. This section focuses more on adapting the algorithm for three-dimensional flight using

a UAV in a dynamic environment with both moving and stationary obstacles that are mostly not known in advance.

One significant drawback of the potential field method is the possibility of the drone getting stuck in a local minimum of the potential field. In certain situations, the UAV can navigate to a point where the resulting force of all potentials acting on it is zero and the UAV gets stuck. In these cases, additional strategies need to be implemented to navigate out of the local minimum.

A property that can cause problems when the algorithm is applied to UAVs is the quick change in resulting forces applied to the UAV when multiple obstacles are close by. This can result in unexpected behavior in the form of quick acceleration movements that can cause overcorrection in the lower level attitude and rate controllers, which can result in crashes. A simple example of this phenomenon is if the UAV tries to fly through a hole towards a goal. In this example, the hole is just about big enough for the UAV to be pulled through by the attractive potential of the goal. First, the repelling forces of the hole-shaped obstacle almost cancel out the attractive force of the goal, resulting in the UAV flying very slowly into the hole. Once it reaches the center of the hole, the direction of the forces applied by the hole quickly aligns with the attractive force, now amplifying it instead of almost canceling it out. This results in the UAV almost being catapulted out of the hole, which can lead to an unstable flight or a crash. The approach presented here implements a measure that prevents these extreme speeds when navigating close to obstacles. This is covered in more detail later in this section. The approach discussed in this work extends the functionality of the original potential field algorithm by adding a third dimension. Compared to the ground-based robot used as an example in the original algorithm, UAVs navigate in 3D space, which made this change necessary.

Analogous to the original potential field navigation, the force applied to the UAV is calculated as the sum of the attractive force of the target and all repelling forces of all obstacles. Therefore, attractive force Vector $\mathbf{F}_t$ of the target is calculated using Equation (3.32).

$$\mathbf{F}_t = \mathbf{F}_{\text{ct}} \cdot \begin{pmatrix} \frac{x_t - x_{\text{u}}}{d_t} \\ \frac{y_t - y_{\text{u}}}{d_t} \\ \frac{z_t - z_{\text{u}}}{d_t} \end{pmatrix} \tag{3.32}$$

With:

- d_t being the distance between UAV and target.
- $\mathbf{F}_{\text{ct}}$ being the attractive force constant.
- x_t, y_t, and z_t being the x-, y- and z-coordinates of the target.
- x_{u}, y_{u}, and z_{u} being the x-, y- and z-coordinates of the UAV.

Compared to the original equation, the vector was extended by the z-dimension.

Calculation of Repellant Forces

Due to the transition from 2D to 3D space, the number of obstacles that need to be considered to calculate repelling forces is way higher. In practice, this can cause significant increases in execution times for the algorithm, leading to control signals not being available in time. To mitigate this problem, only obstacles within a radius of 3 m around the UAV are used. This is an acceptable compromise because the potential of obstacles that are farther away is so small that it does not significantly influence the resulting force vector. In contrast to the original two-dimensional, grid-based approach, an OctoMap represents the obstacles in 3D space. To further improve calculation speed and simplify the concept, the original certainty value that describes the probability of a cell inside the grid being an obstacle is replaced with a binary (obstacle / free) state for each voxel.

The overall repellant force Vector $\mathbf{F}_r$ is calculated using Equation (3.33).

$$\mathbf{F}_r = \frac{1}{n_{\text{obs}}} \cdot \sum_{i=0}^{n_{\text{obs}}} \frac{\mathbf{F}_{\text{cr}}}{d_i} \cdot \begin{pmatrix} \frac{x_i - x_{\text{u}}}{d_i} \\ \frac{y_i - y_{\text{u}}}{d_i} \\ \frac{z_i - z_{\text{u}}}{d_i} \end{pmatrix} \tag{3.33}$$

With:

- d_i being the distance between UAV and obstacle i.
- i being the obstacle.
- $\mathbf{F}_{\text{cr}}$ being the repulsive force constant.
- n_{obs} being the number of obstacles.
- x_i, y_i, and z_i being the x-, y- and z-coordinates of the obstacles.
- x_{u}, y_{u}, and z_{u} being the x-, y- and z-coordinates of the UAV.

Compared to the original approach, $\frac{1}{n_{\text{obs}}}$ norms the resulting force vector, so the resulting force does not increase with a large amount of obstacles nearby. Also, the certainty value is not included as it is always 1, and the constant for the width of the robot is hidden inside $\mathbf{F}_c r$.

In the case that no obstacles are present inside the observed radius ($n_{\text{obs}} = 0$), $\mathbf{F}_r$ is $\mathbf{0}$ to avoid division by 0. The resulting force $\mathbf{R}$ can be calculated using Equation (3.34).

$$\mathbf{R} = \mathbf{F}_r + \mathbf{F}_t \tag{3.34}$$

This force is then converted into a control velocity for the UAV. This is done by first determining the absolute (as in: without directional information) velocity using Equation (3.35).

$$v_{\mathrm{u}} = \min(d_{\min} \cdot v_{\mathrm{adv}}, v_{\max}) \tag{3.35}$$

With:

- v_{u} being the velocity of the UAV.
- $d_{\min}$ being the distance to the closest obstacle.
- v_{adv} being a constant factor for obtaining the advised velocity based on the distance to an obstacle.
- $v_{\max}$ being the maximum possible velocity of the UAV.

The velocity is then converted into a vector that can be passed as a control input for the UAV using Equation (3.36).

$$\mathbf{v}_{\mathrm{u}} = \frac{1}{\|\mathbf{R}\|} \cdot \mathbf{R} \cdot \mathbf{v}_{\mathrm{u}} \tag{3.36}$$

With $\mathbf{v}_{\mathrm{u}}$ being the velocity vector of the UAV.

This results in the speed of the UAV becoming slower the closer it gets to an obstacle. Only the direction of the calculated potential is used to navigate the UAV into the right direction. This is contrary to the behavior of the original potential field method, in which closer distances to obstacles result in larger repelling potentials and, thus, larger velocities. This is done as a safety precaution to prevent unexpected sudden movement of the UAV when navigating in an environment with many close-by obstacles.

Orientation of UAV and Gimbal

In addition to calculating a linear velocity vector, the UAV also expects a rotational velocity for its yaw-axis and a pitch angle for its gimbal. The yaw angle of the gimbal is set to 0°relative to the UAV, so the camera always faces in the direction of the UAV while the UAV actively controls the gimbal roll angle to keep the cameras' horizon level. For controlling the yaw-orientation of the UAV, a limited P-Controller that outputs a yaw-rate is used. The calculation of the yaw rate set point $\dot{\psi}_{\mathrm{sp}}$ that is sent to the UAV is done using Equation (3.37).

$$\dot{\psi}_{\mathrm{sp}} = \max(\min(e_\psi * P_{\dot{\psi}}, \dot{\psi}_{\max}), -\dot{\psi}_{\max}) \tag{3.37}$$

With:

- e_ψ being the yaw position error $\psi\mathrm{sp} - \psi\mathrm{u} \mod 360$.
- $P_{\dot{\psi}}$ being the proportional gain of the controller.
- $\dot{\psi}_{\max}$ being the maximum absolute yaw rate for limiting the output.

The gimbal's pitch is set to the relative angle from the gimbal position to the Point of Interest. Since the UAVs used for implementing the system control gimbal positions internally, it is directly sent to the UAV as a setpoint.

Fallback Strategy when Encountering Local Minima
As mentioned before, one significant drawback of the potential field algorithm is the possibility of the robot getting stuck inside a local minimum. To work around this problem, a way of detecting this scenario is needed in conjunction with a strategy for navigating out of the local minimum. The detection of the local minimum is done by observing the linear velocity of the UAV over a longer period of time. In practice, this is done by continuously calculating the sliding average over the change in position of the UAV. If the average position change over the last 20 time steps has been less than 0.001 m, the UAV is believed to be stuck in a local minimum. With an update rate of 10 Hz, this corresponds to a time of 2 seconds with an average UAV speed of less than $1\ \frac{cm}{s}$. To avoid the detection of false positives, the speed is only monitored when the UAV is navigating (i.e., has not arrived at the Viewpoint yet).

A random walk is performed as a simple strategy to navigate out of the local minimum once it is detected. This temporarily replaces the target for the potential field navigation with an arbitrary point within a radius of 7 m around the UAV. The UAV then navigates towards this point for a fixed duration before the target is restored to the original waypoint. The basic idea is that by then, the UAV has altered its position enough not to re-encounter the same local minimum. If it does anyway, the process is repeated until it finally reaches its goal.

3.4.2 Anytime Dynamic A*

The potential field navigation's susceptibility to getting stuck in local minima, in conjunction with the crude solution using a random walk strategy, motivated the exploration of the Anytime Dynamic A* algorithm as a possible alternative. While the basic working principle of the Anytime Dynamic A* algorithm has been discussed in Section 2.3.1, the following paragraphs cover the modification of the algorithm to the problem of planning an efficient path from one Viewpoint to another.

The general idea of the approach is to quickly provide an initial path calculation to the UAV, which continuously gets updated through replanning when new environment updates in the form of obstacle sensor measurements arrive.

Sampling of points for the graph

For representing the environment, the graph nodes are distributed in a uniform grid with equidistant spacing. This simplifies calculations, storage, and the search itself while also increasing performance in various places. The resolution r specifying the voxels per meter of each axis allows calculating the total number of voxels contained in an environment using Equation (3.38).

$$N = l \cdot w \cdot h \cdot r^3 \tag{3.38}$$

With:

- N being the total number of voxels.
- l, w and h being the dimensions of the area in meters.

With a value for r of only two and the dimensions of the available flight area for laboratory testing of 8 m x 16 m x 6 m this equates to 6144 voxels and therefore nodes in the graph. Due to the increasing runtime with the number of nodes and edges, nodes are only connected with their six direct neighbors instead of the 26 possible when considering all possible diagonal connections.

Possible UAV positions are represented in the form of a voxel grid instead of using a graph. This enables their storage in dictionaries, which ensures an access time within $O(1)$. Dictionaries also allow for storing their corresponding data only as needed, drastically reducing memory consumption. The function $f : \mathbb{R}^3 \to \mathbb{Z}^3$ mapping a position $\mathbf{p}$ to a grid index $\mathbf{v}$ is described in Equation (3.39).

$$f(\mathbf{p}) = \begin{pmatrix} \lfloor r \cdot p_x \rfloor \\ \lfloor r \cdot p_y \rfloor \\ \lfloor r \cdot p_z \rfloor \end{pmatrix} \tag{3.39}$$

In addition, the function $i : \mathbb{Z}^3 \to \mathbb{R}^3$ is defined to convert a grid index $\mathbf{v}$ to the coordinates of the center of the corresponding voxel.

$$i(\mathbf{v}) = \frac{\mathbf{v}}{r} + \frac{1}{2r} = \begin{pmatrix} \frac{v_x}{r} \\ \frac{v_y}{r} \\ \frac{v_z}{r} \end{pmatrix} + \frac{1}{2r} \tag{3.40}$$

Edges of each node are also represented as a dictionary to ensure an access time within $O(1)$.

Heuristics and Cost Function

The Euclidean distance is used as the heuristic function to estimate the remaining distance to the goal. Since the implementation uses six neighbors for each voxel, the edge cost between neighbors that are not marked as an obstacle is identical for all nodes. Therefore, the only information that must be stored as the edge cost is whether the edge can be used for a path. The edge cost representation in a dictionary allows only storing an infinite cost value for blocked edges, with all unpopulated values assumed to have

a cost of 1. The cost function $cost(s, s')$ for traveling to neighboring nodes s and s' is universally defined by Equation (3.41).

$$cost(s, s') = \begin{cases} 1 & \text{if there is no obstacle between s and s'} \\ \infty & \text{otherwise} \end{cases} \tag{3.41}$$

Replanning After Environment Updates
After each obstacle sensor measurement, the detected obstacle measurements are transformed from the local coordinate frame of the UAV to the global coordinate frame used for the calculations used in the AD* algorithm. The corresponding position in the voxel grid for each obstacle measurement is then calculated using $f(\mathbf{p})$ defined in Equation (3.39). The affected node is then marked as invalid and its edge costs to all neighbors set to ∞. Additionally, all edge costs of its neighbors to the affected node are also set to ∞, and the nodes are added to the open queue so that replanning can occur. As described in Section 3.4.3, the obstacle representation is stored in an OctoMap data structure, which clears occupied voxels once the UAV's sensors detect no more obstacles in a particular area. To account for this, a similar replanning strategy is used here: occupancy values are deleted from affected edges, invalid markers are removed, and affected nodes are added back into the open list.

Generating a Velocity Command for the UAV
To maintain compatibility with the potential field navigation, the algorithm must output the velocity vector $\mathbf{v}_a^u$ for the UAV to follow. This vector is first calculated in the global frame as $\mathbf{v}_w^u$ using Equation (3.42).

$$\mathbf{v}_w^u \begin{cases} \|\mathbf{p}_{next} - \mathbf{p}_u\| \cdot v_{max} & \text{if} f(\mathbf{p}_u) \neq f(\mathbf{p}_{next}) \\ \mathbf{p}_{next} - \mathbf{p}_u & \text{otherwise} \end{cases} \tag{3.42}$$

With:

- N being the total number of voxels.
- l, w and h being the dimensions of the area in meters.
- v_{max} being the maximum speed of the UAV.

This results in the UAV flying towards the direction of the goal with maximum speed if it is still located inside a neighboring voxel and gradually reducing speed when arriving inside the target voxel. To be usable in navigation, the velocity vector $\mathbf{v}_w^u$ is rotated from the global coordinate system using Equation (3.43) to the coordinate system of the assembly.

$$\mathbf{v}_a^u = \mathbf{R}_w^a \cdot \mathbf{v}_w^u \tag{3.43}$$

With:

- $\mathbf{v}_a^u$ being the velocity measurements in the reference frame of the assembly.
- $\mathbf{R}_w^a$ being the rotation matrix from assembly to the global frame.

This ensures compatibility with the potential field navigation, which also outputs its velocity in the coordinate frame of the assembly. The hardware interface layer for the specific UAV performs the transformation of this vector to either the global world frame or the local coordinate frame of the UAV depending on which is required using the respective rotation matrices.

The yaw orientation of the drone and camera gimbal orientation is calculated as described in Section 3.4.1 to ensure that the camera is always facing towards the Point of Interest.

3.4.3 Collision Avoidance with Limited Sensors

When inspecting an industrial or production environment, a UAV encounters a highly dynamic and unknown environment. The constant work done on the facility makes it impossible to model all obstacles the UAV could encounter in advance. For example, a worker's tool cart can easily pose as an obstacle if it is left inside the planned path of the UAV. The position of this tool cart may change several times over the course of a day as the worker it belongs to works on different areas inside the facility. This makes it impossible to consider as an obstacle during the planning phase. The same applies to the same extent to the workers themselves, who constantly move around the facility, and a collision with them should be avoided at all cost.

A second factor specific to relative inspection is that when using the full potential of the relative positioning, the assembly that must be inspected can be positioned freely inside the production facility. This makes considering any obstacles that are not fixed to the assembly (like, for example, a stand) impossible at the planning phase. Therefore, the inspection UAV must be equipped with an effective way to detect and avoid obstacles in its surroundings. The focus of this work is on performing inspections with small consumer UAVs inside of structures like airplane fuselages. Their smaller size comes at the cost of lower quality obstacle sensors. While larger UAVs can easily be equipped with 360° LIDAR scanners that offer high-resolution distance measurements to the closest obstacles, smaller UAVs are fitted with lower quality sensors like sonar rangefinders [208] that have a much larger detection radius and make localizing an obstacle harder. These sensors often also lack the range a proper LIDAR scanner could provide, making obstacles only visible to the UAV at a closer distance. In addition, the API provided by these UAVs might further process and abstract the raw sensor

data to provide a more *user-friendly* way to use the data. The API of the DJI Mavic 2 UAV [253] for example only provides processed sensor measurements in the form of distance measurements of each four sectors in the front and back and a distance to the closest obstacle on both sides. Access to the raw values of the individual obstacle sensors is not possible. This highly abstracted, low-resolution data poses a challenge for detecting and avoiding obstacles. An additional challenge is that obstacles detected at one point of the inspection might not be present throughout the entire duration of the inspection flight. An example of this would be a tool cart that is moved by a worker while the UAV is performing an inspection. While the cart might be in the way of the UAVs' path to the assembly, making it necessary to plan its trajectory around the obstacle, a worker might notice this and move the cart. This means that the UAV could take the direct path back. Solely marking an area as occupied by an obstacle once and, from then on, moving around this area results in a suboptimal trajectory in this case. An even more extreme example would be a worker crossing in front of the UAV. This would cause obstacle detections in front of the UAV that would cover a vast area if just stored and never checked again, resulting in an unnecessarily long trajectory while avoiding an obstacle that is not there anymore. Therefore, the following sections cover the concepts developed for storing and updating obstacles to avoid such problems and give an overview of comparable solutions in literature.

World Representation in OctoMaps

To increase the effectiveness of the obstacle avoidance system, it is beneficial to not only consider obstacles currently detected by the UAV but also keep a history of previously detected obstacles in memory. Hence, An OctoMap (see: Section 2.4) was chosen to store the position of all areas that the UAV believes to be obstructed. It achieves this by representing the presence or absence of an obstacle assigning each voxel a corresponding value. The following paragraphs explain how obstacles that are known before the inspection can be preloaded into this map to improve navigation performance, as well as how the information inside the map can be updated with live sensor measurements.

Using information about Assembly and Surroundings

One way to significantly improve the knowledge about obstacles right from the start of the inspection is to preload information about the assembly and the environment into the OctoMap before takeoff. By doing this, potential dead ends caused by these objects are known in advance and can be used for trajectory planning directly from the start. Otherwise, the UAV would have to explore each dead end first to receive measurements that a particular path cannot continue beyond the dead end. The insertion of preoccupied spaces can be done by converting the CAD- or meshed model used for model tracking into a voxel representation. When doing the conversion, the main parameter is the resulting voxel size. This parameter is set to the same value as in the OctoMap. This voxel representation can then be inserted into the OctoMap at the beginning of the inspection and serve as additional help for trajectory planning. Using this procedure, the model and other obstacles, such as supporting structures or the entire production facility, can be preloaded as obstacles.

Updating the OctoMap with Sensor Data
During flight, the obstacle sensor measurements of the UAV constantly update the OctoMap. Each new sensor measurement causes voxels to be added to the map. The inaccuracies and wide area of measurement of the obstacle avoidance sensors make direct insertion of an obstacle measurement at the point of detection not particularly effective. Instead, a more complex strategy is needed that is also specific to the concrete type of sensor and UAV. The strategies developed for the UAVs used in this work are covered in detail in Section 4.2.2. The general idea of these strategies is to find a solution for dealing with the problem that a detected obstacle could be anywhere inside the detection area. For sensors with particularly large sectors, the solution is to insert only one voxel at the center of the sector at the detected distance. When using potential field navigation, a single obstacle at that point is enough to keep the UAV clear of the area. With AD*, the voxel size for the planning graph has to be set sufficiently large to ensure that the UAV does not try to use the space that is not marked as occupied but is still within the detection range of the sensor as this might cause a collision.

Dynamic obstacles pose a different challenge to the system. If obstacles are only inserted into the OctoMap and never removed, a worker crossing the UAVs' path from its left to right would cause many obstacle detections. This would result in a wall of obstacles inside the map requiring the UAV to fly above it even when it is unnecessary. Therefore, a means of removing voxels that are no longer relevant is needed. This can be implemented by removing all existing voxels located between the UAV and the current obstacle distance measurements of the UAV. This is because if an obstacle is detected in an arbitrary sector of the UAVs' obstacle sensor, it is safe to assume that no other obstacle with a closer distance can exist in this sector because, otherwise, it would be detected and reported as the closest obstacle. To detect voxels that can be removed, ray casts [237] are performed from the UAV's position through the area sector of the obstacle Sensor. The rays that collide with a voxel at a distance closer to the UAV than the current measurement are removed. The wide sectors of detection from cheaper UAVs are advantageous here as the wider sectors clear out large areas with this strategy.

However, special care must be taken when the assembly has to be inspected while moving. There are two possible ways to store the obstacle data in the OctoMap. Voxel positions can either be defined in the reference frame of the assembly or relative to a global reference frame positioned at a fixed point of the factory. When choosing the assembly as a reference frame, in case of a moving assembly, all voxels caused by the environment (either measured or preloaded) lose their validity because they move relative to the assembly. On the other hand, when representing the voxel data in a global reference frame, their absolute position stays constant. This requires defining a global frame that can be set arbitrarily and mainly serves to align the world and assembly voxels. Also, the transformation from assembly to the world frame must always be known to calculate the transformation from UAV to assembly (via its relative positioning approach) and from there to the world frame to insert measured obstacle data at the correct position in the world frame. When representing the voxel data in a global frame, assuming the transformation from UAV to world via the assembly is correct, the obstacle data of the stationary surroundings stays valid when the assembly

is in motion. However, the assembly's obstacle data is invalidated when the assembly is moved. This can be counteracted relatively easily by removing the voxels that were previously occupied by the assembly and reinserting the voxels at the new position of the assembly in the world frame. Therefore, this architecture was chosen for the final implementation.

Using the World Representation for Navigation

To use the detected obstacles for navigation, a two-stage obstacle avoidance system is proposed. The general idea of this system is the combination of an obstacle avoiding navigation strategy combined with an emergency collision avoidance system that interferes if the obstacle avoiding navigation fails. Figure 3.15 shows the general Archi-

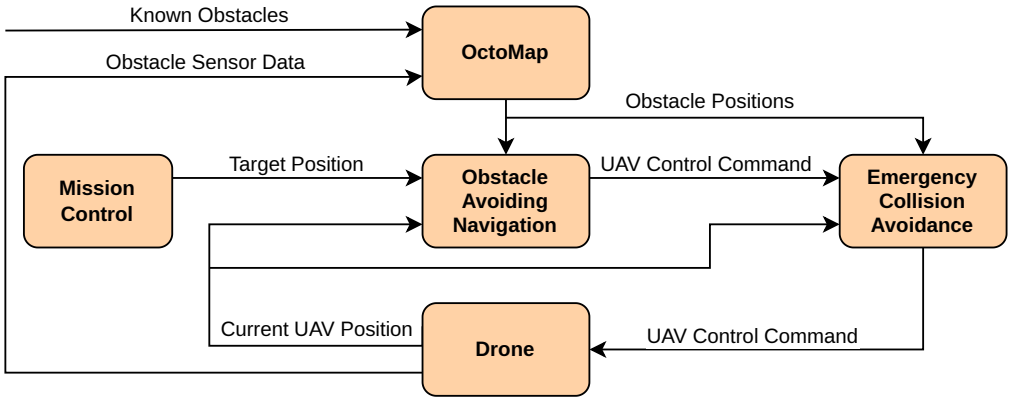


Figure 3.15. Overview of the obstacle avoidance system architecture.

tecture of the system. Obstacle positions are provided to the OctoMap in the form of previously known obstacles and sensor data provided by the UAV. The obstacle positions accumulated by the OctoMap are passed to the obstacle avoiding navigation, which uses the current UAV position alongside a target position provided from the mission controller. The navigation system then generates a control command as a target velocity vector for the UAV, which is sent to the emergency collision avoidance system. This system also accesses the obstacle positions provided by the OctoMap and monitors the position of the UAV. During regular operation, the emergency collision avoidance system forwards the control command to the UAV. In case the obstacle-avoiding navigation system fails and tries to maneuver the UAV too close to an obstacle, the emergency collision avoidance system overrides the control command velocity vector to $\mathbf{0}$, which causes the UAV to stop.

3.4.4 Related Work and State of the Art

According to Yasin et al. [297], four different types algorithms for collision avoidance systems exist:

- **Geometric Algorithms** use relative positions and relative velocities between the vehicle and other vehicles / obstacles to calculate whether a collision is about to happen. Based on this information, the flight path is altered in a way that prevents the collision. An example would be the use of collision cones [36, 68, 69]. Watanabe et al. [294] use an onboard camera and a Kalman Filter to detect and track obstacles close to the UAV. The collision cone, a set of relative velocities between two moving objects that will lead to a collision is then calculated and if the actual velocity lies within this cone, it is altered in a way to move outside the cone.
- **Optimization Based Algorithms** try to calculate an optimal or near-optimal path for the UAV without colliding with other UAVs and obstacles. Due to their computational complexity, the paths are usually precalculated and replanning is often not possible or requires a replanning of the entire path, which makes them only suitable for environments without dynamic obstacles. An exception to this is the AD* algorithm described in this work as it is able to dynamically and efficiently replan only parts of the planned path that are affected by the obstacle.
- **Sense and Avoid Algorithms** are used to quickly react using a computationally simple solution to avoid an obstacle. To achieve quick reaction time, these approaches only work based on the provided local sensor data and do not use the global planning for the calculation of the UAV's movement. This quick reaction time comes at the cost of the possibility of a globally suboptimal path and the susceptibility to local minima.
- **Force Field Algorithms** are based on virtual forces that attract or repel the UAV. Usually, obstacles exert a repulsive force on the vehicle while the goal attracts the UAV. The movement direction and velocity of the UAV is derived by the direction and length of the sum of all forces.

Since sense and avoid algorithms are more focused on reacting quickly to obstacles and tend to neglect following an ideal trajectory for this purpose, they are mainly used in highly dynamic environments where the trajectory calculation is simple.

UAV-based inspections of structures as proposed in this work are performed at relatively low speeds and do not require especially quick reaction times. However, the complex structure of the assemblies themselves and the production facilities in which they are inspected require navigation algorithms that focus on efficient trajectories to perform the inspection successfully and within the limited battery life of the UAV.

Therefore, the following sections will mainly focus on the state of the art of force field algorithms as well as Graph Search and RRT-based obstacle avoiding algorithms since they are used most widely for this application.

RRT-Based Algorithms

Algorithms based on a rapidly-exploring random tree (RRT) [150] use a probabilistic approach that continuously extends a tree from a specific goal until the starting point is reached by one of the leaves. The planning is often performed once before the trajectory is executed, which prevents the UAV from reacting to dynamic obstacles. To

circumvent this, the trajectory can be replanned every time an obstacle occurs, which can be computationally expensive, preventing the use of the algorithm in highly dynamic environments like a manufacturing facility. Ramon-Soria et al. [230] use RRT* [135] in their solution for inspecting infrastructures using UAVs to plan trajectories between individual Viewpoints. The drone navigates relative to a 3D map of the environment that must be provided to the planning algorithm. How such a map may be created is not mentioned but based on the images included in the paper, the map might be a three-dimensional point cloud scan obtained by a manual mapping flight of a similar drone.

Bircher et al. [24] also uses a Viewpoint-based navigation algorithm for coverage inspections of structures performed by UAVs. Their approach first checks if obstacles exist between two consecutive Viewpoints. If no obstacles are present, the UAV directly flies to the next Viewpoint in a straight line. If the direct path is obstructed by obstacles, an obstacle-free path is planned using the RRT*-algorithm.

Graph-Based Search Algorithms

Graph-based search algorithms can be used to find an optimal or near-optimal path between two points in a graph. To be used in 3D-navigation, discrete geometric points are sampled and used as nodes to create the graph (see: Section 2.3.1). A path from the goal to the node representing the current position of the UAV can then be searched using algorithms like A* [110], D* Lite [143] or Anytime Dynamic A* (AD*, see: Section 2.3.1). Similar to RRT-based algorithms, graph search algorithms require recalculation of the path when new obstacles are detected, which can be too computationally complex for real-time obstacle avoidance.

Hrabar [119] applies the D* Lite algorithm obstacle-avoiding navigation with UAVs. Their approach uses a stereoscopic camera to detect obstacles in front of the UAV. They demonstrate their concept by performing several flights through an environment containing obstacles. While the authors state that their stereoscopic obstacle detection is too unreliable for real-world use and needs further work, the navigation using the D* lite algorithm works as expected. However, in testing performed as part of this dissertation, the D* Lite algorithm's replanning performance was found inadequate for highly dynamic manufacturing environments without detailed prior knowledge about the environment. Likhachev et al. [158] try to mitigate this problem with Anytime Dynamic A* by only replanning nodes of the graph affected by the obstacle. As this approach appears promising, an Anytime Dynamic A*-based navigation algorithm was implemented in this work. Its performance is compared to the Artificial Potential Field method in Section 5.2.5.

Chen et al. [42] use a two-step approach to improve reaction times to dynamic obstacles when using the A* algorithm. Initially, a global path is planned using a map of the environment and the A* algorithm. The UAV navigates along the path until it encounters an unknown obstacle. The trajectory is then altered by generating multiple cubic splines allowing to navigate around the obstacle. Using a set of cost functions, the ideal spline is chosen and then used as trajectory until the UAV can return to the original

trajectory. The calculation and selection of these alternative trajectories takes about 1.5 s on an Intel®Core™i5-3470 CPU, limiting the UAV speed with which the algorithm can successfully prevent collisions. Also, the algorithm was only implemented for 2D-environments further limiting its function. The simple local avoidance strategy also might not work well in environments, where most of the obstacles are not known in advance like the inspection of large structures, where the model of the structure is known but no information about the surrounding manufacturing facility is available.

Force Field Algorithms

Force Field-based methods use virtual forces or potentials to steer the drone towards the goal. In these approaches, the goal is positively *charged*, while the UAV and all obstacles receive a virtual, negative charge. This results in the drone being attracted by the goal and repelled from all obstacles. Budiyanto et al. [30] use the potential field algorithm, a force-field-based approach. Multiple experiments performed in simulation demonstrated the ability to navigate a UAV through various environments. The situations ranged from simple, static environments to ones that were more complex and using moving obstacles. The study was limited to simulations using the Parrot AR Drone 2.0 quadcopter model and did not involve real-world testing. Also, the local minimum problem was not addressed in the study. Chang et al. [37] use this approach to control multiple UAVs in formation flight.

Choi et al. [45] propose a solution for the local minimum problem using an *enhanced curl-free vector field* approach. It works by generating force fields that *rotate* around an obstacle, thus reducing the chance of local minima occurring. However, it is not clear, how this method would prevent a drone from getting stuck in a local minimum caused by a U-shaped obstacle, since the potential field is only curl-free for each obstacle. Summing up the individual potential fields for obstacles and goal may still result in a local minimum. Chen et al. [41] try to work around the local minimum problem by strategically generating a series of guide points and using them to navigate around local minima. Keyu et al. [138] propose a new potential field function using fuzzy logic and a relative velocity repulsion potential field for dynamic obstacles.

Zacharia et al. [299] present a control and navigation approach for multiple UAVs for covering the surface of an object of interest. They use an earth-fixed coordinate frame for navigation. The authors create a linearized model of the UAV dynamics and calculate the acceleration setpoint of the UAV as a control input. The control input is derived by minimizing a cost function while taking constraints like UAV dynamics, remaining in the inspection region and avoiding collisions with other UAVs. The control mechanism of each UAV relies on local sensor measurements and information shared by other UAVs. Inspection points are derived by outward projection of points on the target surface. To avoid collisions between the inspected object and the UAV, a repulsive force is calculated based on all surface points of the object. To counteract this force and allow the UAVs to perform inspections close to the object, an attractive force function with a Gaussian distribution centered around the inspection point is used to pull UAVs towards the target points. A PD-controller is used to generate a control input moving the object towards

the center of a Voronoi region [288, 289] occupied by the UAV. This results in the UAVs flying individual trajectories along the object covering its surface.

The Heat Equation Driven Area Coverage (HEDAC) algorithm developed by Ivić et al. [124, 125] is intended for coverage inspections of structures using multiple UAVs (see Section 2.3.2 for an in-depth explanation). Repelling potentials by the inspected objects and other UAVs ensure collision-free trajectories while the gradient of a pre-calculated and continuously updated target density field (representing the areas that need to be visited by the UAVs) determines the direction of the inspection trajectory. The sum of these vectors is used as control input for each UAV. While the algorithm works great for calculating effective inspection-trajectories before a flight, it is to computationally complex to be used for dynamic obstacle avoidance.

Kitamura et al. [142] use a potential field-based approach for navigating a robot to a target point. An octree data structure is used to represent obstacles. The approach focuses on simplicity and is designed to be efficient with a focus on parallel computation.

Woods and La [296] perform dynamic target tracking with a drone to follow a human target. The proposed system uses a potential field controller working with relative positions and velocities to track dynamic targets and avoid obstacles in real-time. A similar approach is used by Jayaweera et al. [127] to follow moving ground targets using a UAV. They present the Dynamic Artificial Potential Field (D-APF) planning system specially developed for relative navigation and following moving targets using UAVs. The approach outperforms several Artificial Potential Field algorithms in their evaluation testing for displacement for the ground moving target and when navigating through obstacles that are spaced closely together.

In other target tracking applications, simple local rules, position controllers or potential field methods are often used to follow [152, 212], navigate to [49, 217] or fly trajectories relative to [40, 290] the target. Force-Field-based navigation strategies and especially algorithms using the potential field method are still widely used and researched [181], [268], [156], [146], [197], [64], [296], underlining their relevance for obstacle avoiding navigation.

3.5 Using UAVs for Inspection

Companies that offer UAV-based inspection services most of the time use either off the shelf, professional camera UAVs [4, 174] or develop specialized hardware themselves [63, 168, 239] to carry the sensors required for the inspection. Due to this fact, these UAVs are equipped with state-of-the-art obstacle sensors, and they are able to carry custom sensors for specific inspection types. However, this also makes the UAVs large and expensive, preventing their use in narrow spaces like inside a tubular fuselage section. A notable exception to this is the relatively recent Elios 3 UAV [239, 240] by Flyability. It features a small and lightweight LiDAR sensor that can be used for navigation and mapping using SLAM [19], as well as obstacle avoidance while still being small enough to navigate inside enclosed spaces. This is further facilitated by ducted propellers and a lightweight cage around the UAV that minimizes damage to the surroundings in case of a collision. However, the manufacturer lists a flight time of 9.1 minutes when using the LiDAR Payload, which might not be sufficient to perform longer inspections. Also, the high price of an individual UAV may require complex workarounds like mobile ground platforms that transport the UAV over long distances in a facility to the place where it is required [154, 210]. Using smaller, inexpensive consumer UAVs can navigate around these problems. Their low price allows for the purchase of multiple units eliminating the need to move a UAV around a facility. The smaller size also enables them to navigate into tighter spaces. However, they come with different limitations like worse obstacle sensor quality, fewer options for customization and remote control via API is only available for some UAVs. Also, the type of sensor available for inspections is limited to a gimbal stabilized optical or in rare cases thermal cameras [251] limiting the types of inspection that can be performed. While most concepts presented in this work can also be applied to more advanced UAVs, the following paragraphs will focus on what inspection tasks can be performed by consumer UAVs in particular.

To demonstrate, how a simple presence detection that determines, whether a particular part was installed or not could be implemented, the following paragraph describes a concept for a simple presence detection system using only a monoscopic camera image and the CAD model as a reference. In a systematic literature review by Yasuda et al. [298], regarding UAV-based inspection systems the authors conclude that the numerous inspection procedures require an inspection system to be built in a modularized form. Also, the availability of off-the-shelf inspection solutions for handheld tablets or ground based robots equipped with cameras [70, 207, 245] makes interfacing them with a UAV-based system relevant developing own solutions less relevant. Therefore, a concept for interfacing existing, manual inspection solutions with the proposed UAV-based system is also presented. Lastly, a detailed overview of the current state of the art of these existing inspection solution is given to demonstrate the possibilities of the proposed interface.

3.5.1 Visual Presence Detection

The field of automatic visual inspection of assemblies during production is very diverse due to the plethora of different types of inspections that need to be performed. While simple presence detection of a part may suffice in some cases, often, more complex tasks such as checking for the correct orientation within high precision or detecting if a part has been bent on installation are needed. Also, inspection scenarios can be highly specific like detecting if a Washer has been installed under a screw or if a cable has been routed adequately through a bracket. Some require incredible precision, with an example being the detection of rivet protrusion on a fuselage part to within a fraction of a mm. As demonstrated in the preceding sections, a variety of different inspection tools already exist on the market to cover most of these applications. This is why developing a universal inspection solution for all types of problems would be beyond the scope of this work. Most of the existing inspection tools rely on manual camera placement or perform the inspection using ground-based robots. Therefore, the focus was on developing concepts for positioning the UAV and its camera at the correct position to perform an inspection. Streaming the camera feed to a powerful inspection tool previously capable of performing manual inspections seemed a more reasonable solution. Therefore, the primary focus of the inspection part of this work was on providing an interface for existing manual inspection tools to interact with the system and decide whether the part the UAV is currently looking at is deemed correctly installed.

Nevertheless, a visual presence detection system that can perform an entire inspection flight without relying on external inspection software is proposed in the following paragraphs. This approach uses the availability of the assembly's CAD data under inspection. Upon arriving at a Viewpoint, a high-definition image of the Point of Interest is captured (see: Figure 3.16) and transmitted to the main computer running the inspection together with the UAV's estimated camera pose relative to the assembly. The pose is then used to render an image of the part that is currently inspected from the same perspective and compare it to the image captured by the UAV (see: Figure 3.20).

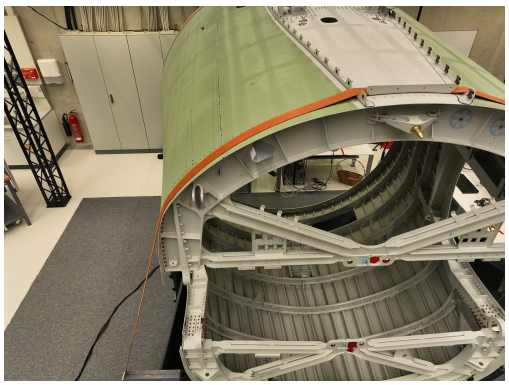


Figure 3.16. Image captured by the UAV during inspection.

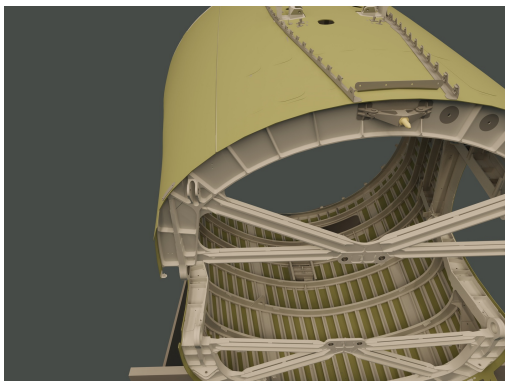


Figure 3.17. Render of the reference model from the same position.

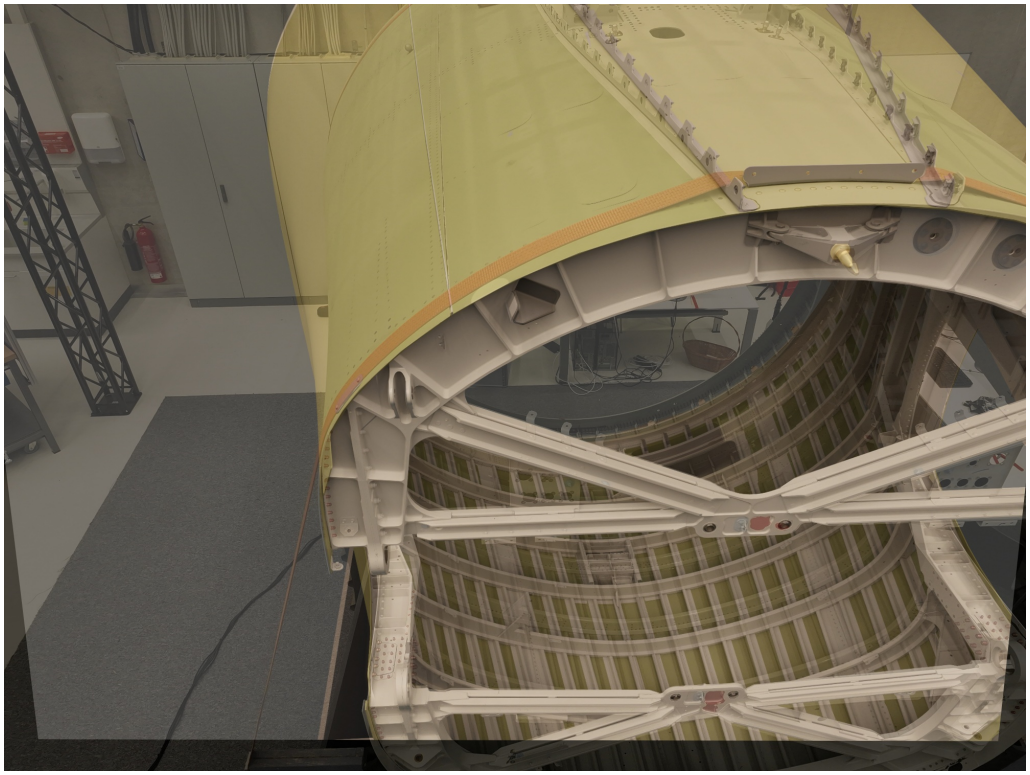


Figure 3.18. Warped render overlaid on original image.

Figure 3.19. Preprocessed camera image.

Figure 3.20. Cropped Render of the Point of Interest from the same position.

The presence detection algorithm works by using the rendered part as a template for a template matching algorithm, which tries to match the object in the render to the camera image.

The proposed process consists of the following steps:

- Rendering an image of the assembly and the Point of Interest from the perspective of the inspection camera.
- Warping the perspective of the reference renders, so they closely match the captured image.
- Using a template matching algorithm to search for the image of the rendered Point of Interest in the warped image.

Reference renders

The warping and template matching step requires a rendered image of the model of the entire assembly and the inspected Point of Interest, respectively. These are created from the perspective of the inspection image. The perspective is defined by the camera location and orientation, which is captured in the reference frame of the assembly during inspection. Examples of these renders are shown in Figures 3.17 and 3.20.

When creating the renders of the CAD file, it is necessary to match the camera and lighting parameters for the scene as closely as possible. For example, when the actual inspection is performed inside an industrial production facility with rows of fluorescent tubes mounted high up in the ceiling, the scene used to render the reference image should have a similar lighting setup. Also, conditions that produce harsh shadows should be prevented during both the real-world inspection and the render scene. Installing a headlight on the UAV and raising the ambient light level in the render scene can also reduce the amount of unwanted shadows. The focal length and aperture of the render camera must match the camera on the UAV. Closing the aperture down as far as possible without creating excessive noise in the image is recommended to ensure that all relevant sections of the Point of Interest are in focus.

Data Preprocessing

Some preprocessing is required for both images to maximize the performance of the template matching algorithm. The image captured by the UAV contains distortion induced by the camera lens. It is described by a set of distortion parameters that also allow for calculating a distortion-free image, which is needed for better comparison. These parameters are also constant and can be determined with a simple calibration process. Because even minor differences in camera position or orientation between render and captured image can produce problems with the template recognition, the renders are compared to the image and warped, so they match the image as closely as possible.

The software *SuperGlue* [172] by MagicLeap is used for this task. It uses a Graph Neural Network in conjunction with an Optimal Matching layer to extract a set of sparse image features from two images and match them. This process is used with the captured image

and the reference of the entire assembly to compute a transformation between the two images. To ensure ideal matching of the center area of the image, where the Point of Interest is most likely to be located, the matching process only uses features from the center of the image. The resulting transformation can then be applied to the render of the assembly and the Point of Interest, so they closely match the image. Figure 3.18 shows an overlay of the warped render over the original image and demonstrates how this process aligns the center region of both images.

In preparation for the template matching step, the warped render of the inspected Point of Interest is converted into a grayscale image and cropped, so unnecessary space is removed, and the part is located in the center of the image (see: Figure 3.20). The camera image is also cropped, so only the center section remains, eliminating unnecessary search space for the template matching. In this particular example with an extremely wide camera lens, a border with the size of 20 % of the image width and height is removed. The image is also converted to grayscale. Figure 3.19 shows an example of the preprocessed image.

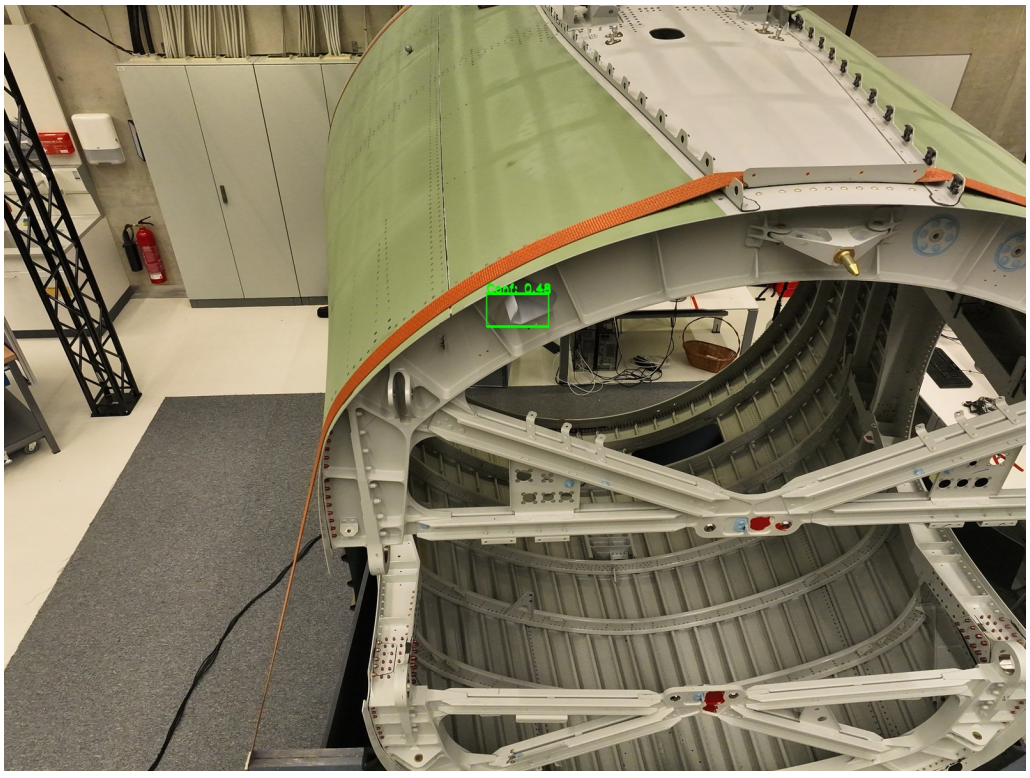


Figure 3.21. Annotated image highlighting the detected Point of Interest.

Template Matching
The render of the part can then be used as a template for a template-matching algorithm, which tries to locate the specified template in the preprocessed camera image. The detected results often contain overlapping matches that need to be removed to avoid duplicate detections. To perform this step, the intersection over union (IoU, see: Szeliski [271]), the ratio of the intersected area of both matches over the area of their union, is used as a metric. For each match, the IoU for all other matches is calculated. If the IoU exceeds a prespecified threshold, the match with the smaller confidence value is removed. This process eliminates clusters of matches around the Point of Interest and only keeps the best match. The matches are also filtered for a minimum confidence threshold and matches below this threshold are discarded. The outlines of the remaining matches are drawn to the image along with their confidence value for later analysis by an inspector (see: Figure 3.21). If at least one match above the certainty threshold remains, the inspected part is marked as detected. If no matches are found, the part is either not present, obstructed, or mounted in the wrong orientation. Depending on the specific inspection scenario, the UAV might use alternative Viewpoints to repeat the process from a different perspective before proceeding to the next Point of Interest.

3.5.2 Interface for Connecting External Inspection Solutions

Due to the aforementioned reasons, implementing an inspection method that covers every possible type of visual inspection is impossible. The availability of sophisticated existing inspection tools for manual inspection or automated inspection using ground robots [100], [269], [112], [300] makes adapting them to work with a UAV-based system highly attractive. In order to facilitate this task, an interface for exactly this purpose is provided by the system proposed in this work. The structure of this interface can be viewed in Figure 3.22. To integrate an existing tool into the system, a small wrapper

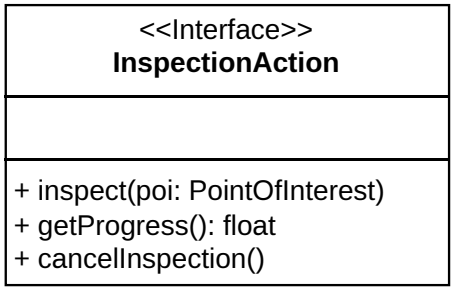


Figure 3.22. Interface for integrating external inspection tools.

program is required to implement this interface and forward necessary information about the Point of Interest to the inspection tool. Since the required information varies from a single camera image, camera positions, a live video stream and other data depending on the tool that is to be adapted, the interface only specifies the information about the inspected Point of Interest as a parameter. Additional required information can be obtained as needed from the system using the message based communication

system and provided services that are presented in Chapter 4. The inspection interface also provides a means to start the inspection using the `inspect` function. The inspection progress can be obtained via the `getProgress` method. It is also required to provide a means to cancel an inspection using `cancelInspection` in case for example a timeout occurs or the inspection is interrupted either manually or by a depleted drone battery. Section 4.2.5 describes the architecture for integrating an inspection system using this interface in more detail.

3.5.3 Related Work and State of the Art

Both industry and scientific literature present a plethora of automated inspection techniques using sensors that are available on cheap consumer UAVs (i.e. visible light and optical cameras, thermal cameras, stereoscopic cameras...). The following sections will therefore give an overview of the current state of the art of inspection types that can be performed using these kinds of sensors that therefore could be adapted to work with the proposed system. First, an overview of the available inspection techniques is given. After that, selected works that cover visual and thermographic inspection using mostly a coverage approach to inspect the entire surface of a part are covered in more detail. Lastly, visual approaches for checking conformity of a part during inspection are presented.

Moreno-Jacobo et al. [194] describe Thermography, Photogrammetry and Laser Scanning as the three main applications in drone-based infrastructure inspection. In Thermography, the surface temperature of the inspected objects is captured and analyzed. Photogrammetry specializes on reconstructing a textured 3D model of an object based on a series of images. Laser Scanning directly measures the 3D geometry of an object in form of a point cloud but lacks the texture information provided by a photogrammetric scan. However, the authors focus their work on infrastructure inspection. Malandrakis et al. [177] list visual inspections, borescope, liquid penetrant, eddy current, ultrasonic, acoustic emission, magnetic particle and radiography as possible non-destructive testing methods in the production of aircraft parts. Bardis et al. [21] focus on in-service inspection of aircraft and list vision, x-ray, ultrasonic, magnetic particle, penetrant, eddy current acoustic emission, infrared thermography and 3D scanning using LiDAR sensors as possible inspection techniques in this field. Yasuda et al. [298] also emphasize the necessity for a modular system that is able to integrate with other systems. Some inspection solutions presented are highly specialized for a specific application, which makes it impossible to develop a universal inspection solution. Still, the majority of scientific literature focuses on vision-based and thermographic inspection techniques. Due to their popularity in the field and their ability to be performed using a standard UAV, the following sections will focus on these types of inspections.

Vision Based Inspections

Since the original manual inspection of aircraft is performed mostly visually by an inspector, these types of inspections are most common for in-service inspections of aircraft and during their production process. Over 80 % of non-destructive inspections on large-scale aircraft is done vision-based [208]. In the aviation industry, Airbus developed several systems for autonomous image capture and photogrammetric capture of aircraft while in service [7, 8]. The company Donecle [50] offers a similar system for in-service aircraft and large components in manufacturing but also provides dent-detection using a specialized dentCHECK sensor. Zeiss developed a specialized scanning sensor [136, 300] that allows scanning of an entire assembly and its precise reconstruction as a 3D model. Additionally, the associated software is able to compare this 3D model to the geometry of the assembly's CAD file and detect discrepancies between the two. Autaza [16, 55, 207] specialize on vision based inspections and their execution either manually or using ground based robots or UAVs. They heavily use artificial intelligence for vision-based inspections. One of their inspection algorithms [55] works by projecting a line pattern on the inspected surface and detecting dents using distortions in the captured projection. This allows to detect dents and other surface irregularities.

In scientific literature, Papa and Ponte [208] present a system to automatically detect hail and lightning damage for in-service inspection of aircraft. The RC EYE One Xtreme drone used in their tests is equipped with a 6-megapixel Raspberry Pi camera module V2. The system processes the captured images on the onboard Raspberry Pi computer and sends the inspection results to a ground control station for review by an inspector. The authors verified their concept with a prototype that successfully detected dents and lightning strikes on sample metal shields in a laboratory environment. Malandrakis et al. [176, 177] and Tzitzilonis et al. [277] present a system to automate the process of testing aluminum fuselage parts in production using dye penetrant testing. This check is done prior to anodising the aluminum part to detect corrosion and cracks in the part. The process works by applying a penetrant liquid on the entire part which is pulled into cracks and defects by capillary forces [277]. After the penetrant is washed off the part, developing powder is applied that draws liquid left in the defects to the surface. Any liquid residue left on the part after this process can be highlighted under UV-light and is indicative of a defect. The authors automate the capture of these residues using a commercially available Bebop 2 Power drone with a UV-flashlight mounted on top. The drone features a digitally stabilized wide-angle camera and captures a set of overlapping images of the surface of an aircraft wing by generating a set of waypoints in a line pattern on both sides of the wing. The defects in the images are automatically detected using two different algorithms. The first algorithm scans for fluorescent areas in every frame of the captured video. If a fluorescent section is detected, a Structural Similarity Index Measure (SSIM) [292] is used to compare the image to a set of baseline images. If the similarity score is high enough (>85 %), the frame is classified as containing a defect. With scores of a certainty between 64 % and 84 %, a histogram comparison on the same set of baseline images decides whether the image contains a defect. The second algorithm uses a Random Forest classifier algorithm that was trained and tested on the texture and color information of the captured reference images. The authors state that

the first algorithm was able correctly identify all defects in their test dataset and the second algorithm achieved an accuracy of 97 % on the validation data. Plastropoulos et al. [215] and Avdelidis et al. [17] cooperated with the Airline TUI to detect defects on in-service aircraft using deep learning. Avdelidis et al. [17] retrained multiple convolution neural networks (CNN). The best performing networks were selected, further optimized and a final optimal model was chosen. During evaluation, the networks are able to classify damages like dents, paint damage and scratches. The Authors use a dataset of 1059 images and achieves an accuracy 81.82 % for the classifier deciding on whether an image contains a defect and 100 % accuracy on the classification of the defect itself. Plastropoulos et al. [215] worked with a dataset of 6816 images and were able to achieve a precision of 71 % with an Area Under the Curve (AUC) of 0.69 for dent detection using the EfficientDet D1 network.

Miranda et al. [190] present an approach to inspect screws on the exterior of aircraft. The authors use computer vision and a Convolutional Neural Network to detect and extract aircraft exterior screws from images captured by a UAV. The Convolutional Neural Network is used to identify zones of interest (ZOI) and extract screws from the images. Classic matching algorithms were used to compare the detected screws to a prior model, improving screw recognition accuracy and identifying missing screws. Hrúz et al. [120] use a DJI Mavic 2 equipped with a visible light and thermal camera to detect defects on in-service aircraft. They use an approach of segmentation, feature extraction and classification to detect dents and similar defects in the captured images of both cameras.

Thermographic Inspection

Thermographic inspection techniques have become increasingly relevant in the production and in-service inspection of modern-day aircraft [56]. Their ability to detect subsurface defects in composite materials in combination with the increased use of these materials in modern passenger airplanes like the Airbus A380 and the Boeing 787 Dreamliner increased the relevance of thermographic inspection [56]. In production, air inclusions, debonding and other manufacturing errors can result in faulty carbon fiber parts. The defects are often undetectable through visual inspections as the problematic areas are located inside the composite structure. While in service, composite parts on aircraft can be damaged through impacts of foreign objects which can cause various defects like matrix cracking, fiber/matrix debonding, surface microbuckling, delamination, and fiber breakage [231]. These defects are often invisible from the surface, but can be detected through thermographic inspection. Usually the material is thermally excited through optical or electromagnetic radiation, ultrasonic waves, or by applying an electric current to conductive materials. The resulting propagation of heat through the material is then analyzed with the help of a thermal camera to detect subsurface defects in the material, which show up as areas with uneven thermal propagation. The company Autaza [16] holds a patent [207] for projecting line pattern in the infrared sectrum on a material and capturing the reflection using a thermal camera. The distortions in the pattern can be used to detect dents and other defects.

Checks for Correct Assembly

While automated checks for correct assembly of components is still rare in aircraft manufacturing, other similar sectors like the automotive industry have adopted the technology. The software FARO Visual Inspect [73] offers a system that allows Augmented Reality overlays of CAD models over real-world products to help an inspection with the visualization of errors. However, no automatic detection of errors is performed. Other systems like the software by Neurocheck [99] and VMT [101] are able to perform assembly inspections by training computer vision systems with a set of reference images for the specific inspection task [245]. Visometry offers the software visionlib [245, 286] that includes systems for detecting correct assembly of components. The software offers presence detection [287] which works by tracking both the assembly and the inspected part simultaneously and comparing their relative position to the prespecified reference [245].

The preceding paragraphs demonstrate that a large variety of distinct inspection methods are required during the manufacture of large assemblies and structures. Most of these can be performed using either visual light or thermal cameras available on consumer UAVs. However, UAV-based inspection solutions implemented in industry and scientific literature focus on implementing a system that performs one specific type of inspection throughout the presented literature. The approaches to path planning and position estimation are similar and are mostly based on either a coverage or Point of Interest-based inspection path planning. A modular and expandable system for inspection path planning as described in this work allows for a focus on developing the inspection-specific sections of the algorithms and avoids reimplementation of common path-planning, positioning and navigation algorithms. Also, this work allows for the integration of conventional inspection techniques and tools for their use with UAVs.

3.6 Photogrammetric Documentation of Assemblies

As discussed in Section 2.3.2, some applications require covering the entire surface of the assembly. An example would be the photogrammetric documentation of an aircraft fuselage part before it leaves a manufacturing plant to be assembled further at a different facility. To be able to attribute potential damages that happen to the assembly to either the local plant, the transport, or the plant next in line, photogrammetric documentation of the assembly before entering or leaving a plant can be helpful. In photogrammetry, multiple 2D images of an object are combined to form a textured 3D model of the documented object. This model, combined with the raw images it was created from, is ideal for documenting the state of an entire assembly at a certain time. The following paragraphs will first discuss using the HEDAC algorithm to generate trajectories for full surface coverage and the required modifications to perform these with relative positioning. Furthermore, an exemplary photogrammetry pipeline that allows for creating textured 3D models from the captured images is presented.

3.6.1 Full Surface Coverage

The following paragraphs cover two concepts developed for the photogrammetry capture of an assembly using the HEDAC algorithm. Both fundamentally use the same trajectory planning algorithm and the software AliceVision Meshroom [104] to generate photogrammetric models from the captured images. The first proposed concept defines an open loop planner that executes the steps of trajectory planning, image capture, and photogrammetric reconstruction strictly sequentially and passes the finished output of one step as input to the next. The closed loop planner uses feedback from the model quality to influence the trajectory planning in a way that encourages the UAV to navigate to areas of the image with the worst coverage so far.

Open Loop Coverage

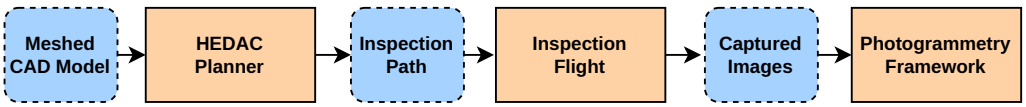


Figure 3.23. Architecture of the proposed open loop photogrammetry system.

The open loop coverage strategy consists of three main parts, which are depicted in Figure 3.23. The HEDAC Algorithm is responsible for planning a trajectory that evenly covers the entire surface of the assembly. The general idea of offline trajectory planning is to calculate linear target velocities using the concept proposed by Ivić et al. [124, 125]. This is done by continuously combining the initial- and coverage potential fields created by the inspection UAVs. While the algorithm can generally calculate trajectories for multiple UAVs working in parallel, the following concept will focus on an inspection flight with only one UAV. The CAD model of the assembly, which serves as the foundation for the coverage algorithm, is converted to a mesh before being passed

to the system. Since the execution time of the algorithm increases with the number of polygons in the mesh, it can be beneficial to reduce the number of polygons and, therefore, the degree of accuracy. A less detailed mesh can significantly improve the algorithm performance and still yield good reconstruction results [178]. The required environment mesh representing all space the UAV can navigate is created by binary subtraction of the assembly mesh from a volume representing the space available for the UAV. This can be a box the size of the room or a more detailed model if information about the structures and obstacles around the assembly is known. Creating the environment mesh aims to represent the free space as precisely as possible.

Camera Orientation
Updating the surface coverage map is done using the vertical and horizontal field of view of the camera, through which the parts of the mesh that are visible to the camera in each step can be determined. This rectangular simulation of the cameras' field of view extends the original round approximation proposed by Ivić et al. It provides a more accurate approximation of what parts of the model are visible in each time step. Ray tracing is also used from the sensor origin to each potentially visible point of the mesh to detect whether it is obstructed by obstacles. Using a UAV with a camera that can be oriented independently of the UAV orientation allows for more flexibility in the camera angles. Therefore, the strategy for orienting the camera was changed from originally just looking at the point closest to the mesh. Instead, the direct neighbors of these points within a prespecified radius around the original are also considered potential targets for the camera orientation. Out of these points, the one with the least amount of coverage is selected as a target point for the camera to look at to encourage capturing images of less well covered areas of the surface.

Viewpoint and Point of Interest generation
In the open loop implementation, the output of the HEDAC algorithm is converted into an inspection path consisting of an ordered list of Viewpoints and Points of Interest as with the selective inspection process described in the preceding sections. To do this, the existing algorithm is extended by storing the point used for camera orientation as well as the UAV position and camera orientation for each time step. These two points are stored in sequence as Point of Interest and Viewpoint. They, in turn, form a path compatible with the navigation system for selective inspection described above. Instead of inspecting each Viewpoint, the UAV captures an image with the camera oriented towards the Point of Interest. Depending on the value of the parameterization, the UAV's distance to the nearest obstacle might fall below the specified `safety_distance`. In this case, the emergency collision avoidance system proposed in Section 3.4.3 would stop the UAV before it collides with the assembly, but manual intervention would be necessary to continue with the inspection flight. A potential solution to this problem would be to increase the HEDAC safety distance or decrease the threshold for the emergency collision avoidance until the slight oversteps of the safety distance do not trigger the collision avoidance anymore. Since the first would decrease the minimum possible inspection distance and the latter could potentially compromise the effectiveness of

the collision avoidance system, a third option is to discard viewpoints that are located too close to an obstacle. While this strategy potentially reduces the number of images taken for the photogrammetric reconstruction, it can be helpful in cases where only a few Viewpoints are located too close to the assembly and cause the emergency collision avoidance to trigger. When removing these Viewpoints, the flight can be performed without interruptions at the cost of losing only a few images that would otherwise have been lost as well.

Photogrammetry System

The images captured at each Viewpoint are stored on the UAV and transferred to the computer executing the photogrammetry software after the completion of the inspection flight. Before the images can be used for photogrammetric reconstruction, they are preprocessed to maximize the dataset's quality. In addition to removing duplicates, images that contain motion blur are automatically detected and removed. In cases where significantly more images were captured than required for the photogrammetric process, a threshold for resemblance between two images can be defined, and images that lie over this threshold can be removed to reduce processing time.

Additionally, image segmentation algorithms (see: Minaee et al. [188]) can separate the inspected assembly from the background and discard the latter. This can improve the quality of the generated model, as objects in the background are irrelevant to the documentation process and are underrepresented in the dataset, resulting in a suboptimal reconstruction of these objects that can sometimes influence the quality of the reconstructed assembly. After preprocessing, the captured images are processed using the AliceVision Meshroom software, which outputs a textured 3D mesh model. The structure of the pipeline used in Meshroom is discussed in Section 4.2.6.

Closed Loop Coverage

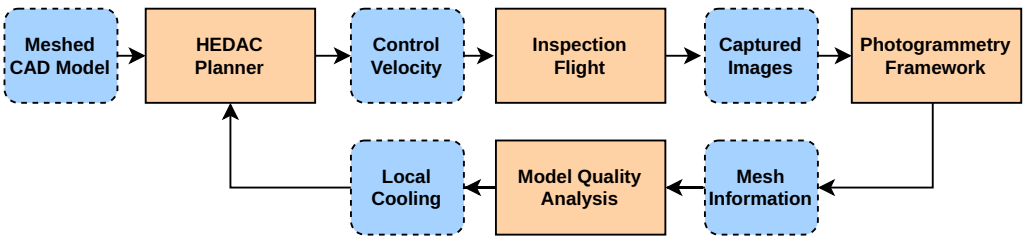


Figure 3.24. Architecture of the proposed closed loop photogrammetry system.

For the closed loop system, information about the current state of the photogrammetric model is fed back into the HEDAC planner to detect areas where coverage is insufficient for a good reconstruction of the model. The general architecture of the proposed system is depicted in Figure 3.24. The major difference to the open loop architecture is the feedback of the quality of the generated model through the model quality analysis

subsystem. Additionally, all steps performed strictly in sequence in the open loop architecture now run in parallel and continuously forward data to the next subsystem.

Trajectory Planning
The trajectory planning system is based on the algorithm proposed in the open loop concept. However, instead of generating a complete path with all Points of Interest and Viewpoints in advance, the algorithm continuously outputs the velocity control vector along with camera orientations to the UAV's navigation system. The Viewpoint mission controller is completely bypassed, and the control velocities are fed directly into the obstacle-avoiding navigation system. Because this system can avoid not only previously known but also dynamic obstacles, it is also possible to remove the collision avoidance system from the HEDAC algorithm. This is needed to increase the algorithms' performance in this near real-time use case. Due to the same reason, the calculation of the camera orientation was greatly simplified. Instead of looking at a point of the mesh close to the camera, the camera is continuously pointed at the volumetric center of the mesh. As an additional step to increase performance, the surface coverage is not calculated at every step as in the original algorithm. Instead, its calculation is outsourced to a separate thread and performed at a reduced rate. This is possible without sacrificing the quality of the generated trajectory as the surface coverage is only required to judge the percentage of the surface that was already covered and, therefore, the overall progress of the inspection. Additionally, a control input is added to integrate feedback of the current state of reconstruction of the model into the trajectory planning. This is done by artificially reducing the amount of perceived coverage of areas that need additional information for better reconstruction. The HEDAC algorithm offers this possibility through a local cooling mechanism. Since the UAV always tries to achieve an even coverage of the entire model, cooling forces it to revisit cooled areas, producing more images for a better photogrammetric reconstruction of the affected area.

Feedback Mechanism
In order to provide the HEDAC algorithm with feedback about the areas of the assembly that require more coverage, a method is needed to derive these areas of interest from the output of the photogrammetry algorithm. The method proposed in this work compares the generated mesh of the photogrammetry tool to the mesh created from the original CAD model. The difference between the current mesh and the reference can be determined through simple Boolean subtraction, and the resulting areas serve as input for the local cooling operation.

Photogrammetry System
Compared to the open loop system, the photogrammetry module was modified to continuously operate and use images captured by the UAV to incrementally create new models based on all available images. As soon as the generation of a model is complete, it is passed on to the feedback mechanism.

Summary. The implementation of the concepts proposed in Chapter 3 are covered in this chapter. Additionally, the architecture for a system that performs a UAV-based inspection on arbitrary assemblies is discussed.

4

Architecture and Implementation

This chapter covers the system implemented to evaluate the concepts proposed in Chapter 3. It was developed as a modular platform, allowing the exchange of different modules and various configurations for different use cases. This was mainly achieved through using the Robot Operating System (ROS) middleware [87], [228] and defining interfaces that allow components to be exchanged as needed. The basic architecture of the system developed for CS1 can be seen in Figure 4.1. The system can be roughly separated into a planning- or offline-phase and an execution- or online-phase. The planning phase generates an inspection path from the original CAD data. To do this, the original CAD file is first augmented with information about the POIs. Additional information, like the inspection type, can also be specified. This annotated CAD file is then used to generate an unordered list of Viewareas. After Viewarea optimization is performed as described in Section 3.2.3, Key Viewpoints are chosen for each VA, and an inspection path is generated. The resulting path functions as input for the

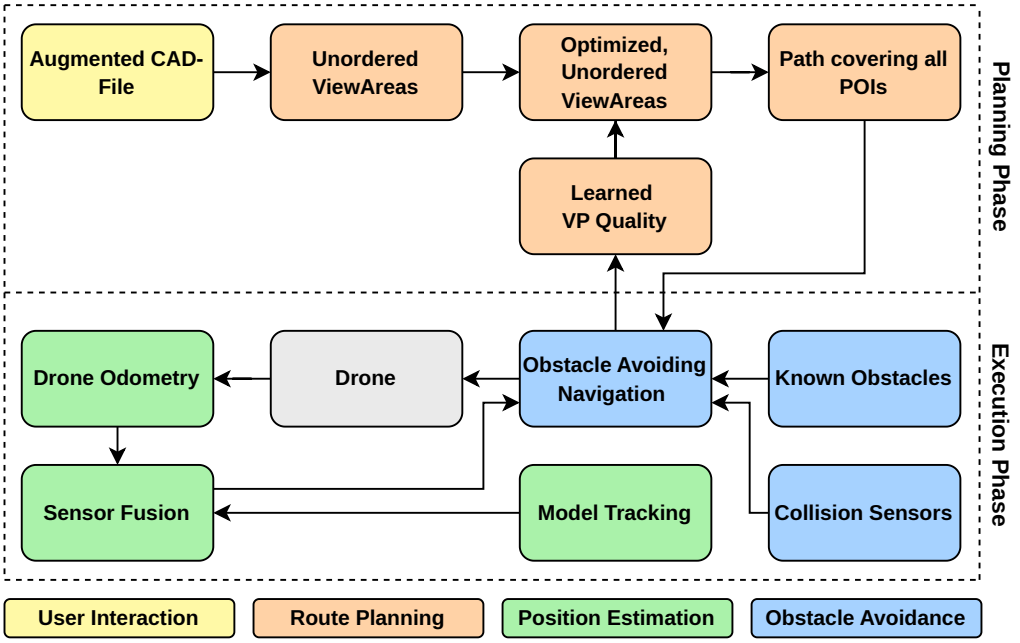


Figure 4.1. General architecture of the inspection system.

execution phase. It is provided to the obstacle-avoiding navigation system that also has access to obstacle sensor data from the UAV and information about known obstacles that were provided before takeoff. The navigation system provides a control input for the UAV, which moves accordingly. The odometry provided by the UAV and the position measurement by the model tracking system is fused according to Section 3.3 and provided as feedback for the navigation system. Finally, after every inspection, information about the quality of the Viewpoints that were navigated to are provided back to the path planning system to generate an optimized path for the next inspection. After the inspector can view the inspection, pictures, videos, and inspection results in a specially designed GUI, and the acquired data can be added to the PLM system.

4.1 Offline Path Planning

The following sections cover the architecture and implementation of the offline path planning subsystem. This includes all steps that are performed in preparation for the inspection flight. As previously described its comprised of the creation of an augmented CAD file by an engineer that specifies the location of the Points of Interest and the types of inspection that need to be performed for the POI. This is the only part of the process that needs manual input. The optimization of the Viewareas and the calculation of an inspection path are fully automated.

4.1.1 Specification of Components to be Inspected

To give engineers an intuitive way to augment the CAD data with information about what parts of the assembly need to be inspected, the QuAD Pre-Inspect Tool was developed, which allows engineers to perform these actions inside a graphical user interface. This user interface (see: Figure 4.2) displays a 3D view of the CAD data and allows the user to specify any part of the assembly as a POI. It also allows for the specification of the inspection type and the orientation and distance range from which the POI has to be inspected.

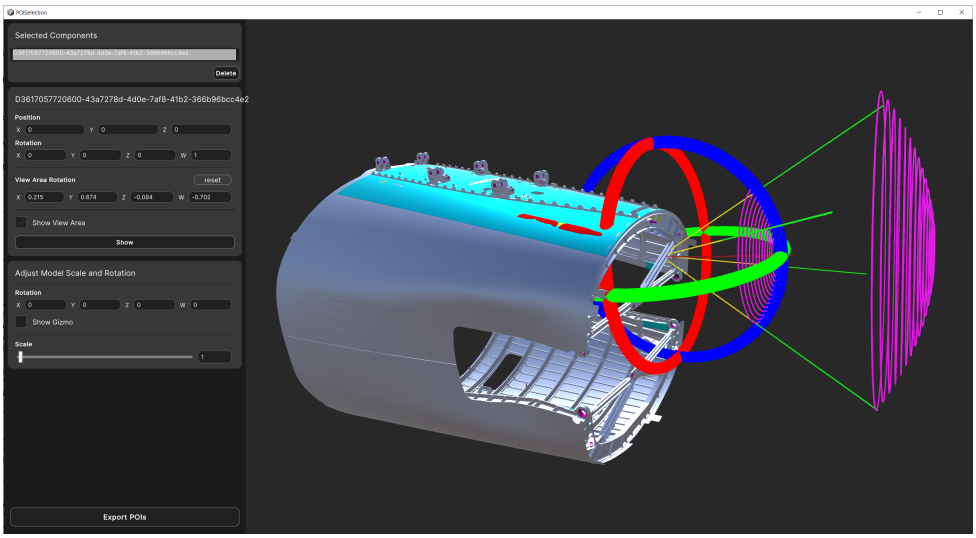


Figure 4.2. Screenshot of the tool for augmenting CAD data with inspection information.

The tool is written in Unity and can open a variety of CAD file formats. The POIs are output in JSON [122] format. The structure of a single POI can be seen in Listing 1.

Each POI is described by a unique identifier (`ID`), its `position` and `orientation` (the normal of the POI) as well of the `width` and `length` of the part when looked at from the specified orientation. The positions where the POIs can be inspected from are further limited by the `min_dist`, `max_dist`, and `max_deviation_angle` parameters. The type of inspections can be specified using the `inspection_type` parameter, and an optional

```
{
    "ID": "POI_TL",
    "position": [-0.16, 0.34, 2.29],
    "orientation": [0.0, 0.0, 0.0],
    "length": 0.4,
    "width": 0.4,
    "min_dist": 1.0,
    "max_dist": 1.8,
    "max_deviation_angle": 50,
    "inspection_type": ["image_capture", "presence_detection"],
    "sem_ann": "bracket, green",
    "frame_id": "s19"
}
```

Listing 1. JSON structure of a single POI.

semantic annotation can be added using the `sem_ann` parameter. The `frame_id` defines the reference frame for the position and orientation values.

4.1.2 Creation of Viewareas

All remaining steps of the path planning are implemented in Python. The creating and optimizing the Viewareas is done geometrically, and therefore, the Python Application Programming Interface (API) of the 3D modeling tool blender [78] was used to facilitate calculations. The program first loads the POI data created by the QuAD Pre-Inspect Tool and uses the information provided about each POI to generate the geometry of a corresponding Viewarea in Blender. This is done using the API to create and intersect two spheres around the Point of Interest and then intersecting the resulting hollow sphere with four planes created as described in Section 3.2.1. This results in the 3D geometry of the Viewareas being available in Blender for further optimization and Viewpoint sampling.

4.1.3 Optimization Through Overlapping Viewareas

Like the creation of Viewareas, the optimization by overlapping Viewareas is performed in Python using the blender API (see: Figure 4.3). The tool tries to intersect every Viewarea with every other Viewarea. If it finds a combination where the resulting intersected Viewarea fulfills the minimum volume requirements, the new intersected Viewarea is added to the pool of available Viewareas. Also, the original Viewareas are deleted and the information about their POIs is attached to the intersected Viewarea. This process is repeated until no more intersections that fulfill the minimum volume requirements are found. Afterward, the tool has generated a set of optimized Viewareas that still have references to their associated POIs.

4.1.4 Sampling of Discrete Viewpoints from Viewareas

Since the UAV needs discrete points to fly to, Viewpoints must be sampled inside the Viewarea to generate possible navigation targets. Like the two previous steps, the sampling algorithm for Viewpoints described in Section 3.2.1 is implemented in Python using the blender API. The implementation generates a grid of equidistantly spaced Viewpoints by filling the area around the Viewarea with cubes with an edge length of the desired Viewpoint spacing (see: Figure 3.5). The area that must be covered is described by the minimum and maximum coordinates of the 3D bounding box of the Viewarea. Every cube is then intersected with its Viewarea and the intersection volume compared to the original volume. If the two volumes do not match, the cube is deleted. This strategy causes only the cubes that lie entirely inside the Viewarea to remain. To derive the Viewpoints from the cubes, their center coordinates are used as positions for the Viewpoint. Lastly, the Key Viewpoint is selected by finding the Viewpoint with the closest distance to the geometric center of the Viewarea. This assures that a Viewpoint is initially navigated to, allowing a direct view of the POI.

The process is repeated for all Viewareas to create a complete set of Viewpoints and Key Viewpoints. The Viewpoints are then exported in another JSON file that acts as an interface for the path planner. Defining an interface at this point allows the exchange of the ACS planner for a different application at a later time should the need arise. The JSON structure of a POI is described in Listing 2. Each POI is identified by a unique `ID` and a `position`. Since Viewpoints inside intersected Viewareas can observer multiple POIs, `observed_pois` and `orientations` to inspect these POIs are structured in lists to allow for these combined Viewareas with multiple observed POIs. Every VP also has a `rating_value` that stores the score that is calculated by the optimization across inspections (see: Section 3.2.4). Lastly, the `type` attribute identifies it as a Viewpoint

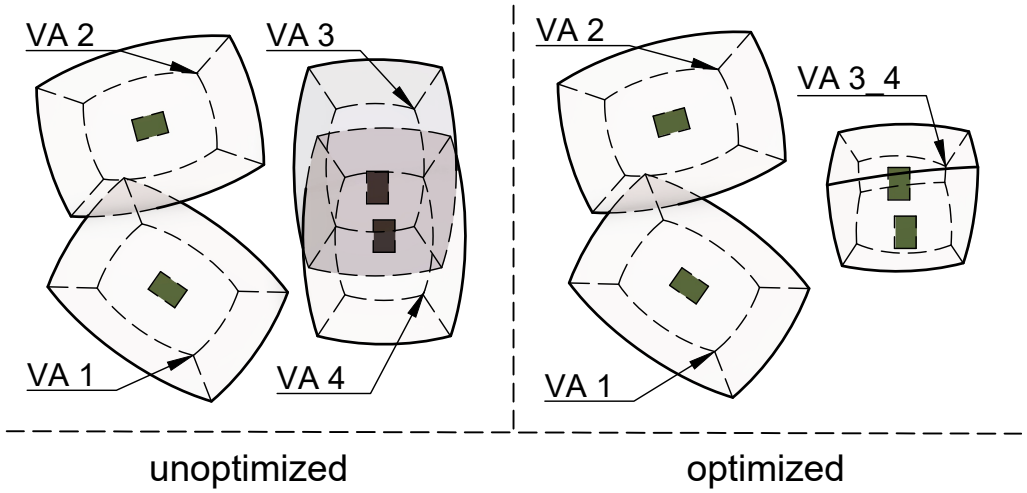


Figure 4.3. Original (left) and optimized Viewareas (right).

or Key Viewpoint, and the `frame_id` defines the reference frame for the position and orientation values.

```
{
  "ID": "TL_Waypoint_A1",
  "position": [-1.06, 1, 3.0],
  "orientations": [[0.0,0.0,1.0,0.0]],
  "observed_pois": ["POI_TL"],
  "rating_value": 0.25739189982414246,
  "type": "viewpoint",
  "frame_id": "s19"
}
```

Listing 2. JSON structure of a single Viewpoint.

4.1.5 Calculation of Inspection Paths

To leverage the computational benefits of the Ant Colony System algorithm, it was implemented in a multithreaded architecture with each thread performing the calculations for one ant. Viewpoints are represented in a fully connected graph and the euclidean norm is used as a distance measure as the movement from one Viewpoint to the next can be performed in a nearly linear movement by a UAV. The architecture of the path planner is focused on interchangeability of the path planning component with interfaces allowing potential later substitution of the ACS algorithm by a different path planning strategy. Additionally, the parameterization of the ACS implementation can be set for every path based on whether a quick, suboptimal solution or a better path requiring more computation time is desired. The latter can be preferential in cases where trajectories are reused repeatedly. The output of the ACS implementation is a path visiting all Points of Interest in the form of an ordered list of Key Viewpoints. The output is formatted in JSON as an array of Key Viewpoints which are structured according to Listing 2.

4.2 Inspection Relative to an Assembly

After the calculation of an inspection path, the execution of the inspection requires the implementation of several subsystems described in Chapter 3. Figure 4.4 describes the architecture of the essential components for performing an inspection. Most of the

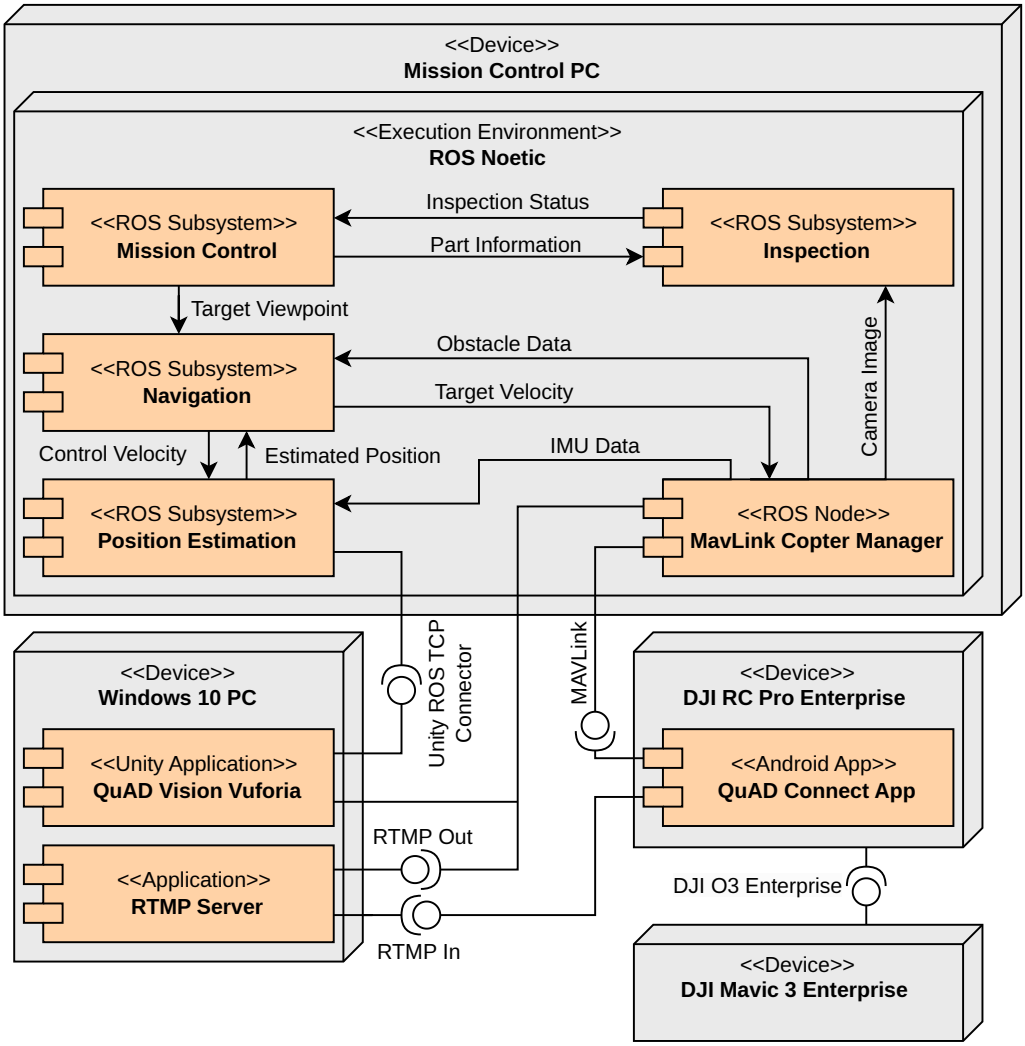


Figure 4.4. Overview of inspection system architecture.

required software can be deployed on a Linux PC running ROS [87], [228]. The plumbing provided by ROS in the form of the publish/subscribe [91] and service-based [90] messaging system allows the individual subsystems to interact. The inspection path generated in Section 4.1 is loaded by the mission control subsystem (see: Figure 4.4). It sends the viewpoints to the navigation subsystem, which uses the estimated position from the position estimation as well as obstacle data from the UAV to send target velocities to the UAV. Once a Viewpoint is reached, the mission control subsystem

provides information about the part to the inspection subsystem, which in turn performs an inspection and gives feedback about the inspection status to the mission control node. If the inspection succeeds, the UAV navigates to the next Key Viewpoint. Otherwise, alternative Viewpoints for the same Point of Interest are tried.

Position estimation requires the camera stream of the UAV to be processed by the Vuforia Engine, the framework chosen for model tracking. The Vuforia engine is only compatible with Windows systems [224], so the software is deployed on a separate Windows PC. The Camera Image is transmitted from the DJI Mavic 3 Enterprise Drone, and the latest version of the system is working with the corresponding remote control via the DJI O3 Enterprise Protocol [249]. The remote runs a custom Android application that relays sensor data and control commands between the UAV and the ROS system, but also streams the camera image to a remote address. The camera stream is provided to an RTMP [161] server running on the Windows PC, allowing other clients to access the stream. One of these clients is the model tracking application (QuAD Vision Vuforia) that uses the stream to estimate the pose of the model tracking. The position is then sent to the position estimation system in ROS using the Unity ROS TCP Connector interface [274]. The position estimation system also receives data from the UAV IMU via its driver node that (in the implementation using the DJI Mavic 3) communicates with the corresponding Android App via MAVLink. This description and diagram only serve as an overview of the connections between the system's individual components. Their implementation is discussed in detail throughout the following sections.

4.2.1 Positioning Relative to an Assembly

Establishing the UAV position is essential for all following navigation and inspection tasks. In the most basic form, it can be achieved using sensor fusion and the data obtained by the UAV odometry and model tracking. The following sections describe in detail how model tracking was implemented to determine a position relative to the assembly. Additionally, an architecture for a sensor fusion approach for relative positioning using a stationary assembly is demonstrated. Lastly, a more complex approach allowing the positioning relative to one or multiple moving objects based on the concepts from Section 3.3.5 is presented.

Implementation in ROS

Figure 4.5 shows the positioning and sensor fusion architecture. The `copter_manager`-node handles all communication with the UAV. While a generic `copter_manager` is shown in the figure, concrete implementations for MAVLink compatible UAVs and a simulated UAV were created. The `copter_manager` acts as an abstraction layer, making the implementation compatible with different types of UAVs by writing a compatible copter manager. For position estimation, the `copter_manager` provides the current velocity of the UAV to both the Kalman Filter node (`dkf_state_estimation_publisher`) and the `camera_tracking_filter`. The latter implements the filtering described in Section 3.3.2 by obtaining the raw transformation from assembly to UAV camera pro-

vided by the Vuforia tracking. The transformation is published again after filtering as `camera_tracking_filtered`.

Since the Kalman Filter calculates the position and orientation of the UAV body's center and not of the camera, the `gimbal_to_baselink_publisher` uses the filtered camera frame to publish a transformation from this frame to the UAV's center using the camera gimbal attitude obtained by the copter manager. This transformation is published as `uav_base_link_frame` which the `dkf_state_estimation_publisher` uses in conjunction with the velocity data from the `copter_manager` and the commanded velocity to publish the final position estimate (`estimated_pose`) to the tf tree. This estimate is used for later navigation.

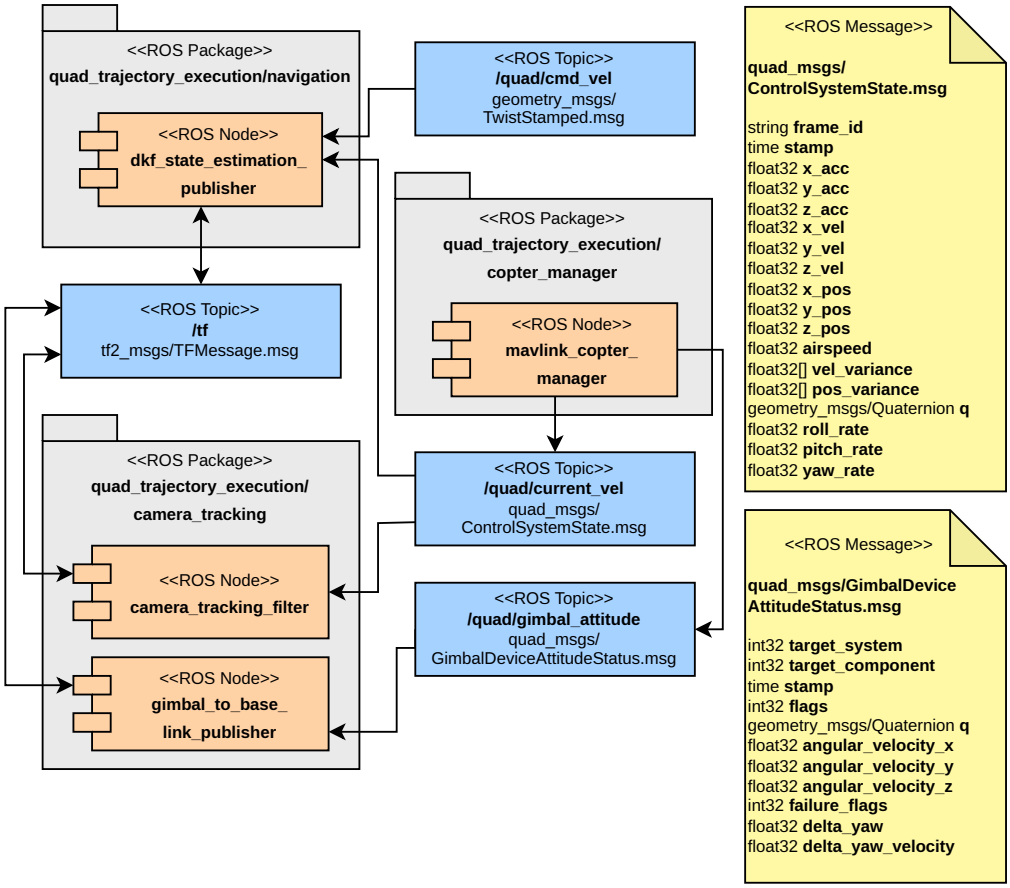


Figure 4.5. Architecture of the position estimation system.

Figure 4.6 shows the resulting transformation tree of this system. The previously not mentioned `comp_odom_publisher`-node defines a point in the production facility as the world origin. This point defines the starting location of the UAV and is calculated using the first model tracking measurement. When working with a moving part, the kinematic model described in Section 3.3.5 is used to update this transformation.

The camera_tracking, camera_tracking_filtered, and estimated_pose-frame are defined relative to the assembly by the Vuforia tracking, the prefilter, and the Kalman Filter, respectively. The uav_base_link_frame provided by the gimbal_to_base_link_publisher is defined relative to the filtered model tracking frame. The tf2 library provided by ROS allows for the lookup of transformations between arbitrary frames that are connected inside the tree, making operations like obtaining the transformation from assembly to the uav_base_link_frame trivial.

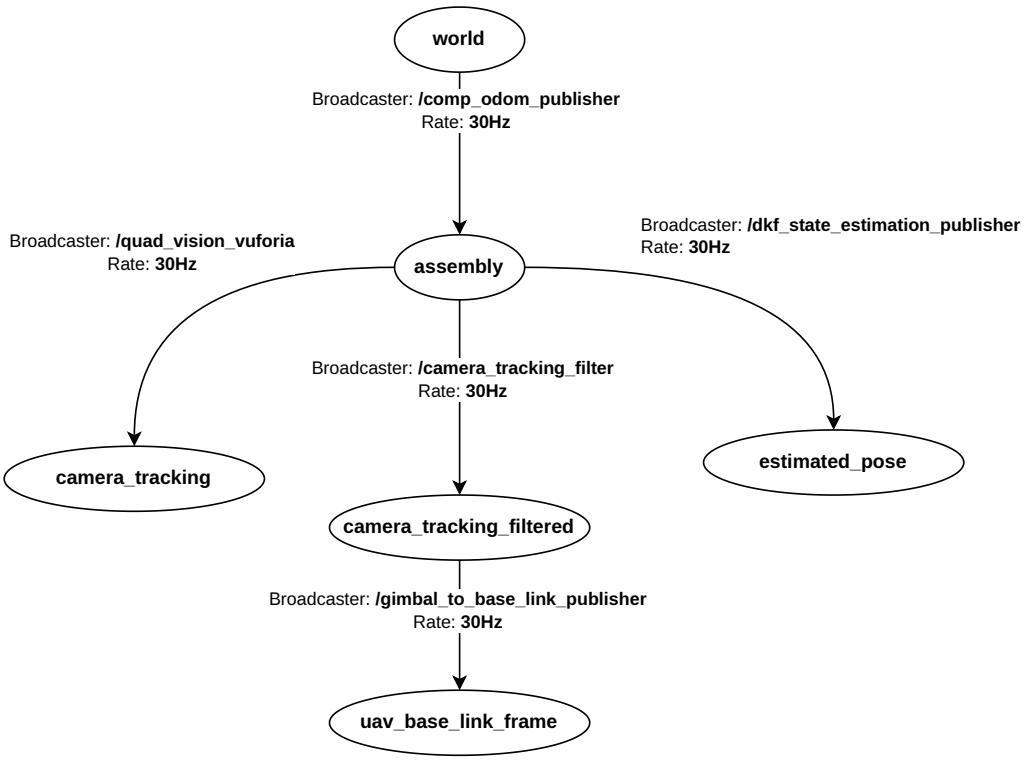


Figure 4.6. Transformation tree created by the position estimation.

Detecting and Handling the Loss of Tracking

In case the visual tracking is lost, the UAV uses its primary means of establishing its position. Continuing the inspection in this state is impossible, and manual intervention is required. While the system is designed to avoid a loss of tracking, it may still occur, and handling the situation properly is crucial to prevent a crash of the UAV.

An independent watchdog node (see: Section 4.3.1) constantly monitors the tracking state. It does this by monitoring the state of the tracking that is sent from Vuforia and the output of the model tracking prefilter. If either the tracking system monitors a tracking loss or the prefilter starts discarding packages due to a detected jump, the inspection is paused if the situation does not improve after a predefined delay. This

causes the UAV to hover at its current position until the situation is resolved either by an operator or the tracking is automatically acquired by the system again.

Model Tracking

Since there are a plethora of model tracking frameworks available, Vuforia Engine [222] was selected to perform the actual task of model tracking. After rigorous testing, its functionality was deemed sufficient for the required task. The major problem with other model tracking frameworks targeted for Augmented Reality applications is their inability to maintain stable tracking when only parts of the model are visible in the camera image. Since the UAV must get reasonably close to the big assemblies it inspects, keeping the entire model in the frame is not always possible. Hence, Vuforia was chosen for its tracking performance. Since its SDK delivers all required features, a commercial solution was chosen instead of an in-house development.

Generating a Tracking Model

Vuforia's model tracking engine requires a tracking model for each assembly that must be inspected. For creating these models, the *Model Target Generator* [225] is provided (see: Figure 4.7).

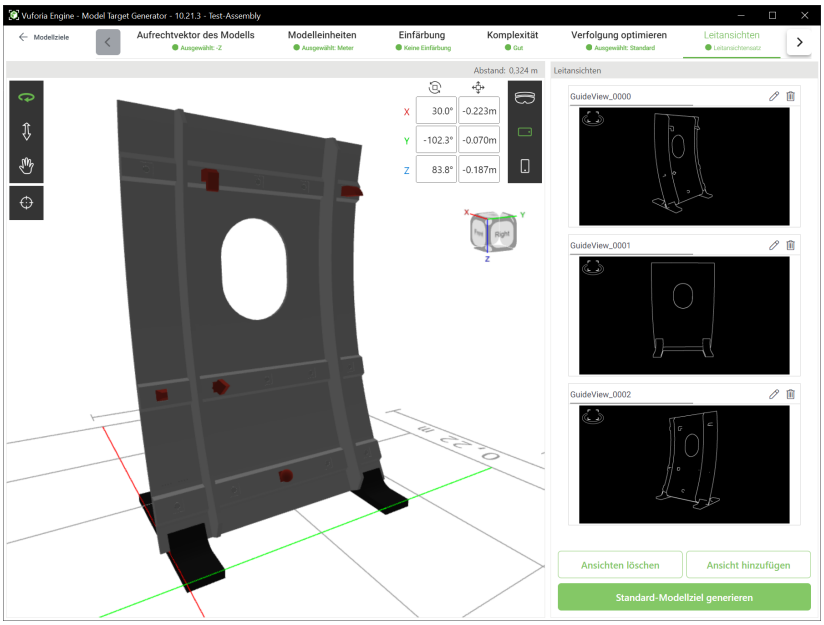


Figure 4.7. Vuforia Engine Model Target Generator.

It allows for arbitrary 3D models to be loaded, their orientation and tracking style to be configured, and it allows for the creation of *Guide Views* [223]. Guide Views are orientations from which the tracking can be initiated. In AR operation, the user must roughly overlap the camera image with the 2D projection of the guide view

to initiate tracking. When using the model for tracked flight, it is recommended to generate guide views for all locations from which the UAV might be launching. Also, it is recommended to create guide views for orientations the UAV encounters at difficult sections of its inspection trajectory. This could be areas with little detail in the model, close distance to the model, high rotation rates, etc. Vuforia also offers a variety of options for *Advanced Views* [220] that allow recognition of models from a wider range of camera orientations and positions, but require processing of the model in the cloud. Due to its requirement for the Unity Engine [273], the model tracking component is implemented as a Unity application and connected to the rest of the system through the Unity ROS TCP Connector [274]. This makes it available in the ROS environment for further sensor processing and fusion.

Limiting Maximum Movement Speed for Tracking

Since model tracking works by tracking the movement of features in the 2D images reported by the camera, quick camera movements where large changes occur between two images need to be avoided. This is because these kinds of movements can result in the software being unable to track the detected features from image to image because they moved too far from their last position, resulting in a loss of tracking. The maximum frame rate of 30 frames per second introduced by the transmission protocol of the UAV also contributes to the problem. Therefore, the UAV movement must be shaped in a way that avoids such movements. This is mainly achieved through limiting the UAVs linear and rotational speeds. In practice, a maximum linear velocity of 0.5 m/s and 30°/s have yielded promising results. In addition, the rotation rate of the camera gimbal is limited to 20°/s in all axes. The combined yaw rate of UAV and gimbal is also limited to 30°/s. In practice, this allows for reasonably fast inspection flights while maintaining stable tracking.

Sensor Data Fusion

The sensor fusion was realized in the `dkf_state_estimation_publisher`-node. It was implemented as described in Section 3.3.4 as a Kalman Filter using the `filterpy` library. As depicted in Figure 4.6, it uses the commanded and measured velocities and orientation as well as the filtered `uav_base_link_frame` as the input and publishes an `estimated_pose` transform relative to the assembly as an output. In addition (but not depicted), the estimated pose is published along with the covariance as `geometry_msgs/PoseWithCovariance Stamped` message for debugging purposes. The covariance for IMU- and model tracking ($\mathbf{R}_i$ and $\mathbf{R}_t$) was measured with the UAV at standstill and set accordingly.

To have the ability to instantiate multiple filters for the sensor fusion with changing reference systems, the Kalman Filter was implemented in a `QuadKalman` class. This also allows for better encapsulation of the functionality and facilitates its exchange against other sensor fusion algorithms. Figure 4.8 shows the structure of the state estimation node.

The `QuadKalman` class acts as a wrapper for the `filterpy.kalman.KalmanFilter` class, initializing all relevant matrices and vectors according to Section 3.3.4. It offers specialized methods for updating the filter with IMU and model tracking data. The `QuadKalman` class itself is used by the `dkf_state_estimation_publisher`-node, which manages communication with ROS. It periodically looks up the model tracking transformation and updates its instance of the `QuadKalman` filter. The node also subscribes to the IMU data published in `/quad/current_vel` and updates the filter using the `update_imu()` method. Periodic state predictions using the commanded velocity obtained from `/quad/cmd_vel` as control input vector, as well as the publishing of position estimates to `/tf`, are also performed by the node in 30Hz intervals.

Position Estimation with Moving Reference Systems

In an optional extension of the system to make it work with moving or changing reference systems, information about the relative velocity of the assembly is needed

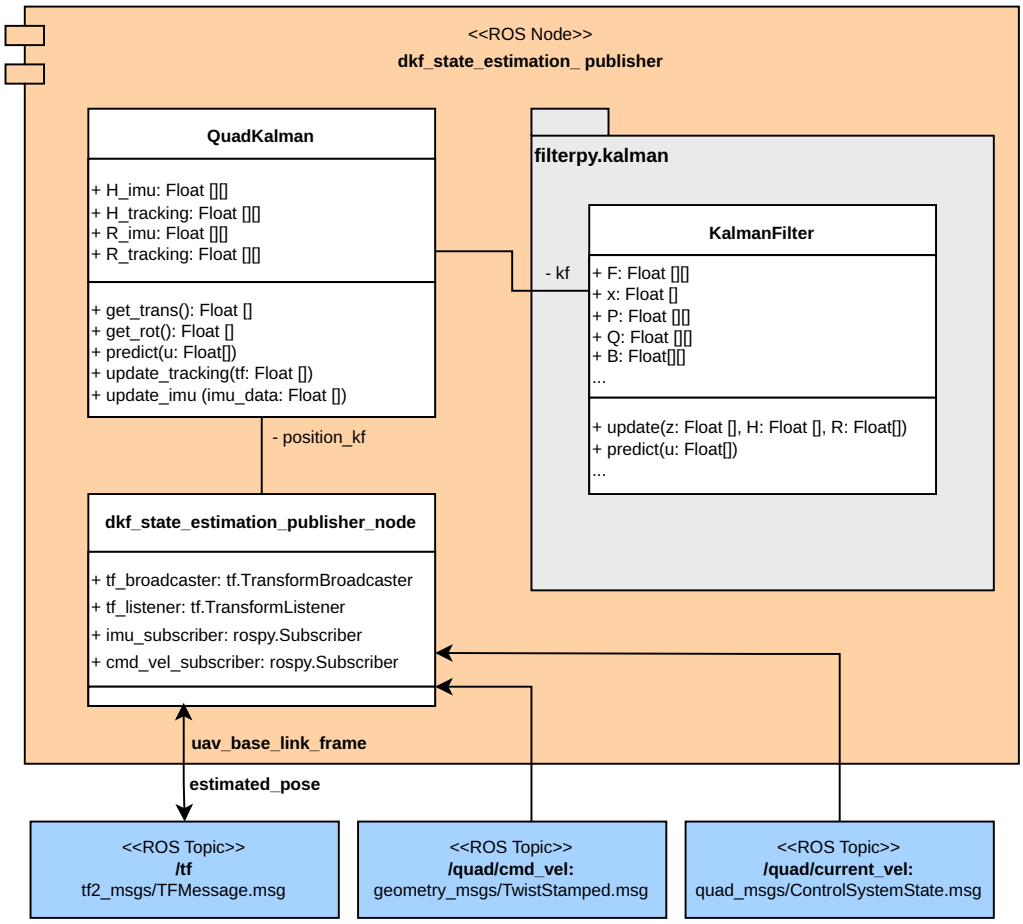


Figure 4.8. Structure of the sensor fusion ROS node.

as an additional input for the sensor fusion node. To achieve this, the kinematics of the assembly and its supporting structure need to be defined first. In ROS, the *URDF* [88] format is used to define kinematic chains with joints and links for arbitrary robots, so it is used to describe the assembly kinematics. Afterward, the Framework *MoveIt* [80] uses the *URDF* description to calculate the resulting trajectories of all joints and the assembly based on the measured joint inputs. The `robot_state_publisher` [83] node is used to publish the transformations of all joints in the `tf` tree. The node `kinematic_localization_publisher` calculates and publishes the velocities of all links of the kinematic chain to the ROS topic [91] `/quad/comp/vel`. The velocities are calculated using numeric differentiation at a rate of 50 Hz. The `QuadKinematicKalman` class (see: Figure 4.9) extends the `QuadKalman` class by adding an `update_kinematic(kin_data: Float[])`-method that stores the velocity data of the assembly in the corresponding attribute. This velocity allows the IMU velocity and commanded velocity to be corrected for the assembly's movement when calling the internal `KalmanFilter`'s `predict()` and `update()`. Similar to the `dkf_state_estimation_publisher` node, the `QuadKinematicKalman` class is instantiated inside a `dkf_kinematic_state_estimation_publisher` node that provides the `QuadKinematicKalman` object with all required sensor data, performs the prediction, and publishes the state estimation in `/tf`. The main difference is that the node subscribes to the `/quad/comp/vel` topic to update its filter with the component velocity data.

Position Estimation with Changing Reference Systems

For dynamic assemblies where multiple filters are needed, the `dkf_multi_kinematic_state_estimation_publisher` is required (see: Figure 4.9).

It is based on the `dkf_kinematic_state_estimation_publisher`, but contains an instance of the `QuadKinematicKalman` filter for each part of the kinematic chain and an additional filter relative to the world frame. Each filter is parameterized as described in Section 3.3.5 and provided with the IMU measurements, tracking data, and the velocity for its corresponding part of the assembly. The currently tracked assembly is determined by the model tracking system in the form of a ROS parameter. This parameter is periodically updated in the state estimation node and the currently tracked `QuadKinematicKalman` instances are provided with the original tracking data, while the tracking data for all other links is transformed into the reference frame of the link first. Additionally, each filter instance is provided with all velocities rotated into their respective reference frame and with kinematic velocities (`kin_data`) relative to its reference frame. Since these extensions require position feedback about the inspected assembly and kinematic information about it or its mounting structure, the advanced position estimation described in this section is only used when strictly necessary.

4.2.2 Obstacle Detection for Collision Avoidance

As mentioned in Section 3.4.3, low-quality sensor data was a big challenge when implementing the sensor detection for the system. Figure 4.10 shows the main architecture of the obstacle detection system. Sensor input is provided again by the `copter_manager`-

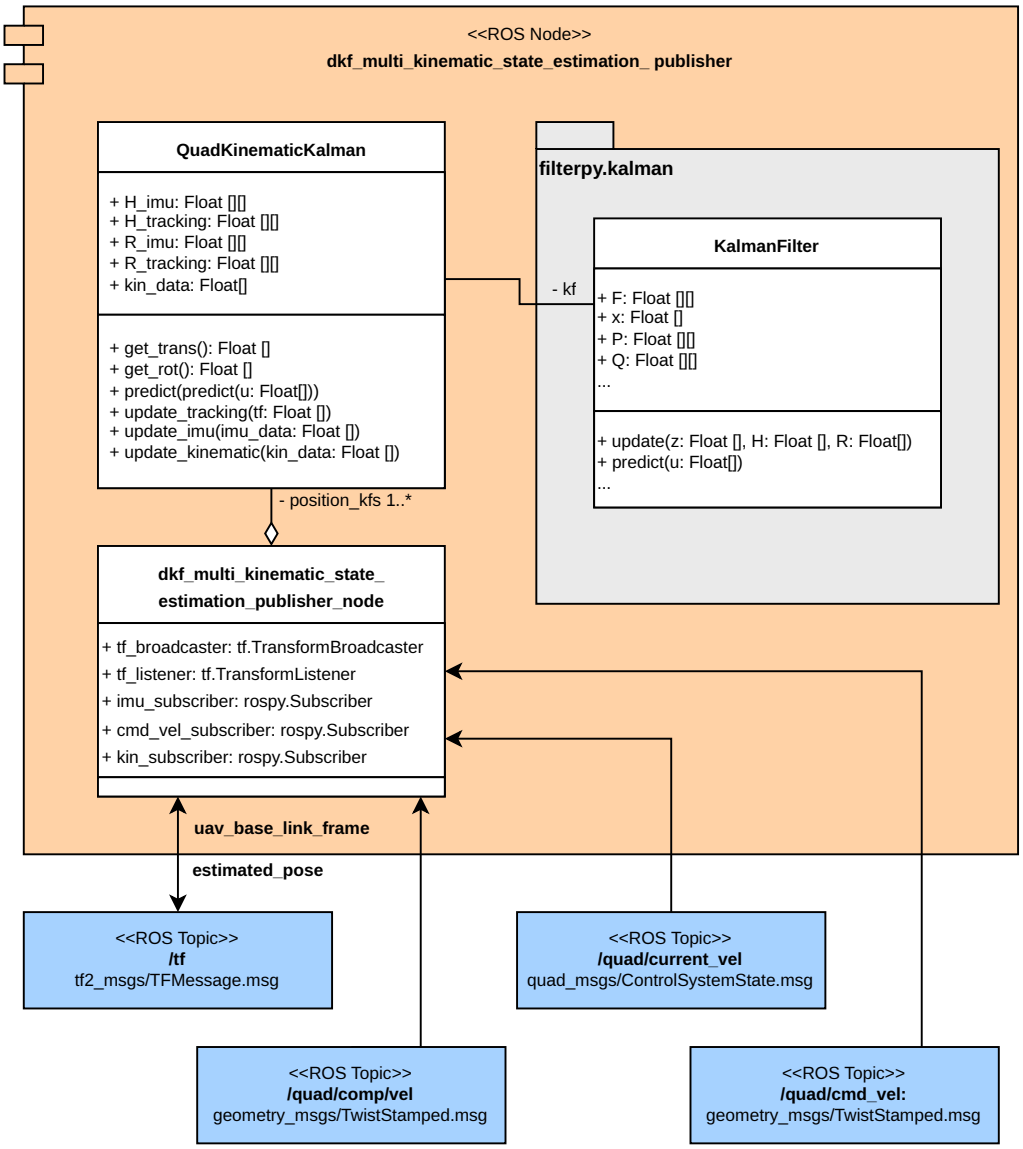


Figure 4.9. Structure of the sensor fusion ROS node for dynamic assemblies.

node in the form of published `ObstacleDistance` measurements. The obstacle avoidance system processes these in the `obstacle_detection` node, which transforms the individual distance measurements into one point cloud [163] each that represents the measured obstacle distances and the measured free space in front of the UAV respectively. These two point clouds are used by a modified implementation of the `octomap_server` to populate the internal OctoMap with new obstacle measurements and clear out areas that have been measured as free. As described in section Section 3.4.3, previously known obstacles can be loaded into the OctoMap on startup. The clearing of obstacles was modified to fit the wide detection range of the sensors of the used UAVs by clearing the

whole sector of the obstacle sensor using multiple raycasts. The separation of measurements that need to be added to the OctoMap and a separate point cloud that is used for *cleanup* allows for the definition of a wider area for removing obstacles like mentioned in Section 3.4.3. The OctoMap server publishes the centers of all currently occupied voxels as another point cloud, which can be used by an arbitrary trajectory planning node (either based on the potential field method or the Anytime Dynamic A* algorithm depicted as `nav_cmd_publisher`). The trajectory planning node can also access the current obstacle sensor readings of the UAV by subscribing to the `/quad/perception`-topic, which is also published by the `obstacle_detection`-node.

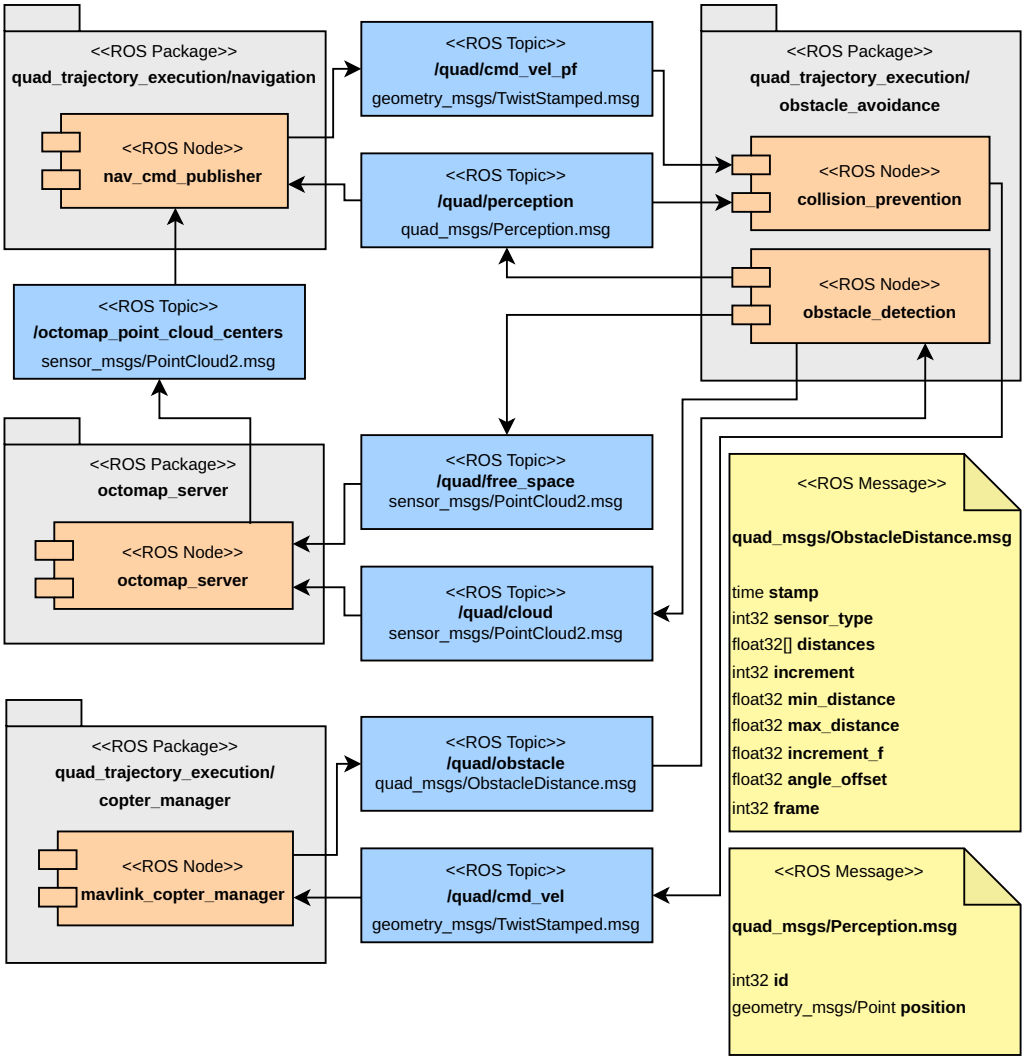


Figure 4.10. Architecture of the obstacle avoidance system.

Secondary Collision Prevention System

As an additional measure to ensure that the UAV cannot collide with its surroundings, independent of the output of the navigation strategy, a secondary `collision_prevention`-node is inserted in series between the output of the `nav_cmd_publisher` and the `quad/cmd_vel`-topic that is used to provide a commanded velocity to the `copter_manager` and therefore directly to the UAV. Note that this architecture bypasses the UAV's internal position controller (see: Section 2.2.3) and replaces it with the `nav_cmd_publisher` by sending target velocities to the drone completely moving position control from the UAV to the proposed system. The `collision_prevention`-node also accesses the obstacles currently detected by the UAV. During regular operation, it just publishes the commanded velocity it receives from the navigation node back to the copter manager. If one of the obstacle sensors measures less than a predefined threshold, it overrides the commanded velocities to 0, which causes the UAV to stop immediately. In this case, the operator has to manually navigate the UAV away from the obstacle before the UAV can continue its flight. To avoid the collision prevention system from triggering, the navigation system is set up in a way that keeps the UAV at a larger distance to obstacles than the threshold set in the collision prevention. This system functions as a second layer of safety if an error in trajectory planning occurs.

4.2.3 Relative Navigation Along Assemblies

In order to perform an inspection flight, the UAV must visit a sequence of Viewpoints in the correct order and plan a collision-free trajectory from each VP to the next. With position estimation and obstacle representation already established, the following section focuses on low-level (VP to VP) and high-level (mission-wide) navigation. First, the high-level mission execution architecture is described before the two implementations for VP to VP navigation are presented.

Mission Control

The mission controllers' primary objective is to send the trajectory planner the preplanned Viewpoints from the path in the correct order. It has to wait until each is reached and, in succession, trigger the inspection of each VP's POI(s). Figure 4.11 depicts how this behavior was implemented in the architecture. The mission execution is done via the `mission_control`-node. It uses the `trajectory_control` node, which loads the preplanned inspection path from a JSON file to obtain information about the next waypoint to fly to. After initialization, it waits for user input before starting the inspection. This user input is provided by starting the mission from the `quad_ground_control` App (see: Section 4.3). This triggers a service call [90] in the mission control node that starts the inspection flight. After performing a takeoff to a preset height, the mission control calls the `/fly_to_action` provided by the `nav_cmd_publisher` implementing the low-level navigation strategy. The `nav_cmd_publisher` gives feedback about the progress of the navigation effort so that the mission control node can detect when a Viewpoint is reached. After arrival on a Viewpoint, the inspection action on the inspection module is called on all specified inspectors for the current Point of Interest. When the inspection

is unsuccessful, the mission control node may navigate to different Viewpoints from the same Viewarea to try to perform the inspection from a different perspective. The number of alternative Viewpoints that are tried before moving on to the next POI can be defined via a parameter. After the maximum number of alternative Viewpoints were tried or successful inspection was performed, the next Key Viewpoint is visited using the `/fly_to_action` again. This is repeated until the inspection of the last POI is completed.

Figure 4.11. Architecture of the navigation subsystem.

Afterward, the Drone returns to the coordinates of its starting point and performs an automated landing.

Implementation of the Obstacle Avoiding Navigation Strategies

The obstacle avoiding navigation strategies using the potential field algorithm and Anytime Dynamic A* described in Sections 3.4.1 and 3.4.2 were implemented as individual ROS nodes as described in the following paragraphs[1].

The architecture of the system was designed in a way that allows for easy replacement of the navigation system implementation using the `quad_msgs/fly_to_pos` action as an interface (see Figure 4.11).

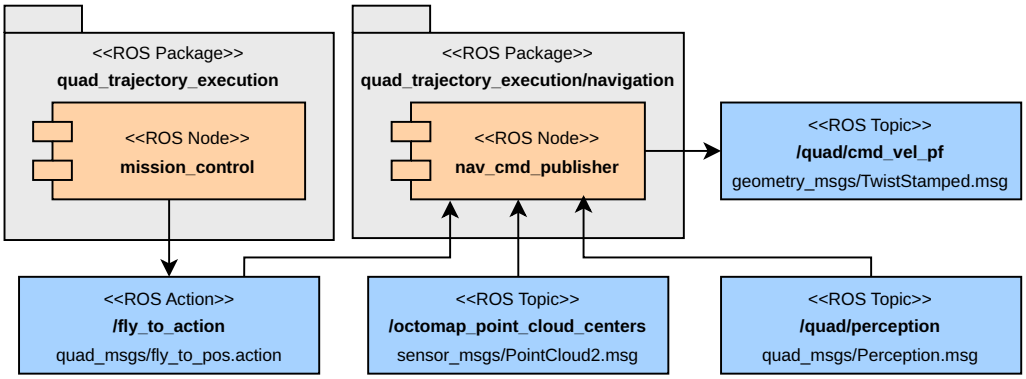


Figure 4.12. Reference architecture for integrating arbitrary navigation algorithms into the system.

Figure 4.11 shows the integration of a sample navigation algorithm into the system as a `nav_cmd_publisher` node. The node implements the `quad_msgs/fly_to_pos` action to be able to receive navigation commands from the `mission_control` node. Obstacle positions and obstacle sensor data can be obtained through the `/octomap_point_cloud_centers` and `/quad/perception` topics respectively. These values are provided by the `octomap_server` and `obstacle_detection` nodes respectively. The node then uses the `/quad_cmd_vel_pf` topic as an interface to send velocity commands to the `collision_prevention` node which in turn commands the UAV using the `mavlink_copter_manager` node as described in the Figure 4.11.

Should additional data be required from the system by the concrete implementation of a collision avoidance algorithm, it can be obtained by implementing a subscriber to the respective topic. Analogously, additional data can be introduced into the system by implementing a publisher and arbitrary services or actions can be executed using Service and Action Clients.

[1] Many thanks to Julian Saupe, who implemented and performed the evaluation of the Anytime Dynamic A* algorithm as part of his bachelor's thesis [243].

4.2.4 Cross-Inspection Learning of Better Key Viewpoints

To provide a way to implement the concepts described in Section 3.2.4, the `trajectory_control`-node the required logic for calculating the scores for each Viewpoint. It is controlled by the `mission_control` node, which orders updating of the rating of a viewpoint. The `mission_control` node can also request alternative Viewpoints for the current Point of Interest in case inspection from the current Viewpoint is not successful. It can do this by calling the `/next_viewpoint_service` of `trajectory_control`.

Calculation of an Optimized Path

To generate a new trajectory based on the updated scores calculated during the inspection, the `tsp_from_viewpoints_action_server` node is added to the system. It encapsulates the implementation of Section 4.1, thus generating a sorted path based on a set of Viewpoints. This service is offered as an action server implementation of the type `quad_msgs/tsp_from_viewpoints.action`. Once the inspection of the last viewpoint is finished, the `trajectory_control` node chooses new Key Viewpoints based on the current scores and calls the `tsp_from_viewpoints` action to calculate an optimized path for the next inspection. This path is then stored as a JSON file with the structure described in Listing 1.

4.2.5 Inspection Subsystem

The inspection subsystem is based on the interface defined in Figure 3.22 in Section 3.5.2. The following sections will first describe how this interface can be used to implement or integrate arbitrary inspection types. Afterward, the implementation of a rudimentary visual presence detection system is demonstrated.

Integration of Inspection Tools

The implemented architecture allows for the flexible addition and specification of arbitrary inspection services into the system. The specification of each Point of Interest, as described in Listing 1, contains a list of inspection types that are used to define the inspection services that need to be used for this specific Point of Interest. Figure 4.11 shows how an inspection service is integrated into the architecture of the system based on a `sample_inspector` node. The node implements the `inspect.action` which is based on the interface described in Figure 3.22. It is used to trigger the inspection for this type of inspection. The mapping is performed through the naming structure of the action servers of the individual inspection nodes. When implementing an inspection service that is triggered, when the type `sample_inspection` is included in the `inspection_type` parameter of the JSON definition of a particular Point of Interest, the inspection service must offer its action server for the `inspect.action` at the location `/inspection/sample_inspection`. The mission control node has information about the Point of Interest that corresponds to the Viewpoint the drone is currently at and can therefore determine the inspection types that are required for this Viewpoint and Point of Interest. Through the naming convention, it is able to call the corresponding action

servers and obtain their inspection status. This architecture allows the addition of new inspections into the system without changes to the existing codebase. As described in Section 3.5.2, the interface for the inspection itself only includes information about the Point of Interest because the individual requirements for data needed to perform an inspection may vary based on the inspection type or tool. When implementing an inspection service node, it can access all data available in the system, including all messages and data that can be obtained through service calls. Therefore, each node can individually access the required information as needed.

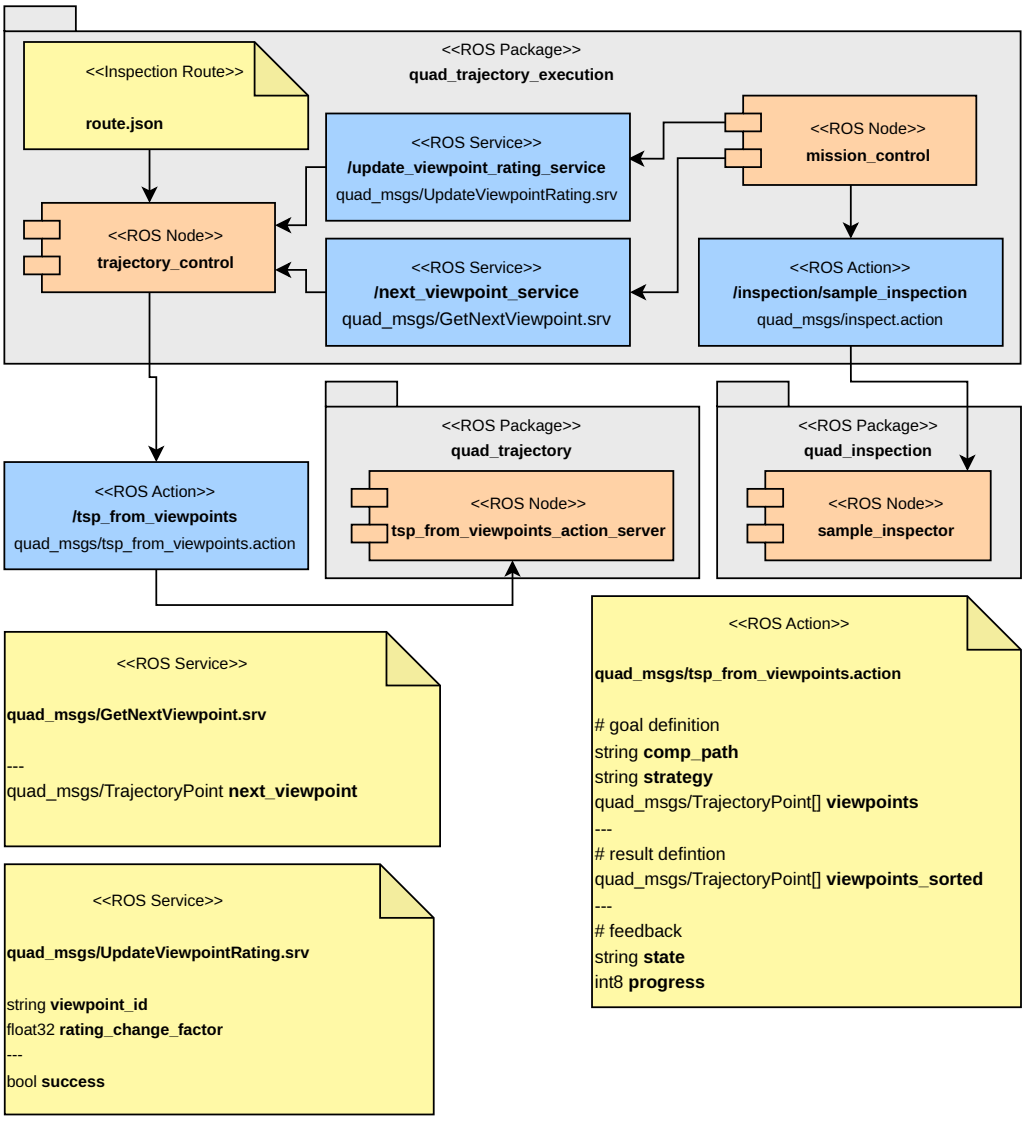


Figure 4.13. Relevant ROS components for the trajectory optimization.

Visual Presence Detection

The implementation of the visual presence detection system was separated into two main components: The first is the `similarity_inspector` that performs the image processing and template matching steps described in Section 3.5.1 using the camera image, provides feedback on correct installation, and visualizes errors. In addition, the render service is needed to generate reference renders for the similarity inspector. The similarity inspector offers an action of the type `inspect` to the mission planner that is called every time a Point of Interest is inspected that has the inspection type `presence_detection` specified. It also interfaces with the render engine via the `generateReferenceRender` service to receive reference renders of the images it receives in the `inspect` call.

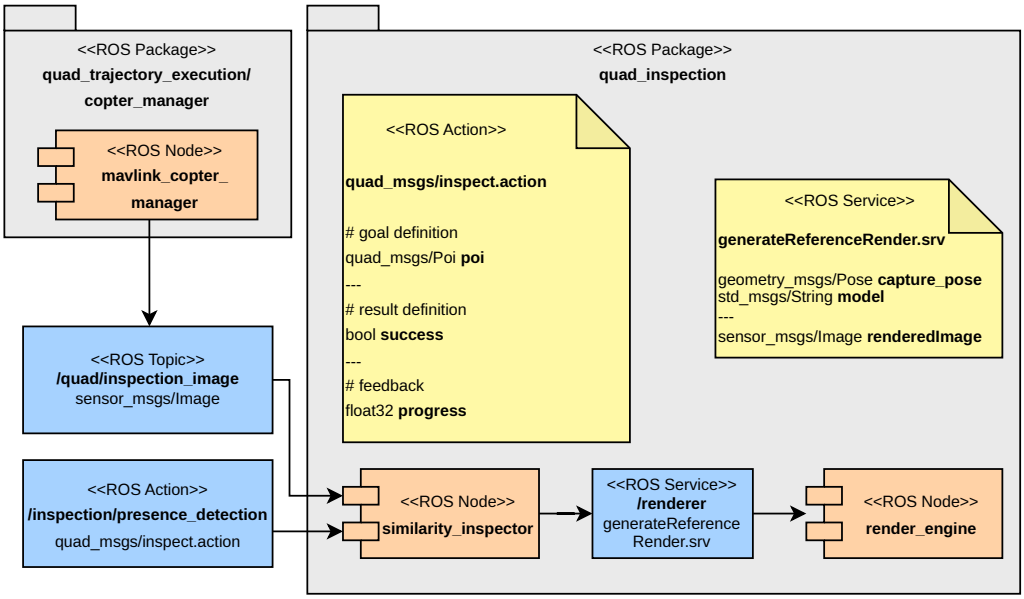


Figure 4.14. Component diagram of the inspection subsystem.

Render Service

The render service is used to provide reference renders to the similarity inspector. It is implemented as a ROS node and provides the `generateReferenceRender` service for this purpose. Upon receiving a service call, it loads the reference model from the path defined in the parameter `model`. It does this like all other 3D-related tasks, using blender [78] as the 3D engine. The model is loaded into a scene where the rendering camera and some lights are added. The cameras' focal length and sensor size are adjusted to match the actual camera. When the `generateReferenceRender` service is called, the camera is positioned and oriented based on the actual camera coordinates at the time of the inspection, which are included as a parameter of the service call. An image is then rendered and used as the return value of the service call.

Similarity Inspector

The similarity inspector was implemented as a ROS node. It provides the ROS action for the inspection. The concepts described in Section 3.5.1 are implemented in this node. It works by detecting the image of a render of the Point of Interest in the captured image that is obtained from the `copter_manager`. The action call returns either true or false, based on whether a match is found. In addition, the node stores the reference render as well as the original image with a graphical overlay highlighting the detected part, so they can be accessed by the post inspection tool (see: Section 4.4).

Upon receiving the action call, the node calls the `generateReferenceRender` service call of the renderer to obtain the references for a correctly installed part and the entire assembly. Subsequently, all images are checked for their dimensions and resized to match the smallest image if necessary. Afterward, the template matching algorithm described in Section 3.5.1 is performed using **SuperGlue** [172] for aligning the renders to the image and **OpenCV** [205] for all other image processing tasks. Template matching is done using the normalized cross-correlation coefficient of the OpenCV's [205] `matchTemplate` function. To visualize the error to an operator, the original image is overlaid with the confidence value of the template matching algorithm and a box at the position where the template was matched. This image is then stored alongside the confidence similarity value of the match and the reference renders for later analysis in the post-inspection tool. After finishing these operations, the action call returns, whether it was able to successfully detect the Point of Interest in the provided image. If the Point of Interest was successfully detected, the `mission_control` subsystem proceeds with inspecting the next Point of Interest. Otherwise, the UAV might try to use alternative Viewpoints to retry the inspection of the Point of Interest from a different perspective.

4.2.6 Photogrammetric Documentation of Assemblies

For implementing the concepts for photogrammetric reconstruction of assemblies using the HEDAC coverage path planning algorithm as described in Section 3.6, a derivate of the existing system for the inspection according to Figure 4.4 is used[2]. In the case of the open loop system, the path is precalculated in a separate program and input to the existing implementation with the inspection system configured to take an image of the Point of Interest at each Viewpoint. For the close loop system, the mission control and inspection node were replaced with a photogrammetry node that calculates target velocities according to feedback it receives from an external instance of AliceVision Meshroom according to Section 3.6. The images for the photogrammetric model creation are provided by the photogrammetry ROS node, which receives them from the UAV via the copter manager. AliceVision Meshroom [104] was chosen as a solution for the photogrammetric reconstruction as it features an intuitive node-based workflow allows the configuration of custom photogrammetry pipelines. This allows the system to be tweaked to the specific use case, including the application programming interface (API)

[2] Many thanks to Simon Hornung, who implemented and evaluated parts of the photogrammetry system as part of his master's thesis [118].

necessary for integration into the control loop. Additionally, the program covers nearly the entire photogrammetry workflow, including image preprocessing and creating a textured, meshed model [283]. The following sections will cover an explanation of the implemented photogrammetry pipeline as well as the architecture of the open- and closed-loop system.

Photogrammetry Pipeline

Photogrammetric reconstruction was performed in the open source tool AliceVision Meshroom [104]. It enables users to create customized *Pipelines* that consist of individual processing nodes connected to each other in a *pipes and filters*-architecture. Each node encapsulates a processing task and it's inputs and outputs can be connected to other nodes to form a directed graph. The tool also provides a Python API to integrate custom functionality into the system.

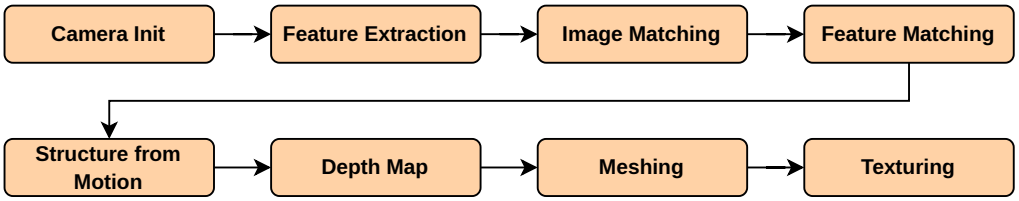


Figure 4.15. Overview of the photogrammetry pipeline.

For creating the photogrammetric reconstruction of the models, a parameterized derivate of the default photogrammetry pipeline provided by AliceVision Meshroom was used. The general architecture of the pipeline is shown in Figure 4.15. In the camera initialization step, camera information like focal length, sensor size, and exposure information are loaded from the image metadata and additional information is synthetically generated or estimated where possible.

The succeeding feature extraction step focuses on detecting features that can be tracked throughout multiple images and thus help with aligning multiple images and determining the cameras' position. The extracted features need to be independent of rotation, scale and lighting. Supported algorithms include natural feature extractors like SIFT [167] or AKAZE [13] and marker-based features like CCTags [32] or AprilTags [203] which can improve feature density on objects with little texture information. However, since the entire positioning system was designed to work without markers, the focus on the photogrammetric reconstruction process in this work will be on the reconstruction of models without the help of additional markers. Examples for extracted features would be edges or corners in the images or areas with a distinct texture. The image matching step uses these features to find pairs of images that contain a high number of identical features. The feature matching step then uses individual image pairs and tries to find each features' corresponding counterpart in the other image. This step is done in preparation for the reconstruction of depth information from the images. The Structure from Motion step (see: Ullman et al. [282]) reconstructs the camera positions

and orientations and 3D positions of features from the input images. This is achieved by detecting and matching features across multiple images. The step uses triangulation from matched points to compute their 3D locations. This generates a sparse point cloud and the camera poses, the final, dense 3D model is calculated in a later step. The Depth Map Estimation step computes a map for each image, which describes the distance from the camera for each pixel. This is achieved by comparing pixels with those in overlapping images to calculate their depth using Semi-Global Matching [114]. The depth maps are then converted into a single 3D mesh consisting of vertices, edges, and faces (triangles) that approximate the surface of the scene. This is done by combining depth information from all images into a single dense point cloud. Subsequently, a surface is generated on this cloud. Additionally, noisy data is filtered, and inconsistent points are removed in this step. The reconstructed geometry in the form of a dense, 3D mesh is then textured in the Texturing step. In this step, the original input images are projected onto the 3D mesh to produce realistic textures by finding the best combination of images to apply to each part of the mesh. The camera poses and visibility information are used to determine which images best cover each face of the mesh. The step also color blending between images, lighting correction, and seam minimization. The final output of the algorithm is a textured 3D model in the form of a mesh file like .obj with associated texture images which can later be viewed by an inspector to look for damages or other manufacturing errors.

Open Loop Coverage

The open loop system implements the HEDAC planner and photogrammetry system as separate components that can be deployed on an arbitrary system. This is possible, due to the strictly sequential nature of all three components (planner, inspection flight and photogrammetric reconstruction). They are separated using the generated Viewarea and Point of Interest JSON files (in case of planner and inspection flight) as well as the captured images (in case of the inspection flight and photogrammetric reconstruction) as interfaces between the individual components. The JSON files are structured as described in Sections 4.1.1 and 4.1.4 using the JSON structure defined in Listings 1 and 2. This allows the existing mission and Viewpoint inspection system to be used for the open loop system. Instead of an inspection, the system takes an image at the location of the Viewpoint. For this purpose, a `photographer` node that offers an `inspect` action that captures an image upon being called using a service provided by the `copter_manager`. After completion, the captured images serve as only input for the photogrammetry framework.

Closed Loop Coverage

For the closed loop system, the planner and photogrammetric reconstruction framework were integrated into the existing system (see: Figure 4.16). The modified components are highlighted in blue. The inspection subsystem and mission control node was replaced with a photogrammetry subsystem that directly generates the target velocity for the obstacle-avoiding navigation system. Captured images from the drone are automatically

downloaded via the copter manager and passed on to the photogrammetry implementation in AliceVision Meshroom via the photogrammetry subsystem. The resulting 3D model is returned to the photogrammetry subsystem where it is compared to the original mesh based on the CAD model as described in Section 3.6. The API of the 3D modeling tool Blender [78] is used for this task. The resulting difference between reference and current render is used to generate information for local cooling for the HEDAC algorithm, which produces a new target velocity for the UAV.

AliceVision Meshroom was deployed on the Windows PC to reduce CPU load on the mission control PC.

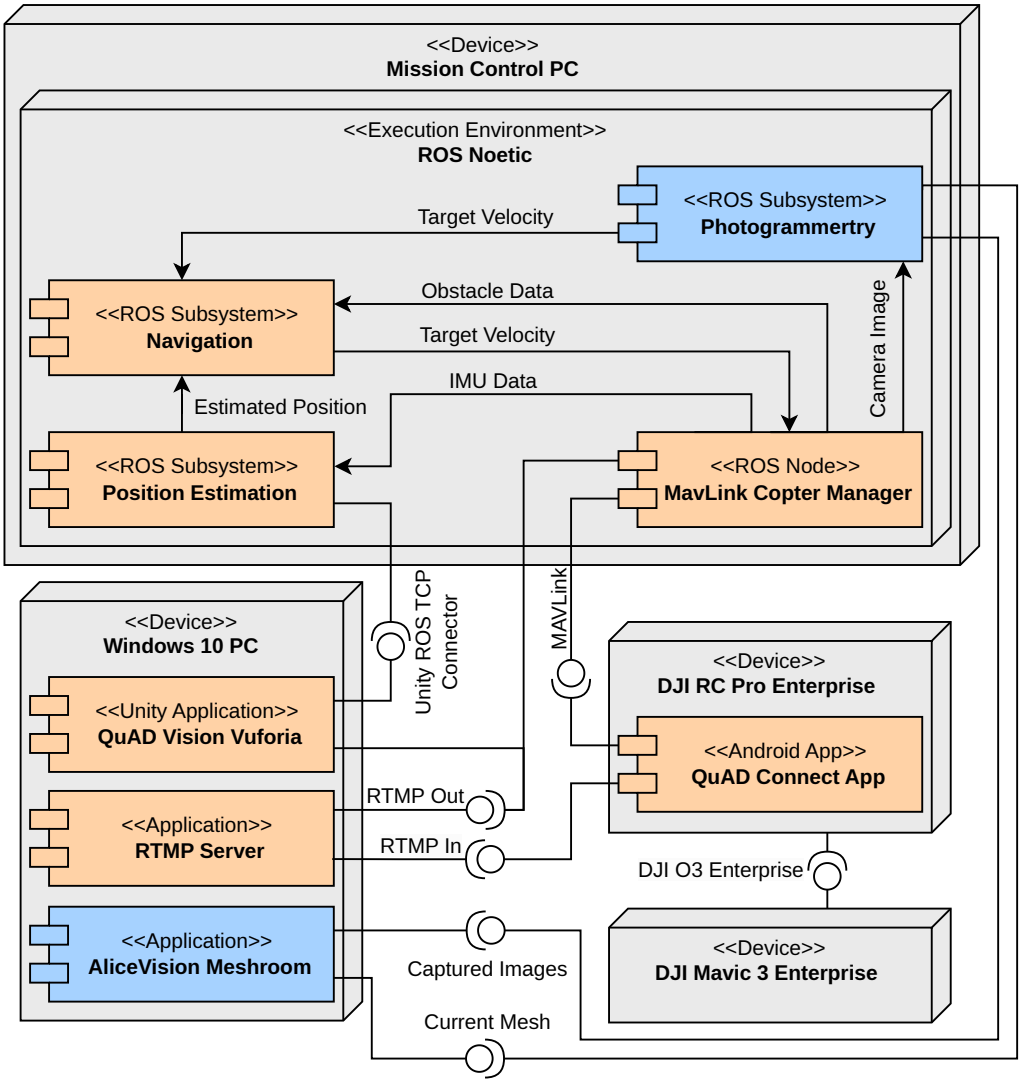


Figure 4.16. Overview of the architecture of the photogrammetric system.

4.3 Monitoring and Controlling the Inspection

Even an automated inspection needs a means for an operator to start the inspection process and monitor its progress. Similar to the situation of self-driving cars, not every scenario in an inspection can be foreseen and covered by the programming of the UAV. In addition, depending on the quality of the obstacle sensors, certain obstacle types may not be recognized, and obstacle avoidance may fail under certain conditions.

For this reason, the *Quad Ground Control*-App was developed[3]. It serves as a ground control station (see: Haque et al. [108]) to give the operator an overview of the state of the mission and aircraft. It allows for the observation of critical parameters of the UAV, verifies its behavior, and allows user intervention if the mission needs to be altered or canceled. This is not only helpful during production use but was of great help during the development of the system. Figure 4.17 displays the App's main screen. The top right area of the screen is used to display the live camera stream of the done. It is also possible to display a virtual camera image generated by the simulation environment if a simulated flight is performed. This is what is visible in the figure. The left sidebar features options for viewing logs and changing settings but mainly provides more information about the status of the aircraft. Information about battery level, connection status, and visual tracking performance are displayed, as well as a visual representation of the obstacle sensor measurements.

The mission status is displayed on the lower left, along with an option to start, pause, and cancel a mission. To the right of this panel, information about the inspection is displayed. This includes the name of the assembly that is inspected as well as the trajectory, POIs, and obstacles that are used to perform the inspection. The next panel to the right displays the current waypoint and the progress on the trajectory to the waypoint. The Position Panel displays the UAV's current estimated position along with the commanded and current velocity. On the right side, four slider buttons allow the operator to send high-level control commands to the UAV. Using these buttons, the mission is paused, and the UAV either hovers, lands, returns to its home positions, or immediately turns off all its engines for an emergency stop. To avoid accidentally triggering these functions, they were implemented as a slider button that requires the white icon on the left to be dragged to the right to perform the action. The application was developed as a ROS node using python and PyQt as framework for the user interface. It interfaces with the rest of the system using the various topics, services and actions described in this chapter.

4.3.1 Watchdogs

To reduce pilot task loading and minimize the impact of human error, several checks are performed automatically by individual watchdogs. These watchdogs also automatically take preventative actions to avoid accidents. The first watchdog is the previously mentioned collision avoidance watchdog. It monitors the UAVs' distance to the closest obstacle and automatically triggers an emergency hover when it falls below the preset

[3] Many thanks to Simon Hornung, who implemented the QuAD Ground Control Application as part of his bachelor's thesis [117].

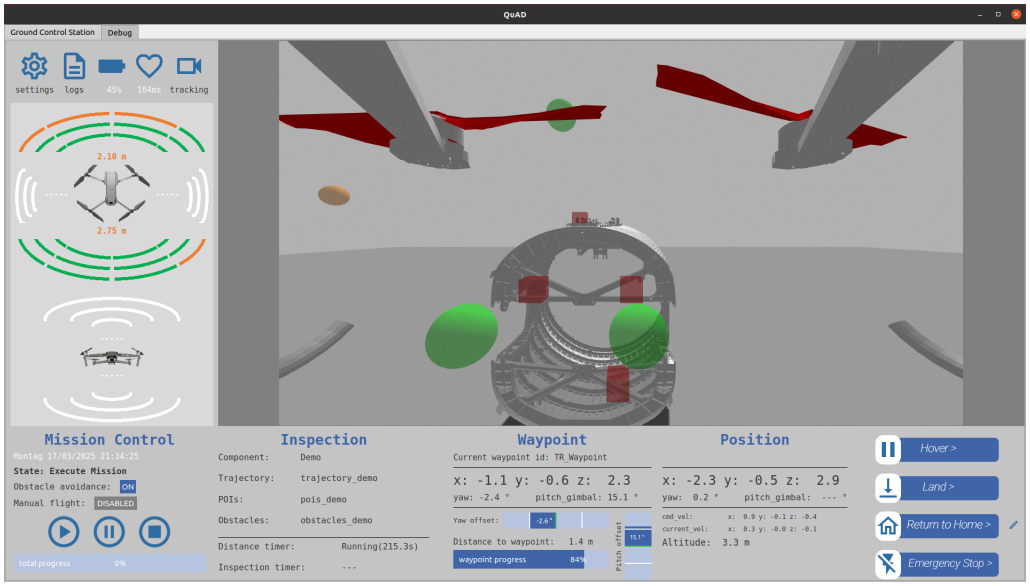


Figure 4.17. Overview of the QuAD Ground Control App.

threshold. This allows the pilot to manually maneuver the UAV clear of the obstacle and continue the inspection afterward. The watchdog for the UAV battery level continuously monitors the battery voltage in the background and triggers alarms if the battery level gets too low. On reaching 20 % remaining battery charge, the mission is paused, and a message is displayed to the operator informing him about the problem. Should the operator continue the inspection, the UAV automatically performs a return to home and lands once it reaches 10 % remaining charge. In addition, the model tracking status is monitored by a separate watchdog. If the model tracking is lost for more than 3000 ms, the watchdog makes the UAV perform an emergency hover to avoid movement in an unwanted direction. Since the UAVs used in the system can perform a stable hover without external positioning systems, hovering without position feedback from the model tracking is possible. All watchdogs also cause notifications in the QuAD Ground Control App that inform the user of the problem and propose potential solutions. All watchdogs are also realized as separate ROS nodes and can be individually included or excluded in the launch configuration should a certain service not be needed.

4.4 Visualization of Inspection Results

A separate tool was developed to visualize the results of the inspection. It was created in C# using the Unity game engine [273]. The required data is exported after the inspection flight in a predefined folder structure containing the flown trajectory, Points of Interest, Viewpoints, captured images and videos as well as the result of the performed inspection actions for each Point of Interest. The tool visualizes the data generated by the inspection in various ways. In the application, the user is presented with a 3D-model of the inspected assembly that was generated from the CAD data and annotations. This part of the application can be seen in Figure 4.18. The model augmented with information about the POIs and UAV trajectory. One of the main features is the ability

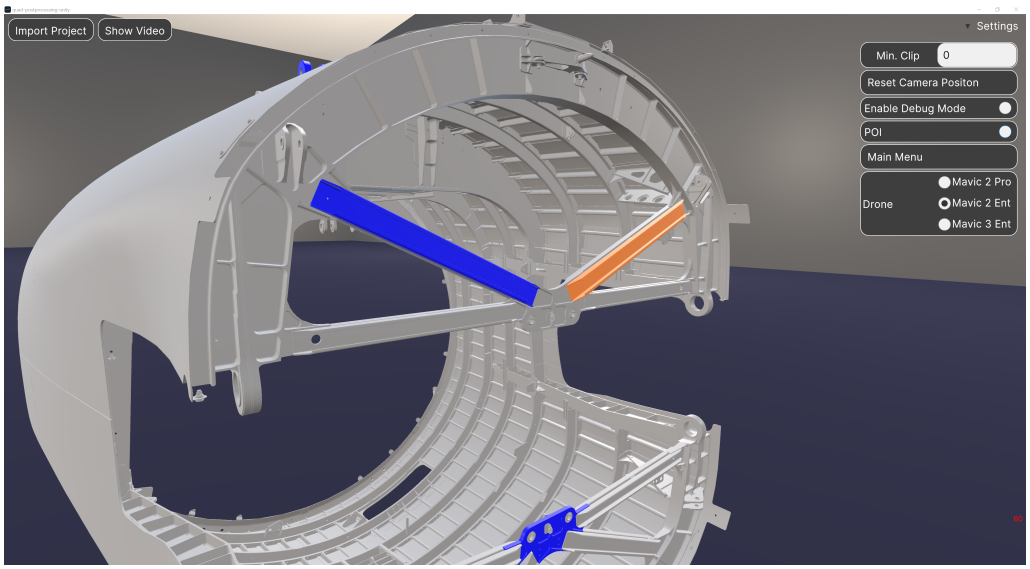


Figure 4.18. Augmented 3D model of the assembly.

to view the recorded video segments of an arbitrary part by selecting it in the model. Based on the flight trajectory of the UAV and the camera focal length, the sections of the video that contain this part are then calculated. The video of the inspection is then presented to the user using a specialized video player. The seek bar of the player contains color-coded information about the visibility of the part at the respective time (see: Figure 4.19). In addition, the player by default only plays the green segments that represent the sections of the video where the selected part is in the shot. After an arbitrary green segment ends, the playback skips forward to the beginning of the following green segment. In addition to video information, the pictures taken during the inspection can also be accessed from the 3D-model-view. The user can view the images that were taken of a POI by clicking on it. If available, additional information about the inspection status of the POI can also be accessed this way. The inspection result is, however, also displayed in the 3D-view through color-coded POIs. While not implemented, it would be possible to add plugins that export the gathered inspection data to various PLM systems to store inspection results at a centralized place.

Figure 4.19. Custom Video Player for selective playback of an inspection flight.

Summary. This chapter presents the hardware used for testing the developed system in form of the UAVs used and assemblies that were obtained or constructed for test-inspections. Furthermore, the developed simulation environment is presented and the results of the proofs of concept and evaluations of the individual concepts are discussed.

5 Prototypes and Evaluations

"In God we trust,
all others bring data."

— W. Edwards Deming

To verify the concepts described in Chapter 3 and their implementation, multiple prototypes were built. These were used to prove the concepts and evaluate the solutions against the current state of the art. This chapter discusses the prototypes that were built for the thesis as well as the mentioned proofs of concept and evaluations that were performed utilizing them. It also covers the results of the evaluations that were performed. To understand the different prototypes, the hardware and simulation environment used for simulated tests are described first. This includes the flight arena in which all non-simulated tests were performed, the different UAVs used, and the structures used for inspection tests. In addition, the simulation environment that was built for the simulated tests is presented. Afterward, the focus is set on the evaluations performed for the case study that covers the inspection of an aircraft fuselage part in production. This includes performance tests of the path planning algorithm and the proposed optimization through the intersection of Viewareas. Then, different evaluations of the relative navigation alongside an assembly are discussed. In addition to evaluating the performance of the inspection of a stationary assembly and the obstacle avoidance system, the advantages of the path optimization across multiple similar inspections are shown. The last proof of concept presented in this section consists of a performance demonstration of the coverage algorithm by performing a photogrammetric reconstruction of two fuselage parts. Because no wind turbine was available for real-life tests and the use case also applies to the manufacturing case study, the coverage algorithm was tested in this case study. Nevertheless, multiple tests were performed to evaluate other concepts of the wind turbine inspection case study. This includes the positioning accuracy relative to a moving wind turbine blade. This was done using a fixed reference frame and while switching reference systems between independently moving parts of the turbine. In addition, the localization performance with and without the help of model tracking is demonstrated.

5.1 Evaluation Hardware and Simulation

The tests in this work were conducted partly with real hardware and partly in simulation. A simulation environment for virtual test flights made it possible to quickly verify implemented concepts and consider scenarios, such as the inspection of a rotating wind turbine for which no suitable hardware was available. For this reason, this section not only discusses the flight arena within which the hardware experiments were conducted, as well as the UAVs used and the components tested, but also introduces the developed simulation environment.

5.1.1 Flight Arena

The indoor flight arena in which all real-world experiments were carried out was also conceptualized and implemented as part of this thesis. The general structure of the arena can be seen in Figure 5.1. It consists of an aluminum truss construction spanning a volume of 16 m (length) by 8 m (width) by 6.5 m (height). The arena is fitted with a camera-based Vicon motion capture system [279] that allows tracking the position and orientation of objects inside the arena that are equipped with infrared markers. These markers work by either passively reflecting infrared light emitted by the camera or actively emitting this light with LEDs. Multiple cameras then capture this light, and a central control unit triangulates the position of each marker based on its relative position in the individual camera images. For the detection of the markers, 20 Vicon Vantage V16 [280] cameras are mounted to the trusses with an additional 10 Vicon Vantage V5 [281] available to be placed on tripods anywhere inside or around the arena. The latter allows for better coverage of areas where the view of the cameras on the trusses may be obstructed. To perform test flights safely, the side walls of the arena are closed off using nylon nets used for sports venues like tennis courts. This prevents UAVs and other objects from leaving the arena. To use different parts of the arena for other experiments, it can also be split into three identically sized sub-arenas using the same kind of nylon nets that line the perimeter of the area. This also allows humans to enter a sectioned-off part of the arena while the rest is used to perform test flights. The large inside volume of the arena allows for test flights with larger structures like the tail section of an Airbus A320 aircraft, described in a later section of this chapter. While the Vicon tracking system is not used for the final position estimation of the UAV, its ability to track objects with sub-millimeter precision [187] allows it to be used as a ground truth for evaluating the precision of the model tracking-based positioning approach. It was also used as a substitute to evaluate navigation and obstacle avoidance algorithms before the model tracking positioning was fully functional. The Vicon Tracker software runs on a separate Windows PC and provides transformations of all tracked objects relative to a predefined origin. These transformations are provided in the ROS transformation graph via the vrpn_client_ros [79] package. Figure 5.1 shows the current state of the flight arena. It offers enough space not only to contain the A320 tail section but also to accommodate multiple other experiments as well as desks for personnel to develop and execute the software needed for these experiments and demonstrators.

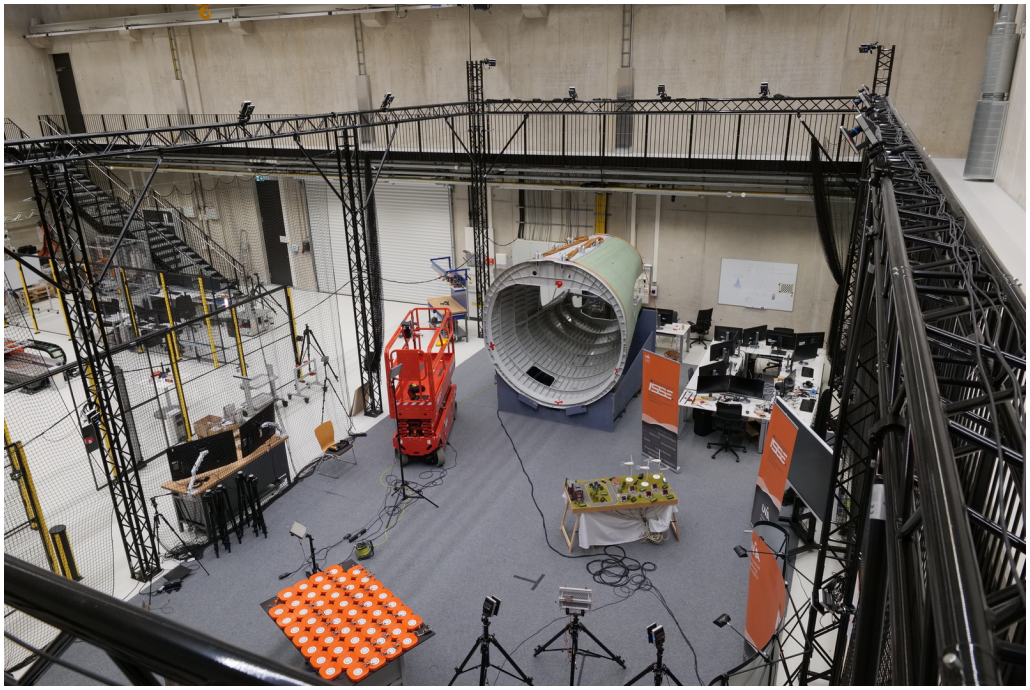


Figure 5.1. ISSE Flight arena.

5.1.2 Inspection Drones

During the development of the system presented in this work, multiple UAVs were used to perform test flights. The UAV models chosen for the experiments were the DJI Mavic 2 Enterprise Dual (see: Figure 5.2) [251] and the DJI Mavic 3 Enterprise (see: Figure 5.4) [252] out of the DJI Mavic product series. Multiple factors contributed to this choice. The DJI Mavic UAVs come with an onboard camera that is stabilized by the UAV on three axes (roll, pitch, yaw) [250]. The camera gimbal can also be manually moved in all axes to point the camera at a part that must be inspected. The camera image can then be transmitted via the network to the model tracking tool to establish position tracking and to any tool that is used to perform the actual inspection. While the live video can be transmitted with up to 1080p, the UAV can also record videos with a resolution of up to 3840×2160 at 30FPS and take pictures with a resolution of up to 48 megapixels. The maximum resolution and frame rate that can be achieved varies from model to model. The captured images and video can be transferred from the UAV automatically using the built-in API. This API and the corresponding Mobile SDK are another reason this series of UAVs was chosen for the experiments. Using an App that was developed for this exact use case with the Mobile SDK V4 [253] and v5 [254], the UAV's functionalities were extended in a way that allows them to accept MAVLink [219] commands that are sent from the main ROS program. More details on the QuAD Connect App follow later in this section. Another factor that led to the selection of the DJI Mavic series UAVs is that they are lightweight and compact enough to perform flights inside confined spaces like the

inside of an aircraft fuselage. The UAVs are built with a plastic chassis and use composite propellers with soft edges. Compared to other UAVs that feature a carbon fiber chassis and props, this is less likely to cause significant damage to the environment should a collision or crash occur. Additionally, the UAVs are equipped with obstacle sensors and an onboard obstacle avoidance system. While the built-in obstacle avoidance system is disabled for all test flights to get closer to the object to be inspected, the obstacle sensor data is used via the QuAD Connect App to perform the obstacle-avoiding navigation described in Chapter 3. In addition, an in-house solution for crash prevention was implemented that allows the UAV to get much closer to obstacles but still prevents collisions reliably if the obstacle-avoiding navigation fails. This system uses another feature that is especially favorable in DJI Mavic UAVs: Stable hover without GNSS lock. While most other UAVs require position feedback from a GNSS system like GPS to achieve stable flight, the DJI Mavic series of UAVs allows for a stable hover of the UAV using only its onboard sensors. While the Manufacturer does not disclose precisely how the UAVs maintain a stable position without a GPS-fix, it is assumed that data from the IMU, obstacle sensors, and a downward-facing optical flow sensor, as well as a barometer, is used to prevent movement in all axes. This allows for the implementation of a *soft emergency stop* that allows the UAV to hover in place if, for example, it gets too close to an obstacle or the communication to the central control program is lost. In addition to this general overview, the following paragraphs discuss the specifications of the individual DJI Mavic models that were used throughout the tests and give a detailed look into the QuAD Connect App that was used to relay commands from the ROS program to the UAV and functions as a means to control the UAV in situations when the operator must intervene with the automated program.

DJI Mavic 2 Enterprise Dual

The first UAV available for experiments is the DJI Mavic 2 Enterprise Dual (M2ED, see: Figure 5.2) [251]. It is equipped with a visible light and a 640 × 360 pixel thermal camera. The thermal camera makes it possible to perform a variety of additional inspection types with this UAV. The thermal camera can be used to detect faults in electrical and electronic components while in operation by detecting anomalies in the circuit's temperature caused by short circuits or high resistance through loose electrical connections. Additionally, faulty mechanical components can be detected when an increase in friction causes increased heating of mechanical components. Relevant for CS2 (see: Section 1.2.2) is the ability to automatically detect defects like cracks, delaminations, or dents using a combination of visual-light and thermal cameras [186].

The visible light camera has an 85°field of view (equivalent to a focal length of 24 mm on a full-frame camera) and can capture images with a resolution of up to 12 megapixels and record 4K videos. However, the resolution of the live image that is streamed to the remote control is limited to 720p. Both cameras are mounted to a single 3-axis gimbal, making it possible to independently stabilize or lock all three axes to a defined angle either relative to the horizon (for pitch and roll) or the UAV (for yaw).

The UAV is also equipped with an *extended port*, which allows the operator to mount accessories to the aircraft. These could be a speaker, a beacon light for flights at night, or an RTK module for increased positional accuracy. Especially relevant for inspections is the spotlight module that enables the UAV to illuminate dark areas when doing an inspection inside a structure (see: Figure 5.3).

The dimensions of the UAV are 445 (length) x 525 (width) x 92 (height) millimeters. This makes it smaller than most other UAVs used for inspections. However, the smaller size comes at the cost of a smaller payload capacity, no modular sensors (except for the RTK-module), and no 2D-LiDAR for obstacle avoidance. The UAV has an advertised flight time of 31 minutes, enabling long inspection flights. The hover accuracy without a GPS-fix is advertised as 0.1 m (in z-direction) and 0.3 m (in x- and y-direction). The obstacle avoidance system is proprietary but presumably works using a combination of stereoscopic cameras as well as ultrasonic and laser distance sensors. It converts the data of these sensors to 12 different distance measurements (four sectors each in the front and rear, one each on the top, bottom left and right), with each sector covering a relatively large area. Unfortunately, the UAV is not fully supported by the DJI Mobile SDK V4, and parts of the documented features are unavailable. The already very basic obstacle detection is not fully supported by the SDK, with only the four front and rear sensors working properly.

Figure 5.2. Image of DJI Mavic 2 Enterprise Dual.

Figure 5.3. Headlight attachment illuminating inside of plane fuselage.

DJI Mavic 3 Enterprise

The DJI Mavic 3 Enterprise (M3E see: Figure 5.4) [252] is the successor to the Mavic 2 Series. It is equipped with a dual camera (one wide angle and one with a zoom lens) mounted on a three-axis gimbal that works like on the Mavic 2 Enterprise Dual. Compared to the M2ED, the primary, wide-angle camera uses a larger 4/3 inch sensor, resulting in better image quality and an image resolution of 20 megapixels, while the resolution of the live camera stream is increased to 1080p. Like its predecessor, it is equipped with an extended port for mounting accessories and has a similar hover-precision when not using GPS. With dimensions of 488 (length) x 560 (width) x 105

(height) millimeters, it is also small enough to perform inspection tasks in relatively tight spaces. The maximum flight time is advertised as 45 minutes, making even longer inspections possible. The main advantages of the M3E over the M2ED are the improved support for the newer Mobile SDK V5 and the improved obstacle detection system. Better sensors on the UAV enable obstacle measurements in 360 individual sectors in 1°increments around the UAVs yaw axis as well as one measurement each for obstacles above and below the UAV. These measurements are fully accessible through the SDK and allow for better obstacle avoidance when navigating.

Figure 5.4. Image of DJI Mavic 3 Enterprise.

QuAD Connect App

The QuAD Connect App was developed as an interface between the ROS environment and the DJI UAVs. It is executed on the respective remote controller of each UAV. The controllers internally run an Android operating system, so the program is implemented as an Android App. It translates MAVLink packages sent by ROS to the proprietary radio protocol of the DJI UAVs using the DJI Mobile SDK and returns telemetry data sent by the UAV to ROS. The MAVLink interface was realized using the `io.dronefleet.mavlink` library, and the controller sends the MAVLink traffic over Wi-Fi over a UDP socket. The App works as an abstraction layer, thus maintaining compatibility of the system with other MAVLink-compatible UAVs.

Apart from working as an adapter between the two protocols, the App provides several other features. A configuration page is available for setting up the live camera stream,

the state of the aircraft and its onboard sensors can be checked, and several advanced control options and settings are available in the App. This includes recording pictures and videos, controlling accessories like the spotlight, turning the internal obstacle avoidance on or off, or performing high-level flight maneuvers like takeoff and landing. Of course, the UAV can also be controlled manually using the gimbals of the remote control. This feature is essential when the obstacle watchdog triggers an emergency hover and the UAV must be manually maneuvered away from an obstacle. Also, debug output in form of log messages can be viewed to diagnose communication problems. Figure 5.5 shows an exemplary screenshot of the App's camera screen. The upper status bar (1) is visible on all screens. It shows important information about the UAVs' battery life, mission state, and connections to ROS, the `copter_manager`-node, and the Ground Control Station. The main screen shows the live camera image taken by the UAV and provides buttons to start a video recording (3), take an image (4), and start a video stream to Vuforia tracking and possible inspection tools. The stream URL can either be manually set (5), or a preset can be selected (6). The status of the recording (2) and stream (8) is also displayed.

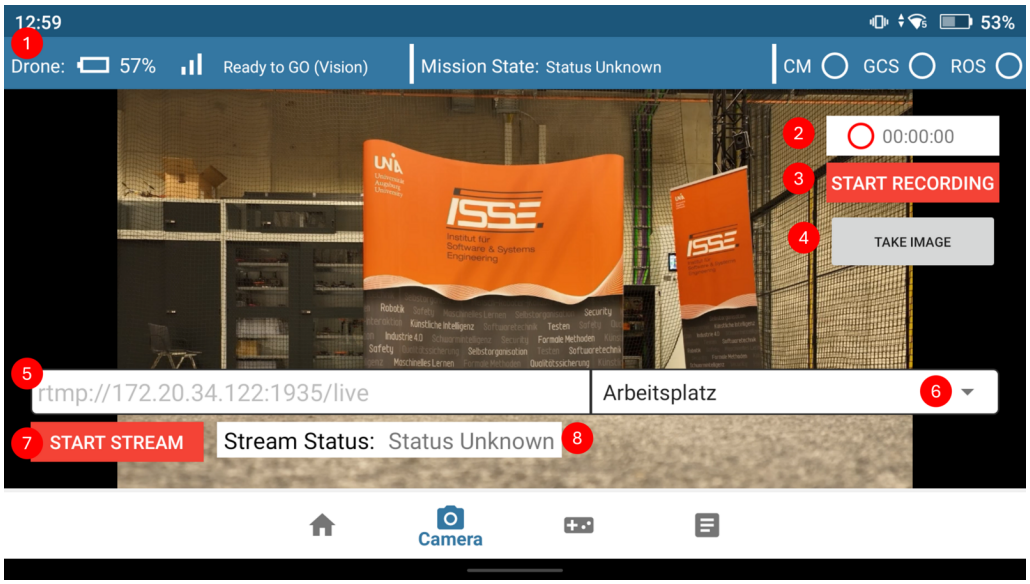


Figure 5.5. Live camera view of QuAD Connect App.

While the QuAD Connect App offers options that allow for configuration of the system at a reasonably low level, during regular operation, an interaction between the remote and the operator is not intended. Most of the interactive features in the App are used for development purposes only.

5.1.3 Assemblies for Inspection Tests

Multiple objects were used as inspection targets to conduct the evaluations described in the following sections. The object that allows for the most realistic recreation of a scenario for case study 1 is part of the tail of an Airbus A320 aircraft (see: Figure 5.6). It is named *Section 19* internally and is located towards the back of the aircraft, right behind the bulkhead for the pressurized cabin. It houses mainly mechanical, hydraulic, and electrical components as well as plumbing for the rudder and elevator. Finally, it contains the structural mounts for the horizontal and vertical stabilizers. During production at Premium Aerotec, numerous brackets are fitted and riveted to the main body of the part and subsequently inspected. This makes it ideal for performing integration tests of the whole system once working as well as for tests where a larger structure is needed. As a side note: The Section 19, which was provided to us for testing, was an actual production part that had to be scrapped due to a wrongly mounted bracket that was detected during inspection. This was further motivation demonstrating that inspection of such parts is essential and must be performed carefully. Premium Aerotec provided the assembly together with an assortment of carbon fiber brackets. In addition, 3D printed custom brackets were designed for some test scenarios. Both types of brackets can be freely positioned on the Section 19 using industrial double-sided tape to simulate different mounting scenarios for brackets that need to be inspected. The full CAD model of this assembly was also provided, allowing for the generation of precise tracking models in the Vuforia Tracking Software. Unfortunately, though, due to legal constraints, Vuforia's AI-enhanced "Advanced Model Targets" could not be used because this would require sending the CAD data to the Vuforia cloud.

The second part that was used for experiments was internally named *the door* because it resembles the shape of an aircraft door (see: Figure 5.7). It is, however, a cutout section of the carbon fiber outer hull of an Airbus A350 aircraft. It features a cutout for a window and multiple stringers that are incorporated into the carbon fiber during construction. Furthermore, it is still quite large with a height of about 1 m but in contrast to the A320 Section 19, it gives the UAV more freedom to manouver around it, because the space between the A320 Section 19 and the outer bounds of the flight arena is quite limited due to its size. This makes it ideal for smaller proof of concept for tasks like testing the coverage algorithm in hardware. Unfortunately, due to the part being a free hand cutout out of a larger part that was to be scrapped, there is no exact CAD model available, which makes the part less suited for tasks like the evaluation of visual tracking or inspection-tools.

Parts of the work for this thesis was done during the Covid-19 pandemic. This resulted in limited access to laboratories for testing. In order to be able to evaluate the performance of various visual tracking and inspection tools without access to a huge lab, a small, 3D-printable assembly was developed titled the Homeoffice-Fuselage (see: Figure 5.8). It resembles the larger, door-shaped carbon fiber part mentioned above, but CAD models are available for all parts due to its in-house construction. Also, there are no constraints caused by the intellectual property of the model, as with the A320 Section 19. This allows for most of the tests needed to evaluate tracking and inspection tools to be performed

with this model. It is also small enough to be printed by most 3D printers and fits on a regular desk when working in a home office. An additional feature that was added to facilitate testing was the integration of magnetic mounts for various 3D-printed brackets. These are also equipped with magnets and snap onto the main fuselage. The rotation is limited to 4 fixed directions by the shape of the socket on the main body. This allows for the repeatable placement of the brackets and is an easy way to reconfigure the assembly quickly for tests.

Figure 5.6. Partially assembled "Section 19" of an Airbus A320 Aircraft used for testing.

Figure 5.7. Cutout of center section of an Airbus A350 Aircraft.

Figure 5.8. 3D-printed mockup of an aircraft fuselage part with reconfigurable mounting brackets.

5.1.4 Simulation Environment

Without an adequate simulation environment, developing and testing most of the system described in Chapter 3 would require constant access to a laboratory where test flights can be performed. A great deal of testing is required during development, especially when developing and testing navigation and obstacle-avoidance algorithms. This can cause delays due to limited physical resources like the number of UAVs, access to the flight arena, etc. A simulation environment was developed to remove the requirement for physical testing. The system simulates an entire inspection flight and allows most of the development and testing phase to be performed in simulation. Simulation of only parts of the system is also possible. This allows, for example, the simulation of the component inspection while the rest of the flight is performed in hardware. Figure 5.9 shows the developed nodes that replace the physical UAV and their interfaces. In addition to replacing the physical copter manager with a simulation using the tool rotorS [94], the distance sensors, model tracking, and IMU data and inspection component are also simulated. The overall strategy when developing simulated replacement nodes for the system was to use the messages, services, and actions the component either uses as interfaces to develop against. The simulated components, therefore, offer and use the same services and actions and publish or receive messages from the same topics as the actual implementation. This allows for arbitrary components to be replaced by

their simulated counterparts. It is also possible to use the tool rosbag [89] provided by ROS to play back recorded messages from real or simulated flights that can be used in conjunction with node implementations of both simulation and reality. The

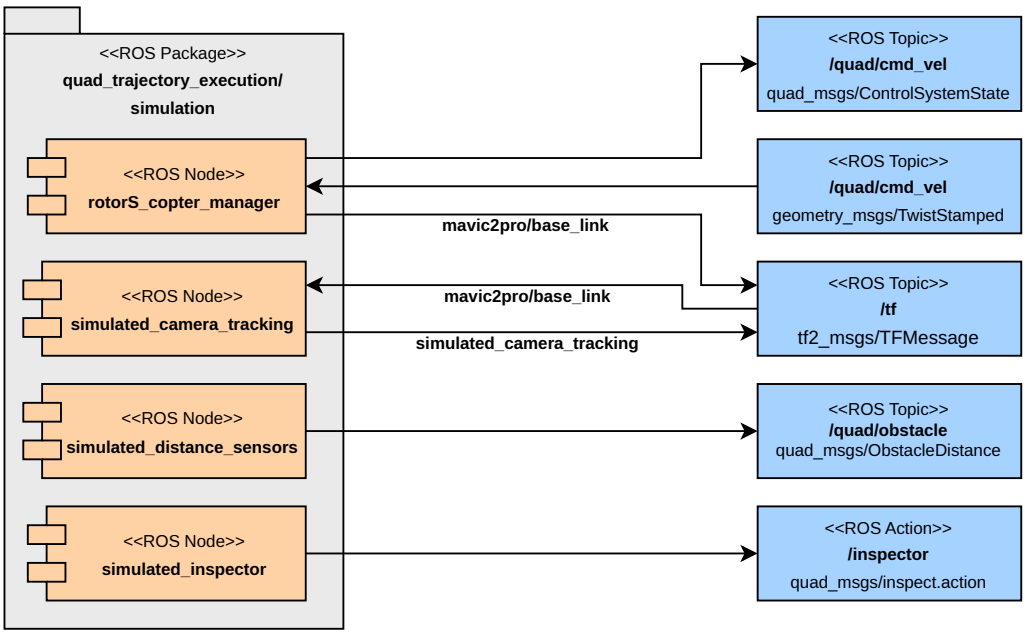


Figure 5.9. Implemented Nodes that replace the real-world UAV for simulation.

flight dynamics of the UAV are simulated in the `rotorS_copter_manager`. It works in conjunction with a simulated UAV set up in the rotorS simulator for Gazebo [86]. The node works as a driver to adapt the interface rotorS offers to the services, actions, and topics required by the rest of the developed system. In rotorS, the simulated UAV was configured, so its flight characteristics match these of the used DJI Mavic UAVs as closely as possible.

Simulated Obstacle Sensors

Developing and testing navigation algorithms and obstacle avoidance strategies require the simulated system to detect obstacles. This part of the system especially benefits from a development in simulation as the multitude of crashes caused by bugs in the obstacle avoidance have significantly less impact in simulation than in a real-world test. RotorS supports the simulation of obstacle sensors that behave similarly to a LiDAR sensor, reporting distance measurements at a precise angle relative to the UAV. Unfortunately, measurements of this sensor correspond to a single point, which does not match the behavior of the obstacle sensor values reported by the M2ED and M3E. Instead, these report the obstacle values in wider sectors and the distance to the closest obstacle in the sector. Therefore, a separate node was implemented to emulate this behavior using the available LiDAR or `laser` sensors. This is done by defining a separate `laser` sensor for each obstacle sector of the respective UAV. The sensor is configured to cover the

same horizontal and vertical angle as the corresponding sector and provide distance measurements in 1°increments in both axes.

The `simulated_distance_sensors` node then processes this raw obstacle data. The node extracts the minimum measurement of each sector and publishes it as the final obstacle message in the format required by the rest of the system. Using this strategy, a good approximation of the behavior of the obstacle sensors of the real UAV can be achieved. Obstacles can be loaded into the simulation in form of meshes representing the objects. This allows building virtual environments for the UAV to navigate through using the simulated sensors. Figure 5.10 shows an example of such an environment. The blue dots on the box-shaped obstacle represent sensor measurements of the simulated obstacle sensors. Obstacle measurements outside the detection range of the UAV are not visualized in the simulation. The obstacle sensors can detect the added box-shaped obstacle, the model of the assembly on the left, and the ground plane.

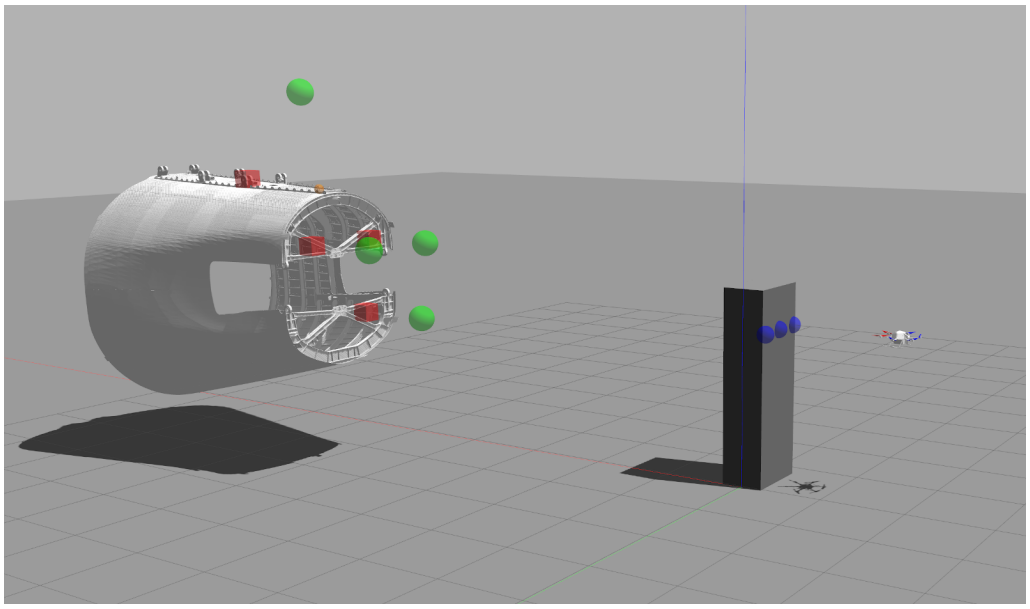


Figure 5.10. Screenshot of the simulated flight in Gazebo.

Simulated IMU and Camera Tracking

To evaluate and test the sensor fusion algorithms in simulation, nodes were created that emulate the behavior of both systems. The simulated IMU emulates the drift introduced by the numeric integration performed by the IMU. To generate a message of the type `quad_msgs/ControlSystemState` as with the `mavlink_copter_manager` (see: Figure 4.5) the `rotorS_copter_manager` uses the transformation of the UAV base link provided by rotorS to calculate the required values for the message. The position and orientation are directly calculated using this transformation. However, they are transformed to use the starting position of the UAV as a reference frame as with the real UAV. Velocities

and accelerations are calculated by numeric differentiation using the current and last position or velocity and the time interval between the two measurements. Rates for roll, pitch, and yaw are calculated the same way. To emulate the error introduced through numeric integration, a small, random error is added to the position and linear velocity values to simulate the drift introduced by the real system. Since the real UAVs use their compass and the gravity vector measured by their accelerometer in their internal sensor fusion to compensate drift in the orientation and rate measurement, these values are left as is. Lastly, the finished values are published as a `quad_msgs/ControlSystemState` message with the same frequency as the real message.

The simulation of the model tracking is done by adding jumps (positional and rotational offsets) at random time intervals to emulate a wrongly matched model. The transformation from assembly to camera frame $\mathbf{T}_a^c$ that is normally provided by the tracking can easily be looked up in the transformation tree of the tf2 package. Since both the position of the UAV and the assembly are defined relative to a global `/world` frame, it can be calculated using Equation (5.1).

$$\mathbf{T}_a^c = \mathbf{T}_w^a \cdot \mathbf{T}_u^w \cdot \mathbf{T}_c^u \tag{5.1}$$

With:

- $\mathbf{T}_w^a$ begin the transformation of the assembly in the world frame, which is manually published to define its position in the world.
- $\mathbf{T}_u^w$ being the inverse of the simulated UAV position in the world frame (published by rotorS).
- $\mathbf{T}_c^u$ being the transformation from the camera gimbal to the UAV base.

Offsets of varying amplitude are added to offset and rotation of the transformation at random time intervals to simulate the effect of jumps before the transformation is published into `/tf` to be used by the position estimation system.

Inspection

The inspection component can also be replaced by a simulated counterpart. In the evaluation of the path optimization across inspections, the inspection process was simulated using the `simulated_inspector` node. This allowed for repeatable timings of the inspection results and, therefore, identical conditions throughout multiple flights. The node implements an action server for the `quad_msgs/inspection.action` through which it receives the ID of the current viewpoint and publishes the inspection progress and a final result. The results and timings of the inspections can be predefined using a JSON file and are automatically published at the appropriate time, emulating the result of an inspection software.

Alternative Position Measurement Using Vicon Tracking

While not a simulation, the position measurement of the model tracking systems can be replaced using the Vicon motion capturing system installed in the flight arena.

Even if it is not intended to be used in the final system, using Vicon for position measurement is more precise and eliminates potential problems with the model tracking when implementing or testing different parts of the system. A pattern of five reflective markers was created on both the M2ED and M3D to achieve good tracking results. The measurement of the model tracking is performed relative to the `gimbal_frame` of the UAV. Therefore, the markers would have to be mounted to the camera at the end of the gimbal of the UAV. Since the camera does not offer enough space to mount the markers, they are placed on the UAV body instead, allowing measurement of the position of the `base_frame` of the UAV. This requires using the transformation from `base_link` to `gimbal_frame` to arrive at an equivalent measurement of the model tracking. Also, the coordinates of the UAV are measured in the reference frame of the Vicon system, which is placed arbitrarily in the arena. Through equipping the assembly with markers and tracking it as well to receive $\mathbf{T}_{vo}^{a}$, the transformation between camera and assembly can be determined using Equation (5.2).

$$\mathbf{T}_{a}^{c} = (\mathbf{T}_{vo}^{a})^{T} \cdot \mathbf{T}_{vo}^{u} \cdot \mathbf{T}_{u}^{c} \tag{5.2}$$

With:

- $\mathbf{T}_{a}^{c}$ being the transformation from assembly to camera frame.
- $\mathbf{T}_{u}^{c}$ being the transformation between camera and UAV.
- $\mathbf{T}_{vo}^{u}$ being the Vicon position of the UAV base.

5.2 Inspection of Aircraft Fuselage

In order to evaluate the effectiveness of the developed system in CS1, several smaller evaluations and proofs of concept were performed. The evaluation of parts of the implementation was already presented in previous work [293], [246], [247] but is covered in full here for the sake of completeness. First, the performance of the planning algorithm and its optimization strategies were validated. Afterward, the accuracy of the position estimation of the UAV was evaluated along with several of the developed navigation strategies and the collision avoidance system. Lastly, the rudimentary inspection algorithm, as well as the photogrammetric reconstruction of assemblies, were tested.

5.2.1 Planning Algorithm Performance

The planning evaluation of the path planning algorithm was performed in three steps. First, an optimal parameterization for the Ant Colony System implementation was determined using the **Att48 TSP benchmark dataset** in order to compare the result to other implementations. In a second step, the optimization using the intersection of multiple Viewareas was evaluated using the example of inspecting an aircraft fuselage part. Lastly, the integration of knowledge acquired during an inspection into future paths was demonstrated using an inspection scenario where the initial Key Viewpoints were either unreachable or obstructions prohibited an inspection from the position of the initial Key Viewpoint. The following sections cover all three steps of the evaluation in detail.

Evaluation of the ACS Implementation

In order to compare the algorithm performance to the current state of the art, a reference dataset that is widely used in other work was used. The dataset used was the **Att48 TSP benchmark dataset** [85]. It contains a set of 48 US cities. The minimal length for a round trip visiting all cities is 33523. The algorithm was parameterized using the information from Pettersson and Lundell Johansson [213]. The number of ants m, for a given number of cities or Key Viewpoints N as well as the values for β and ρ are shown in Equation (5.3).

$$\beta = 5, \;\; \rho = 0.5, \;\; m = \lfloor 0.3 \cdot N \rfloor \tag{5.3}$$

The intent of the evaluation was to examine whether increasing the number of iterations from the originally proposed $\lfloor \frac{2000}{m} \rfloor$ by Pettersson and Lundell Johansson [213] would yield better results. To do this, the algorithm was executed with different iterations ranging from twice to 7 times the original amount of iterations (see: Table 5.1). The average distances calculated over 10 passes with each parameter set can be seen in Table 5.1, along with the required calculation time.

The best single result was a distance of **34023.35**, achieved with $\lfloor 2000/(m/7) \rfloor$ iterations. This is 577.36 km less than the best distance found by Chaudhari and Thakkar [38] (34600.71 km) on the Att48 record. Figure 5.11 shows the relation between execution time

and quality of solution found in this evaluation. Although the best distance was found with $\lfloor 2000/(m/7) \rfloor$ iterations, on average, using more than $\lfloor 2000/(m/2) \rfloor$ iterations did not yield better results.

Application to an Inspection Scenario

After a successful evaluation of the algorithm on the **Att48** dataset, a hypothetical inspection scenario was created to evaluate the performance on an inspection. The inspected part and its POIs can be seen in Figure 5.12. Overall, 49 POIs were specified on the assembly. From these POIs, 49 corresponding Viewareas and Key Viewpoints were calculated. The Key Viewpoints were used as input for the ACS-algorithm, which was executed for 10 iterations and produced a trajectory with a length of **66.2 m**. The resulting trajectory can be seen in Figure 5.14b.

Iterations	**∅ length [in km]**	**∅ time [in s.]**
2000 / (m / 2)	36483.416	84.262
2000 / (m / 3)	35937.116	124.487
2000 / (m / 4)	36261.336	173.377
2000 / (m / 5)	36031.142	217.897
2000 / (m / 6)	36061.551	247.518
2000 / (m / 7)	35929.504	293.229

Table 5.1. Comparison of the achieved lengths and execution times of the ACS algorithm with different number of iterations (Table source: Wanninger et al. [293]).

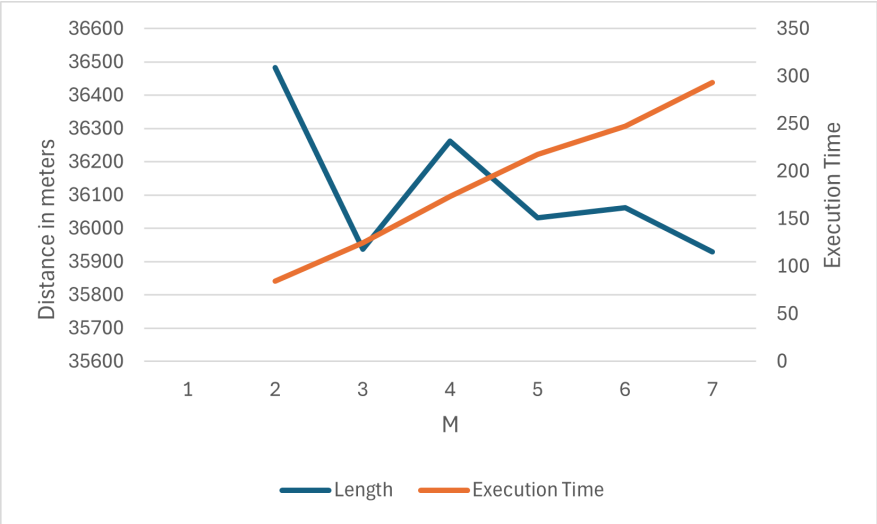


Figure 5.11. Execution time and average distance based on number of iterations.

5.2.2 Path Optimization Through Viewarea Intersection

After verification of the ACS implementation, the inspection scenario developed for Section 5.2.1 was used as a baseline for the optimization based on Viewarea Intersection presented in Section 3.2.3. This evaluation aims to determine how much the inspection path can be shortened using the VA intersection. After the calculation of VAs from the initial list of 49 POIs, the developed system intersects the resulting VAs. Afterward, the initial 49 Viewareas are reduced to 21 that meet the minimum size requirement. Figure 5.13 shows the difference between unoptimized (left) and optimized (right) Viewareas.

The Key Viewpoints of the resulting Viewareas are then used to plan an inspection path using the ACS implementation. This results in a trajectory length of **41.3 m**. Compared to the **66.2 m** of the unoptimized trajectory, meaning the optimization results in a **37.6 %** reduction of trajectory length. In addition, by reducing Viewareas, the UAV only needs to navigate to 21 Viewpoints instead of 49. This results in a **57 %** reduction in braking and acceleration maneuvers, not only speeding up the inspection but also conserving battery power. Figure 5.14 illustrates the optimized vs. unoptimized trajectory paths and clearly demonstrates how Viewpoints that lie in proximity to each other are combined into a new VP.

The evaluation demonstrates that the optimization through the intersection of VAs reduces trajectory length and UAV stops, decreasing total inspection time. It also simplifies the trajectory planning by reducing the problem space. This lowers the required computation time for path calculation.

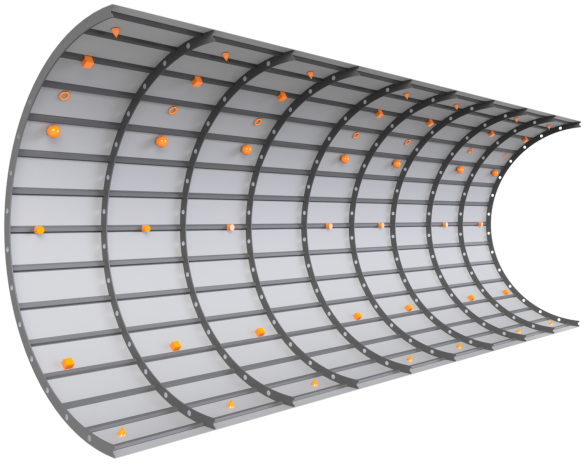


Figure 5.12. Model of the inspected assembly.

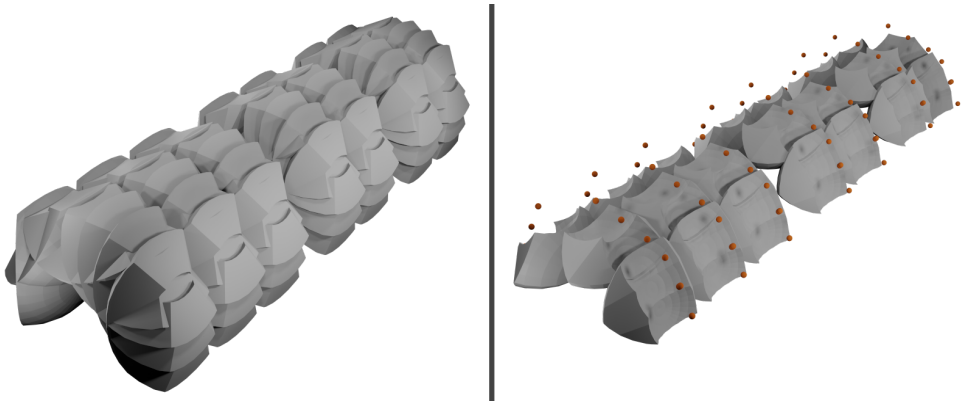
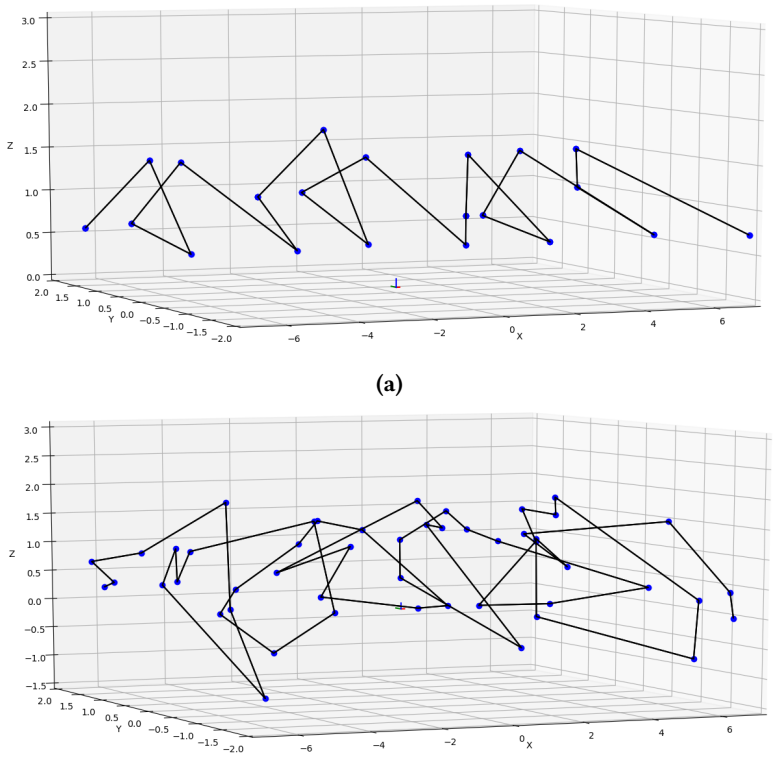


Figure 5.13. Result of the unoptimized (left) and optimized (right) Viewarea calculation. The optimization reduces the number of Viewareas from 49 to 21.



(a)

(b)

Figure 5.14. (a) optimized trajectory consisting of 21 Key Viewpoints with a length of 41.3 m and (b) unoptimized trajectory consisting of 49 Key Viewpoints with a length of 66.2 m. Image source: Schörner et al. [246].

5.2.3 Path Optimization Across Inspections

In order to evaluate the optimization across multiple similar inspections described in Section 3.2.4, a different inspection scenario was created. A structure with three different POIs (POI1-3) is used for the UAV to inspect (see: Figure 5.15). To test the performance of the cross-inspection-learning, an obstacle can be placed strategically so one of the original Key Viewpoints cannot be reached. It is also possible to prevent the simulated inspection from returning an inspection result for an arbitrary VP. This simulates an obstacle blocking the view of the POI from the point of the UAV. Using the described procedure, Viewareas and Viewpoints are determined based on the specified POIs, and an inspection path is calculated. The evaluation aims to prove the concept and evaluate the amount of time saved through the optimization. Using the simulation environment, a UAV performs the inspection using the calculated path in one of three scenarios.

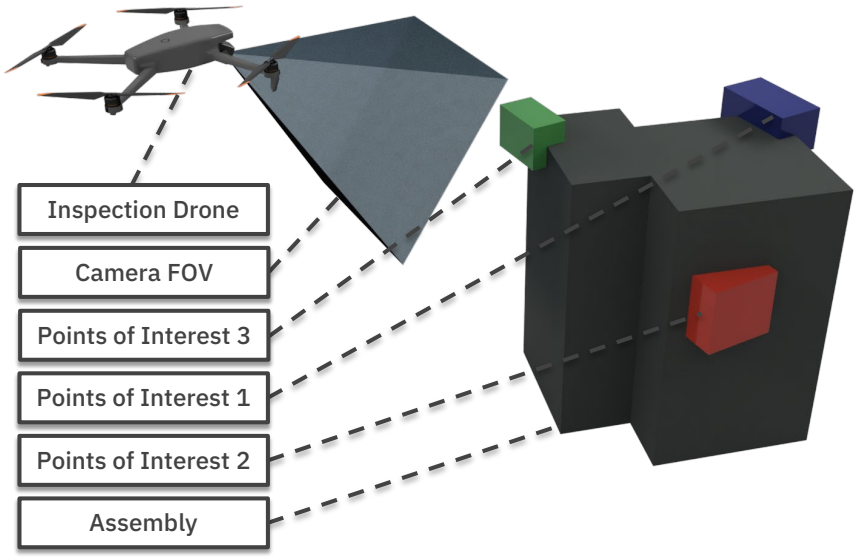


Figure 5.15. Render of the inspection scenario showing three POIs on an assembly and the inspection UAV.

Scenario 1

In Scenario 1, the view of the POI from one of the Key Viewpoints is blocked. This results in the inspection module not delivering an inspection result indefinitely. Figure 5.16 shows the path of the first flight performed using this obstacle configuration. The UAV cannot inspect POI1 from the initial Key Viewpoint. After 10 seconds, a timeout occurs, and an alternate VP is chosen to perform the inspection. Since the obstacle does not

block the view of POI1 from this alternate VP, the inspection of POI1 succeeds, and the inspection continues. In a second flight (see: Figure 5.17), the scoring system causes the previously successful alternate VP to be the new Key Viewpoint of POI1, preventing the unnecessary waiting time of the first flight. This results in a reduction of the inspection time from 110 s to 176 s and a reduction in traveled distance from 34.3 m to 31.7 m.

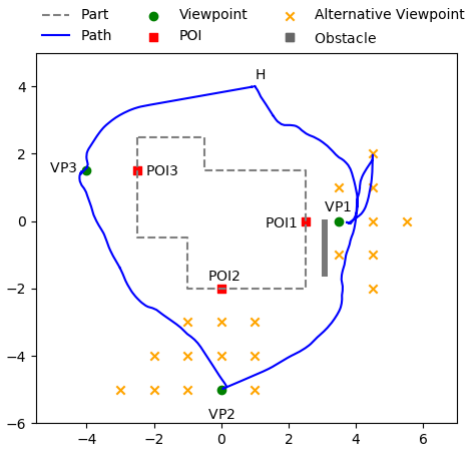


Figure 5.16. Trajectory of the first flight of scenario 1 in top-down view. POI1 is not visible from the initial Key Viewpoint, so a different VP is chosen. Image based on: Schörner et al. [247]

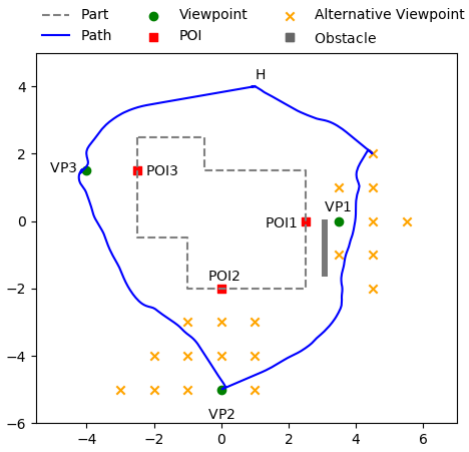


Figure 5.17. Trajectory of the second flight of scenario 1 in top-down view. The rating system elected the previously successful VP as new Key Viewpoint for POI1. Image based on: Schörner et al. [247]

Scenario 2

In the second scenario, an obstacle is strategically placed around the Key Viewpoint of POI2 to prevent the UAV from reaching the original Key Viewpoint. After a while, the system detects that it cannot reach the VP and chooses a different one from the alternative VPs. This one is reachable and the inspection of POI2 succeeds from this VP. The resulting trajectory can be seen in Figure 5.18. On the second flight, the system selects the previously successful alternative VP as new Key Viewpoint for POI2 based on the updated scores from the first flight. While this only results in a minimal reduction of trajectory length from 33.6 m to 32.6 m (which is also visible in the trajectory graphs), the inspection time is reduced significantly from 217 s to 158 s as the UAV does not unsuccessfully try to navigate to the original Viewpoint first.

Scenario 3

Scenario 3 is a combination of Scenarios 1 and 2 where the camera view from POI1's Key Viewpoint is obstructed, and an obstacle prevents the UAV from reaching the original Key Viewpoint of POI2. As in the previous scenarios, the UAV detects these problems, reacts by choosing alternative VPs from the same VA, and performs a successful inspection

from there (see: Figure 5.20). The scores of all involved VPs are updated accordingly, which results in the UAV selecting the previously successful VPs as Key Viewpoints for flight 2 (see: Figure 5.21). As before, this resulted in a reduced inspection distance and time for the second flight. The traveled distance is reduced from 36.8 m to 33.2 m while the duration of the inspection flight is shortened from 267 s to 159 s.

In summary, the evaluation demonstrates that the algorithm works as intended, replacing suboptimal Key Viewpoints with better candidates.

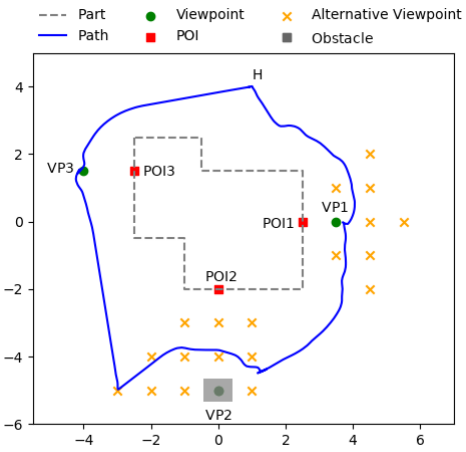


Figure 5.18. Trajectory of the first flight of scenario 2 in top-down view. The Key Viewpoint of POI2 is not reachable, so a different VP is chosen. Image based on: Schörner et al. [247]

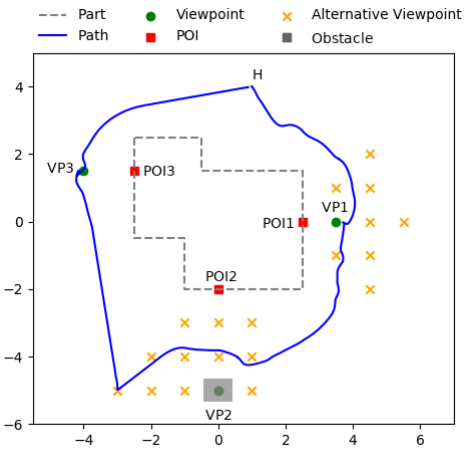


Figure 5.19. Trajectory of the second flight of scenario 2 in top-down view. The rating system elected the previously successful VP as new Key Viewpoint for POI2. Image based on: Schörner et al. [247]

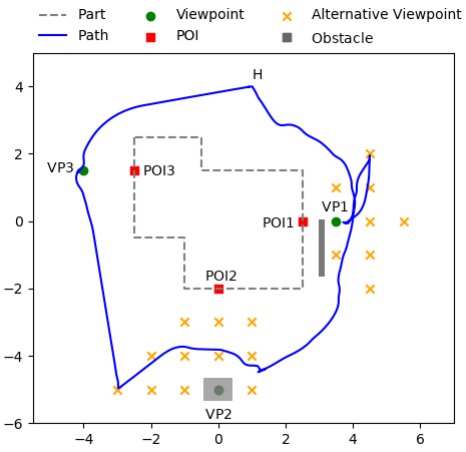


Figure 5.20. Trajectory of the first flight of scenario 3 in top-down view. The Key Viewpoint of POI2 is not reachable, and POI1 is not visible from the initial Key Viewpoint, so different VPs are chosen. Image based on: Schörner et al. [247]

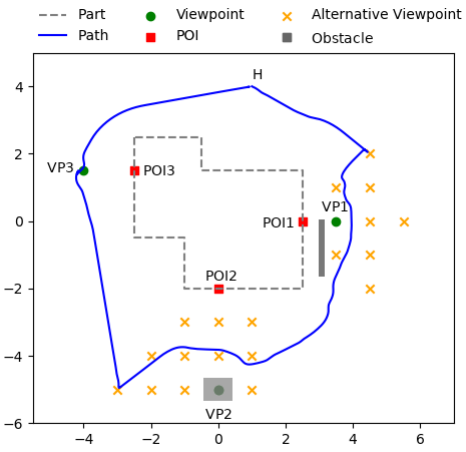


Figure 5.21. Trajectory of the second flight of scenario 3 in top-down view. The rating system elected the previously successful VPs as new Key Viewpoints for POI1 and POI2. Image based on: Schörner et al. [247]

5.2.4 Positioning Performance

The performance of the positioning was evaluated in two stages. In previous work [248], the functionality of the model tracking prefilter was demonstrated (see: Section 3.3.2), as well as the sensor fusion algorithm using simulated sensor data. This was done in conjunction with a proof of concept of the potential field navigation and demonstrated a promising position estimation performance. This section, however, focuses on evaluating the position estimation system in real-world flights. The focus is on evaluating the performance of the Kalman Filter and the filtering of the model tracking data. To evaluate these systems, an inspection path is generated that lets a DJI Mavic 3 Enterprise inspect two Points of Interest on the outside of the fuselage of the Section 19 (see: Section 5.1.3). The UAV is equipped with retroreflective markers, which are tracked in the Vicon motion capture system to provide ground truth information. Figure 5.22 shows the inspection's flight trajectory in a top-down view, while the altitude over time can be seen in Figure 5.23. Each graph contains the ground truth information provided by the Vicon system, the filtered model tracking data and the position estimate of the discrete Kalman Filter (DKF). The flights were performed with a maximum velocity of 0.3 $\frac{m}{s}$.

While it would be possible to navigate inside the Section 19, this would have made comparing the estimated position to a ground truth impossible as the retroreflective markers would not be visible to the Vicon cameras once the UAV enters the fuselage. The pose provided by the Vicon system is defined in the global `world` coordinate system and, therefore, must be transformed to the coordinate system of the assembly. This is done

by tracking the assembly and calculating its transformation to the `world` frame. This allows all Vicon measurements to be converted into the reference frame of the assembly, independent of its placement in the arena. It was discovered during testing that the quality and frequency of the provided velocities by the used UAVs is insufficient. The values, that can be obtained from the DJI Mavic 2 Enterprise Dual and Mavic 3 Enterprise via the used SDK have an update rate of 1Hz and quantized to 0.1 $\frac{m}{s}$ precision. Due to this fact, they did not contribute useful data for the sensor fusion process. Figure 5.24 illustrates the problem. The IMU measurements (red) significantly deviate from the actual velocities (yellow), while the commanded velocities (blue, but mostly hidden under the estimated velocity in pink) form a better representation of the ground truth as the UAV tracks the commanded velocity well. Therefore, the control input and process

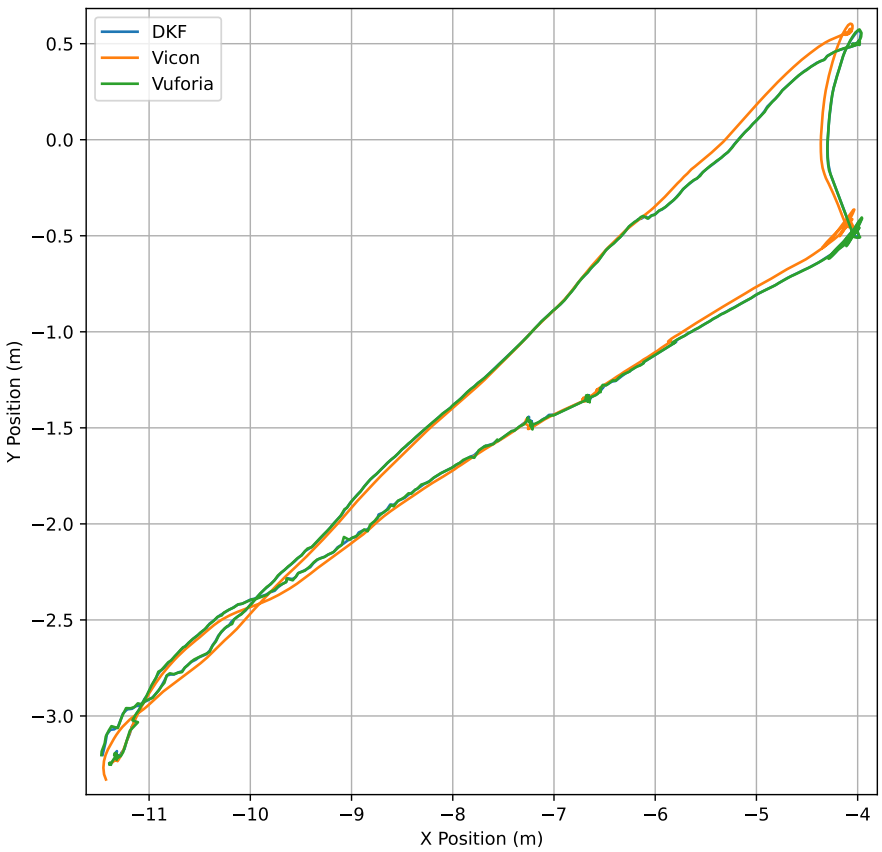


Figure 5.22. Flight path in a top-down view with the model tracking, DKF estimate, and Vicon position as a reference.

noise covariance matrix as well as the measurement noise covariance for the IMU update were manually adjusted, so the linear velocities of the state vector lean more towards tracking the control input. This lead to a reduced error in the linear velocities (see: Figure 5.24). Additionally, the pre-filter was effective in removing the majority of tracking-induced errors. Measurements were only passed on to the Kalman Filter, if they passed the pre-filtering step and the Kalman Filter operated on IMU-measurements, Control Input and predictions alone.

To quantify the quality of the position estimate, it was used to calculate the error using the ground truth data from the Vicon system. Figure 5.25 shows a section of the linear error of the filtered Vuforia model tracking and the estimated position by the Kalman Filter (see: Figure 5.26). This section of the flight was deliberately chosen as it contains a small jump in the tracking measurement, which is largely reduced by the Kalman Filter. Overall, the position estimate mostly followed the filtered model tracking data, but the quality of the data was sufficient for performing various inspection flights reliably. The accuracy of the data provided by Vuforia was mostly within 0.2 m of the ground truth. A reduction in quality was noticeable with the UAV extremely close to the assembly, but in an evaluation of various model tracking tools, Vuforia was still among the candidates that performed best in this scenario. The orientation data provided by the UAVs was found more suitable for its use in the sensor fusion process. Quality and Frequency were improved compared to the linear velocities. Therefore, resulting orientation estimate

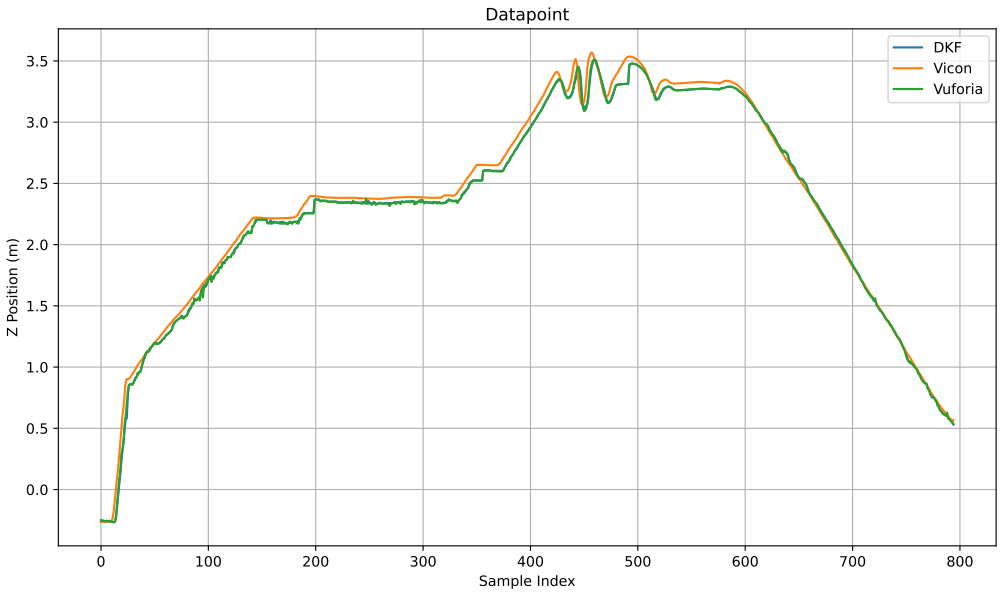


Figure 5.23. Drone altitude over time with the model tracking, DKF estimate, and Vicon position as a reference.
Drone altitude over time with the model tracking, DKF estimate, and Vicon position as a reference. One time step corresponds to 200 ms.

was highly robust against errors in tracking. Figure 5.26 shows the angular errors of the filtered model tracking data in a situation where tracking errors could not be filtered by the pre-filter. The erroneous measurements hardly influence the state estimate and the system quickly returns to the original state after the tracking measurements recover.

While the proposed model tracking approach might not yield sub-millimeter precision, its precision is comparable to other positioning systems for UAVs like GPS. Most inspections that would normally be performed manually using handheld devices, are designed to perform their task from a wide range of positions and orientations as human workers can not move the device with the same precision a robot would be able to. In cases where additional accuracy is strictly necessary, placing markers that are tracked by the UAV on the inspected object might be a possible solution, since the model tracking provides the primary position measurements and the error seems to originate from the tracking data and not from the Kalman Filter. Vuforia already offers a marker tracking system so integrating a marker-based position input is feasible.

The accuracy of the developed system allowed for reliable navigation during various inspection flights that were performed in the context of this dissertation. Also, the precision of the position estimate was sufficient to perform flights inside the Section 19 fuselage.

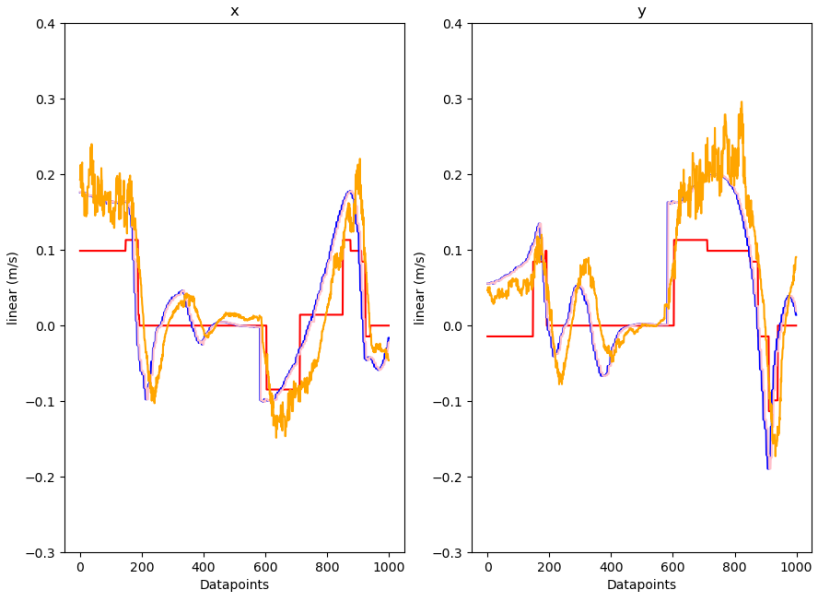


Figure 5.24. Comparison of recorded velocities of the IMU (red), control vector (blue), state estimate (pink) and ground truth (yellow) in the x and y direction. One time step corresponds to 50 ms.

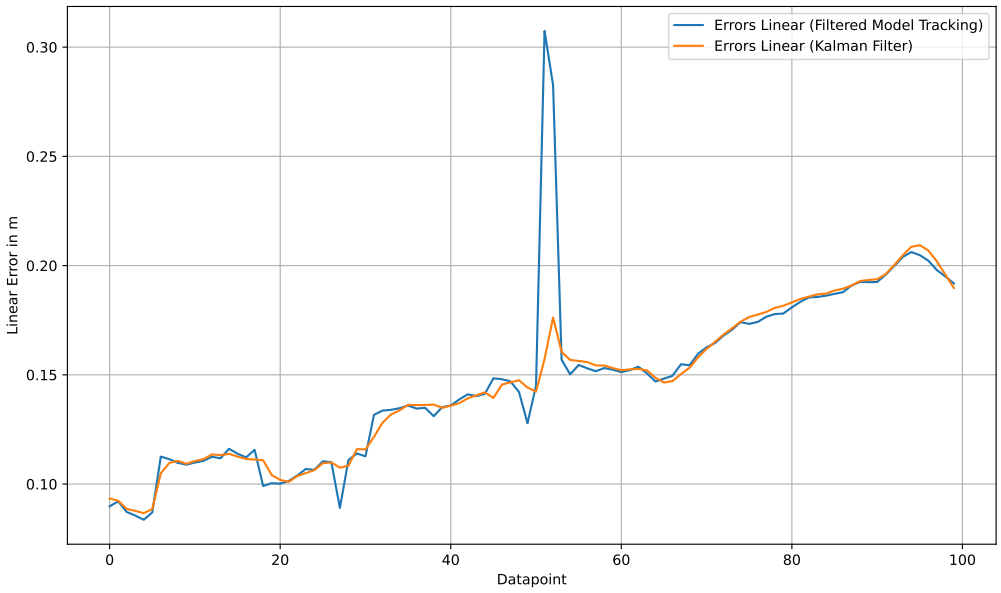


Figure 5.25. Linear error of model tracking and DKF estimate based on Vicon position as a reference. One time step corresponds to 200 ms.

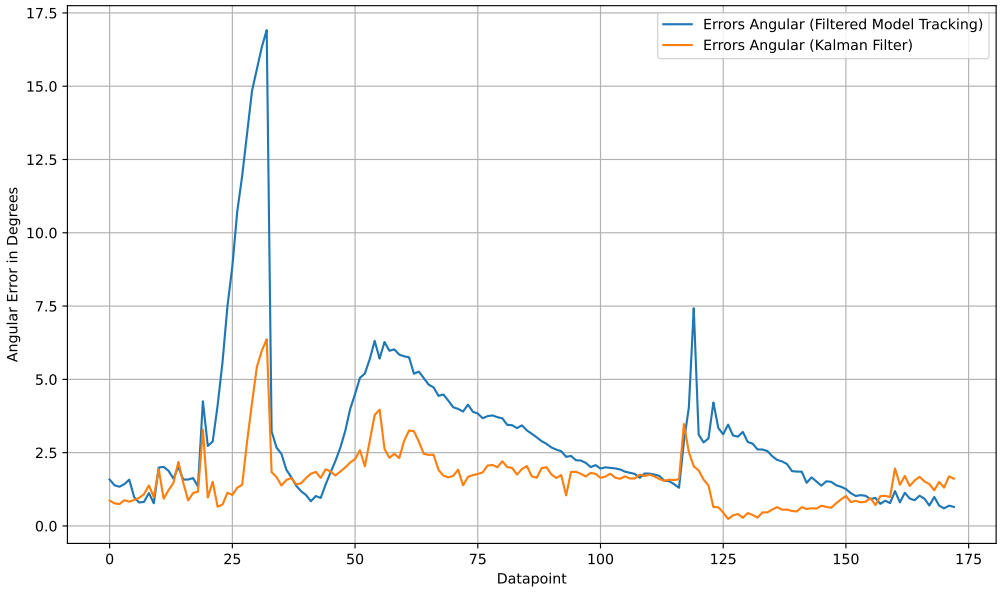


Figure 5.26. Angular error of model tracking and DKF estimate based on Vicon position as a reference. One time step corresponds to 200 ms.

5.2.5 Obstacle Avoiding Relative Navigation

The following paragraphs describe the evaluation of the obstacle avoiding navigation strategies implemented during this dissertation. This includes the implementation of the potential field navigation algorithm, which is tested in conjunction with the obstacle representation in form of test flights performed in several simulated and real-world environments. Additionally, the performance of the potential field method is compared to the Anytime Dynamic A* implementation using test flights in various simulated test environments.

Potential Field Navigation

The obstacle-avoiding navigation was performed in two stages. First, the obstacle avoidance system and world representation in OctoMaps were tested in simulation, and its performance was compared with an approach that uses only local sensor data[1]. This was done as an initial proof of concept and to demonstrate the system's feasibility. In a second step, the OctoMap-based approach was evaluated in a real-world scenario with both known and unknown obstacles that were both static and dynamic in nature. To be able to compare the experiments, both were performed using potential field navigation as the underlying navigation system. The following sections describe these two evaluations in detail.

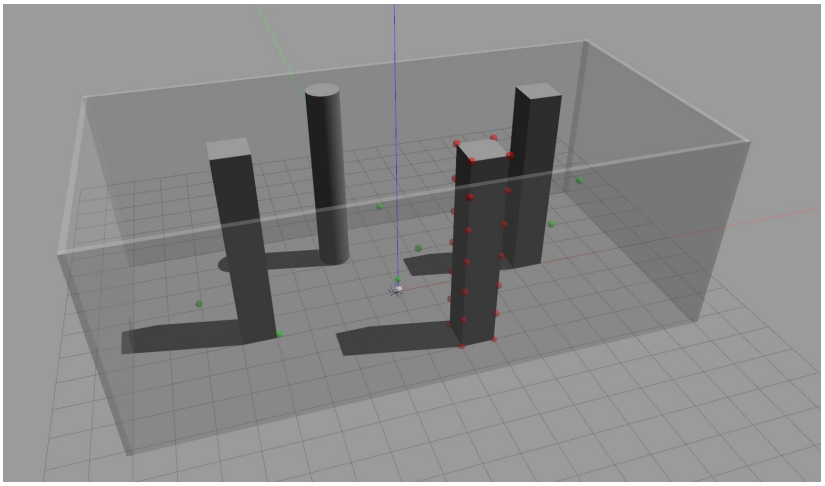
Evaluation of OctoMap Performance in Simulation

To prove the concept of obstacle-avoiding navigation using an obstacle representation in OctoMaps, the system was first evaluated in different simulated scenarios. This was done due to the ease of creating multiple different, but easily repeatable scenarios and virtual worlds for the UAV to fly through. Using the simulation environment described in Section 5.1.4, five environments were created that contain a combination of static (non-moving) obstacles or dynamic (moving) obstacles. Environments **Static 1** to 3 contained only static obstacles, while the environments **Dynamic 1** and 2 also contain moving obstacles. All experiments were conducted on an Ubuntu 20.04 PC with an Intel i5-3470 CPU and a GeForce GTX 1050 Ti GPU to be able to compare the execution times of the individual approaches.

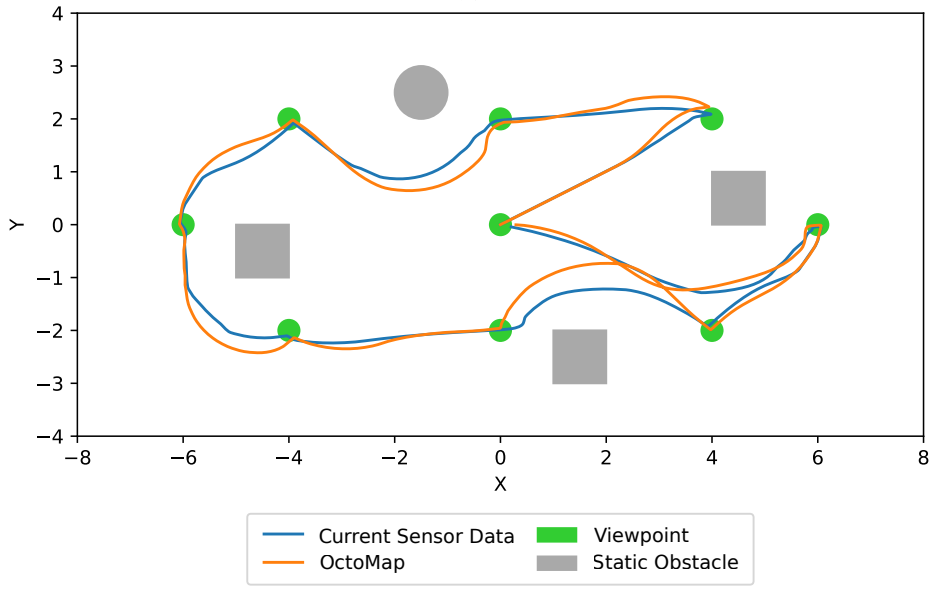
Performance in Static Environment Three different scenarios of **Static 1**, **Static 2**, and **Static 3** of varying obstacle complexity were used to evaluate the navigation performance in static environments. Scenario **Static 1** consists of an environment with only four pillar-shaped obstacles and the outer wall of the environment that was present in all scenarios. The location of one of the pillars is known in advance and is represented as several point-shaped obstacles in the OctoMap. Their locations can be identified by the red dots visible in Figure 5.27a. The inspection path and, thereby, the position of all Viewpoints is identical over all simulated inspection scenarios. The trajectories flown

[1] Many thanks to Raban Popall, who implemented and performed the evaluation of the obstacle avoidance system described in this section in simulation as part of his bachelor's thesis [216].

by both the OctoMap-based approach and the navigation using only currently detected sensor data can be seen in Figure 5.27b. The UAV performed this and all other flights in a counter-clockwise direction. The OctoMap-based approach seems to keep more distance to the obstacles, which can be explained through the additional repellant forces acting through the OctoMap, even when obstacles are outside the detection radius of the UAV. A more in-depth analysis of the flights and performances will follow later in this section.



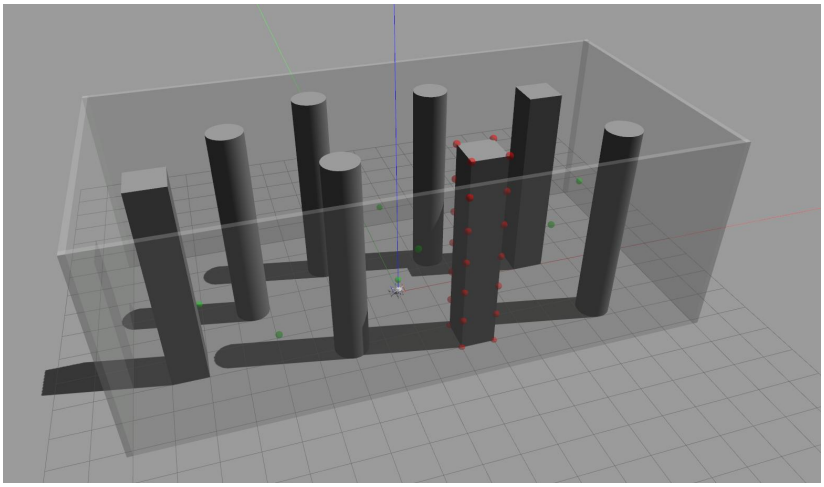
(a)



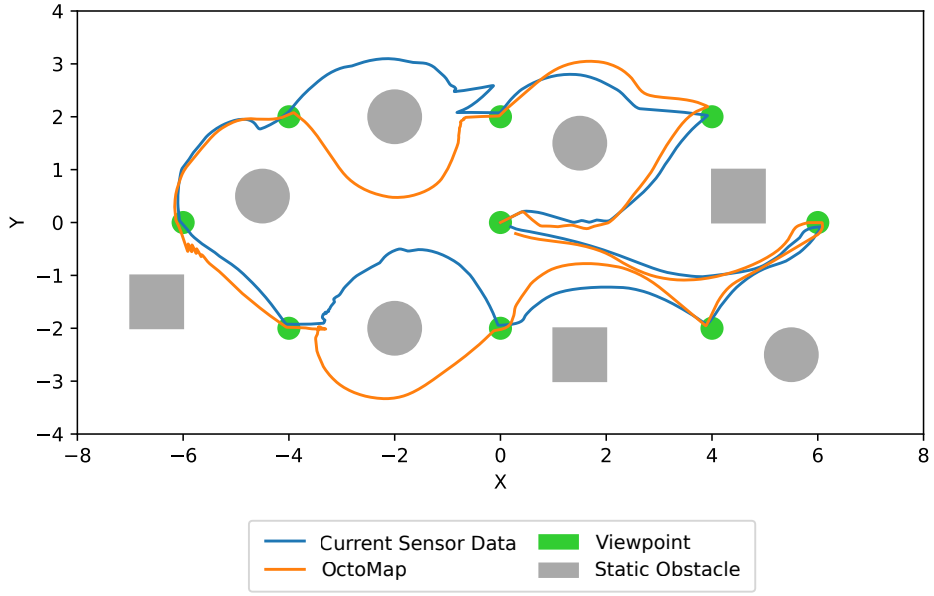
(b)

Figure 5.27. Render of simulated obstacle avoidance scenario **Static 1** (a) and flown trajectories (b). Image source: Raban Popall [216].

Scenario **Static 2** adds more complexity by increasing the number of pillars present in the environment (see: Figure 5.28). The increased obstacle density causes the paths of both algorithms to deviate from each other for the first time. A weakness of the local sensor data approach is also visible in Figure 5.28b after visiting the center Viewpoint located at the top of the image. The UAV first flies towards the pillar until the sensor detects an obstacle and is then quickly repelled by it until the obstacle is too far away to be detected, which causes it to approach again, resulting in a significant oscillation in



(a)



(b)

Figure 5.28. Render of simulated obstacle avoidance scenario **Static 2** (a) and flown trajectories (b). Image source: Raban Popall [216].

the trajectory. While this behavior can be observed in the OctoMap-based approach as well, the amplitude of the oscillations is reduced significantly.

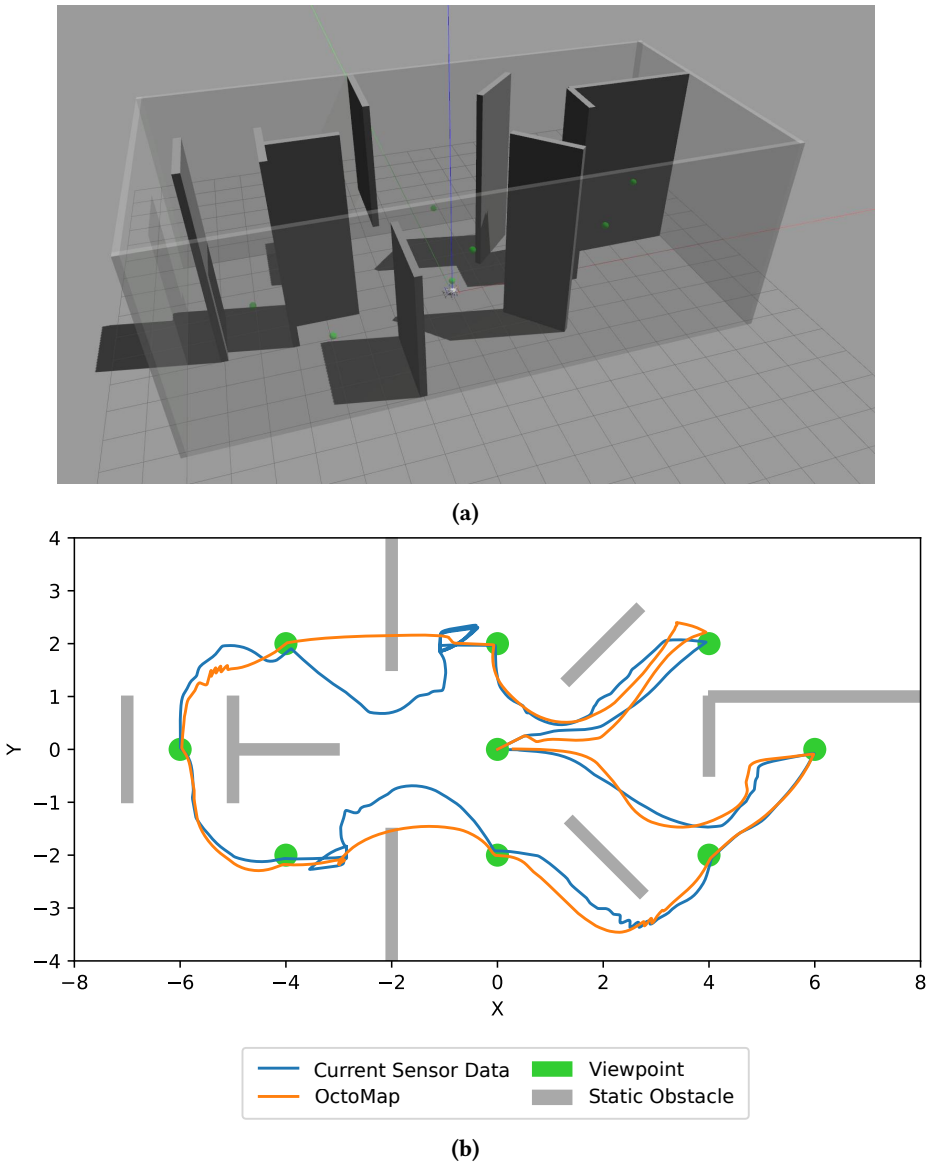


Figure 5.29. Render of simulated obstacle avoidance scenario **Static 3** (a) and flown trajectories (b). Image source: Raban Popall [216].

Scenario **Static 3** is built more like a traditional room with wall-shaped obstacles and introduces obstacles that do not span the entire height of the environment and, therefore, can be flown over or under (see: Figure 5.29a). The trajectories visible in Figure 5.29b show that the sensor-only navigation algorithm struggles with these new types of obstacles as it flies in circles in front of the obstacle before finally flying around it.

The OctoMap-based approach is able to navigate over or under the obstacles without significant detours.

This scenario was also used to evaluate the execution time of a single iteration of the navigation algorithm using both approaches. Since the navigation algorithm has to process an increasing number of detected obstacles in the OctoMap-based approach, an increase in execution time is expected. As visible in Figure 5.30, the execution time of the algorithm rises over time with an increasing amount of detected obstacles. The overall update rate of the system still stays below 3 ms with only a few outliers, probably caused by external interrupts slowing down execution as the algorithm was not run on a real-time system. Since the navigation node sends updates with a rate of 10 Hz, this leaves a time budget of 100 ms for calculating a new target velocity. An execution time of 3 ms is way within this limit and, therefore, acceptable.

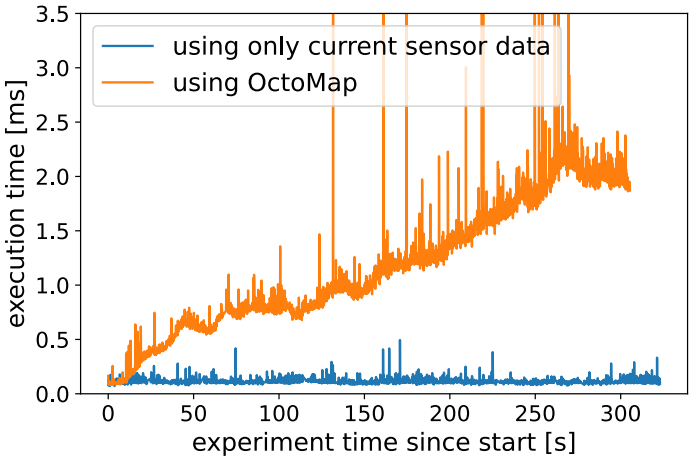
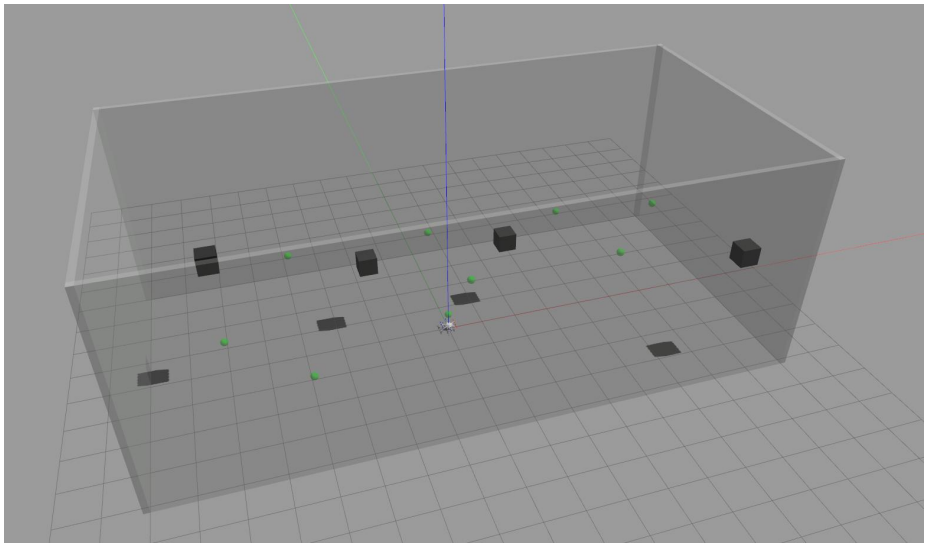


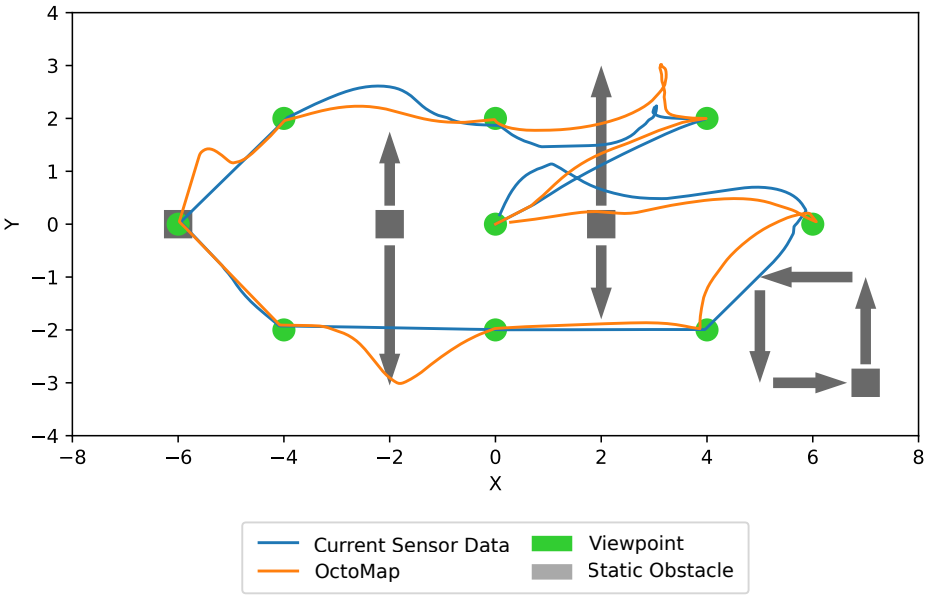
Figure 5.30. Execution time of the potential field algorithm in scenario **Static 3** for the sensor-only and OctoMap-based approach. Image source: Raban Popall [216].

Performance in Dynamic Environments Scenarios with dynamic or moving obstacles are especially challenging for obstacle avoidance algorithms that keep a record of detected obstacles. Obstacle measurements of a single time point persist in memory and are treated as permanent obstacles, which can lead to inconsistencies between the OctoMap representation and the real world.

To evaluate navigation performance with moving obstacles, two dynamic scenarios were created. In scenario **Dynamic 1**, several moving boxes represent dynamic obstacles in an otherwise empty environment. The corresponding environment and flown trajectories are displayed in Figure 5.31. Note that the obstacle on the left of Figure 5.31b is moving up and down. Except for minor differences caused by different timing in relation to the moving obstacles, no significant differences in the trajectories could be observed.



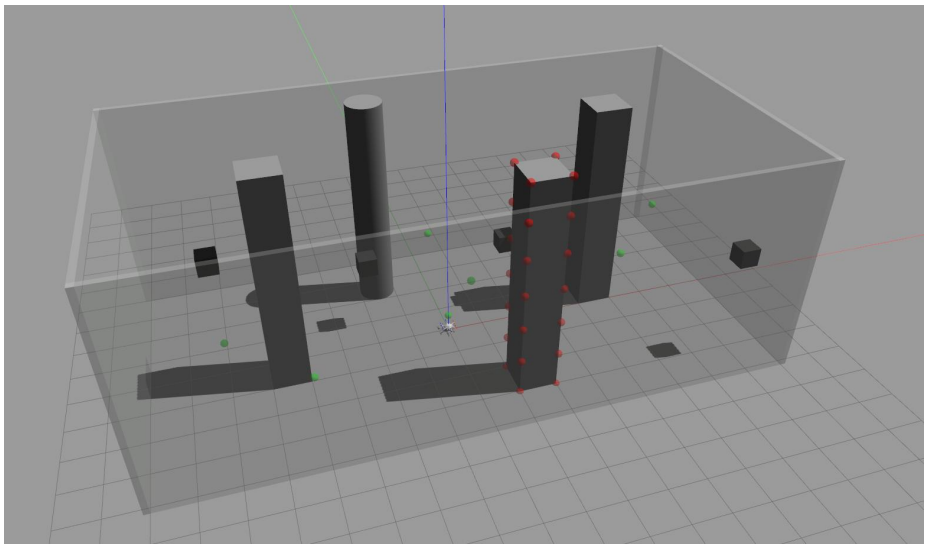
(a)



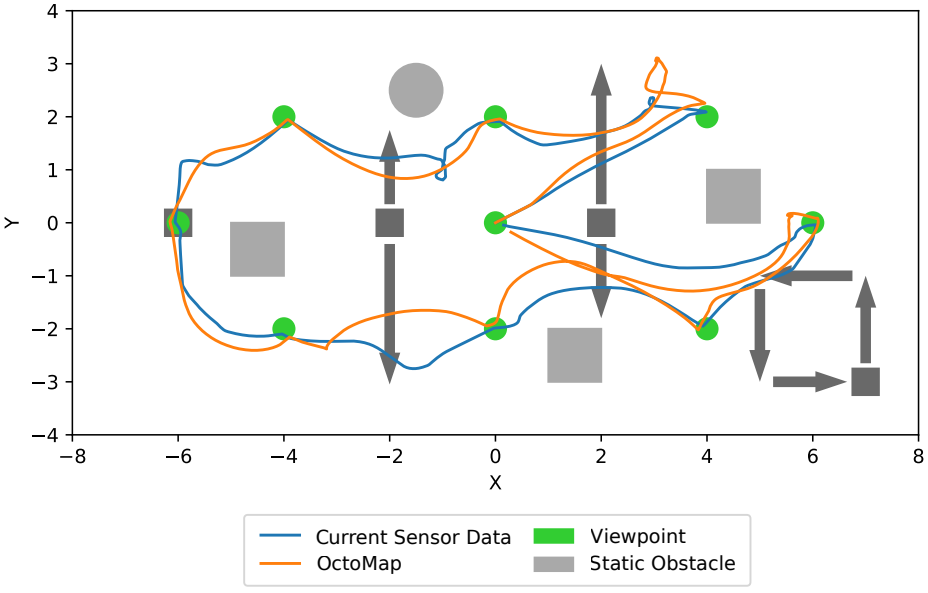
(b)

Figure 5.31. Render of simulated obstacle avoidance scenario **Dynamic 1** (a) and flown trajectories (b). Image source: Raban Popall [216].

The second dynamic scenario combines the obstacles from scenario **Static 1** and **Dynamic 1** to form a scenario with both static and dynamic obstacles with only one of the static pillars being known before execution. The flown trajectories are very similar again. Most of the differences were caused by timing differences when encountering dynamic obstacles.



(a)



(b)

Figure 5.32. Render of simulated obstacle avoidance scenario **Dynamic 1** (a) and flown trajectories (b). Image source: Raban Popall [216].

General Findings After reviewing the data, several conclusions can be drawn about the length and duration of the trajectory, as well as the computation time needed to execute the navigation algorithm. The individual trajectory lengths and flight are depicted in Figure 5.33. The overall trend seems to be a slightly longer trajectory and flight time for the OctoMap-based approach. However, the difference between both approaches is relatively small. While the current sensor data approach is computationally more efficient and performs better in dynamic environments, the OctoMap-based approach is useful in static environments. This is because it allows for more optimized long-term planning but at the cost of higher computation time. The strengths of the OctoMap-based approach lie in complex, static environments. This can be seen in scenario **Static 3**, where the OctoMap approach is able to navigate over and under obstacles that do not span the entire height of the room on the first try. The current sensor data approach performs multiple tries and ultimately still navigates around the side of the obstacle. This results in the OctoMap approach being able to perform the flight faster and with an almost identical trajectory length.

As expected, the OctoMap approach struggled with dynamically changing obstacles, as it relied on previously stored data, which sometimes became outdated. This led to slower reactions and occasional incorrect avoidance maneuvers. In general, the OctoMap approach was deemed promising for complex environments with only a few dynamic obstacles. Therefore, the system was evaluated in a real test flight.

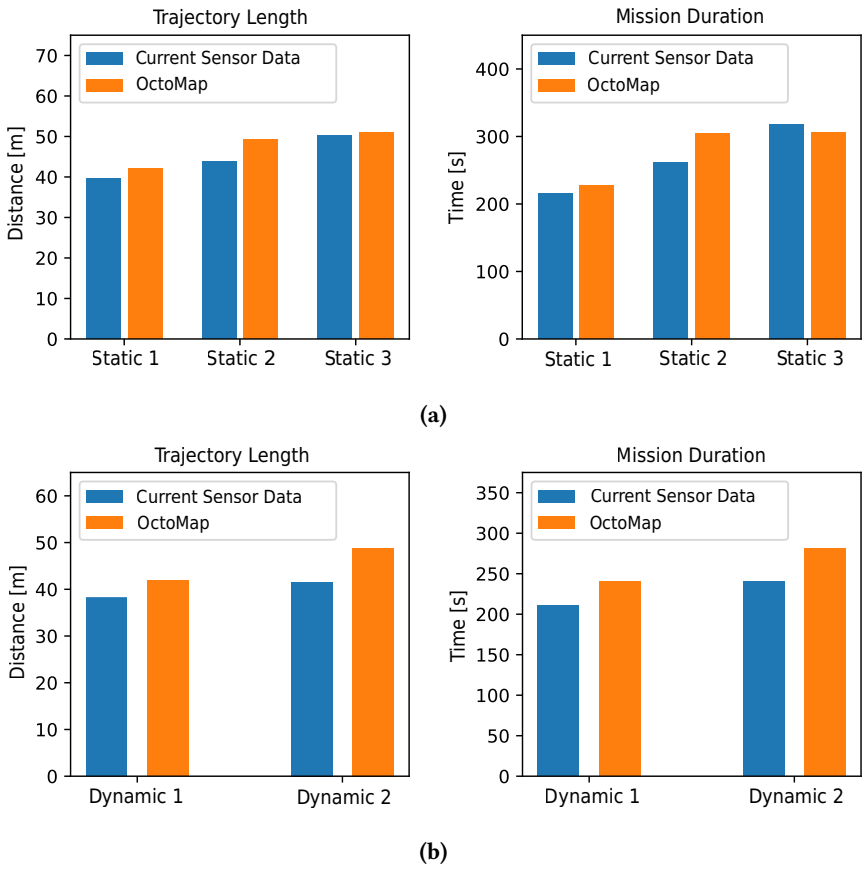


Figure 5.33. Lengths and durations of the flown trajectories of the static (a) and dynamic (b) evaluation scenario. Image source: Raban Popall [216].

Evaluation of Dynamic Obstacle Performance in Real World Scenario
To evaluate if the performance of the OctoMap-based obstacle avoidance system translates from simulation to reality, another experimental setup was created. The experiment's goals were to verify that the obstacle avoidance works with real-life hardware and that removing obstacle measurements that are no longer accurate is performed as intended. In this scenario, the UAV performs a test inspection on four points of interest on the outside of the Section 19 assembly. The DJI Mavic 2 Enterprise Dual (see: Section 5.1.2) was used for the experiment as it has worse obstacle avoidance sensors than the DJI Mavic 3 Enterprise, creating a "worst case scenario". The structure of the assembly was loaded into the OctoMap as a known obstacle. The starting point of the

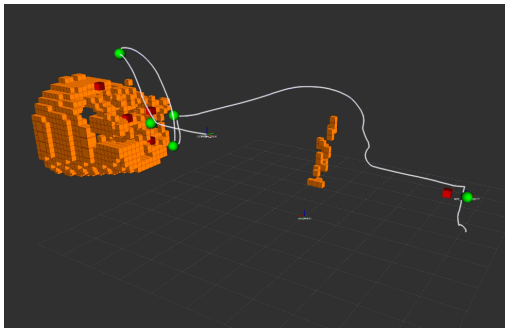


Figure 5.34. Obstacles detected on the way from start (right) to assembly (left).

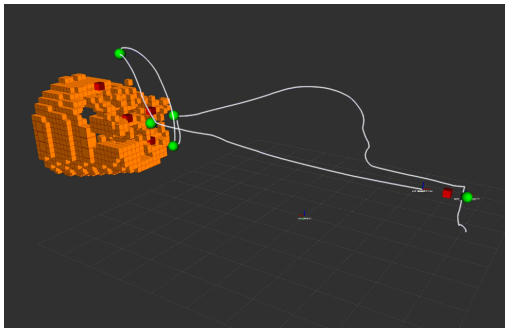


Figure 5.35. Cleared obstacles after returning from the inspection.

UAV is placed several meters from the assembly. To demonstrate the detection and removal of a dynamic obstacle, the *door-shaped* section of the Airbus A350 fuselage was placed on a tool cart and initially positioned between the starting point of the UAV and the assembly. After takeoff, the UAV successfully detected the cart placed in its way and navigated around the obstacle. The flown trajectory can be seen in Figure 5.34. It also visualizes the state of the OctoMap in orange with the detected obstacle clearly visible. It then continued with the inspection, reaching all Viewpoints and inspecting all Points of Interest while keeping clear of and updating the obstacle voxels of the Section 19 assembly. During the inspection, the tool cart was moved out of the way to determine whether the system would detect the absence of the obstacle and take the direct path back to its landing point. As intended, the obstacle sensors of the UAV recognized that the previously detected obstacles were no longer present and removed them before they could affect the trajectory planning (see: Figure 5.35). As a result, the UAV took the direct path back to its starting point.

In summary, the potential field implementation was able to navigate the real test scenario effectively while avoiding collisions. Preexisting obstacles were considered for trajectory planning and observed obstacles were added to the obstacle representation in the OctoMap. Upon removing the dynamic obstacle, the UAV detected its absence through the onboard obstacle sensors and removed the previously detected obstacles from the OctoMap. Overall, this approach has demonstrated its effectiveness for performing

inspections in indoor scenarios and its reliability is proven by the numerous development and demonstration flights that were performed using this system without crashes or other major incidents that could be attributed to the obstacle avoiding navigation.

Anytime Dynamic A*

The following paragraphs describe the evaluation of the Anytime Dynamic A* algorithm implementation. Due to performance problems encountered with the initial implementation written in python, the computational performance is compared to the subsequently developed, functionally identical C++ implementation. Additionally, the trajectory lengths and inspection times are compared to the potential field implementation in a variety of simulated scenarios. Lastly, the ability of both approaches to navigate past a u-shaped obstacle designed to induce the local-minimum problem in the potential field method is tested.

Computational Performance

ROS1 mainly supports C++ and python as programming languages for implementing nodes using their client libraries [81]. C++ code is compiled to binary code prior to execution, which allows for the implementation of high performance real-time or near real-time applications. The `rospy` python client library was created with a focus on implementation speed [81] rather than efficiency. The object-oriented, dynamically typed scripting language allows rapidly developing and debugging large systems without time-consuming compilation processes or complex memory management at the cost of slight inefficiency. All ROS nodes of the QuAD system were implemented in python as neither of the implemented algorithms required the additional performance gained through a C++ implementation. The only exception to this statement is the implementation of the Anytime Dynamic A* algorithm. Compared to most other nodes, it requires a great deal of computations to be performed and also has a large memory footprint due to the internally stored obstacle graph. This led to performance problems during the implementation and caused the final code to be ported to C++. The performance of both systems was evaluated using graphs comprised of 15000 to 32000 nodes. On average, the C++ implementation was able to achieve a 32.4 % time reduction per iteration of the algorithm. However, when testing the time required for replanning the path after an additional obstacle was added to the graph, no significant reduction of calculation times could be observed.

Comparison of Performance Against Potential Field Method

In a second set of simulated test flights, the performance of the AD* implementation was compared to the potential field method with trajectory length, covered distance and collision-free flight as evaluation criteria. For these test flights, the same virtual worlds were used as in the evaluation of the OctoMap-based obstacle avoidance, albeit with a different inspection path. Furthermore, a fourth world (**Static 4**), consisting of only a single pillar as an obstacle, was included in this test series. The inspection path was flown in all environments using the AD* and potential field implementation.

Environment	Anytime Dynamic A*			Potential Field Method		
	Duration	Distance	Result	Duration	Distance	Result
Dynamic 1	02:34.70	31.678 m	success	02:27.78	36.396 m	success
Dynamic 2	02:36.51	31.581 m	success	02:30.11	39.925 m	success
Static 1	02:44.10	31.895 m	success	02:31.70	27.386 m	success
Static 2	-	-	collision	03:43.94	31.962 m	success
Static 3	-	-	collision	-	-	collision
Static 4	02:27.62	31.322 m	success	03:10.80	30.680 m	success

Table 5.2. Results of the inspection flights comparing the APF and AD* algorithms. Table source: Julian Saupe [243].

Table 5.2 shows the result of these flights. The UAV was able to navigate through all scenarios except for **Static 2** and **Static 3** using the AD* implementation. In these cases, the emergency collision avoidance system was triggered as the UAV was unable to keep the required distance to the closest obstacle. On average, the resulting times for the execution of the trajectories are comparable. While the traveled distance was shorter for AD* in both dynamic environments, the overall differences in flight time and trajectory vary only slightly between both implementations. The high computational requirements of the algorithm, especially in large or complex environments are an additional disadvantage.

Behavior in Local Minima

The biggest weakness of artificial potential field method based algorithms is their susceptibility to getting stuck in local minima. To verify, whether the AD*-based navigation could solve this problem, several environments and inspection paths were created that deliberately forced the UAV into a situation, where it would encounter a local minimum in the potential field (see: Figure 5.36).

When encountering the obstacle using solely potential field navigation, the UAV would get stuck close to the obstacle in most cases. However, depending on the obstacle height and the location of the goal, it was slowly *pushed* over the obstruction due to the fact that a slight, self upward motion was introduced due to most obstacles being detected and stored beneath the UAV, resulting in an upward-pointing velocity vector. When adding the random-walk-based fallback strategy for local minima, the UAV was able to navigate around the obstacle reliably. However, the time this took was entirely dependent on the quality of the randomly generated fallback waypoints. Overall, this strategy worked but did not produce reliable or efficient results. The AD* implementation was able to perform the inspection in all scenarios containing the u-shaped obstacles. The resulting trajectories were still inefficient with the UAV performing patterns of alternating lines along the grid used as graph-representation internally, but the reactions were consistent and not based on the quality of a random guess.

In summary, the AD* implementation was able to navigate in the simulated test environments with its performance similar to the potential field algorithm in most cases. It proved slightly less robust in keeping the required safety distance to obstacles in the simulated tests but was able to navigate around a variety u-shaped obstacles without problem. The evaluation of both algorithms led to an approach that relies on the potential field method for regular navigation and uses an AD*-based fallback strategy when local minima are encountered. This also allows for a restriction of the planning space for the algorithm to the area between UAV and the target (with an appropriate boundary zone) and thus keep the execution time for the algorithm low. The implementation and evaluation of this approach are the subject of future work.

5.2.6 Performance of the Coverage Algorithm

After successful testing and validation of the obstacle avoidance, positioning, and trajectory execution subsystems, the trajectory planning for coverage scenarios could be tested. First, the performance improvements that were made to the HEDAC algorithm to support real-time feedback were tested in two different inspection scenarios. First, the planning performance was tested on the fuselage section of the Airbus A350 (see: Figure 5.7). Due to its low geometric complexity, the online planner was deemed most likely to perform fast enough for the closed-loop system on this part. Afterward, a performance analysis was performed with the Section 19 of an Airbus A320 (see: Figure 5.6). The second part of this section covers the proof of concept in the form of inspection flights on both assemblies and the results of the photogrammetric reconstruction of the acquired image data.

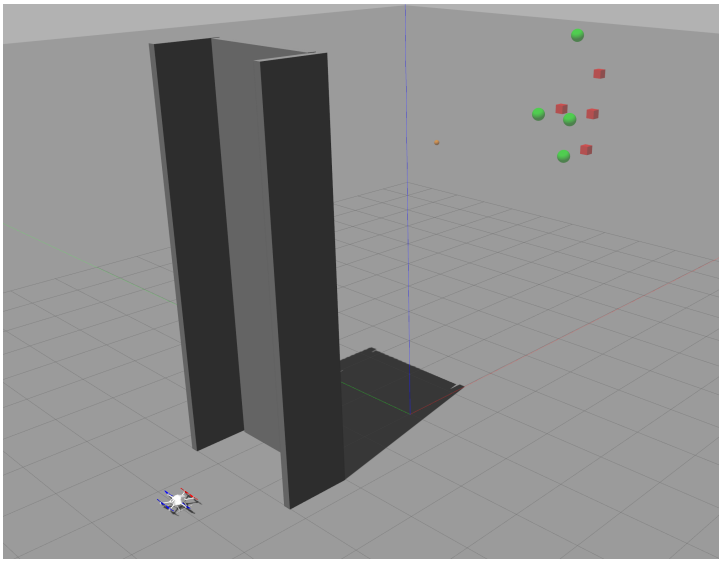


Figure 5.36. U-shaped obstacle located between UAV and Viewpoints.

Evaluation of the Real-Time Capabilities of the HEDAC Algorithm

In the paper presenting the algorithm [125], the authors state that its performance in the current state is too slow for real-time operation. However, removing the collision avoidance logic, simplifying the camera orientation calculation, and moving the surface coverage calculation to a separate thread greatly improves performance. This might make using the system in a closed-loop scenario viable. The following paragraphs will elaborate on the evaluation of the improved algorithm's performance planning trajectories for the Airbus A350 fuselage section and the A320 Section 19.

The algorithm was executed, and the average execution times of the individual parts of the calculations were determined. A MacBook Pro featuring an M3 Max processor was used to perform the calculations. The execution time per iteration of the algorithm that is required for successful operation within a control loop, as described in the closed-loop system, is determined by the rate of velocity updates required by the UAV used. The manufacturer of the DJI Mavic 3 states a required update rate of 5 Hz for velocity setpoints, which equates to a time interval of 200 ms. Therefore, a computation time significantly smaller than 200 ms is needed to still leave time for the rest of the trajectory execution system to perform additional calculations and transmit the signal to the UAV. The choice of the two different test assemblies was based on their geometric complexity, as a more complex model increases the execution times of the algorithm. The relatively simple geometry of the A350 fuselage section results in a mesh consisting of 8440 points. The more complex model of the Section 19 requires 35121 points for an accurate representation.

Parameter	**A350 Fuselage Section**	**A320 Section 19**
Final Spatial Coverage	90 %	97 %
Δt Reference	790 ms	1664 ms
Δt Optimization	122 ms	632 ms
- Δt HEDAC	50 ms	373 ms
- Δt Coverage	61 ms	253 ms
- Δt Collision Avoidance	11 ms	6 ms
Mesh Points	8440	35121

Table 5.3. Comparison of the execution times of the original and optimized version of the HDEAC algorithm during tests with the A350 fuselage section and A320 Section 19.

Table 5.3 shows the execution times per iteration of the individual parts of the algorithm. In the original implementation, the total average time per iteration was 0.79 s for the A350 fuselage section. Also, a high variance of the individual calculation times was observed with many outliers taking over 2.5 s to complete. Running the surface coverage calculation in a parallel thread reduced the average execution time to 0.122 s with 0.05 s caused by the HEDAC control algorithm 0.061 s by the calculation of the spatial coverage and 0.011 s by the collision avoidance algorithm. The variance could be significantly improved as well with none of the observed execution times exceeding 0.2 s. However,

even with this comparatively simple geometry, an average execution time of 0.122 s leaves less than 0.9 s for additional calculations that need to be performed by the system prior to sending the computed target velocity to the UAV.

While the online execution of the algorithm is plausible using simple models, the more complex Section 19 increases execution times per iteration way over the required threshold. With the original algorithm, an average execution time of 1.664 s per iteration was observed with a high variance and a number of steps taking over 3 s to complete. The improved algorithm requires 0.626 s per iteration on average with a significant decrease in variance comparable to the A350 fuselage section. 0.373 s of this time was caused by the HEDAC control algorithm 0.253 s by the calculation of the spatial coverage and 0.006 s by the collision avoidance algorithm. These times exceed the maximum viable threshold for reliable control of the DJI Mavic 3, even without considering additional calculations.

While the results of the performed optimizations look promising, additional optimizations are required to allow for reliable execution times short enough for online execution. Possible ways to further increase the performance of the algorithm include exploring how far mesh geometry can be simplified before negatively impacting planning performance, and parallelization of the spatial coverage calculation and HEDAC control algorithm. However, this is more complex than the parallelization of the surface coverage calculations, as both parts are dependent on each other's output.

Photogrammetric Documentation of Assemblies

Since the performance of the closed-loop planning algorithm was not sufficient for inspection flights, the following chapter evaluates the integration of the open-loop navigation into the presented system[2]. As described in Sections 3.6 and 4.2.6, the implementation of the HEDAC algorithm was used to generate a path consisting of Points of Interest and Viewpoints by sampling discrete points along the generated trajectory. This path formed the input of the `trajectory_control` node to perform an inspection flight using the relative navigation system presented in Section 4.2.3. The Potential Field Navigation strategy was used since the individual points are closely spaced, and the HEDAC algorithm produces a trajectory that keeps a safe distance from all obstacles. Therefore, the risk of encountering local minima is negligible. The HEDAC algorithm generated an inspection path for the A350 fuselage section and the A320 Section 19. The algorithm was set up to plan the inspection trajectory up to an inspection duration of 600 s. Figure 5.39 shows the planned trajectory and camera orientations for selected points in the trajectory for the Section 19 assembly. Figures 5.37 and 5.38 show the convergence of the spatial and surface coverage of the generated trajectories, which resulted in a surface coverage of over 90%. With a maximum velocity of 0.1 $\frac{m}{s}$, both trajectories reach the maximum surface coverage at around 300 s. The UAV navigated without violating the safety distance with the exception being the beginning of the Section 19 trajectory caused by an unfortunate combination of the low distance field

[2] Many thanks to Simon Hornung, who implemented and evaluated parts of the photogrammetry system as part of his master's thesis [118].

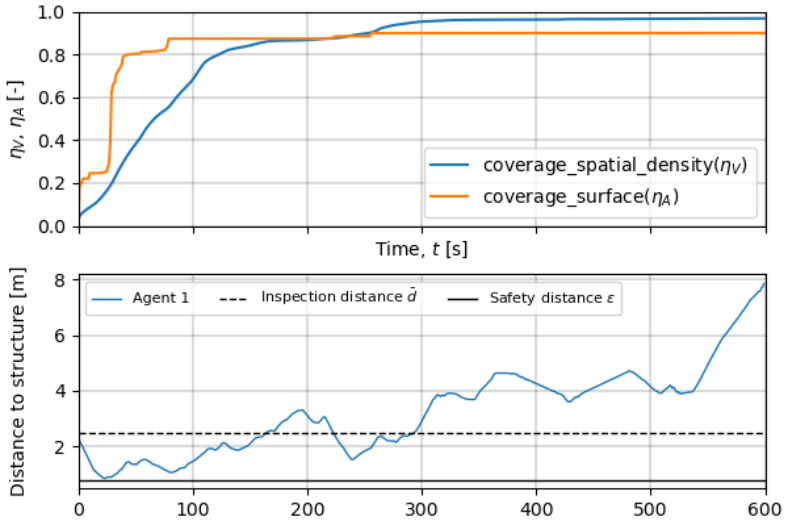


Figure 5.37. Spatial density and surface coverage as well as distance to the closest obstacle for the A350 fuselage section test assembly. Image source: Simon Hornung [118].

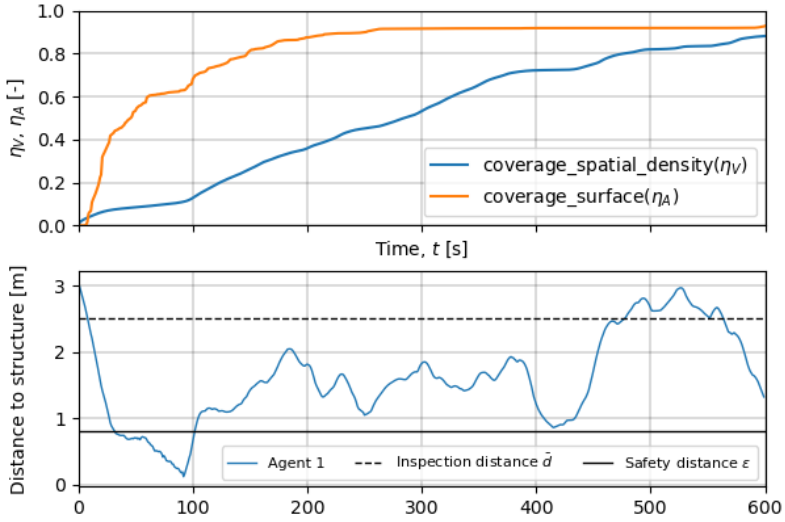


Figure 5.38. Spatial density and surface coverage as well as distance to the closest obstacle for the Section 19 test assembly. Image source: Simon Hornung [118].

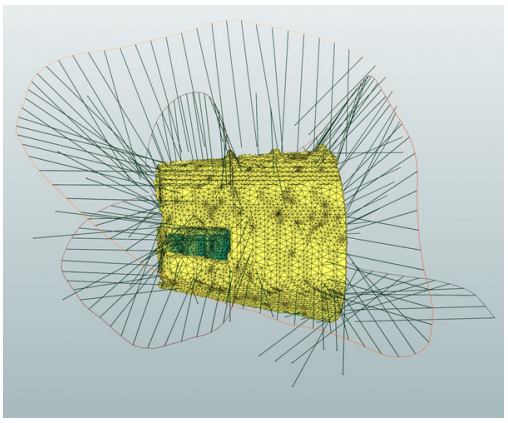


Figure 5.39. Trajectory and camera orientations calculated by the HEDAC algorithm for the Section 19 test assembly.

Figure 5.40. Result of the photogrammetric reconstruction of the Section 19 test assembly.

inside the concave fuselage section and the target density field pushing the UAV inside the assembly (see: Figure 5.39).

However, this was automatically detected in the path creation step, and no Viewpoints were sampled in this section of the trajectory. Additionally, the potential field navigation and collision avoidance watchdog would have prevented a collision during execution. The path creation worked as expected, with the system automatically deleting Viewpoints located outside the range that can safely be flown inside the flight arena. Afterward, inspection flights were performed based on the generated paths. Photogrammetric reconstruction was performed with the proposed photogrammetry pipeline. Figure 5.40 shows the result of the photogrammetric reconstruction of the Section 19. During the inspection flights, the UAV autonomously captured high-quality images of the specified Points of Interest. The photogrammetry system was able to reconstruct textured 3D models of the assembly based on these images. However, the photogrammetric reconstruction process was not entirely successful due to the thin sheet-metal or carbon fiber geometries of the fuselage sections. Figure 5.40 shows a number of holes that appear at particularly thin sections of the model where the algorithm had trouble reconstructing the 3D geometry of the parts. The tool cart on which the A350 was resting was reconstructed without these holes, suggesting that the thin material might be the cause of this problem.

Using the Captured Images to Texture the CAD Model

As described in the previous section, the main problem during the photogrammetric process was the reconstruction of an accurate mesh of the reconstructed model. However, due to the available CAD data, an accurate mesh can be derived from this data through tessellating the CAD model. Therefore, the feasibility of texturing this model using the data generated from the photogrammetric process was evaluated. This mesh was loaded in the 3D modeling software MeshLab [47] along with the captured images and

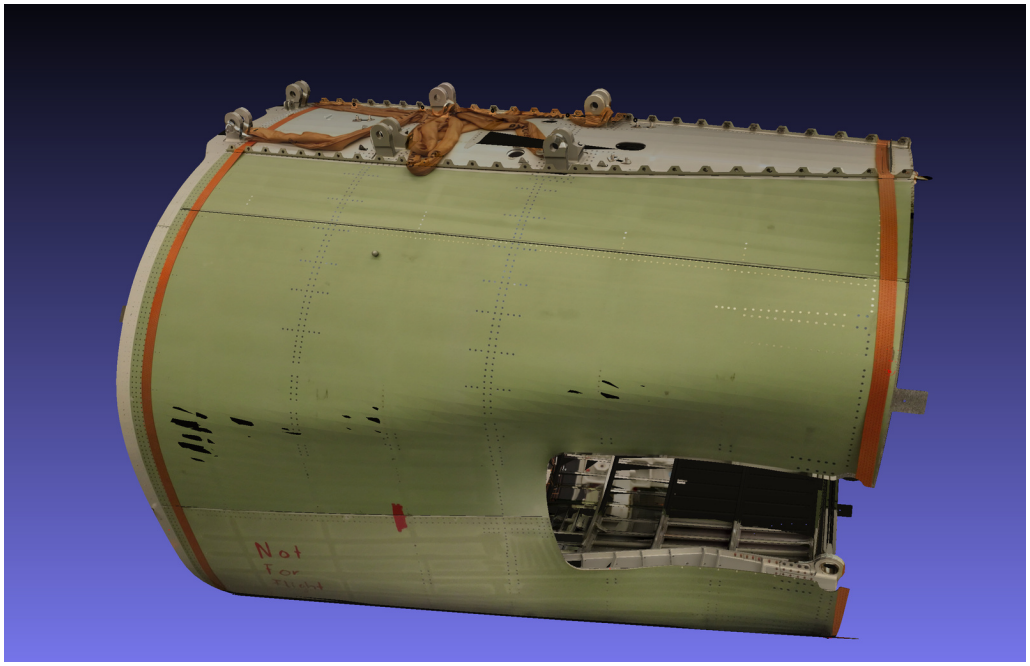


Figure 5.41. Textured mesh generated from CAD data and camera images.

the camera poses generated by AliceVision Meshroom during the photogrammetric reconstruction process. The origin of the mesh data generated from the CAD data was manually modified to account for the arbitrary origin of the camera poses generated in AliceVision Meshroom. The positions were also scaled up to compensate for the incorrect scale of the photogrammetric model. Afterward, the camera poses can be used to align the camera images to the 3D model and generate a texture for the CAD-based mesh. Testing revealed that this process requires significantly fewer images of the model to produce a high-quality mesh, and reducing the number of images yielded in better results, as this resulted in less overlapping areas that result in redundant texture information that must be blended together. Figure 5.41 shows the result of the texturing process. As expected, the geometry of the mesh is far better than the photogrammetric model, and the generated texture reveals more details of the surface of the part.

Figure 5.42 shows a close-up comparison between the surface of a photogrammetrically reconstructed and textured CAD model. The increased resolution and detail is even more pronounced in this comparison.

Discussion of the Evaluation Results

During performance evaluation of the optimized HEDAC algorithm, an increase in performance could be achieved, which resulted in a decrease in execution time by 84.6% and 41.7% in the tested scenarios. Furthermore, the variance of the execution times per iteration was significantly reduced to a level suitable for online execution of the algorithm. However, the achieved reduction in calculation time per iteration was not

sufficient for safe, closed-loop flight. With future work on increased parallelization, reduction of mesh complexity, and using a higher-performance CPU, the execution times might be reduced enough for use in a closed-loop system. The inspection flights performed by the open-loop system captured high-quality images of the Points of Interest calculated by the original HEDAC algorithm. However, the developed photogrammetry pipeline had problems reconstructing the 3D geometry of the thin structures of the aircraft fuselage parts. The photogrammetry process might be further improved by changing the meshing step of the photogrammetry pipeline, so thinner sections of the mesh geometry are less likely to be filtered out. A different approach using the mesh information of the existing CAD model, images captured by the inspection flight, and the camera poses calculated by the photogrammetry system was able to achieve better results. The geometry of the model exactly matches the CAD data, and the texture generated from the high-resolution images enables an accurate documentation of the assembly's state at the time of the inspection.

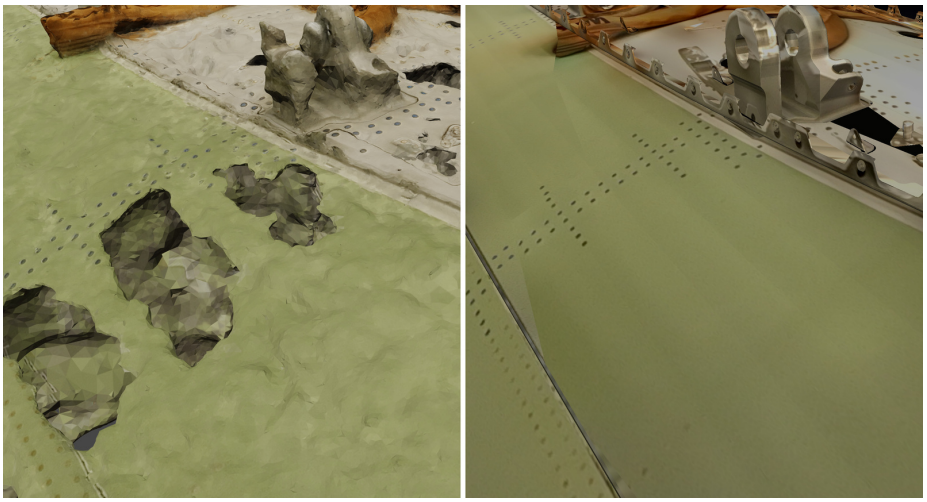


Figure 5.42. Close up comparison between the photogrammetric reconstruction (left) and textured CAD model (right).

5.2.7 Presence Detection on an Assembly in a Production Scenario

The following paragraphs provide the proof of concept of the inspection subsystem of the QuAD platform. To verify the functionality of the interface for external inspection tools, the presence detection functionality of the software VisionLib [286] was integrated, and its performance was evaluated through the inspection of a bracket mounted to the Section 19 fuselage. The functionality of the presence detection system based on the concept presented in Section 3.5.1 is demonstrated through a similar inspection scenario.

Integration of Existence Checks Based on VisionLib

To demonstrate the integration of an existing inspection tool into the system, an existence-check based on the software VisionLib (see: Section 3.3.3) [286] was implemented. The existing example code for an existence check in the Unity Software Development Kit was extended to display a colored overlay of the tracked assembly and the inspected part. Figure 5.43 shows the AR overlay on the live video stream from the

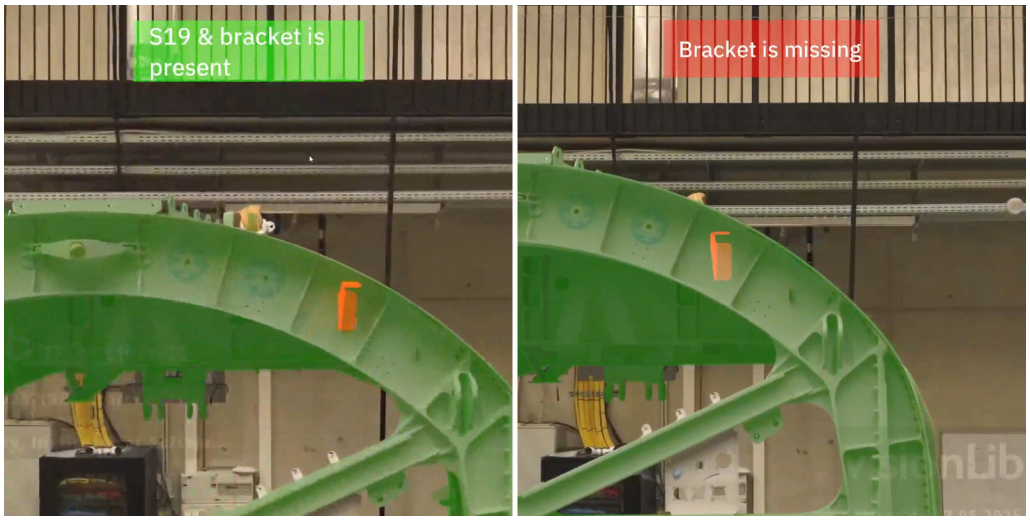


Figure 5.43. AR overlay for the performed presence detection with a correctly installed (left) and missing (right) part.

UAV and the inspection status for a correctly installed (left) and missing (right) part. The example code was further modified to use the Unity ROS TCP Connector interface [274] to communicate with the main system running in ROS. An additional ROS node was added implementing an action server for the existence check using the `inspect.action` as an interface. This node passes information about the currently inspected part to VisionLib and forwards the inspection status as a result for the action call.

Presence Detection with Template Matching

The effectiveness of the template matching-based presence detection system proposed in Section 3.5.1 was evaluated using a set of 98 images of the Section 19 assembly, of which 23 contained a correctly installed bracket, 15 had the bracket installed correctly, and 60 had no bracket installed. The tests revealed that this approach is effective in detecting the correct installation of the detected brackets in ideal conditions. The image alignment of the rendered images using the Super Glue algorithm was able to reliably match the alignment of the center area of the renders with the images. Figure 5.44 shows an overlay of the warped render over the original image (a) and crops to the area of the inspected bracket with the same overlay for several inspection images (b). These crops were automatically generated by applying OpenCV's `threshold()` function to the warped image containing only the render of the bracket. This function sets all images below a specific threshold (the background) to black, while all other pixels (the bracket) are set to white. Using the resulting image, the minimum and maximum values of each white pixel in both axes can be used to calculate the coordinates of a bounding box around the bracket, which in turn can be used to extract the relevant area of the original image and save it for later review or processing. Using this technique, an area where the bracket should be located could be successfully extracted in 94.6 % of the tested images.

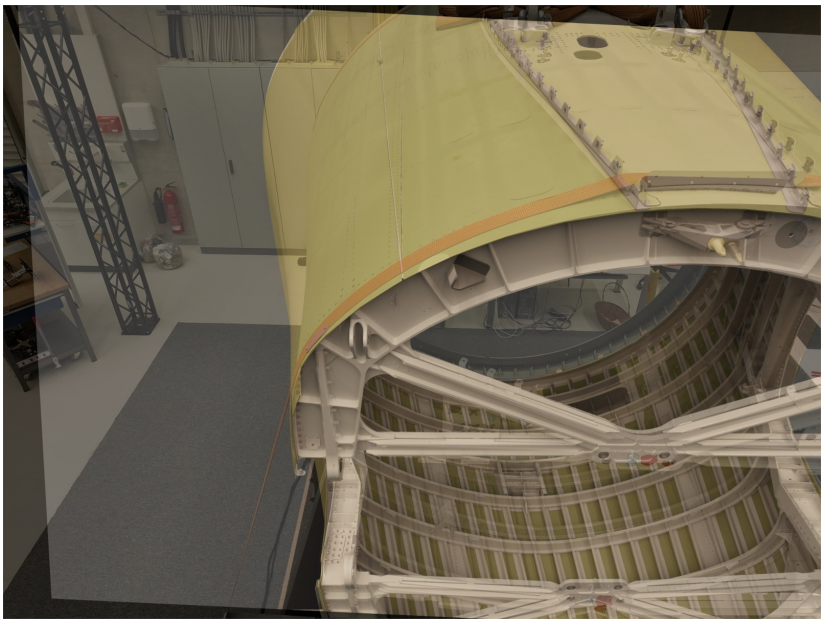
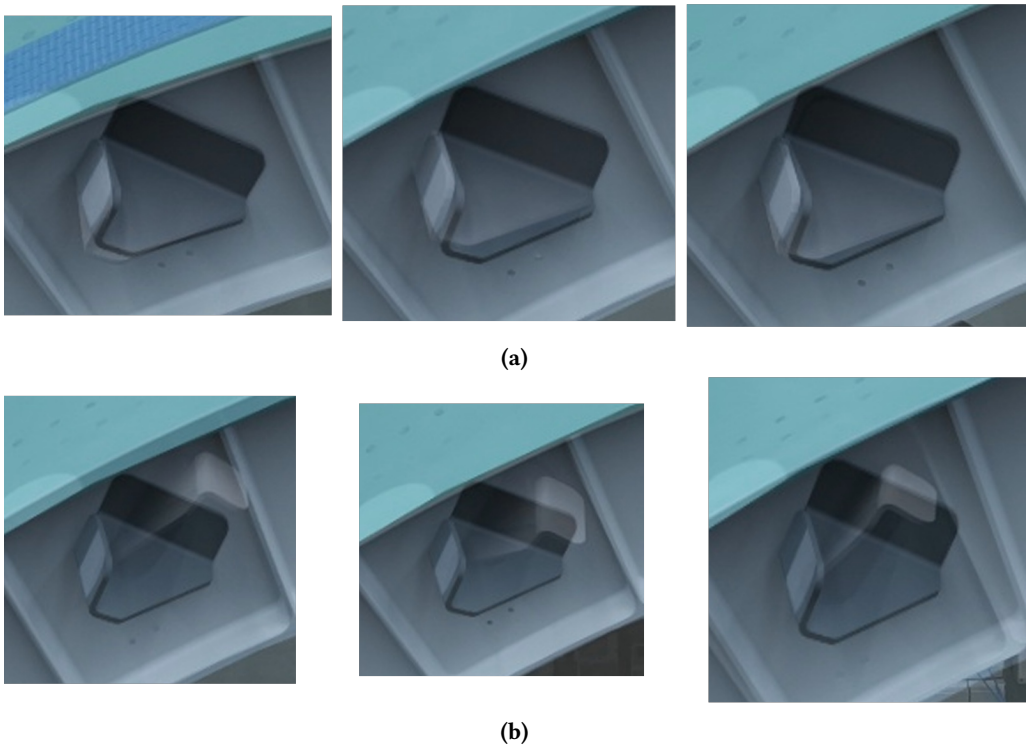


Figure 5.44. Overlay of the warped render over the captured image.

Unfortunately, the template recognition only worked under ideal conditions. The template matching algorithms provided by OpenCV require ideally matched conditions between the camera image and the render. Especially recreating the correct lighting in the renders was found to have a significant impact on the detection rate. While

the template matching worked in some conditions, the approach was found to be too unreliable for the application in a real-world scenario. However, the precisely extracted regions of the bracket locations might be used in conjunction with different detection algorithms. In particular, model-based zero-shot pose estimators like MegaPose [148]. These algorithms take an object and a 2D image that contains the object as input and produce a pose estimate as an output. Internally, they work by rendering the object at different orientations and determining the probability that an object similar to the render of a particular orientation is present in the image. Since the detection step is based on a classifier neural network, the internal models had to be retrained for each particular object that needed to be detected. This made the application of these algorithms unattractive for inspection scenarios where the detected objects change frequently. Zero-shot pose estimators like MegaPose do not require retraining and are therefore more suited for inspection applications. However, they still require the region containing the object to be isolated in advance.



(a)

(b)

Figure 5.45. Automatically isolated image sections containing the area where the bracket should be mounted with an overlay of the rendered model. Brackets in (a) are correctly installed, and (b) shows the overlay over images with an incorrectly installed bracket.

Usually, this is done using object detection networks like EfficientDet [272], which often need to be retrained to detect the object reliably. The complexity of the retraining process might be reduced if, instead of retraining with renders of the object from all orientations, only the generated render of the object in the correct orientation is used.

However, this might result in a decreased detection performance, especially if the object is mounted (and therefore photographed) in the wrong orientation.

Figure 5.46. Selection of areas where the inspected should be located based on the mask generated by thresholding the rendered bracket.

If the input can only be generated, if the object is already in the right orientation, this makes the pose estimation redundant. The approach for cropping the captured inspection images described above works independently of the orientation or presence of the object, as the algorithm uses the image and the entire assembly as input. This allows for a robust extraction of the area containing the bracket, which can serve as an input for the pose estimator. While the pose estimation was not implemented as part of this dissertation, the quality of the masking provided by the proposed approach (see Figure 5.46) seems promising enough to realize a pose optimization-based presence, position, and orientation detection system in the future.

5.3 Inspection of a Wind Turbine

As described in the second case study, the inspection of a wind turbine offers an excellent opportunity to evaluate the relative positioning system. The complex kinematic model of a wind turbine is ideal for demonstrating the approach for relative positioning using changing reference systems proposed in Section 3.3.5 and implemented in Section 4.2.1. The following sections describe the results of simulated inspection flights performed along a rotating wind turbine. The simulated UAV navigates to a series of Viewpoints, which are defined relative to the respective parts of the wind turbine.

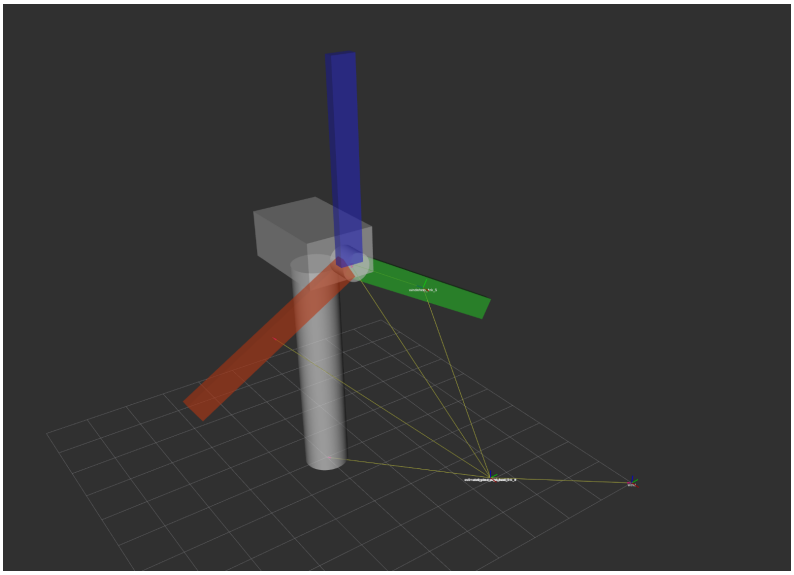


Figure 5.47. Setup for the simulated test flights performed on the moving wind turbine.

Figure 5.47 illustrates the test setup.

The wind turbine is modeled as a kinematic robot using the *URDF* notation and controlled using the Framework *MoveIt* [80]. At the beginning of the flight, the wind turbine is stationary. At a specific point in the inspection flight, the turbine hub starts rotating with a constant speed, while the UAV navigates to Points of Interest defined relative to the (now moving) rotor blades. To evaluate the performance of the velocity compensation mechanism based on kinematic models described in Section 3.3.5, this approach is compared to the filter that does not consider assembly velocities (see: Section 3.3.4). Since model tracking is the most critical component for reliable position estimation, random jumps in the tracking position and dropouts are simulated to evaluate the performance of the state estimate under suboptimal conditions. The simulated IMU measurements contain an artificially generated drift and noise (see: Section 5.1.4) to emulate real-world conditions. Also, the numeric differentiation of the link velocities inherently introduces noise into this measurement.

5.3.1 Influence of Component Velocities on the State Estimate

To evaluate the performance increase through compensating the commanded velocity and measured IMU velocity, the previously described inspection flight is performed with both implementations of the sensor fusion algorithm running in parallel. In this inspection flight, all Viewpoints are defined relative to the first rotor blade of the wind turbine. The `dkf_kinematic_state_estimation_publisher` (Velocity compensated Kalman Filter, VKF) is used for position inputs by the navigation algorithm. The `dkf_state_estimation_publisher` (KF) is merely used to provide a reference for determining the performance improvement.

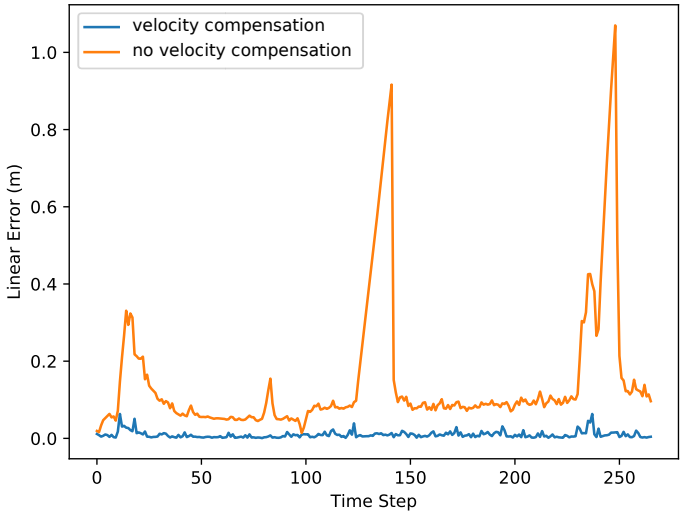


Figure 5.48. Position error of the state estimate with and without compensation of the assembly velocity. One time step corresponds to 200 ms.

Figure 5.48 shows the position error recorded for both filters throughout the flight. The graph begins shortly after takeoff, and the rotor starts moving at time step 12. The rotor movement induced a position error in the uncompensated filter as the IMU measurements and the simulated model tracking became inconsistent. The error of the VKF implementation hardly changed, suggesting that the compensation of the IMU and control velocity effectively reduces the position error. Missing model tracking updates, either caused by the tracking filter removing improbable measurements or through a manually induced loss of tracking, can be observed in time steps 78, 97, around 140, and 240. While the Kalman Filter's position considerably drifted when operating solely on IMU data and its internal model, the compensated velocities in the VKF significantly reduced this drift. The mean average position error (MAE) for the KF during the flight was 0.149 m while the VKF was able to achieve a significantly lower MAE of 0.009 m.

5.3.2 Kinematic Localization with Changing Reference Frames

The quality of the position estimation system relative to changing reference frames (Multiple reference frame velocity compensated Kalman Filter, MVKF) was evaluated[3] using a similar inspection scenario with the UAV navigating a path described by a series of waypoints, which are defined relative to different reference coordinate frames. The implementation was validated against the performance of the VKF in two different test runs. At first, both filters operated entirely without model tracking input to simulate a tracking loss. When changing the reference frame, the sudden change in position and orientation caused a peak in position and orientation error in the VKF's single state vector. The position error equates to the distance between the old and new reference frames. During testing, errors of around 2.5 m were observed in all axes. After the change, the internal state requires several seconds to reduce the position error to less than 0.2 m. When the MVKF implementation's reference frame is changed, its position output changes to the internal filter corresponding to the frame. This filter operated using transformed measurements in its reference frame during the entire flight, so the new measurements do not conflict with the values of its state vector. As a result, the change in reference frames results in a clean transition of the pose estimate to the new reference frame without significant error.

The experiment was repeated with simulated model tracking data provided to the sensors, which resulted in a similar, but less pronounced effect. The tracking measurements caused a significantly faster adjustment of the VKF's state to the new reference frame. The initial position error disappeared after about 200 ms while the orientation in pitch and roll axis showed minor oscillations with a peak amplitude of around 30°and a frequency of 2 Hz, which lasted for around one second. The MVKF performed similarly to the first flight. When the reference frame was changed, its output changed to the internal Kalman Filter of the corresponding reference frame, which produced a clean transition of the output signal. As with the evaluation of the VKF using a constant reference frame, the system's benefits are more pronounced in suboptimal conditions (i.e., when position measurements of the model tracking are bad or drop out completely).

[3] Many thanks to Raphael Katschinsky, who evaluated the state estimation system for changing reference frames as part of his master's thesis [137].

Summary. In this chapter, the research results obtained in the course of the dissertation are reviewed and evaluated.

6 Conclusion and Outlook

This work aimed to develop a platform for autonomous inspection of large structures (e.g., aircraft fuselage parts). Throughout this dissertation, the concept, architecture, and implementation of the individual parts of this platform were described in detail, and their functionality was demonstrated using a variety of evaluations, prototypes, and tests. This includes a path-planning solution for targeted inspection of individual parts as well as a trajectory generation method for full-surface inspections of entire assemblies. In addition, a novel relative positioning and vision-based, collision-avoiding navigation approach using only the UAV's internal sensors was introduced. It consists of a relative position estimation system based on a Kalman Filter, which uses input from IMU measurements of the UAV and a model tracking system to determine the UAV's position relative to the inspected assembly. This position is used in a modular obstacle-avoiding navigation system, which provides an octree-based representation of detected obstacles from the UAV's onboard sensors to an arbitrary navigation algorithm. Throughout this dissertation, two interchangeable approaches based on potential field navigation and the graph-search-based Anytime Dynamic A* algorithm were implemented and tested. While this work focuses on relative path planning, position estimation, and navigation, the system developed and integrated two inspection methods for presence detection and photogrammetric documentation of assemblies. The following paragraphs summarize the achieved results of the subsystems mentioned above.

Path Planning

The inspection-centered path planning system developed in this work is versatile and extensible. It allows for the specification and calculation of efficient inspection paths based on Points of Interest, Viewareas, and Viewpoints using a graphical user interface designed for this task, as well as planning paths covering the entire surface of an assembly. The implemented path planning system based on the Ant Colony System algorithm can calculate efficient inspection paths, with its performance evaluated in

Section 5.2.1. The concept of Viewareas allows for the optimization of inspection path length and duration as well as planning time which was demonstrated through planning an inspection path along a fuselage section where a 37.6 %, reduction in inspection path length and 57 % less braking and acceleration maneuvers could be achieved (see: Section 5.2.2). The functionality of a second path optimization strategy, where the quality of individual Viewpoints is *learned* throughout multiple similar inspections, was verified in simulation (see: Section 5.2.3). While the path planning system focuses on targeted inspections of individual Points of Interest, it is versatile enough to allow for inspections where coverage of the entire surface of the assembly is required. This was demonstrated through the photogrammetric reconstruction of two aircraft fuselage parts using the HEDAC coverage algorithm to generate a trajectory and sampling Viewpoints based on this trajectory (see: Section 5.2.6).

Position Estimation

This work proposes a novel approach for relative navigation for UAVs relying entirely on onboard sensors. The system works independently of external positioning systems like GPS. The key component of the system is measuring the UAV's position relative to an object using a model tracking algorithm. The approach combines this pose with onboard IMU measurements from the UAV in a Kalman Filter, which provides a stable position estimate. The functionality of the proposed position estimation system has been evaluated in simulation in previous work [248]. This dissertation provided further verification based on real-world test flight data (see: Section 5.2.4). Additionally, an extension of the system that provides better position estimation based on kinematic modeling of the tracked assembly was proposed and evaluated. Its functionality was demonstrated in simulated test flights, which significantly improved the quality of the position estimate when navigating relative to moving objects.

Navigation

As part of this dissertation, two obstacle-avoiding navigation strategies for selective inspections were implemented and tested. One approach is based on the potential field method, while the other uses the graph-search-based Anytime Dynamic A* algorithm. Both were evaluated in simulated and real-world testing. While both approaches successfully navigated towards a goal efficiently, the results of the tests suggest that a combination of both methods may yield even better performance (see: Section 5.2.5). Optimizations made to the HEDAC trajectory planning system for generating trajectories covering the surface of an assembly significantly reduced the required execution time. However, they were not sufficient for execution at runtime (see: Section 5.2.6). However, the tests revealed further possibilities for improvements, which can be explored in future work.

Collision Avoidance

This work presented a system for processing low-quality obstacle sensor data and creating a representation of the detected obstacles for navigation to achieve collision-free flight. The system uses an octree-based representation for efficient storage of detected obstacles. Several test flights performed throughout this work demonstrated the effectiveness of this approach and its ability to handle previously known and unknown obstacles, as well as static and moving obstacles. This system has been in continuous use almost for all test and demonstration flights and in combination with the collision avoidance watchdog, it has not only enabled flights inside complex and difficult to navigate environments like the inside of a fuselage section, it has significantly contributed to the fact, that no major crash has occurred during the almost six years of developing the platform and the only damage inflicted to the used UAVs were a set of broken propellers.

Inspection

The inspection subsystem of the platform was intentionally implemented to be open for extension by external inspection tools. The extensibility of the system was demonstrated with the integration of the VisionLib *existence check* functionality (see: Section 5.2.7). Additionally, an approach for performing a rudimentary existence check based on template matching using a render of the inspected part on a captured image of the Point of Interest was implemented and tested with limited success. However, the approach provides a precise overlay of the target state of the assembly for easier manual verification. A promising strategy for isolating the image region containing the POI was also developed. This strategy can augment the relevant area for the inspector and will serve as input for a pose-estimation-based inspection algorithm in future work. Lastly, the platform's effectiveness in coverage-based inspection scenarios was demonstrated by processing images gathered during flights covering the entire surface of the assembly. The images were used to perform a photogrammetric reconstruction of the assembly (see: Section 5.2.6) as well as a photorealistic texture for the CAD model, which can be used for inspections and documentation of the assembly's state.

Bibliography

[1] 8tree GmbH. 8tree: dentcheck: Dent inspection within seconds, 2025. URL https://www.8-tree.com/products-services/dentcheck-2/. (Accessed: 2025-04-27).

[2] F. Ababsa. A new 3d model-based tracking technique for robust camera pose estimation. *International Journal of Advanced Computer Science and Applications*, 3(4), 2012.

[3] F. Ababsa and M. Mallem. Robust camera pose estimation combining 2d/3d points and lines tracking. In *2008 IEEE International Symposium on Industrial Electronics*, pages 774–779, 2008. doi: 10.1109/ISIE.2008.4676964.

[4] Aerones. Wind turbine inspection - robotic wind turbine care systems | aerones, 2025. URL https://aerones.com/services/inspection/drone-inspections/. (Accessed: 2025-02-02).

[5] Aerones. Wind turbine inspection - robotic wind turbine care systems | aerones, 2025. URL https://aerones.com/services/inspection/. (Accessed: 2025-02-02).

[6] Aerones. Blitzschutzsystem - robotic wind turbine care systems | aerones, 2025. URL https://aerones.com/de/dienstleistungen-2/inspektion/leitfahigkeit/. (Accessed: 2025-04-15).

[7] Airbus. Airbus demonstrates aircraft inspection by drone at farnborough, 2016. URL https://www.airbus.com/en/newsroom/press-releases/2016-07-airbus-demonstrates-aircraft-inspection-by-drone-at-farnborough. (Accessed: 2025-04-08).

[8] Airbus. Airbus launches advanced indoor inspection drone to reduce aircraft inspection times and enhance report quality, 2018. URL https://www.airbus.com/en/newsroom/press-releases/2018-04-airbus-launches-advanced-indoor-inspection-drone-to-reduce-aircraft. (Accessed: 2025-04-08).

[9] Airbus. Accident rates by category and generation accidentstats.airbus.com, 2025. URL https://accidentstats.airbus.com/accident-rates-by-category-generation/. (Accessed: 2025-05-06).

[10] Z. AIS. Industrial climbing - zeppelin ais, 2025. URL https://zeppelin-ais.com/en/services/special-services/industrial-climbing. (Accessed: 2025-01-03).

[11] A. Akca and M. Ö. Efe. Multiple model kalman and particle filters and applications: A survey. *IFAC-PapersOnLine*, 52(3):73–78, 2019.

[12] J. L. Alarcon-Herrera, X. Chen, and X. Zhang. Viewpoint selection for vision systems in industrial inspection. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4934–4939, May 2014. doi: 10.1109/ICRA.2014.6907582.

[13] P. F. Alcantarilla and T. Solutions. Fast explicit diffusion for accelerated features in nonlinear scale spaces. *IEEE Trans. Patt. Anal. Mach. Intell*, 34(7):1281–1298, 2011.

[14] B. S. Aleshin, A. I. Chernomorsky, E. D. Kuris, K. S. Lelkov, and M. V. Ivakin. Robotic complex for inspection of the outer surface of the aircraft in its parking lot. *Incas Bulletin*, 12:21–31, 2020.

[15] M. O. Aqel, M. H. Marhaban, M. I. Saripan, and N. B. Ismail. Review of visual odometry: types, approaches, challenges, and applications. *SpringerPlus*, 5:1–26, 2016.

[16] Autaza. Ai experience | autaza, 2025. URL https://www.autaza.com/en/projects-8. (Accessed: 2025-04-15).

[17] N. P. Avdelidis, A. Tsourdos, P. Lafiosca, R. Plaster, A. Plaster, and M. Droznika. Defects recognition algorithm development from visual uav inspections. *Sensors*, 22(13):4682, 2022.

[18] M. Babic, M. A. Farahani, and T. Wuest. Image based quality inspection in smart manufacturing systems: A literature review. *Procedia CIRP*, 103:262–267, 2021. ISSN 2212-8271. doi: https://doi.org/10.1016/j.procir.2021.10.042. URL https://www.sciencedirect.com/science/article/pii/S2212827121008830. 9th CIRP Global Web Conference – Sustainable, resilient, and agile manufacturing and service operations : Lessons from COVID-19.

[19] T. Bailey and H. Durrant-Whyte. Simultaneous localization and mapping (slam): part ii. *IEEE Robotics Automation Magazine*, 13(3):108–117, Sep. 2006. doi: 10.1109/MRA.2006.1678144.

[20] W. Bank. Air transport, passengers carried | data, 2025. URL https://data.worldbank.org/indicator/IS.AIR.PSGR. (Accessed: 2025-02-04).

[21] K. Bardis, N. P. Avdelidis, C. Ibarra-Castanedo, X. P. V. Maldague, and H. Fernandes. Advanced diagnostics of aircraft structures using automated non-invasive imaging techniques: A comprehensive review. *Applied Sciences*, 15(7), 2025. ISSN 2076-3417. doi: 10.3390/app15073584. URL https://www.mdpi.com/2076-3417/15/7/3584.

[22] F. Bellalouna. The augmented reality technology as enabler for the digitization of industrial business processes: case studies. *Procedia CIRP*, 98:400–405, 2021.

[23] N. Biggs. The traveling salesman problem a guided tour of combinatorial optimization, 1986.

[24] A. Bircher, K. Alexis, M. Burri, P. Oettershagen, S. Omari, T. Mantel, and R. Siegwart. Structural inspection path planning via iterative viewpoint resampling with application to aerial robotics. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6423–6430, 5 2015. doi: 10.1109/ICRA.2015.7140101.

[25] D. Bohdanov and H. Liu. Vision-based quadrotor micro-uav position and yaw estimation and control. In *AIAA Guidance, Navigation, and Control Conference*, page 5048, 2012.

[26] K. S. Bohra and Y. S. Dharmadhikari. History of fly by wire, the tech that expanded human horizons in aviation. In *2023 8th IEEE History of Electrotechnology Conference (HISTELCON)*, pages 46–51, 2023. doi: 10.1109/HISTELCON56357.2023.10365861.

[27] F. Bourgeois, L. Kneip, S. Weiss, and R. Siegwart. Delay and dropout tolerant state estimation for mavs. In *Experimental Robotics: The 12th International Symposium on Experimental Robotics*, pages 571–584. Springer, 2014.

[28] G. Bressan, D. Invernizzi, S. Panza, M. Lovera, et al. Attitude control of multirotor uavs: cascade p/pid vs pi-like architecture. In *5th CEAS Specialist Conference on Guidance, Navigation and Control-EuroGNC*, pages 1–20, 2019.

[29] S. Bruno, S. Lorenzo, V. Luigi, and O. Giuseppe. Robotics: modelling, planning and control, 2010.

[30] A. Budiyanto, A. Cahyadi, T. B. Adji, and O. Wahyunggoro. Uav obstacle avoidance using potential field under dynamic environment. In *2015 International Conference on Control, Electronics, Renewable Energy and Communications (ICCEREC)*, pages 187–192. IEEE, 2015.

[31] N. Bäns. Automation @ airbus - from the past to manufacturing 4.0, Nov. 2018. URL https://doi.org/10.5281/zenodo.2317453.

[32] L. Calvet, P. Gurdjos, C. Griwodz, and S. Gasparini. Detection and accurate localization of circular fiducials under highly challenging conditions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 562–570, 2016.

[33] J. F. Canny. A variational approach to edge detection. In *AAAI*, volume 1983, pages 54–58, 1983.

[34] V. Capitalist. Visual capitalist - the largest aircraft makers in 2024, 2025. URL https://www.visualcapitalist.com/cp/the-largest-aircraft-companies-in-2024/. (Accessed: 2025-03-13).

[35] D. Cazzato, M. A. Olivares-Mendez, J. L. Sanchez-Lopez, and H. Voos. Vision-based aircraft pose estimation for uavs autonomous inspection without fiducial markers. In *IECON 2019 - 45th Annual Conference of the IEEE Industrial Electronics Society*, volume 1, pages 5642–5648, 2019. doi: 10.1109/IECON.2019.8926667.

[36] A. Chakravarthy and D. Ghose. Obstacle avoidance in a dynamic environment: A collision cone approach. *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans*, 28(5):562–574, 1998.

[37] K. Chang, Y. Xia, and K. Huang. Uav formation control design with obstacle avoidance in dynamic three-dimensional environment. *SpringerPlus*, 5:1–16, 2016.

[38] K. Chaudhari and A. Thakkar. Travelling salesman problem: An empirical comparison between aco, pso, abc, fa and ga. In N. R. Shetty, L. M. Patnaik, H. C. Nagaraj, P. N. Hamsavath, and N. Nalini, editors, *Emerging Research in Computing, Information, Communication and Applications*, pages 397–405, Singapore, 2019. Springer Singapore. ISBN 978-981-13-6001-5.

[39] K. Chaudhari and A. Thakkar. Travelling salesman problem: an empirical comparison between aco, pso, abc, fa and ga. In *Emerging Research in Computing, Information, Communication and Applications: ERCICA 2018, Volume 2*, pages 397–405. Springer, 2019.

[40] J. Chen and S. Shen. Using a quadrotor to track a moving target with arbitrary relative motion patterns. *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5310–5317, 2017.

[41] X. Chen and J. Zhang. The three-dimension path planning of uav based on improved artificial potential field in dynamic environment. In *2013 5th International Conference on Intelligent Human-Machine Systems and Cybernetics*, volume 2, pages 144–147, 2013. doi: 10.1109/IHMSC.2013.181.

[42] X. Chen, M. Zhao, and L. Yin. Dynamic path planning of the uav avoiding static and moving obstacles. *Journal of Intelligent & Robotic Systems*, 99:909–931, 2020.

[43] K. C. Cheok, M. Radovnikovich, P. Vempaty, G. R. Hudas, J. L. Overholt, and P. Fleck. Uwb tracking of mobile robots. In *21st Annual IEEE International Symposium on Personal, Indoor and Mobile Radio Communications*, pages 2615–2620, 2010. doi: 10.1109/PIMRC.2010.5671780.

[44] C. Choi and H. I. Christensen. Real-time 3d model-based tracking using edge and keypoint features for robotic manipulation. *2010 IEEE International Conference on Robotics and Automation*, pages 4048–4055, 2010.

[45] D. Choi, K. Lee, and D. Kim. Enhanced potential field-based collision avoidance for unmanned aerial vehicles in a dynamic environment. In *AIAA scitech 2020 forum*, page 0487, 2020.

[46] F. Ciampa, P. Mahmoodi, F. Pinto, and M. Meo. Recent advances in active infrared thermography for non-destructive testing of aerospace components. *Sensors*, 18(2), 2018. ISSN 1424-8220. doi: 10.3390/s18020609. URL https://www.mdpi.com/1424-8220/18/2/609.

[47] P. Cignoni, M. Callieri, M. Corsini, M. Dellepiane, F. Ganovelli, and G. Ranzuglia. Meshlab: an open-source mesh processing tool. In *European Interdisciplinary Cybersecurity Conference*, 2008.

[48] G. Cioffi, L. Bauersfeld, E. Kaufmann, and D. Scaramuzza. Learned inertial odometry for autonomous drone racing. *IEEE Robotics and Automation Letters*, 8(5):2684–2691, 2023.

[49] P. Ciro, N. D., and P. Alberto. Target Aware Optimal Visual Navigation for UAVs. *arXiv.org*, 2017.

[50] M. Claybrough. System and method for automatically inspecting surfaces, 2019. URL https://patents.google.com/patent/US10377485B2/en.

[51] I. Colomina and P. Molina. Unmanned aerial systems for photogrammetry and remote sensing: A review. *ISPRS Journal of Photogrammetry and Remote Sensing*, 92:79–97, 2014.

[52] T. Danner and L. E. Kavraki. Randomized planning for short inspection paths. In *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No.00CH37065)*, volume 2, pages 971–976 vol.2, 4 2000. doi: 10.1109/ROBOT.2000.844726.

[53] G. Dantzig, R. Fulkerson, and S. Johnson. Solution of a large-scale traveling-salesman problem. *Journal of the operations research society of America*, 2(4):393–410, 1954.

[54] S. David. *Disasters and accidents in manned spaceflight*. Springer Science & Business Media, 2000.

[55] J. A. de Bonfim Gripp, E. A. de Souza, R. Padovani, and C. Y. Sakuramoto. Method and system for automatic quality inspection of materials and virtual material surfaces, 2021. URL https://patents.google.com/patent/US11024020B2.

[56] S. Deane, N. P. Avdelidis, C. Ibarra-Castanedo, H. Zhang, H. Yazdani Nezhad, A. A. Williamson, T. Mackley, M. J. Davis, X. Maldague, and A. Tsourdos. Application of ndt thermographic imaging of aerospace structures. *Infrared Physics and Technology*, 97: 456–466, 2019. ISSN 1350-4495. doi: https://doi.org/10.1016/j.infrared.2019.02.002. URL https://www.sciencedirect.com/science/article/pii/S1350449518309563.

[57] J. Denavit and R. S. Hartenberg. A kinematic notation for lower-pair mechanisms based on matrices. *Journal of Applied Mechanics*, 22(2):215–221, 06 2021. ISSN 0021-8936. doi: 10.1115/1.4011045. URL https://doi.org/10.1115/1.4011045.

[58] C. Deng, S. Wang, Z. Huang, Z. Tan, and J. Liu. Unmanned aerial vehicles for power line inspection: A cooperative way in platforms and communications. *J. Commun.*, 9(9): 687–692, 2014.

[59] A. Deruaz-Pepin. Method and system for determining the position of a moving craft, 2017. URL https://patents.google.com/patent/WO2017121936A1/.

[60] Deutsche Bahn AG. Kompetenzcenter Multicopter DB, 2025. URL https://www.dbsicherheit.de/dbsicherheit-de/Unsere_Leistungen/Weitere-Leistungen-/02_Multicopter-7749316.

[61] Donecle. Home - donecle, 2025. URL https://www.donecle.com/. (Accessed: 2025-01-03).

[62] Donecle. Component brochure - donecle, 2025. URL https://www.donecle.com/component-brochure/. (Accessed: 2025-05-04).

[63] Donecle. Products - donecle, 2025. URL https://www.donecle.com/products/#iris_GVI. (Accessed: 2025-04-13).

[64] Y. Du, X. Zhang, and Z. Nie. A real-time collision avoidance strategy in dynamic airspace based on dynamic artificial potential field algorithm. *IEEE Access*, 7:169469–169479, 2019.

[65] P. Eide and P. Maybeck. An mmae failure detection system for the f-16. *IEEE Transactions on Aerospace and Electronic Systems*, 32(3):1125–1136, 1996. doi: 10.1109/7.532271.

[66] G. Einicke and L. White. Robust extended kalman filtering. *IEEE Transactions on Signal Processing*, 47(9):2596–2599, 1999. doi: 10.1109/78.782219.

[67] J. Elliott. Nasa's advanced control law program for the f-8 digital fly-by-wire aircraft. *IEEE Transactions on Automatic Control*, 22(5):753–757, 1977. doi: 10.1109/TAC.1977.1101608.

[68] T. Elmokadem. A 3d reactive collision free navigation strategy for nonholonomic mobile robots. In *2018 37th Chinese Control Conference (CCC)*, pages 4661–4666. IEEE, 2018.

[69] T. Elmokadem and A. Savkin. A hybrid approach for autonomous collision-free uav navigation in 3d partially unknown dynamic environments drones 2021, 5, 57. *Feature Papers of Drones*, page 369, 2021.

[70] F. Engel. In-process fault inspection using augmented reality, 2018. URL https://patents.google.com/patent/US9916650B2.

[71] B. Englot and F. S. Hover. Sampling-based coverage path planning for inspection of complex structures. *ICAPS 2012 - Proceedings of the 22nd International Conference on Automated Planning and Scheduling*, 06 2014.

[72] R.-S. et al. Github - bardo91/motion-planning, 2025. URL https://github.com/bardo91/motion_planning. (Accessed: 2025-05-07).

[73] I. FARO Technologies. Faro visual inspect information brochure, 2025. URL https://media.faro.com/-/media/Project/FARO/FARO/FARO/Resources/2_TECH-SHEET/TechSheet_Visual_Inspect/TechSheet_Visual_Inspect_DE.pdf. (Accessed: 2025-04-21).

[74] C. Favre. Fly-by-wire for commercial aircraft: the airbus experience. *International Journal of Control*, 59(1):139–157, 1994.

[75] H. Ferdinando, H. Khoswanto, and D. Purwanto. Embedded kalman filter for inertial measurement unit (imu) on the atmega8535. In *2012 International Symposium on Innovations in Intelligent Systems and Applications*, pages 1–5. IEEE, 2012.

[76] D. Fernandez and A. Price. Visual odometry for an outdoor mobile robot. In *IEEE Conference on Robotics, Automation and Mechatronics, 2004.*, volume 2, pages 816–821. IEEE, 2004.

[77] J. Fernández-Hernandez, D. González-Aguilera, P. Rodríguez-Gonzálvez, and J. Mancera-Taboada. Image-based modelling from unmanned aerial vehicle (uav) photogrammetry:

an effective, low-cost tool for archaeological applications. *Archaeometry*, 57(1):128–145, 2015.

[78] B. Foundation. blender.org - home of the blender project - free and open 3d creation software, 2025. URL https://www.blender.org/. (Accessed: 2025-01-10).

[79] O. R. Foundation. vrpn-client-ros - ros wiki, 2025. URL http://wiki.ros.org/vrpn_client_ros. (Accessed: 2025-03-05).

[80] O. R. Foundation. Moveit motion planning framework, 2025. URL https://moveit.ros.org/. (Accessed: 2025-01-10).

[81] O. R. Foundation. Client libraries - ros wiki, 2025. URL https://wiki.ros.org/Client%20Libraries. (Accessed: 2025-05-06).

[82] O. R. Foundation. robot-localization wiki robot-localization 2.6.12 documentation, 2025. URL https://docs.ros.org/en/melodic/api/robot_localization/html/index.html. (Accessed: 2025-04-22).

[83] O. R. Foundation. robot state publisher - ros wiki, 2025. URL http://wiki.ros.org/robot_state_publisher. (Accessed: 2025-03-11).

[84] O. R. Foundation. tf2 - ros wiki, 2025. URL http://wiki.ros.org/tf2. (Accessed: 2025-01-12).

[85] O. S. R. Foundation. Mp-testdata - the tsplib symmetric traveling salesman problem instances, 2025. URL http://elib.zib.de/pub/mp-testdata/tsp/tsplib/tsp/att48.tsp. (Accessed: 2025-02-25).

[86] O. S. R. Foundation. Gazebo - robot simulation made easy, 2025. URL http://gazebosim.org/. (Accessed: 2025-02-16).

[87] O. S. R. Foundation. Ros official website, 2025. URL http://ros.org. (Accessed: 2025-03-05).

[88] O. S. R. Foundation. Universal robotic description format, 2025. URL https://wiki.ros.org/urdf. (Accessed: 2025-03-05).

[89] O. S. R. Foundation. Rosbag documentation, 2025. URL http://wiki.ros.org/rosbag. (Accessed: 2025-03-05).

[90] O. S. R. Foundation. Ros service documentation, 2025. URL http://wiki.ros.org/Services. (Accessed: 2025-03-05).

[91] O. S. R. Foundation. Ros1 topics documentation, 2025. URL http://wiki.ros.org/Topics. (Accessed: 2025-03-05).

[92] R. P. Foundation. Raspberry pi camera modules, 2025. URL https://www.raspberrypi.com/documentation/accessories/camera.html. (Accessed: 2025-05-04).

[93] R. P. Foundation. Raspberry pi 2 model b product information, 2025. URL https://www.raspberrypi.com/products/raspberry-pi-2-model-b/. (Accessed: 2025-05-04).

[94] F. Furrer, M. Burri, M. Achtelik, and R. Siegwart. Rotors—a modular gazebo mav simulator framework. *Robot Operating System (ROS) The Complete Reference (Volume 1)*, pages 595–625, 2016.

[95] E. Galceran and M. Carreras. A survey on coverage path planning for robotics. *Robotics and Autonomous Systems*, 61:1258–1276, 12 2013. doi: 10.1016/j.robot.2013.09.004.

[96] J. Gall, B. Rosenhahn, and H.-P. Seidel. Robust pose estimation with 3d textured models. In *Pacific-Rim Symposium on Image and Video Technology*, 2006.

[97] L. Gambardella and M. Dorigo. Solving symmetric and asymmetric tsps by ant colonies. In *Proceedings of IEEE International Conference on Evolutionary Computation*, pages 622–627, 1996. doi: 10.1109/ICEC.1996.542672.

[98] M. R. Garey and D. S. Johnson. *Computers and intractability*, volume 29. wh freeman New York, 2002.

[99] N. GmbH. Montagekontrolle mit neurocheck sichert qualitat im produktionsprozess, 2025. URL https://www.neurocheck.de/systemloesungen/anwendungsgebiete/montagekontrolle/. (Accessed: 2025-04-21).

[100] T. GmbH. Mira, airbus augmented-reality-anwendung bei testia erhältlich, 2025. URL https://www.testia.com/de/news-de/mira-airbus-augmented-reality-anwendung-bei-testia-erhaeltlich/. (Accessed: 2025-03-14).

[101] V. B. GmbH. Vmt vison technology information brochure, 2025. URL https://vmt-vision-technology.com/files/225/vmt-unternehmen-de-lq.pdf. (Accessed: 2025-04-21).

[102] R. Gonzalez, F. Rodriguez, J. L. Guzman, C. Pradalier, and R. Siegwart. Combined visual odometry and visual compass for off-road mobile robots localization. *Robotica*, 30(6): 865–878, 2012.

[103] H. González-Banos. A randomized art-gallery algorithm for sensor placement. *Proc. 17th ACM Symp. Comp. Geom.*, pages 232–240, 01 2001. doi: 10.1145/378583.378674.

[104] C. Griwodz, S. Gasparini, L. Calvet, P. Gurdjos, F. Castan, B. Maujean, G. De Lillo, and Y. Lanthony. Alicevision meshroom: An open-source 3d reconstruction pipeline. In *Proceedings of the 12th ACM multimedia systems conference*, pages 241–247, 2021.

[105] L. Group. Ai drone inspect - lufthansa group innovation runway, 2025. URL https://innovation-runway.lufthansagroup.com/en/focus-areas-projects/operations-excellence/ai-drone-inspect.html. (Accessed: 2025-01-06).

[106] S. Gupte, P. Mohandas, and J. Conrad. A survey of quadrotor unmanned aerial vehicles. In *Southeastcon, Proceedings of IEEE*, pages 1–6. IEEE, 2012.

[107] R. A. Hamzah, R. Abd Rahim, and Z. M. Noh. Sum of absolute differences algorithm in stereo correspondence problem for stereo matching in computer vision application. In *2010 3rd International Conference on Computer Science and Information Technology*, volume 1, pages 652–657. IEEE, 2010.

[108] S. R. Haque, R. Kormokar, and A. U. Zaman. Drone ground control station with enhanced safety features. In *2017 2nd International Conference for Convergence in Technology (I2CT)*, pages 1207–1210, 2017. doi: 10.1109/I2CT.2017.8226318.

[109] J. Hardy, J. Strader, J. N. Gross, Y. Gu, M. Keck, J. Douglas, and C. N. Taylor. Unmanned aerial vehicle relative navigation in GPS denied environments. In *2016 IEEE/ION Position, Location and Navigation Symposium (PLANS)*, pages 344–352. IEEE, 4 2016. doi: 10.1109/plans.2016.7479719. URL http://dx.doi.org/10.1109/PLANS.2016.7479719.

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

[111] W. Hess, D. Kohler, H. Rapp, and D. Andor. Real-time loop closure in 2d lidar slam. In *2016 IEEE international conference on robotics and automation (ICRA)*, pages 1271–1278. IEEE, 2016.

[112] Hexagon. Automated inspection | hexagon, 2025. URL `https://hexagon.com/solutions/automated-inspection`. (Accessed: 2025-01-15).

[113] H. Hirschmuller. Accurate and efficient stereo processing by semi-global matching and mutual information. In *2005 IEEE computer society conference on computer vision and pattern recognition (CVPR'05)*, volume 2, pages 807–814. IEEE, 2005.

[114] H. Hirschmuller. Stereo processing by semiglobal matching and mutual information. *IEEE Transactions on pattern analysis and machine intelligence*, 30(2):328–341, 2007.

[115] G. Hoffmann, D. Rajnarayan, S. Waslander, D. Dostal, J. Jang, and C. Tomlin. The stanford testbed of autonomous rotorcraft for multi agent control (starmac). In *Digital Avionics Systems Conference, DASC 04. The 23rd*, volume 2, pages 12–E. IEEE, 2004.

[116] A. Hornung, K. M. Wurm, M. Bennewitz, C. Stachniss, and W. Burgard. Octomap: An efficient probabilistic 3d mapping framework based on octrees. *Autonomous robots*, 34: 189–206, 2013.

[117] S. Hornung. Konzept einer bodenkontrollstation zur Überwachung der inspektion großer bauteile mithilfe eines quadcopters mit prototypischer evaluation. bachelor's thesis, 2021.

[118] S. Hornung. Realisierung eines systems zur automatisierten photogrammetrischen erfassung großer strukturen mithilfe von uavs. master's thesis, Augsburg University, 2024.

[119] S. Hrabar. 3d path planning and stereo-based obstacle avoidance for rotorcraft uavs. In *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 807–814, 2008. doi: 10.1109/IROS.2008.4650775.

[120] M. Hrúz, M. Bugaj, A. Novák, B. Kandera, and B. Badánik. The use of uav with infrared camera and rfid for airframe condition monitoring. *Applied Sciences*, 11(9), 2021. ISSN 2076-3417. doi: 10.3390/app11093737. URL `https://www.mdpi.com/2076-3417/11/9/3737`.

[121] P. P. Ikubanni, A. A. Adeleke, O. O. Agboola, C. T. Christopher, B. S. Ademola, J. Okonkwo, O. S. Adesina, P. O. Omoniyi, and E. T. Akinlabi. Present and future impacts of computer-aided design/computer-aided manufacturing (cad/cam). *Journal Européen des Systèmes Automatisés*, 55(3):349, 2022.

[122] E. International. The json data interchange syntax, 2025. URL `https://ecma-international.org/wp-content/uploads/ECMA-404_2nd_edition_december_2017.pdf`. (Accessed: 2025-02-22).

[123] N. Iversen, O. B. Schofield, L. Cousin, N. Ayoub, G. vom Bögel, and E. Ebeid. Design, integration and implementation of an intelligent and self-recharging drone system for autonomous power line inspection. In *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4168–4175, 2021. doi: 10.1109/IROS51168.2021.9635924.

[124] S. Ivić, B. Crnković, and I. Mezić. Ergodicity-based cooperative multiagent area coverage via a potential field. *IEEE transactions on cybernetics*, 47(8):1983–1993, 2016.

[125] S. Ivić, B. Crnković, L. Grbčić, and L. Matleković. Multi-uav trajectory planning for 3d visual inspection of complex structures. *Automation in Construction*, 147:104709, 2023.

[126] C. R. Jarvis. Operational experience with the electronic flight control systems of a lunar-landing research vehicle. Technical report, NASA Flight Research Center, Edwards, California, 1966.

[127] H. M. P. C. Jayaweera and S. Hanoun. A dynamic artificial potential field (d-apf) uav path planning technique for following ground moving targets. *IEEE Access*, 8:192760–192776, 2020.

[128] S. Jordan, J. Moore, S. Hovet, J. Box, J. Perry, K. Kirsche, D. Lewis, and Z. T. H. Tse. State-of-the-art technologies for uav inspections. *IET Radar, Sonar Navigation*, 12(2):151–164, 2018. doi: 10.1049/iet-rsn.2017.0251.

[129] S. J. Julier and J. K. Uhlmann. New extension of the kalman filter to nonlinear systems. In *Signal processing, sensor fusion, and target recognition VI*, volume 3068, pages 182–193. Spie, 1997.

[130] D. Jurado-Rodríguez, R. Muñoz-Salinas, S. Garrido-Jurado, F. J. Romero-Ramírez, and R. Medina-Carnicer. 3d model-based tracking combining edges, keypoints and fiducial markers. *Virtual Reality*, 27:3051 – 3065, 2023.

[131] I. Kalinov, E. Safronov, R. Agishev, M. Kurenkov, and D. Tsetserukou. High-precision uav localization system for landing on a mobile collaborative robot based on an ir marker pattern recognition. In *2019 IEEE 89th Vehicular Technology Conference (VTC2019-Spring)*, pages 1–6. IEEE, 2019.

[132] R. E. Kalman. A new approach to linear filtering and prediction problems. *Transactions of the ASME–Journal of Basic Engineering*, 82(Series D):35–45, 1960.

[133] E. D. Kaplan and C. Hegarty. *Understanding GPS/GNSS: principles and applications*. Artech house, 2017.

[134] P. Karaked, W. Saengphet, and S. Tantrairatn. Multi-sensor fusion with extended kalman filter for indoor localization system of multirotor uav. In *2022 19th International Joint Conference on Computer Science and Software Engineering (JCSSE)*, pages 1–5. IEEE, 2022.

[135] S. Karaman and E. Frazzoli. Sampling-based algorithms for optimal motion planning. *The International Journal of Robotics Research*, 30(7):846–894, 2011. doi: 10.1177/0278364911406761. URL https://doi.org/10.1177/0278364911406761.

[136] M. Karl and G. Ghazaei. Computer-implementiertes verfahren und vorrichtung zum prüfen einer korrektheit eines zusammenbaus, 2024. URL https://patents.google.com/patent/DE102023102196A1.

[137] R. Katschinsky. Drohnen-inspektion großer bauteile – positionsbestimmung und navigation relativ zu bewegten bauteilen. master's thesis, Augsburg University, 2021.

[138] L. Keyu, L. Yonggen, and Z. Yanchi. Dynamic obstacle avoidance path planning of uav based on improved apf. In *2020 5th International Conference on Communication, Image and Signal Processing (CCISP)*, pages 159–163, 2020. doi: 10.1109/CCISP51026.2020.9273463.

[139] A. Khalil, M. A. Jaradat, S. Mukhopadhyay, and M. F. Abdel-Hafez. Autonomous control of a hybrid rolling and flying caged drone for leak detection in hvac ducts. *IEEE/ASME Transactions on Mechatronics*, 29(1):366–378, 2024. doi: 10.1109/TMECH.2023.3279870.

[140] Y. Khosiawan and I. N. and. A system of uav application in indoor environment. *Production & Manufacturing Research*, 4(1):2–22, 2016. doi: 10.1080/21693277.2016.1195304. URL https://doi.org/10.1080/21693277.2016.1195304.

[141] S. Kim, D. Paes, K. Lee, J. Irizarry, and E. N. Johnson. Uas-based airport maintenance inspections: Lessons learned from pilot study implementation. In *ASCE International Conference on Computing in Civil Engineering 2019*, pages 382–389. American Society of Civil Engineers Reston, VA, 2019.

[142] Y. Kitamura, T. Tanaka, F. Kishino, and M. Yachida. 3-d path planning in a dynamic environment using an octree and an artificial potential field. In *Proceedings 1995 IEEE/RSJ International Conference on Intelligent Robots and Systems. Human Robot Interaction and Cooperative Robots*, volume 2, pages 474–481 vol.2, 1995. doi: 10.1109/IROS.1995.526259.

[143] S. Koenig and M. Likhachev. D*lite. In *Eighteenth National Conference on Artificial Intelligence*, page 476–483, USA, 2002. American Association for Artificial Intelligence. ISBN 0262511290.

[144] D. Koller, G. Klinker, E. Rose, D. Breen, R. Whitaker, and M. Tuceryan. Real-time vision-based camera tracking for augmented reality applications. In *Proceedings of the ACM Symposium on Virtual Reality Software and Technology*, VRST '97, page 87–94, New York, NY, USA, 1997. Association for Computing Machinery. ISBN 089791953X. doi: 10.1145/261135.261152. URL https://doi.org/10.1145/261135.261152.

[145] Y. Koren and J. Borenstein. Potential field methods and their inherent limitations for mobile robot navigation. In *Robotics and Automation, 1991. Proceedings., 1991 IEEE International Conference on*, pages 1398–1404. IEEE, 1991.

[146] C. Kownacki and L. Ambroziak. A new multidimensional repulsive potential field to avoid obstacles by nonholonomic uavs in dynamic environments. *Sensors*, 21(22):7495, 2021.

[147] E. F. Krause. Taxicab geometry. *Mathematics Teacher*, 66(8):695–706, 1973.

[148] Y. Labbé, L. Manuelli, A. Mousavian, S. Tyree, S. Birchfield, J. Tremblay, J. Carpentier, M. Aubry, D. Fox, and J. Sivic. Megapose: 6d pose estimation of novel objects via render and compare, 2022. URL https://arxiv.org/abs/2212.06870.

[149] S. Lange, N. Sunderhauf, and P. Protzel. A vision based onboard approach for landing and position control of an autonomous multirotor uav in gps-denied environments. In *Advanced Robotics, International Conference on*, pages 1–6. IEEE, 2009.

[150] S. M. LaValle. *Planning algorithms*. Cambridge university press, 2006.

[151] S. Lee and Y. Choi. Reviews of unmanned aerial vehicle (drone) technology trends and its applications in the mining industry. *Geosystem Engineering*, 19(4):197–204, 2016.

[152] R. Leishman. *A Vision-Based Relative Navigation Approach for Autonomous Multirotor Aircraft*. Ph.d. thesis, Brigham Young University, Provo, Utah, 2013. URL https://www.proquest.com/openview/a7067e1eae86d5df37df5ca849cc395b/1. (Accessed: 2025-04-13).

[153] J. R. Leiva, T. Villemot, G. Dangoumeau, M.-A. Bauda, and S. Larnier. Automatic visual detection and verification of exterior aircraft elements. In *2017 IEEE international workshop of electronics, control, measurement, signals and their application to mechatronics (ECMSM)*, pages 1–5. IEEE, 2017.

[154] S. Li, P. Durdevic, and Z. Yang. Hovering control for automatic landing operation of an inspection drone to a mobile platform. *IFAC-PapersOnLine*, 51(8):245–250, 2018. ISSN 2405-8963. doi: https://doi.org/10.1016/j.ifacol.2018.06.384. URL https://www.sciencedirect.com/science/article/pii/S2405896318307146. 3rd IFAC Workshop on Automatic Control in Offshore Oil and Gas Production OOGP 2018.

[155] Y. Li and C. Liu. Applications of multirotor drone technologies in construction management. *International Journal of Construction Management*, 19(5):401–412, 2019.

[156] L. Lifen, S. Ruoxin, L. Shuandao, and W. Jiang. Path planning for uavs based on improved artificial potential field method through changing the repulsive potential function. In *2016*

IEEE Chinese Guidance, Navigation and Control Conference (CGNCC), pages 2011–2015. IEEE, 2016.

[157] M. Likhachev, G. J. Gordon, and S. Thrun. Ara*: Anytime a* with provable bounds on sub-optimality. *Advances in neural information processing systems*, 16, 2003.

[158] M. Likhachev, D. I. Ferguson, G. J. Gordon, A. Stentz, and S. Thrun. Anytime dynamic a*: An anytime, replanning algorithm. In *ICAPS*, volume 5, pages 262–271, 2005.

[159] H. Lim, J. Park, D. Lee, and H. Kim. Build your own quadrotor: Open-source projects on unmanned aerial vehicles. *IEEE Robotics & Automation Magazine*, 19(3):33–45, 2012.

[160] H. Lim, J. Park, D. Lee, and H. J. Kim. Build your own quadrotor: Open-source projects on unmanned aerial vehicles. *IEEE Robotics & Automation Magazine*, 19(3):33–45, 2012.

[161] A. S. S. I. Limited. No title found, 2025. URL `https://rtmp.veriskope.com/pdf/rtmp_specification_1.0.pdf`. (Accessed: 2025-02-10).

[162] S. Lin and B. W. Kernighan. An effective heuristic algorithm for the traveling-salesman problem. *Operations research*, 21(2):498–516, 1973.

[163] L. Linsen. *Point cloud representation*. Univ., Fak. für Informatik, Bibliothek Technical Report, Faculty of Computer ..., 2001.

[164] W. Liu, D. Zou, D. Sartori, L. Pei, and W. Yu. *An Image-Guided Autonomous Navigation System for Multi-rotor UAVs*, pages 513–526. Springer Singapore, 2019. ISBN 9789811377587. doi: 10.1007/978-981-13-7759-4_45. URL `http://dx.doi.org/10.1007/978-981-13-7759-4_45`.

[165] Z. Liu, D. S. Forsyth, A. Marincak, and P. Vesley. Automated rivet detection in the eol image for aircraft lap joints inspection. *NDT & e International*, 39(6):441–448, 2006.

[166] P. Locht, K. Thomsen, and P. Mikkelsen. Full color image analysis as a tool for quality control and process development in the food industry. *ASAE Annual International Meeting*, page 15 pp., 1997. ISSN 0149-9890.

[167] D. G. Lowe. Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, 60:91–110, 2004.

[168] I. Luftronix. Technology | my website, 2025. URL `https://luftronix.com/technology`. (Accessed: 2025-04-15).

[169] K. M. Lynch and F. C. Park. *Modern robotics*. Cambridge University Press, 2017.

[170] O. Maghazei and T. Netland. Drones in manufacturing: exploring opportunities for research and practice. *Journal of Manufacturing Technology Management*, 31(6):1237–1259, 2020.

[171] O. Maghazei, T. H. Netland, D. Frauenberger, and T. Thalmann. Automatic drones for factory inspection: The role of virtual simulation. In A. Dolgui, A. Bernard, D. Lemoine, G. von Cieminski, and D. Romero, editors, *Advances in Production Management Systems. Artificial Intelligence for Sustainable and Resilient Production Systems*, pages 457–464, Cham, 2021. Springer International Publishing. ISBN 978-3-030-85910-7.

[172] I. Magic Leap. Github - magicleap/supergluepretrainednetwork: Superglue: Learning feature matching with graph neural networks (cvpr 2020, oral), 2025. URL `https://github.com/magicleap/SuperGluePretrainedNetwork`. (Accessed: 2025-05-13).

[173] Mainblades. Mainblades aircraft inspection automation, 2025. URL `https://www.mainblades.com/`. (Accessed: 2025-02-12).

[174] Mainblades. mainblades | solutions, 2025. URL https://www.mainblades.com/solutions. (Accessed: 2025-04-15).

[175] P. Maini, P. Tokekar, and P. B. Sujit. Visual monitoring of points of interest on a 2.5d terrain using a uav with limited field-of-view constraint. *IEEE Transactions on Aerospace and Electronic Systems*, 57(6):3661–3672, 2021. doi: 10.1109/TAES.2021.3082668.

[176] K. Malandrakis, A. Savvaris, J. A. G. Domingo, N. Avdelidis, P. Tsilivis, F. Plumacker, L. Z. Fragonara, and A. Tsourdos. Inspection of aircraft wing panels using unmanned aerial vehicles. In *2018 5th IEEE International Workshop on Metrology for AeroSpace (MetroAeroSpace)*, pages 56–61, 2018. doi: 10.1109/MetroAeroSpace.2018.8453598.

[177] K. Malandrakis, A. Savvaris, J. A. G. Domingo, N. Avdelidis, P. Tsilivis, F. Plumacker, L. Z. Fragonara, and A. Tsourdos. Inspection of aircraft wing panels using unmanned aerial vehicles. In *2018 5th IEEE International Workshop on Metrology for AeroSpace (MetroAeroSpace)*, pages 56–61, 6 2018. doi: 10.1109/MetroAeroSpace.2018.8453598.

[178] S. S. Mansouri, C. Kanellakis, E. Fresk, D. Kominiak, and G. Nikolakopoulos. Cooperative coverage path planning for visual inspection. *Control Engineering Practice*, 74:118–131, 2018.

[179] L. Marković, M. Kovač, R. Milijas, M. Car, and S. Bogdan. Error state extended kalman filter multi-sensor fusion for unmanned aerial vehicle localization in gps and magnetometer denied indoor environments. In *2022 International Conference on Unmanned Aircraft Systems (ICUAS)*, pages 184–190. IEEE, 2022.

[180] L. Matikainen, M. Lehtomäki, E. Ahokas, J. Hyyppä, M. Karjalainen, A. Jaakkola, A. Kukko, and T. Heinonen. Remote sensing methods for power line corridor surveys. *ISPRS Journal of Photogrammetry and Remote Sensing*, 119:10–31, 2016. ISSN 0924-2716. doi: https://doi.org/10.1016/j.isprsjprs.2016.04.011. URL https://www.sciencedirect.com/science/article/pii/S0924271616300697.

[181] A. Ma'Arif, W. Rahmaniar, M. A. M. Vera, A. A. Nuryono, R. Majdoubi, and A. Çakan. Artificial potential field algorithm for obstacle avoidance in uav quadrotor for dynamic environment. In *2021 IEEE International Conference on Communication, Networks and Satellite (COMNETSAT)*, pages 184–189, 2021. doi: 10.1109/COMNETSAT53002.2021.9530803.

[182] B. A. McElhoe. An assessment of the navigation and course corrections for a manned flyby of mars or venus. *IEEE Transactions on Aerospace and Electronic Systems*, AES-2(4): 613–623, 1966. doi: 10.1109/TAES.1966.4501892.

[183] D. Meagher. Geometric modeling using octree encoding. *Computer Graphics and Image Processing*, 19(2):129–147, 1982. ISSN 0146-664X. doi: https://doi.org/10.1016/0146-664X(82)90104-6. URL https://www.sciencedirect.com/science/article/pii/0146664X82901046.

[184] N. Mehreganian and A. S. Fallah. Blast loading effects on aircraft fuselage. In *Multiphysics Simulations in Automotive and Aerospace Applications*, pages 239–285. Elsevier, 2021.

[185] L. Meier, P. Tanskanen, F. Fraundorfer, and M. Pollefeys. Pixhawk: A system for autonomous flight using onboard computer vision. In *2011 ieee international conference on robotics and automation*, pages 2992–2997. IEEE, 2011.

[186] M. Memari, M. Shekaramiz, M. A. S. Masoum, and A. C. Seibi. Data fusion and ensemble learning for advanced anomaly detection using multi-spectral rgb and thermal imaging of

small wind turbine blades. *Energies*, 17(3), 2024. ISSN 1996-1073. doi: 10.3390/en17030673. URL https://www.mdpi.com/1996-1073/17/3/673.

[187] P. Merriaux, Y. Dupuis, R. Boutteau, P. Vasseur, and X. Savatier. A study of vicon system positioning performance. *Sensors*, 17(7):1591, 2017.

[188] S. Minaee, Y. Boykov, F. Porikli, A. Plaza, N. Kehtarnavaz, and D. Terzopoulos. Image segmentation using deep learning: A survey. *IEEE transactions on pattern analysis and machine intelligence*, 44(7):3523–3542, 2021.

[189] C. Mineo, C. MacLeod, M. Morozov, S. G. Pierce, T. Lardner, R. Summan, J. Powell, P. McCubbin, C. McCubbin, G. Munro, et al. Fast ultrasonic phased array inspection of complex geometries delivered through robotic manipulators and high speed data acquisition instrumentation. In *2016 IEEE International Ultrasonics Symposium (IUS)*, pages 1–4. IEEE, 2016.

[190] J. Miranda, S. Larnier, A. Herbulot, and M. Devy. UAV-based Inspection of Airplane Exterior Screws with Computer Vision. In *14h International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications.*, Prague, Czech Republic, Feb. 2019. URL https://laas.hal.science/hal-02065284.

[191] F. Mohamadi. Vertical takeoff and landing (vtol) small unmanned aerial system for monitoring oil and gas pipelines, 2014. URL https://patents.google.com/patent/US8880241B2/en.

[192] O. Moolan-Feroze, K. Karachalios, D. N. Nikolaidis, and A. Calway. Improving drone localisation around wind turbines using monocular model-based tracking. In *2019 international conference on robotics and automation (ICRA)*, pages 7713–7719. IEEE, 2019.

[193] O. Moolan-Feroze, K. Karachalios, D. N. Nikolaidis, and A. Calway. Simultaneous drone localisation and wind turbine model fitting during autonomous surface inspection. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2014–2021. IEEE, 2019.

[194] D. Moreno-Jacobo, G. Toledo-Nin, A. Ochoa-Zezzatti, V. Torres, and F. Estrada-Otero. *Evaluation of Drones for Inspection and Control in Industry 4.0*, pages 579–595. Springer International Publishing, Cham, 2021. ISBN 978-3-030-68655-0. doi: 10.1007/978-3-030-68655-0_29. URL https://doi.org/10.1007/978-3-030-68655-0_29.

[195] G. Morgenthal, N. Hallermann, J. Kersten, J. Taraben, P. Debus, M. Helmrich, and V. Rodehorst. Framework for automated uas-based structural condition assessment of bridges. *Automation in Construction*, 97:77–95, 2019. ISSN 0926-5805. doi: https://doi.org/10.1016/j.autcon.2018.10.006. URL https://www.sciencedirect.com/science/article/pii/S0926580518305156.

[196] D. Mourtzis, V. Siatras, and J. Angelopoulos. Real-time remote maintenance support based on augmented reality (ar). *Applied Sciences*, 10(5), 2020. ISSN 2076-3417. doi: 10.3390/app10051855. URL https://www.mdpi.com/2076-3417/10/5/1855.

[197] M. Nieuwenhuisen, M. Schadler, and S. Behnke. Predictive potential field-based collision avoidance for multicopters. *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, 40:293–298, 2013.

[198] P. Nooralishahi, C. Ibarra-Castanedo, S. Deane, F. López, S. Pant, M. Genest, N. P. Avdelidis, and X. P. Maldague. Drone-based non-destructive inspection of industrial sites: A review and case studies. *Drones*, 5(4):106, 2021.

[199] S. Numata, M. Koeda, K. Onishi, K. Watanabe, and H. Noborio. Performance and accuracy analysis of 3d model tracking for liver surgery. In *International Conference on Human-Computer Interaction*, pages 524–533. Springer, 2019.

[200] H. F. nv. Inspection flights - helicopterflights.be - helicopter company - air baptism - pilot training, 2025. URL https://helicopterflights.be/en/inspectievluchten-helikopter/. (Accessed: 2025-01-08).

[201] OctoMap. Octomap - 3d occupancy mapping, 2025. URL https://octomap.github.io/. (Accessed: 2025-01-05).

[202] E. Oland, R. Kristiansen, and J. T. Gravdahl. A comparative study of different control structures for flight control with new results. *IEEE Transactions on Control Systems Technology*, 28(2):291–305, 2020. doi: 10.1109/TCST.2018.2873507.

[203] E. Olson. Apriltag: A robust and flexible visual fiducial system. In *2011 IEEE international conference on robotics and automation*, pages 3400–3407. IEEE, 2011.

[204] E. Olson. Apriltag: A robust and flexible visual fiducial system. In *2011 IEEE international conference on robotics and automation*, pages 3400–3407. IEEE, 2011.

[205] OpenCV. Opencv - open computer vision library, 2025. URL https://opencv.org/. (Accessed: 2025-02-21).

[206] J. O'Rourke. *Art Gallery Theorems and Algorithms*. Oxford University Press, Inc., New York, NY, USA, 1987. ISBN 0-19-503965-3.

[207] R. Padovani, E. A. de Souza, and D. L. De Campos. Methods and systems for the automatic quality inspection of materials using infrared radiation, 2024. URL https://patents.google.com/patent/US20240221143A1.

[208] U. Papa and S. Ponte. Preliminary design of an unmanned aircraft system for aircraft general visual inspection. *Electronics*, 7(12):435, 2018.

[209] S. Papaioannou, P. Kolios, T. Theocharides, C. G. Panayiotou, and M. M. Polycarpou. Uav-based receding horizon control for 3d inspection planning. In *2022 International Conference on Unmanned Aircraft Systems (ICUAS)*, page 1121–1130. IEEE, June 2022. doi: 10.1109/icuas54217.2022.9836051. URL http://dx.doi.org/10.1109/ICUAS54217.2022.9836051.

[210] A. Paris, B. T. Lopez, and J. P. How. Dynamic landing of an autonomous quadrotor on a moving platform in turbulent wind conditions. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 9577–9583. IEEE, 2020.

[211] R. Pavlyuk, V. Kozak, K. Sonnenleiter, and D. Spell. Fusion of vision and depth sensors for navigation in complex environments, 2020. URL https://patents.google.com/patent/US10527423B1.

[212] J. Pestana, J. L. Sanchez-Lopez, P. Campoy, and S. Saripalli. Vision based GPS-denied Object Tracking and following for unmanned aerial vehicles. In *2013 IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR)*, pages 1–6. IEEE, 10 2013. doi: 10.1109/ssrr.2013.6719359. URL http://dx.doi.org/10.1109/SSRR.2013.6719359.

[213] L. Pettersson and C. Lundell Johansson. Ant colony optimization - optimal number of ants, 2018.

[214] PIX4D. Professional photogrammetry and drone mapping software - pix4d, 2025. URL https://www.pix4d.com/product/pix4dmapper-photogrammetry-software/. (Accessed: 2025-04-14).

[215] A. Plastropoulos, K. Bardis, G. Yazigi, N. P. Avdelidis, and M. Droznika. Aircraft skin machine learning-based defect detection and size estimation in visual inspections. *Technologies*, 12(9):158, 2024.

[216] R. Popall. Lokale kollisionsvermeidung von drohnen mit eingeschränkter sensorik in unbekannten umgebungen. bachelor's thesis, 2021.

[217] C. Potena, D. Nardi, and A. Pretto. Effective target aware visual navigation for UAVs. In *2017 European Conference on Mobile Robots (ECMR)*, pages 1–7. IEEE, 9 2017. doi: 10.1109/ecmr.2017.8098714. URL http://dx.doi.org/10.1109/ECMR.2017.8098714.

[218] R. C. Prim. Shortest connection networks and some generalizations. *The Bell System Technical Journal*, 36(6):1389–1401, 1957. doi: 10.1002/j.1538-7305.1957.tb01515.x.

[219] D. Project. Mavlink developer guide, 2025. URL https://mavlink.io/en/. (Accessed: 2025-03-24).

[220] PTC. Advanced views - vuforia engine library, 2025. URL https://developer.vuforia.com/library/vuforia-engine/images-and-objects/model-targets/guide-views/advanced-views/. (Accessed: 2025-03-24).

[221] PTC. Cylinder targets - vuforia engine library, 2025. URL https://developer.vuforia.com/library/vuforia-engine/images-and-objects/cylinder-targets/cylinder-targets/. (Accessed: 2025-03-24).

[222] PTC. Vuforia engine | create ar apps and ar experiences | ptc, 2025. URL https://www.ptc.com/en/products/vuforia/vuforia-engine. (Accessed: 2025-03-24).

[223] PTC. Guide view - vuforia engine library, 2025. URL https://developer.vuforia.com/library/vuforia-engine/images-and-objects/model-targets/guide-views/model-target-guide-view/. (Accessed: 2025-03-24).

[224] PTC. Supported versions - vuforia engine library, 2025. URL https://developer.vuforia.com/library/vuforia-engine/platform-support/supported-versions/. (Accessed: 2025-03-24).

[225] PTC. Model target generator - vuforia engine library, 2025. URL https://developer.vuforia.com/library/vuforia-engine/images-and-objects/model-targets/model-target-generator/model-target-generator-user-guide/. (Accessed: 2025-03-24).

[226] PTC. Model targets - vuforia engine library, 2025. URL https://developer.vuforia.com/library/objects/model-targets. (Accessed: 2025-03-24).

[227] PTC. Multi targets - vuforia engine library, 2025. URL https://developer.vuforia.com/library/vuforia-engine/images-and-objects/multi-targets/multi-targets/. (Accessed: 2025-03-24).

[228] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng. Ros: an open-source robot operating system. In *ICRA workshop on open source software*, volume 3, page 5. Kobe, 2009.

[229] T. Raj, F. H. Hashim, A. B. Huddin, M. F. Ibrahim, and A. Hussain. A survey on lidar scanning mechanisms. *Electronics*, 9(5), 2020. ISSN 2079-9292. doi: 10.3390/electronics9050741. URL https://www.mdpi.com/2079-9292/9/5/741.

[230] P. Ramon-Soria, M. Perez-Jimenez, B. C. Arrue, and A. Ollero. Planning system for integrated autonomous infrastructure inspection using uavs. In *2019 Interna-*

tional Conference on Unmanned Aircraft Systems (ICUAS), pages 313–320, 2019. doi: 10.1109/ICUAS.2019.8797874.

[231] N. Razali, M. Sultan, F. Mustapha, N. Yidris, and M. Ishak. Impact damage on composite structures–a review. *Int. J. Eng. Sci*, 3(7):08–20, 2014.

[232] J. Ren, D.-X. Liu, K. Li, J. Liu, Y. Feng, and X. Lin. Cascade pid controller for quadrotor. In *2016 IEEE International Conference on Information and Automation (ICIA)*, pages 120–124, 2016. doi: 10.1109/ICInfA.2016.7831807.

[233] Renfrow, Liebler, and Denham. F-14 flight control law design, verification, and validation using computer aided engineering tools. In *1994 Proceedings of IEEE International Conference on Control and Applications*, pages 359–364. IEEE, 1994.

[234] D. A. Rodríguez, C. Lozano Tafur, P. F. Melo Daza, J. A. Villalba Vidales, and J. C. Daza Rincón. Inspection of aircrafts and airports using uas: A review. *Results in Engineering*, 22:102330, 2024. ISSN 2590-1230. doi: https://doi.org/10.1016/j.rineng.2024.102330. URL https://www.sciencedirect.com/science/article/pii/S2590123024005851.

[235] M. K. Rohil, S. Mahajan, and T. Paul. An architecture to intertwine augmented reality and intelligent tutoring systems: towards realizing technology-enabled enhanced learning. *Education and Information Technologies*, pages 1–30, 2024.

[236] W. Rone and P. Ben-Tzvi. Mapping, localization and motion planning in mobile multi-robotic systems. *Robotica*, 31(1):1–23, 2013.

[237] S. D. Roth. Ray casting for modeling solids. *Computer graphics and image processing*, 18 (2):109–144, 1982.

[238] L. Ruiqian, X. Juan, and Z. Hongfu. Automated surface defects acquisition system of civil aircraft based on unmanned aerial vehicles. In *2020 IEEE 2nd International Conference on Civil Aviation Safety and Information Technology (ICCASIT*, pages 729–733, 2020. doi: 10.1109/ICCASIT50869.2020.9368540.

[239] F. SA. Elios 3 - digitizing the inaccessible, 2025. URL https://www.flyability.com/elios-3. (Accessed: 2025-04-08).

[240] F. SA. Brochure: Elios 3 - technical specifications, 2025. URL https://www.flyability.com/de/elios-3-documentation-collection. (Only available upon request. Accessed: 2025-04-08).

[241] F. SA. Ultimate guide to wind turbine inspection techniques, 2025. URL https://www.flyability.com/blog/wind-turbine-inspection. (Accessed: 2025-01-09).

[242] L. V. Santana, A. S. Brandao, M. Sarcinelli-Filho, and R. Carelli. A trajectory tracking and 3d positioning controller for the ar. drone quadrotor. In *2014 international conference on unmanned aircraft systems (ICUAS)*, pages 756–767. IEEE, 2014.

[243] J. Saupe. Robuste navigation mit hindernisvermeidung bei der drohnengestützten inspektion großer baugruppen. bachelor's thesis, 2024.

[244] D. Schmidt and K. Berns. Climbing robots for maintenance and inspections of vertical structures—a survey of design aspects and technologies. *Robotics and Autonomous Systems*, 61(12):1288–1305, 2013.

[245] F. Schmitt, M. Schmitt, S. Grohmann, B. Audenrith, and L. Giesler. Procedure and arrangement for the quality inspection of an object, 2023. URL https://patents.google.com/patent/DE102020134680B4.

[246] M. Schörner, R. Katschinsky, C. Wanninger, A. Hoffmann, and W. Reif. Towards fully automated inspection of large components with uavs: Offline path planning and view angle dependent optimization strategies. In O. Gusikhin, K. Madani, and J. Zaytoon, editors, *Informatics in Control, Automation and Robotics*, pages 105–123, Cham, 2022. Springer International Publishing. ISBN 978-3-030-92442-3.

[247] M. Schörner, C. Wanninger, R. Katschinsky, S. Hornung, C. Eymüller, A. Poeppel, and W. Reif. Uav inspection of large components: determination of alternative inspection points and online route optimization. In *2023 IEEE/ACM 5th International Workshop on Robotics Software Engineering (RoSE), 15 May 2023, Melbourne, Australia*, pages 45 – 52, 2023. doi: 10.1109/RoSE59155.2023.00012.

[248] M. Schörner, M. Bettendorf, C. Wanninger, A. Hoffmann, and W. Reif. Uav inspection of large components: Indoor navigation relative to structures. In *Proceedings of the 18th International Conference on Informatics in Control, Automation and Robotics - ICINCO*, pages 179–186. INSTICC, SciTePress, 2021. ISBN 978-989-758-522-7. doi: 10.5220/0010556301790186.

[249] D.-J. I. Science and L. Technology Co. *DJI Mavic 3E / 3T User Manual*, 2022. URL https://dl.djicdn.com/downloads/DJI_Mavic_3_Enterprise/20221115/DJI_Mavic_3E_3T_User_Manual_DE.pdf. (Accessed: 2025-04-04).

[250] D.-J. I. Science and L. Technology Co. Mavic 2 - product information - dji, 2025. URL https://www.dji.com/mavic-2/info. (Accessed: 2025-03-24).

[251] D.-J. I. Science and L. Technology Co. Support for mavic 2 enterprise - dji, 2025. URL https://www.dji.com/de/support/product/mavic-2-enterprise. (Accessed: 2025-03-24).

[252] D.-J. I. Science and L. Technology Co. Technical data - dji mavic 3 enterprise series - dji enterprise, 2025. URL https://enterprise.dji.com/de/mavic-3-enterprise/specs. (Accessed: 2025-03-24).

[253] D.-J. I. Science and L. Technology Co. Dji developer, 2025. URL https://developer.dji.com/mobile-sdk-v4/. (Accessed: 2025-03-24).

[254] D.-J. I. Science and L. Technology Co. Github - dji-sdk/mobile-sdk-android-v5: Msdk v5 sample, 2025. URL https://github.com/dji-sdk/Mobile-SDK-Android-V5. (Accessed: 2025-03-24).

[255] D.-J. I. Science and L. Technology Co. Support for matrice 200 series - dji, 2025. URL https://www.dji.com/support/product/matrice-200-series. (Accessed: 2025-04-14).

[256] D.-J. I. Science and L. Technology Co. Support für matrice 300 rtk - dji, 2025. URL https://www.dji.com/support/product/matrice-300. (Accessed: 2025-02-10).

[257] D.-J. I. Science and L. Technology Co. Dji mini 4 pro product page - dji, 2025. URL https://www.dji.com/de/mini-4-pro. (Accessed: 2025-03-24).

[258] A. SE. Orders and deliveries | airbus, 2025. URL https://www.airbus.com/en/products-services/commercial-aircraft/orders-and-deliveries. (Accessed: 2025-01-30).

[259] A. SE. Robotics | airbus, 2025. URL https://www.airbus.com/en/innovation/digital-transformation/industry-4-0/robotics. (Accessed: 2025-02-24).

[260] J. Seo, L. Duque, and J. Wacker. Drone-enabled bridge inspection methodology and application. *Automation in construction*, 94:112–126, 2018.

[261] S. Shen, Y. Mulgaonkar, N. Michael, and V. Kumar. Vision-based state estimation for autonomous rotorcraft mavs in complex environments. In *2013 IEEE International Conference on Robotics and Automation*, pages 1758–1764. IEEE, 2013.

[262] R. Sihombing and V. Coors. *Object-Based Mobile Augmented Reality for a 3D Model*, pages 646–655. Deutsche Gesellschaft für Photogrammetrie, Fernerkundung und Geoinformation e.V., 03 2018.

[263] G. L. Smith, S. F. Schmidt, and L. A. McGee. *Application of statistical filter theory to the optimal estimation of position and velocity on board a circumlunar vehicle*, volume 135. National Aeronautics and Space Administration, 1962.

[264] S. L. Smith and F. Imeson. Glns: An effective large neighborhood search heuristic for the generalized traveling salesman problem. *Computers & Operations Research*, 87:1–19, 2017.

[265] M. Stokkeland, K. Klausen, and T. A. Johansen. Autonomous visual navigation of unmanned aerial vehicle for wind turbine inspection. In *2015 International Conference on Unmanned Aircraft Systems (ICUAS)*, pages 998–1007. IEEE, 2015.

[266] T. Stützle, M. Dorigo, et al. Aco algorithms for the traveling salesman problem. *Evolutionary algorithms in engineering and computer science*, 4:163–183, 1999.

[267] V. Sudevan, A. Shukla, and H. Karki. Current and future research focus on inspection of vertical structures in oil and gas industry. In *2018 18th International Conference on Control, Automation and Systems (ICCAS)*, pages 144–149. IEEE, 2018.

[268] J. Sun, J. Tang, and S. Lao. Collision avoidance for cooperative uavs with optimized artificial potential field algorithm. *IEEE Access*, 5:18382–18390, 2017.

[269] D. Systèmes. Delmia augmented experience for accurate augmented reality quality inspection | dassault systèmes, 2025. URL `https://www.3ds.com/products/delmia/augmented-experience/quality-inspection`. (Accessed: 2025-03-14).

[270] G. Szafranski and R. Czyba. Different approaches of pid control uav type quadrotor. In *International Micro Air Vehicles conference summer edition, Proceedings of the*, 2011.

[271] R. Szeliski. *Computer vision: algorithms and applications*. Springer Nature, 2022.

[272] M. Tan, R. Pang, and Q. V. Le. Efficientdet: Scalable and efficient object detection. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 10781–10790, 2020.

[273] U. Technologies. Unity real-time development platform - 3d, 2d, vr and ar engine, 2025. URL `https://unity.com`. (Accessed: 2025-03-16).

[274] U. Technologies. Github - unity-technologies/ros-tcp-connector, 2025. URL `https://github.com/Unity-Technologies/ROS-TCP-Connector`. (Accessed: 2025-03-24).

[275] D. Tenorio, V. Rivera, J. Medina, A. Leondar, M. Gaumer, and Z. Dodds. Visual autonomy via 2d matching in rendered 3d models. In *Advances in Visual Computing: 11th International Symposium, ISVC 2015, Las Vegas, NV, USA, December 14-16, 2015, Proceedings, Part I 11*, pages 373–385. Springer, 2015.

[276] W. Tong. *Wind power generation and wind turbine design*. WIT press, 2010.

[277] V. Tzitzilonis, K. Malandrakis, L. Zanotti Fragonara, J. A. Gonzalez Domingo, N. P. Avdelidis, A. Tsourdos, and K. Forster. Inspection of aircraft wing panels using unmanned aerial vehicles. *Sensors*, 19(8):1824, 2019.

[278] M. Uenohara and T. Kanade. Vision-based object registration for real-time image overlay. *Computers in Biology and Medicine*, 25(2):249–260, 1995.

[279] V. M. S. L. UK. Vicon Object Tracking, 2025. URL https://www.vicon.com/motion-capture/engineering. (Accessed: 2025-03-08).

[280] V. M. S. L. UK. Vicon v16 camera specifications - vantage documentation - vicon help, 2025. URL https://help.vicon.com/space/Vantage/15041618/V16+camera+specifications. (Accessed: 2025-03-08).

[281] V. M. S. L. UK. Vicon v5 camera specifications - vantage documentation - vicon help, 2025. URL https://help.vicon.com/space/Vantage/15042553/V5+camera+specifications. (Accessed: 2025-03-08).

[282] S. Ullman. The interpretation of structure from motion. *Proceedings of the Royal Society of London. Series B. Biological Sciences*, 203(1153):405–426, 1979.

[283] I. Đurić, I. Vasiljević, M. Obradović, V. Stojaković, J. Kićanović, and R. Obradović. Comparative analysis of open-source and commercial photogrammetry software for cultural heritage. In *Proceedings of the IneCAADe 2021 International Scientific Conference*, pages 8–10, 2021.

[284] F. S. S. U.S. Department Of Transportation, Federal Aviation Administration. *Aircraft inspection for the general aviation aircraft owner*, 1978. URL https://www.faa.gov/regulations_policies/advisory_circulars/index.cfm/go/document.information/documentID/22051.

[285] B. Veritas. Inspektion von windkraftanlagen | bureau veritas, 2025. URL https://www.bureauveritas.de/unsere-services/inspektion-von-windkraftanlagen. (Accessed: 2025-03-24).

[286] Visometry. Visionlib augmented reality tracking for industries, 2025. URL https://visionlib.com/. (Accessed: 2025-03-24).

[287] Visometry. vlsdk: Visometry.visionlib.sdk.examples.existencecheck class reference, 2025. URL https://docs.visionlib.com/v3.2.2/api_reference/class_visometry_1_1_vision_lib_1_1_s_d_k_1_1_examples_1_1_existence_check.html. (Accessed: 2025-04-21).

[288] G. Voronoi. Nouvelles applications des paramètres continus à la théorie des formes quadratiques. deuxième mémoire. recherches sur les parallélloèdres primitifs. *Journal für die reine und angewandte Mathematik (Crelles Journal)*, 1908(134):198–287, 1908.

[289] G. F. Voronoı. Deuxieme mémoire: recherches sur les paralléloedres primitifs. *J. reine angew. Math*, 136:67–181, 1909.

[290] V. Walter, N. Staub, A. Franchi, and M. Saska. Uvdar System for Visual Relative Localization With Application to Leader–Follower Formations of Multirotor UAVs. *IEEE Robotics and Automation Letters*, 4(3):2637–2644, 7 2019. ISSN 2377-3766. doi: 10.1109/lra.2019.2901683. URL http://dx.doi.org/10.1109/LRA.2019.2901683.

[291] H. Wang, H. Zhou, H. Liu, Z. Huang, and M. Feng. Research on determining the inspection point of multirotor uav power tower. *Mathematical Problems in Engineering*, 2021(1): 8894055, 2021. doi: https://doi.org/10.1155/2021/8894055. URL https://onlinelibrary.wiley.com/doi/abs/10.1155/2021/8894055.

[292] Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli. Image quality assessment: from error visibility to structural similarity. *IEEE transactions on image processing*, 13(4): 600–612, 2004.

[293] C. Wanninger, R. Katschinsky, A. Hoffmann, M. Schörner, and W. Reif. Towards fully automated inspection of large components with uavs: Offline path planning. In *Proceedings of the 17th International Conference on Informatics in Control, Automation and Robotics - ICINCO*, pages 71–80. INSTICC, SciTePress, 2020. ISBN 978-989-758-442-8. doi: 10.5220/0009887900710080.

[294] Y. Watanabe, A. Calise, and E. Johnson. Vision-based obstacle avoidance for uavs. In *AIAA guidance, navigation and control conference and exhibit*, page 6829, 2007.

[295] S. Weiss, M. W. Achtelik, S. Lynen, M. Chli, and R. Siegwart. Real-time onboard visual-inertial state estimation and self-calibration of mavs in unknown environments. In *2012 IEEE international conference on robotics and automation*, pages 957–964. IEEE, 2012.

[296] A. C. Woods and H. M. La. Dynamic target tracking and obstacle avoidance using a drone. In *Advances in Visual Computing: 11th International Symposium, ISVC 2015, Las Vegas, NV, USA, December 14-16, 2015, Proceedings, Part I 11*, pages 857–866. Springer, 2015.

[297] J. N. Yasin, S. A. Mohamed, M.-H. Haghbayan, J. Heikkonen, H. Tenhunen, and J. Plosila. Unmanned aerial vehicles (uavs): Collision avoidance systems and approaches. *IEEE access*, 8:105139–105155, 2020.

[298] Y. D. Yasuda, F. A. Cappabianco, L. E. G. Martins, and J. A. Gripp. Aircraft visual inspection: A systematic literature review. *Computers in Industry*, 141:103695, 2022. ISSN 0166-3615. doi: https://doi.org/10.1016/j.compind.2022.103695. URL https://www.sciencedirect.com/science/article/pii/S0166361522000926.

[299] A. Zacharia, S. Papaioannou, P. Kolios, and C. Panayiotou. Distributed control for 3d inspection using multi-uav systems. In *2023 31st Mediterranean Conference on Control and Automation (MED)*, pages 164–169, 2023. doi: 10.1109/MED59994.2023.10185881.

[300] Zeiss. Inspection and quality control applications | capture 3d, 2025. URL https://www.capture3d.com/applications/inspection-quality-control. (Accessed: 2025-03-24).

[301] C. Zhang and J. M. Kovacs. The application of small unmanned aerial systems for precision agriculture: a review. *Precision Agriculture*, 13(6):693–712, 12 2012. ISSN 1573-1618. doi: 10.1007/s11119-012-9274-5. URL https://doi.org/10.1007/s11119-012-9274-5.

[302] J. Zhang, L. Liu, B. Wang, X. Chen, Q. Wang, and T. Zheng. High speed automatic power line detection and tracking for a uav-based inspection. In *2012 Intern. Conf. on Industrial Control and Electronics Engineering*, pages 266–269, 2012. doi: 10.1109/ICICEE.2012.77.

[303] F. Zhao, Q. Huang, and W. Gao. Image matching by normalized cross-correlation. In *2006 IEEE International Conference on Acoustics Speech and Signal Processing Proceedings*, volume 2, pages II–II, 2006. doi: 10.1109/ICASSP.2006.1660446.

[304] Y. N. Zolotukhin, K. Y. Kotov, A. Mal'tsev, et al. Using the kalman filter in the quadrotor vehicle trajectory tracking system. *Optoelectronics, Instrumentation and Data Processing*, 49(6):536–545, 2013.

[305] J.-T. Zou and R. G. V. Drone-based solar panel inspection with 5g and ai technologies. In *2022 8th International Conference on Applied System Innovation (ICASI)*, pages 174–178, 2022. doi: 10.1109/ICASI55125.2022.9774462.

List of Figures

List of Tables

Supervised Theses

- Raphael Katschinsky. “Drohnen-Inspektion großer Bauteile - Trajektorienplanung mit vollständiger Sichtabdeckung”. Bachelorarbeit. Universität Augsburg, 2019

- Moritz Vogelsang. “Drohnen-Inspektion großer Bauteile – Relative Orientierung zur Laufzeit”. Bachelorarbeit. Universität Augsburg, 2019

- Michelle Bettendorf. “UAV inspection of large components - Adaptive navigation at runtime”. Bachelorarbeit. Universität Augsburg, 2020

- Simon Hornung. “Konzept einer Bodenkontrollstation zur Überwachung der Inspektion großer Bauteile mithilfe eines Quadcopters mit prototypischer Evaluation”. Bachelorarbeit. Universität Augsburg, 2021

- Raphael Katschinsky. “Drohnen-Inspektion großer Bauteile - Positionsbestimmung und Navigation relativ zu bewegten Bauteilen”. Masterarbeit. Universität Augsburg, 2021

- Niklas Vogel. “Drohnenhandygimbal für QUAD mit automatischer Blickwinkelberechnung”. Bachelorarbeit. Universität Augsburg, 2021

- Raban Popall. “Lokale Kollisionsvermeidung von Drohnen mit eingeschränkter Sensorik in unbekannten Umgebungen”. Bachelorarbeit. Universität Augsburg, 2021

- Marko Persic. “Bewertungsgrundlagen und Algorithmen für Graphen im Semantic Web”. Bachelorarbeit. Universität Augsburg, 2022

- Jan Dräger. “Realisierung einer Simulationsumgebung für drohnengestützte Inspektionsflüge in ROS2”. Bachelorarbeit. Universität Augsburg, 2024

- Julian Saupe. “Robuste Navigation mit Hindernisvermeidung bei der drohnengestützen Inspektion großer Baugruppen”. Bachelorarbeit. Universität Augsburg, 2024

- Simon Hornung. “Realisierung eines Systems zur automatisierten photogrammetrischen Erfassung großer Strukturen mithilfe von UAVs”. Masterarbeit. Universität Augsburg, 2024

Own Publications

1. Martin Schörner, Constantin Wanninger, Raphael Katschinsky, Simon Hornung, Christian Eymüller, Alexander Poeppel and Wolfgang Reif. "UAV inspection of large components: determination of alternative inspection points and online route optimization". In 2023 IEEE/ACM 5th International Workshop on Robotics Software Engineering (RoSE), 15 May 2023, Melbourne, Australia. IEEE, Piscataway, NJ, 45-52 DOI: 10.1109/RoSE59155.2023.00012
2. Martin Schörner, Raphael Katschinksy, Constantin Wanninger, Alwin Hoffmann and Wolfgang Reif. "Towards fully automated inspection of large components with UAVs: offline path planning and view angle dependent optimization strategies". In Oleg Gusikhin, Kurosh Madani and Janan Zaytoon (Ed.). Informatics in Control, Automation and Robotics: 17th International Conference, ICINCO 2020, Lieusaint - Paris, France, July 7–9, 2020, Revised Selected Papers. Springer, Cham (Lecture Notes in Electrical Engineering ; 793), 105-123. DOI: 10.1007/978-3-030-92442-3_7
3. Martin Schörner, Constantin Wanninger, Alwin Hoffmann, Oliver Kosak and Wolfgang Reif. "Architecture for emergency control of autonomous UAV ensembles". In 3rd International Workshop on Robotic Software Engineering (RoSE'21), co-located with ICSE 2021, Virtual, Madrid, Spain, May 23 – 29, 2021. IEEE, Piscataway, NJ, 41-46 DOI: 10.1109/RoSE52553.2021.00014
4. Constantin Wanninger, Sebastian Rossi, Martin Schörner, Alwin Hoffmann, Alexander Poeppel, Christian Eymüller and Wolfgang Reif. "ROSSi a graphical programming interface for ROS 2". In 2021 21st International Conference on Control, Automation and Systems (ICCAS), 12-15 October 2021, Jeju, Republic of Korea. IEEE, Piscataway, NJ, 255-262 DOI: 10.23919/ICCAS52745.2021.9649736
5. Constantin Wanninger, Luca Alfano, Martin Schörner, Alwin Hoffmann, Oliver Kosak and Wolfgang Reif "Semantic plug and play: an architecture combining linked data and reconfigurable hardware". In Dick Bulterman, Atsushi Kitazawa, David Ostrowski, Phillip Sheu and Jeffrey Tsai (Ed.). 2021 IEEE 15th International Conference on Semantic Computing (ICSC), 27-29 January 2021, Laguna Hills, CA, USA, virtual event. IEEE, Piscataway, NJ, 203-206. DOI: 10.1109/icsc50631.2021.00043
6. Martin Schörner, Michelle Bettendorf, Constantin Wanninger, Alwin Hoffmann and Wolfgang Reif. "UAV inspection of large components: indoor navigation relative to structures". In Oleg Gusikhin, Henk Nijmeijer and Kurosh Madani (Ed.). Proceedings of the 18th International Conference on Informatics in Control, Automation and Robotics - ICINCO, July 6-8, 2021. SciTePress, Setúbal, 179-186. DOI: 10.5220/0010556301790186
7. Martin Schörner, Constantin Wanninger, Alwin Hoffmann, Oliver Kosak, Hella Ponsar and Wolfgang Reif. "Modeling and execution of coordinated missions in reconfigurable robot ensembles". In Davide Brugali, Jean-Claude Latombe,

Phillip Sheu, Rouh-Mei Hu (Eds.). IEEE Robotic Computing IRC 2020 - The Fourth IEEE International Conference on Robotic Computing, 9-11 November 2020, Taichung, Taiwan. IEEE, Piscataway, NJ, 290-293 DOI: 10.1109/IRC.2020.00053

8. Constantin Wanninger, Raphael Katschinksy, Alwin Hoffmann, Martin Schörner and Wolfgang Reif. "Towards fully automated inspection of large components with UAVs: offline path planning". In Oleg Gusikhin, Kurosh Madani and Janan Zaytoon (Ed.). Proceedings of the 17th International Conference on Informatics in Control, Automation and Robotics - ICINCO, July 7-9, 2020. SciTePress, Setúbal, 71-80. DOI: 10.5220/0009887900710080