This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 861111

Drones4Safety

Research & Innovation Action (RIA)

Inspection Drones for Ensuring Safety in Transport Infrastructures

Test and Validation Report
D5.4

Due date of deliverable: 31.05.2021 (M24)

Start date of project: June 1st, 2020

Type: Deliverable
WP number: WP5

Responsible institution: Syddansk Universitet (SDU)

Editor and editor's address: Rune Hylsberg Jacobsen (rhj@ece.au.dk)

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 861111

Version 1.0
Release Date: May 31, 2021

Project funded by the European Commission within the Horizon 2020 Programme		
Dissemination Level		
PU	Public	✓
CO	Confidential, only for members of the consortium (including the Commission Services)	

Authors/Contributors

The following members of the project have contributed to the content and writing of this deliverable:

- Rune Hylsberg Jacobsen (AU, editor)
- Kaspar Hageman (AU)
- Lea Matlekovic (SDU)
- Liping Shi (AU)
- Emad Samuel Malki Ebeid (SDU)

Change Log

Rev.	Date	Who	Site	Change
0.1	01/06/2020	Annika Lindberg	SDU	Created initial version.
0.2	2022-05-10	Rune H Jacobsen	AU	Document outline updated after WP5 meeting.
0.3	2022-05-17	Kaspar Hageman	AU	Added "Group management in Flying Ad Hoc Networks" (4.1.1) and "Charging Protocol" (4.3.3) sections
0.4	2022-05-17	Rune H Jacobsen, Lea Matlekovic	AU, SDU	Added section on LoRa communication (Sec. 3.1) and 5G characterization (Sec. 3.5.2). Reviewed Sec. 4.1 Secure group management protocol. Created Appendix A. Added section on global path planning (Sec. 4.2.1) and edited Sec. 4.3 Task allocation
0.5	2022-05-22	Rune H. Jacobsen	AU	Introduction added (Sec. 1). Added description of validation environments (Sec. 2.2)
0.6	2022-05-23	Liping Shi	AU	Added content according to comments.
0.7	2022-05-23	Rune H. Jacobsen	AU	Added content regarding the Boid model for formation flying.
0.8	2022-05-27	Rune H. Jacobsen	AU	Version prepared for review.
1.0	2022-05-30	Rune H. Jacobsen	AU	Final version. Updated after internal review.

1 Executive Summary

Deliverable D5.4 reports on research and innovation for the collaborative multi-drone system designed in WP5 of the Drones4Safety project. It focuses on the documentation of the results from the test and validation activities in WP5 with inputs from Task 5.1, 5.2, 5.3, and 5.4.

The report introduces the drone hardware and the test and validation environments. The drone platforms used in the project are custom built from state-of-the-art robotics components. The inspection and harvesting drone support the design and validation of autonomous inspection with AI and the energy harvesting module whereas the swarming drone prototype platform has been used for the design and validation of collaborative functions of the multi drone system. In addition, the report introduces the test and validation environments including simulation environments as well as indoor and outdoor test facilities. A mockup of a bridge was built in the indoor test facility based on Leonardo da Vinci's famous bridge design.

The presentation of the results of the test and validation activities in WP5 are divided in two main categories addressing 1) communications of the multi-drone system and 2) algorithms to support swarming functions.

The deliverable provides an elaborate test of Long-Range radio communication (LoRa) based on the latest LoRa chipset operating in the license free 2.45 GHz band. We find that a drone-to-ground communication channel of at least 17.6 kbps for distances up to 2.4 km can be supported. Using spreading factors of $SF \leq 8$ can provide upper limits of latency of less than 50 ms. This means that LoRa can be used to support a command-and-control (C2) communication channels supporting telemetry data exchange. Drone-to-drone communication requires higher data rates than LoRa can support.

We have investigated the use of Robot Operating System (ROS) and ROS2 communication over WiFi (IEEE 802.11) operating WiFi in both ad hoc and infrastructure mode. Measurements of the inter process communication with varying ROS message sizes shows that ROS2 was able to deliver messages with low latency (<30 ms with 75% confidence) for message of up to 1 Kb. In contrast ROS1, showed a markedly increase of latency with increasing message size. However, for small message sizes ROS can still support a latency <60 ms with 75% confidence in drone-to-drone communication.

Concerning drone-to-cloud communication we provide results for characterization of 5G during early rollout of 5G services in Denmark. 5G is seen as an important technology for offering a data uplink connection to the cloud to support the drone system deliver images during inspection mission operation. European 5G rollout will be focused on urban areas and the coverage in rural areas where most inspections are foreseen is not well-studied. Results from measurements in a public 5G network in a rural area of southern Denmark is presented. We studied the link latency performance, packet loss, and data throughput in a rural area public 5G network in southern part of Denmark comparing a single and multi-connectivity scenario.

The second part of the test and validation concerns algorithms and methods for autonomous swarm function of the multi-drone system. We propose and validate a secure group management protocol that allows drones to dynamically join and leave the swarm during the inspection mission. The scheme is based on a cryptographic key management scheme exploiting the trust of a centralized group leader (controller) of the swarm. The proposed concept is validated through a set of test cases.

Motion path planning is a key functionality of the multi-drone system. From an architectural point of view, motion path planning can be divided into global path planning and inspection path planning. Global path planning is provided as a cloud service. It is based on map information and provides a high-level route plan. We have studied the global route planning services and compared the well-known A* and RRT* algorithms with a simple policy-based swarm control mechanism based on the boid model.

The ability of the swarm to move together in a coordinated and collision-free manner was studied using a game theoretical approach. Test and validation of the proposed method shows that it is possible to achieve

coordination and the desired motion path planning of the swarm. Furthermore, we test and validate a method for inspection path planning using sequential optimization. The method is based on a 3D model of the inspection targets e.g., a bridge, and considers the constraints of the inspection camera field of view. The proposed method can outperform the classical travel salesman optimization algorithm for the inspection scenario under consideration. A final motion path planning study investigated the tethered drone system. We propose a policy-based method that ensures the drone-collision-avoidance and rope-collision-avoidance. This reveals a potential of using tethered drones for inspections in case where flight needs to be restricted for e.g., safety reasons.

The two most dominating tasks of the inspection drones are the inspection task and the charging task. Essentially, the inspection task concerns the positioning of the drone at a specific waypoint, with a specific angle to the inspection camera, near the targeted surface of the inspection object. When the drone is low on battery it will need to abort the inspection to charge. We validate the cloud services from WP6 designed to allocate inspection tasks to the multi-drone system. Furthermore, we propose a charging protocol that provides an optimal charging schedule for a swarm of drones in case where there is a scarcity of charging point. Our mechanism demonstrates that an optimal charging schedule will lead to a reduced time for completing the inspection mission. Furthermore, we validate the effect of formation flying using a leader-follower scheme. Formation flying is an advantage in linear infrastructure inspection as several drone can position around the inspection target and proceed as an integrated using.

The deliverable concludes with some outlook towards beyond visual line of sight (BVLOS) inspections.

Contents

Acronyms

Acronym	Description
2D, 3D	Two dimensional, Three dimensional
5G	Fifth Generation mobile
ACK	Acknowledgement
AI	Artificial Intelligence
API	Application Programming Interface
BVLOS	Beyond Visual Line of Sight
C2	Command & Control
CAD	Computer-Aided Design
CCDF	Complementary CDF
CDF	Cumulative Distribution Function
CGAL	Computational Geometry Algorithms Library
CIA	Confidentiality, Integrity and Availability
COTS	Commercial Off The Shelf
CPU	Central Processing Unit
CRC	Cyclic Redundancy Check
CSV	Comma-Separated Values
DL	Download
EU	European Union
FOV	Field of View
FSL	Free Space Loss
GCS	Group Control Station
GPS	Global Positioning System
HCI	Host Controller Interface
HSM	Hardware Security Module
HTTPS	Secure HyperText Transfer Protocol
IP	Internet Protocol
ISM	Industrial, Scientific, and Medical (frequency band)
JSON	Javascript Object Notation
LoRa	Long Range (radio)
LoRaWAN	LoRa Wide Area Network
LOS	Line of Sight

LTE	Long Term Evolution
MAC	Medium Access Control
MIC	Message Integrity Code
MILP	Mixed Integer Linear Programming
MPSoC	Multiprocessor system on a chip
NACK	Negative ACK (or not acknowledged)
NSA	Non-Stand Alone
OAEP	Optimal Asymmetric Encryption Padding
OBC	Onboard Computer
OSM	Open Street Map
OWD	Oneway Delay
PER	Packet Error Rate
PLY	Polygon File Format
PPS	Pulse Per Second
QoS	Quality of Service
RGB	Red, Green, Blue
RGB-D	Red, Green, Blue, Depth (sensor)
ROS	Robot Operating System
RRT / RRT*	Rapidly exploring Random Tree (Star)
RSA	Rivest-Shamir-Adleman
RSSI	Received Signal Strength Indicator
RTT	Round Trip Time
SIG	Signature
SINR	Signal to Interference plus Noise ratio
SITL	Software in the Loop
SNR	Signal to Noise ratio
SPI	Serial Peripheral Interface
TC	Test Case
TCP	Transmission Control Protocol
TDD	Time Division Duplex
UAS	Unmanned Aerial System
UAV	Unmanned Aerial Vehicle
UDP	User Datagram Protocol
UL	Upload
USB	Universal Serial Bus

VIO	Visual Inertial Odometry
VLOS	Visual Line of Sight
VNC	Virtual Network Computing
VRP	Vehicle Routing Problem

2 Introduction

The design of a system for autonomous inspection with collaborative drones requires a set of key innovations resulting from a highly interdisciplinary effort. The goal is to provide a multi-drone system (a.k.a. the drone swarm) that can autonomously inspect critical infrastructure in a beyond visual line of sight (BVLOS) operations. However, such endeavor requires a set of key innovations resulting from a highly interdisciplinary effort. The design of a system for autonomous inspection with collaborative drones needs to progress along a sequence of innovation steps (Figure 1) each providing its unique challenges and requiring specific technological advancements.

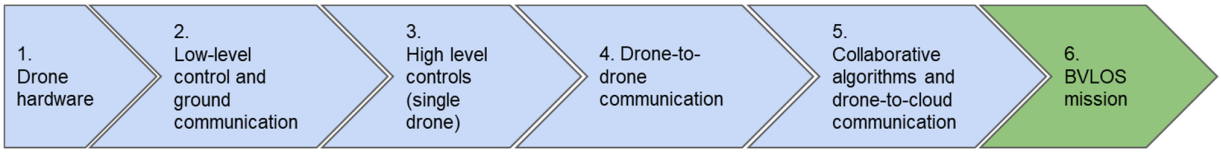


Figure 1: Steps towards autonomous BVLOS inspections with a multi-drone system

First, a robust and long durability single-drone system must be designed including sensors to support system's perception, onboard computing hardware, low-level control algorithms. Second, communication between ground control and the drone needs to support manual piloting as a first step advancing a shift to autonomous operation where the pilot lets the autonomous functions on the drone take over. Third, high-level control functions are applied to provide functions as obstacle avoidance and decision making concerning e.g., charging needs. Fourth, drone-to-drone communication must be established as a foundation of collaborative algorithms to be applied to the multi-drone system. Fifth step adds collaborative functions to the multi-drone systems. Collaborative functions can involve the multi-drone system only such as formation flying or be in combination with services provided by the cloud (cf. WP6) such as global path planning and the overall task allocation of a mission. This latter functionality requires drone-to-cloud communication to be established to support the exchange of information between the cloud and the multi-drone system.

From a methodology point of view, different types of verification environments have been used to progressively advance the system verification and validation. The multi-drone system has first been validated in a simulated environment (Gazebo [18]). In a second stage the system was validated in an indoor controlled environment at AU Deep Tech Experimental Hub (AU facility) followed by an outdoor validation of swarm operations at Hans Christian Andersen Airport (SDU UAS Test Centre).

2.1 Purpose and Scope

Deliverable D5.4 aims at documenting the results from the test and validation activities carried out in T5.6. It provides system-level validation for the multi-drone subsystem focusing on validation on the parts needed as input to system integration, test bed establishment, validation, and demonstration (WP7). Emphasis for the validation is put on the key functionality needed to support the use cases defined in D2.4: Use-case Document [17].

2.2 Document outline

The deliverable is organized as follows. First, we briefly present the different drone hardware platforms used in WP5. Since the ambition for this deliverable is on functional validation of the multi-drone system, we provide only few details on the hardware design. Following the hardware description, we will briefly introduce the used environment for verification and validations.

3 Methods and materials

In the following we introduce the drone hardware platforms and the test and evaluation environments used in design and validation activities in WP5. The WP has been following an iterative design process where hardware and software system have been incrementally advanced, and validation has been advancing with an increasingly higher complexity of the test environment starting with simulations and progressing to indoor testing and concluding with outdoor testing.

3.1 Drone prototypes

The inspection and energy harvesting drone has been the key platform used by the UAS team at SDU (Section 3.1.1). The swarming drone (Section 3.1.2) was used by the AU team for the design and validation of collaborative functions of the multi-drone system.

3.1.1 Inspection and energy harvesting drone

The drone prototype for navigating and recharging from the power is consists of the following components:

- MPSoC: Ultra96v2
- Sensors: mmWave, Camera, and magnetometers
- Flight controller: CUAV V5+ autopilot

The first prototype is shown in Figure 2 and encompasses a small airframe for rapid testing one functionality at a time. The details of hardware and software drone system architecture is explained in D5.3 [19].

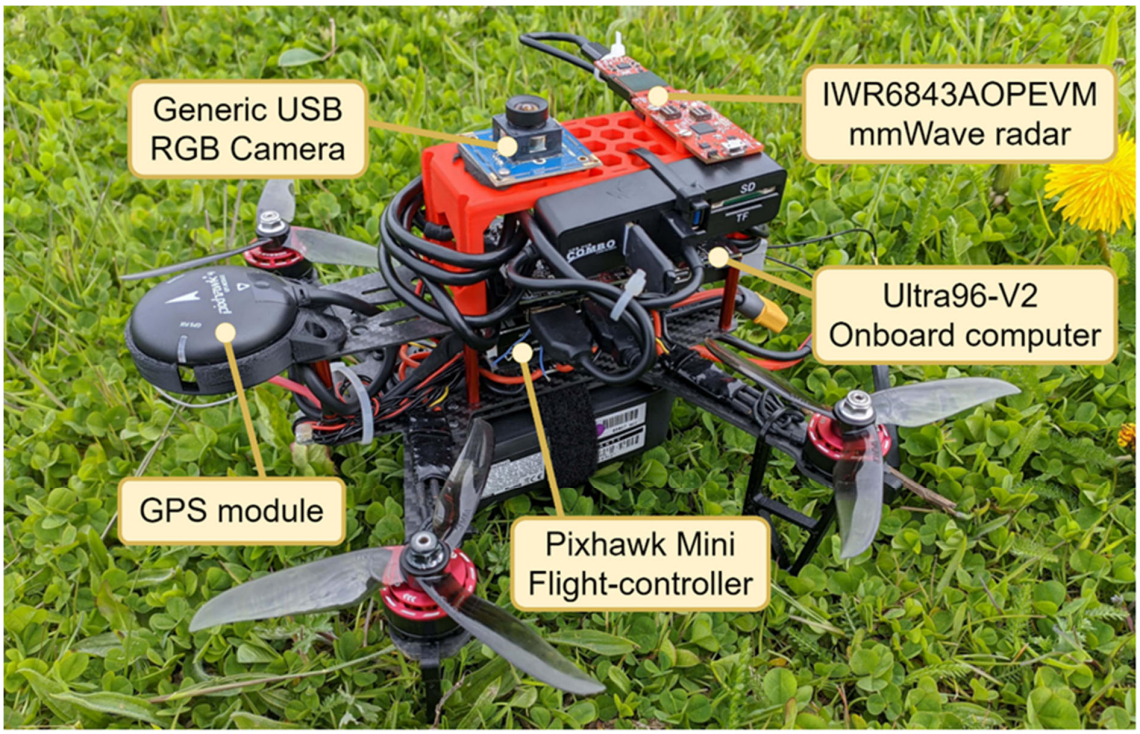


Figure 2: Hardware design of the drone.

The drone platform is also integrated with the split-core energy harvester that is developed in WP3 and presented in D3.3. Figure 3 shows the drone with the energy harvester while approaching a powerline.

Figure 3: Drone with the energy harvester.

3.1.2 Swarming drone prototype

The drone prototype for validating swarming function is built based on ModalAI VOXL platform and S500 ARF Frame Kit. Figure 4 shows the hardware details of the drone prototype platform based on the VOXL platform. Compared to the inspection and harvesting drone, the ModalAI VOXL platform provides a more self-contained low-level flight control allowing design and validation to focus on high level control functions needed for the multi-drone system's collaborative functions.

Figure 4: Hardware of swarming drone prototype is based on the ModelAI VOXL platform.

Figure 5 shows two drones on formation flight in an outdoor test field. In the swarming function, onboard WiFi provides drone to drone communication. The test and validation are further described in Section 5.4.2.

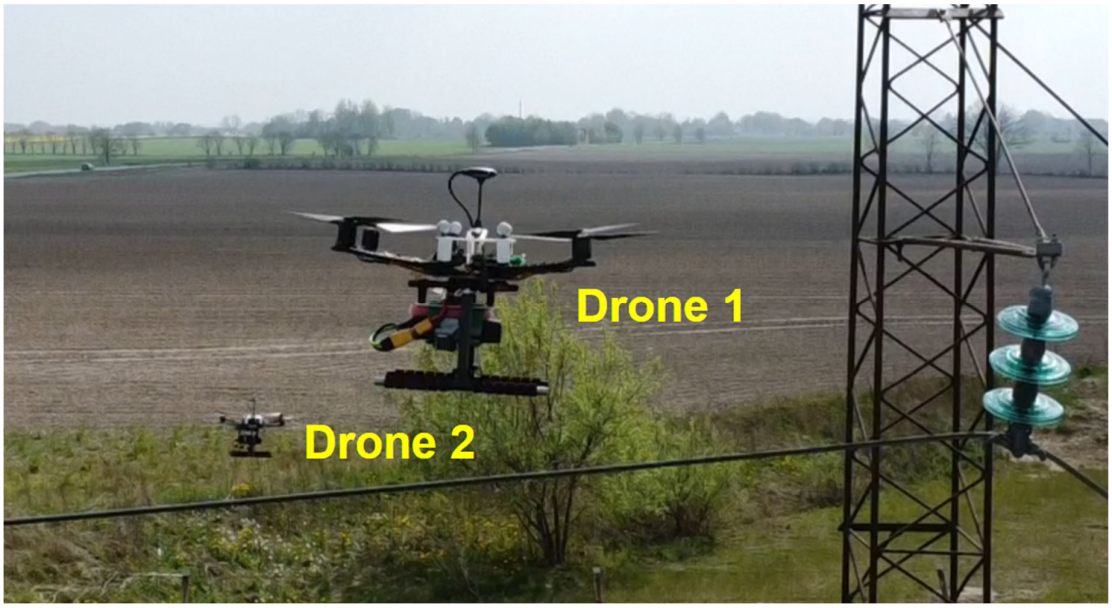


Figure 5: Outdoor formation flight test.

3.2 Simulation and validation environments

Simulation is an essential tool for developing autonomous drone systems. A well-designed simulator makes it possible to test algorithms and design drone systems rapidly and safely before attempting to fly in the real world. As selected for the Drones4Safety project, Gazebo provides the ability to simulate multiple UAVs accurately and efficiently in different outdoor environments [18]. Gazebo offers a robust physics engine, high-quality graphics, and convenient programmatic and graphical interfaces. Besides the Gazebo simulation environment, the PX4 development community also provides the simulation toolchain to allow PX4 flight code [22] to control a computer modeled drone in a simulated environment, which provides the same interactive interface with this vehicle as users might have with a real vehicle. The PX4 toolchain supports the Software-in-the-loop (SITL) simulation, where the flight stack as well as developed algorithms, usually written for a particular flight controller, is tested within a modeling environment.

3.2.1 PX4 SITL with Gazebo

This section describes the data link of the proposed drone simulation environment. The diagram in Figure 6 shows a schematic of the SITL simulation environment. Gazebo is used as the simulator [18]. Different parts of the system connect with UDP and can be run on either the same computer or another computer in the same network. PX4 connects the simulator (Gazebo) using the Gazebo MAVLink API. This API defines a set of MAVLink messages that supply sensor data from the simulated world to PX4 and return motor and actuator values from the flight code that will be applied to the simulated vehicle. PX4 uses standard MAVLink module and MAVLink message set to connect to ground station (QGroundControl) and external developer APIs like ROS on each dedicated port. Serial connection is used to connect Joystick controllers via QGroundControl.

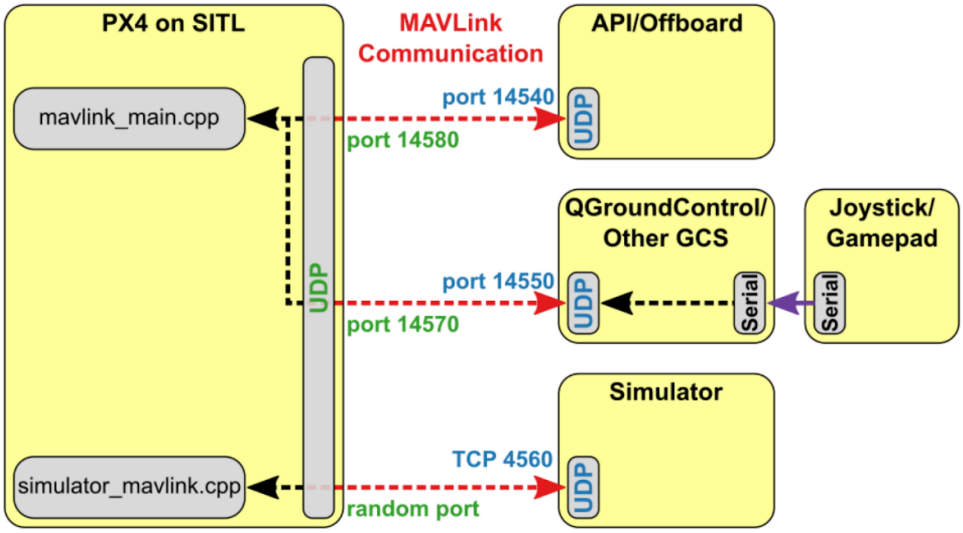


Figure 6: PX4 SITL simulation environment using Gazebo as the simulator.

The multi-drone simulation environment is developed based on the single drone SITL simulation setup. There are two types of multi-drone simulation, as shown in Figure 7 and Figure 8. The setup in Figure 7 simulates multiple UAVs in a shared Gazebo simulation world. Each drone has its unique PX4 software but is controlled by the same ROS middleware by using different ROS messages for different drones. QGroundControl as a ground control station is optionally used to monitor the status of all connected drones.

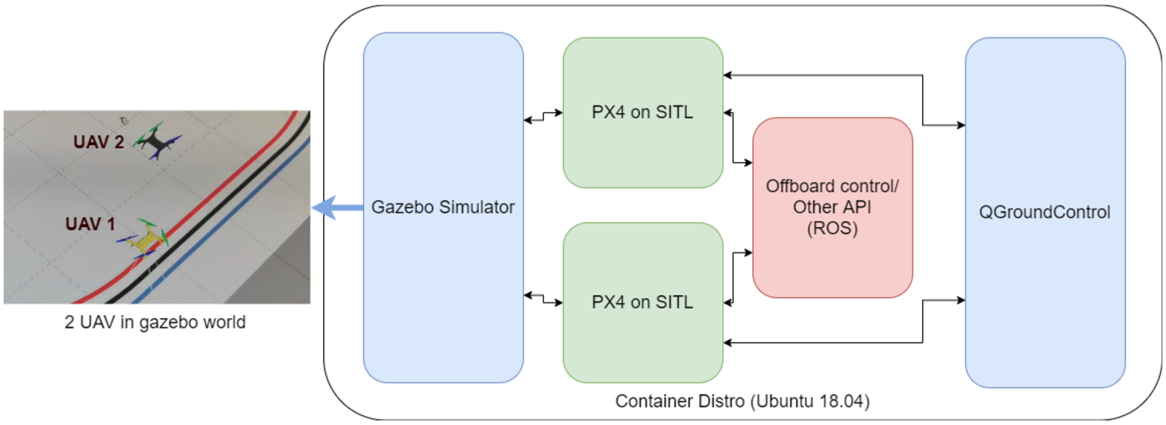


Figure 7: PX4 multi-UAV SITL simulation environment. The example simulates a formation flight with two drones using a leader-follower algorithm design.

In Figure 8 a second multi-drone simulation environment, which is closer to the real software deployment scenario for a multi-UAV system. Each UAV in this simulation has been assigned a Docker container with an independent container distro. Containers are connected in a local virtual IP network. The drone software in the container communicates with each other using ROS middleware through TCP/IP (Multi-ROS-Master communication). Each Docker container provides a graphical user interface based on VNC for users to monitor and configure the simulation. As shown in the right side of Figure 8 a camera view of the simulated world and a visualization of the obstacle perception of two drones are presented.

Figure 8: PX4 multi-drone SITL simulation environment.

3.2.2 Gazebo simulation environments mimicking the UAS test facility

To complement the physical testing, we created a Gazebo simulation environment based on the test location at the Hans Christian Andersen Airport, Odense. The terrain was photograted at the test site and stitched together using photogrammetry, with size and height of the setup being measured. This data was then taken to reproduce a size-accurate and positionally-accurate environment. To increase the realism of the environment, weather conditions such as Wind and sky boxes were also implemented using plugins incorporated into the Gazebo software, proving invaluable during system verification.

a) b)

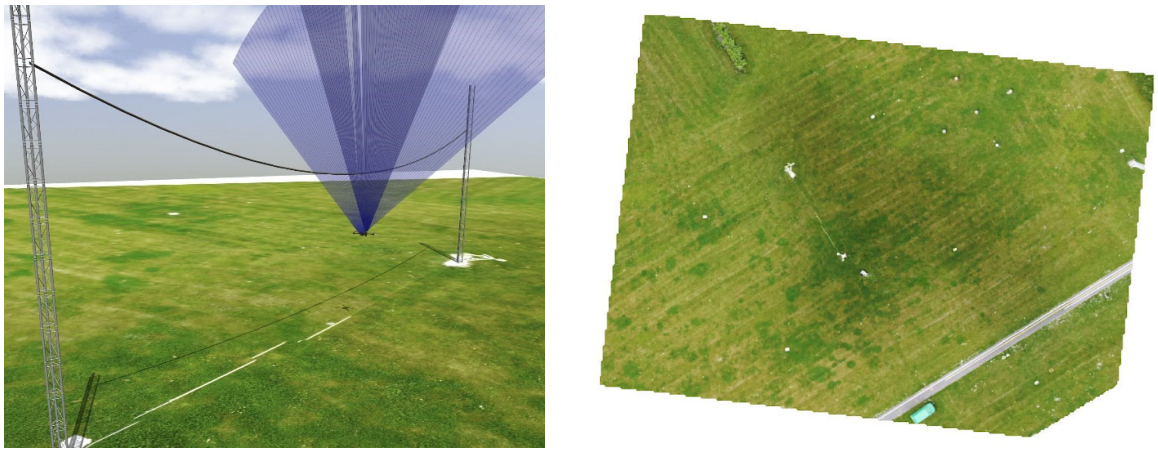


Figure 9: (Left) Simulation environment with test setup, (right) composed orthomosaic with absolute orientation, location, and size metadata,

Additionally, the CAD models for the drone reflecting an earlier iteration of the drone shown in Figure 10, were also processed for use within the Gazebo simulator, ensuring the performance and weight of the drone accurately represented the physical counterpart.

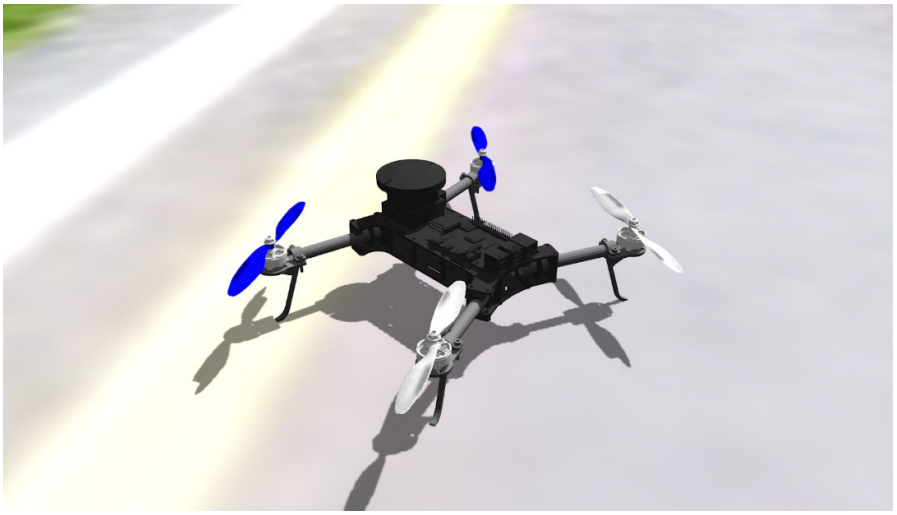


Figure 10: Prototype frame imported into the simulation environment.

3.2.3 Indoor drone testing facility

We built up an indoor test environment for functional validation. For Indoor localization, we use either a VICON motion capture system or onboard Visual Inertial Odometry (VIO) system. Figure 11 shows the indoor test environment for validating the formation flight function.

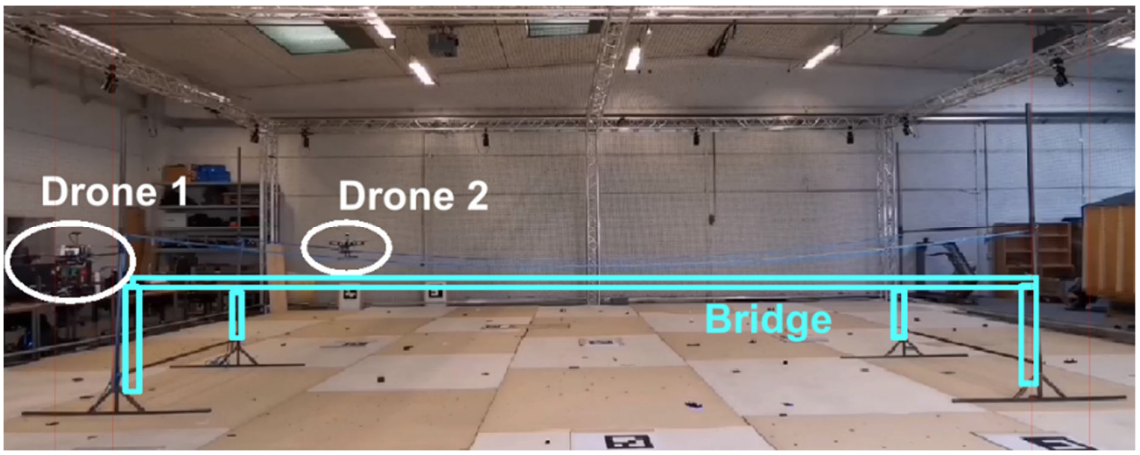


Figure 11: Indoor test facility for formation flight function validation.

Figure 12 shows another test environment with a mockup bridge for validating the inspection motion planning function. In the above two test environments, WiFi equipment is provided for supporting drone to drone communication and drone to ground control station communication.

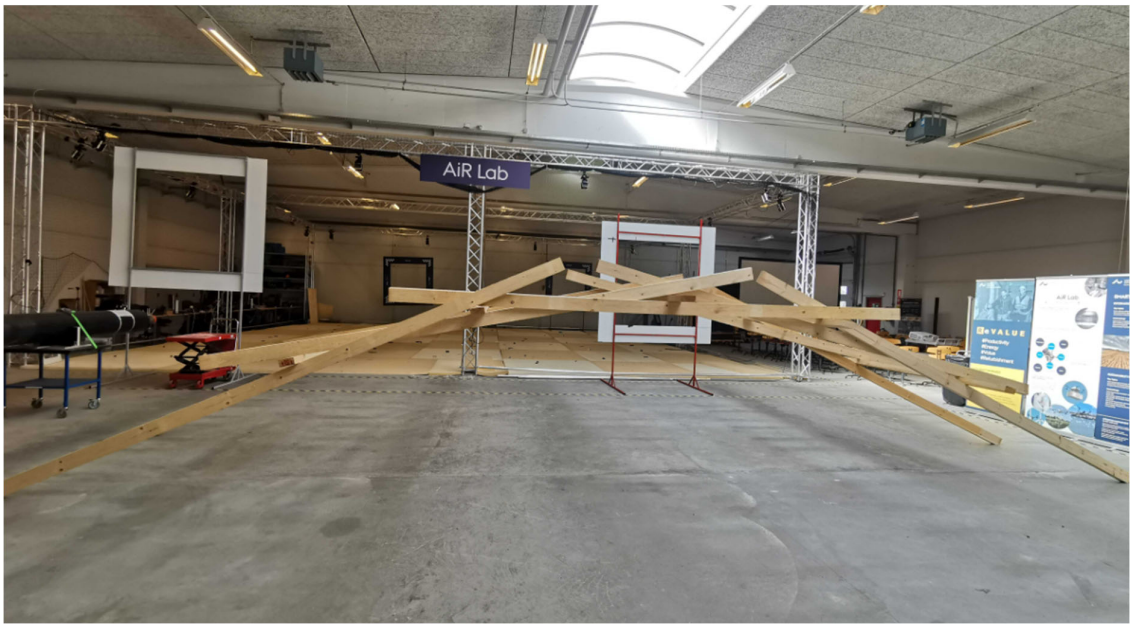


Figure 12: Bridge mockup built in the indoor test environment.

Autonomous test the obstacle avoidance function with one drone in the indoor test environment. In the example below (Figure 13) the obstacle avoidance function of a single drone was tested. The drone estimates its state based on onboard sensors using visual-inertial odometry tracking software and detects obstacles using an Intel Realsense D435 RGB-D camera. Flight experiments were conducted to validate the obstacle avoidance algorithm. In the video example, the drone first switches to offboard mode and flies to its initial waypoint in

front of the obstacle. A goal waypoint is then set behind the obstacle. The drone moves past the obstacle following a collision free trajectory.

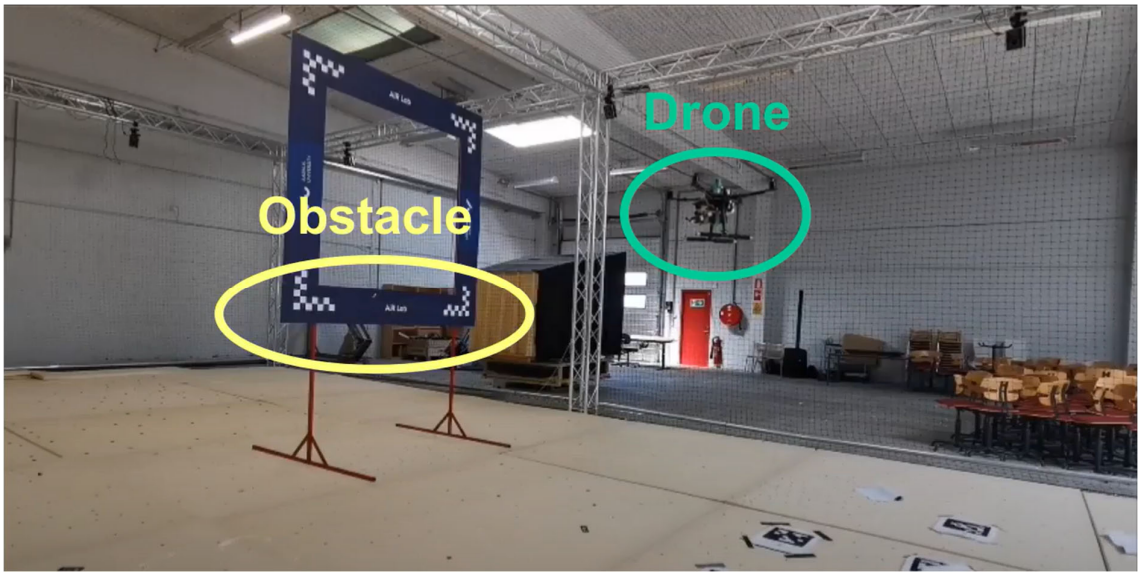


Figure 13: Example from the validation of obstacle avoidance algorithm.

The scenario depicted in Figure 13 showed the drone autonomously approaching and brakes before an obstacle represented by the blue frame. Given a guiding waypoint behind the obstacle, the drone was able to calculate an alternative path and navigate past the obstacle.

3.2.4 Outdoor drone testing facility

Outdoor powerline testing environments were deployed at SDU UAS Test Centre in Odense, Denmark for powerline inspections using drones. At approximately 10 m tall and 35 m long, the outdoor power line environment is a small-scale 3-phase energized powerline setup constructed from previously deployed energy grid parts. Figure 14 shows the setup; cable diameters are ø20 mm for the energized segments and ø10 mm for the ground segment.

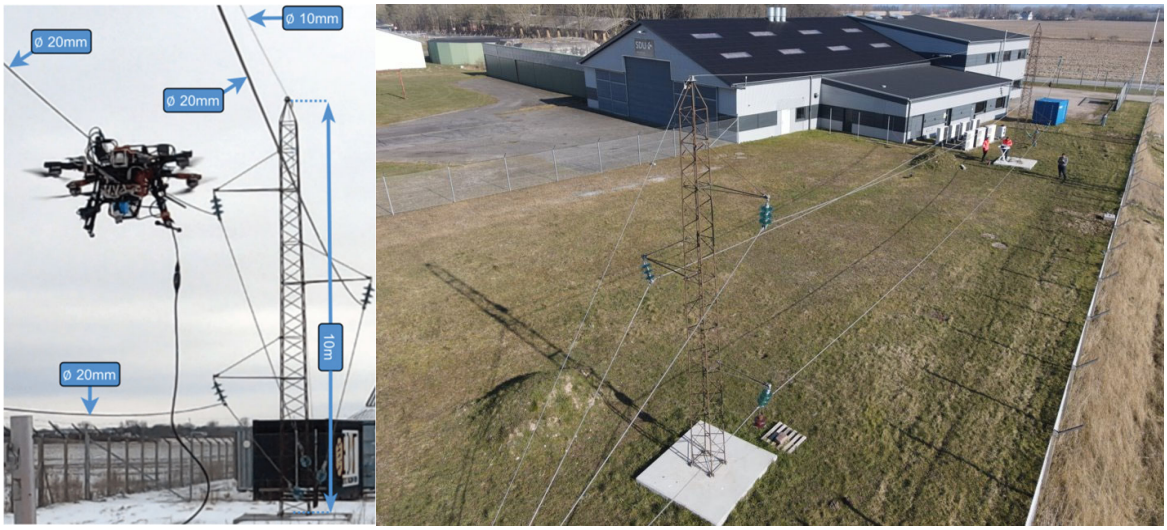


Figure 14: Powerline test setup at SDU UAS Test Center, Hans Christian Andersen Airport, Odense.

4 Communication performance

Connectivity is essential for the performance and success of the multi-drone inspection missions. Deliverable D5.1 "Specification of the multi-drone system" of the Drones4Safety project described the overall concept of operation of the multi-drone inspection and introduced the supporting communication infrastructure [1]. The system architecture distinguishes between three modes of communication each with a set of unique communication requirements.

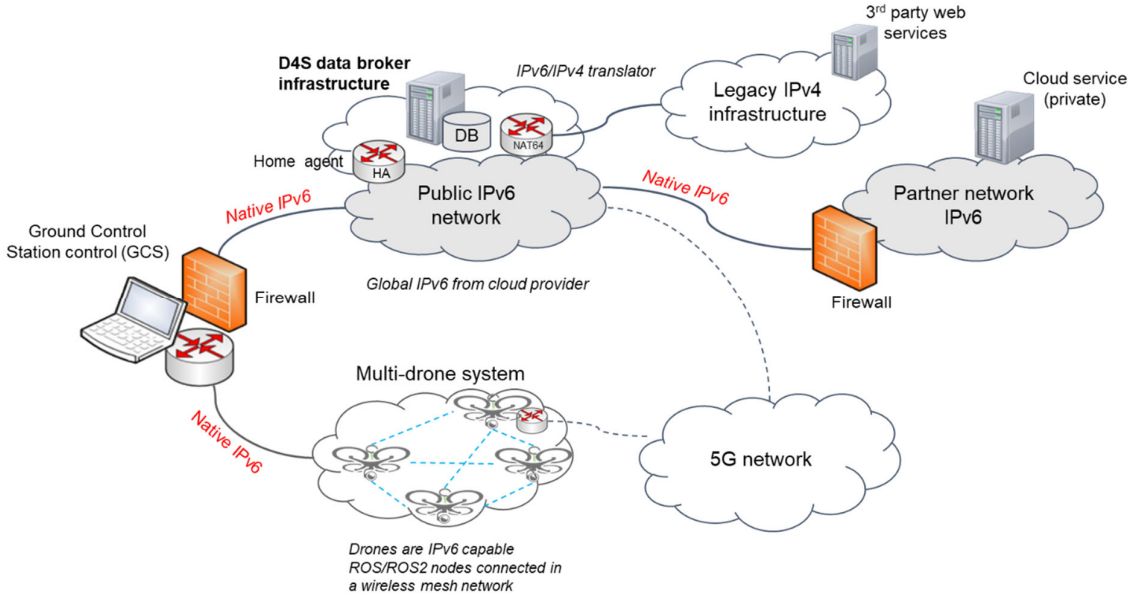


Figure 15: Drones4Safety communication infrastructure.

The swarm of drones has the capability of collaborating with cloud services. Data exchange between drones and cloud service are primarily achieved through long range wireless communication based in LoRa and secondary through a public mobile network infrastructure. LoRa is primarily used as the inspection mission

operator, i.e., the drone operator, can be fully in control with the network coverage. Mobile networks provide a good opportunity for connectivity but network coverage, in particular in rural areas, cannot be guaranteed.

The validation of the communication in WP5 has focused on the assessment of LoRa as drone-to-ground communication technology for BVLOS operation. In addition, we have studied the use of 5G mobile communication currently being rolled out in many European countries. In this regard, our aim has been on 5G's capability to support drone-to-cloud communication.

4.1 Long Range radio (LoRa)

LoRa is usually deployed with Long Range Wide Area Network (LoRaWAN) for the link layer with contention-based Aloha Medium Access Control (MAC), where there exist duty cycle regulations of 0.1-1% for the end nodes' transmissions in the previous bands, thus diminishing the throughput even more [13]. To alleviate these known constraints, a more recent LoRa chipset by Semtech has been released in the 2.4 GHz ISM [14], where typically WiFi is deployed, providing considerable larger data rates than the sub-1 GHz counterparts, with no duty cycle limitation to achieve full rates, and aiming to keep the same range. Such developments aiming at new market segments raise again the question of feasible reliable LoRa C2 links for drone applications in terms of range, application throughput and latency in rural areas. In contrast to LoRa operating at sub-gigahertz frequencies, the 2.4 GHz LoRa is not subject to duty-cycle requirements. Another advantage of LoRa is its resistance to interference from other communication devices using the free ISM frequency band. A tradeoff between range and data rate exists. On the one hand we want the drones to have reliable (and low latency) communication on a long range and on the other hand we would like a reasonably high data rate to be able to transmit video streams from the drones to an operator or a mission control entity. This poses the question on practically achievable performance of LoRa.

4.1.1 Experimental setup

To validate the performance of LoRa several outdoor experiments was performed in a rural setting as sketched in Figure 16. Our measurement was carried in the Mollerup forest area near Aarhus, Denmark (Figure 17). This area considers plains, low hills, and very sparsely distributed trees, that represent typical rural areas where infrastructure, such as power lines, could be inspected.

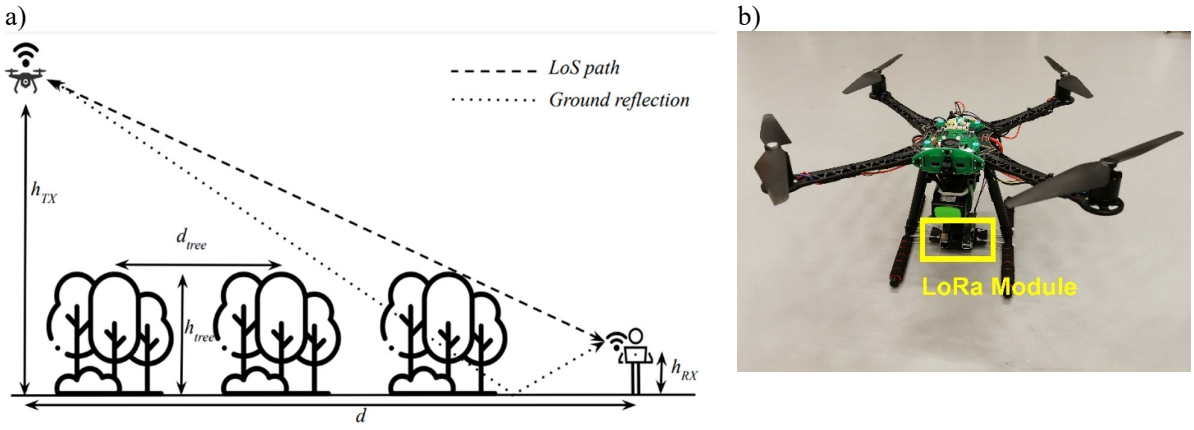


Figure 16: a) Sketch of the LoRa outdoor testing setup. b) Picture of the drone platform carrying the LoRa module used in the testing.

Figure 17: Pictures from the field experiments with LoRa testing in Mollerup Forest. The forest has a mix of open grass areas and more dense areas with vegetation.

Drone setup, data-collection scripts, parameter space and test descriptions, with as many pictures and diagrams as possible. Mention that the setup is challenging because of battery lifetime (1 battery charge = 1 flight).

The setup for the test is as follows: The LoRa transmitter is connected through Serial Peripheral Interface (SPI) to a Raspberry Pi 4 mounted on a drone assembled with COTS components. The LoRa receiver is connected through USB Interface to a personal computer. The drone has been equipped with a PX4 flight controller using the Raspberry Pi as its On-Board Computer (OBC), and remote controller interface in 433 MHz for manual control of a drone pilot.

The LoRa module is equipped with two Inverted-F antennas but for this experiment only one of them is used. It is expected that performance will improve if using an antenna switching technique such as best received RSSI. The antennas are omnidirectional so no special precautions have been made for placement of the modules while during the tests.

Data collection scripts have been made in Python to carry out the measurement using the Host Controller Interface (HCI). To simulate data transmissions of a drone, a MAVLink message with a length of 40 bytes was used for the measurements. To streamline the measurements and get most results from the limited battery capacity, a script was used to easily switch between different spreading factors while the drone was in the air. The Python scripts on the drone were executed through a SSH connection using a mobile phone's WiFi hotspot.

For the path loss measurements, the drone uses a script that sends an unlimited amount of MAVLink messages while the personal computer is using a script to receive the messages and log the statistics. The following information is stored in a CSV file which can be used for further analyzing payload, RSSI, SNR and whether a CRC error has occurred or not.

To do the RTT measurements, a script is used on the personal computer that transmits an unlimited number of messages that requires the receiver to respond with acknowledgement messages of whether the transmissions went successful or not. The RTT timer is started right before the HCI message is sent that requests the LoRa Module to transmit a packet. The timer is stopped as soon as the HCI message containing the response message is received. A window timer is used to determine the amount of time to wait before a packet is assumed lost. The script generates a CSV file that stores the RTT and in case a TIMEOUT, NACK or CRC occurs.

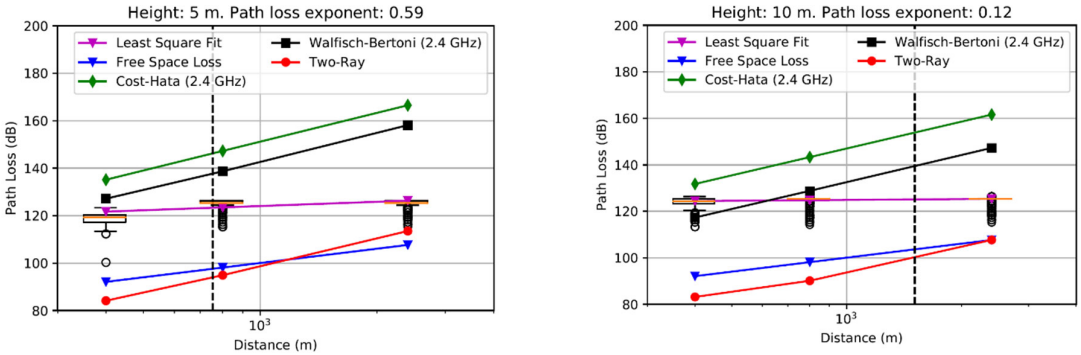
4.1.2 Analysis and results

The LoRa signal gets attenuated due to a path loss when going through the wireless medium. Fading and shadowing effects due to nearby scatterers reduce the Signal to Interference plus Noise Ratio (SINR) and introduce delays at the receiver. We define the transmitter to receiver net signal loss in the link budget as:

$$PL(d) = P_{TX} - P_{RX}(d) - G_{TX} - G_{RX}$$

Our path loss measurements account for the transmitter power P_{TX}, the received power $P_{RX}(d)$ dependent on the distance, and misalignment from the maximum antenna gains, G_{TX}, G_{RX} at an angle θ. For a given frequency, an empirical path loss model is dependent only on the distance between transmitter, receiver, propagation conditions, and related terrain geometry. Different path loss models can be applied for any given scenario. All quantities P_{TX}, P_{RX}, G_{TX}, and G_{RX} are measured in decibel (dB).

Figure 18 shows measured path loss for distances of 400 m to 2.4 km.



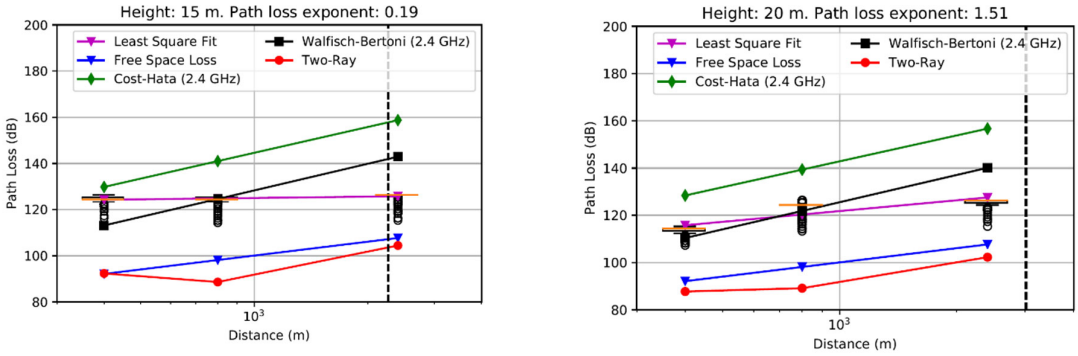


Figure 18: Pathloss measurements at different drone heights: 5 m, 10 m, 15 m, and 20 m.

4.1.2.1 Single Slope Path Loss model

In the case of a single slope, the path loss is given by:

$$PL(d) = PL_0 + 10\,\gamma \log\left(\frac{d}{d_0}\right) + X_\sigma \quad \text{where} \quad d > d_0$$

PL_0 is the initial path loss at a given distance $d_0 = 1$ m. PL_0 may be calculated from the Free Space Loss (FSL) model as:

$$PL_0 = FSL(d_0, km) = 92.45 + 20\log(f_{GHz}) + 20\log(d_0, km)$$

In this equation, f_{GHz} is the frequency of the transmitting signal in GHz, and d_0, km the distance measured in km. is the path loss exponent where higher values of $\gamma > 2$ represents rural to urban scenarios with large signal strength loss, and of where low values $\gamma \leq 2$ represents path loss in a guided medium up to free space ($\gamma = 0$). X_σ is Gaussian random variable of zero mean and variance σ^2.

In Figure 18, the free space path loss model is shown alongside the measurements. For the measurements a least square fit provides the path loss exponent. As can be seen, the path loss exponents fall into the category $0 \leq \gamma \leq 2$ showing that the LoRa communication is partly guided by the environment.

For comparison, Figure 18 also shows the calculated pathloss based on the COST-Hata and the Walfisch-Bertoni models. The COST-Hata Model This model accepts physical barriers between transmitter and receiver antennas as homogeny. Because of the frequency band limitation of the more commonly used Okumura-Hata model (150-1500 MHz), the original model was recreated later to address the frequency band of 1500-2000 MHz hereafter referred to as the COST 231 extension to the Hata model i.e., COST-Hata model [24]. The COST-Hata model is considered to be valid for frequencies between 1.5-2 GHz, heights between 30 m < h_{TX} < 200 m, 1 m < h_{RX} < 10 m, and distances ranging from 1-20 km.

4.1.2.2 Two-Ray Ground Reflection Model

In some cases, the phase difference between the Line of Sight (LOS) component and the reflected one is not negligible. The two-ray model is used when a single ground reflection dominates the multipath effect for the propagation radio wave. In this setting the LOS component, modeled by the free-space propagation loss model, is superimposed by a reflected wave considering the reflection coefficient and the phase shift of the reflected wave [24].

Diffraction models: The Walfisch-Bertoni model considers diffraction from rooftops and building in a cellular system [24]. The model assumes that the transmitted wave propagates based on free-space propagation and is

then reradiated by the scatterer with a reduced transmit power. Figure 16 shows the Walfish-Bertoni path loss model alongside the measured

An elaborate propagation model combines the above approaches and considers the superposition of all rays with non-negligible contributions to the resulting signal and the receiver. This concerns a combination of LOS, reflected and diffracted components. As the different components may weight differently at different distances and is often desirable to model the path loss as a piecewise linear (in dB scale) function accounting allowing parameter models to be adjusted in different distance intervals. This is known as the piecewise linear path loss model shown with the two-ray model in Figure 18.

4.1.2.3 Communication delay

To support a C2 channel the communication delay is of utter importance. LoRa was originally not designed to be a low-latency communication channel. The robustness of the channel was given priority. The use of different spreading factors with LoRa achieves this robustness by spreading the signal over time. Consequently, use of higher spreading factors yields longer communication delay. To assess the potential of use LoRa communication for low-latency communication was verified in the experiments described above. Figure 19 shows the cumulative density function plots of the round-trip-time at different spreading factors.

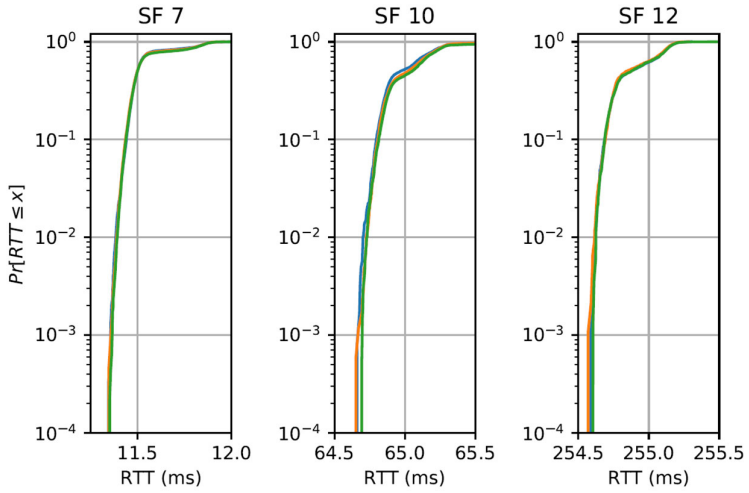


Figure 19: Measured RTT for LoRa communication at different spreading factors SF: 7, 10 and 12. The drone height was 20 m in all experiments.

As can be seen from Figure 19, LoRa at the low spreading factors can provide a below 50 ms latency as required by the C2 channel with very high probability. For high spreading factors SF > 9 this a sufficiently low delay cannot be obtained to support C2. Nevertheless, LoRa is still applicable for telemetry reporting. Based on our validation of LoRa we conclude that C2 can be supported with spreading factors of SF = 8 and below.

Finally, achievable data rates were measured through a series of goodput measurements. The goodput measures the amount of data that can be transmitted to an application running over the wireless channel and thereby disregards protocol overhead. Figure 20 shows the goodput measurements.

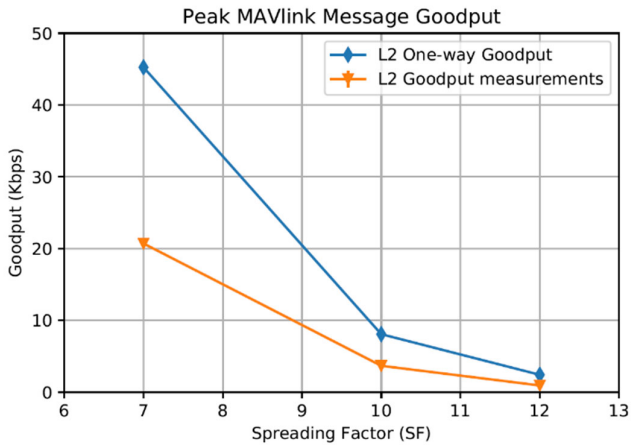


Figure 20: Measured goodput for different spreading factors.

As expected from LoRa communication the goodput drops as the spreading factor increases. We estimate that for telemetry data a minimum goodput data rate of 17.6 kbps for a team of 4 collaborative drones working [8]. This should be compared to the One-way goodput measurement of Figure 20. Again, we conclude that spreading factors SF=8 or below are suitable for a low-data rate drone communication channel to support telemetry.

4.2 WiFi and wireless networking

The exchange of information between drones i.e., the drone-to-drone communication requires higher data rates to be supported that can be provided by LoRa [25]. In WP5, we choose WiFi based in the IEEE802.11 standard for the inter-drone communication. In this section, the present results of the analysis of using ROS and ROS2 communication over WiFi including a study of its performance.

4.2.1 WiFi and Multi-master ROS network

As shown in Figure 21, remote inter-process communication is evaluated by deploying ROS/ROS2 publisher and subscriber respectively in two machines connected using WiFi, Intel NUC8i3 with 16 GB RAM and integrated wireless module (Intel Wireless-AC 9560) and Intel NUC8i5 with 16 GB RAM and integrated wireless module (Intel Wireless-AC 9560). The influence of changing WiFi mode, direct link using WiFi ad-hoc mode (Ubuntu WiFi configuration) and link passing through WiFi access point with default WiFi mode (Ubuntu WiFi configuration), are measured. Measurements are taken in a common indoor environment in the presence of WiFi signals from other networks. Two machines are put on the table with 1 m distance. To avoid inaccurate system synchronization between two machines, the latency is measured based on a round trip, of which the return trip is implemented by a direct C++ socket communication.

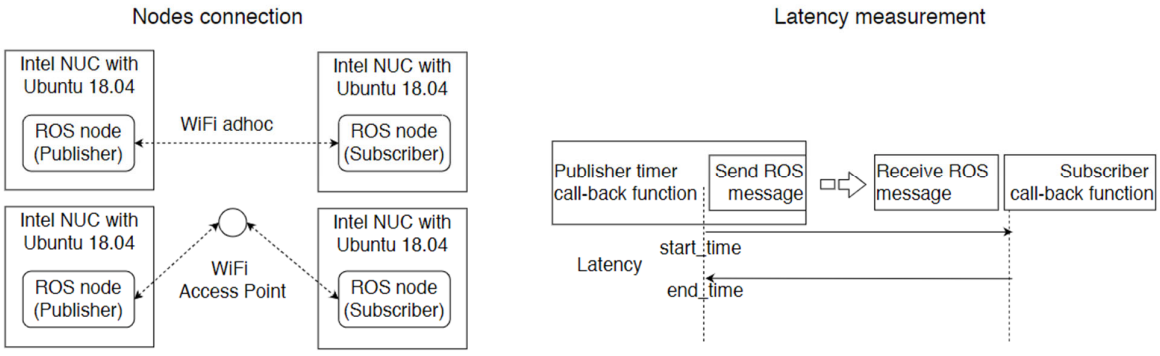


Figure 21: Experiment configuration for remote inter-process communication.

Figure 3 represents the data of remote interprocess communication from 120 messages of per message size. ROS2 Foxy shows a more stable latency then ROS Melodic on small message size, Figure 22, a) and b). However, reversed results are observed when using large message sizes, especially on the size of 4M byte, (Figure 22, c and d).

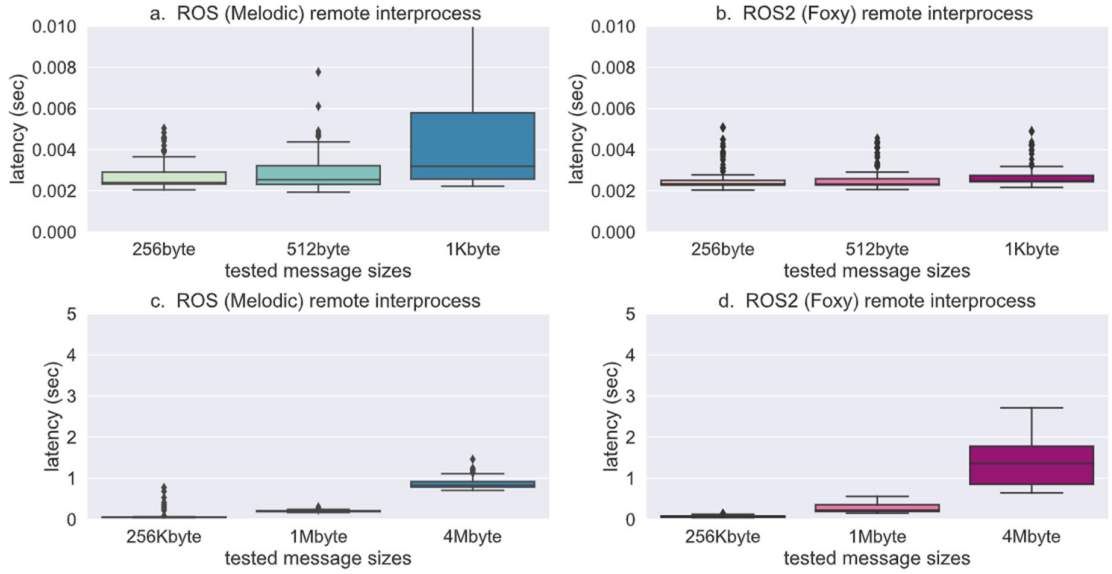


Figure 22: ROS and ROS2 remote interprocess communication with default configuration using WiFi access point mode.

Figure 23 shows the performance difference between the WiFi ad-hoc mode and the default access point mode. In the case of ROS Melodic, WiFi ad-hoc mode provides a more stable and shorter latency than using a WiFi access point on small message sizes.

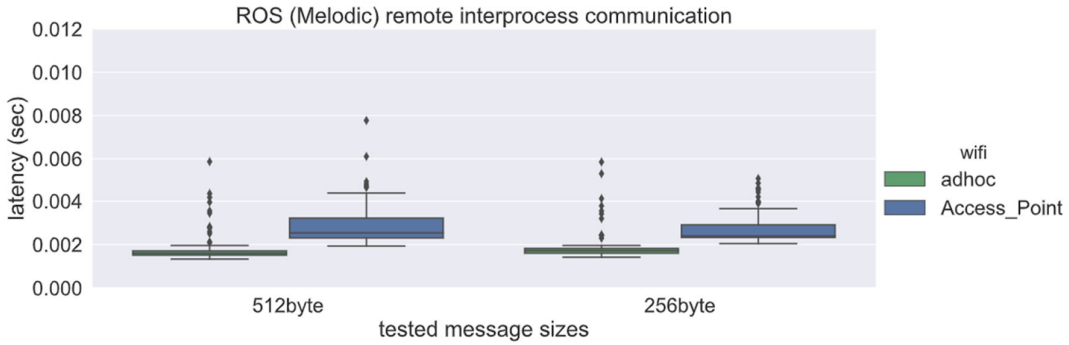


Figure 23: Remote interprocess communication using either WiFi ad-hoc mode or access point mode.

4.3 Communication with Cloud services

As can be seen in the Drones4Safety network architecture (Figure 15), 5G mobile communication is identified as a key enabling technology for connecting the multi-drone system to the Internet and cloud services [1]. To assess the potential of 5G for drone-to-cloud communication, we performed field test experiments to validate the current potential of 5G mobile communication to support autonomous inspections with drones.

4.3.1 5G communication service

Fifth generation mobile (5G) technology will be able to provide higher capacity, higher data rates, lower latency, and increased energy efficiency than previous generations of mobile cellular technologies. Therefore, 5G is expected to be an enabler for more flexible and efficient solutions for vehicles systems such as autonomous drones. With the new capabilities of 5G, telecom operators strive to improve the user experience of existing mobile customers, while enabling cloud connectivity and targeting new business opportunities and use cases such as robot and drone control [27], manufacturing, as well as agriculture [28]. To achieve this, the main design development of 5G has focused on bringing down communication latencies to the millisecond level to support real-time wireless control loops while also improving the overall network capacity to allow scaling of the networks.

Presently, 5G is being rolled out in countries across the globe and consumers can already purchase mobile phone subscriptions which give access to the new services. However, it is still early days for 5G, and the technology is still immature. 5G deployment is done in a rolling manner, where features are piecemeal rolled out as they become available. This means the current early 5G systems might not yet provide many of the promised features and benefits, which will come later with Rel.16 (expected to be commercially available by the end of 2023) with ultra-low latency features, and Rel.17, which will enable positioning over 5G.

In collaboration with Aalborg University and the EU project Internet of Farms 2020 (IoF2020), where Aarhus University is a partner, a series of 5G connectivity experiments were conducted to characterize the network coverage and network performance during the Danish 5G roll-out phase. Rural areas in Denmark were targets for this investigation as this was a better match for both the Drones4Energy and the IoF2020 project. For the testing we were using TDC's public 5G infrastructure.

4.3.1.1 Experimental setup

Aalborg University contributed with specific 5G Quality of Service (QoS) test equipment. Tests were conducted based on the previous cellular network characterization methodology, while the dedicated equipment was built on top of the reference design originally designed for 5G Industry 4.0 manufacturing applications. To empirically assess the 5G network capabilities, the primary focus was the implementation of a measurement device able to monitor and record the relevant communication QoS parameters: link latency, Packet Error Rate (PER), and throughput (data rate). For further reference, the device was designed to also

record timing, Global Positioning System (GPS) position, and 5G signal strength (as received power in dBm), and 5G signal quality in terms of Signal-to-Noise Ratio (SNR). Recording and analyzing all these parameters over a specific measurement route, allows for direct assessment of the 5G capabilities for services in each area.

The 5G measurement setup was further designed to support advanced multi-connectivity, which is a connectivity solution based on hardware duplication that might be of interest for autonomous drones requiring reliable wireless support for their industrial applications. Multi-connectivity is implemented by considering two 5G network modem interfaces (instead of one as is normally done) as user equipment to increase the availability and reliability of the 5G connection by using the best of the two or a combination of both. Multi-connectivity schemes rely on the uncorrelation in time and space of poor network performances or failures on the different interfaces. This means than when one of the interfaces might be experiencing long delays or transmitting low data due to high number of connected users, long distance to the serving cell, or is performing a cell change; the other might be connected to a nearer different cell and, therefore experiencing a better connection with better latency and/or higher data rate transmissions.

The scenario selected for the data-driven 5G trial took place in a rural area near Padborg, in the south of Denmark. In this area, there was 5G coverage from the Danish telecom operator TDC. This 5G network was selected since TDC has already reported 5G coverage in 98.4% of the country, including main rural areas. At the moment of the trial, TDC's public 5G network was an early deployment of 5G Non-Stand Alone (NSA), operating in Time Division Duplex (TDD) mode over 100 MHz bandwidth in the 700 MHz band, using 1800 MHz as 4G LTE anchor band. This means that the network was not yet able to provide the full feature set of 5G. As the trial was done right after the initial rollouts, in June 2021, no impact from other simultaneously connected devices was expected since not many TDC public network users had taken up 5G services yet. To evaluate the 5G QoS parameters in the test scenario, the measurement route for a ground vehicle (a tractor) was defined. The route was planned to include a mixture of realistic environments with both on- and off-road driving. The roads were primarily small to medium size country roads, while the off-road driving consists of driving on an open agricultural field and inside a forest confirming the availability of 5G coverage throughout most of the track. An average vehicle speed of 40-50 km/h was used when on-road and approximately 10 km/h when off-road.

The baseline platform is an ARM-based Gateworks GW6405 industrial computer, chosen due to its small size, support for up to four mini-PCIe extension cards and integrated GPS hardware with Pulse Per Second (PPS). This GPS hardware allows recording the position of any given 5G measurement sample and having easy access to an accurate time source for synchronization. By using the LinuxPPS API it is possible to synchronize the industrial computer clock with down to 10-100 µs accuracy. The computer board is housed in a metal enclosure and equipped with 8 omnidirectional antennas, feeding two Simcom SIM8300G-m2 cellular modems (with four antenna ports each) installed in the GW6405's mini-PCIe slots through an M.2 to mini-PCIe converter board. The SIM8300G-m2 is chosen based on recommendations from TDC as it supports both their 4G and 5G sub-6 GHz bands. A GPS antenna is connected to the GW6405 and routed to the roof of the vehicle.

4.3.1.2 Analysis and results

The link latency performance was individually evaluated for DL and UL in terms of OWD and combined in terms of RTT. Figure 24 displays the CCDF of DL and UL OWD. It illustrates a very similar behavior for both modem 1 and modem 2, with a median ($10^{-0.5}$) DL OWD of 11.5 ms and 12.1 ms, respectively and with a DL OWD lower than 100 ms in 99.3% of the cases (Figure 24a). The tails of the distributions are slightly different between modem 1 and modem 2, reaching in both cases a maximum DL OWD of approximately 0.5 s at the 99.999% ($1 - 10^{-5}$) level. As displayed, the multi-connectivity scheme based on the selection of the best interface (min) leads to an improved DL OWD, with a median DL OWD of 11.3 ms, and a much more

deterministic tail, which contained DL latency lower than 100 ms in 99.9% ($1 - 10^{-3}$) of the cases. With multi-connectivity, at the 99.999% level, the maximum DL OWD was 219.1 ms.

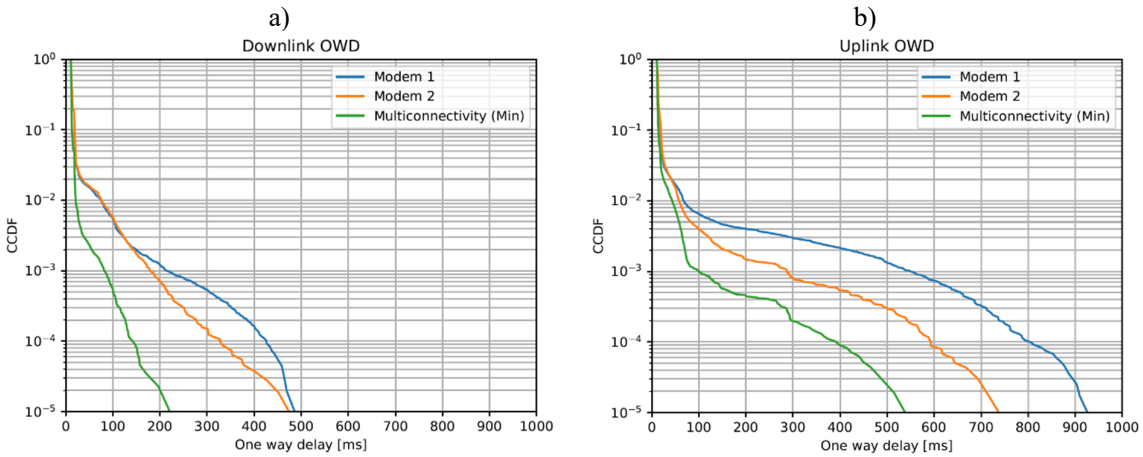


Figure 24: Empirical CCDF of DL OWD (left) and UL OWD (right) for modem 1, modem 2 and latency-optimized multi-connectivity.

The UL OWD performance results (Figure 24b). UL latency is larger than DL latency due to the fact that in 5G networks, DL transmissions are scheduled almost instantaneously at the base station, while UL communications, initiated by the 5G end device, need to wait for the base station to perform resource allocation and issue an UL grant permission before the data can be transmitted. Both modem 1 and modem 2 exhibited a similar median UL OWD of 11.8 ms and 13.0 ms, respectively, bounded by 100 ms in approximately 99.5% of the cases. The large tails indicated that in 0.1% (10−3) of the cases, the UL OWD was larger than 550 ms for modem 1, and 287.4 ms for modem 2. The multi-connectivity gains in UL OWD are smaller than in the DL OWD case. The UL OWD distribution in the multi-connectivity case showed 11.1 ms median value, with a tail that reaches 100 ms at the 99.9% ($1 - 10^{-3}$) level and exceeds 0.5 s at the 99.999% level.

Figure 25 sows the estimated RTT latency. As expected, on median level, RTT equals the sum of the median UL OWD and DL OWD contributions, for all the different configurations explored (23.3 ms for modem 1, 25.6 ms for modem 2, and 22.5 ms for the multi-connectivity scheme). As observed, RTT is heavily impacted by the UL performance, also exhibiting a large tail for modem 1 and modem 2, with a RTT latency larger than 100 ms in 1% (10^{-2}) of the cases. With multi-connectivity, the tail is reduced, exhibiting a latency lower than 100 ms in only 0.1% of the cases, and bounding the maximum RTT delay to 333.6 ms at the 99.999% level, much lower than that observed individually for modem 1 or modem 2, which were 958.8 ms and 891.5 ms, respectively.

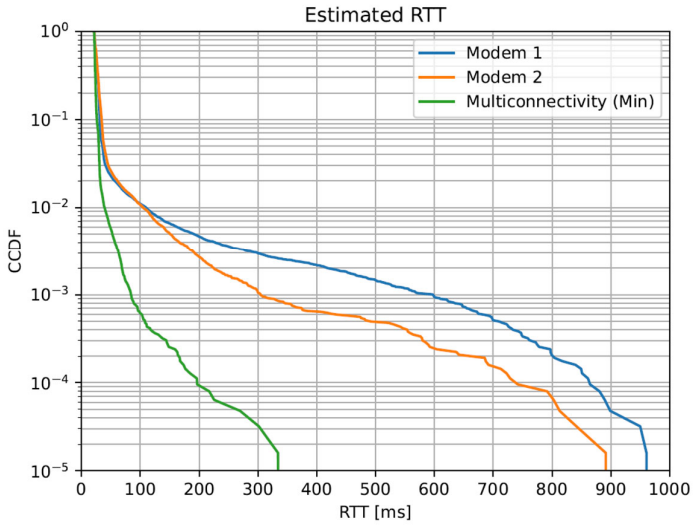


Figure 25: Empirical CCDF of the estimated RTT for modem 1, modem 2 and latency-optimized multi-connectivity.

The reliability of the different configurations was also evaluated by means of PER (or packet loss), quantified from the lost packets in the different tests. As detailed in Figure 26, PER was slightly larger in DL than in UL. PER was 0.21% in DL and 0.17% in UL for modem 1. Similar values were observed in modem 2, with a PER of 0.22% in DL and 0.12% in UL. Duplicating the information over the two interfaces of the 5G end user devices has a clear benefit, not only in terms of latency, but also in reliability of information delivery, as the PER can be reduced to values close to zero (0% in DL and 0.02% in UL) by using multi-connectivity.

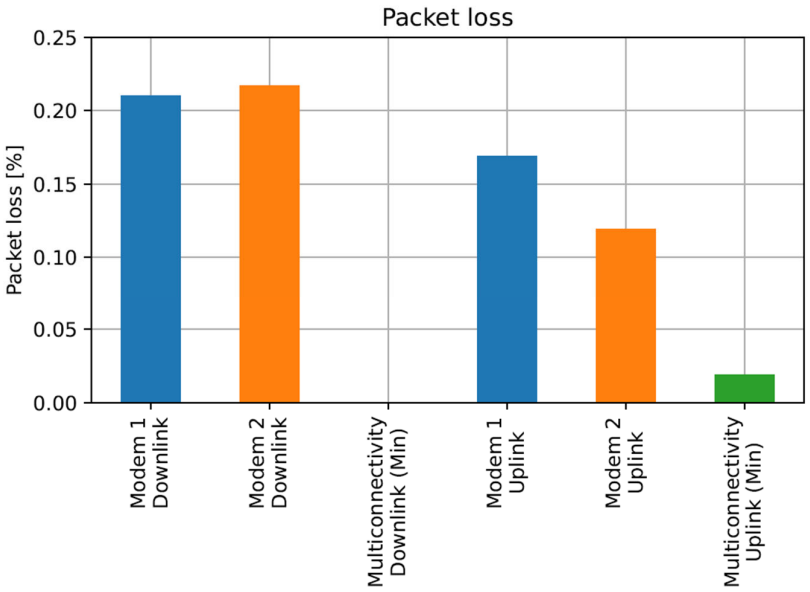


Figure 26: PER for modem 1, modem 2 and latency-optimized multi-connectivity.

In global terms, PER was better than 1% (10^{-2}) for single modem configurations, and better than 0.1% (10^{-3}) with multi-connectivity.

The results from the throughput tests are reported in terms of Cumulative Distribution Function (CDF) in Figure 27, for DL (left) and UL (right), respectively. This measurement ran into a limitation of the capacity of USB ports on the GW6405 PC, which limits the modem to 89 Mbit/s. Therefore, it should be noted that, in practice, the expected DL performance of modem 2 (if unlimited) would be very similar to the one from modem 1, reaching values above 200 Mbit/s. The presented results should still be valid, as the main target of these measurements was to understand what was the minimum level of data rates guaranteed/offered by the 5G network. This limitation has a smaller impact on the UL data rate performance, as observed in Figure 22 (right), the throughput values are lower than in DL. The maximum UL data rate measured in modem 1 was 102 Mbit/s. DL throughput was found to be larger than 1 Mbit/s 100% of the time for modem 1, and 99.8% of the time for modem 2. At least 10 Mbit/s were experienced in 95.5% and 93.5% of the cases for modem 1 and modem 2, respectively. 27.8% of the DL throughput samples measured with modem 1 presented values above 100 Mbit/s. As illustrated, the multi-connectivity scheme based on the selection of the best interface (max) leads to a slight improvement in DL throughput as well. This is since data rates benefit from optimizing the latency as we are able to transfer data faster. This effect is noticeable mainly in the tails of the distribution, where the low data rates are improved, offering at least 4.1 Mbit/s of minimum DL throughput, instead of the 0-1.5 Mbit/s experienced by individual modems. At median level, this type of multi-connectivity offers 88.7 Mbit/s, also showing some improvement as compared with the individual 75.0 Mbit/s for modem 1 and 85.3 Mbit/s for modem 2. The second multi-connectivity scheme targeting data rate-optimization (sum), leads, as expected, to high gains in throughput based on coordinated exploitation of both interfaces.

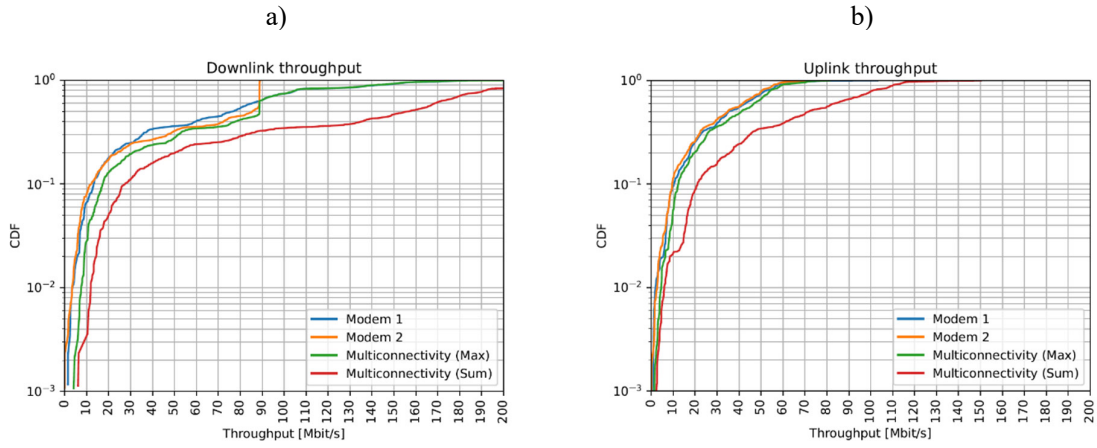


Figure 27: Empirical CDF of the DL (left) and UL (right) throughput for modem 1, modem 2, latency-optimized multi-connectivity, and data rate-optimized multi-connectivity.

With this scheme, minimum, median, and maximum DL throughput were improved to 6.0 Mbit/s, 154.8 Mbit/s, and 297.3 Mbit/s, respectively. This performance translates into 1 Mbit/s in 100% of the cases, 10 Mbit/s in 99.8% of the cases, and 100 Mbit/s in 71.1% of the cases. UL throughput values are lower than DL throughput values. UL throughput was found to be very similar for both modems, with a median value of 35.5 Mbit/s, experiencing 1 Mbit/s and 10 Mbit/s in approximately 99.5% and 91% of the cases, respectively. In terms of multi-connectivity, the scheme based on selection of the best of the two interfaces provides smaller gains in UL as compared to DL (due to the impact of the 5G network access mechanisms explained before). With this type of multi-connectivity, the minimum data rate experienced with modem 1 and modem 2, that

was 0-1 Mbit/s, is increased to 1.3 Mbit/s. Median values are slightly increased to 41.3 Mbit/s, as well as the levels at which 1 Mbit/s and 100 Mbit/s can be guaranteed, which are raised up to 100% and 94.7%, respectively. While no large gains are observed at lower data rates for the multi-connectivity case that makes active simultaneous use of both interfaces (1.8 Mbit/s of minimum throughput), enhances the median data rate to 72.2 Mbit/s and maximum throughput to 150 Mbit/s. This has a positive impact on the levels of specific data rates of interest, where 1 Mbit/s is experienced in 100% of the cases, and 10 Mbit/s and 100 Mbit/s in 97.3% and 22.8% of the cases. As it may have already been observed, some of the UL throughput tests for modem 1 and DL throughput tests for modem 2 reported 0 Mbit/s. The reason for such performance was investigated, finding that those exact tests were performed at those test route areas where poor signal conditions or eventual disconnections were experienced.

5 Algorithms and Protocols for Swarming

Drones4Safety is addressing the increasing uptake of autonomous drones for carrying out complex and potentially unsafe tasks in society. While most demonstrations of the application of drones today have involved only a single drone, the industry is rapidly advancing the transition towards the deployment of groups of collaborating autonomous drones, also known as “swarms”, driven by a demand for cost reductions. Swarming involves the coordinated operation of multiple drones to accomplish a large-scale or complex mission. Swarms may be composed of multiple drones or groups of homogeneous drones controlled by a centralized or decentralized algorithm. The benefits of swarming include improved performance on tasks that can run in parallel, the ability to perform multiple actions simultaneously in different locations, as well as increased fault tolerance.

Autonomous Inspection concerns the drone swarm’s ability to fly autonomously and carry out inspection. Inspections are modeled as a set of tasks where the complete set becomes the mission. A task is constructed in a way that a drone can perform such a task during a charging cycle. To inspect a linear part of the infrastructure the drone system can fly in well-defined formations, to allocate and coordinate tasks i.e., parts of the complete inspection missions to be carried out by individual drones. For more complex infrastructure parts, such as power pylons, a drone should be capable of determining an inspection path for its motion planning.

Communication is a foundation for drone control, data sharing and collaboration between drones. The use case concerns the use of LoRa to provide a Command and Control (C2) channel between a Ground Control Station (GCS) and a drone. As the roll-out of 5G mobile network services is progressing, it is becoming of increasing interest to validate the applicability of 5G for drone communication. Mobile operators are prioritizing network coverage in urban areas and expectedly secondary priority is given to rural areas where most of the powerline inspections are performed. It is therefore of utmost importance to validate the applicability of 5G to support drone inspection in rural areas.

At least but not last, the need for security of the drone system is further stressed from the autonomous inspection mission. Vulnerabilities and a threat analysis of the multi-drone system was provided in deliverable D5.2 [11]. The results of the threat analysis based on the Drones4Safety use cases and its influence on the multi-drone system show that the different entities/nodes as well as communication between them need to be secure. The most severe threats cover spoofing and DoS attacks. However, concerning security in the design process, techniques such as tamper-evident logging, intrusion detection, and establishment of safety protocols are as important.

5.1 Secure group management protocol for the drone swarm

The drones in the swarm are expected to exchange sensitive information among each other and with the cloud infrastructure, such as the positioning of drones and collected images from the infrastructure under inspection.

This information should be exchanged securely, making it inaccessible to external adversaries within communication range. We propose a protocol that allows a swarm to self-organize in groups, manage the membership of drones in the groups and facilitate the secure communication in these groups.

5.1.1 Network topology

Drones self-organize in disjoint groups of drones. Each group is composed of several member drones and a "controller" drone. The controller has three primary responsibilities: 1) to manage the members of its group, 2) to facilitate communication between groups and 3) to facilitate communication with the GCS. We assume that the groups form a fully connected network, implying that there is no need for routing messages since each drone can reach one another. These concepts are illustrated in Figure 28, which consists of eight drones split into two groups. The drones within a group are trusted and information exchanged between a pair of drones in the group is not necessarily intended to remain secret from the other members.

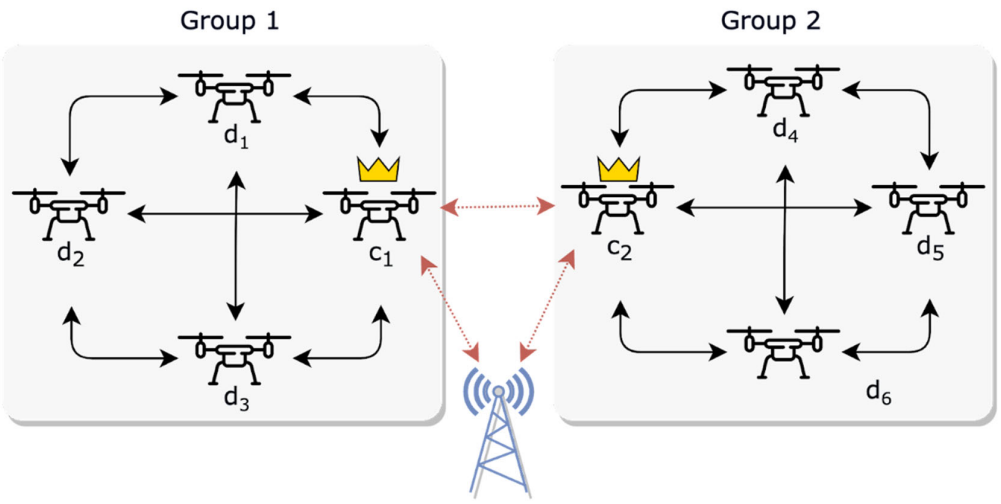


Figure 28: Network topology with two groups. Both groups consist of three member drones and a controller (depicted with a crown symbol). The radio tower represents the GCS.

5.1.2 Requirements

We define a list of requirements that drives the design of the secure group management protocol. The functional requirements are as follows:

1. Drones must be able to identify groups through broadcast messages from the controller.
2. Drones must be able to *join* a group.
3. Drones must be able to *leave* a group.
4. The protocol should handle non-responsive members.
5. The protocol should be resilient against controller failure.
6. Group members should be able to communicate among each other.

In general, these requirements ensure that a swarm of drones can form a (resilient) group and communicate among each other. We introduce a few security requirements that specify the types of security features our protocol must have:

7. The protocol should provide confidentiality.
8. The protocol should provide integrity protection.
9. Only pre-approved drones should be allowed to join groups.
10. The protocol should be resilient against replay attacks.
11. The protocol should strive for forward and backward secrecy.

These security requirements are partially derived from the CIA model [5], which describes confidentiality (protection against unauthorized information release), integrity (protection against unauthorized information modification) and availability (protection against unauthorized denial of use). The protocol focuses primarily on the first two (Req. 6 and 7). Req. 9 should prevent an external adversary from joining groups, where Req. 10 prevents the adversary from interfering with the operations of the group. Furthermore, Req. 11 demands that the disclosure of several group keys does not lead to the discovery of older or newer group keys [10].

5.1.3 Cryptographic key management

There are several cryptographic keys that are managed within the drone swarm to provide the necessary security guarantees. These keys are used in a combination of symmetric encryption, asymmetric encryption, message integrity codes (MICs) (like Zigbee [7]) and digital signatures. We denote the different keys that are being used and the purpose that each of them serve. Keys are stored on the individual drones in a hardware security module (HSM), a trusted network computer that is used to perform cryptographic operations, such as key management, key exchange, and encryption [6]. Importantly, the HSMs are tamper resistant, such that adversaries that have access to the drone hardware are unable to access the key stores in them.

Table 1: Overview of cryptographic keys used in the drone swarm.

Key type	Requirements addressed	Description and significance of key
Root key	Req. 9	Pre-approved drones must have a secret that an adversary does not have. This secret is a symmetric root key that is pre-loaded on each drone before the execution of a mission. The root key can be renewed in between missions to provide forward and backward secrecy. The root key is used in the joining process to prove that the drone is allowed to join the group, and after this process any communication is secured through the other (session) keys.
Group key	Req. 11	A shared symmetric group key is used to encrypt messages exchanged between members of the group. This key is distributed by the controller of the swarm to any new member. Any message that is encrypted using the key can be decrypted by any of the group members. This mechanism allows the group to broadcast (instead of unicasting to all other members), reducing the number of exchanged messages. The key is renewed every time a new member joins or an existing member leaves swarm. As a result, the group key has a version number, which is incremented each time the key is renewed. Only drones in possession of the most recent group key can communicate among each other.

Public/ Private (asymmetric) key pair		Each drone has its own unique asymmetric key pair. This key pair is used for digitally signing messages to prevent spoofing of messages or to encrypt messages intended for a single drone.

5.1.4 Protocol specification

Each drone in the swarm has a state from the perspective of that group's controller. Through exchanging messages with the controller, a drone changes state according to Figure 29. Drones are initially 'unjoined' and either start to advertise themselves as a controller (thereby becoming a controller themselves) or by attempting to join an existing group. drones that are a current member can choose to leave or try to resynchronize with the controller. At any point, a member can be removed from the group through a timeout. Once a drone is 'joined', it can exchange data with other members of the group (including the controller). Our proposed protocol does not implement any leader selection algorithm that establishes which drone in the swarm is assigned as the controller of groups.

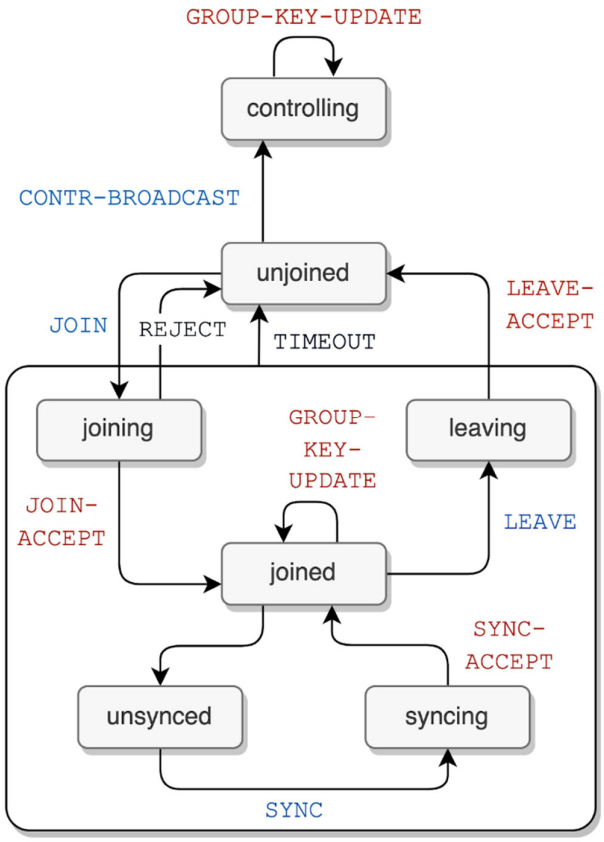


Figure 29: State machine diagram of a drone from the perspective of the group's controller. State changes occur through messages sent by the drone (blue) and controller (red) or timeouts.

.

As shown in Figure 29, drones can transition between states throughout communication with the controller. This section denotes the communication patterns that our protocol supports, or the activities that a drone and controller go through.

5.1.4.1 Act as controller

An unjoined drone can decide to become a controller and start advertising its own group by periodically sending a broadcast message. From the perspective of a controller, the controller cannot leave its own group. In case of failure of the controller, the members of its group leave the current group and establish a new one (i.e., one drone advertises as controller and the other drones immediately join it).

5.1.4.2 Joining a swarm

Any unjoined drone is aware of the nearby groups, through the controller broadcast messages. The non-member can join a group by sending a JOIN request, which - if cryptographically valid - will be accepted by the controller through a JOIN-ACCEPT message or rejected if not.

5.1.4.3 Leaving a swarm

A joined drone can leave a group via two mechanisms: 1) initiated by the drone or 2) initiated by the controller. In the former case, the drone sends a LEAVE message, which is acknowledged by a LEAVE-ACCEPT message from the controller. The latter case is triggered when a controller has not received any message from the drone for an amount of time and times out. Examples of where this can occur are a drone that flies out of communication range of the controller or failures in the drone (e.g., crashes or failing communication hardware). The drone is removed from the group forcefully by the controller and needs to initiate the join process again when entering the communication range of the controller.

5.1.4.4 Key synchronization

Packet loss is expected in wireless communication. It means that group members may miss the important group key updates. If a group member identifies that it has not received the most recent group key update from its controller, it can engage with the controller in a key synchronization process. The group member sends a SYNC request in which it must prove the possession of the former group key to the controller, which – if verified successfully – is accepted by the controller and the current group key is shared with the drone.

5.1.4.5 Group key update

Each time a drone joins or leaves the swarm, the group key is renewed by the controller and distributed to the drones. This update is sent through a single broadcast message. Even though this activity does not lead to a state change in Figure 29, it is a direct cause for a drone to fall out of sync with the controller thereby leading to a need for a key synchronization step. Since group key updates can occur frequently (in volatile group compositions), it is necessary for these messages to be disseminated to all group members simultaneously as opposed to unicasting to each member individually.

Details on the protocol message design can be found in Appendix A.

5.1.5 Validation

To validate our proposed secure group management protocol, we developed an initial implementation that follows the protocol specification. We then used this prototype to evaluate several test cases, which consist of a scenario, a number of actions to perform and an expected outcome. By applying the actions to the scenario, we can then verify whether the actual outcome matches the expected outcome.

We implemented the protocol in a Python application, in which messages are simulated as function calls, and the handling of the messages is implemented in the functions. We rely on PyCryptodome [8] for the cryptographic operations. For asymmetric encryption, RSA with OAEP padding is used [9]. For signature generation, a signature generator derived from RSA is used [9, Section 8.2.1]. The device and controller nonces

are implemented as a counter that increments, which makes the verification of the nonce simple i.e., only the largest nonce needs to be stored.

5.1.5.1 Test cases

In total, we evaluated ten test cases (TCs). Each individual test case covers a sequence of actions that the protocol is expected to encounter in a real-world scenario. For each test case, we illustrate the behavior of the protocol implementation through the logs that the implementation provides.

TC1: Join request	
Scenario	This scenario consists of a drone overhearing a broadcast from a controller and attempting to join the group. The expected behavior is a valid JOIN request being sent from the drone to the controller and a JOIN-ACCEPT response being sent by the controller.
Validation	

TC2: Join Accept	
Scenario	The JOIN-ACCEPT message is sent by the controller, correctly interpreted by the drone and that the drone is added to the swarm.
Validation	

TC3: Leave with no task	
Scenario	This test case consists of a group composed of a controller and single member. The member initiates a leaving process, while not having assigned a task. As such, the member does not require an acknowledgement from the controller. The drone sends a LEAVE request (with the ACK/NAC value set to zero) and leaves the group without waiting for a confirmation. The controller correctly removes the member from the group without sending a confirmation.

Validation	`Sending leave request to controller with ID: b'\x00\x00\x00\x01'`
	`ControllerID: b'\x00\x00\x00\x01'`
	`SwarmAddr: b'\x00\x00\x00\x01'`
	`ACK/NACK: b'\x00'`
	`DevNonce b'\x00\x00\x00\x01'`
	`MIC: b'{^\xe9a'`
	`Leaving the swarm with controller b'\x00\x00\x00\x01'`
	`Received a leave request from UAV with swarmAddr: b'\x00\x00\x00\x01'`
	`Message does not require an acknowledgement, as no task is allocated to this member`
	`Removed from the swarm UAV with DeviceID: b'\x00\x00\x00\x02' and SwarmAddr: b'\x00\x00\x00\x01'`

TC4: Leave mid task	
Scenario	In this scenario, a member drone removes itself from the group by sending a LEAVE message. As opposed to the previous scenario, the drone does have a task assigned to itself, and must therefore wait for a LEAVE-ACCEPT acknowledgement. The controller correctly receives a LEAVE request and responds with an acknowledgement. It also confirms that the member receives the acknowledgement and only afterwards considers itself a non-member.
Validation	`Received a leave request from UAV with swarmAddr: b'\x00\x00\x00\x01'` `Message needs an acknowledgement, as task is allocated to this member` `Sending leave accept message to UAV with SwarmAddr: b'\x00\x00\x00\x01'` `ControllerID: b'\x00\x00\x00\x01'` `SwarmAddr: b'\x00\x00\x00\x01'` `ControllerNonce: b'\x00\x00\x00\x02'` `Signature: b'\x92\xce\xe5u\xd0\x0c\xf5(\x86\xb4\xff1i\x10a\x99\xc6\xb0\x8b\xff\x8b\xcb\xd4\x95L`\xd9{Z\x92\x9d\xa3\xe8\xfa\x94\xca` `\xe0=ub\xc1zE)\x12\xbe\xa97\x8cO$\x13\x14\x04u\x8a\xf2\xe7\xa3.\x7fb\x88b\xee\x92\x02\x1f\x82\xcfA\xeb\t\xe6\x8a\\\x85/n7\x94oUn` `\x7f\x9f\xe8&\x85\xd6n\xf73\\\xe9\xfbX4z\x87\xb0\xa2.IW\x0c\xd3$\xf2\xa1\xb3\xd0\xf7\xaf\x98\x877\xcc\xbf\x8f\xb5\x8e\x03\x13\x8b` `\xa0\x84\xf3'` `Removed from the swarm UAV with DeviceID: b'\x00\x00\x00\x02' and SwarmAddr: b'\x00\x00\x00\x01'` `Received a leave accept message from controller with ID: b'\x00\x00\x00\x01'` `Leaving the swarm with controller b'\x00\x00\x00\x01'`

TC5: Leave mid task without notice	
Scenario	This test case simulates the scenario in which a controller that has not received a message from its member recently. A timeout should trigger the removal of the drone from the group. The controller correctly identifies the timeout trigger and removes the member from the group.

Validation	`No recent messages received from UAV with SwarmAddr: b'\x00\x00\x00\x01'`
	`UAV assumed to have left the swarm`
	`Removed from the swarm UAV with DeviceID: b'\x00\x00\x00\x02' and SwarmAddr: b'\x00\x00\x00\x01'`

TC6: Replay attack	
Scenario	This test case considers a situation in which a drone legitimately joins a group. An adversary overhears the JOIN message and replays the message. The controller must recognize this as a replay attack and reject the adversary from joining the group. A legitimate joining process and the overhearing by the adversary (Case A). So far, this behavior is identical to the JOIN request test case. When the adversary replays the JOIN message the controller correctly recognizes the message as a replay attack and rejects the message (Case B).

Validation	Case A:
	`Sending a join request to controller with ID b'\x00\x00\x00\x01'`
	`ControllerID:  b'\x00\x00\x00\x01'`
	`DeviceID: b'\x00\x00\x00\x02'`
	`DevNonce: b'\x00\x00\x00\x00'`
	`DevPubKey: b'-----BEGIN PUBLIC KEY-----\nMIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQCawnUisibxI9PkT2w99XOHC/`
	`Y9\nItLsHTjzpIRHp085jeYYNWqyymk0r6rvexe3hPIsA2WvEpAL83rq5eaOLCZhXMX7\nMehFQtZG`
	`+FJK6buxIUd36IUIsHPVu3DHjVD0PnEADRPQ2WLPi0duoxOngoWxmCwK\nnLt5MjNvWmkSeMaldQIDAQAB\n-----END PUBLIC KEY-----'`
	`MIC: b'\x1e\xb4\xf48'`
	`Malicious actor overheard and copied join request`
	`Received a Join request from UAV with DeviceID: b'\x00\x00\x00\x02'`
	`Sending join accept to UAV with DeviceID: b'\x00\x00\x00\x02'`
	`ControllerID: b'\x00\x00\x00\x01'`
	`SwarmID: b'\x00\x00\x01'`
	`SwarmVer: b'\x00\x00\x00\x01'`
	`SwarmAddr: b'\x00\x00\x00\x01'`
	`GroupKey: b'\x8b\x023$\x91\xaf\x1a\xa1\xb4\xc8\xc5"\xc6\x96\x86\xa2'`
	`ControllerNonce: b'\x00\x00\x00\x01'`
	`MIC: b'\rH\x89\x00'`
	Case B:
	`Malicious actor replays original join request`
	`ControllerID:  b'\x00\x00\x00\x01'`
	`DeviceID: b'\x00\x00\x00\x02'`
	`DevNonce: b'\x00\x00\x00\x00'`
	`DevPubKey: b'-----BEGIN PUBLIC KEY-----\nMIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQCawnUisibxI9PkT2w99XOHC/`
	`Y9\nItLsHTjzpIRHp085jeYYNWqyymk0r6rvexe3hPIsA2WvEpAL83rq5eaOLCZhXMX7\nMehFQtZG`
	`+FJK6buxIUd36IUIsHPVu3DHjVD0PnEADRPQ2WLPi0duoxOngoWxmCwK\nnLt5MjNvWmkSeMaldQIDAQAB\n-----END PUBLIC KEY-----'`
	`MIC: b'\x1e\xb4\xf48'`
	`Received a Join request from UAV with DeviceID: b'\x00\x00\x00\x02'`
	`UAV already part of the swarm checking for replay attack`
	`Message was a replay attack`

TC7: Controller failure	
Scenario	This test case evaluates the protocol reaction to a controller failure. The original controller of a group with two members stops communication and after a timeout the remaining group members must create a new group with one of the members taking the role of the controller. The members identify the failure of their controller and leave the group. One of the drones – chosen arbitrarily in our implementation – assumes the role of controller and starts advertising its group through broadcasting messages. The second former member overhears this broadcast and joins the newly formed group, matching the expected behavior.

Validation	

TC8: Group key update	
Scenario	This test case evaluates the group key updating process, which should be triggered after any group membership change. The scenario consists of a controller that initially has no members. The following occurs then: drone 1 joins the group drone 2 joins the group drone 1 leaves the group drone 1 re-joins the group After steps 2, 3 and 4, the controller is expected to renew the group key and update its current members with the updated key. Case A shows the correct behavior of a group key update process when the second drone joins the group. The last line confirms that the first drone updates its group key according to the update broadcast. Afterwards, the first drone leaves the swarm, which triggers the expected key update (Case B). Finally, the first drone re-joins the swarm again, triggering a third group key update (Case C). In all three cases, the group keys are different from one another and are distributed correctly across the group members.
Validation	Case A:

	Heard a controller broadcast from controller with ID: b'\x00\x00\x00\x01' Sending a join request to controller with ID b'\x00\x00\x00\x01' Received a Join request from UAV with DeviceID: b'\x00\x00\x00\x02' The group key has been updated, the new group key is: b'\x0e\x97\xb9jN;\x19K\x03Ip\x8e\xc7u\xe7\xd7' Sending join accept to UAV with DeviceID: b'\x00\x00\x00\x02' Received a join accept from controller with ID b'\x00\x00\x00\x01' Valid join accept received now part of swarm, b'\x00\x00\x01' Assigned swarm address is b'\x00\x00\x00\x01' New UAV wants to join the swarm Heard a controller broadcast from controller with ID: b'\x00\x00\x00\x01' Sending a join request to controller with ID b'\x00\x00\x00\x01' Received a Join request from UAV with DeviceID: b'\x00\x00\x00\x03' The group key has been updated, the new group key is: b'\x16\xe8f\x8b\xcf\x18\x94N\xd2\x80\xb2\xc1\x9b\xc9\x10\x7f' Sending join accept to UAV with DeviceID: b'\x00\x00\x00\x03' Received a join accept from controller with ID b'\x00\x00\x00\x01' Valid join accept received now part of swarm, b'\x00\x00\x01' Assigned swarm address is b'\x00\x00\x00\x02' Sending out group key update ControllerID: b'\x00\x00\x00\x01' Group Key: b'\x16\xe8f\x8b\xcf\x18\x94N\xd2\x80\xb2\xc1\x9b\xc9\x10\x7f' ControllerNonce: b'\x00\x00\x00\x04' Signature: b"~%02m\xb6\xa6VA\xfb\xb0\xaf5k\xf0\np\xeb\xbf]\xa6\t\xd2>\xe5\x0c\xbfB\x19\xa6&J4\xc9\xe1c7P\xf1QXv}~2\xadHCi \xd5\x91\xcf\xc3\x10\x88R\xc0\xb8,'J\x0e]4\r\x87\x1e\x14\xc2x\xd4\x88$\xfe\x95\xa1\xac\x98\xee\xb4CEQ\x0cv\x07\xc6\x94T/\xbf \x15\x9a\x16\xb9\xfb\xab\xfa'Z\x92~\x14\xad\x04\xb5M\xa1\x83\xa6T\x97\xe2\xb86\n?\xa7\x88\xb5T\xff\x7f\xb3p\x03n0" Received a group key update from controller with ID b'\x00\x00\x00\x01' UAV with SwarmAddr: b'\x00\x00\x00\x01' has updated group key to: b'\x16\xe8f\x8b\xcf\x18\x94N\xd2\x80\xb2\xc1\x9b\xc9\x10\x7f' Case B: UAV with SwarmAddr, b'\x00\x00\x00\x01' leaves the swarm Sending leave request to controller with ID: b'\x00\x00\x00\x01' Leaving the swarm with controller with ID: b'\x00\x00\x00\x01' Received a leave request from UAV with swarmAddr: b'\x00\x00\x00\x01' Message does not require an acknowledgement, as no task is allocated to this member Removed from the swarm UAV with DeviceID: b'\x00\x00\x00\x02' and SwarmAddr: b'\x00\x00\x00\x01' The group key has been updated, the new group key is: b"v>E\xe1ZK\x97\xb7\xea5\x89\xe7}'W\x9c" Sending out group key update ControllerID: b'\x00\x00\x00\x01' Group Key: b"v>E\xe1ZK\x97\xb7\xea5\x89\xe7}'W\x9c" ControllerNonce: b'\x00\x00\x00\x05' Signature: b'\xb5\xe7\x1e1nf,\xf9/\x06s0\x841\x1a\x00\xc2~4\xe3\xe3`\xbd\xfcG\x95\x07\xf3\x02\x04+s%A\xcd\xa0\xcc*\xe5\x1d\xbe \xb7\xf5n\x82\x8dN\x05\x8bT\xad;{\xc4\xec\|\x1b\xa2=\xe4\xbea,{4\x99\xe6\xfe\xd8\xa2E\xcf\x171K\x83\x7f\x0bc\xfd\xa2\xe7p1O\xcdq \x11\x84N\tv\xa88\xc9#[\x94\x99C\x97F\x97\xaa\xdd\xac\xe4\x7f\x14\xf9\t\xf94\x18o\xa2\xae\x10\xdaY\x13\x8cE\x1d\x00}r\xa3' Received a group key update from controller with ID b'\x00\x00\x00\x01' UAV with SwarmAddr: b'\x00\x00\x00\x02' has updated group key to: b"v>E\xe1ZK\x97\xb7\xea5\x89\xe7}'W\x9c" Case C: UAV with SwarmAddr, b'\x00\x00\x00\x01' wants to rejoin the swarm Heard a controller broadcast from controller with ID: b'\x00\x00\x00\x01' Sending a join request to controller with ID b'\x00\x00\x00\x01' Received a Join request from UAV with DeviceID: b'\x00\x00\x00\x02' The group key has been updated, the new group key is: b'\xc0\x16\x04\xa4C)X\xb0Hq\xc06r\xd1\x05\xad' Sending join accept to UAV with DeviceID: b'\x00\x00\x00\x02' Received a join accept from controller with ID b'\x00\x00\x00\x01' Valid join accept received now part of swarm, b'\x00\x00\x01' Assigned swarm address is b'\x00\x00\x00\x03' Sending out group key update ControllerID: b'\x00\x00\x00\x01' Group Key: b'\xc0\x16\x04\xa4C)X\xb0Hq\xc06r\xd1\x05\xad' ControllerNonce: b'\x00\x00\x00\x08' Signature: b'\x0e\x8c\xf0j5"\xb3\x84\x06k\x03\xb2\xc4\|\xf3\xf2\xc6\x9c\xe7\xde\x8c\xf6\xc2\xf0\xaf\xa6\xb7\xb6A:\xc8\xe8\xb5\xd02) \xc1g)\xd6$\x18\x9c\x14\r\xd83[y\x97Y2\x0f\xda\xd8\x16v\xd3\xbc9*l\xcb\xf2\x93\x08)\x16\xb3\x93\x07\xcc\xe1\xfe\xce \xa7Ff"\x91\x92\x1c\xc1a\xde\x1b0\xf7\xf4\xe5#Ql\x17\x83}\xe4\xb1\xb7`\x05\xbbyt\x14a\x8c\xb5;\xeb\xe9W\x94;\x1f\xb5\xfc\xcb\x8e \xd5M\x99)\x88\x84u\xc0}' Received a group key update from controller with ID b'\x00\x00\x00\x01' UAV with SwarmAddr: b'\x00\x00\x00\x02' has updated group key to: b'\xc0\x16\x04\xa4C)X\xb0Hq\xc06r\xd1\x05\xad'

TC9: Drone synchronization	
Scenario	This test case verifies whether an out-of-sync drone can regain the current group key. The scenario consists of an empty group, with the following activities happening: • drone 1 joins the group • drone 2 joins the group • The group key is **not** correctly updated for drone 1

	• A test message is sent to drone 1 • The first drone must recognize that it is out of sync with the controller and must initiate a synchronization process. Case A shows the first three activities, up to the group key that is not correctly distributed to the first drone. After accepting a test message, the first drone correctly sends a SYNC message to the controller, which responds with the SYNC-ACCEPT message (Case B). As a result, drone 1 correctly updates its group key.
Validation	Case A: `Heard a controller broadcast from controller with ID:  b'\x00\x00\x00\x01'` `Sending a join request to controller with ID b'\x00\x00\x00\x01'` `Received a Join request from UAV with DeviceID: b'\x00\x00\x00\x02'` `The group key has been updated, the new group key is: b'{\x1cZS\xb1\xf3\x0e\n\xeb\xe7i\x0cC\xcc\xb6``'` `Sending join accept to UAV with DeviceID: b'\x00\x00\x00\x02'` `Received a join accept from controller with ID b'\x00\x00\x00\x01'` `Valid join accept received now part of swarm, b'\x00\x00\x01'` `Assigned swarm address is b'\x00\x00\x00\x01'` `Heard a controller broadcast from controller with ID: b'\x00\x00\x00\x01'` `Sending a join request to controller with ID b'\x00\x00\x00\x01'` `Received a Join request from UAV with DeviceID: b'\x00\x00\x00\x03'` `The group key has been updated, the new group key is: b'\xf9_\x97O\x88\xc2c5\xd3\xe1\xdf\x81\xc2!\x83\xe8'` `Sending join accept to UAV with DeviceID: b'\x00\x00\x00\x03'` `Received a join accept from controller with ID b'\x00\x00\x00\x01'` `Valid join accept received now part of swarm, b'\x00\x00\x01'` `Assigned swarm address is b'\x00\x00\x00\x02'` Case B: `Sending test message` `The UAVs version is out of sync by one version` `Controller received a test message` `Sending resync message to controller with ID: b'\x00\x00\x00\x01'` `ControllerID: b'\x00\x00\x00\x01'` `SwarmAddr: b'\x00\x00\x00\x01'` `DevNonce: b'\x00\x00\x00\x01'` `MIC: b'xD\xd6\x0f'` `Received a resync request from UAV with SwarmAddr: b'\x00\x00\x00\x01'` `Sending resync accept message to UAV with SwarmAddr: b'\x00\x00\x00\x01'` `ControllerID: b'\x00\x00\x00\x01'` `SwarmAddr: b'\x00\x00\x00\x01'` `GroupKey: b'\xf9_\x97O\x88\xc2c5\xd3\xe1\xdf\x81\xc2!\x83\xe8'` `ControllerNonce: b'\x00\x00\x00\x04'` `Signature: b'\x12\xde\xd2\x04cQ\xa1\xdb\n\xf2\xbe\x08\xa1,\x8a\xa2\xf3\xfa\xf5\xf0\x8a\xb3\xbfd\xb9[\xc5\x06hX8\xc7\x0b\x199vF` `\xd9\x80\xd5i\x98A\x89\xc1\xe4\xe9\xbc\xa4\t\x98\x8f\xe8PtT\x00\xac\t\x9a\xdd\x11\x9e\xc7\x9b\x07\x16\x85\xb7\xa7\x8e\xb2\x9fYI` `\x85"\xe1?\xb64\x80a\xf6o#*\x8e\xb4G \xdfN\x1a\xe2\x15F\x96\xfd\x92"\x14\xc6\xb1\xb7\x953\x84\x8d\xa4\x9b,\xfd\x17Sq\xe7L\x02\xcd` `\xd2\x1dy6\xa9\xd0\x97\xd5'` `Received a resync accept message from controller with ID: b'\x00\x00\x00\x01'` `UAV with SwarmAddr: b'\x00\x00\x00\x01' has been resynched and updated the group key to: b'\xf9_\x97O\x88\xc2c5\xd3\xe1\xdf` `\x81\xc2!\x83\xe8'`

TC10: Wrong MIC	
Scenario	This test case evaluates whether the implementation correctly responds to an invalid MIC. In this scenario, a drone wishes to join a controller, and through a (simulated) error, one of the bits in the join message payload flips. The controller should recognise a MIC error and reject the join request. The calculated MIC over the message (`9+\\\xd2`) does not match the transmitted MIC (`a\x9e\x84\xdb`), and that the message is rejected correctly.

Validation	`Heard a controller broadcast from controller with ID:  b'\x00\x00\x00\x01'`
	`Sending a join request to controller with ID b'\x00\x00\x00\x01'`
	`MIC: b'a\x9e\x84\xdb'`
	`Something went wrong in the communication and a bit error happened`
	`Received message with MIC: b'a\x9e\x84\xdb'`
	`Calculated MIC is: b'9+\\\xd2'`
	`Incorrect MIC, the message or key is incorrect`

We propose a group membership management protocol, in which a group of nodes self-organize themselves into a group under the leadership of a 'controller' node. The protocol allows nodes to join groups when in possession of a shared 'root key' and leave when necessary. Communication between group members is facilitated by asymmetric encryption using a shared 'group key'. This group key is renewed whenever the composition of the group changes, ensuring that new members cannot discover preceding group keys (thereby providing backward secrecy in the protocol). We validate a proof-of-concept implementation of our proposed protocol by subjecting it to several test cases.

5.2 Motion path planning

5.2.1 Global path planning

Global path planning refers to high-level path planning using the data known prior to the autonomous mission. For the scope of the Drones4Safety project, the data involves infrastructure locations gathered from the Open Street Map (OSM). As the project aims at developing a continuous autonomous operation of inspection drones, we implemented a high-level planning strategy that ensures that drones always fly near the infrastructure. Such a strategy was chosen to avoid flights in residential areas and facilitate obtaining flight permits in the future. For routing to the inspection infrastructure, we implemented a graph structure representing the locations of power towers and power lines. As initial research showed, around 70% of bridges are located within 3 kilometers of the power and rail infrastructure. We added bridge locations to the graph, enabling the drones to autonomously reach a bridge. The addition of bridge locations required the implementation of the algorithm for finding central points in polygons that represent bridges, extracted from the OSM. Each central point was connected to the closest node in the graph and all nodes were connected with weighted edges. Each edge was represented with three elements: the nodes it connects, the distance between the nodes, and the penalty factor. The penalty factor is increased for those edges which traverse over the open air instead of power lines to avoid flying away from the infrastructure whenever that is possible. The weight is computed by multiplying the distance with the penalty factor and is used by the A* algorithm when finding the most proper path. Therefore, the algorithm will not always find the shortest path. Instead, it will check if the path above the power lines is not drastically longer than the shortest one above the open air and, if it is not, choose the one above the lines.

The algorithm was implemented in microservices where each service provides a functionality. Services are containerized and deployed to the cloud in a Kubernetes cluster. A user interacts with services through the web interface exposed to the internet. Bridges and towers included in the graph are visualized on the interface for the user to select routing/inspection targets. As seen in the figure below, power towers are represented as light blue circles, drones are in orange, and bridges in purple. When a user selects a target location, a green circle appears as well as the option to calculate the route. The route calculation services are engaged, and the final route is visualized in red.

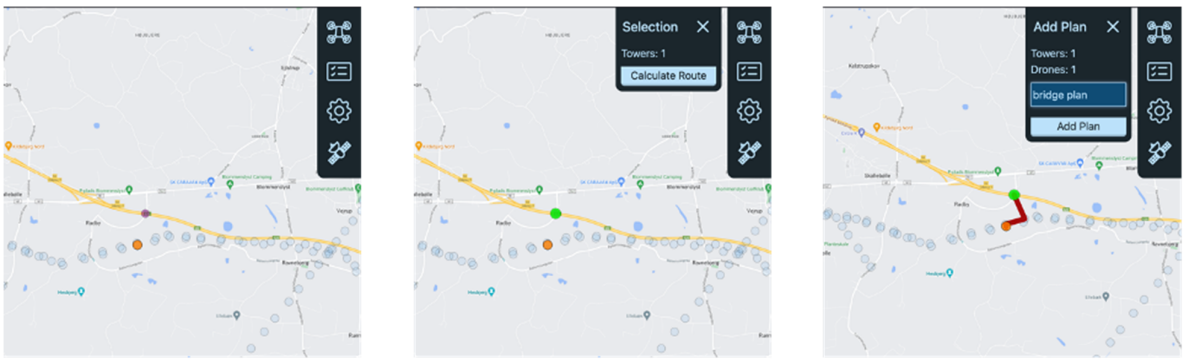


Figure 30: Examples of output of the global path planning.

5.2.2 Collaborative path planning

This work considers a system consisting of multiple drones connected within a wireless network. We address the motion planning problem of moving multiple drones such that they 1) keep themselves out of collisions with other drones or obstacles in the environment. 2) can be guided and generate behaviors according to team-level goal definition, e.g., a goal position for a team of drones without goal assignment for each drone.

We propose a game theoretical framework for drones' path coordination. The framework extends existing optimization-based drone path plan solutions by involving the theory of Generalized Potential Games to directly plan according to a high-level goal description for a team of drones.

Our proposal is formulated as a sequence of multi-agent games seeking for equilibrium solutions, which generates an approximate optimal solution for a team of drones. The computation of equilibrium guides each drone in the multi-drone system to calculate its control inputs subjected to constraints, e.g., robot dynamic limitations and obstacles. We developed a docker based platform for simulating multiple drones (UAVs) and evaluated the motion planner on collision avoidance in numerical and Software-in-the-Loop (SITL) simulations.

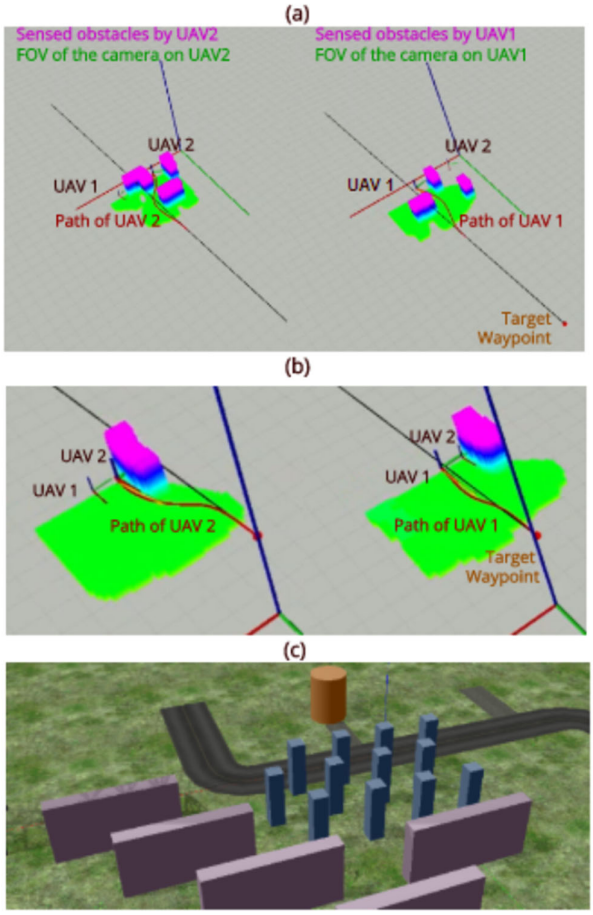


Figure 31: Coordinated path planning for two drones (UAVs) in an obstacle existing environment. (a) Two UAVs are represented by their body coordinate frames in the RViz. (b) The UAV 1 plans a path to keep safe distance to the UAV 2. (c) The simulated environment in Gazebo.

In Figure 31 each drone (UAV) senses the obstacles within the field of view (FOV) of the on-board cameras. Two drones plan the path to fly towards a predefined waypoint while avoiding obstacles.

Figure 31a shows an example that two drones (UAVs) plan paths, in red curve, towards a target team-level waypoint position while avoiding encountered obstacles. The drone (UAV) is controlled by position and velocity commands. Obstacles are sensed in real-time with the RealSense D435 model and visualized as voxels. The visualization of the UAV 1 and 2 are shown on the right and left side on Figure 31a, respectively. Due to the limited field of view of the camera and the diversity of the UAV's locations, visualized obstacles are different.

The target team-level position waypoints are set by the user and shared to drones through the local network using *multimaster_fkie* package. Cooperative path planning algorithm is triggered if the drone-to-drone distance is less than a predefined coordination distance. Within the coordination distance, the UAV updates its path to keep a safe distance from the other UAVs. Figure 31b shows the path of the UAV 1 being pushed away by the path of the UAV 2 due to the safe distance constraints during the path generation.

5.2.3 Inspection path planning

A novel inspection path planning method for achieving a complete and efficient inspection using drones was designed. The method uses a point cloud generated from a 3D mapping service to represent complex inspection

targets and provided as the input of the path planning method. The method is designed as a sampling-based sequential optimization to calculate and optimize an inspection path while considering the limitations of the sensors, inspection efficiency, and safety requirements of the drones.

5.2.3.1 Inspection Path Planning Pipeline

The path planning pipeline consists of three modules (Figure 32). It reads a point cloud set, e.g., PLY file, as input and exports a sequence of waypoints with related sensor orientations, e.g., camera angles.

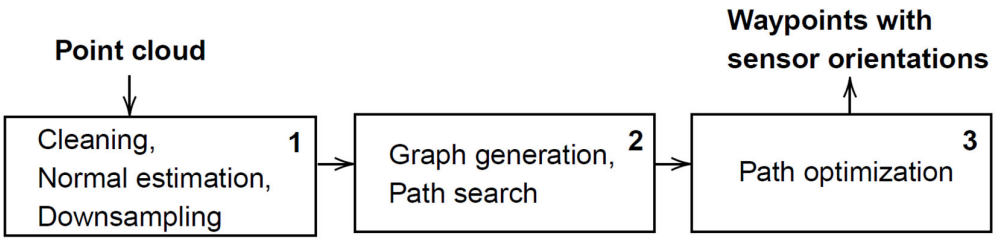


Figure 32: Overview of the path planning pipeline for inspection.

Step 1 works on data cleaning, normal vector estimation, and downsampling. Step 2 generates a graph based on downsampled data and then searches a feasible traversal path for the graph. Based on the traversal path and the normal vectors, step 3 constructs the optimization problem to find and optimize the flight waypoints and sensor orientations.

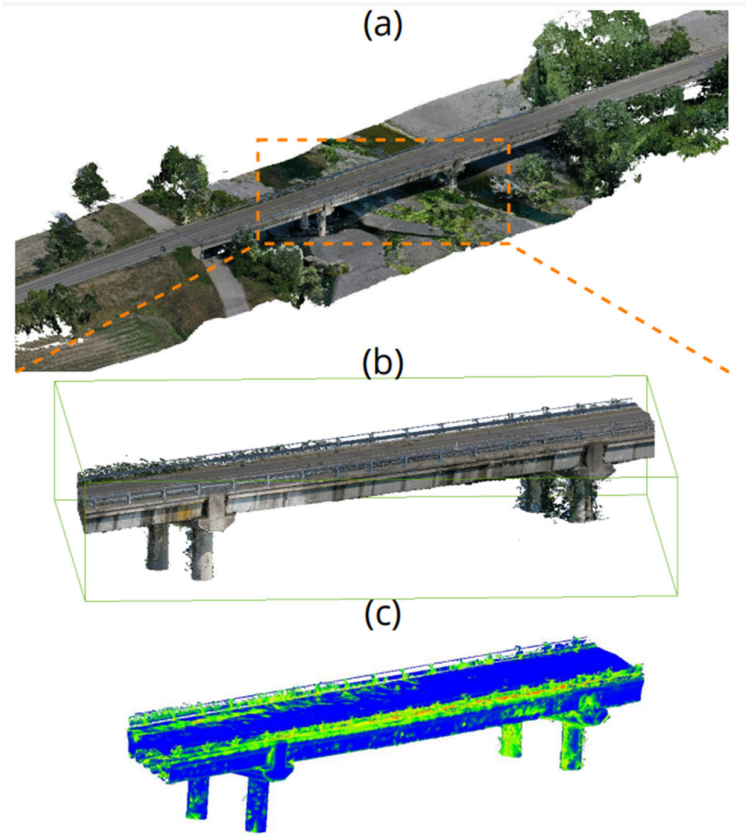


Figure 33: (a) A point cloud (PLY file) is generated by the 3D mapping service. (b) The inspection area is selected and cropped. (c) The point cloud is cleaned and smoothed. The smoothness is represented by the difference in color.

We evaluate the proposed inspection path planning method using a bridge in Italy [1]. We assume a 3D map of the inspection target is provided as a point cloud file (Figure 33).

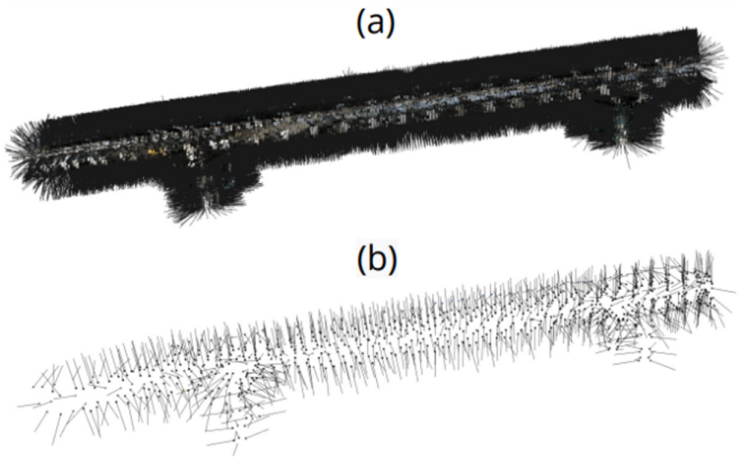


Figure 34: (a) A normal vector of each point is estimated and oriented perpendicular to the consistent tangent plane. (b) The point cloud with its normal vectors is downsampled using voxelization.

Afterward, points are sampled from the mesh with a predefined surface-point density. In addition, their normal vectors are estimated (Figure 34) from examining adjacent points in the point cloud.

The drone flight becomes efficient when the guided path is simple, e.g., a path that avoids repeating trajectories or directional changes. It enhances the safety to leave more margin to react to unexpected situations in the field. However, a path search, e.g., finding the shortest traversal path from a graph representation of an inspection target, often leads to complex results. To control the complexity of the path search a segmentation step is involved. The point cloud is segmented into 6 clusters based on the normal vectors of the 6 facets of the oriented bounding box (Figure 35a). Figure 35b represents an example of a segmented point cluster of which the point normal vectors are close to the normal vector of the top facet of the bounding box .

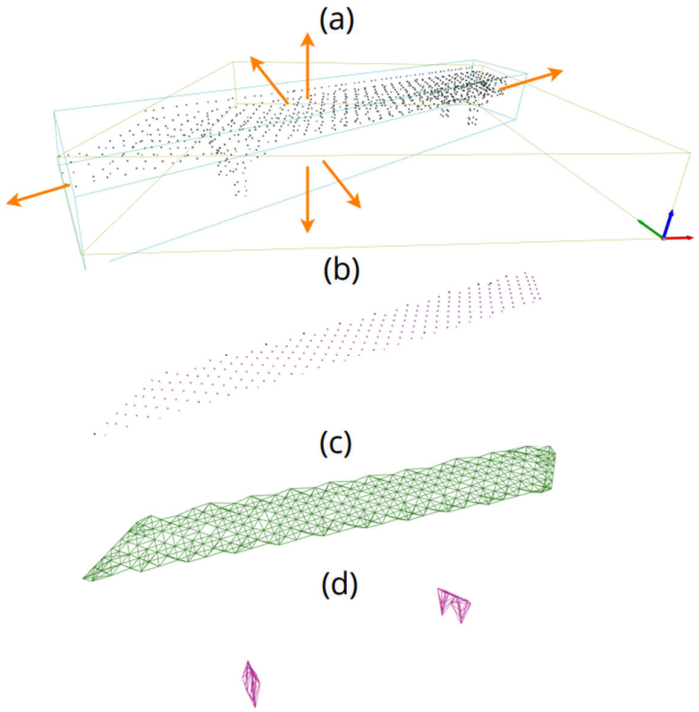


Figure 35: (a) The point cloud is clustered based on the normal vectors shown with orange arrows, of the oriented bounding box in blue. The axis-aligned bounding box is shown in yellow with a local coordinate system visualized in the bottom right corner. (b) A cluster of the points that represents the top part of the bridge. (c) A graph is generated based on the point cluster in (b). (d) A graph generated from a different point cluster contains two components, i.e., two induced subgraphs in which any two vertices are connected to each other by paths.

By searching paths for 6 clusters of the segmented point cloud based on the generated graphs, the inspection sequences for 6 clusters are calculated. As the output of the second step, Figure 36 presents the segmented cloud point (6 clusters), normal vectors, and the traversal path, which guaranteed a complete visit of all points. Each point cluster is visualized with the axis-aligned bounding box. The axis represents the local coordinate system.

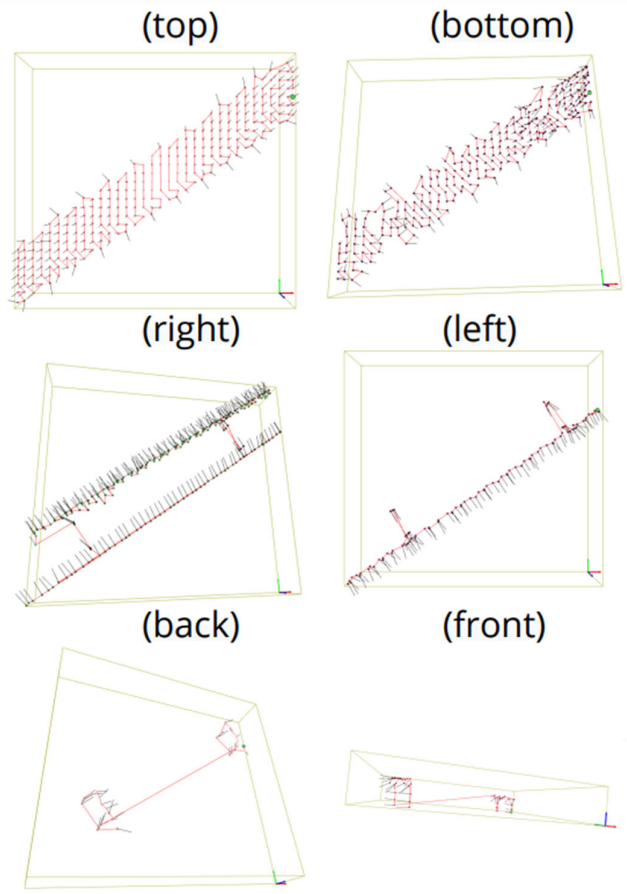


Figure 36: An efficient path, shown in red, is searched to fully visit all points in each point cluster of the bridge, which is generated based on normal vectors representing 6 different directions. The green sphere, in the top left corner, denotes the start point of the path search.

To control the complexity of the inspection path, task segmentation is involved. The graph is generated for path search, which searches a traversal path that defines the inspection sequence. In Figure 36, the red line shows the traversal path of each segment, while the green point represents the beginning of the sequence. At last, the optimized inspection path of each segment is visualized in blue (Figure 37).

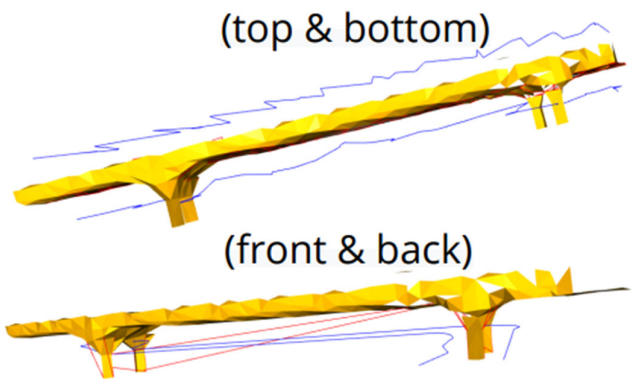


Figure 37: The bridge is represented as a triangle mesh in yellow. Red lines indicate the result of the path search. Optimized inspection paths for the bottom-side, front-side, back-side, and front-side of the bridge are presented in blue.

5.2.3.2 Path optimization algorithm

The path optimization step converts the inspection sequence (traversal path) to an efficient flight path while considering safety constraints, and sensor limitations. The flight path is optimized in terms of the flight distance and the path smoothness. The path optimization is constructed as a sequential convex optimization by combining point cloud data, normal vectors, and the traversal path. Mathematically the optimization problem can be formulated as follows:

$$\min_{\Delta_j} \quad \sum_{j=1}^{J} \|\Delta_j\|_2 + \beta \sum_{j=1}^{J-1} \|\Delta_{j+1}^{x,y} - \Delta_j^{x,y}\|_2 + \mu \sum_{j=1}^{J-1} \|\Delta_{j+1}^{z} - \Delta_j^{z}\|_2 \tag{2}$$

$$\begin{aligned} s.t. \qquad g_1 &= P_{start} + \Delta_1 \\ g_{j+1} &= g_j + \Delta_{j+1}, \ \forall j \in \{1, 2, ..., J-1\} \\ (g_j - v_{j1})^T n_j &\geq d_{min} \\ -(g_j - v_{j1})^T n_j &\geq -d_{max} \\ (g_j - v_{ji})^T h_{ji} &\geq 0, \ \forall i \in \{1, 2, 3, 4\} \end{aligned} \tag{3}$$

Where $\Delta_j = [\Delta_j^x, \Delta_j^y, \Delta_j^z]$ denotes the position control variable in 3D space. $\Delta_j^{x,y}$ denotes the *x* and *y*-axis components of Δ_j. Δ_j^z denotes its *z*-axis component. β and μ are constant values that represent the scalar of the constructed cost function. P_{start} indicates the start position of the inspection task. $g_j = [g_j^x, g_j^y, g_j^z]$ denotes the position of j^{th} viewpoint, where the inspection sensor e.g., the camera, is triggered to sense an area defined based on the j^{th} sampled point in the point cloud. The number of sampled points is *J*. An efficient traversal sequence of the sampled points is calculated at the path searching step. The equation above describes the objective function for the path optimization, which minimizes the total path length and the change of the consecutive position control variable at the *x, y,* and *z*-axis. We select β*=1* and μ*=10* to punish more for the difference generated from the *z*-axis, i.e., the altitude of the drone.

The constraints begin with equality constraints, defining the initial value and the update function for the calculation of viewpoint g. Then the inequality constraints represent the incidence angle limitations, the safety minimal distance, and the maximal inspection distance. Inequality constraints are visualized in Figure 38, where the incidence angle limitations are constructed by the inner product between the vector (g_j-v_{ji}) and the normal vector h_{ij} of the hyperplanes. The hyperplane is defined by the incidence angle and the edge of the square constructed perpendicular to the normal vector n of the point p.

The orientation of the sensor for each inspection area is defined by g and p, where the sensor is placed at the position of g and oriented towards the position of p, i.e., the sensor orientation is settled by the vector (p-g).

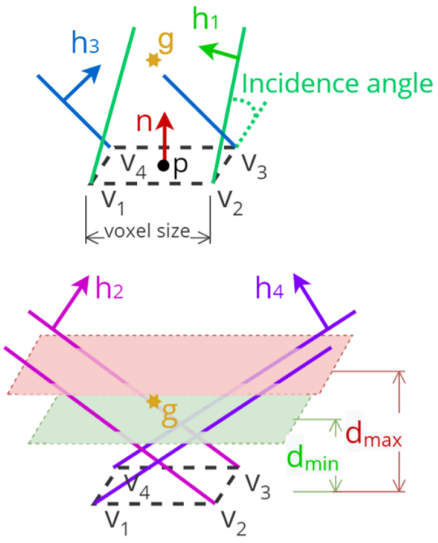


Figure 38: The constraint space for the path optimization.

As shown in Figure 38, a convex constraint space is constructed. p is a point in the downsampled point cloud. n is its estimated normalized normal vector. Marks $v_i, i \in \{1,2,3,4\}$ denote the vertices of the constructed square, of which the center is the position of p, the length is defined by the voxel size in the voxel downsampling step. Vectors $h_i, i \in \{1,2,3,4\}$ represent the normalized normal vectors of the hyperplanes, which are constructed based on edges of the square and the incidence angle constraint, e.g., h_1. Two hyperplanes in green and red parallel to the square are constructed at the side defined by the normal vector n and the distance defined by d_{min} and d_{max}. The *g denotes the viewpoint, of which the position is to be optimized subject to the constraints.

5.2.3.3 Experiments and validations

A Python-embedded modeling language, CVXPY [29], is used for modeling the sequential convex optimization problem. The embedded conic solver (ECOS) [30], an interior point solver, is mainly used for calculating the optimal result. Figure 37 below presents the result of the path optimization, where the incidence angle limitation is set to 30 degrees, the minimal safety distance, and the maximal inspection distance (d_{min} and d_{max}) are set to 1 and 2.5 meters, respectively.

It is recommended to deploy the proposed processing steps on a cloud computing platform, which provides sufficient computation power for path search, path optimization, as well as point cloud visualization, especially for large-scale inspection tasks. Cloud deployment also leads to convenient integration with the 3D mapping service, e.g., webODM, which provides input data for the processing chain. The processing chain is designed to provide path guidance for the inspection task. It is sufficient to generate the path before the task is stated and then send the path data to autonomous drones as global guidance. The drone is expected to have an on-

board local path planner for waypoints following and obstacle avoidance to respond to unexpected turns of events during the inspection.

5.2.3.4 Computational performance analysis

A computation analysis is presented to provide an overview of the time consumption of the proposed method. The evaluation of the computation uses Intel i7-8650 CPU as the platform. Figure 39 presents the time consumption of the path optimization and the path searching for processing different sections of the inspection target.

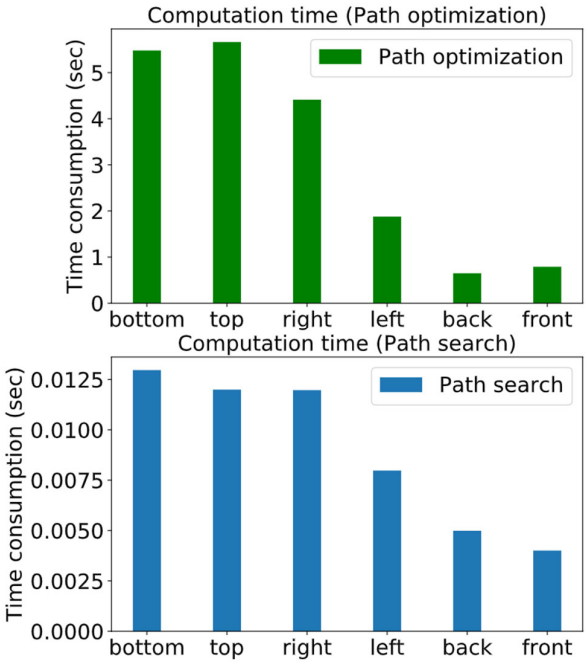


Figure 39: Computation time of the path optimization (top) and path search (bottom) for different decomposed point clusters.

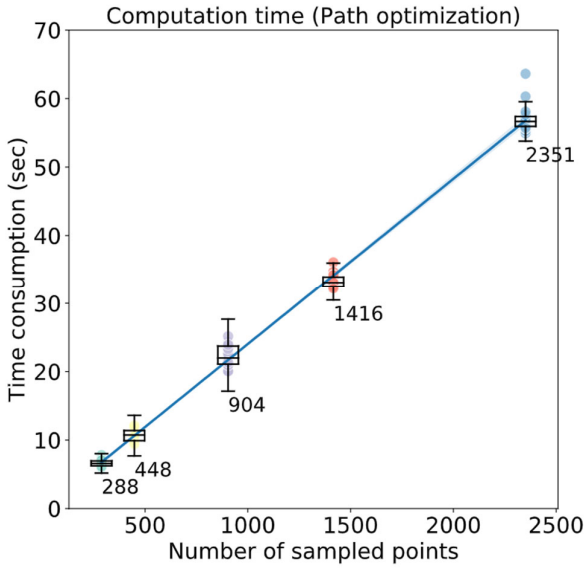


Figure 40: Computation time of the proposed path optimization algorithm. The labels below the scatter plot points indicate the number of sampled points. A linear regression yields a value of 0.9945.

The total time consumption of the described method is presented in Figure 41. The time consumption shows a linear increase with the sampling points. The total number of sampled points depends on the voxel size, a parameter for the voxel downsampling. Most of the time consumption is used for processing path optimization. Normal estimation, graph generation, and decomposition are the subsequent steps that are time-consuming.

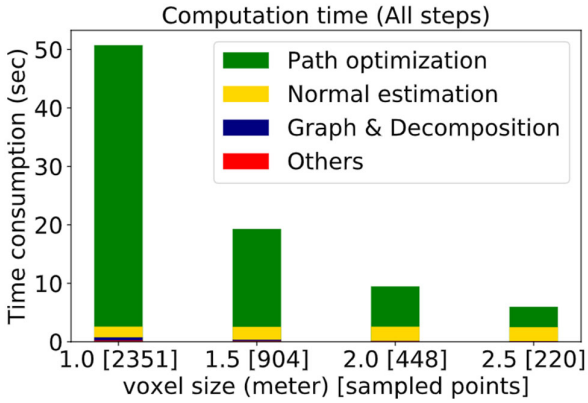


Figure 41: The total computation time of the proposed method in terms of the voxel size. The number of the sampled points using voxel downsampling is given. The time consumption of different steps is provided individually to show the distribution. The red section (Others) includes steps of voxel downsampling, bounding box calculation, and path searching.

To evaluate the scalability of the method, we measured the computation time of the path optimization for 5 different sizes of the problem, 20 measurements for each problem. Modeled as a sequential convex optimization. The proposed optimization algorithm has O(n) time complexity. It is observed in Figure 41, a linear relation between the computation time of the path optimization and the number of sampled points,

reflecting the size of the problem. The number of sampled points is influenced by the voxel size parameter required by the voxel downsampling step.

5.2.4 Reactive motion planning

The collision avoidance is evaluated by measuring the minimum rope to obstacle distance during the spray-painting task. We collect data while the end-effector is moving around a generic 3-dimensional obstacle.

Figure 42 presents three different behaviors of the drone together with the rope (i.e., A, B, C) reacting to the same position (shown in red circle) of the end-effector with different combinations of policies:

- Case A: both *drone-collision-avoidance* and *rope-collision-avoidance* are enabled.
- Case B: the *drone-collision-avoidance* is enabled.
- Case C: no collision avoidance.

In case A, the distribution of the minimum rope to obstacle distance has been maintained above 0.2 m with a median value of 0.4 m. In case B, the median value of the minimum rope to obstacle distance is higher than the one in scenario C since the drone collision avoidance may move the tethered rope away from the obstacle in some cases. The minimum rope to obstacle distance for case C is mainly distributed between 0 to 0.4 m with a median value close to 0 m.

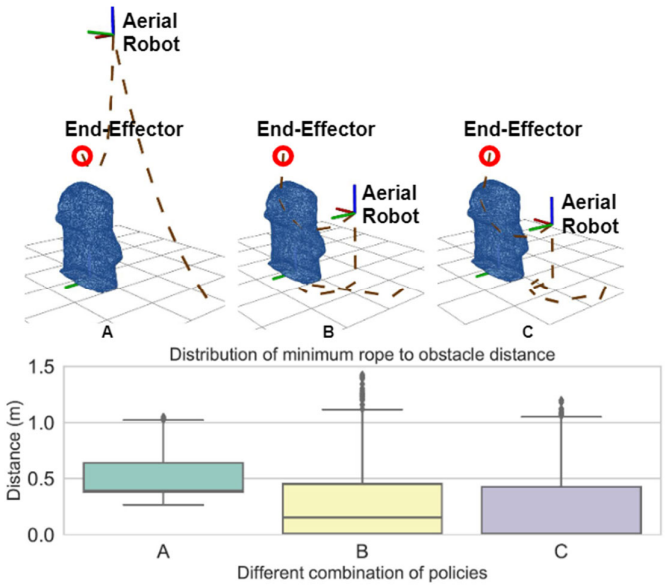


Figure 42: Collision avoidance performance is evaluated under three different combinations of policies. The policy combination grows one by one from C to A. In the scenario of C, goal following policy and straight forcing policy are enabled. B: The drone collision avoidance policy is enabled in addition to the policies used in the scenario of C. A: Furthermore, the rope collision avoidance policy is enabled.

We evaluated the computation of the reactive motion planner. The proposed planner is highly efficient and can easily be run at reactive, real-time frequencies without special hardware. Figure 43 shows the time cost of the top-3 time-consuming functions for each motion planning iteration (20 Hz).

On a computer with an Intel i5-4200M CPU and 8 GB memory, the planner uses approx. 5-7% of one CPU core, measured using *rqt_top*. Of this computational budget, about 33% are spent on motion planning and

integrating accelerations, 45% on obstacle detection and mesh operations using CGAL, and 16% on the rope simulation, profiled using *callgrind*.

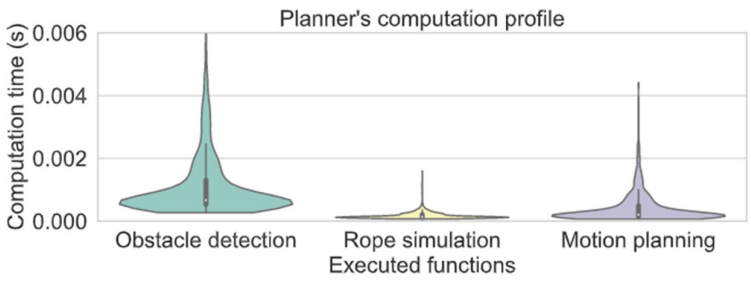


Figure 43: The computation time of top-3 time-consuming executed functions in the planner software: motion planning, rope simulation and obstacle detection.

5.3 Task allocation

5.3.1 Centralized - cloud services

As part of the global path planning strategy, additional service to determine the order of multiple drones visiting multiple towers has been developed. The figure below shows services with additional VRP service solving the vehicle routing problem [15]. The service receives inspection targets and drone locations and uses OR-Tools to determine which drone should fly to which target. It creates requests to the A* service asynchronously to get the shortest path for each combination of drone and target. Then, it builds the distance matrix and, based on path length, finds a near-optimal solution to the vehicle routing problem. It returns the path for each drone as a set of waypoints containing locations in latitude and longitude.

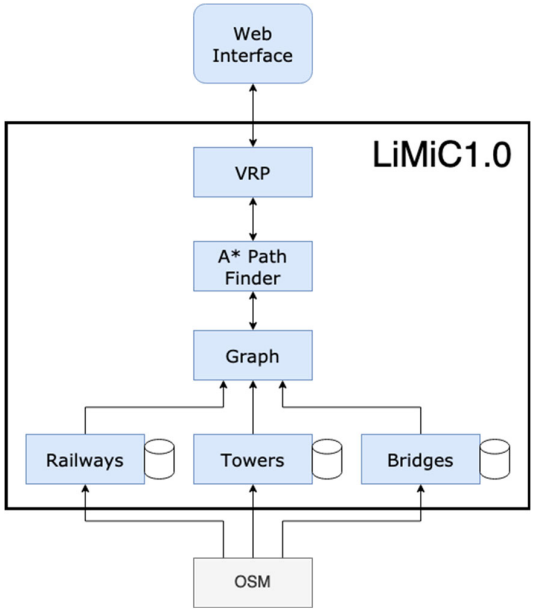


Figure 44: Application structure of LiMiC1.0 cloud based on capabilities [15].

When a user selects multiple inspection/routing targets on the web interface, the near-optimal order of visiting is determined, and routes are visualized as in the figure below. Final routes can be stored or sent to the drones

to start executing the mission. The cloud platform developed in WP6 enables mission planning, execution, monitoring, and data exchange with drones. It provides data storage, therefore, creating a global data space for sharing a priori knowledge about the environment with drones [16].

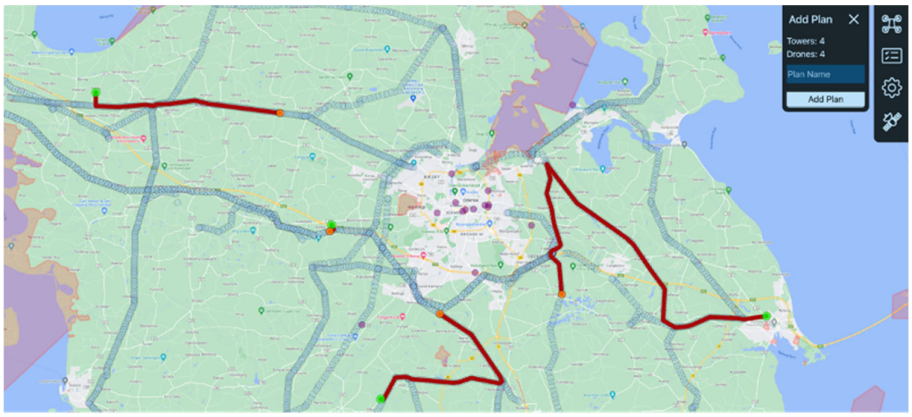


Figure 45: Web interface showing calculated inspection routes in red, for each UAV reaching a target [16].

5.3.2 Charging protocol

The limited battery capacity of a drone results that the inspection of infrastructure cannot always be completed without any recharging mechanism. Depending on the type of mission, the drones charge on overhead power lines (for railway inspections) or at a designated charging station (for bridge inspections). In the latter case, the charging station is assumed to be capable of serving a limited number of drones, and as such it becomes a shared and limited resource. It then becomes necessary to produce a schedule that allocates the charging stations to the drones, and the quality of the schedule impacts the execution time of the mission. We introduce a charging schedule protocol that aims to minimize the mission execution time.

We define the allocation of charging stations to drones as an optimization problem. In this problem, we assume that each drone has a set of three-dimensional waypoints that form a sequence that the drone needs to take. Near these waypoints there are a few fixed charging stations located, whose location is known to each drone and each of which can charge one drone simultaneously. The velocity and battery and charging depletion rates (percent per second) are assumed to be constant and known in advance. Given this situation, the objective of the optimization problem is to:

"Schedule the path of each UAV along its waypoints and charging stations to minimize the mission execution time"

As constraints to this problem, during the mission (1) the batteries of the drones cannot fully deplete and (2) each charging station cannot serve more than one drone at any point during the mission.

In our problem definition, each drone must follow a *path*, which consists of its allocated waypoints and visits to charging stations. For each allocated waypoint, the optimization model decides to directly move to the next waypoint, or visit one of the charging stations (cf. Figure 46). Furthermore, the model should decide how much time each drone must (1) wait before charging and (2) charge its battery.

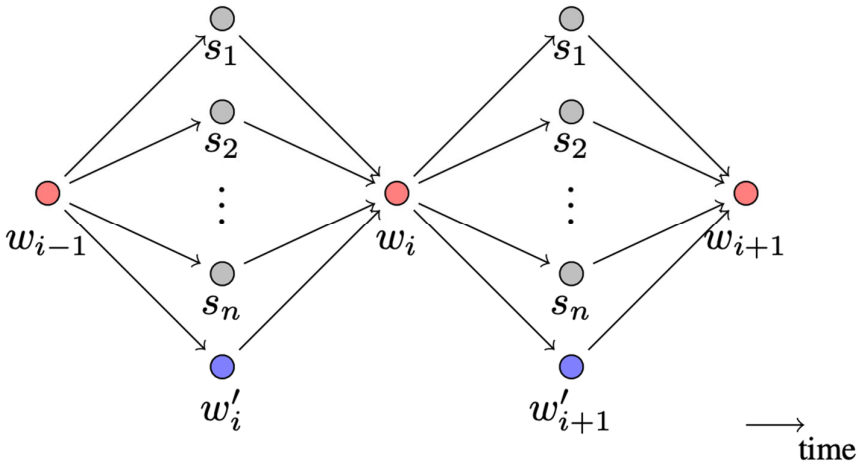


Figure 46: The different paths a drone can take between three of its waypoints (w_{i-1}, w_i, and w_{i+1}). After each waypoint, the drone has the option to visit any charging station (gray nodes) or move directly to its next waypoint (blue nodes).

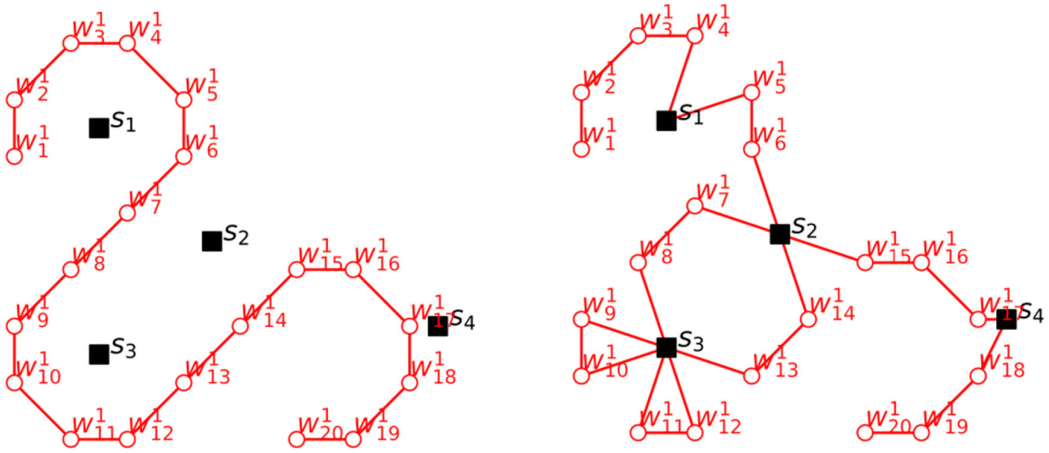
5.3.2.1 Model formulation

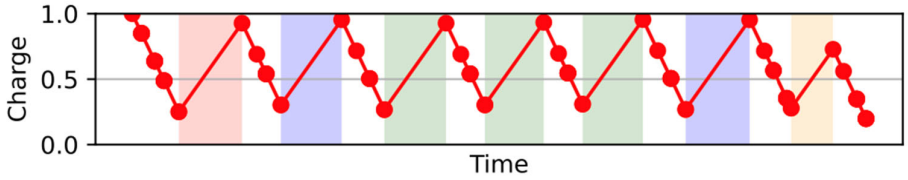
We define the problem as a mathematical problem, as a Mixed Integer Linear Programming (MILP) program. Such a problem consists of an objective function of decision variables, which can be minimized or maximized by a solver. This objective function is subject to a set of constraints, which put restrictions on the decision variables. These problems and algorithms are well studied and there are software implementations of the algorithms available for solving the problem, i.e., finding the optimal composition of variables that minimizes our objective.

The mathematical model of the optimization problem is provided in Appendix B.

5.3.2.2 Verification and validation

We show the outcome of our scheduler by implementing the model in the Pyomo [3] library, using the Gurobi solver to find the optimal solution [4]. We manually define a sequence of waypoints (in a two-dimensional space for visualization) and define the charge rate, charge depletion rate and velocities such that the drone must charge a few times during the mission execution. In this example, the minimum battery charge (B_{min}) is set to 20%. Figure 47 shows the outcome of solving a simple example problem. The sequence of waypoints that the UAV follows is shown in (a), and the resulting path – including the charge locations – in (b). The battery charge over time (c), illustrates that the battery never falls below the minimum charge rate.



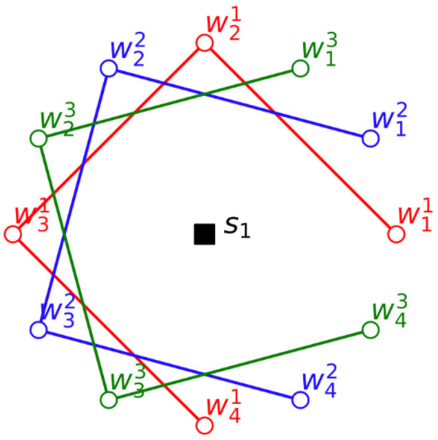


(c) Battery profile when following the optimal schedule. The background bars represent the period at which the period is charging and the visited charging station (defined by the color).

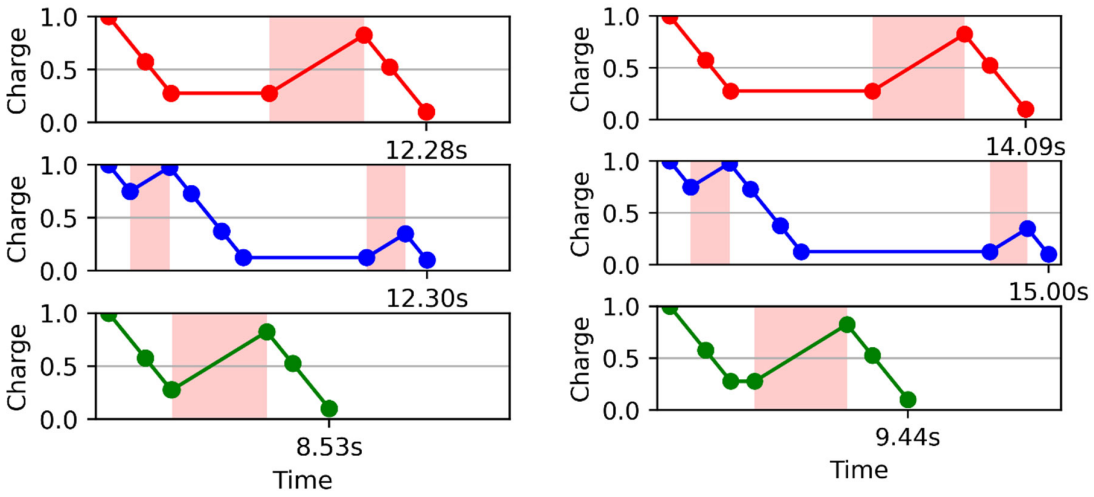
Figure 47: Example of the optimal solution for a single drone, following twenty waypoints and four charging stations along the way.

5.3.2.3 Verifications and validation

Similar to the initial model, we manually define a simple problem and find an optimal charge schedule. In this case, we define three drones, each circling around a single charging station following four waypoints. Again, the (dis)charge rates and velocities are chosen such that the drones must charge at least once. The velocities of the three drones are 1, 1.2 and 1 m/s respectively. Figure 48a shows the waypoints, with Figure 48b and Figure 48c showing the battery profiles of the three drones for a different selection of the parameter. The battery profiles illustrate that the charging station is never occupied by multiple drones simultaneously, since drones are waiting for another to finish. To minimize the total execution time, the blue drone charges twice shortly.



(a) Waypoints for three drones



(b) =0.1

(c) =1

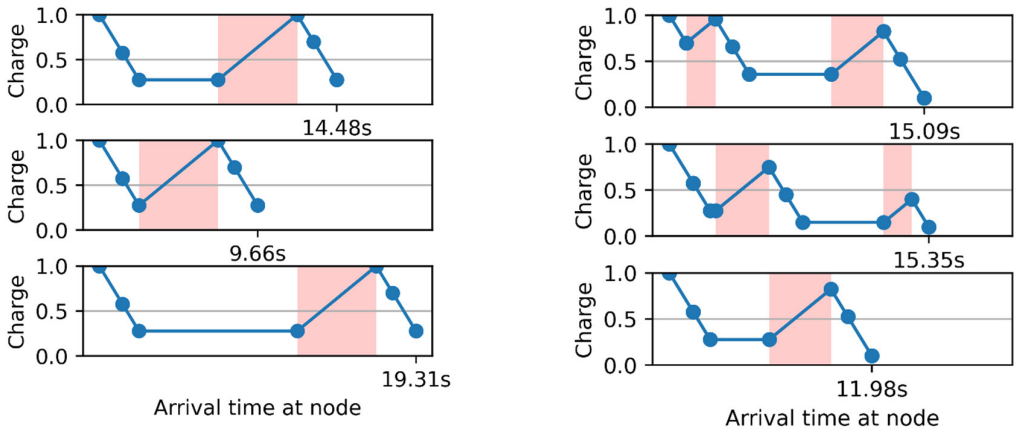
Figure 48: Example waypoints and solution for a multi-UAV model. (b) and (c) illustrate the impact of which causes the charging windows to be 'pushed away' from each other at larger values.

We compare our approach with a naive solution to illustrate our model results in a shorter execution time. In this naive approach, we assume each drone to be self-centered and short-sighted. For a fair comparison, the naive approach must decide on the same decision variables (i.e., path of waypoints and charging stations, wait times at charging stations and charging time at charging station) and is subject to the same constraints (i.e., the battery cannot deplete, and each charging station cannot serve more than one drone simultaneously). The naive approach is implemented under the following rules:

1. For each waypoint, a drone evaluates whether *it can reach any charging station if it directly moves to the next waypoint*:
    a. If it can, it will move directly to the next waypoint
    b. If not, it will move to the closest charging station

2. If the charging station is occupied, a drone will wait until it is available
3. A UAV will always fully charge its battery

We apply both the naive approach and our MILP model to the same use case showing the resulting battery profiles from both schedules in Figure 49.



(a) Naive approach

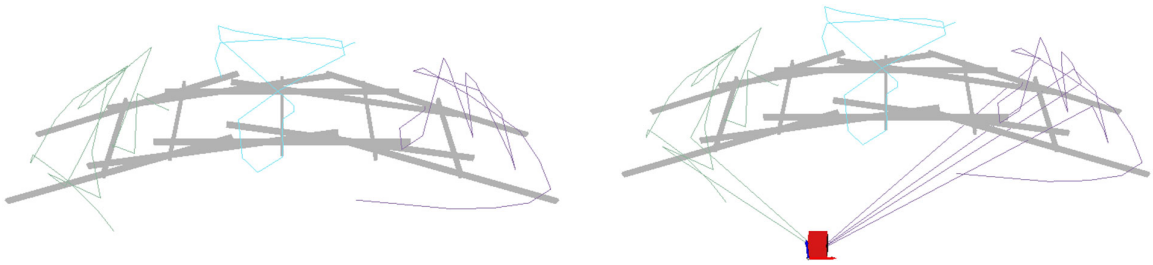
(c) MILP

Figure 49: Charging schedule for (a) the naive approach and (b) our proposed MILP solution.

In the naive approach, the third drone must wait for the other two drones to completely complete their charging. This problem is circumvented by the MILP schedule, which lets the first and second drone charge twice. The mission execution time on both schedules is 19.31 and 15.35 seconds respectively, making the MILP mission nearly 4 seconds faster.

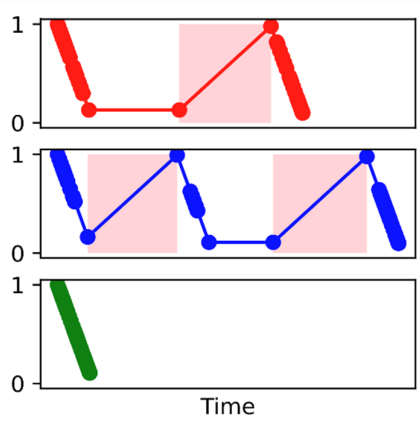
5.3.2.4 Integration with path planning

Our proposed model can naturally be integrated with the results from Section 5.2. After computing the waypoints to visit for a given piece of infrastructure, our approach can be applied to prevent any battery depletion. To illustrate this integration, we applied our approach to an example bridge (Leonardo bridge). Figure 50 shows the results, with the original computed paths in Figure 50a, the paths including visits to the charging station in 4b and the resulting battery profile in Figure 50. Note that the color scheme between Figure 50c and the other figures does not match. The figures demonstrate that the drones visit the charging station one, two and zero times, respectively.



(a) Generated waypoints for three drones inspecting the bridge

(b) The flight path including charging



(c) Battery charge over time

Figure 50: Integration of proposed charging protocol with path planning.

The charging protocol is supported by a few assumptions about the environment in which the inspection mission is taking place. The model relies on a simple relationship between the distance between waypoints and charging stations

$$energy - consumption = distance \times velocity \times depletion - rate,$$

which is unlikely to accurately represent a real-world scenario, where external factors such as weather conditions and obstacles impact the energy consumption. Secondly, solving an optimization problem that coordinates all drones in a swarm is done centrally once prior to a mission execution, and does not consider these external factors that play a role in real environments. Combined, this indicates that the proposed solution does not guarantee that a charging station is occupied by two drones simultaneously when deployed. Implementing our proposed solution requires adapting to changes during the mission execution.

A second limitation of our method is the computation cost involved in solving the model. The mathematical model suggests that the number of variables and constraints grow quadratically with the number of drones and waypoints in our model. A reasonable problem with multiple drones and tens of waypoints leads to an optimization problem of hundreds of thousands of variables and constraints which makes it expensive to solve.

One option to circumvent this limitation is to simplify the problem, by for example sample the number of waypoints or to solve the problem for only a subset of the waypoints. Alternatively, approximation methods such as genetic algorithms or particle swarm optimization can be employed. However, these

In this work, we propose a mixed integer linear programing (MILP) model that finds an optimal charging schedule for several drones that share a limited number of charging stations. It minimizes the total mission execution time, while ensuring that the batteries of the drones do not fully deplete and that each charging station does not serve more than one drone simultaneously. Our proposed scheduling model can be naturally integrated with path planning methods proposed elsewhere in this deliverable and we show that it outperforms compared to a naive scheduling strategy. We aim to alleviate the limitations of our approach in future work, by reducing the complexity of the problem and resorting to approximation techniques.

5.4 Formation Flying

Within the field of autonomous multi-robot systems, several methods have been studied on how robots can organize and adhere to formations. Formation control schemes can typically be divided into centralized or distributed control methods [12]. The following subsections introduces design and verification of a distributed and a centralized control method, respectively. First, we test a policy-based mechanism based on the boid model. Following this we present a centralized formation control mechanism based on the leader-follower scheme. This latter formation algorithm is also demonstrated as part of Deliverable D5.3 of the Drones4Safety project [19].

5.4.1 Policy-based formation control with the boid model

This study investigates a high-level control method that works in a distributed manner. The proposed method combines the boid model and path planning algorithms for the use of autonomous swarms of drones. The simulation tool has been created so that configuration space can be generated for a given map for the use of the path planning algorithms. To provide a guidance for the swarm we study an integration and a comparison of the A* and RRT* path finding algorithms. For verification, a simulator has been created to test the proposed control method by evaluating it in its time performance and the number of collisions. It is found that the proposed method can be successfully used as a high-level control algorithm for autonomous drones, which can be used to perform collision-free swarm control.

5.4.1.1 The Boid model and its adaptation

Bio-inspired methods belong to the policy-based as these typically are based on simple rules and may exhibit emergent properties such as swarm reconfigurability. Like migratory birds, small fish and insects, a swarm is generally defined as a group of behaving entities that together coordinate to produce a significant or desired result [20][21]. The attractiveness of these bio-inspired methods rests on the possibility to distribute the control mechanism onto many drones. In a distributed control algorithm, the computation is done in each drone independently from each other using localized data.

A bird swarm model, called the boid model, is used to make the drones behave as a swarm and to avoid collision with each other. The use of the boid model implements the control algorithm in a distributed manner, where the computation of the motion is done individually by the drones. The boid model is a bioinspired model that mimics the movements of a flock of birds or a school of fish. The model is founded on three distinct rules: separation, cohesion, and alignment (Figure 51).

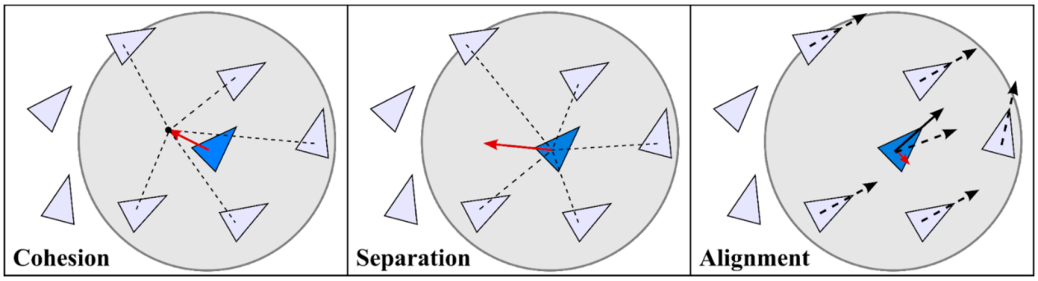


Figure 51: Rules of the boid model. Cohesion ensures boids to be attracted towards the "center of mass'" of other boids within the view distance (grey shaded circle). Separation counters the collision of a boid with its neighbors. Alignment drives boids to move in a common direction.

Each boid, which represents a drone, is an object that exists in a geometric 2D plane. It has a view distance, which is analogous to the distance in which a bird can observe other birds in the swarm. Analogously, the view distance of a drone corresponds to the sensor range.

A cohesion rule keeps the boids together. It is the rule that makes the boid move towards the center of other boids within the view distance denoted by d_{vd}. This calculation excludes the boids that are out of the visual distance. This means practically, if a boid gets far apart from the swarm, they are not able to get back together unless they get back within the visibility range. The cohesion rule may be formulated as follows:

$$a_{co} = \frac{x_{ctr} - x_0}{||x_{ctr} - x_0||}, \qquad x_{ctr} = \frac{1}{n}\sum_{i \in \mathcal{V}} x_i$$

where ||...|| is the Euclidean distance and V is the set of boids in the visibility range of the boid of interest positioned in x_0 with n=|V| being the number of boids in the set.

The separation rule is ensuring that collisions between the boids are avoided. As two boids get close to each other, after some threshold they start moving in the opposite direction. The separation rule may be formulated as follows:

$$a_{se} = -\sum_{i \in \mathcal{V}} \frac{x_i - x_0}{||x_i - x_0||}, \qquad d_{min} \leq x_i \leq d_{vd} \ .$$

We have introduced a minimum distance d_{min} that ensures boids not to avoid colliding. The distance of the visibility range is denoted d_{vd}.

The alignment rule controls the direction of the motion of the swarm as a whole. A boid looks at the other boids within its visibility range and finds the average direction and speed of the swarm. After the average velocity is found, the boid of interest aligns itself and adjusts its velocity to fit the swarm. The alignment rule may be formulated as follows:

$$a_{al} = \frac{\dot{x}_{avg}}{||\dot{x}_{avg} - \dot{x}_0||}, \qquad \dot{x}_{avg} = \frac{1}{n}\sum_{i \in \mathcal{V}} \dot{x}_i,$$

where $\dot{x}$ is the velocity of a boid, $\dot{x}_{avg}$ being the time average.

The simple boid model is extended with two additional rules to provide object avoidance and integrate with a path planning component. The path guidance rule takes waypoints as input from the path planning algorithm.

To avoid obstacles, we introduce a rule analogous to the separation rule based on observed obstacles within the view distance.

$$a_{oa} = -\frac{x_{oa} - x_0}{||x_{oa} - x_0||}$$

where $x_{oa} >= 0$ is the distance to a nearby object. Furthermore, the rule that guides the swarm along a path is formulated as:

$$a_{gd} = \frac{x_{gd} - x_0}{||x_{gd} - x_0||}$$

where x_{gd} is a vector providing guidance towards the next way point.

All the rules mentioned act as physical forces on the swarm in their retrospective direction weighted by a set of weight factors.

5.4.1.2 Path planning

A path planning component creates a sequence of waypoints guiding the motion of the drones. Several path planning algorithms have been reported in the literature [23]. Among these the A* and the RRT* algorithms are most often encountered with UAV path planning. The A* path-search algorithm uses a heuristic function $h(n)$ to minimize the cost of reaching the goal y_g of the path from any other node. We use the Euclidean distance between the current node and the goal i.e., the goal in our work: $h(n) = |y_g - y_n|$. In contrast, the Rapidly-exploring Random Tree Star, denoted as RRT*, is an algorithm designed to search in an area by covering it with a space filling tree. RRT is a single-query planner that sample points in the configuration map until a path is found although this may not be the optimal path. The RRT algorithm is executed until a path is found. An improved version of the RRT algorithm is the RRT* algorithm.

5.4.1.3 Simulator implementation

To study the performance of the swarm control based on the boid model, a continuous time simulation was built. The boid model and the path planning algorithms are implemented into the drone objects to test the proposed control method. The simulator controls the path planning and the boid modules independently.

To navigate the environment, drones are equipped with a digital map defining the workspace for the drone. We create the workspace from orthophotos of the inspection area of interest corrected by using an elevation model to adjust actual heights to the 2D plane.

To transform a workspace into a configuration space, a map is first digitized to mark each pixel if it contains an obstacle (coded with “1”) or is free (coded with “0”). A map is further enhanced by creating an artificial buffer enlarging the area of obstacles. The artificial buffer provides a certainty that the paths generated will not get too close to obstacles.

For simplicity we have assumed that a drone has a physical size smaller than the size of a single pixel and therefore can be represented by a point. The integration has been done by splitting the path planning computation for the map in the prepossessing phase and feeding in waypoints into the drones. For the travel between the waypoints, the object avoidance features of the boid model have been used. Hence, the waypoints guide the swarm in a global scale across the map, while sensor data is used to navigate around obstacles. This allows individual drones to not be burdened by the real-time computation of path planning, but rather to follow the global path while reacting to their environments.

The next step is to create the simulation medium. A discrete continuous time simulator is created to iterate the motion of the drones in the environment. The simulator stores the metrics of number of iterations, path lengths, number of collisions and the number of drones that reach the goal. The option of viewing the simulation is added in this step so that the simulated flight of the swarm can be observed (Figure 52).

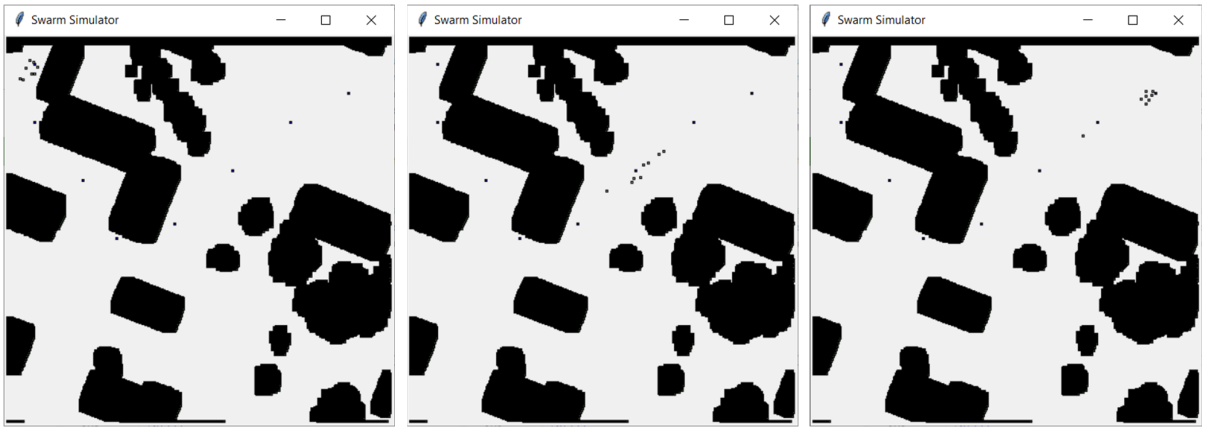
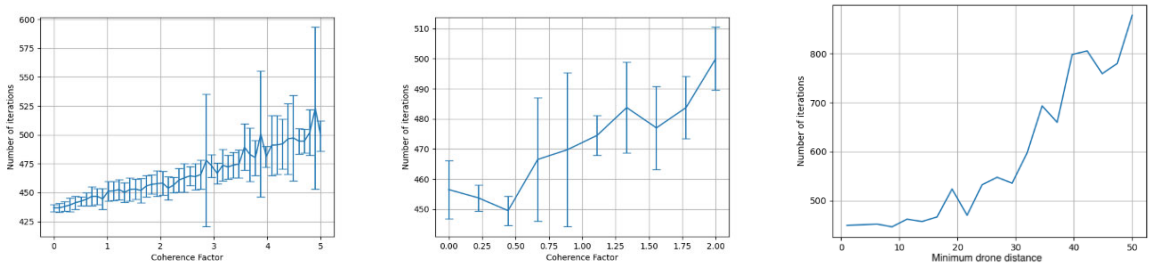


Figure 52: Display of the simulator with boids and waypoints marked as pots. The digitized map of the flying area is based on actual building and vegetation adding a buffer to keep boids at a safe distance to the obstacles.

5.4.1.4 Experiments and results

The first set of experiments exploits the relationship between the coherence factor and the time it takes for the UAV swarm to reach the goal.



(a) No obstacles (b) Convex obstacles (c) Minimum distance

Figure 53: Experiments with time dependence and cohesion and minimum distance. Each data point in the graph represents the mean of 20 simulation runs.

Figure 53a shows the relationship between the cohesion factor and the time it takes to reach the goal measured in the number of iterations (epochs). The experiment is done 20 times for each parameter. Each run is counted as complete when all the drones reach within the goal node. The time it takes to reach the goal increases with the increase of the cohesion factor. Drones that are on the right trajectory pay a lot for the drones that are left behind. On the other hand, even though the main swarm is slowed down, single escaped drones tend to recover from the departure when the coherence factor is high.

This graph alone suggests keeping the cohesion factor to a minimum for best time performance, but it is not completely true since this will lead the swarm to fly completely separately. In the cases where the cohesion factor is approaching 0, the only factor other than the goal motivation factor is the obstacle and drone avoidance

factors. Since there is no force to keep the drone back together, the drone avoidance factor shows its effect and disbands the swarm in all directions, showing the significance of the coherence factor in the unity of the swarm.

Figure 53b explores the next step is to check the effect of the cohesion factor on obstacle environments. For this a digitized map portion has been selected that contains convex obstacles. Convex obstacles mean that in the path between the start goal and the destination goal, there are no concave "cave-like" shapes that can trap the swarm.

Such geometry has been chosen for this test as the aim of this experiment is not to see the effectiveness of the swarm to avoid concave holes, but rather the time difference of the swarm with changing cohesion factor. Since the obstacles are difficult to get around, the A* path planning algorithm is added to allow the swarm to go through the obstacles.

The minimum distance parameter, which is the parameter which designates the minimum distance allowed between the drone, has a similar effect as the coherence factor. The minimum distance parameter is similar to the coherence factor in the sense that it makes the swarm less flexible when the minimum distance is increased, as seen in an example shown in Figure 53c.

On the other hand, the overall surface area that is covered by the swarm is significantly increased. This is an important fact to consider, as it may be useful in specific situations such as inspection for large areas, such as farms.

Reaching the Goal: A shortcoming of the boid model is that drones are not guaranteed to reach the destination goal. This is due to the forces that act on the drones may counter the forward motion in the path. One of the main contributing factors of drones to miss reaching the goal is the cohesion factor. Experimentation shows that increasing the cohesion factor reduces the number of drones that reach the goal. This effect can be seen in Figure 54a:

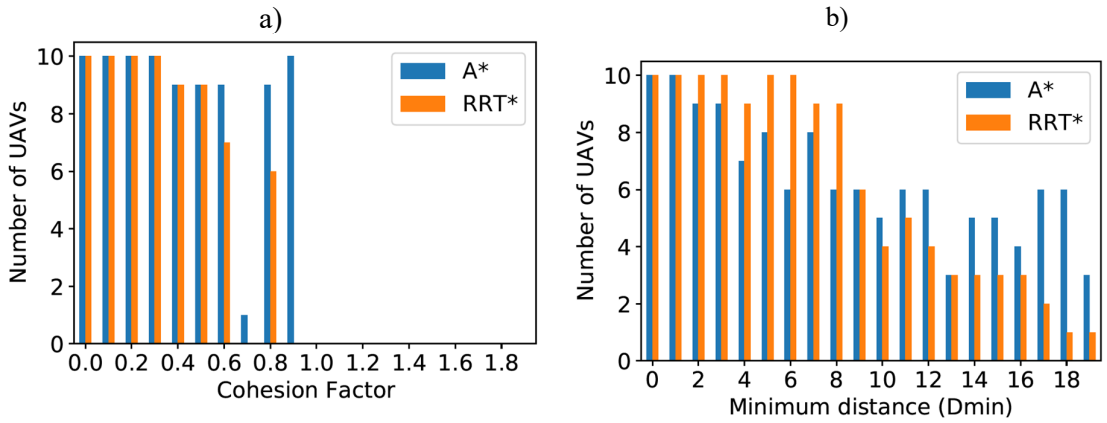


Figure 54: The number of drones that reach the goal for the path created by A and RRT* algorithms.*

An important consideration is that after the cohesion factor obtains the value of 1, the swarm becomes immobile as this factor out-weights all the other forces driving the motion of the swarm.

The second most influential parameter on the number of drones reaching the goal is the minimum drone distance parameter. This parameter reduces the density of the swarm as the parameter is increased. A high minimum distance means that the drones cannot get close to each other, making the overall size of the swarm larger. This makes it difficult for the swarm to pass through narrow free space.

Figure 54b shows the number of drones that reached the goal for going around a building in the map. It can be seen in the figure above that the number of drones that reach the goal node significantly reduces as the minimum drone distance increases. It is important to note that this decrease is caused due to the tight passage and that this effect is not expected in an open field.

In this section, we report on the design and evaluation of a high-level swarm control algorithm based on the boid model. The method is distributed on individual drones to make decisions based on local sensory input. Our proposed control method extends the boid model with path planning and obstacle avoidance. As a result, it uses five simple rules to both avoid all obstacles and follow a path. The control method is tested in a continuous time simulator that has been created for this research. The combination of the path planning as an input to the boid model proved to be successful, as all the experimentation done uses this algorithm. It has been concluded that the created algorithm is a promising approach for controlling autonomous swarms. A shortcoming of the proposed control method in its current form is that it does not handle concave obstacles in a robust manner.

5.4.2 Leader follower scheme

The formation flying function is designed using a leader-follower scheme. Specifically, the leader drone receives the mission description (a mission file) and mission control commands (service calls) from the ground control station. A mission file consists of a list of waypoints for both the leader drone and follower drones. These waypoints are pre-generated according to the mission. Then, the leader drone coordinates mission progress with the follower drone by controlling the distribution of the mission element, i.e., waypoints. At last of each iteration, the waypoint is executed by each drone using onboard position control. Figure 55 shows the communication protocol to achieve a formation flying.

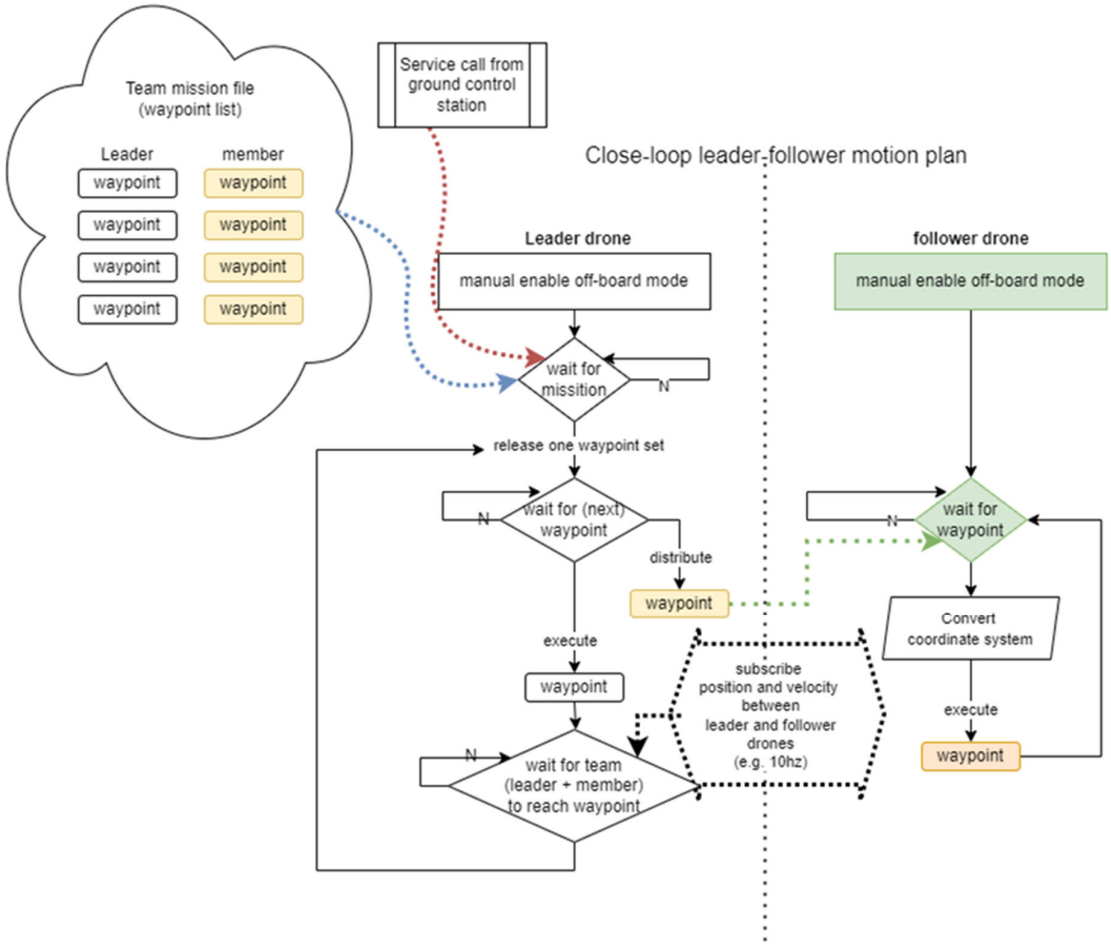


Figure 55: Communication protocol for formation flying.

The formation flying function is designed using a leader-follower scheme. Specifically, the leader drone receives the mission description (a mission file) and mission control commands (service calls) from the ground control station. A mission file consists of a list of waypoints for both the leader drone and follower drones. These waypoints are pre-generated according to the mission. Then, the leader drone coordinates mission progress with the follower drone by controlling the distribution of the mission element, i.e., waypoints. At last of each iteration, the waypoint is executed by each drone using onboard position control.

Formation flying requires communication between drones. The communication system is established based on a WiFi network with an access point. The following figure shows that the network connects drones and a ground control station (Figure 56).

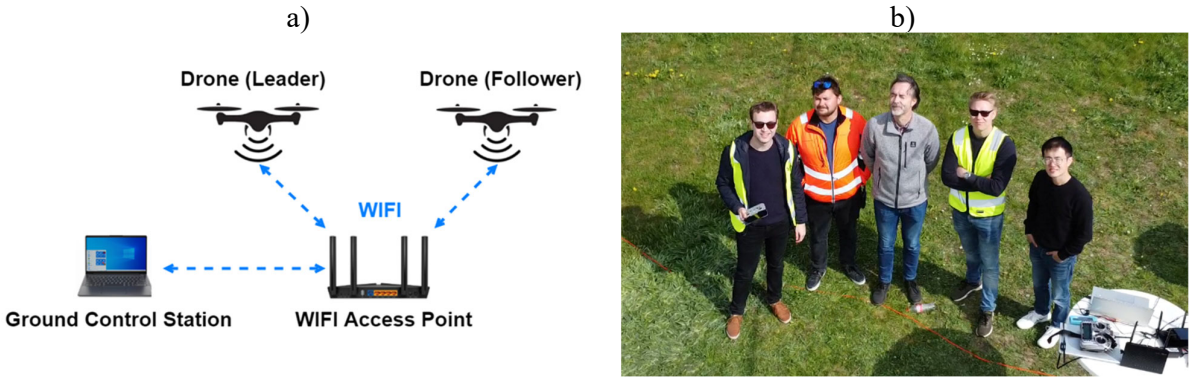


Figure 56: a) Communication setup for formation flying, b) Photo of team effort for the validation of the formation flying at the HC Andersen Airport.

A demonstration of formation flying with two drones in the outdoor test environment is given in deliverable D5.3 [19]. The picture in Figure 5 on page 13 show two drones flying in formation.

6 Towards BVLOS Swarm Operation

According to current EU regulation Beyond Visual Line of Sight (BVLOS) operations are formally allowed within the Specific/Certified category (according to the level of risk). The main operative constraints are related to the time and effort required for authorizations and UAS/ pilots/operator qualifications.

The new EASA drone Regulations aims to standardize the different regulations of the Member States and to regulate the civil use of drones regardless of their size or weight. The new European regulatory framework applies to all UASs, whether autonomous or remotely piloted, and regardless of their mass or use. Current regulations in the open drone category demands to always keep the UAS in the line of sight. The 'First Person View' and 'Follow-me' flight modes can be considered under certain conditions as VLOS [31]. However, it is expected that a deregulation will gradually occur and BVLOS operation will be practically possible in the future starting with single drone operation and later advancing to swarm operation under in controlled settings.

This section documents steps taken as part of effort in WP5 to support BVLOS operation in the future. In this regard, focus has been on data support and technology evaluations.

6.1 Information service for BVLOS operation

To support the BVLOS operation of the autonomous drone swarm, we developed a cloud platform for mission management where information about the global flight environment is stored as well as information collected during the flight. Drones exchange information with cloud services through HTTPS protocol and services are connected to databases. Databases store information about infrastructure locations, restricted areas, planned and executed missions, telemetry received from the drones, and images collected during the missions. Restricted areas are stored within the No-fly service in JSON format, visualized on the web interface, and can be transmitted to the swarm upon request. Infrastructure locations are stored in databases corresponding to Towers, Bridges, and Railways services. The data is used for building a graph and calculating inspection paths and transmitted to the drones in the form of a mission plan. Detailed mission plans are stored in a separate database, providing routes and specific tasks to the drones. Drones report their status during the mission and data is stored in the Drone Log database. Images received from the inspection are stored in Object Storage. The following subsections describe database development and organization in detail.

6.1.1 Railways, Towers, and Bridges Database

MongoDB is a source-available cross-platform document-oriented database program. Classified as a NoSQL database program, MongoDB uses JSON-like documents with optional schemas. We use it to store infrastructure data in JSON-like documents which are easily queried. MongoDB supports geo-queries when the database contains geographic locations. The user can query elements within a specified distance or range, which is very useful in our case for determining which nodes in the graph are neighboring nodes. Services that use MongoDB are Towers, Railways, and Bridges. The data is represented with models shown in Table 2. Data models determine a form the data is stored in the database. The database contains information about power towers, lines, bridges, and railway's locations and specifications. Towers are defined as "nodes" and the data model contains location. Lines are defined as "ways" and contain an array with unique identifiers for nodes. Nodes' location is stored with the same data model used for towers. Bridges' locations are stored as ways containing an array of nodes forming the polygon around the bridge. Each node contains geolocation with latitude and longitude. Railways are stored using the same data model used for lines.

Table 2: Data models as stored in the database

Towers	Lines	Bridges	Railways
_id: ObjectId type: "node" id: tower_id location: Object	_id: ObjectId type: "way" id: line_id nodes: Array tags: Object	_id: ObjectId type: "way" id: bridge_id nodes: Array tags: Object	_id: ObjectId type: "way" id: railway_id nodes: Array tags: Object
	_id: ObjectId type: "node" id: node_id location: Object	_id: ObjectId type: "node" id: node_id lat: latitude lon: longitude	_id: ObjectId type: "node" id: node_id location: Object

6.1.2 Missions and Drone Log Database

PostgreSQL, also known as Postgres, is a free and open-source relational database management system emphasizing extensibility and SQL compliance. In Missions and Drone Log services, we created relations using PostgreSQL because this database will experience user queries and complex join queries for which NoSQL databases are not well adapted.

Data needs to be transferred between services in the cloud, to the drone, and from the drone to the cloud. Database relations specify mission data (including inspection tasks), telemetry data, and inspection results data. Therefore, we designed databases in Missions and Drone Log services to create global data space. Figure 57 shows relations implemented in these services and relationships between them. Relations contain attributes described as:

- **Mission relation** – the relation contains mission data, like mission name, the status of the mission, swarm participating in the mission, routes associated with it, geofence regions where the flight is permitted, and time information.
- **Task relation** – the relation specifies tasks given to the drone. It describes the inspection type, location of task execution, and the drone the task has been delegated. It stores specifications describing which sensors to use e.g., RGB camera, frequency of data acquisition, speed of the drone during data acquisition, etc.

- **Result relation** – the relation stores inspection results and related data. It contains time data when the results are received, the location where the result was obtained, the fault description, and the image UID associated with the faulty image.
- **Telemetry relation** – the relation stores drone telemetry data like position, orientation, velocity, speed from different sensors. It also stores the drone's status, battery status, and onboard computer status.
- **Swarm relation** – the relation stores swarm identifiers for swarms participating in the mission.
- **Drone relation** – the relation stores drones participating in the mission.

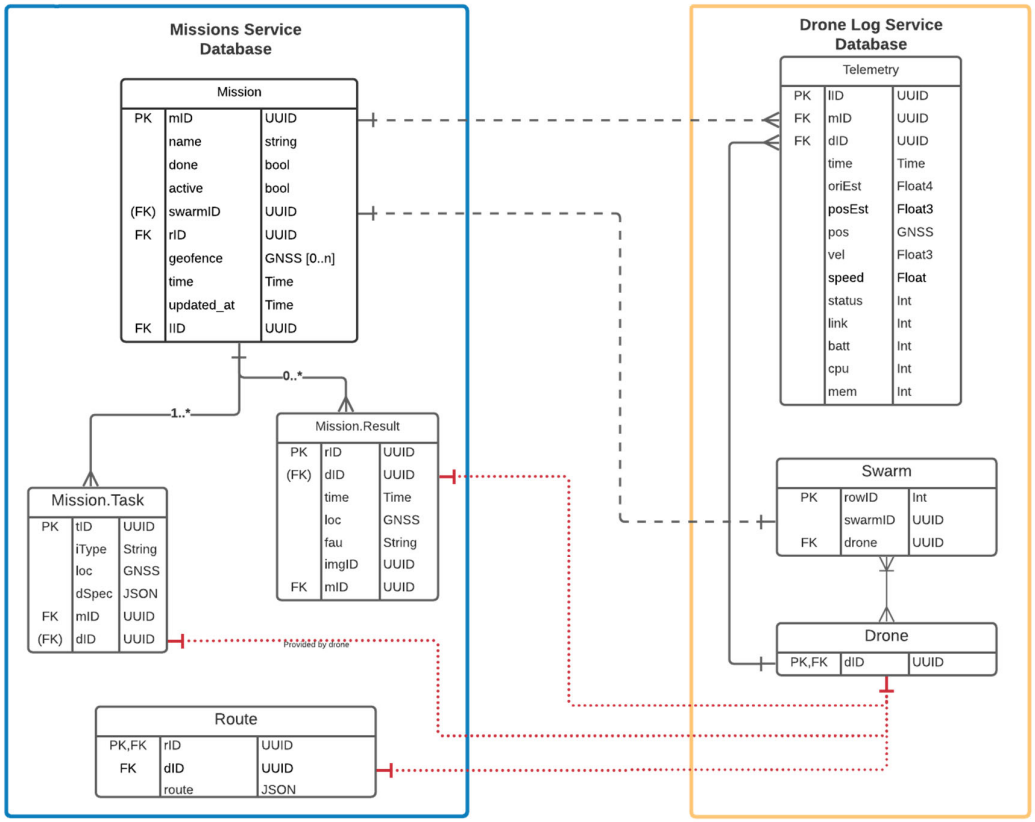


Figure 57: Database diagram.

6.1.3 Object Storage Database

The Object Storage database is a MongoDB database storing images in a binary format. The database stores images received from the drones with corresponding metadata. MongoDB enables geo-querying which facilitates querying based on locations where the images were taken. For the proof-of-concept application, we use cloud storage offered by MongoDB similarly to databases storing power towers, power lines, bridges, and railways. The database contains a collection named "Image" storing useful data related to each image received. The data model is shown in Table 3. The data model determines the form in which each image is stored in the database. Object ID is a unique identifier automatically set by MongoDB when inputting a new entry. Image ID is an image unique identifier enabling search for a specific image. Mission ID stores the unique identifier of the mission during which the image was captured. Drone ID stores the unique identifier of the drone which captured the image. By storing the latitude and longitude of the location where the image was captured we

enabled geo-querying and search for the images taken near the location chosen on the map. This design enables the user to search for images collected in previous autonomous missions either by the unique identifiers or by geolocation.

Table 3: Image data as stored in the Object Storage.

Image
_id: ObjectId img_id: int mission_id: int drone_id: int lat: latitude lon: longitude metadata: Object img: Binary Object

7 Conclusions

This deliverable summarizes test and validation activities of WP5. Activities address the communication needed to support the multi-drone system as well as the key function, algorithms, and protocols to offer the needed autonomy and control of an inspection mission. We have presented results from using LoRa radio communication for drone-to-ground communication, drone-to-drone communication using ROS over WiFi and we have characterized and assessed 5G mobile communication as a potential technology for the drone-to-cloud communication.

Test and validation of algorithms demonstrates the working of key methods needed in the autonomous inspection mission including global motion path planning, inspection path planning, policy-based motion control, secure group management and a protocol for drone charging.

8 References

[1] Drones4Safety, “D5.1 - Specification of the Multi-Drone Swarm System”, 31.03.2021.

[2] Bisschop, J. (1993). AIMMS - Optimization Modeling.

[3] Hart, William E., Jean-Paul Watson, and David L. Woodruff. "Pyomo: modeling and solving mathematical programs in Python." Mathematical Programming Computation 3(3) (2011): 219-260.

[4] Gurobi Optimization, LLC. (2022). Gurobi Optimizer Reference Manual. https://www.gurobi.com

[5] Samonas, S., & Coss, D. (2014). The CIA strikes back: Redefining confidentiality, integrity and availability in security. Journal of Information System Security, 10(3).

[6] Smirnoff, P. (2017, September 13). Understanding Hardware Security Modules (HSMs). Cryptomathic. Retrieved May 16, 2022, from https://www.cryptomathic.com/news-events/blog/understanding-hardware-security-modules-hsms

[7] Farahani, S. (2011). ZigBee wireless networks and transceivers. Newnes.

[8] Welcome to PyCryptodome's documentation — PyCryptodome 3.14.1 documentation. Retrieved May 17, 2022, from https://pycryptodome.readthedocs.io/en/latest/index.html

[9] RFC 8017 - PKCS #1: RSA Cryptography Specifications Version 2.2. (2016, November 2). IETF Tools. Retrieved May 17, 2022, from https://datatracker.ietf.org/doc/html/rfc8017

[10] Kim, Y., Perrig, A., & Tsudik, G. (2000, November). Simple and fault-tolerant key agreement for dynamic collaborative groups. In Proceedings of the 7th ACM Conference on Computer and Communications Security (pp. 235-244).

[11] D5.2: Multi-drone system threat analysis and specification of the security system design, Drones4Safety project, 2021

[12] Ouyang, Q., Wu, Z., Cong, Y., & Wang, Z. (2022). Formation control of unmanned aerial vehicle swarms: A comprehensive review. Asian Journal of Control.

[13] Zhou, B., Gao, F., Pan, J., & Shen, S. (2019). Robust Real-time UAV Replanning Using Guided Gradient-based Optimization and Topological Paths. Proceedings - IEEE International Conference on Robotics and Automation, 1208–1214.

[14] Semtech, "Semtech sx128x long range datasheet," 2019.
Url: https://semtech.my.salesforce.com/sfc/p/#E0000000JelG/a/2R000000HVET/HfcgiChyabtiPTh6EjcDM6ZEwAOQV7IirEmRULgggMM (accessed on 2 May 2022).

[15] Matlekovic L, Schneider-Kamp P. From Monolith to Microservices: Software Architecture for Autonomous UAV Infrastructure Inspection. CLOUD2022 2022 Apr 5.

[16] Matlekovic, Lea, Filip Juric, and Peter Schneider-Kamp. "Microservices for autonomous UAV inspection with UAV simulation as a service." Simulation Modelling Practice and Theory (2022)

[17] Deliverable D2.4: Use-case Document, Drones4Safety project.

[18] Gazebo Robotics simulator. URL: https://gazebosim.org/home

[19] D5.3 - Collaborative drone swarm system hardware and software, Drones4Safety project. Date: 2022-05-31.

[20] Tahir, A., Böling, J., Haghbayan, M. H., Toivonen, H. T., & Plosila, J. (2019). Swarms of unmanned aerial vehicles—a survey. Journal of Industrial Information Integration, 16, 100106.

[21] Bürkle, A., Segor, F., & Kollmann, M. (2011). Towards autonomous micro uav swarms. Journal of intelligent & robotic systems, 61(1), 339-353.

[22] PX4 User Guide - Simulation. URL: https://docs.px4.io/v1.12/en/simulation/

[23] Elbanhawi, M., & Simic, M. (2014). Sampling-based robot motion planning: A review. IEEE access, 2, 56-77

[24] A. Goldsmith, Wireless Communications, Cambridge university Press, 2005, Chapter 2.

[25] L. Shi, N. J. H. Marcano and R. H. Jacobsen, "A Survey on Multi-unmanned Aerial Vehicle Communications for Autonomous Inspections," 2019 22nd Euromicro Conference on Digital System Design (DSD), 2019, pp. 580-587, doi: 10.1109/DSD.2019.00088.

[26] L. Shi, N. J. H. Marcano and R. H. Jacobsen, "A Survey on Multi-unmanned Aerial Vehicle Communications for Autonomous Inspections," 2019 22nd Euromicro Conference on Digital System Design (DSD), 2019, pp. 580-587, doi: 10.1109/DSD.2019.00088.

[27] B. Li, Z. Fei and Y. Zhang, "UAV Communications for 5G and Beyond: Recent Advances and Future Trends," in IEEE Internet of Things Journal, vol. 6, no. 2, pp. 2241-2263, April 2019.

[28] A. Fornes-Leal, R. Gonzalez-Usach, C. E. Palau, M. Esteve, D. Lioprasitis, A. Priovolos, G. Gardikis, S. Pantazis, S. Costicoglou, A. Perentos, E. Hadjioannou, M. Georgiades, and A. Phinikarides, “Deployment of 5G experiments on underserved areas using the open5genesis suite,” in 2021 International Conference on Smart Applications, Communications and Networking (SmartNets), 2021, pp. 1–4.

[29] S. Diamond and S. Boyd, “CVXPY: A Python-embedded modeling language for convex optimization,” Journal of Machine Learning Research, vol. 17, no. 83, pp. 1–5, 2016

[30] A. Domahidi, E. Chu, and S. Boyd, “ECOS: An SOCP solver for embedded systems,” in European Control Conference (ECC), 2013, pp. 3071–3076.

[31] European commission, COMMISSION IMPLEMENTING REGULATION (EU) 2019/947 of 24 May 2019 on the rules and procedures for the operation of unmanned aircraft, https://www.easa.europa.eu/document-library/regulations/commission-implementing-regulation-eu-2019947

Appendix A: Group Management Protocol Message specification

This appendix details the message structure for the Secure group management protocol introduced in Section 5.1.

A.1 Message structure

This section describes the structure of the messages that are exchanged during the previously described activities and the cause of state changes in Figure 29. To visualize the cryptographic keys for a particular message, we rely on the following color scheme to indicate the fields that are covered by the following keys:

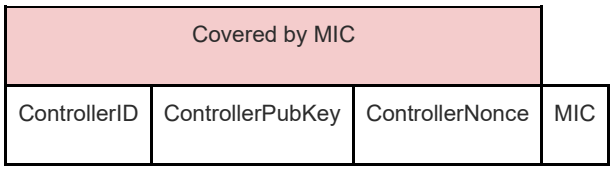
Unencrypted	UAV public key	Current group key	Former group key	Root key	Controller public key

All messages have a group key version and message type field. The former allows the receiver to know what group key to use for decryption operations (or for a member to identify it needs to synchronize with the

controller), and the message type indicates to the receiver how to interpret the remaining data in the message. Both fields are unencrypted, so that no decryption is required to read the message.

Controller broadcast

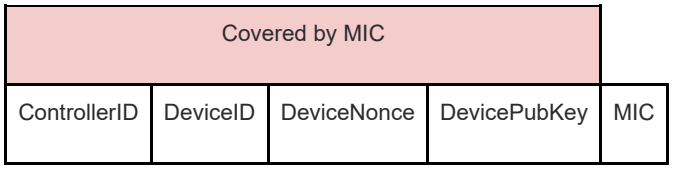
The controller broadcast message is used to advertise the unique ID of the controller and its public key. The identifier differentiates this broadcast from broadcast messages from nearby other group controllers, and the public key is used in further communication to prevent spoofing attacks on the controller. A nonce prevents replay attacks and a MIC ensures the integrity of the messages.



Note: for this message, the version field can be ignored by the receiver, as it is not part of a group yet.

Join request

A join request is sent from a non-member drone to a controller. The message includes both the identifier of the controller to join, and the identifier of the drone (or device) that wishes to become a group member. To prevent a replay attack, a nonce is added and the public key of the member is added that allows the controller to send secure messages to the drone. A MIC is computed over the full message using the root key, preventing tampering.

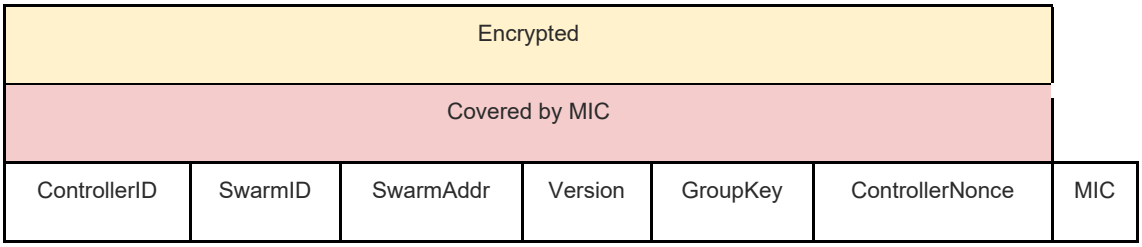


A controller receiving a join request can verify the validity of the message (1) verifying the MIC with the root key, (2) checking if the controller ID in the message matches its own identifier and (3) verifying that it has never received the nonce for the given member before.

Note: for this message, the version field can be ignored by the receiver, as it is not part of a group yet.

Join accept

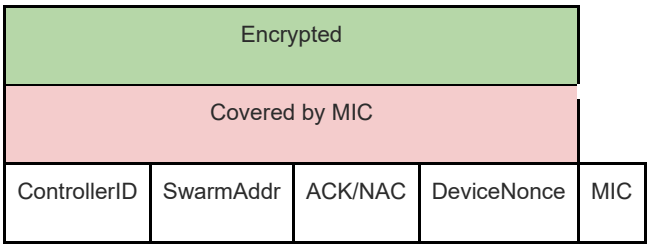
After successfully verifying a join accept, a controller responds by sending an accept message. This message contains the identifier of the controller and the swarm. The controller assigns a swarm-wide address to the member that allows the member to address other UAVs in the swarm across groups. The current group key version and the group key itself is embedded in the messages, as well as a nonce (to prevent replay attacks). A MIC computed over the message ensures integrity. The full message is encrypted using the private key of the UAV. The root key cannot be used for encryption, since this would expose the group key to non-member drones within communication range.



Note: for this message, the version field can be ignored by the receiver, as it is not part of a group yet.

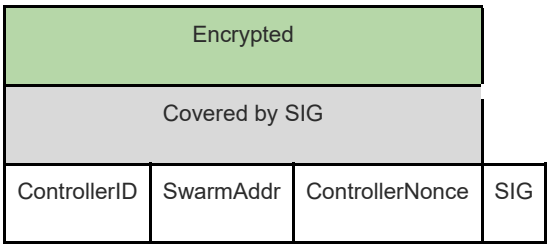
Leave request

The leave request indicates a drone wishes to leave a group. As such, it indicates the identifier of the controller whose group to leave, and the member's address within the swarm. An acknowledgement field indicates whether the drone expects an acknowledgement from the controller or not. Acknowledgement is relevant for a use case in which a drone is executing a task and the task must be re-assigned by the controller. In this case, the member needs confirmation that its task will be reassigned before leaving. The message contains a nonce against replay attacks and a MIC to guarantee integrity. This message is encrypted using the current group key.



Leave accept

When a controller receives a leave request that requires an acknowledgement, it sends a leave message. This message contains the controller's ID and the address of the member which wishes to leave. The message contains a nonce and is digitally signed by the controller using its private key to ensure only controllers can accept members leaving (and not other drones in the group).

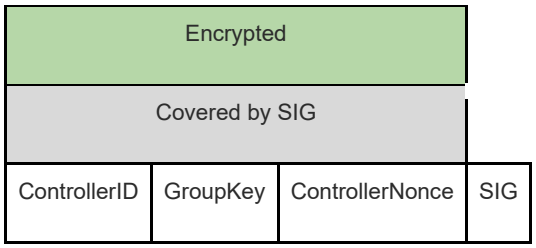


Group key update

Updates of the group key are broadcasted by the controller in case of group membership changes. This message contains the identifiers of the controller, the new key of the group and a nonce. The message is encrypted using the current group key to maintain the secrecy of the new group key within current group members and the message is signed by the controller's private key to ensure only the controller can update the group key.

An important note is that by encrypting this message with the current group key, the forward secrecy requirement is broken; a member that just left the group (and as such should not have access to more recent

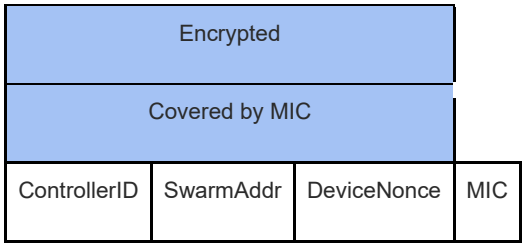
group keys) within the broadcast range of the group key update can decrypt it and obtain the new group key. This process can be repeated indefinitely, but relies on the former group member to intercept all group key updates.



Note: after broadcasting this message, the controller increments the version number of the group key in subsequent messages and so do all member dronesthat receive the update.

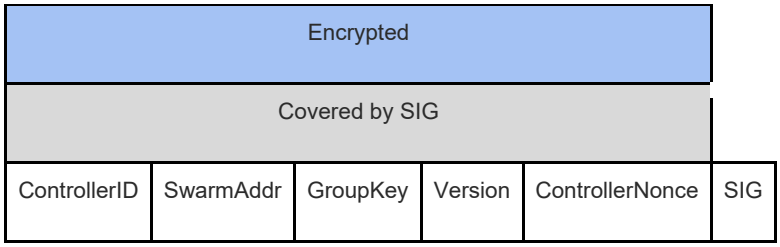
Synchronization request

A member can update to the most recent version of a group key by sending a synchronization request. This request includes the identifier of the controller and the member who sends the request. A nonce prevents replay attacks, a MIC prevents tampering and the message is encrypted for confidentiality. An older group key is used for both the MIC and the encryption.



Synchronization accept

As response to a synchronization request no more than one group key generation behind, an accept message should be sent. The message contains the identifier of the controller and the member, and the new group key. Since the member is out of sync, it will also be notified of the current version of the group key. The message is encrypted with the same (former) group key that was used to encrypt the synchronization request to provide confidentiality. Furthermore, the message is signed by the controller's public key.



A.2 Field descriptions

The fields (and their purpose) that are used in the messages are described here in more detail.

Version is a field used to indicate what iteration the swarm is on, which also indicates what iteration of the group key messages is encrypted with.

Message type is a field used to identify the type of message being sent. It is used by the protocol to determine how to handle the data in the message.

MIC is an integrity field appended to all messages to ensure that messages are sent by a member of the swarm and that the content of a message has not been altered. The MIC is created and validated by a symmetric key.

SIG is a digital signature added to the message to ensure that the message is from the controller. The digital signature is created and validated by a private and public key respecitvely.

ControllerID is a globally unique identifier that uniquely identifies an entity able to act as controller.

ControllerPubKey is the public key of the controller and is used by the drones in the swarm to verify the signature in messages from the controller

ControllerNonce is a field used by the drones to protect against replay attacks on messages from the controller

DeviceID is a globally unique device identifier which uniquely identifies a drone.

DeviceNonce is a field generated by the drones and is used to protect against replay attacks. The field is used by the controller to check that the DeviceNonce has not been used by the drone before. The value can be random or incremental depending on hardware capabilities and constraints.

DevicePubKey is the public key of the drone that wants to join the swarm and is used by the controller to encrypt the response to the join request.

SwarmID is a field identifying the swarm which the drones will belong to once the join request has been accepted.

SwarmAddr is a device address assigned to drones by the controller to identify the drones within the current swarm.

GroupKey is the group key that members of the swarm should use for encryption and decryption of messages within the swarm.

ACK/NAC is a field used to indicate if a drone needs to wait for an acknowledgement before leaving the swarm or if it can just leave immediately. The purpose of this field is to ensure better task allocation.

Appendix B: Mathematical model of charging optimization problem

The charging protocol described in Section 5 is based on the solution of a MILP problem.

B.1 Notations

Before mathematically defining our model's objective and constraints, we introduce the notation used from here on in Table 4 (constants) and Table 5 (indices). The value of the constants depends on the specific problem that the model is applied to. For instance, for a (relatively) short inspection of ten waypoints with seven drones and a single charging station, the set of constants is set to (10, 7 ,1).

Table 4: Constants used in the model, unique for each

Constant	**Description**
N_d	Number of drones
N_w	Number of waypoints per drone
N_s	Number of charging stations

Table 5: Indices of the model.

Index	**Description**
d	drone
s	charging station ($\{1, \dots, N_s\}$)
w	waypoint ($\{1, \dots, N_w\}$)
w_s	Waypoint the precedes another waypoint ($\{1, \dots, N_w\text{-}1\}$)
n	node in path ($\{1, \dots, N_s+1\}$)

Furthermore, each problem is parameterized by a number of values denoted in Table 3.

Table 6: Parameters used by the model.

Parameter	Description
D^{N}_{d,n,w_s}	distance from waypoint ws to its path node n for drone d
D^{W}_{d,n,w_s}	distance from path node n to the next waypoint ws for drone d
r^{+}_{d}	battery charging rate for drone d
r^{-}_{d}	battery depletion rate for drone d
v_d	velocity of drone d
B^{start}_{d}	starting battery charge of drone d (%)
B^{min}	minimum allowed battery charge (%)
B^{max}	maximum allowed battery charge (%)

The distance matrices D^N and D^W represent the Euclidean distances between the (x,y,z) coordinates of the waypoints and charging stations, which must therefore be provided as an input to the model beforehand.

Based on these parameters, we can compute a maximum charge duration (in seconds) for drone d as

$$C_d^{max} = \frac{B^{max} - B^{min}}{r^{+}_{d}}.$$

The model variables are shown in Table 4, where there is a distinction between *control* and *state* variables [2]. The former are independent variables where the latter are dependent on the value of the control variables.

Table 7: Decision variables of the model.

Variable	Type	Description
P_{d,n,w_s}	Control	1 - if drone d moves via path node n from waypoint ws 0 - otherwise

W_{d,w_s}		waiting duration of drone d after waypoint ws
C_{d,w_s}		charge duration of drone d after waypoint ws
$b^{arr}_{d,w}$	State	battery charge of drone d when arriving at waypoint w
b^{-}_{d,w_s}		battery charge of drone d when arriving at next path node after waypoint ws
b^{+}_{d,w_s}		battery charge of drone d when after charging at next path node after waypoint ws

B.2 Objectives and constraints

Using the definitions of tables above, we introduce the objective of the model in several steps. First, we express the time it takes to move from drone d to its next waypoint w_s as follows:

$$t_{d,w_s} = \frac{1}{v_d}\sum_n [P_{d,n,w_s}(D^N_{d,n,w_s} + D^W_{d,n,w_s})]$$

In essence, the equation divides the distance (in the summation part) by the velocity of the drone. For each drone, we can express the total execution time of the mission by summing over time taken at all preceding waypoints:

$$E_d = \sum_{w_s}[C_{d,w_s} + W_{d,w_s} + t_{d,w_s})]$$

The final objective is to minimize the total execution time across all drones:

$$\textbf{Minimise} \ \max_d E_d$$

The objective is subject to the constraints defined in this section.

$$\sum_n P_{d,n,w_s} = 1 \qquad \forall d, w_s$$

The first equation specifies that a drone d must follow precisely one path node after each waypoint w_s. The battery charge at each point of the mission execution is calculated by the following equations:

$$b^{arr}_{d,1} = B^{start}_{d} \qquad \forall d$$

$$b^{-}_{d,w_s} = b^{arr}_{d,w_s} - \frac{\bar{r}_d}{v_d}\sum_n [P_{d,n,w_s} D^{N}_{d,n,w_s}] \qquad \forall d, w_s$$

$$b^{+}_{d,w_s} = b^{-}_{d,w_s} + r^{+}_{d} C_{d,w_s} \qquad \forall d, w_s$$

$$b^{arr}_{d,w_s+1} = b^{+}_{d,w_s} - \frac{\bar{r}_d}{v_d}\sum_n [P_{d,n,w_s} D^{W}_{d,n,w_s}] \qquad \forall d, w_s$$

Given constraints 2-5, each drone can be charging (or wait) when the drone moves directly between two consecutive waypoints, without visiting a charging station. To prevent this from happening, we constraint the charging and waiting duration as follows:

$$C_{d,w_s} \leq (1 - P_{d,N_s+1,w_s}) C^{max}_{d} \qquad \forall d, w_s$$

$$W_{d,w_s} \leq (1 - P_{d,N_s+1,w_s}) C^{max}_{d} \qquad \forall d, w_s$$

We further constrain the range of the decision variables:

$$P_{d,n,w_s} \in \{0, 1\} \qquad \forall d, n, w_s$$

$$b^{arr}_{d,w} \geq B^{min} \qquad \forall d, w$$

$$b^{-}_{d,w_s} \geq B^{min} \qquad \forall d, w_s$$

$$b^{+}_{d,w_s} \leq B^{max} \qquad \forall d, w_s$$

The equations above prevent the drone batteries from depleting as well as preventing overcharging of the battery.

Simultaneous charge prevention

The previously defined equations work well for a single drone but cannot be applied to a multi-drone situation without modifications. The model does not prevent multiple drones from charging at the same charging station simultaneously. We address this by introducing a new set of constraints. The underlying idea is that each drone has a time window at which they charge after each of their waypoints, and that time windows of two drones are not allowed to overlap when the respective drones occupy the same charging station.

First, we define the start and end time of the time windows, referred to as T^s and T^e respectively:

$$T^{s}_{d,w_s} = \sum_{w_p}[C_{d,w_p} + W_{d,w_p} + t_{d,w_p}] + \qquad \forall d, w_s$$

$$\frac{1}{v_d}\sum_{n}[P_{d,n,w_s} D^{N}_{d,n,w_s}] + W_{d,w_s}$$

$$T^{e}_{d,w_s} = T^{s}_{d,w_s} + C_{d,w_s} \qquad \forall d, w_s$$

Where $w_p \in \{1, \dots, w_s - 1\}$. In Eq. 14, the first summation term sums the execution time of all previous waypoints, and the second term represents the time taken from moving from the current waypoint to the start of the charging at the next waypoint. The end of the time window adds the charging time to the start of the window.

Furthermore, we introduce a binary variable $O_{d,d',\,ws,\,ws'}$ that expresses whether drone d and d' charge at the same charging station after their respective waypoints w_s and w_s'.

$$O_{d,d',w_s,w_s'} = \sum_{s}[P_{d,w_s} P_{d',w_s'}] \qquad \forall d \neq d',\ w_s,\ w_s'$$

Note that this equation is nonlinear and in practice is converted into a series of linear equations, keeping the problem a MILP. For two windows to not overlap, the starting time of both drone d and d' is not allowed to fall inside the time window of the other. This is covered by the following constraints:

$$T^{e}_{d,w_s} \leq T^{s}_{d',w_s'} - \epsilon + M(1 + \gamma_{d,d',w_s,w_s'} - O_{d,d',w_s,w_s'}) \qquad \forall d \neq d',\ w_s,\ w_s'$$

$$T^{s}_{d',w_s'} \leq T^{s}_{d,w_s} - \epsilon + M(2 - \gamma_{d,d',w_s,w_s'} - O_{d,d',w_s,w_s'}) \qquad \forall d \neq d',\ w_s,\ w_s'$$

For a sufficiently large M. The equations above forces drone d' to start charging after d ends and it forces drone d' to start charging before d starts. The newly introduced binary variable ensures that either of the equations above must hold. The non-zero ensures that 1) two windows cannot start exactly at the same time and 2) that time windows are separated by a buffer time. Whenever two drones do not charge at the same time (i.e., the binary overlap variable O is zero), the $M(1 - O)$ terms in the equations ensure both constraints are non-binding and therefore that windows are allowed to overlap.