Master of Science Thesis in Electrical Engineering
Department of Electrical Engineering, Linköping University, 2023

Collaborative Mapping with Drone Swarms Utilizing Relative Distance Measurements

Johan Forsman and Carl Tidén

Master of Science Thesis in Electrical Engineering

Collaborative Mapping with Drone Swarms Utilizing Relative Distance Measurements

Johan Forsman and Carl Tidén

LiTH-ISY-EX--23/5541--SE

Supervisor: **Anja Hellander**
ISY, Linköpings universitet
Linus Wiik
Saab Dynamics AB

Examiner: **Gustaf Hendeby**
ISY, Linköpings universitet

Division of Automatic Control
Department of Electrical Engineering
Linköping University
SE-581 83 Linköping, Sweden

Abstract

The field of use for unmanned aerial vehicles, UAVs, has completely exploded in the last decade. Today they are used for surveillance missions and inspecting places that are difficult for people to access. To increase the efficiency and robustness in the execution of these types of missions, swarms of cooperating drones can be used. However, that places new demands on which solutions are used for positioning and navigating the agents. This thesis investigates, implements, and evaluates solutions for relative positioning and mapping with drone swarms.

Systems for estimating relative poses by fusing velocity data and pairwise distance measurements between agents using an extended Kalman filter (EKF) are investigated and presented in the report. A filter that builds upon an existing approach to estimate relative poses is developed, modified to include all pairwise distances available in the constellation, leading to up to 47 percent more accurate positioning. A multi-dimensional scaling (MDS) initialization procedure is also developed, capable of determining, with good accuracy, the initial relative poses within a swarm, assisting nearly instant convergence for the EKF. Furthermore, another EKF, using MDS coordinate estimates as input, is developed and tested.

The drones are equipped with range detectors that measure the distances to the walls in four directions. The distance data is inserted into a grid, discretizing the environment. A method to account for the uncertainty in UAV position when mapping the environment is implemented, leading to improved results. Two ways for a swarm to create a map are tested and shown to be applicable in different setups. If the drones in the swarm have a common coordinate system, the drones update the same grid and create a map. If the coordinate systems of the drones differ, the maps are created individually and merged instead. Generally, the method for collaboratively constructing a map performs better and does not require complex solutions for map merging. To merge the maps, a cost function is needed that measures how well the maps match. Three different cost functions are compared and evaluated. The mapper is evaluated for a swarm exploring the environment using both known global positions and relative pose estimates.

The precision achieved with the pre-existing positioning filter is proven to be sufficiently high to generate maps with decimeter resolution when feeding relative pose estimates to the mapping system. A higher mapping resolution is possible in the simulation environment, but requires much more computation time, and was therefore not tested.

Acknowledgments

We would like to thank our supervisor Anja Hellander for the helpful feedback and interesting discussions during the course of the work. We'd also like to thank Linus Wiik and Torbjörn Crona, our supervisor and manager at SAAB, for giving us the opportunity and the necessary equipment to finish the work at the highest possible level.

We would like to thank our examiner Gustaf Hendeby who provided great guidance in the process and, together with our supervisors, helped us keep the work to a high standard.

Lastly all the colleagues of Saab who made us feel welcome and invited us to numerous interesting discussions during the many coffee breaks, also deserve our warm and grateful gratitude.

Linköping, January 2023
Johan Forsman and Carl Tidén

Contents

Notation

Symbols

Notation	Meaning
$x_{k\|k-1}$	Predicted value of x for iteration k, given an estimate of its value in iteration $k-1$.
$\dot{x}$	Time derivative of x.
$X^{[a:b,c:d]}$	Row a to (including) b and column c to (including) d of the matrix X. Assumes that the first row and column are indexed with 1.
$\mathbf{0}_i$	Row vector of i zeros or matrix of $i \times i$ zeros, inferred from usage.
$f(x)_{\|x=x_k}$	The function f evaluated at $x = x_k$.
$HF : i$	Horizontal frame of UAV i.
A_{ij}	The entry on row i and column j of the matrix A.
$\oplus$	Map merging operator.

Abbreviations

Abbreviation	Meaning
UAV	Unmanned Aerial Vehicle
SLAM	Simultaneous Localization and Mapping
LIDAR	Light Detection and Ranging
EKF	Extended Kalman filter
UWB	Ultra-wideband
ToF	Time of Flight
HF	Horizontal Frame
MDS	Multidimensional Scaling

1 Introduction

The master's thesis is a collaboration between Linköping University and Saab Dynamics AB. This chapter introduces the problem studied and provides an outline of the thesis. The thesis will investigate how a map can be constructed with range detection sensors, and how the relative poses within a UAV swarm can be determined.

1.1 Background

The use of unmanned aerial vehicles (UAVs), also known as UAVs, has completely exploded in the last decade. Today, they are used for monitoring missions and inspections of places that are difficult for humans to access. To increase the efficiency and robustness of the implementations performing these kinds of missions, it can be of interest to use swarms of UAVs.

For UAVs to collaboratively perform missions they are in many cases dependent on access to their relative positions. One approach to obtaining this data is to use a global navigation satellite system (GNSS) such as the Global Positioning System (GPS), but such a system might malfunction in some scenarios. Another possibility is to use a motion-capturing (MoCap) system. This, however, requires expensive infrastructure. Moreover, in many scenarios, it might not be preferable or even possible to place navigational beacons in the area where the UAVs should be put to work. In other words, even though named applications are often robust and give high accuracy, they are not applicable in all environments.

One method for estimating the relative positions of UAVs in a swarm without installation of infrastructure is presented in [21], where ultra-wideband (UWB) communication is used. UWB nodes attached to the UAVs of a swarm utilize bidirectional two-way ranging (TWR) communication to estimate distances between pairs of UAVs. Sensor data from each UAV, such as translational and rotational

velocity, is also communicated to other UAVs in the swarm. Using this information, the relative pose of each UAV is estimated with an extended Kalman filter (EKF). A drawback of the approach is that not all pairwise distances between the UAVs in the swarm are used. The number of distances used in the approach is equal to the amount of UAVs.

In [29], another positioning approach for agents equipped with UWB-based ranging sensors is presented. This method incorporates all combinations of distances available within the swarm, but unfortunately, it also relies on positions provided by GNSS satellites. As mentioned earlier, in some scenarios it might be wise to not depend at all on GNSS position information. The authors acknowledge the downsides of using GNSS information by reasoning around potential accuracy drops due to signal blockage in for example urban environments.

Another relative positioning system is proposed in [11]. This one is also partially reliant on GNSS information, and the authors remark on this by mentioning that the positioning accuracy of a satellite-based system might be low and that there might also be issues with deliberate interference. A large part of the development of the method is focused on dealing with these issues, just as in [29], but the work attacks the problem in another way. Here, a few systems using different variants of multidimensional scaling (MDS) are investigated. A new system is developed that includes a method for dividing the UAV swarm into patches and uses super multidimensional scaling (SMDS) to localize agents relative to the others in each patch. The patches are then merged using a low-complexity algorithm. While involving a low-complexity method in a relative positioning system might make it sound useful in situations when dealing with limited hardware, as can be the case when running code on processors mounted on UAVs, the method has drawbacks other than its GNSS reliance. One of those is that the positioning within patches uses measured relative angles between UAVs. Information like that might not always be easily measurable. A system for determining relative yaw angles between agents in a UAV swarm without depending on direct measurements of angles is introduced in [22]. There would exist a clear problem if trying to use that system for continuous localization, however. The issue would be that it requires the UAVs in the swarm to move in a specific way to estimate their relative yaw angles. This is probably not preferable when conducting real missions with UAV swarms.

Going forward from localizing UAVs, in many applications, it might also be desirable to have the capability of constructing a map of the area where the UAVs fly. In [12], a multi-agent simultaneous localization and mapping (SLAM) system is proposed where each of the robots uses its environment readings as sensor input. All the generated individual maps are then merged using an appearance-based method that identifies overlaps, to form the complete map. Two types of agents are used, namely exploring and monitoring nodes. Exploring nodes are responsible for maintaining their maps, while monitoring nodes are the ones where the map overlaps are inspected, and maps are merged. While [12] acknowledges that there might be limitations on the hardware being run on the agents, the article does not consider going with an approach where heavier computing is performed on a central computer, such as in [32]. This method allows the use of

so-called nano UAVs since intensive tasks are carried out on a central machine. An analysis of the time consumption for the computationally intensive tasks is done both in the case where a single UAV is used, and also when testing with a pair. The analysis concludes that times are increasing when the map gets larger, both in the single-UAV test and the multi-UAV test. In the multi-UAV case, about 80 % of the time consumption is due to bundle adjustments of map landmarks.

Though [32] establishes that the approach tested is capable of merging and optimizing maps in many situations, the method is reliant on global position information fed to each agent. The proposed strategy is thus more focused on mapping than localization. In [18], a framework for collaborative SLAM is developed where CPU-demanding tasks are also outsourced to a central computer. Here, the agents are not dependent on global positioning data, and accuracy results from experiments are on par with existing state-of-the-art SLAM algorithms. Similar strategies, in the sense that they do not at all rely on global position information, are developed in [20, 28]. While [28] places a lot of focus on the limitations of agents' computational resources and a server-centered way of managing maps, [20] chooses to focus more on how to optimally reject perception outliers. Many of the previously commented methods assume each UAV is equipped with a camera as the main sensor feeding data to the SLAM algorithm. It might not always be feasible to equip UAVs with cameras, for example when dealing with small UAV platforms like Crazyflie 2.0.

Another way of conducting simultaneous localization and mapping is with the LOCUS algorithm [26]. It is based on utilizing LIDAR data to estimate the odometry of a UAV. An issue with LOCUS is that it is dependent on densely located measurements of the surroundings. It is pointed out that the LOCUS system can be adapted to different platforms, though the system is only tested on robots carrying relatively heavy sensors [26, p. 4]. The potential problem of carrying extensive equipment on nano-UAVs arises yet again.

To summarize, for efficient execution of missions using UAV swarms in unknown terrains without access to a global positioning system (GPS) or similar alternatives, the UAVs need to map their environments, as well as estimate positions on their own [23]. At the same time, limited resources and physical characteristics imply constraints on where the computational tasks can be handled, as well as which data can be collected.

1.2 Purpose and goal

With the presented background in mind, it is of great interest to examine the possibilities of further development in the fields of localization within UAV swarms and using swarms to map unknown places. Since a lot of the previous literature has focused on the relative positioning of UAVs and mapping unknown territories as separate missions, a compelling mission is to try fusing the two areas. The goal for this thesis is therefore to develop a functional system for mapping using a swarm of Crazyflie UAVs, aided by a system producing relative pose estimates. Since it is not possible to fit a rotating LIDAR on a Crazyflie UAV, a simpler unit

must be used instead, which adds to the challenge of producing accurate maps. The main goal of the thesis can be divided into smaller sub-goals. More precisely, this thesis will try to answer:

- How can UWB communication, conveying ego data and relative distance information, be used to determine relative poses within a swarm of Crazyflie UAVs?
- How can uncertainty in sensor data be considered when collaboratively constructing a map?
- How can sensor data (five-directional laser scans, IMU data, and relative position information) from multiple UAVs be fused to collaboratively construct a map?

1.3 Individual contributions

Carl reviewed the implementation of the relative positioning system described in [21], developed extensions to the existing filter, and implemented a filter using MDS measurements. Results are presented in Chapter 4.

The methods and strategies for constructing a map were investigated by Johan. This is introduced in related sections in the theory chapter, and results are presented and discussed in Chapter 5.

We developed the simulator together. All experiments were performed by both of us. We jointly formulated the problem, analyzed the results, and drew general conclusions. The sections of the paper were divided between us according to the above description.

1.4 Outline

The introduction is given here in Chapter 1. A theoretical background is given in Chapter 2, followed by a system description in Chapter 3. In Chapter 4 and Chapter 5, results concerning positioning and mapping, respectively, are presented. Chapter 6 summarizes the results together with some suggestions for future developments.

2

Theory

This chapter provides theoretical background to some of the topics covered in the thesis. The main parts covered are filtering for position estimation and occupancy grid mapping.

2.1 Rotation of a point in the three-dimensional space

In the three-dimensional Euclidean xyz space a coordinate point can be rotated around any axis. Let ϕ, θ, and ψ denote the angles with which to rotate a point around the $x-$, $y-$, and $z-$axis, respectively. The rotation matrices for each axis are given by

$$\mathcal{R}_x(\phi) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\phi) & -\sin(\phi) \\ 0 & \sin(\phi) & \cos(\phi) \end{bmatrix}, \tag{2.1}$$

$$\mathcal{R}_y(\theta) = \begin{bmatrix} \cos(\theta) & 0 & \sin(\theta) \\ 0 & 1 & 0 \\ -\sin(\theta) & 0 & \cos(\theta) \end{bmatrix}, \tag{2.2}$$

and

$$\mathcal{R}_z(\psi) = \begin{bmatrix} \cos(\psi) & -\sin(\psi) & 0 \\ \sin(\psi) & \cos(\psi) & 0 \\ 0 & 0 & 1 \end{bmatrix}. \tag{2.3}$$

These matrices are frequently used when working in three-dimensional space, and essential in many coordinate transformations.

2.2 Coordinate frames

There are three types of coordinate frames related to the thesis work, the *global frame*, the *body frame*, and the *horizontal frame* (HF). There is one single global frame common for all UAVs, and for each UAV there is an associated body frame and a HF.

The global frame is a Cartesian coordinate system fixed in the room in which a set of drones are acting. This system has the z-axis pointing in the opposite direction of gravity. An example of the global frame is depicted in Figure 2.1a.

The body frame is a Cartesian coordinate system fixed in a specific UAV, with the z-axis pointing in the upwards direction of the UAV, and the x-axis pointing in the forward direction of the UAV, at all times. The x and y directions are perpendicular to both each other and the z-axis, such that the x, y, and z directions follow the right-hand rule. Such a frame is depicted in Figure 2.1b.

The third type of coordinate system, the HF, has its origin fixed in a UAV, in the same way as the body frame. The difference is, however, that the x- and y-directions in this frame are always horizontal (and perpendicular to both the z-axis and each other, with the x, y, and z axes following the right-hand rule). The z-axis in a HF always points upwards and is parallel to the z-axis in the global frame. The x-axis points in the same direction as the projection of the x-axis of the body frame onto a horizontal plane. A HF system is depicted in Figure 2.1c.

Assume that the position $p_{cf}^i = \left[x_{cf}^i, y_{cf}^i, z_{cf}^i\right]^T$ and attitude ϕ_{cf}^i, θ_{cf}^i and ψ_{cf}^i (roll, pitch and yaw) of UAV i is expressed in the global frame. The transformation of a point p_b from the body frame of UAV i to the global frame is then given by (see Figure 2.2a for an example)

$$p_g = p_{cf}^i + \mathcal{R}(\phi_{cf}^i, \theta_{cf}^i, \psi_{cf}^i)p_b, \tag{2.4}$$

where

$$\mathcal{R}(\phi_{cf}^i, \theta_{cf}^i, \psi_{cf}^i) = \mathcal{R}_x(\phi_{cf}^i) \cdot \mathcal{R}_y(\theta_{cf}^i) \cdot \mathcal{R}_z(\psi_{cf}^i).$$

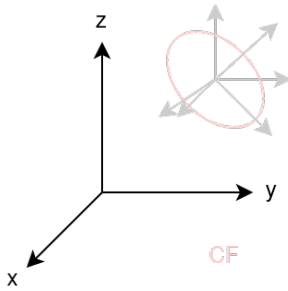
To transform a point $p_{HF:\,i}$ in the HF of UAV i into the global frame, for instance to compare relative position estimates to ground truth values, only the yaw rotation matrix of drone i, $\mathcal{R}_z(\psi_{cf}^i)$, is needed (described in (2.3)). The transformation can be performed as (see Figure 2.2b for an example)

$$p_g = p_{cf}^i + \mathcal{R}_z(\psi_{cf}^i)p_{HF:\,i} \tag{2.5}$$

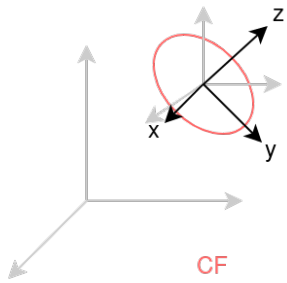
where p_g is the point expressed in the global frame.

To transform a velocity $v^T = \begin{bmatrix} v_x & v_y & v_z \end{bmatrix}$ expressed in the body frame of UAV i into a two-axis velocity $\bar{v}_{xy}^T = \begin{bmatrix} \bar{v}_x & \bar{v}_y \end{bmatrix}$ expressed in the HF of the UAV, a rotation, $\bar{v}_{xy} = (\mathcal{R}_x(\phi_{cf}^i)\mathcal{R}_y(\theta_{cf}^i))^{[1:2,1:3]}v$, can be performed [21] (see Figure 2.2c). The indexing on the multiplied rotation matrices means using the first two rows of the product. In essence, this operation of rotating and slicing indices is equivalent to projecting the velocity v onto a horizontal plane. In order to transform the yaw rate of UAV i expressed in the body frame into the HF, a calculation like

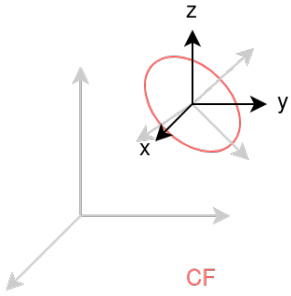
$\dot{\psi}_{HF} = -\frac{\sin(\theta^i_{cf})}{\cos(\phi^i_{cf})}\bar{p} + \frac{\cos(\theta^i_{cf})}{\cos(\phi^i_{cf})}\bar{r}$ can be done [21]. Here, $\bar{p}$ and $\bar{r}$ are the rotation velocities of UAV i in the x- and z-directions of its body frame.



(a) The global frame.



(b) Body frame coordinate system, with z-axis pointing in the upwards direction of the drone.

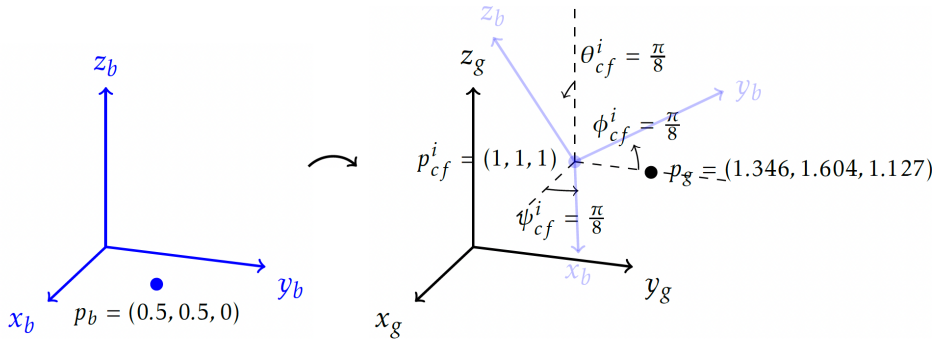


(c) Coordinate system local to a drone but with the z-axis pointing upwards. This is called a horizontal frame (HF).

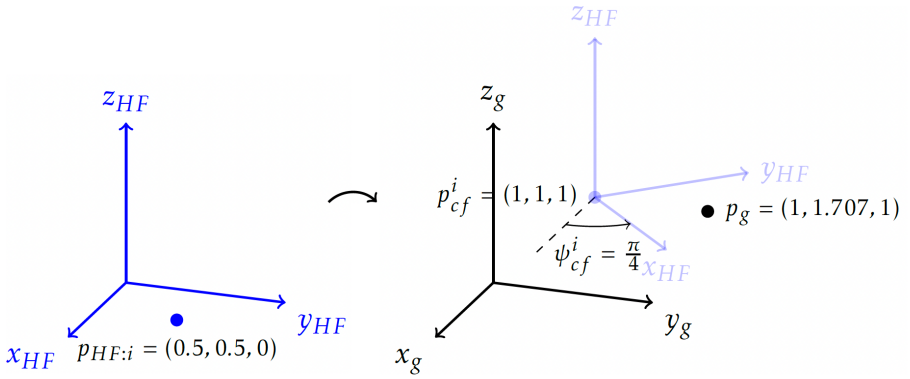
Figure 2.1: *Different coordinate systems.*

2.3 Extended Kalman filter

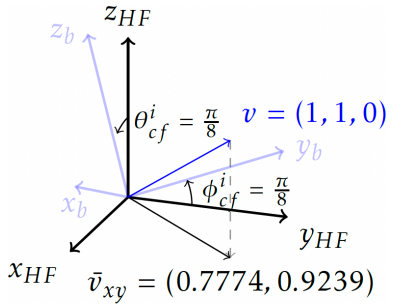
When trying to determine the unknown state of a linear system subject to process and measurement noise, a Kalman filter can be used. The Kalman filter is the best linear unbiased estimator in the sense of minimizing the mean square error of the state [14]. However, in order for the optimality to hold, some assumptions have to be made about the system and the observations. Firstly, the system model needs to be an exact match of the real system. Secondly, the process and measurement



(a) *The point p_b = (0.5 0.5 0) in the body frame is transformed onto the point p_g = (1.346 1.604 1.127) in the global frame. The pose of the UAV in the global frame is described by p^i_{cf} = (1 1 1) and $\phi^i_{cf} = \theta^i_{cf} = \psi^i_{cf} = \pi/8$.*



(b) *The point $p_{HF:i}$ = (0.5 0.5 0) in the HF is transformed onto the point p_g = (1 1.707 1) in the global frame. The pose of the UAV in the global frame is described by p^i_{cf} = (1 1 1) and $\psi^i_{cf} = \pi/8$ (only this angle is needed for the transformation).*



(c) *The velocity v = (1 1 0), expressed in the body frame, is projected onto the two-axis velocity $\bar{v}_{xy}$ = (0.7774 0.9239), expressed in the HF.*

Figure 2.2: *Different transformations.*

noise processes must be white and have zero cross-correlation with each other. Lastly, the covariances of the noise processes need to be known [19].

Consider a system described by

$$x_{k+1} = Fx_k + w_k \tag{2.6a}$$

where x_k is the state vector at time-step k, F is the state transition matrix from a time-step to the next, and w is a white noise process with known covariance matrix R. Observations, z, of the state can be made through

$$z_k = Hx_k + v_k, \tag{2.6b}$$

where H is the observation matrix and v is measurement noise. The covariance matrix for v is Q and it has zero cross-correlation with the process noise, as assumed previously. In this situation, a Kalman filter can be used to estimate the true values of x_k. The following equations are produced when deriving the Kalman filter.

$$\hat{x}_{k|k-1} = F\hat{x}_{k-1} \tag{2.7a}$$
$$P_{k|k-1} = FP_{k-1}F^T + Q \tag{2.7b}$$
$$K_k = P_{k|k-1}H^T(HP_{k|k-1}H^T + R)^{-1} \tag{2.8a}$$
$$\hat{x}_k = \hat{x}_{k|k-1} + K_k(z_k - H\hat{x}_{k|k-1}) \tag{2.8b}$$
$$P_k = (I - K_kH)P_{k|k-1}. \tag{2.8c}$$

Above, K_k is the Kalman gain, $\hat{x}_k$ is the estimated state, and P_k is the covariance matrix of the error of the state estimate at time-step k. The variables $\hat{x}_{k|k-1}$ and $P_{k|k-1}$ are predicted values for $\hat{x}$ and P based on the knowledge from the previous iteration [27].

Though the Kalman filter is optimal, in the sense previously mentioned, under the conditions mentioned earlier, there are many situations in which the model describing a particular system is not linear, and the regular Kalman filter can not be used. In these scenarios, the extended Kalman filter can be employed to provide state estimates in a very similar manner. The EKF works by linearizing the nonlinear parts of a system model using a Taylor expansion of a certain degree [17]. Suppose the system including the states of interest can be written on the form

$$x_{k+1} = f(x, u)_{|x=x_k, u=u_k} + w_k \tag{2.9a}$$
$$z_k = h(x)_{|x=x_k} + v_k. \tag{2.9b}$$

In (2.9), f is a nonlinear differentiable function of the state x, and the input u. The process noise is denoted w, and v is measurement noise with covariance matrices Q and R, respectively. The observation function h also needs to be differentiable for applying the EKF. Let F_k and H_k be Jacobians of f and h with respect

to x, evaluated at the state estimate and input values of time-step $k-1$.

$$F_k = \frac{\partial f}{\partial x}_{|x=\hat{x}_{k-1}, u=u_k}, \tag{2.10a}$$

$$H_k = \frac{\partial h}{\partial x}_{|x=\hat{x}_{k|k-1}}. \tag{2.10b}$$

The EKF then works by following the same procedure as for the standard Kalman filter, but substituting F and H in (2.7b), (2.8a) and (2.8c) with F_k and H_k. The right hand side of (2.7a) is also substituted with $f(\hat{x}_{k-1}, u_k)$, and the term $H\hat{x}_{k|k-1}$ in (2.8b) is replaced with $h(\hat{x}_{k|k-1})$.

If a state update model is given on the continuous form $\dot{x} = f(x, u)$, where f is differentiable, a first-order Taylor expansion can be used to create a discrete approximation on a similar form as the model in (2.9). If there is no noise involved, the discrete model can be expressed as

$$x_{k+1} = x_k + f(x_k, u_k)\Delta t = F(x_k, u_k). \tag{2.11}$$

In the above, Δt is the interval between the time-point $k+1$ and k. If instead, the input is noisy, the impact of the noise on the process can be approximated through another first-order Taylor expansion to generate the model

$$x_{k+1} = F(x_k, u_k) + \frac{\partial F}{\partial u}_{|u=u_k} w_{u,k}. \tag{2.12}$$

Above, u is the desired input to the system and w_u is the white Gaussian noise with which the input is disturbed. With the input noise covariance matrix denoted Q, the covariance of the resulting process noise becomes $\frac{\partial F}{\partial u}_{|u=u_k} Q \frac{\partial F}{\partial u}_{|u=u_k}^T$ [30]. The form of (2.12) is the same as in (2.9) and can be used in an EKF as presented.

2.4 Relative positioning

This section describes the general theory and methods relevant to the relative positioning systems covered in this report. It is important to note that all systems related to relative positioning assume drones that fly at the same height. This allows for omitting height coordinates when working with horizontal frames (described in Section 2.2) and viewing the problems in two dimensions.

2.4.1 Motion model for drone states

In order to predict state (relative positions and rotations) changes in the relative positioning systems covered in this report, a motion model is needed. The model takes positional and rotational velocities of individual agents, as well as current states, to determine how the states change during a short time interval. Since the positioning systems covered in the report handles poses expressed in the HF of a specific agent in the swarm, see Section 2.2, the motion model also needs to

use this sort of coordinate system. As the positioning systems are implemented in two dimensions, the motion model only needs to consider state changes in the xy-plane. Consider the scenario depicted in Figure 2.3, where the UAVs have velocities expressed in their own respective HF $v_i = \begin{bmatrix} v_{x,i} & v_{y,i} \end{bmatrix}^T$ and $v_j = \begin{bmatrix} v_{x,j} & v_{y,j} \end{bmatrix}^T$, and the yaw rates $r_i = \dot{\psi}_i$ and $r_j = \dot{\psi}_j$, also expressed in their own respective HF.

The velocity of UAV j in the HF of UAV i ($HF : i$), $v_{j,HF:i} = [\dot{x}_{ij}, \dot{y}_{ij}]^T$, where x_{ij} and y_{ij} are the x- and y-position of UAV j in $HF : i$, can be divided into a translational and a rotational contribution due the movement of both UAVs [15]. The translational contribution represents the velocity resulting from the translational movement of the UAVs. Using (2.3), and the notation in Figure 2.3, this contribution is

$$v^{(\mathrm{TR})}_{j,HF:\,i} = \mathcal{R}_z^{[1:2,1:2]}(\psi_{ij})v_j - v_i. \tag{2.13}$$

The rotation of UAV i also causes UAV j to move in $HF : i$, this is what is referred to as a rotational contribution. If UAV i rotates with the rate r_i [rad/s] around the z basis vector of its own HF, $\hat{z}_{HF:i}$, the rotational contribution for UAV j in $HF : i$, $v^{(\mathrm{ROT})}_{j,HF:\,i}$, becomes

$$v^{(\mathrm{ROT})}_{j,HF:\,i} = -(r_i \hat{z}_{HF:i} \times \begin{bmatrix} x_{ij} \\ y_{ij} \\ 0 \end{bmatrix})^{[1:2]} = r_i \begin{bmatrix} y_{ij} \\ -x_{ij} \end{bmatrix}, \tag{2.14}$$

The yaw angle of UAV j relative to UAV i is

$$\psi_{ij} = \psi_j - \psi_i, \tag{2.15}$$

where ψ_i and ψ_j are the yaw angles of UAV i and j expressed in the global frame. The relative yaw rate is calculated by taking the derivative of (2.15),

$$\dot{\psi}_{ij} = r_j - r_i. \tag{2.16}$$

All in all, a model of change in the x- and y-positions and yaw-angle of j in $HF : i$ can be summarized in the following vector

$$\begin{bmatrix} \dot{x}_{ij} \\ \dot{y}_{ij} \\ \dot{\psi}_{ij} \end{bmatrix} = \begin{bmatrix} v^{(\mathrm{TR})}_{j,HF:i} + v^{(\mathrm{ROT})}_{j,HF:i} \\ r_{j,HF:i} \end{bmatrix} = \begin{bmatrix} \cos(\psi_{ij})v_j^x - \sin(\psi_{ij})v_j^y - v_i^x + y_{ij}r_i \\ \sin(\psi_{ij})v_j^x + \cos(\psi_{ij})v_j^y - v_i^y - x_{ij}r_i \\ r_j - r_i \end{bmatrix}. \tag{2.17}$$

2.4.2 Basic relative positioning system

This section describes the positioning system from [21], which fuses velocity information from agents within a Crazyflie swarm with time-of-flight (ToF) based distance measurements provided by UWB ranging decks mounted on the agents, to provide relative position and rotation estimates. The fusion is done through an extended Kalman filter.

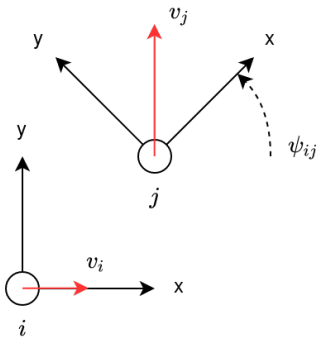


Figure 2.3: *Two UAVs i and j, their respective HF, and their translational velocities v_i and v_j.*

The mentioned ToF measurements are provided by sending messages back and forth between two drones according to the scheme presented in Figure 2.4. Naming the flight time of the messages t_f, and multiplying the apparent relationships $t_4 = 2t_f + t_3$ and $t_5 = 2t_f + t_6$,

$$t_4t_5 = (2t_f + t_3)(2t_f + t_6) \implies t_4t_5 - t_3t_6 = 2t_f(2t_f + t_3 + t_6) =$$
$$= 2t_f(t_4+t_6) = 2t_f(t_3+t_5) \implies t_f = \frac{t_4t_5 - t_3t_6}{2(t_4 + t_6)} = \frac{t_4t_5 - t_3t_6}{2(t_3 + t_5)} = \frac{t_4t_5 - t_3t_6}{t_3 + t_4 + t_5 + t_6}. \tag{2.18}$$

The final expression is the one used to estimate ToF in the positioning system. It makes up the basis for alternative double-sided two-way ranging and minimizes the negative effects on the accuracy of $\hat{t}_f$ due to clock drift. Using the expression, the timing error can be modeled as $\hat{t}_f - t_f = et_f$, where e models the deviation (typically expressed in parts per million) from nominal clock frequency [25]. The ToF messages also let agents inform each other about their ego velocities.

Only one UAV can be in transmitter mode at each time instant, and the switching of roles is also handled with a specific scheme, shown in Figure 2.5. The switching scheme is important because the information delays introduced from having agents wait to share and receive velocity and distance data can have an effect on the performance of the localization. The positioning work of this report, however, is more focused on filtering methodologies and does not go into detail about hardware delay issues. For more elaborate descriptions of both the ToF message and switching schemes, the reader is referred to [21, 25].

An extended Kalman filter is employed by the positioning system to estimate relative position and yaw for each drone $j, j = 1, ..., n, j \neq i$, in the HF of drone i (n is the number of drones in the swarm). Since the filtering equations are the same for each relative pose to estimate, the filter is described for one pair of UAVs only. Let X_{ij} contain the pose of UAV j expressed in $HF : i$,

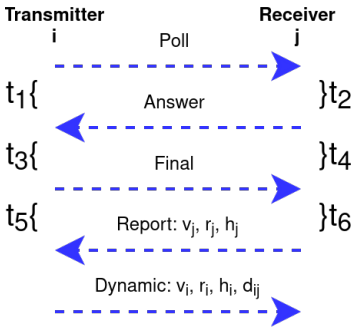


Figure 2.4: *The communication protocol for inter-drone distance measurements and sharing of velocity data.*

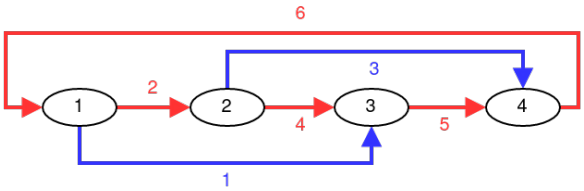


Figure 2.5: *The method for passing around the transmitter role. Switching role type occurs at the red arrow communications (both involved drones switch roles).*

$$X_{ij} = \begin{bmatrix} x_{ij} & y_{ij} & \psi_{ij} \end{bmatrix}^T, \tag{2.19}$$

where x_{ij} and y_{ij} are the x- and y-positions of UAV j in $HF : i$, and ψ_{ij} is the yaw angle of UAV j in $HF : i$. The input to the filter, U_{ij}, is the measured translational and rotational velocities of UAVs i and j,

$$U_{ij} = \begin{bmatrix} v_i^T & r_i & v_j^T & r_j \end{bmatrix}^T = \begin{bmatrix} v_i^x & v_i^y & r_i & v_j^x & v_j^y & r_j \end{bmatrix}^T. \tag{2.20}$$

In the above, v denotes translational velocity and r denotes rotational velocity. All velocities are expressed in the HF of each respective UAV. The motion model presented in Section 2.4.1 is used to describe the rate of change in X_{ij} given a current state and input,

$$\dot{X}_{ij} = f(X_{ij}, U_{ij}) = \begin{bmatrix} \cos(\psi_{ij})v_j^x - \sin(\psi_{ij})v_j^y - v_i^x + y_{ij}r_i \\ \sin(\psi_{ij})v_j^x + \cos(\psi_{ij})v_j^y - v_i^y - x_{ij}r_i \\ r_j - r_i \end{bmatrix}. \tag{2.21}$$

A first-order Taylor expansion based on the relationship of (2.21) is done to create a discrete model for state update,

$$X_{ij,k+1} = f(X_{ij,k}, U_{ij,k})\Delta t + X_{ij,k} = F(X_{ij}, U_{ij})_{|X_{ij}=X_{ij,k}, U_{ij}=U_{ij,k}}. \tag{2.22}$$

The velocities used as input to the filter are in reality noisy estimates of the true velocities of the UAVs, and the process model needs to take this into account. The effect of the noise can be approximated with another first-order Taylor expansion, as described in Section 2.3, to generate a model that includes process noise,

$$X_{ij,k+1} = F(X_{ij,k}, U_{ij,k}) + \frac{\partial F}{\partial U_{ij}}_{|U_{ij}=U_{ij,k}} w_k. \tag{2.23}$$

In the above, w denotes the input noise. Let A be the Jacobian of F w.r.t. X_{ij} and B be the Jacobian of F w.r.t. U_{ij},

$$A = \frac{\partial F}{\partial X_{ij}} = \begin{bmatrix} 1 & r_i\Delta t & (-\sin(\psi_{ij})v_x^j - \cos(\psi_{ij})v_j^y)\Delta t \\ -r_i\Delta t & 1 & (\cos(\psi_{ij})v_x^j - \sin(\psi_{ij})v_j^y)\Delta t \\ 0 & 0 & 1 \end{bmatrix}, \tag{2.24a}$$

$$B = \frac{\partial F}{\partial U_{ij}} = \begin{bmatrix} -1 & 0 & y_{ij} & \cos(\psi_{ij}) & -\sin(\psi_{ij}) & 0 \\ 0 & -1 & -x_{ij} & \sin(\psi_{ij}) & \cos(\psi_{ij}) & 0 \\ 0 & 0 & -1 & 0 & 0 & 1 \end{bmatrix}. \tag{2.24b}$$

Let A_k and B_k be the values of A and B when evaluating them at the state estimate $\hat{X}_{ij,k-1}$ and the input $U_{ij,k}$. The measurements fed to the filter are inter-drone distances. Since the filtering is described for one pair of UAVs, only one distance measurement is considered. Let h be a function describing the true inter-drone distance between UAV i and j, z, based on the state variables x_{ij} and y_{ij},

$$z = h(X_{ij}) = \sqrt{x_{ij}^2 + y_{ij}^2}. \tag{2.25}$$

Let the vector H be the Jacobian of h, and let H_k denote the value of H when evaluating it at the state prediction $\hat{X}_{ij,k|k-1}$,

$$H = \begin{bmatrix} \frac{x_{ij}}{\sqrt{x_{ij}^2+y_{ij}^2}} & \frac{y_{ij}}{\sqrt{x_{ij}^2+y_{ij}^2}} & 0 \end{bmatrix} = \begin{bmatrix} x_{ij}/z & y_{ij}/z & 0 \end{bmatrix}. \tag{2.26}$$

Let the covariance matrix of the input and measurement noise be denoted Q and R, respectively. Q is 6×6 diagonal and R is a scalar.

$$Q_S = \begin{bmatrix} \sigma_{v_{xy}}^2 & 0 & 0 \\ 0 & \sigma_{v_{xy}}^2 & 0 \\ 0 & 0 & \sigma_{v_r}^2 \end{bmatrix}, Q = \begin{bmatrix} Q_S & \mathbf{0}_3 \\ \mathbf{0}_3 & Q_S \end{bmatrix} \tag{2.27a}$$

$$R = \sigma_d^2. \tag{2.27b}$$

Above, $\sigma_{v_{vy}}$ is the standard deviation for the error in translational velocity estimates, σ_{v_r} is the deviation for rotational velocity estimates and σ_d is the deviation for ranging measurements. Finally, with the notations presented above and the theory from Section 2.3, the prediction step for the filtering is described by

$$\hat{X}_{ij,k|k-1} = F(\hat{X}_{ij,k-1}, U_{ij,k}) \tag{2.28a}$$

$$P_{k|k-1} = A_k P_{k-1} A_k^T + B_k Q B_k^T, \tag{2.28b}$$

and the update step is described by

$$K_k = P_{k|k-1} H_k^T (H_k P_{k|k-1} H_k^T + R)^{-1} \tag{2.29a}$$

$$\hat{X}_{ij,k} = \hat{X}_{ij,k|k-1} + K_k(z_k - h(\hat{X}_{ij,k|k-1})) \tag{2.29b}$$

$$P_k = (I - K_k H_k) P_{k|k-1}. \tag{2.29c}$$

2.4.3 Multidimensional scaling

Multidimensional scaling (MDS) is an algorithm that, among other applications, can be used to generate positional coordinates based on pairwise ranging measurements between sensor nodes. A general MDS procedure [33] for positioning n objects in two dimensions is presented in the list below.

1. Create an $n \times n$ matrix $\hat{D}$ that has the estimated distance between agent i and j, $\bar{d}_{ij}$, in its position (i, j). Apply the centering operation to obtain B, $B = -\frac{1}{2} C \hat{D} C$, where $C = I - \frac{1}{n} J_n$ is the centering matrix. I is an $n \times n$ identity matrix and J_n is an $n \times n$ matrix of ones.

2. Calculate the two (since output coordinates should be two-dimensional) largest eigenvalues of B and their respective eigenvectors.

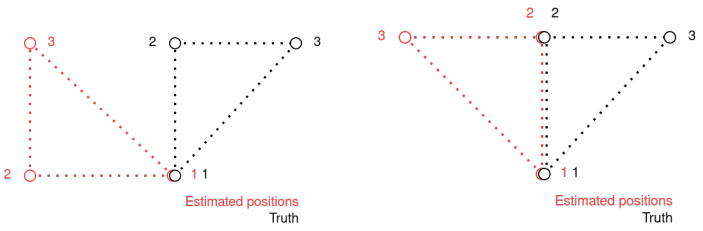
3. The MDS result is $\mathcal{S}^* = U_m \Lambda_m^{\frac{1}{2}}$, where Λ_m is a diagonal matrix with the two largest eigenvalues on the diagonal (in order from smallest to largest) and U_m is a matrix with the eigenvalues' respective eigenvectors as columns (in corresponding order). $\mathcal{S}^*$ becomes an $n \times 2$ matrix with the x and y coordinates of all drones in the swarm as rows [34]. The coordinates are expressed in the HF of drone 1 but might be mirrored and rotated around the origin such that the matrix of true relative positions, S, can be expressed as either $\mathcal{S} = M(\theta) F \mathcal{S}^*$ or $\mathcal{S} = M(\theta) \mathcal{S}^*$, depending on if there is an incorrect mirroring involved or not [22]. The rotational error is in this case θ and the matrices M and F are described in (2.30a) and (2.30b).

$$M(\theta) = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix}, \tag{2.30a}$$

$$F = \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix}. \tag{2.30b}$$

Considering the three-agent scenario in Figure 2.6a, the determined positions of agents 2 and 3 are rotated around agent 1, without violating the distance constraints. In this case, the relationship between $\mathcal{S}$ and $\mathcal{S}^*$ is $\mathcal{S} = M(\frac{-\pi}{2})\mathcal{S}^*$. Their positions might also be flipped similar to what is shown in Figure 2.6b. In that case, the relationship is $\mathcal{S} = M(0)F\mathcal{S}^* = F\mathcal{S}^*$.

In [22], a solution to this ambiguity problem is proposed. The approach suggested there is one that is capable of dealing with ambiguities as well as being resilient to noisy ranging measurements. It works by moving one of the agents in the swarm two times (while estimating the movement distances and directions with onboard sensors) and making standard MDS position estimates before, in between, and after the two moves. The algorithm is summarized in Algorithm 2.1. Since the underlying mathematics of the steps involved in Algorithm 2.1 is described across several pages in [22], and the usage of the procedure related to this report is rather ad hoc, the equations for implementing the procedure is presented here in a concrete and minimal way. The interested reader is referred to [22] for more detail and intuition.



(a) *Wrongly rotated position estimates and true positions.*

(b) *The estimates might be mirrored along the y-axis while still satisfying distance constraints.*

Figure 2.6: *The ambiguity problem depicted.*

In Algorithm 2.1, the first step consists of collecting the pairwise distances and creating a distance matrix $\hat{D}$. The standard MDS procedure, as described in the list above, is then applied to calculate $\mathcal{S}^*$.

After $\mathcal{S}^*$ has been calculated, the next step is to move one of the agents (in this description agent 1) to a new position. $\Delta x_1'$ and $\Delta y_1'$ is the movement of the agent described in its prior-to-movement HF (see Section 2.2). Then, distances are once again measured and a new distance matrix $\hat{D}'$ is created using the same method as earlier in the process. The first estimate of the angle with which to clear out the rotation ambiguity, $\hat{\theta}_{R,1}$, is defined as

$$\hat{\theta}_{R,1}(\mathcal{S}^*, \Delta\mathcal{S}', \hat{D}') = \arg\min_{\theta\in\theta_\lambda} \sum_{i=1}^{n} (a_i + b_i\cos(\theta) + c_i\sin(\theta))^2 . \tag{2.31}$$

Here, a_i, b_i, c_i is defined as in (2.32) and θ_λ as the set described in (2.33). In (2.33), λ_j represent the roots of the equation that comes from deriving the sum

Data: Pairwise distance measurements
begin
  Collect $\bar{d}_{ij}$ for $i, j = 1, ..., n$ and determine $\hat{D}$
  Calculate $\mathcal{S}^*$
end
begin
  Move agent 1 to $[\Delta x_1', \Delta y_1']^T$
  Collect $\bar{d}'_{ij}$ for $i, j = 1, ..., n$ at the new position and determine $\hat{D}'$
  Calculate $\hat{\theta}_{R,1} = \hat{\theta}_R(\mathcal{S}^*, \Delta\mathcal{S}', \hat{D}')$
end
begin
  Move agent 1 to $[\Delta x_1'', \Delta y_1'']^T$
  Collect $\bar{d}''_{ij}$ for $i, j = 1, ..., n$ at the new position and determine $\hat{D}''$
  Calculate $\hat{\theta}_{R,2} = \hat{\theta}_R(\mathcal{S}^{**}, \Delta\mathcal{S}'', \hat{D}'')$
  Estimate the position matrix $\hat{\mathcal{S}}$
end

Algorithm 2.1: The positioning algorithm based on multidimensional scaling coordinates presented in [22]. Some of the notations are introduced in the running text.

$\sum_{i=1}^{n} (a_i + b_i \cos(\theta) + c_i \sin(\theta))^2$ from (2.31) and setting it equal to zero.

$$\begin{aligned} a_i &= {d_{1,i}}^2 - {d'_{1,i}}^2 + {\Delta x'_1}^2 + {\Delta y'_1}^2 \\ b_i &= -2(x_i^* \Delta x'_1 + y_i^* \Delta y'_1) \\ c_i &= 2(x_i^* \Delta y'_1 - y_i^* \Delta x'_1). \end{aligned} \tag{2.32}$$

$$\begin{aligned} \theta_\lambda &= \{\arcsin(\lambda_j), g(\pi - \arcsin(\lambda_j)) : j \in \{1, 2, 3, 4\}\} \in [-\pi, \pi), \\ g(x) &= x - 2\pi \left\lfloor \frac{x}{2\pi} + \frac{1}{2} \right\rfloor. \end{aligned} \tag{2.33}$$

The last step of Algorithm 2.1 starts off in the same way as the second. Agent 1 is moved to another new position, the move distance is measured, and $\Delta x_1''$ and $\Delta y_1''$ are defined as the total movement from the agent's initial position (before the first move). Then a new distance matrix $\hat{D}''$ is estimated. $\mathcal{S}^{**}$ is calculated by rotating $\mathcal{S}^*$ with the angle $\hat{\theta}_{R,1}$ obtained in the previous step, as in (2.34).

$$\mathcal{S}^{**} = \begin{bmatrix} \cos(\hat{\theta}_{R,1}) & \sin(\hat{\theta}_{R,1}) \\ -\sin(\hat{\theta}_{R,1}) & \cos(\hat{\theta}_{R,1}) \end{bmatrix} \mathcal{S}^*. \tag{2.34}$$

When $\mathcal{S}^{**}$ has been determined, $\hat{\theta}_{R,2}$ can be calculated as in (2.31) but with $d'_{1,i}$, x_i^*, y_i^*, $\Delta x'_1$ and $\Delta y'_1$ in (2.32) replaced with the new corresponding values from $\hat{D}''$, $\mathcal{S}^{**}$ and $\Delta\mathcal{S}''$. The algorithm is concluded by determining the final position

estimates

$$\hat{\mathcal{S}} = \begin{cases} M(\hat{\theta}_{R,1})\mathcal{S}^* & |\hat{\theta}_{R,2}| \leq |l| \\ M(\hat{\theta}_R(F\mathcal{S}^*, \Delta\mathcal{S}', \hat{D}'))F\mathcal{S}^* & |\hat{\theta}_{R,2}| > |l|, \end{cases} \tag{2.35}$$

where

$$l = \frac{1}{2}g(-2\arctan2(\Delta x_1', \Delta y_1') + 2\arctan2(\Delta x_1'', \Delta y_1'')). \tag{2.36}$$

2.5 Constructing a map

Mapping the environment can be done in both 2D and 3D and gives a picture of what the environment looks like. Mapping is a task that relies heavily on processing sensor data or capturing features in the image. By estimating the distance between identified features, information about where obstacles are (or are not) located can be added to the map. Mapping with a UAV involves estimating the position of the UAV, collecting information about the environment, and simultaneously inputting more information about the environment based on what is seen. If the position of the UAV is known and the sensors are not noisy, a common approach is to create a point cloud in which the position of each identified feature is converted from the body frame of the sensing UAV to the global frame and stored on the map [23].

Noisy sensor data or uncertain UAV position will result in faulty positions for features, resulting in a blurry point cloud map, and in the worst case an unreadable map. By discretizing the world and converting the map into a grid a readable map can be obtained at the expense of lower resolution. Each grid cells represent a square in 2D space or a cube in 3D space and are labeled as either free or occupied. However, discrete labeling is risky because it results in a discretized point cloud map. The probabilistic occupancy grid stores the probability that each cell is occupied, leading to a simple update rule [31].

2.5.1 Probabilistic occupancy grid

The aim of occupancy grid mapping is to determine the joint probability for every cell in a grid mesh being occupied. Each cell $m[i, j]$ in the (2D) grid keeps track of the probability, $p(m[i, j])$, that the cell is occupied. In practice, this means that a cell with value 0 corresponds to an unambiguously free cell whilst a cell with value 1 corresponds to an unambiguously occupied cell. Values in-between correspond to more or less certainty that the cell is occupied. The grid is initialized with the value 0.5 for all cells if no prior information is given [31]. It is assumed that the grid has a fixed resolution and that the area corresponding to a cell is either completely free or completely occupied. It is also assumed that the world is static and that the cell values are independent of each other. The probability that a cell is occupied, given the prior probability of the cell being occupied, $p(m[i, j])$, and an obstacle measurement z_t at time t is then, according to Bayes rule

$$p(m[i, j]|z_t) = \frac{p(z_t|m[i, j])p(m[i, j])}{p(z_t)}, \tag{2.37}$$

where $p(z_t)$ denotes the certainty that the measured cell $m[i, j]$ is occupied. This term is the so-called inverse sensor model [31]. For N measurements the joint probability is

$$p(m[i, j]|z_{1:N}) = \prod_{t=1}^{N} p(m[i, j]|z_t). \tag{2.38}$$

The odds of an event occurring are defined as $o(A) = \frac{p(A)}{p(\neg A)}$, and the odds that a cell is occupied, given measurements $z_{1:N}$, are then

$$o(m[i, j]|z_{1:N}) = \frac{p(m[i, j]|z_{1:N})}{p(\neg m[i, j]|z_{1:N})} = \frac{p(m[i, j]|z_N)}{p(\neg m[i, j]|z_N)} \frac{p(m[i, j]|z_{1:N-1})}{p(\neg m[i, j]|z_{1:N-1})} \frac{p(\neg m[i, j])}{p(m[i, j])}. \tag{2.39}$$

By taking the logarithm and defining the prior term

$$l_0 = \log \frac{p(m[i, j])}{1 - p(m[i, j])} = -\log \frac{1 - p(m[i, j])}{p(m[i, j])}, \tag{2.40}$$

the previous term

$$l_{t-1} = \log \frac{p(m[i, j]|z_{1:t-1})}{1 - p(m[i, j]|z_{1:t-1})}, \tag{2.41}$$

and the log odds of (2.37)

$$s_t = \log \frac{p(m[i, j]|z_t)}{1 - p(m[i, j]|z_t)}, \tag{2.42}$$

the standard update strategy for probabilistic occupancy grids is achieved [31],

$$l_t(m[i, j]) = s_t(m[i, j], z_t) + l_{t-1}(m[i, j]) - l_0(m[i, j]). \tag{2.43}$$

As previously stated, the map is initialized with the prior $p(m[i, j]) = 0.5$ when no previous information about the environment is given. This gives $l_0 = \log \frac{1-0.5}{0.5} = 0$, and eliminates (2.40) from (2.43) to get the update algorithm

$$l(m[i, j]) = \sum_{t=1}^{N} s_t(m[i, j], z_t) \tag{2.44}$$

with the ability to retrieve $p(m[i, j])$ as

$$p(m[i, j]) = 1 - \frac{1}{1 + \exp l(m[i, j])}. \tag{2.45}$$

Note that the multiplication in (2.38) is now an addition. In the update algorithm (2.44) for each cell in a grid, (2.42) determines the value to increment (or decrement) a cell with [31].

2.5.2 Inverse sensor model

The inverse sensor model is determined by the type of sensor and specifies the occupation probability for the cells indicated by the sensor. A common model for a range sensor such as sonar or LIDAR works by updating all cells within the sensor cone with a value of $\log o(p_{free})$ and a value of $\log o(p_{occ})$ to cells at the measured range. The sensor cone is the area of free space the sensor has detected until the target. Nothing can be said about the occupation probability for cells farther away than the detected cells. For an ideal inverse sensor model $p_{free} = 0$ and $p_{occ} = 1$ [16, 31]. [16] incorporates uncertainty into this model by convolving the ideal sensor model with a Gaussian distribution. The ideal inverse sensor model is shown in Figure 2.7 along with the Gaussian sensor model introduced in [16]. The width of the bar corresponds to the accuracy of the sensor reading.

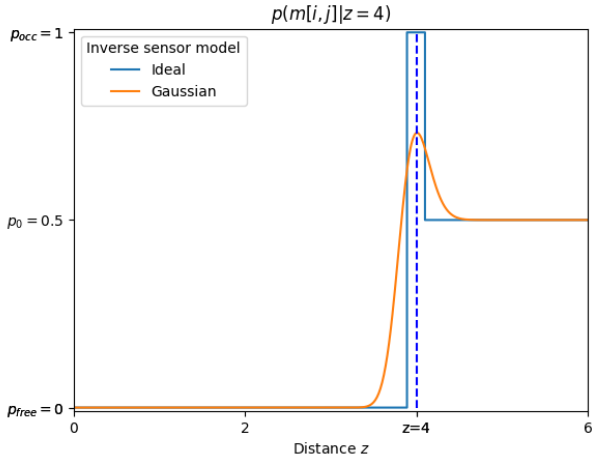


Figure 2.7: *The ideal inverse sensor model and the ideal inverse sensor model convolved with a Gaussian noise model.*

2.5.3 Bresenham line algorithm

Given the position of a UAV $p_{cf} = (x_1, y_1)$ and a measured point $z_t = (x_2, y_2)$ in the global frame, it is of interest to know all points in the grid in-between since they are free (unoccupied). The Bresenham line algorithm approximates a straight line in a grid lattice and allows the free cells [10] to be updated. The algorithm differs slightly depending on the direction of the line. With the coordinates in the body frame of the UAV, the line will always start at the origin. The measured point expressed in the body frame is

$$p_b = s_r \begin{bmatrix} \cos(s_b) \\ \sin(s_b) \\ 0 \end{bmatrix}, \tag{2.46}$$

where s_b is the sensor bearing, that is the angle in the body frame at which the sensor reads the distance s_r, called the sensor range. The measured point in the body frame is converted to the global frame with (2.4). The algorithm will return a discretized sequence of coordinates (X, Y) from p_{cf} to z_t.

Consider a line in the standard two-dimensional Cartesian coordinate system. Let the slope of the line be less than one, and let the line be in the first quadrant. Since the line has a gradient of less than one, the sequences will be on the form

$$X = [\lfloor x_1 \rfloor, \lfloor x_1 \rfloor + 1, \ldots, \lfloor x_2 \rfloor]^T$$

and

$$Y = [\lfloor y_1 \rfloor, \ldots, \lfloor y_1 \rfloor, \lfloor y_1 \rfloor + 1, \ldots, \lfloor y_1 \rfloor + 1, \ldots, \lfloor y_2 \rfloor]^T .$$

Algorithm 2.2 applies the Bresenham line algorithm. The error between the real value of y and the Bresenham value of y given x accumulates in P_k and determines when y must be incremented. For lines in other quadrants or other gradients, the line is transformed by swapping x and y and (or) changing incrementation to decrementation. Figure 2.8 illustrates the algorithm using $p_{cf} = (0.5, 1.5)$ and $z_t = (9.5, 4.5)$.

Data: Start (x_1, y_1), end (x_2, y_2)
Result: List with coordinates, $L = (X, Y)$
Ensure $x_2 > x_1$, otherwise, flip the start and end points.
$dx \leftarrow x_2 - x_1$, $dy \leftarrow y_2 - y_1$
Ensure $dy/dx \in [0, 1]$, otherwise switch x and y.
$x \leftarrow \lfloor x_1 \rfloor$, $y \leftarrow \lfloor y_1 \rfloor$
$P_k \leftarrow 2dy - dx$
while $x \leq x_2$ **do**
  Append (x, y) to L
  if $P_k \geq 0$ **then**
    $y \leftarrow y + 1$
    $P_k \leftarrow P_k - 2dx$
  end
  $x \leftarrow x + 1$
  $P_k \leftarrow P_k + 2dy$
end

Algorithm 2.2: Bresenham line algorithm.

2.5.4 Uncertain pose

The assumption that the pose of a UAV is known exactly is rarely true in practice, especially without a global positioning system. If the pose has a large uncertainty, the resulting map will not be an accurate representation of the real world. The pose consists of the position and orientation of a UAV. [16] addresses the problem by updating a set of possible positions for the observed obstacle. Each cell in the distribution of positions is associated with a weight, corresponding to the

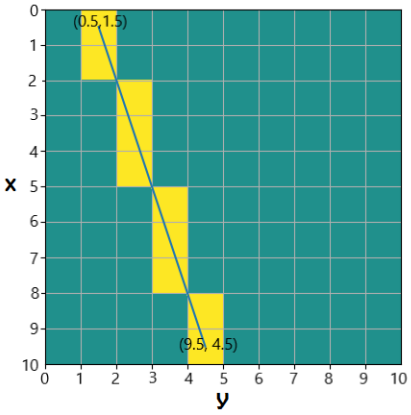


Figure 2.8: *The Bresenham line algorithm returns the yellow cells. The original line from* $(0.5, 1.5)$ *to* $(9.5, 4.5)$ *is seen in blue.*

likelihood of the obstacle being there. The update value (2.42) is computed for each position and multiplied with the corresponding weight before updating the cell's value in the grid. If no given distribution is available, the position is convolved with a Gaussian distribution by sampling n points around the estimated position. The weights are all $1/n$. Instead of trusting the estimated position and updating one cell, the update is smeared out in the grid around the cell. The time complexity for n sampled points is $\mathcal{O}(n)$.

2.6 Map-merging for multiple UAVs

There are two alternatives for a swarm to map collaboratively. Either they update the same map, or one map is maintained individually by each agent and merged offline.

2.6.1 Simultaneous map update

For a swarm to update the same map, the global coordinate system must coincide for all UAVs in the swarm. With ideal sensors, the order that the map is updated does not matter. Assuming that the position estimate of each agent does not drift, the map updates for a swarm can therefore be done in parallel or sequentially. For a sequential solution, there is no difference between one UAV traversing two trajectories or two UAVs traversing one trajectory each. To understand the principle, it helps to observe the case where the prior term in (2.44) is not neglected. With two sequential runs, the final map after the first run becomes the prior term for the second. Of course, the prior term can be neglected here as well, since

$$l(m[i,j]) = \sum_{t=1}^{N} s_t = \sum_{t=1}^{M} s_t + \sum_{k=M+1}^{N} s_k,\ 1 < M < N \tag{2.47}$$

can present two subsequent sequential turns. Here, M partitions the N measurements. For a parallel run, the UAVs will have an equal number of samples N updating the grid,

$$l(m[i,j]) = \sum_{t,k=1}^{N} (s_t + s_k), \tag{2.48}$$

where s_t, and s_k are the individual sensor model updates for the two UAVs. Observe that, if s_t and s_k update the same cells in (2.47) as in (2.48) the two equations are equivalent. For a solution where a swarm updates the same map, the positional perception must coincide for all UAVs. This is however not the case if no global positioning system is used, and there will be cases when the UAVs disagree about the map. The map will show what it is updated with most times.

2.6.2 Multirobot map merging

Another approach is to let each UAV construct a map in its local coordinate system and afterward merge the maps. There exist numerous methods for pairwise map-merging, but [35] suggests an optimization method to find the rotation and translation applied to one of the maps that maximize the overlap between the maps. The rotation and translation from one map to another are determined by three parameters, the angle to rotate the grid map with, θ, and the translation in the x- and y-direction, t_x and t_y, respectively. The function associated with these parameters takes a coordinate, (x, y) in the plane, rotates it around the origin by angle θ, and translates it with t_x, t_y [13].

The transformed map $m' = T_m(\theta, t_x, t_y)$ applied to map m is calculated by mapping each cell in m to one cell in m'. The mapping between $m[i,j]$ and $m'[i,j] = m[i',j']$ where

$$(i',j') = \left\lfloor \left(\begin{pmatrix} \cos\theta & -\sin\theta & t_x \\ \sin\theta & \cos\theta & t_y \end{pmatrix} \begin{pmatrix} i \\ j \\ 1 \end{pmatrix} \right)^T \right\rfloor. \tag{2.49}$$

The floor operator is necessary to give a discrete index for the cell.

In order to find the optimal rotation and translation given two maps, m_1 and m_2, a cost function

$$w(m_1, m_2) \tag{2.50}$$

that measures how much the two maps agree is maximized. The task of finding an adequate function can be challenging and dependent on the properties of the grid map. In [9] a cost function is suggested that compares the exact value in each grid cell

$$w(m_1, m_2) = \sum_{i=1}^{N} \sum_{j=1}^{M} \mathrm{Eq}\,(m_1[i,j], m_2[i,j]), \tag{2.51}$$

where $\mathrm{Eq}(a,b) = 1$ if $a = b$ and 0 otherwise, and $m_1, m_2 \in \mathbb{R}^{N\times M}$. This method works well when working with binary grids, where each cell is either occupied,

free or unknown. Implementing the probabilistic grid map in its log-odds form allows floating classification and requires a function that instead measures the number of falsely classified cells. For this, the accuracy measure implemented in [13] can be useful, where the metric

$$w(m_1, m_2) = \frac{\text{agr}(m_1, m_2)}{\text{agr}(m_1, m_2) + \text{dis}(m_1, m_2)}, \tag{2.52}$$

is defined. In (2.52) $\text{agr}(m_1, m_2)$ is the number of correctly classified cells, which is the number of cells that are both free or both occupied, and $\text{dis}(m_1, m_2)$ is the number of incorrectly classified cells, which is the number of cells that in m_1 is occupied but free in m_2 or vice versa.

With the cost function $w(m_1, m_2)$ and the transformation $T_m(\theta, t_x, t_y)$ for map m it is of interest to find

$$\theta^*, t_x^*, t_y^* = \underset{\theta, t_x, t_y}{\arg\max}\, w\left(m_1, T_{m_2}(\theta, t_x, t_y)\right). \tag{2.53}$$

For n maps there are $\binom{N}{2}$ pairwise parameters to optimize for, and the maps can be merged in an equal number of ways. In [13] the merging is done recursively pairwise until one final map is obtained. For four different maps the merging would be

$$(m_1 \oplus m_2) \oplus (m_3 \oplus m_4) = m_{12} \oplus m_{34},$$

where $m_i \oplus m_j$ fuses two maps by summing the odds for each cell. This fusion will improve the classification (free or occupied) in regions where the fused maps agree.

The method then needs to find $n - 1$ transformations. [9] discusses how the map cost function can be extended to cover the overlap between all these maps. The cost function then calculates

$$w(m_1, m_2, \ldots, m_n)$$

where the Eq function is extended to overlap for all maps, that is $\text{Eq}(a_1, a_2, \ldots, a_n) = 1$ if $a_1 = a_2 = \cdots = a_n$ and 0 otherwise. The task is then to determine the $n - 1$ transformations

$$T_{m_k}(\theta, t_x, t_y)$$

from m_1 to m_k that maximize

$$w(m_1, T_{m_2}(\theta, t_x, t_y), \ldots, T_{m_n}(\theta, t_x, t_y)).$$

However, there is no investigation of the suggested method as [9] performs pairwise merging. The merging is then done either by summing or taking the mean values of the log-odd grid values [9, 13].

3

System Description

This chapter provides a description of the Crazyflie platform and the systems that have been developed during the course of the thesis.

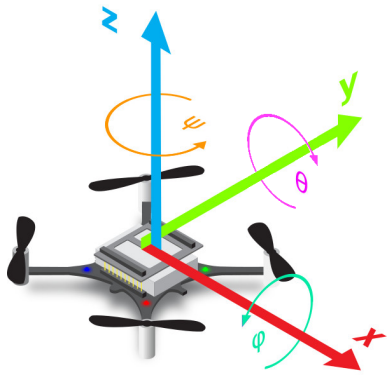


Figure 3.1: *The body coordinate system of the Crazyflie. x is the heading direction [3].*

3.1 The Crazyflie platform

The hardware for which the algorithms are tuned and tested, are nano-drones of type *Crazyflie 2.0* developed by Bitcraze and can be seen in Figure 3.1. Out of the box, each drone weighs around 30 grams and fits in the palm of a hand. The drones can be equipped with Bitcraze's extension decks: a multi-ranger deck that has five (left, right, front, rear, and up) time of flight (ToF) LIDAR sensors for distance measurements to the surroundings, a flowdeck for optical position-

ing relative to the floor, and a Loco positioning deck for UWB communication [1, 4, 5, 8]. The drones communicate independently via Crazyradio PA with a computer on which the mapping algorithm is executed, but each drone has the capability of estimating its own relative position when using the system described in Section 2.4.2.

The LIDARs mounted on the crazyflie can detect objects up to 4 m away with an accuracy of a few millimeters [8]. The precision was validated by moving a drone from 0.25 m from an obstacle to 1 m away. Figure 3.2a illustrates the measured ranges and the reference value. From the measurements where reference values exist the errors were calculated and are illustrated in Figure 3.2b. The standard deviation of the error is 2 mm.

Out of the box, the Crazyflie comes with an extended Kalman filter, used to estimate rotation, position, and velocity of the UAV [24]. The EKF, illustrated in Figure 3.3, is built to work with any configuration of decks mounted on the Crazyflie. A configuration can involve internal sensors, the flow deck, and in part, the Loco positioning system [2]. With the Loco positioning system, specified in [7], the position estimation has an accuracy of 1 dm [6].

3.1.1 Velocity estimation

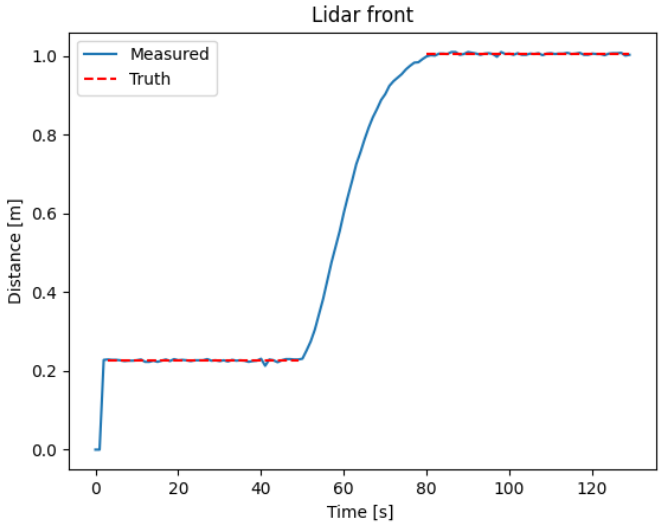
The internal EKF velocity estimator which is part of the Crazyflie firmware express velocities in the body frame of the drone, as depicted in Figure 2.1b. For the positioning systems described in this report, x and y velocities expressed in the HF of each drone are needed. Transformations from body frame to HF are covered in Section 2.2.

3.1.2 Measuring inter-drone distances

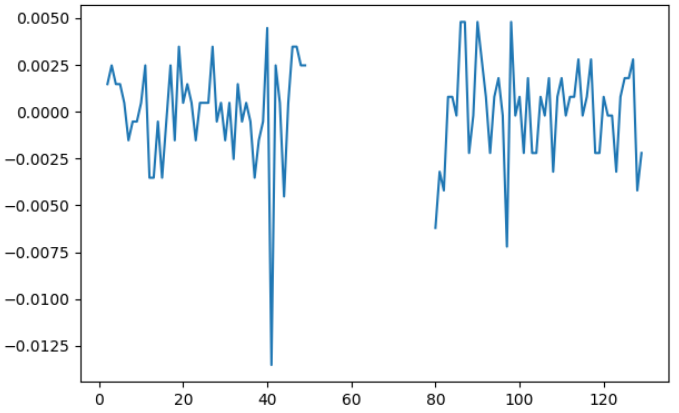
The Crazyflies can, as mentioned, be equipped with UWB decks, available from Bitcraze. This deck allows for communication between UAVs. The ToF for the messages passed back and forth between a pair of agents can be utilized to estimate the distance between them.

3.2 Mapping node

The mapping node run in parallel to the navigation module, described in Section 3.3, and relative localization. The grid map is controlled by two parameters, the resolution R_{map} [m] describing the side length of each square in the grid and the size S_{map} [m]. The product of these two parameters determines the matrix dimensions of the grid. For a swarm, the methods introduced in Section 2.6 was used. When the Crazyflies fly simultaneously there is a chance that the LIDAR detects other UAVs, and inserts them as obstacles. This is not a wanted feature, as they do not make up occupied space in the map nor are static. These scans are rejected by checking the position of the scan with the estimated positions of the other drones. Note that the UAVs must share a global coordinate system for this to work. If there is no shared coordinate system so that the UAVs can not reject



(a) *LIDAR measurements in blue and reference values in dashed red.*



(b) *Distance error for the parts that have a reference value.*

Figure 3.2: *LIDAR distances to a wall when moving a drone, and the errors compared to reference values. The error values have a standard deviation of 2 mm.*

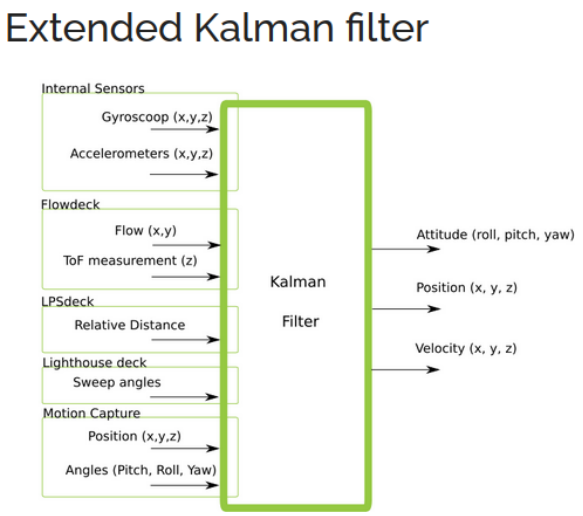
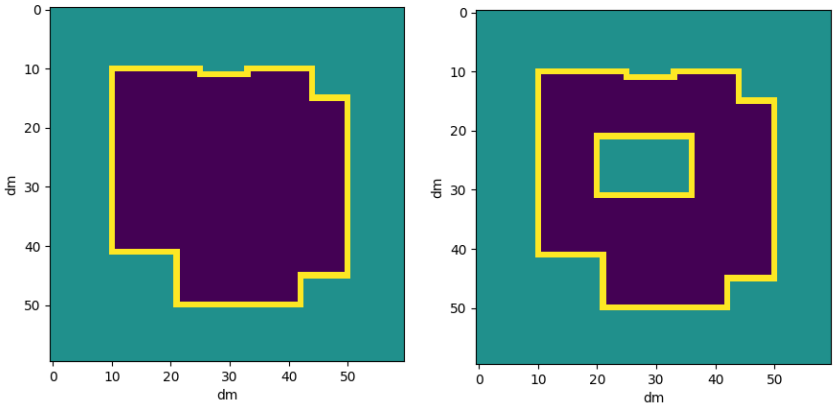


Figure 3.3: *The inputs and outputs for the built-in extended Kalman filter on the Crazyflie platform [2]. The roll and pitch angle is expressed in the HF of the agent. The yaw and position values are expressed in a coordinate system fixed in the global frame, with axes pointing in the same directions as the axes of the HF at UAV startup. The velocities are expressed in the body frame of the agent.*

these scans, they are inserted into the map. When the UAVs fly around, the grid index that they are seen at from another UAV constantly changes. For a longer flight, each UAV will measure the index being free more times than occupied, more or less eliminating the issue of inserting a UAV as an obstacle.

The test environment consisted of cardboard boxes limiting the area to explore. The ground truth maps seen in Figure 3.4 make up a scenario to test how well the mapper captures corners, and a scenario to test the capture of static obstacles inside the free space. The test was performed both using the Loco system and with relative positioning. The map in Figure 3.4b has 1.32 times more wall length than the map in Figure 3.4a. On the other hand, the free space is 14% less.

When constructing the map, the size must be greater than the explored area for the map to fit. For these maps to fit in the grid with any rotation, the side lengths of the grid$S_{map} = 6$ [m]. A higher resolution enables a more detailed map but increases the number of cells in the grid quadratically. The used resolution is $R_{map} = 10$ [1/m]. In order to not rely entirely on each update $p_{free} > 0$ and $p_{occ} < 1$, this is also needed in any implementation to keep the update values real. For the update of free and occupied cells, $p_{free} = 0.3$ and $p_{occ} = 0.8$, respectively, is used. For a merged map to be as similar as possible to a simultaneously updated map, the merging of two maps is performed by summing the log odds form of the grids. The maximal sampling frequency of the Crazyflie LIDARs was 100 Hz. This sampling frequency was therefore also used in the simulation environment. Each sample consisted of sensor range and positional data together



(a) Map scenario 1 *(b)* Map scenario 2

Figure 3.4: *Map scenarios. Purple indicates free space, yellow occupied, and turquoise unknown.*

with a timestamp.

The map cost function (2.50) was introduced in Section 2.6.2. The suggested cost function in (2.51) gives the number of identical cells. This number can be divided by the total number of cells to give the accuracy of the map. This depends on both the free area and the unknown space. It is also dependent on binary classification. The cost function in (2.52) on the other hand ignores the unexplored space but is also dependent on the area of the free space. It does take classification into account, which gives a better indication of the performance. To get rid of the dependency on the area of free space we suggest a slightly modified cost function

$$w(m_1, m_2) = \frac{\text{agr}_{\text{occ}}(m_1, m_2)}{\text{agr}_{\text{occ}}(m_1, m_2) + \text{dis}(m_1, m_2)}, \tag{3.1}$$

where agr_{occ}, unlike agr, ignores the number of cells that both are free. This instead measures the performance regarding correctly placed walls.

3.3 Navigation during mapping

The trajectory planning of the Crazyflies is kept simple and lets the UAVs explore the environment in a controlled manner. There is no planning strategy involved, but the UAVs fly based on LIDAR data. A UAV flies straight ahead if no wall is close in front or on the side of it. It will maintain the distance to a wall if either of the sides is close. The UAV turns left or right, based on the surrounding, if an obstacle is detected ahead. Algorithm 3.1 implements this strategy. On Line 3 the turn angle is randomized but close to 90 degrees. While the navigation method for the Crazyflies is rather simple, the trajectory will cover the whole 2D plane

eventually. As the LIDAR detects not only walls but other UAVs as well, the UAVs will not collide. Figure 3.5 illustrates the trajectory taken by three UAVs in the simulator when exploring map scenario 1. The heading of a UAV and its traveling direction does not have to coincide.

```
Data: Lidar scans
while state is planning do
    front = measurement.scan.front;
    left = measurement.scan.left;
    right = measurement.scan.right;
    if front is close then
        if left is close then
            turn right;
        else
            turn left;
        end
    else
        if left is close then
            hover right;
        end
        if right is close then
            hover left;
        end
        go straight;
    end
end
```

Algorithm 3.1: Guidance for the UAVs.

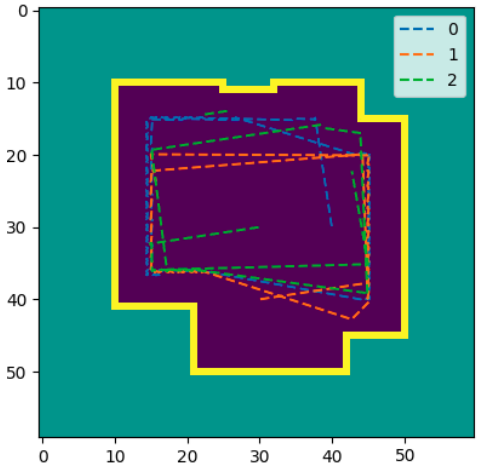


Figure 3.5: *The trajectory of three UAVs in the simulation environment. The label denotes the different UAVs.*

3.4 Extended relative positioning system

A relative positioning system is described in this section, extended compared to the system described in Section 2.4.2 in the sense that it can utilize all pairwise distances within a swarm of UAVs. In other words, this system builds upon the motion model associated with the basis system, but does not limit itself to using pairwise distances where the origin-UAV must be one part of each pair.

3.4.1 Complementary initialization procedure

The extended relative positioning system described in this section can be complemented with an initialization procedure. This is to decrease the convergence time of the subsequent filtering process. The goal of the procedure is to estimate the relative positions and yaw angles of the agents in a swarm. In the following, the Cartesian coordinate system located, scaled, and oriented as the HF of UAV 1 prior to the start of the procedure, is referred to as the initial HF.

The procedure begins with estimating the relative positions of all UAVs by performing the steps of Algorithm 2.1, moving agent 1 for 0.5 seconds in each of the two moves. The resulting $n \times 2$ matrix $\hat{\mathcal{S}}$ contains estimated x and y positions of all UAVs in the swarm, expressed in the initial HF, along its rows. Let the new position of UAV 1 in the initial HF be denoted $\begin{bmatrix}\Delta x_1 & \Delta y_1\end{bmatrix}$, and let

$$\mathcal{S}_1 = \hat{\mathcal{S}} - \mathbf{1}_n \begin{bmatrix}\Delta x_1 & \Delta y_1\end{bmatrix}. \tag{3.2}$$

Subtracting $\begin{bmatrix}\Delta x_1 & \Delta y_1\end{bmatrix}$ from all rows of $\hat{\mathcal{S}}$, as in the above, yields coordinates expressed in the HF of UAV 1, under the condition that the total movement of UAV 1 did not affect its rotation. Figure 3.6a depicts coordinates in $\mathcal{S}_1$.

Next, all agents of the swarm, except UAV 1 and UAV 2, are moved for 0.5 seconds with a speed of 1 m/s in their respective x-directions. The result of this is depicted in Figure 3.6b. After moving the agents, new position estimates are calculated using the standard MDS procedure presented in Section 2.4.3. The positions are stored along the rows of the matrix $\mathcal{S}_2$. The blue circles in Figure 3.6c depict example $\mathcal{S}_2$ coordinates.

Since the coordinates in $\mathcal{S}_2$ are subject to flip ambiguity,

$$\mathcal{S}_3 = \mathcal{S}_2 \begin{bmatrix}-1 & 0 \\ 0 & 1\end{bmatrix} \tag{3.3}$$

is introduced. Clearly, $\mathcal{S}_3$ contains the coordinates of $\mathcal{S}_2$ flipped along the x-axis of the HF of UAV 1. Yellow circles in Figure 3.6c depicts coordinates of $\mathcal{S}_3$.

Rotations around the origin of $HF : 1$ are then performed on $\mathcal{S}_2$ and $\mathcal{S}_3$ to move their respective UAV 2 position representations as close as possible to the position of UAV 2 in $\mathcal{S}_1$. Rotated versions of $\mathcal{S}_2$ and $\mathcal{S}_3$, $\mathcal{S}_{2,R}$ and $\mathcal{S}_{3,R}$, can be seen in Figure 3.6d. By doing this, rotational ambiguity is resolved.

The final step starts with picking either $\mathcal{S}_{2,R}$ or $\mathcal{S}_{3,R}$ based on which one has UAV 3 positioned closest to 0.5 m from where it is placed in $\mathcal{S}_1$. The chosen matrix is referred to as $\mathcal{S}_4$. This step resolves flip ambiguity, and $\mathcal{S}_4$ contains

non-ambiguous relative poses of UAVs 3 to n. To estimate the relative yaw angles of those agents, a comparison of the coordinates in $\mathcal{S}_1$ and $\mathcal{S}_4$ is used together with the knowledge that the agents have moved in their respective x-directions. Strictly, the yaw angle of UAV i, $3 \leq i \leq n$, is set to $\arctan2(x_i, y_i)$,

$$x_i = (\mathcal{S}_4 - \mathcal{S}_1)_{i1}, \; y_i = (\mathcal{S}_4 - \mathcal{S}_1)_{i2}. \tag{3.4}$$

Determining the yaw angle of UAV 2 is done by repeating all steps (except the one involving Algorithm 2.1), but using $\mathcal{S}_4$ as $\mathcal{S}_1$ and keeping UAV 3, instead of UAV 2, still.

3.4.2 Improved filtering

The filter described in Section 2.4.2 uses pairwise distances as measurement input. However, only the pairwise distances where the origin-UAV is part of the pair are used. In a swarm of n UAVs, there are $\binom{n}{2} = \frac{n^2-n}{2}$ distances available for measurement and usage in a positioning filter, but only $n-1$ are used by each such filter. The more UAVs in the swarm, the more possible distance measurements are left unused. In this section, a filter for n UAVs is proposed, where all available distance values are used. The filter assumes that the agent with index 1 is the origin-UAV. See Table A.1 in the Appendix for a description of the filter notations in the special case of a swarm with three UAVs.

Since the state of individual UAVs become dependent on the state of others when using all pairwise distances, i.e. the position state of UAV 3 can be affected by a measurement of the distance between UAV 1 and 2, the filter described here uses a state that contains information about the whole swarm. Let X be the relative poses of all other UAVs in the HF of UAV 1,

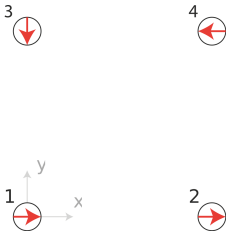
$$X = \begin{bmatrix} \mathbf{p_{12}} \\ \vdots \\ \mathbf{p_{1n}} \end{bmatrix} = \begin{bmatrix} x_{12} & y_{12} & \psi_{12} & \dots & x_{1n} & y_{1n} & \psi_{1n} \end{bmatrix}^T,$$

and let the input values that constitute U be translational velocities, v, and rotational velocities, r, both expressed in the HF of each individual UAV

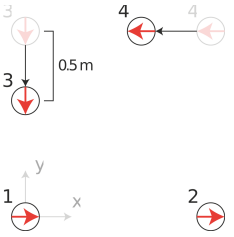
$$U = \begin{bmatrix} v_1^T & r_1 & \dots & v_n^T & r_n \end{bmatrix}^T,$$

$$v_i = \begin{bmatrix} v_{i,HF:i}^x & v_{i,HF:i}^y \end{bmatrix}^T.$$

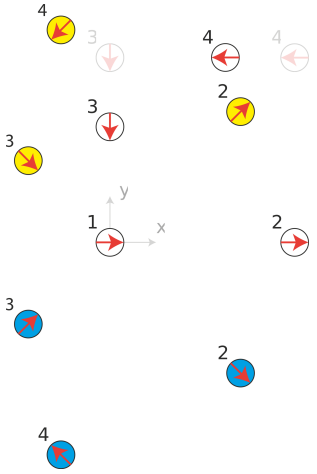
The process model equations used by the filter are essentially identical to the ones presented in Section 2.4.2, with the difference that $\dot{X}_{ij}$ of (2.21) is replaced



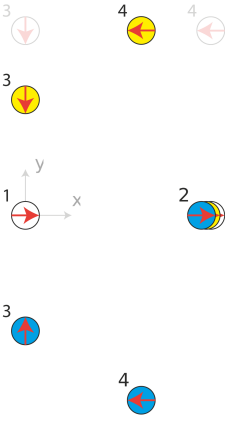
(a) *The initial poses of the agents, contained in the matrix $\mathcal{S}_1$.*



(b) *The poses of the agents after moving UAVs 3 to n.*



(c) *Here, $\mathcal{S}_2$ (blue) and $\mathcal{S}_3$ (yellow) are included. The rotation ambiguity has not been cleared out.*



(d) *Here, $\mathcal{S}_2$ and $\mathcal{S}_3$ have both been rotated. It is apparent that the flipped version $\mathcal{S}_{3,R}$ contains the true coordinates since the position of UAV 3 in $\mathcal{S}_{2,R}$ is clearly off from where it should be after moving the agent half a meter.*

Figure 3.6: *Some steps of the process described in Section 3.4.1 depicted. $\mathcal{S}_1$, $\mathcal{S}_2$, and $\mathcal{S}_3$ are assumed to be noiseless for visibility purposes.*

with a vector modeling the motion of all UAVs,

$$\dot{X} = \begin{bmatrix} \cos(\psi_{12})v_2^x - \sin(\psi_{12})v_2^y - v_1^x + y_{12}r_1 \\ \sin(\psi_{12})v_2^x + \cos(\psi_{12})v_2^y - v_1^y - x_{12}r_1 \\ r_2 - r_1 \\ \vdots \\ \cos(\psi_{1n})v_n^x - \sin(\psi_{1n})v_n^y - v_1^x + y_{1n}r_1 \\ \sin(\psi_{1n})v_n^x + \cos(\psi_{1n})v_n^y - v_1^y - x_{1n}r_1 \\ r_n - r_1 \end{bmatrix}.$$

Equations (2.22) and (2.23) are still relevant and correct for this implementation. The matrix A is expanded into a block diagonal matrix with $A_2, ..., A_n$ along the diagonal. The matrix B consists of $B_2, ..., B_n$ stacked vertically. In the expressions below, $i \geq 2$.

$$A_i = \begin{bmatrix} 1 & r_1\Delta t & (-\sin(\psi_{1i})v_i^x - \cos(\psi_{1i})v_i^y)\Delta t \\ -r_1\Delta t & 1 & (\cos(\psi_{1i})v_i^x - \sin(\psi_{1i})v_i^y)\Delta t \\ 0 & 0 & 1 \end{bmatrix},$$

$$B_i = \begin{bmatrix} -1 & 0 & y_{1i} & \mathbf{0}_{3(i-2)} & \cos(\psi_{1i}) & -\sin(\psi_{1i}) & 0 & \mathbf{0}_{3n-3i} \\ 0 & -1 & -x_{1i} & \mathbf{0}_{3(i-2)} & \sin(\psi_{1i}) & \cos(\psi_{1i}) & 0 & \mathbf{0}_{3n-3i} \\ 0 & 0 & -1 & \mathbf{0}_{3(i-2)} & 0 & 0 & 1 & \mathbf{0}_{3n-3i} \end{bmatrix}.$$

The meaning of A_k and B_k is the same as in Section 2.4.2. The measurement values are all pairwise distances in the swarm, in the case of n UAVs, where $n > 3$, the measurement vector becomes as follows, where the value d_{ij} is the distance between agent i and agent j,

$$z = \begin{bmatrix} d_{12} & \dots & d_{1n} & d_{23} & \dots & d_{2n} & \dots & d_{(n-1)n} \end{bmatrix}^T.$$

Let h be a function describing the inter-UAV distances as a function of the state. It is, as mentioned earlier in the report, assumed that the UAVs all have the same altitude. The positional difference along the x axis between UAV i and $j \neq i$, can be expressed as $|x_{1i} - x_{1j}|$. The corresponding goes for the difference along the y axis. Consequently, the distance between UAVs i and $j \neq i$ can be expressed as $\sqrt{(x_{1i} - x_{1j})^2 + (y_{1i} - y_{1j})^2}$, yielding an expression for $h(X)$,

$$h(X) = z = \begin{bmatrix} \sqrt{x_{12}^2 + y_{12}^2} \\ \vdots \\ \sqrt{x_{1n}^2 + y_{1n}^2} \\ \sqrt{(x_{13} - x_{12})^2 + (y_{13} - y_{12})^2} \\ \vdots \\ \sqrt{(x_{1n} - x_{12})^2 + (y_{1n} - y_{12})^2} \\ \vdots \\ \sqrt{(x_{1n} - x_{1(n-1)})^2 + (y_{1n} - y_{1(n-1)})^2} \end{bmatrix}.$$

Let H denote the Jacobian of h. For the rows of z where UAV 1 is part of the distance measurement (the first $n-1$ rows), the rows $H_1, ..., H_{n-1}$ of H become

$$H_i = \begin{bmatrix} \mathbf{0}_{3(i-1)} & x_{1i}/z_i & y_{1i}/z_i & 0 & \mathbf{0}_{3(n-i-1)} \end{bmatrix},$$

and latter rows of H, representing distances between UAV i and j, where $i \neq 1$ and $j > i$, get the following form

$$H_x = \begin{bmatrix} \mathbf{0}^T_{3(i-2)} \\ -(x_{1j} - x_{1i})/z_x \\ -(y_{1j} - y_{1i})/z_x \\ 0 \\ \mathbf{0}^T_{3(j-i-1)} \\ (x_{1j} - x_{1i})/z_x \\ (y_{1j} - y_{1i})/z_x \\ 0 \\ \mathbf{0}^T_{3(n-j)} \end{bmatrix}^T.$$

In the above, x is the index of the row in z that corresponds to the distance between UAV i and j.

The prediction and update equations are the same as the ones used for the filter presented in Section 2.4.2, with the difference that R is now a $\frac{n^2-n}{2} \times \frac{n^2-n}{2}$ diagonal matrix with σ_d^2 on the diagonal and Q a $3n \times 3n$ diagonal matrix with $\begin{bmatrix} \sigma^2_{v_{xy}} & \sigma^2_{v_{xy}} & \sigma^2_{v_r} \end{bmatrix}$ repeated on its diagonal.

3.5 Extended relative positioning system with MDS measurements

Since a relatively large portion of the work related to the thesis focuses on MDS, it was a close-at-hand addition to develop and test a filter that uses coordinates generated with the general MDS procedure as measurement values. All equations and relationships other than those involved in using measurement values are the same for the filter described here as for the extended filter described in Section 3.4.2. For example, the state vector has the same content and the process model as well as the A and B matrices are the same.

Following the steps described in the list at the beginning of Section 2.4.3, relative coordinates for all agents of a swarm, expressed in the HF of agent 1, can be estimated. The MDS positions generated will have two zeros in its top row, implicitly placing agent 1 in the origin of its own HF. Since an MDS formation is subject to ambiguity regarding rotation and mirroring, the measurement coordinates are rotated before being fed to the filter, so that UAV 2 gets the same rotational polar coordinate in the HF of agent 1 as in the state prediction. The formation is also mirrored along the line from the origin of $HF:1$ to the rotated measurement position of UAV 2 if this causes the MDS-position of UAV 3 to get

closer to its position in the predicted state. The positions are then rearranged to make up the measurement data of the filter, which can be expressed as

$$z = \begin{bmatrix} x_{12,MDS} & y_{12,MDS} & \dots & x_{1n,MDS} & y_{1n,MDS} \end{bmatrix}^T.$$

As usual, n denotes the number of UAVs in the swarm. Using this measurement vector, the function that describes the measurement in terms of state variables is simply

$$h(X) = \begin{bmatrix} x_{12} & y_{12} & \dots & x_{1n} & y_{1n} \end{bmatrix}^T.$$

The Jacobian of h w.r.t. the state X, H, becomes a $2(n-1) \times 3(n-1)$ matrix with the following structure. All undescribed entries of H is zero,

$$H^{[2i+1:2i+2,3i+1:3i+3]} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}, i \in [0, ..., n-2] \in \mathbb{N}.$$

With the new definitions of z, h, and H, the same prediction and update equations used for the filter described in Section 3.4.2 are used. The difference is that the R matrix is now $2(n-1) \times 2(n-1)$ and contains the variance of the positional error norm of the MDS measurements, σ_{xy}^2, repeated on its diagonal.

4

Performance analysis for relative localization systems

This chapter covers the results collected from tests of various relative positioning systems. Results from tests of velocity estimation and ranging accuracy for real Crazyflie UAVs are presented in Section 4.2. General settings used for all simulation tests are covered in Section 4.3.

Simulated tests of both convergence times and converged state accuracies are performed. The results of the simulated convergence time tests are presented in Section 4.4. Results from simulation regarding the accuracy of converged state estimates are presented in Section 4.5. The two types of tests were conducted both for the basic filter and extended variants developed throughout the thesis. This was to allow a comparison of the developments to a ”benchmark” system.

Section 4.6 covers the results of a real-world test of the system described in Section 2.4.2 with actual hardware UAVs, as well as the results of a comparative simulation. The last part of the chapter, Section 4.7, summarizes the results of all tests of the positioning systems. It also provides a discussion of the results.

4.1 Simulation environment

Since both the relative positioning and mapping systems described in the report work in two dimensions, the simulation environment described here is also implemented that way. This means that the roll and pitch angles are implicitly set to zero, and the heights of the UAVs are considered constant and identical.

The simulation environment was developed for testing out improved localization methods as well as mapping methods. The system builds upon the motion model described in [21] and utilizes some code already implemented by the authors of that report[1]. Specifically, the function for determining the updated

[1] Source code: https://github.com/shushuai3/multi-robot-localization

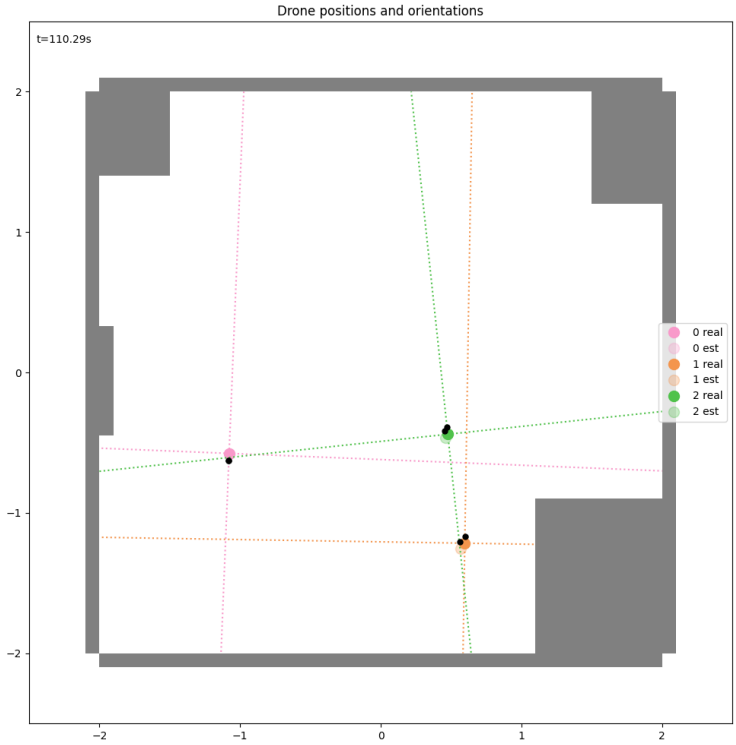


Figure 4.1: *The view during simulation. The black dots describe orientation and the transparent markers are estimates provided by a relative positioning system. Since the relative positioning is done with the pink UAV as the origin-UAV, its pose is set to ground truth in the simulation.*

position of a UAV based on x-, y- and yaw-rotation velocities (motion model), previous pose, and time delta is used, as well as the function for running the positioning filter itself in simulation (the EKF proposed in [21]).

The environment also supports generating simulated LIDAR measurements. These are used when testing out mapping algorithms, together with ground truth poses of the UAVs which are naturally available in simulation. They are also useful when implementing navigation methods capable of avoiding crashing UAVs into walls and other agents. The simulation program generates a live view of the simulation in an animated figure, a snapshot of the animation is presented in Figure 4.1.

4.2 Noise in velocity estimation and UWB ranging

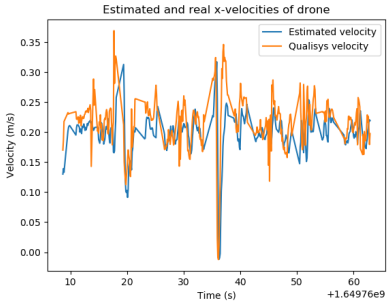
In order for simulation tests of different positioning systems to provide results similar to what can be expected in real-world scenarios, it is important feed real-

istic noise to the systems. Therefore, investigations into the noise characteristics of velocity estimates and ranging measurements were conducted.

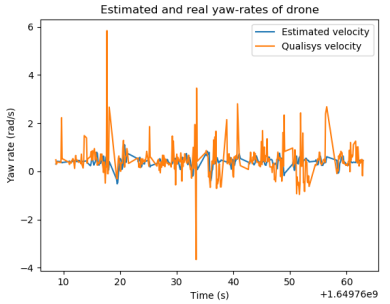
4.2.1 Noise in velocity estimates

Flight tests were performed with real UAVs in an environment where a high-quality motion capture system (Qualisys) was available. Velocity estimates for one UAV expressed in its HF were calculated based on values expressed in its body frame and estimates of pitch and roll. The Qualisys velocity estimates were also transformed into the HF of the UAV, using a rotation matrix and the yaw estimate from the MoCap system. Both the transformed x-velocities are presented in Figure 4.2a. Figure 4.2b shows estimates of the yaw rate of the UAV from both the internal estimator and Qualisys.

Ignoring data from the MoCap system and assuming constant speeds, the standard deviations for the x- and yaw-velocity estimates from the Crazyflie have values $\sigma_{v_x} = 0.03497$ m/s and $\sigma_{v_{yaw}} = 0.1967$ rad/s, respectively. The MoCap data is not used as reference when determining standard deviations because its noise, judging especially from Figure 4.2b, seems to have approximately the same amplitude (or worse) as the noise in the Crazyflie estimates.



(a) Estimates of the x-velocity of a UAV from its own estimator and Qualisys.



(b) Estimates of the yaw rate of a UAV from its own estimator and Qualisys. The Qualisys data contains large erroneous spikes.

Figure 4.2: *Estimated x- and yaw velocities for a UAV.*

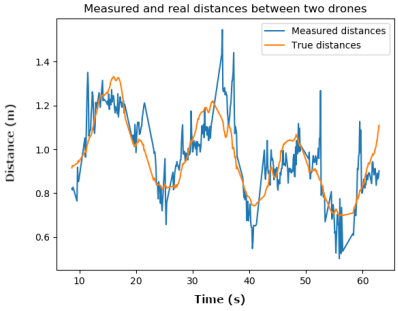
4.2.2 Noise in UWB ranging measurements

The noise in distance measurements was investigated by comparing UWB ranging values with Qualisys data for two pairs of UAVs. Comparisons of measurement and reference values are shown in Figure 4.3. Reference values were subtracted from measured values and a histogram of the errors based on one UAV pair is presented in Figure 4.4. The result is similar to a normal distribution.

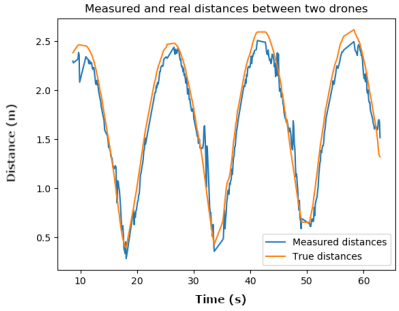
Table 4.1: *Standard deviation in the ranging error based on measurements from two UAV pairs.*

Data from	Standard deviation of the error (m)
Figure 4.3a	0.1100
Figure 4.3b	0.1365

Standard deviations for the distance errors based on the values in Figure 4.3 were calculated, and the results are presented in Table 4.1.



(a) *Measured and true distance between a pair of UAVs.*



(b) *Measured and true distances between another pair of UAVs.*

Figure 4.3: *Measured and real distances between two pairs of UAVs over the course of the same flight. The measured distances in the right plot seem to have a lower noise content, but this is partially due to the different scaling of the y-axis. The span of the distances is larger in the right plot.*

4.3 Initiation, flight method, update frequency and noise levels in simulated tests

Prior to performing simulated tests for the relative localization systems described in this report, a method for initiation of the simulated ground truth poses was chosen. The method decided upon set the global ground truth x and y positions of each UAV in the swarm randomly according to a uniform distribution on the interval $[-2, 2]$ [m], and the global ground truth yaw angle of each UAV was initialized according to a uniform distribution on the interval $[-\pi, \pi]$ [rad]. The rationale behind the method is that in real applications, feasible starting poses can vary a lot. Since simulation is meant to mimic potential real scenarios, random poses are chosen to cover a broad range of possible states. Initial state estimates were assigned differently based on whether convergence time or average filtering error was to be investigated. The methods for the initialization of state estimates

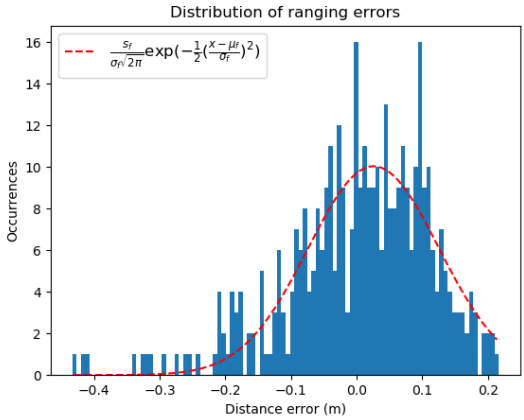


Figure 4.4: *Distribution of the ranging errors shown in Figure 4.3a. The dotted red line is an optimally fitted (using interval midpoints and occurrences) Gaussian with parameters* $s_f = 2.5109$, $\mu_f = 0.02577$ *and* $\sigma_f = 0.09981$.

are covered in the related sections.

In order to allow for a fair comparison of the relative localization systems, a suitable navigation method also had to be chosen. This is because the performance of the filters can be affected by the method with which the UAVs are navigating. By using a navigation method that chooses random velocities, and basing test results on a large number of flights for each positioning system, fair and comparable results can be achieved, while still allowing variations in UAV movements for different test runs. The decided algorithm for navigation is described below.

1. For each UAV, set the x and y velocities randomly according to a uniform distribution on the interval $[-2, 2]$ [m/s]. Set the yaw rate randomly according to a uniform distribution on $[-0.5, 0.5]$ [rad/s].

2. When two seconds have elapsed, reverse the signs of the current velocities. This is to make sure the UAVs stay within a reasonable range of each other, which might be required in real scenarios.

3. When another two seconds have elapsed, restart from the first step.

This navigation method is used for all simulations presented in the report. It does entail a risk of collisions between the UAVs, but minor tweaks to the navigation method to avoid this should not have too much impact on localization performance. When using the MDS-initiation procedure described in Section 3.4.1, the navigation method is used after the first two seconds of a test have passed (the initiation procedure takes two seconds).

A result of the general initialization and navigation method is that the area the UAVs can occupy during a simulated test is bounded by a 12×12 m square.

If UAVs get initialized at the edges of the 2×2 m square mentioned above, and happen to be assigned random velocities of 2 m/s away from the center of the square for 2 s, they reach the bounds. However, it is important to remember that this is an unlikely scenario, and in simulations, the UAVs will probably often keep to a smaller area.

Simulated tests use 100 Hz as the update frequency. Both prediction and measurement steps are conducted at each update. The noise levels used took the results in Section 4.2.1 and Section 4.2.2 into consideration, but were in a conservative manner set a bit higher. The standard deviation of the noise in translational velocity readings, commonly referred to as $\sigma_{v_{xy}}$ in the positioning system descriptions, was set to 0.25 m/s. This was a compromise between the much lower standard deviation observed in the tests of Section 4.2.1 and the fact that velocities change rapidly with the chosen navigation method. The deviation for the noise in yaw rate readings, σ_{v_r}, was set to 0.4 rad/s, following the same reasoning as for the translational velocity noise. The standard deviation of the ranging measurement noise, σ_d, was assigned a value of 0.1 m. For tests of the MDS filter, the standard deviation for the xy-error, σ_{xy}, was set to 0.5 m. This was to allow reasonably quick convergence since the filter often seemed to get stuck in the wrong states when using a smaller deviation.

4.4 Time of convergence in simulation

Real uses of UAVs entail many hardware related issues to take into consideration. Limits on battery life, for instance, can constrain the amount of time a swarm of UAVs is able to stay in the air, and a quick convergence time for positioning filters might be of interest. This section investigates convergence times for the various filters described in the report.

Let convergence for a swarm be defined as the point in time at which all UAVs have had a positional error norm in the HF of the origin-UAV of less than one meter, for a consecutive time period of ten seconds. The time of convergence is then defined as the time point that begins the ten seconds. In the tests presented here, 100 test runs were done for each system, and every test run went on for a maximum of 500 seconds. The total average convergence time is the average of the convergence times from all test runs, not counting the ones that did not reach convergence. It is difficult to say how much more time is needed for convergence in each of those, hence the omittance. If runs needing significantly more time than 500 seconds to converge were accounted for in the mean, it could be highly affected. Therefore, it is important to remember this bias. The amount of runs that did not reach convergence for each test is presented in Table 4.3.

The method used to initialize the filter state in all tests of convergence time was to set it equal to a vector of zeros. The reason was to enable a fair comparison of times needed, without any information whatsoever about the initial UAV poses. Another reason was to minimize the influence of chance. Had some method of random state initialization been used, convergence times could have been unintentionally affected.

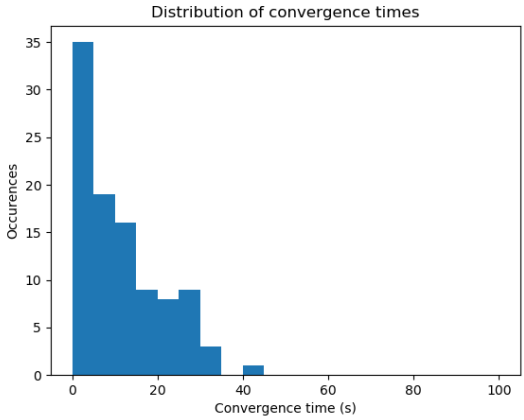


Figure 4.5: *Distribution of convergence times for 100 test runs of the basic system. The test used a swarm consisting of three UAVs.* $\mu_t = 11.35$ *s,* $\sigma_t = 9.605$ *s, all test runs converged.*

4.4.1 Basic system

One test was done for the basic system, described in Section 2.4.2, using a simulated swarm of three UAVs. This is because the basic system is independent of the number of UAVs in the swarm, since it only uses pairwise distances where the origin-UAV constitutes one part. In Figure 4.5, a histogram over the distribution of convergence times is presented. It can be seen that the number of runs belonging to each bin generally decreases with increasing time values, with a majority of the runs converging in the first 20 seconds.

4.4.2 Extended system

The extended filtering system described in Section 3.4.2 was tested for five combinations of swarm UAV count and available measurement values. This was to investigate in what way both UAV and distance count could affect convergence performance. The combinations are listed below.

a) Four UAVs and all pairwise distances 1-2, 1-3, 1-4, 2-3, 2-4, 3-4 (all).

b) Four UAVs and distances between UAVs 1-2, 2-3, 3-4, 1-4.

c) Three UAVs and distances between UAVs 1-2, 1-3, 2-3 (all).

d) Three UAVs and distances between UAVs 1-2, 2-3.

e) Eight UAVs and all pairwise distances.

The results, in order of mention, are presented in Figure 4.6a, Figure 4.6b, Figure 4.6c, Figure 4.6d and Figure 4.6e. As can be seen in the histograms, the distributions generally begin with bins containing many test runs. The number of

runs in each bin then decreases with increasing time values along the horizontal axis. This is to be expected since test runs can sometimes get good initial guesses of the state, at the point of the first measurement.

It is important to keep in mind that not all test runs are represented in the various histograms. The histograms show the runs which converged during the first 100 seconds, and some test runs took longer than that. Some runs did not converge at all, see the captions below the histograms for the number of runs that reached convergence in less than 500 seconds.

4.4.3 Extended system using MDS initialization procedure

For the case of three UAVs and usage of all pairwise distances, the MDS-initiation procedure defined in Section 3.4.1 was also tested. The results are presented in Figure 4.7. The convergence times have clearly decreased compared to those presented for the other systems. A majority of all test runs reach convergence during the first five seconds. To be more precise, 80 out of the total 100 of the runs converged in under five seconds, and 98 converged in under 30 seconds. There were two outliers that needed between 90 and 95 seconds, possibly due to unfavorable initial ground truth poses that made it difficult for the initialization procedure to produce good state estimates.

4.5 Accuracy in simulation

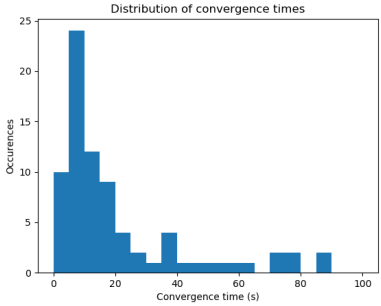
An interesting aspect of the performance of a relative localization system, besides its convergence time, is the accuracy. Therefore, tests were conducted to assess the accuracies of the positioning filters presented in the report. The accuracy of a filter is preferably measured after convergence and needs a definition, the one used to generate positioning accuracy results in this report follows here.

Accuracy means the average positional error norm for one specific UAV in a swarm, in the HF of the origin-UAV, across all test runs. In the tests presented here, 100 test runs were done for each system, and every test run went on for 200 seconds. However, in the error plots, results from only five test runs are presented to avoid indistinguishability among the lines. Furthermore, plots that are both zoomed in along the horizontal axis and plots displaying errors across a longer time horizon are provided for each test presented. This is to display both short-term dynamics in the errors as well as indications on for example how high the errors can reach in worst cases.

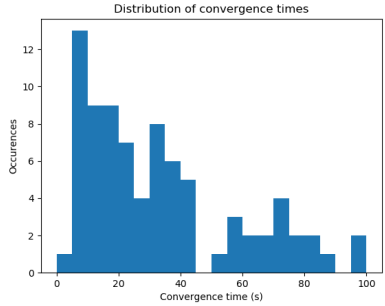
The filter state initialization method for tests of accuracies is based on placing estimates close to their ground truth values. More precisely, relative x, y, and yaw estimates (expressed in the HF of the origin-UAV) are initialized with true values added with Gaussian noise of zero mean and 0.2 [m, rad] in standard deviation.

4.5.1 Basic system

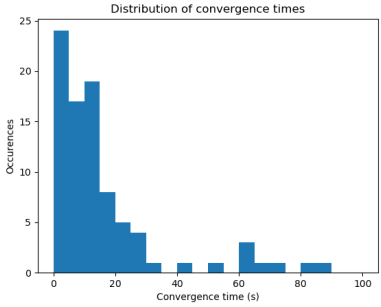
The accuracy of the basic system was investigated in one test, to generate comparative values for the extended system. Figure 4.8a shows the errors of UAV 2



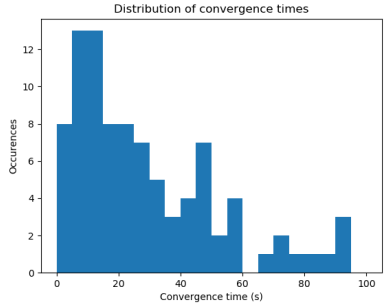
(a) *4 UAVs and all pairwise distances. $\mu_t = 48.42$ s, $\sigma_t = 84.29$ s, 88 out of 100 test runs converged.*



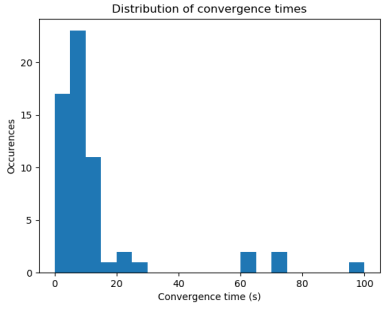
(b) *4 UAVs and distances 1-2, 2-3, 3-4, 1-4. $\mu_t = 64.52$ s, $\sigma_t = 85.72$ s, 97 out of 100 test runs converged.*



(c) *3 UAVs and all pairwise distances. $\mu_t = 30.21$ s, $\sigma_t = 63.88$ s, 93 out of 100 test runs converged.*



(d) *3 UAVs and distances 1-2, 2-3. $\mu_t = 41.60$ s, $\sigma_t = 50.38$ s, all test runs converged.*



(e) *8 UAVs and all pairwise distances. $\mu_t = 29.02$ s, $\sigma_t = 65.10$ s, 64 out of 100 test runs converged.*

Figure 4.6: *Convergence time distributions from the tests of the extended system, described in Section 3.4.2. Note that the means and standard deviations are based only on the test runs that did converge, see Table 4.3 for how many runs converged in each case.*

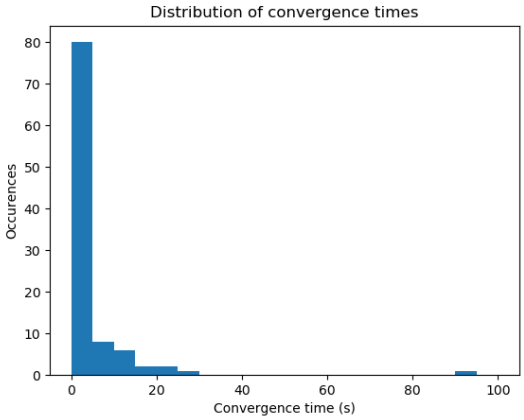


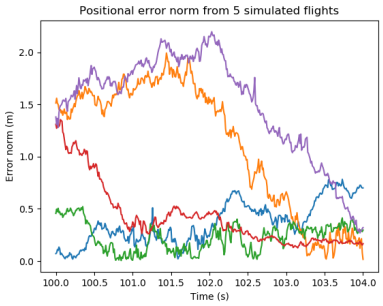
Figure 4.7: *Distribution of convergence times for the extended system described in Section 3.4.2, using the MDS initialization procedure described in Section 3.4.1. The swarm was made up of three UAVs and all pairwise distances were available. All 100 test runs converged, and the average time and standard deviation for convergence were $\mu_t = 4.868$ s, $\sigma_t = 10.07$ s, and all test runs converged.*

in a swarm of 4 UAVs for five runs during four seconds. Figure 4.8b shows the error of UAV 2 for five test runs over 100 seconds. As can be seen in both plots, the positional error varies in relatively large intervals. Usually, the errors keep at low values, like the green line in Figure 4.8a, but position estimates can get much larger errors, like the orange line in Figure 4.8b at about 40 seconds.

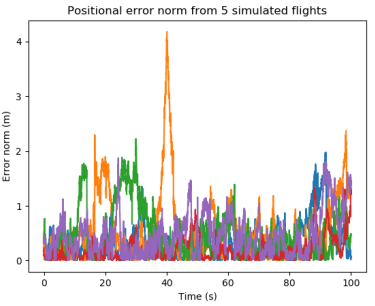
4.5.2 Extended system

Accuracy results for the extended system are based on seven tests where different combinations of swarm UAV count and available pairwise distances were tried. In contrast to the single accuracy test of the basic system, this is an increase. The reason behind the enlarged test set is that the accuracy of the extended system can be expected to vary with UAV count, which is not true for the basic system.

Two tests were conducted for a swarm of four UAVs. Namely, a test with four UAVs and all pairwise distances, and a test with four UAVs and distances 1-2, 2-3, 3-4, and 1-4. Five tests were conducted for a swarm of eight UAVs. Namely, a test with eight UAVs and all pairwise distances, a test with eight UAVs and distances 1-2, 2-3, 3-4, 4-5, 5-6, 6-7, 7-8, and 1-8, a test with pairwise distances according to Figure 4.10a, a test with eight UAVs and distances according to Figure 4.10b, and a test with eight UAVs and randomized pairwise distances. Randomized pairwise distances mean all possible pairwise distances were used, but for each measurement step of the filtering process, each distance was included with a 50 % probability.



(a) Result of five test runs during four seconds.



(b) Result of five test runs over 100 seconds.

Figure 4.8: *The positional error of UAV 2 in the swarm when using the basic relative positioning system on a swarm of four UAVs.*

Testing with as many as eight UAVs allowed for much variation in available distance count. This allowed for a deep investigation of how that affected performance. Also, in the case of asymmetric distance availability (compare e.g. UAV 2 and UAV 8 in Figure 4.10a), accuracies of different UAVs within the same swarm could be compared. Randomized pairwise distances were used to mimic a potential real-world scenario with defective inter-drone communication. The test using randomization was meant to investigate the performance degradation concerning the accuracy in such a situation.

In the tests with a swarm of four UAVs, average errors were calculated for UAV 2. For the case of eight UAVs and distances 1-2, 2-3, 3-4, 4-5, 5-6, 6-7, 7-8, and 1-8, average errors were calculated for UAVs 2 and 5. For the case of eight UAVs and distances according to Figure 4.10a, average errors were calculated for UAVs 2 and 8. In the same way, average errors for the tests with eight UAVs and distances according to Figure 4.10b were calculated for both UAVs 2 and 8.

The accuracy results are presented in Table 4.3. Figure 4.11 and Figure 4.12 show the errors of UAV 2 from five test runs during four seconds, for each respective combination of UAV count and available pairwise distances. The reason why only five test runs are shown in the plots is that displaying more, or even all 100, runs would render the lines more or less indistinguishable. Figure 4.13 and Figure 4.14 show the errors of UAV 2 from five test runs over 100 seconds, for each respective combination of UAV count and available pairwise distances.

In the same way as in the plots depicting error values for the basic system, the errors in the different tests vary greatly. It can be seen, however, that the error values shown in Figure 4.12a and Figure 4.12e seem to generally stay lower than those shown in the other plots of Figure 4.11 and Figure 4.12. This observation is confirmed by the average error values stated in Table 4.3.

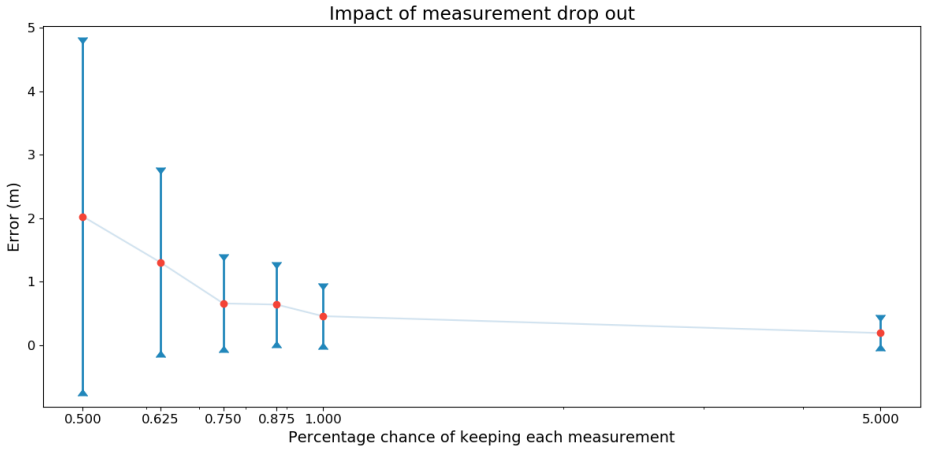


Figure 4.9: *The impact of losing potential pairwise distance measurements. The tests utilized a swarm of eight UAVs. Standard deviations associated with each average error are depicted by vertical bars. The x-axis is logarithmically scaled.*

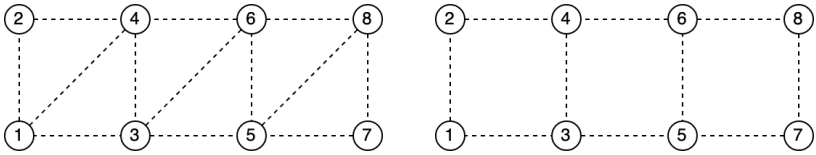
4.5.3 Measurement dropout analysis for the extended system

Since the test described in Section 4.5.2 where pairwise distance measurements are included with a probability of 50 percent did not affect accuracies much (see Table 4.3), this section presents a further analysis of the impact on accuracy of measurement dropout. Six additional tests were conducted, using the same setup as for the test with 50 percent measurement dropout risk, but changing the probability of including each possible measurement to 5, 1, 0.875, 0.75, 0.625, 0.5 percent, respectively, and only doing ten 200 second runs for each test.

The average and standard deviation of the positional error of UAV 2 were calculated for each new test, and the results are depicted in Figure 4.9, as well as Table 4.2. As can be seen both in the plot and the table, dropout effects are not appearing clearly until the probability of including measurements is very low. At a five percent chance of including each measurement, the accuracy is comparable to the accuracy in tests with no dropout risk. At 0.5 percent chance of including measurements, the average error is around two meters, which means the filter does not output much valuable information. Since the UAVs keep to a 12×12 m square during simulation (and probably stay closer to each other than that most of the time), the average distance between UAVs could very well be around 2 m.

4.5.4 System based on filter using MDS measurements

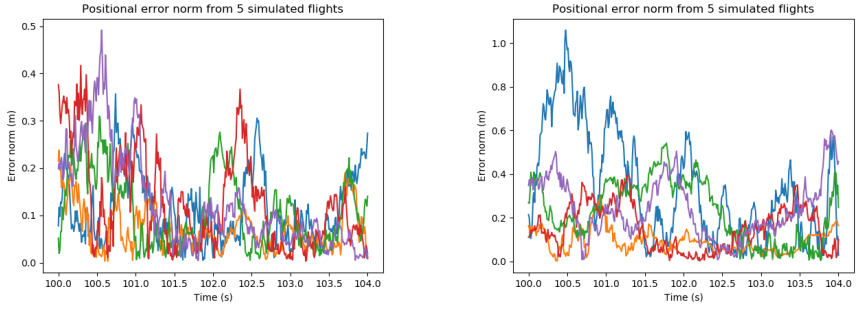
One test was performed for the system using MDS measurements, utilizing four UAVs. The main reason as to why only one test was conducted was the low accuracy. The accuracy was calculated based on the errors of UAV 2, and can be seen



(a) *A combination of UAV count and pairwise distances tested with the extended positioning system.*

(b) *Another combination of UAV count and pairwise distances tested with the extended positioning system.*

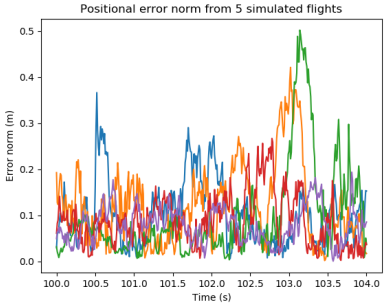
Figure 4.10: *Two combinations of available distances tested with the extended positioning system.*



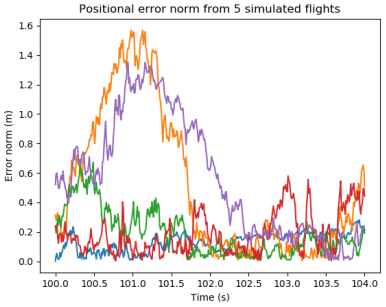
(a) *All pairwise distances were used.*

(b) *Distances 1-2, 2-3, 3-4 and 1-4 were used.*

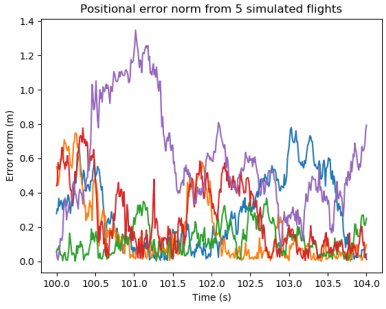
Figure 4.11: *Positional error norm for UAV 2 using the extended positioning system, based on two tests with a swarm of four UAVs. The result of five test runs during four seconds are shown in each plot.*



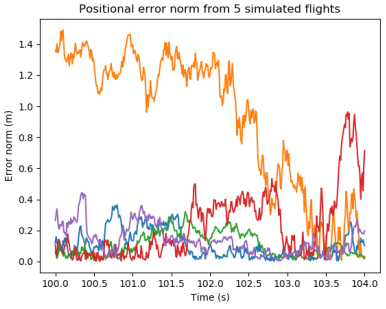
(a) *All pairwise distances were used.*



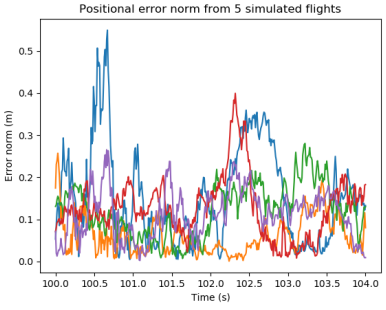
(b) *Distances 1-2. 2-3, 3-4, 4-5, 5-6, 6-7, 7-8, and 1-8 were used.*



(c) *Distances according to Figure 4.10a were used.*



(d) *Distances according to Figure 4.10b were used.*

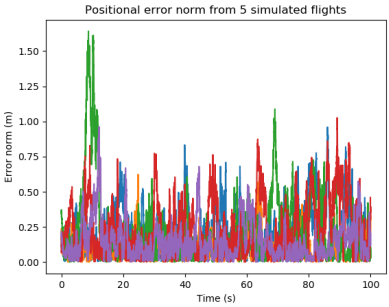


(e) *Randomized pairwise distances were used.*

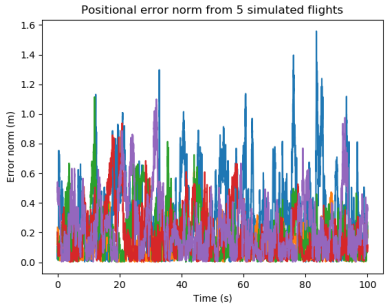
Figure 4.12: *Positional error norm for UAV 2 during four seconds using the extended positioning system, from five tests with a swarm of eight UAVs.*

Table 4.2: *Averages and standard deviations of the errors from the dropout tests described in Section 4.5.3. The values in the first column denote probabilities of keeping each measurement.*

P (%)	Average error (m)	Standard deviation of errors (m)
5	0.1892	0.1864
1	0.4555	0.4171
0.875	0.6379	0.5803
0.75	0.6561	0.6768
0.625	1.306	1.402
0.5	2.027	2.724



(a) *All pairwise distances were used.*



(b) *Distances 1-2, 2-3, 3-4 and 1-4 were used.*

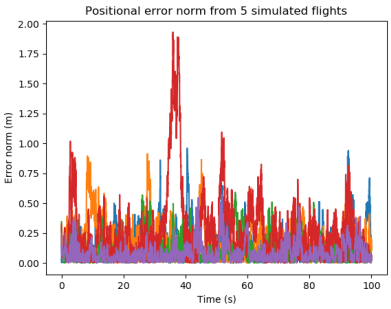
Figure 4.13: *Positional error norm for UAV 2 using the extended positioning system, based on two tests with a swarm of four UAVs. The results of five test runs over 100 seconds are shown in each plot.*

in Table 4.3. Clearly, the error values are not nearly as good as those for the other systems. Potential reasons for the low performance are discussed in Section 4.7.

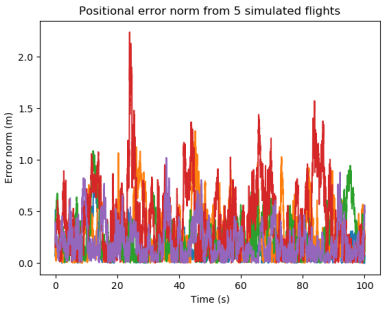
4.6 Real-world tests of the basic system

Tests were conducted for the basic system on a swarm of three actual UAVs. The navigation method for the real-world tests was based on the agents moving in circles with diameters of 1 m and always heading in the x-directions of their respective HF. No filtering was conducted on the UAV hardware, but instead, they logged their velocities and range measurements to a central computer. This logging was done with a frequency of 10 Hz, due to limits on the computational capacities of the agents. The relative positions and yaw values of the UAVs were initialized with their true values before starting the filtering.

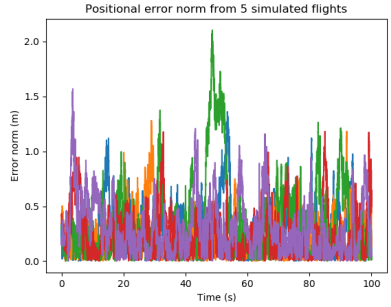
An external MoCap system was run simultaneously to record global ground



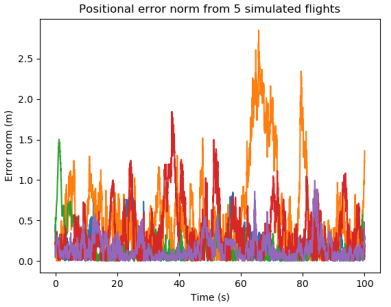
(a) *All pairwise distances were used.*



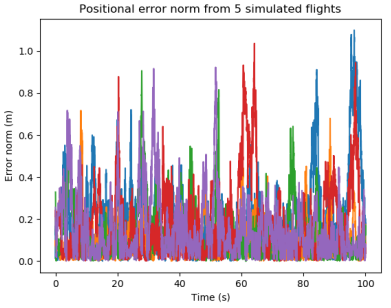
(b) *Distances 1-2. 2-3, 3-4, 4-5, 5-6, 6-7, 7-8, and 1-8 were used.*



(c) *Distances according to Figure 4.10a were used.*



(d) *Distances according to Figure 4.10b were used.*



(e) *Randomized pairwise distances were used.*

Figure 4.14: *Positional error norm for UAV 2 over 100 seconds using the extended positioning system, from five tests with a swarm of eight UAVs.*

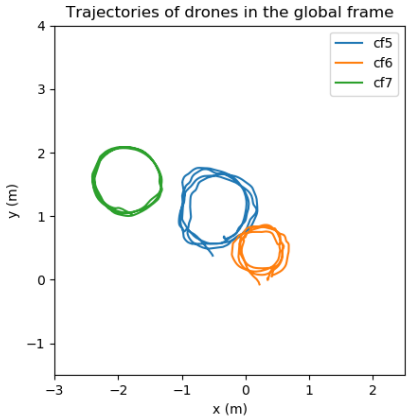


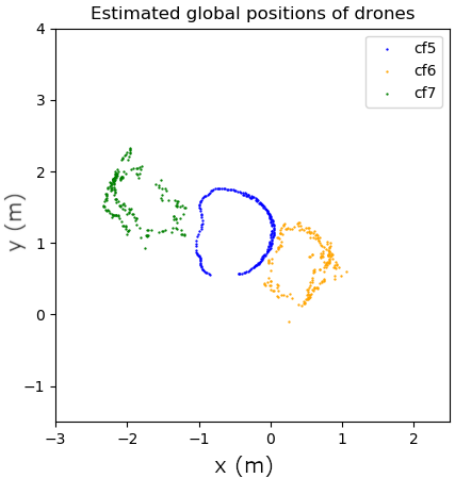
Figure 4.15: *Global ground-truth coordinates of the UAVs during the real-world flight described in Section 4.6.*

truth coordinates and yaw values, to allow subsequent evaluation of the performance of the relative localization system. A plot of the global ground truth coordinates from the test is presented in Figure 4.15. Figure 4.16a presents the result of running the basic filter offline for the ”cf5” UAV with the collected velocity and ranging data. The plot shows estimated states which are transformed into coordinates expressed in the global frame.

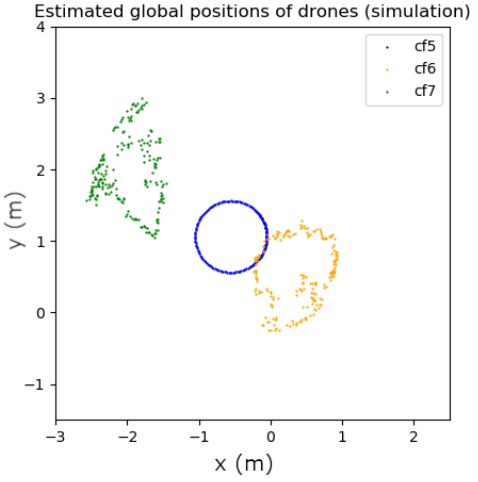
Since the results of the real-world test was not as good as hoped, a simulated version of the test was conducted to validate that it was the low update frequency that was causing issues. Figure 4.16b shows the results of a simulation that was made to mimic the real-world flight. In this simulation, the UAVs were assigned the same initial poses and navigation method as in the real test. The estimates in the simulation were much more accurate when performing the exact same test but instead using a 100 Hz update frequency.

4.7 Summary and discussion

The convergence times and converged state accuracy values from the simulation are summarized in Table 4.3. A star next to the convergence time values indicates all test runs did not converge. The number in the parenthesis then shows how many runs did converge. The maximum number of test runs that possibly could converge for each test is 100 since this is the number of test runs for each combination of the positioning system, UAV count, and available distances. In the fields of the column ”System & distances”, the first number in the parenthesis indicates how many UAVs were used in the swarm. The potential second number in the same parenthesis indicates which UAV’s average error is listed. In the rows of the column that do not include either the word ”Basic” or ”MDS filter”, the extended system was used to generate the results. ”Circle” means that



(a) *Relative positions transformed to coordinates in the global frame. The basic filter was used to generate the estimates, with velocities and ranging measurements from real UAVs. Initial states were set to their respective true values.*



(b) *Results of a simulated test with the same initial positions, yaw angles, and navigation method used in the real-world test described in Section 4.6. The simulation was run with a 10 Hz update frequency. Initial relative states were set to their true values. The basic filter was used in the simulation.*

Figure 4.16: *Running the basic positioning system on real and simulated velocities and ranging measurements. The filtering was conducted for 20 s in each of the two tests.*

pairwise distances forming a closed loop in the swarm were used. In the case of four agents, it means the distances 1-2, 2-3, 3-4, and 1-4, and in the case of eight agents the distances 1-2, 2-3, 3-4, 4-5, 5-6, 6-7, 7-8 and 1-8.

Something that can be noted from the convergence time data presented in Table 4.3, is that the convergence time seems to increase with more available distances. Comparing the average convergence time of the basic system, 11.35, with that of the extended system using three UAVs, 30.21, they are separated by a factor of almost three. When using the extended system with all distances on a swarm of four UAVs, the convergence time is even larger. This is to be expected since a swarm of four UAVs requires one more UAV to keep an error of less than a meter for ten seconds to be considered converged. Still, the number 48.42 is considerably higher than 30.21. The number of test runs that converge when using the extended system with all pairwise distances is decreasing with increasing UAV count. For eight agents and all distances, only 64 of 100 test runs reach convergence. This is causing the average convergence time for that test to be misleading. If the 46 test runs that did not converge would have taken much longer than 500 seconds, this would have pulled the average value (and probably also the standard deviation, considering data from other tests) up by quite a lot. The reason why the value 29.02, is so low, is probably because the test runs that converged did so by getting an advantageous state estimate after the first measurement of the filter. This same reasoning goes for tests of convergence time in other setups as well.

An interesting thing that can be highlighted is that the fraction of test runs that converge generally increases when removing pairwise distances. When using the extended system on a swarm of three agents, but with distances 1-2, 2-3 (a line of distances from the origin-UAV to the last), all test runs converge. When using a swarm of four agents and the distances 1-2, 2-3, 3-4, and 1-4, 97, which is almost all runs, reach convergence. In comparison with the corresponding test where all distances are used, this is an improvement. An explanation for this is that removing distance measurements release constraints on the estimated formation. States that are rotated around the origin-UAV far off from their true values by the first measurement are intuitively harder for the filter to correct when dealing with a ”stiffer” formation. That the average convergence times seem to increase with removed pairwise distances when using the extended system could have to do with the reasons discussed previously, if more test runs converge, possibly after close to 500 seconds, this increases the average required time.

One more thing to consider when it comes to the convergence time data is that the standard deviations are quite large. Often, they are reasonably higher than the averages. It is at the same time natural for the standard deviations to be large, given the test setups. It is hard to think of a solution that enables a fair comparison of convergence times between the different systems and also decreases the deviation. Letting the tests run for more than 500 seconds, and allowing all test runs to converge would generate less ambiguous average convergence time values, but would probably increase the deviations. Additionally, there are practical dilemmas with letting test runs run indefinitely since this implies very time-demanding simulations.

When using the MDS initialization procedure described in Section 3.4.1, the convergence time is greatly decreased. The extension steps to the standard MDS procedure, sorting out ambiguities, naturally contribute a lot to this result. The decrease would be seen also if testing with more agents since there are no practical dependencies in the steps of the initialization method on the UAV count. When using the extended filter in a real setting with real hardware UAVs, a low convergence time could be essential. The battery life of a Crazyflie UAV is limited and if the mapping is to be performed utilizing relative poses, it would probably be preferable to have realistic relative poses available as quickly as possible.

Results of converged state accuracy are also presented in Table 4.3. First off, it can be seen that using all pairwise distances in a swarm of four UAVs leads to lower average errors than when removing some distances and only using 1-2, 2-3, 3-4, and 1-4 (”Circle”). Still, the results are better than for the basic system. The lowest average errors are achieved when using all pairwise distances in a swarm of eight UAVs. This follows the intuition that more distance information makes for better accuracy. When more measurements are available related to a quantity, the error of each measurement practically averages and the deviation essentially becomes smaller. The more measurements to average, the better accuracy emerges. Following this reasoning, it could be stated that relative positioning systems as the extended examined in this report benefit from working on larger swarms. Larger swarms imply more possible pairwise distances, following the expression $(n^2 - n) / 2$. However, in real-world applications with hardware UAVs, working with more agents can lead to practical troubles. Using the ”Circle” distance combination for a swarm of eight agents worsens the performance compared to when all distances are used. The average error is worse than when using four UAVs and ”Circle” distances. This indicates that swarms of more agents can instead worsen performance compared to swarms of fewer when all pairwise distances are not available.

The test of converged state accuracy where the randomized distances were used, indicates a slight increase in average error compared to when all distances are available. The subtility of the increase could have to do with the distances being randomized in each measurement step of the filter, making each distance probable to occur frequently enough for the filter to make good use of it. It is interesting that this phenomenon persists all the way down to where the probability of including each measurement reaches only around five percent. No substantial drops in accuracy are seen before then. A test where the availability of each distance would be selected randomly every, say, 100 update steps, would potentially show a larger decrease in performance. Such a method is arguably more realistic since real UAVs flying in a space composited of multiple rooms might be prone to lose ranging abilities between each other for longer intervals than one iteration.

Considering the figures showing errors during four seconds, it can be seen that when the extended system uses all pairwise distances, the error lines get less smooth. Comparing for example Figure 4.12a and Figure 4.12b, the lines in the first contain more and larger spikes. The extended positioning system is apparently quicker at correcting errors when more distances are available, leading to spiky formations. This effect can intuitively be understood by imagining a

swarm not using all possible distance measurements. In that kind of setup, the formation can be skewed to a larger degree (increasing errors) and might stay in the skewed state longer (fewer spiky errors), in comparison with a setup using all possible distances. Like in the convergence tests, the standard deviations are high also for the converged state error values.

The performance of the MDS filter is clearly not as good as that of the other systems. This can be deduced by looking at the accuracy value in Table 4.3. As stated in Section 3.5, the MDS coordinates are rotated and flipped in a specific way to fit well with the current (predicted) states. If there is much noise in the velocity estimates yielding the state prediction, the subsequent measurement update can serve to validate these errors rather than correct them. One potential fix for this issue might be a solution where the transformation of the MDS coordinates takes the positions of all agents in the swarm into consideration when resolving ambiguity.

Looking at the real-world tests in Section 4.6 reveals that the basic system performs poorly when running it with a low update frequency (10 Hz in the case of the tests). Tests suggested that this bad performance could be connected to the motion model being used, which is presented in Section 2.4.1 and comes into play in all filters covered in this report. A more accurate higher-order discretization of the continuous motion model could be beneficial in order for the filtering to work better when using lower iteration frequencies.

Table 4.3: *Summary of results on convergence time and average error for the different relative positioning systems. An asterisk indicates that not all runs reached convergence during a test. Numbers in parenthesis next to asterisks display how many runs reached convergence (out of 100 potential).*

Convergence time		
System & distances	**Average (s)**	**Standard deviation (s)**
Basic (3)	11.35	9.605
All pairwise (3)	30.21* (93)	63.88* (93)
1-2, 2-3 (3)	41.60	50.38
All pairwise (4)	48.42* (88)	84.29* (88)
1-2, 2-3, 3-4, 1-4 (4)	64.52* (97)	85.72* (97)
All pairwise (8)	29.02* (64)	65.10* (64)
MDS initialization (3)	4.868	10.07
Converged state error		
System & setup	**Average (m)**	**Standard deviation (m)**
Basic (4)	0.3105	0.3418
All pairwise (8, 2)	0.1639	0.1736
Circle (8, 2)	0.2238	0.2501
Randomized (8, 2)	0.1687	0.1718
Figure 4.10a (8, 2)	0.1886	0.1950
Figure 4.10a (8, 8)	0.1989	0.1970
Figure 4.10b (8, 2)	0.2175	0.2336
Figure 4.10b (8, 8)	0.2509	0.2936
All pairwise (4, 2)	0.1777	0.1823
Circle (4, 2)	0.2181	0.2253
MDS filter (4, 2)	4.853	4.519

5

Performance analysis for the mapping system

This chapter covers the results collected from tests of the mapping system. A simulation study is performed that evaluates the different methods for merging maps and discusses how the mapper can handle noisy sensor data. The evaluation is verified by real-world tests.

5.1 Simulation study

This section will provide evaluations of how the described methods for grid mapping influence the result of the constructed map. In Section 5.1.1 the introduced map cost function (2.52) is evaluated. In Section 5.1.2 the cost function is used to evaluate the strategy used for multimap merging, and how the order in which the maps are merged affects the final map. Strategies to handle noisy measurements are evaluated in Section 5.1.3, together with some results on how a merged map constructed offline (after flight) differs from a map constructed collaboratively online (during flight). Lastly, relative positions are used to construct a map in Section 5.1.4.

5.1.1 Map cost function

The map cost functions (2.51), (2.52), and (3.1) were reviewed to investigate their properties. To test if these functions all are able to find the ideal transformation matrix, and to compare the accuracy measurements, a simulation involving three UAVs flying for 2 minutes was conducted. In each iteration, the UAVs explored the map scenarios given in Figures 3.4a and 3.4b. For each iteration, the final map was compared with the true map to calculate the accuracy. In total, ten iterations were made, with individual offset and angle adjustments. The results are seen in Table 5.1. To get a percentage, and not a score for (2.51), w is divided by n, where

n is the total number of cells. The area of the unexplored space is doubled by increasing the size of the grid at initialization. The area of the free space differs only from the mapping scenario. The table shows how the different cost functions scale depending on the properties of the map. The classification measures, (2.52) and (3.1) are both static when the area of unknown space is doubled, but (2.51) increases. The performance difference between mapping scenarios scales with the area of free space for (2.52), and with the length of walls for (3.1). Observe that the classification is independent of the mapping method, and that is not the case for the equality function. For the rest of the thesis, the cost function considering only the position of the walls is used.

Table 5.1: *Map comparison between achieved online- and offline-created maps to the exact map for different cost functions. The numbers are given in percentages. Numbers given in parenthesis are variances in percentage points.*

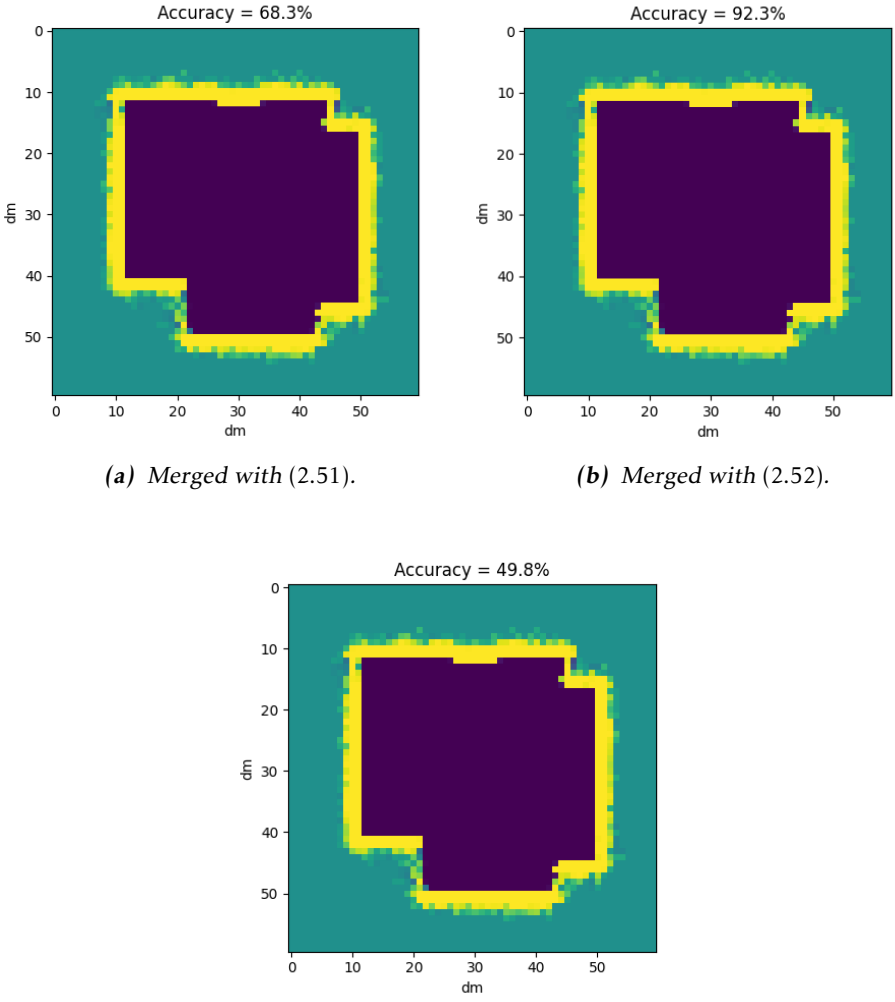
Mapping	(2.51)	(2.52)	(3.1)
Map scenario 1			
Offline	68.7 (0.46)	92.3 (0.02)	49.6 (0.56)
Online	75.5 (1.16)	92.5 (0.03)	52.2 (0.28)
Map scenario 2			
Offline	74.1 (0.53)	83.1 (0.48)	42.1 (0.86)
Online	83.1 (0.91)	83.2 (0.06)	42.8 (0.73)

Double the area of unknown space			
Mapping	(2.51)	(2.52)	(3.1)
Map scenario 1			
Offline	82.5 (0.16)	92.5 (0.01)	49.2 (0.47)
Online	86.2 (0.37)	92.5 (0.03)	52.2 (0.28)
Map scenario 2			
Offline	85.5 (0.17)	82.7 (0.44)	41.9 (0.45)
Online	90.5 (0.29)	83.2 (0.06)	42.8 (0.73)

For the maps explored here, all cost functions work and create similar maps. Figure 5.1 illustrates the offline constructed maps that the different cost functions construct for map scenario one. For online constructed maps the cost function does only express the similarity to the true map and does not affect the constructed map.

5.1.2 Map merging

For the first test, three UAVs are in the simulation environment, with exact positions and ideal sensor measurements. When maps are merged offline, the order in which they are merged matters. That is, with three maps, the final map differs depending on the pairwise order in which the maps are merged. In general, $(m_1 \oplus m_2) \oplus m_3 = m_{12} \oplus m_3 = m_{123} \neq m_{132} = m_{13} \oplus m_2 = (m_1 \oplus m_3) \oplus m_2$, but note that $m_1 \oplus m_2 = m_{12} = m_{21} = m_2 \oplus m_1$. A demonstration of this is shown



(a) *Merged with* (2.51).

(b) *Merged with* (2.52).

(c) *Merged with* (3.1).

Figure 5.1: *Maps generated by three UAVs for map scenario 1. The maps are merged with different cost functions. The accuracy is expressed as the accuracy towards the true map calculated with the respective cost function.*

in Figure 5.3, where the maps along each row are identical, but there are differences along the columns. The difference is not large, but it is noticeable. The red-circled areas are regions where the difference is more noticeable. The individual maps are displayed in Figure 5.2. The focus is not to achieve the best map, but to emphasize the difference in merging order.

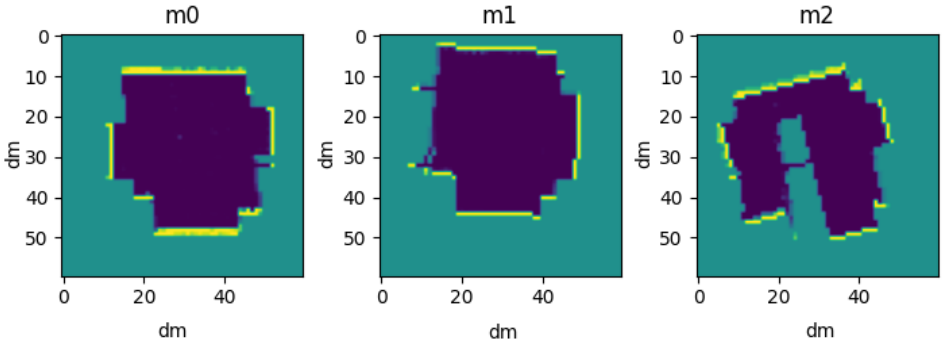


Figure 5.2: *The individual maps merge in different orders. UAV 0, responsible for map 0 has been initialized such that the map it generates is not rotated nor translated compared with the true map, while the other UAVs are initialized with random rotations and translations.*

5.1.3 Noisy measurements and position uncertainty

In a more realistic simulation, white Gaussian noise is added to the odometry data and to the range readings. The standard deviation is 1 dm for the position, 0.1 rad for the yaw angle, and 2 mm for the range readings. The imposed disturbance is realistic when the Loco positioning system is used. The UAVs still share a coordinate system. The mapping algorithm considers the uncertainty in position as described in Section 2.5.4.

As the map is two-dimensional, the altitude is ignored when sampling poses to handle uncertainty in the position. As $\theta_{cf} = \phi_{cf} = 0$ in the two-dimensional simulations, these parameters are ignored as well. This can also be said for real flights as the mapping surrounding is built of cardboard boxes with limited height, and measurements with larger angles are ignored as the LIDAR can detect false obstacles over the boxes. The pose variables to sample are then $x_{cf}, y_{cf}, \psi_{cf}$. When sampling positions (x, y) and yaw angles ψ, the standard deviations for these parameters correspond to their respective standard deviations in estimation. The number of sampled points n linearly increases the time it takes to finish the map, and can therefore not be too large when mapping in flight. For simulated and offline data, a larger number of sampled points can be tested.

The standard deviation for the LIDARs is, as seen in Figure 3.2b, 2 mm. It can therefore be argued that it is unnecessary to use a Gaussian sensor model and that the ideal inverse model can be used.

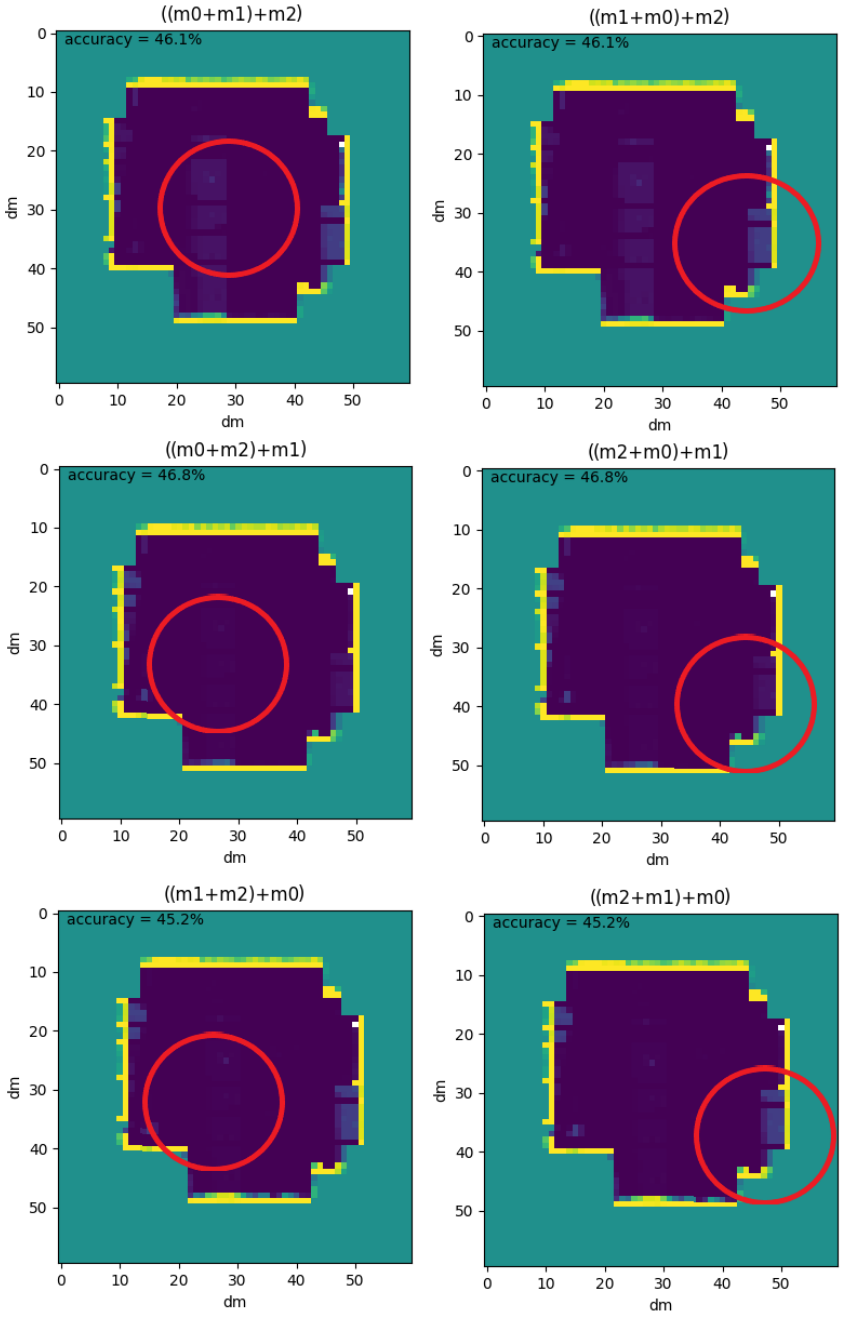


Figure 5.3: *Maps constructed by three drones, merged in different orders. The final maps are identical along the rows but differ across the columns. The accuracy measures wall placement. The red circles emphasize areas where the difference is more noticeable.*

Table 5.2 reports the impact of the number of samples. The estimated pose is held constant at $x_t = 2, y_t = 5, \psi_t = 0.4$. The time complexity scales linearly, as it should. Since the simulator and the Crazyflies have a sampling frequency of 100 Hz, a value of less than 100 iterations/s would not work for online mapping. According to Table 5.2, this limit is about 20 samples. At five samples, the standard deviation is about 1/4 of the error in the Loco estimation. At 100 samples, the standard error has dropped to below one centimeter, which would be an acceptable size. Based on the distribution of positions and yaws shown in Figure 5.4, 20 samples are quite a few to obtain a Gaussian distribution and correct weights. This is seen by comparing Figures 5.5b and 5.5d with Figures 5.7a and 5.7b. Using noisy measurements in the simulator, three UAVs fly for 1 minute. The data is saved in order to construct both an online map and an offline merged map. Maps constructed with 20 samples is illustrated in Figures 5.5 and 5.6. The online and offline created maps display both mapping scenarios with, and without, considering an uncertain position in the mapping algorithm. Tests involving a higher number of samples for maps have been performed, but the visual result does not differ much from the results with 20 samples. In Figure 5.7 the mapper has processed 100 samples. The time consumption, on the other hand, increases, making the extra number of samples unnecessary. The improvement when considering the uncertainty in the pose estimation is seen along each wall and at the corners. The major shared improvement is seen by comparing the top right corner. The positions of the walls and corners are not altered but are more prominent.

Table 5.2: *Account for how the number of sampled points affects the time complexity and distribution. Iterations/s describe how many measurements the mapping algorithm processes on average each second. For the standard deviation of the position, σ_{pos}, the Euclidean distance error is used. σ_ψ is the standard deviation for the yaw-angle. All calculations are made for* 10000 *samples. For $n = 1$, the original pose is used.*

Samples n	Iterations/s	σ_{pos} [m]	σ_ψ [rad]
1	2275	0	0
5	457	0.029	0.027
10	225	0.021	0.019
20	109	0.014	0.013
100	21	0.007	0.006
1000	2	0.002	0.002

5.1.4 Mapping with relative positions

If the mapping is to take place live, the global coordinate system for each UAV must coincide. With the relative positioning system described in Section 2.4.2, the global poses can be estimated from a reference drone, if the reference drone has a known global pose. This pose is in the simulator given from the motion model. For now, it is assumed that the task of knowing the global position of the

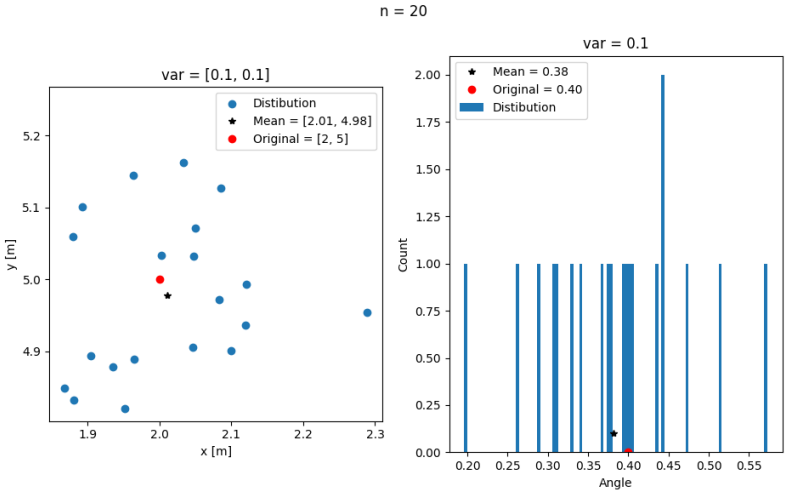
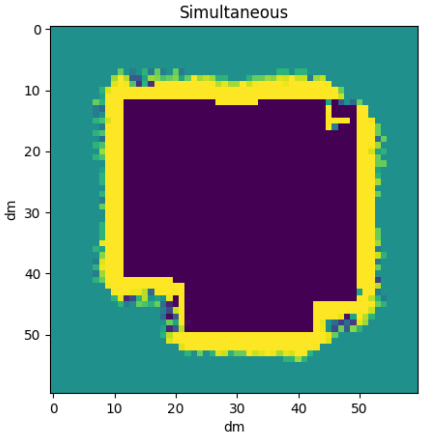


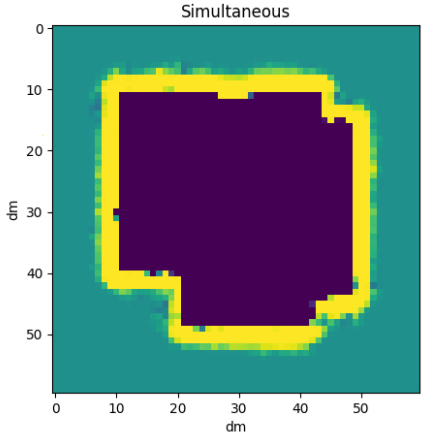
Figure 5.4: *An example of how the position and yaw angles are distributed when sampling 20 poses. The left image shows the x and y position of each sample, the mean position for all sampled points, and the original position. The right image shows the distribution for sampled yaw angles.*

UAV is solved. It is also assumed that the transformations between the UAVS IHF are known. In the simulator, one UAV has the task of estimating the positions of the other UAVs. White noise is added to the position of each UAV, which will propagate to the estimated positions of the other UAVs. The mapper sample 20 poses to handle uncertainty in the position. The used standard deviation corresponds to the results achieved with the basic filter in Table 4.3 for the position, and 0.1 rad for the angle. In Figure 5.8a the map is seen. In Figure 5.8c the resolution of the map is increased to centimeter precision but the identification of walls is worsened. Each pixel in the grid maintains one square centimeter in reality, and the thickness of the walls would ideally be of this size. Such a wall is hard to see, even with perfect sensors and an ideal inverse sensor model.

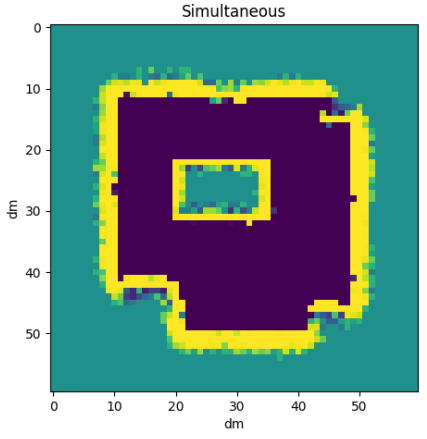
A suggestion to handle the increased resolution and still distinguish the walls is to change the inverse sensor model so that the range reading extends w meters from the read distance. This modification will for a sensor reading r say that the space $0 \rightarrow r$ is free, $r \rightarrow r+w$ is occupied and $r+w \rightarrow \infty$ is unknown. The width w should be smaller than the minimum width of captured obstacles on the map. In the maps used, the maximal width would be 0.78 m, corresponding to the shortest distance between two edges on the map. The modification is implemented by multiplying the occupied space, $1 * R_{map}$ to $(1+w) * R_{map}$, with the sensor reading s_r. Note that this scales $\mathcal{O}(w * R_{map})$ in complexity. In previously displayed maps, a wall width of 1 dm is enough for the walls to be seen clearly. The mapper was tested with $w = 0.1$ [m], and the result is seen in Figure 5.8d. The walls are seen much clearer, and the corners are captured with centimeter precision. The total



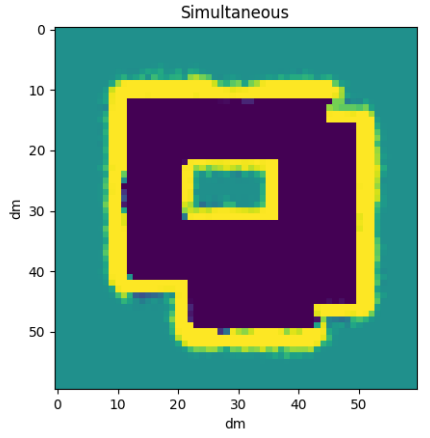
(a) Scenario 1.



(b) Scenario 1. Uncertainty in the position was considered, with 20 samples and the same variance as the noise.

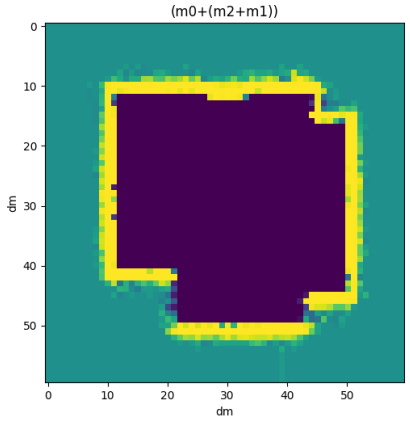


(c) Scenario 2.

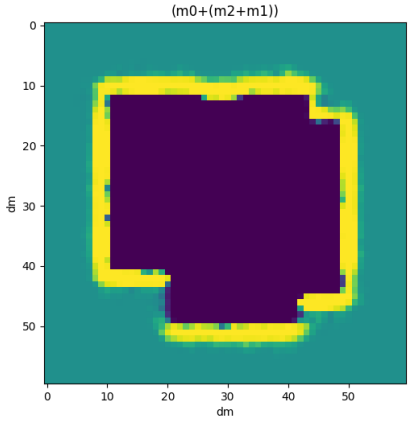


(d) Scenario 2. Uncertainty in the position was considered, with 20 samples and the same variance as the noise.

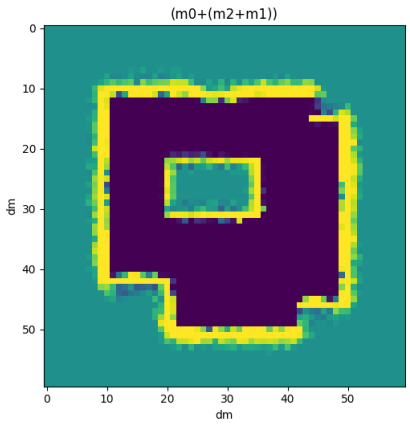
Figure 5.5: *Maps generated by three UAVs, with noisy measurement data, for both map scenarios. The figures illustrate the different cases when the map takes position uncertainty into account or not.*



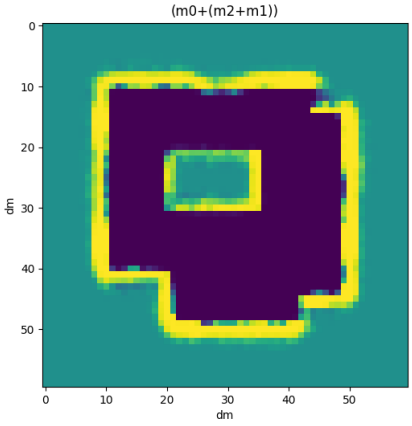
(a) Scenario 1.



(b) Scenario 1. Uncertainty in the position was considered, with 20 samples and the same variance as the noise.

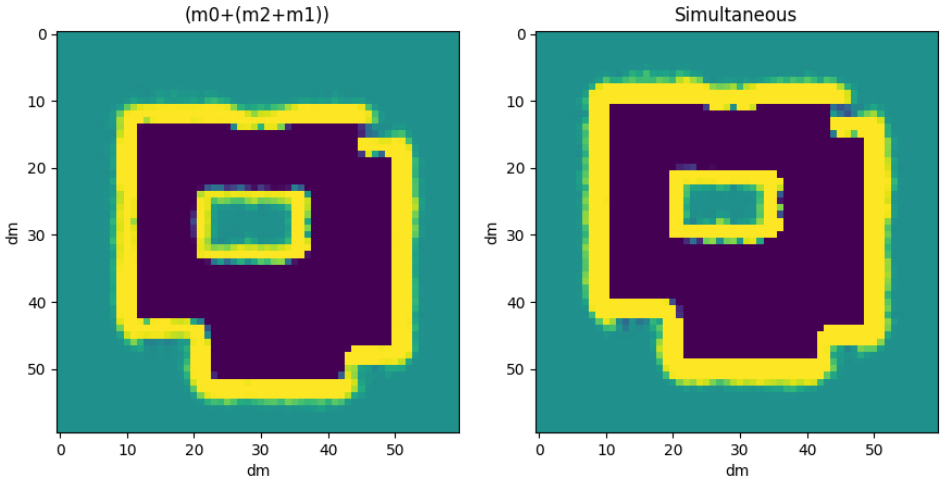


(c) Scenario 2.



(d) Scenario 2. Uncertainty in the position was considered, with 20 samples and the same variance as the noise.

Figure 5.6: *Maps generated by three UAVs, with noisy measurement data, for both map scenarios. The individual constructed maps were merged pairwise. The figures illustrate the different cases when the map takes position uncertainty into account or not.*



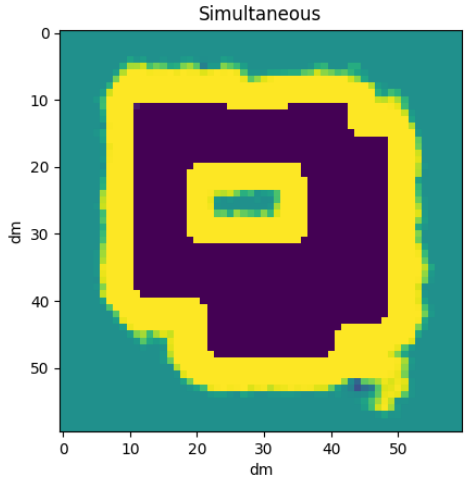
(a) Offline constructed map. *(b)* Online constructed map.

Figure 5.7: *Maps constructed online and offline created by three drones for map scenario 2. The uncertainty in the position was considered sampling* 100 *positions. When comparing with the maps constructed with* 20 *samples the improvement is meager.*

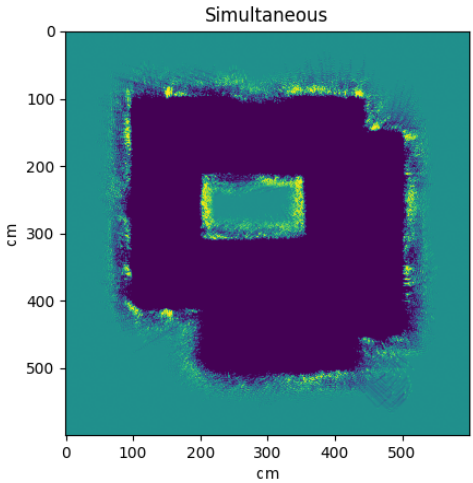
increase of time is $\mathcal{O}(R^3_{map} * n_{pos} * n_{cf} * w)$, and the map takes six hours to complete with this configuration. The walls are easier to see, and the distances between the obstacles are nearly perfect. It is clear when comparing with Figure 5.8b that the map is improved when also compensating for uncertainty in the position.

5.2 Using absolute position (Loco)

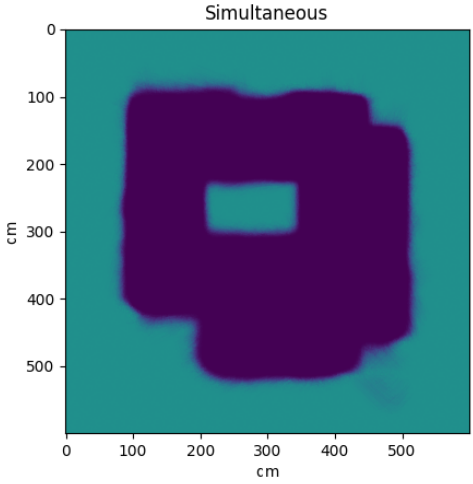
With the Loco positioning system, the global positions of the UAVs are known with a precision of 1 dm. As the positions are known in a global coordinate system, the map can be updated simultaneously. Three UAVs explore the environment for one minute. The measurement samples are saved on a computer to construct the map both with and without uncertainty in the position from the same sensor data. In Figure 5.9, the maps seen are all constructed simultaneously, with a varying number of sampled positions. By comparing Figures 5.9a and 5.9c with Figures 5.9b and 5.9d, it is clear that the uncertainty in position does indeed improve the map. The slightly tilted walls indicate a small clockwise rotation of the map. The mapping system was, for both map scenarios, able to capture the major obstacles in the map. It is clear that considering uncertainty in the position estimates does improve the positioning of the walls and corners. The top right corner and bottom right corner were improved a lot in map scenario 1. In scenario 2, the UAVs did not capture the corners as well, which is also seen when comparing the maps that are produced when not considering uncertainty in the



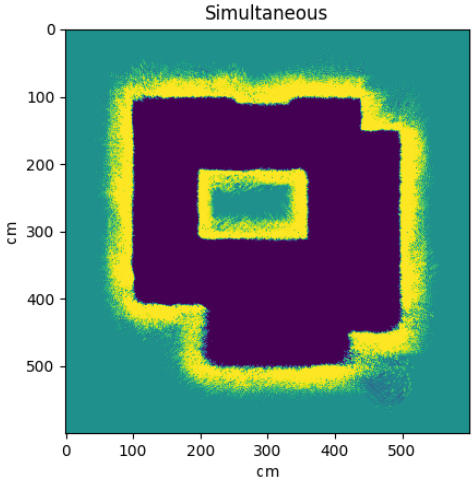
(a) Map with 1 dm as resolution. Uncertainty in the position was considered, with 20 samples and the same variance as the noise.



(b) Map with 1 cm as resolution. Uncertainty in the position was not considered. Created with the wide-walls algorithm.



(c) Map with 1 cm as resolution. Uncertainty in the position was considered, with 20 samples and the same variance as the noise.



(d) Map with 1 cm as resolution. Uncertainty in the position was considered, with 20 samples and the same variance as the noise. Created with the wide-walls algorithm.

Figure 5.8: *Maps generated by three UAVs for map scenario 2. Illustration for the mapper performance when using relative distances between the UAVs, and some methods to use when a higher resolution is wanted.*

position. The improvement for map scenario 2 is instead seen along the walls or for the obstacle in the middle of the map.

5.3 Using relative position

The global position is harder to estimate without an external positioning system and relies on estimates from internal filters instead. The estimated positions of Crazyflies rely on Bitcraze's EKF which uses flowdeck and IMU data. These estimates tend to drift. The position estimate is local with each UAV, allowing each UAV to explore the map in its frame. The individually explored maps are merged after. For online mapping, the position estimate of each UAV is used to synchronize their internal coordinate system, utilizing a shared coordinate system.

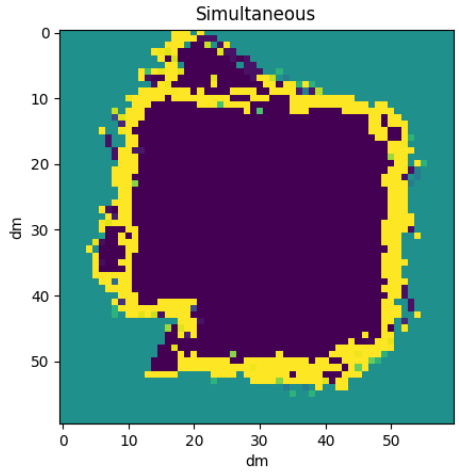
5.3.1 Separate coordinate systems

When the UAVs have separate coordinate systems, the maps must be merged. As presented in Section 2.6.2, there is no difference between creating a map from several UAVs or from several runs. If each run is kept short, and numerous runs are used, the position estimation will not drift as much. The UAVs manage to circle the area of the map one or two times in one minute. In the limited area, it is not possible to use more than about four UAVs to give each enough space. This also decreases the probability for the UAVs to insert other UAVs as obstacles on the map.

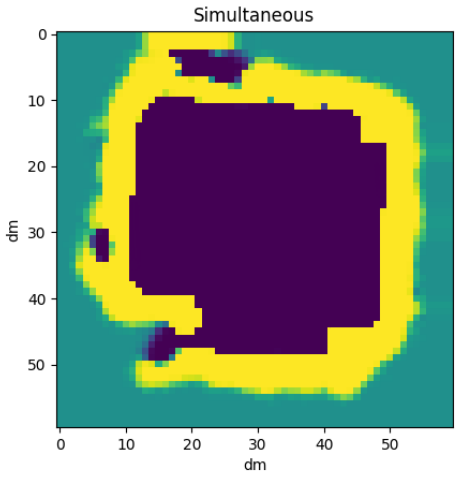
The result seen in Figure 5.10a shows four UAVs flying for half a minute exploring the environment. Observe that the map has a different scale compared to previously displayed maps. The mapper considered the uncertain pose by sampling 20 poses. The map is rotated 90 degrees clockwise compared to the true map in Figure 3.4a. The UAVs have captured the big corner, seen in the upper left corner of the map, and there is some indication of the other corners with diffuse edges. The maps are rotated into the orientation of map m_0. Any rotation leads to some information loss, see Appendix B, so the result can be expected to be worse than that of previous maps. The distances between the walls are quite correct, as is the placement of the top left corner (which would be the bottom lower corner in Figure 3.4a).

5.3.2 Shared coordinate system

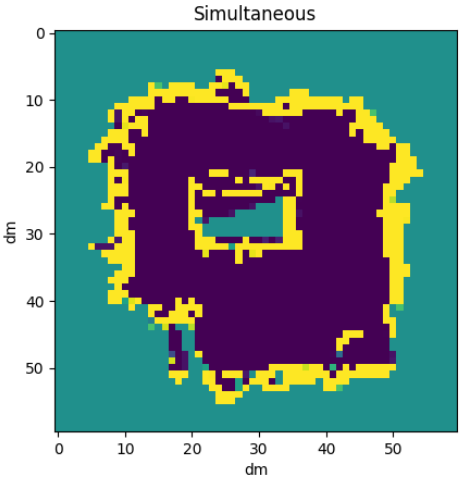
With a shared coordinate system, the positions of all UAVs must either be given in the same coordinate system, or the positions must be known relative to one of the UAVs. The test in Section 5.1.4 uses the basic relative positioning system to determine the formation of the swarm. The relative poses are transformed using the global pose of a reference UAV. The mapper then relies on the global position of the reference UAV and its estimates of where the other UAVs are located. Considering Table 4.3, the uncertainty in the position was around 3 dm for the basic system. This uncertainty was used when constructing the map, and 20 poses were sampled. Figure 5.10b illustrates the result, where the map is rotated



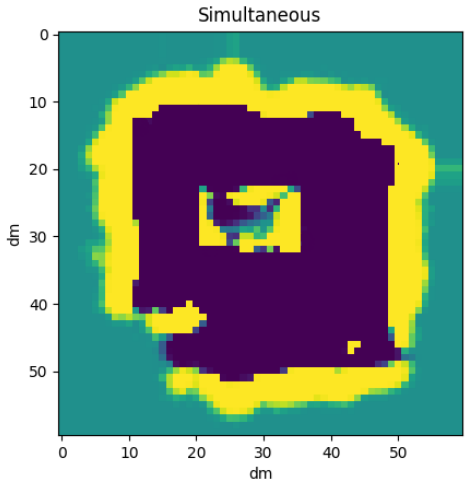
(a) Scenario 1.



(b) Scenario 1. Uncertainty in the position was considered, with 20 samples and the same variance as the noise.



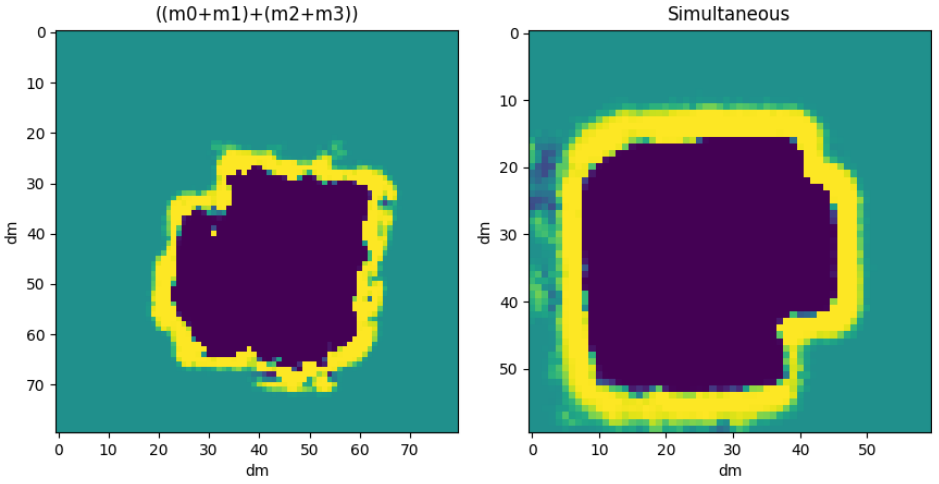
(c) Scenario 2.



(d) Scenario 2. Uncertainty in the position was considered, with 20 samples and the same variance as the noise.

Figure 5.9: *Maps generated by three UAVs, for both map scenarios. The Crazyflies are positioned by the Loco positioning system. The Figures illustrate the different cases when the map takes position uncertainty into account or not.*

90 degrees counter-clockwise. The corners were difficult to capture due to the big uncertainty. However, the major corner, seen in the bottom right corner, is well-defined.



(a) *Merged map by four UAVs for map scenario 1. Each UAV constructed the map in their local frame.*

(b) *Map constructed collaboratively by three UAVs for map scenario 1. The UAVs started from a known formation and used a shared coordinate system.*

Figure 5.10: *Constructed maps from real flights with separate or shared coordinate systems. The mapper considered an uncertain position.*

5.4 Summary and discussion

For a swarm, one can let the UAVs work on individual maps to be merged. The merging is in this report only performed after a flight is done, but can be done during the flight by copying the grid from each UAV and merging them. The merged map can then be introduced to each UAV as the prior term in the grid map. As seen in Section 5.1.2 the merging order matters, but has minimal impact on how the map looks. It is therefore useful to fix one map and perform pairwise merging of the maps until one final map is achieved. The measure used from the introduced cost function from Section 2.6.2 gives a hint of how the performance of the achieved map is, and can both be used to know how much two maps to be merged overlap or to compare the similarity with the true map. For other maps, there are however indications from the results that some cost functions will not find the optimal translation and (or) rotation. For example, (2.51) will have trouble with a map where the area of the $\frac{\text{unknown}}{\text{mapped}}$ space $\rightarrow$ becomes very large, (2.52) will have trouble when the $\frac{\text{free}}{\text{occupied}}$ space becomes very large. The modified version, (3.1), measuring the precision for the positions of the obstacles is more robust in these extremes but might have trouble finding the true transformation if the created map is subject to ambiguity regarding rotation and flip. This is however the case for all presented cost functions.

In Table 5.1 the three suggested cost functions were investigated, and it was shown that (2.52) and (3.1) scaled as expected with the area of free space and length of the walls, respectively, both for offline and online mapping. The equality check (2.51) between maps scaled with the area of unknown space. However, the function differs between online and offline mapping, which is something to remark on.

The difference was investigated, and the best guess for the difference is due to the distinction in how the maps are created. For offline maps, the probability grids are converted to log-odds grids and then merged, followed by conversion back to a probability map. In the true map, free space is represented by zeros, and occupied space is represented by ones. For cells to be equal, there must either be calculations involving division by zero or logarithms of zero, as free and occupied space in the exact map corresponds to a value of 0 and 1, respectively. These operations are not only mathematically undefined but will not return exact 0 nor 1, without any floating point rest, in numerical calculations. With the conversions back and forth, some information is lost.

The grid introduced here is initialized with a fixed size, which requires knowledge about how large the area to be mapped is. The method of merging maps does not require any communication within the swarm, nor to a computer during flight. The swarm would explore the area and each UAV log its estimated pose and sensor data. The data can be collected after the flight and used to construct the map. Such an implementation can be useful in areas where communication is the main problem but requires that each UAV has a good internal filter for pose estimation.

Relative positioning is required in an implementation involving collaborative mapping. The investigated method assumed that the pose of one UAV was known

globally, this UAV was denoted the reference UAV. The global poses of the other UAVs were derived from the poses relative to the reference UAV and were given in the horizontal frame of the reference UAV. All positions then drifted with the position of the reference UAV. There are numerous scenarios where one UAV is equipped with better sensor data for estimating its global position, and only one UAV needs to know its pose in the global frame. With the Loco positioning system, the global position had an accuracy of 1 dm, and would not drift. When not using the Loco system, the position of the reference UAV drifted. It might be interesting to look at the swarm's common picture of the global coordinate system. That is an implementation where the position estimate of each UAV is used to align the coordinate systems between the UAVs. Such an implementation could decrease the drift.

When considering uncertainty in the position, the maps were improved. In the simulator, white noise with a standard deviation of 1 dm was added to the position of the reference drone, corresponding to the accuracy of the Loco positioning system. However, the error in the positions given by the Loco positioning system is not Gaussian distributed, leading to less improvement than for the simulated data. Even though 20 samples were quite few when sampling poses it was enough to complete a map. Since the data were collected at 100 Hz, the UAVs only managed to move a small distance between each sample.

The time complexity when mapping is of great interest, balancing time versus details in the map. The map with centimeter resolution instead of decimeter resolution introduced more details in the map, but with great cost regarding time consumption. When comparing the maps based on simulated data to the maps based on real-world data, the difference is huge.

6

Conclusions and further work

This chapter summarizes the main results presented in Chapter 4 and Chapter 5 and presents possible ideas for future work.

6.1 Conclusions

The results presented in Chapter 4 show that making more distances available when filtering does improve accuracy. As an example, the average filter error is reduced by about 47.2 percent compared to the basic filter when using all available distances within a swarm of eight UAVs, a vast improvement. The accuracy achieved has a value of 0.1639 m. Using UWB distance measurements and ego velocity data within a swarm of UAVs, together with an extended Kalman filter designed as described in Section 3.4.2, clearly enables relative pose estimation, yielding an answer to the first question stated in the introduction. Mapping was shown to work with the basic filter, which indicates that the extended filter, with its nearly double accuracy, should handle the task neatly.

Still, improved accuracy from using more distance measurements comes at the cost of potentially pushing the computational limits of hardware UAVs. It remains somewhat unclear how the delays of real UWB communication, the method for collecting distance measurements mentioned in the first report question, can lead to accuracy problems. Delays are, additionally, likely to get worse when increasing the UAV count of a swarm. Convergence also gets harder to achieve when using many UAVs and starting with unknown states. This issue is clearly displayed by convergence tests presented in the results section, where a swarm of eight UAVs using all available distance measurements reaches convergence only in about half the test runs.

For the convergence issues, effective remedies can evidently be applied, like the MDS initialization method that was described in Section 3.4.1. In real sce-

narios, this method is also meant to use UWB ranging and ego velocity data, the sensor data mentioned in the first report question, and can beneficially be used together with the extended filtering system. The relative poses can thereby be accurately estimated nearly instantly. As for problems with computational capacities on hardware UAVs, large segments of the positioning systems can be moved to a central computer. However, since there are limits on how many Crazyflies can connect to a computer, there might still be computational limits imposed, in other ways, on the relative positioning systems.

Switching to the other focus of the report, the mapping strategy relies on knowing where drones are located, and what each individual drone sees, in terms of sensor data. The data is inserted into an occupancy grid where the map is constructed. By infusing the position of the drone with uncertainty, irregularities in the walls or corners get smoothed out. The uncertainty in the position is achieved by uniformly sampling positions around the estimated position. When inserting sensor data into the grid, the relative position information is used to know where the obstacles detected by the LIDAR should be inserted. Before inserting information about the obstacles, the range detections are transformed to the horizontal frame with information from the IMU. If the resolution is to be increased, the inverse sensor model needs to be adjusted for the obstacles to be seen on the map.

For a swarm, the collaboratively constructed map was successfully created in two ways, both by merging the locally created maps and by cooperating on a common grid. If local maps are used, the position estimate of each drone is in the local map, and all maps must be merged to know the position of the swarm on the global map. As the merging of maps takes time, the positions of the drones will have drifted until the estimated position can be seen. The collaboratively constructed map relied on global pose estimates for at least one UAV in the swarm, called the reference or origin-UAV. The mapper then uses the pose of the reference drone and its estimation of the poses of the other drones in the swarm to determine the positions of all UAVs in a shared coordinate system. Both implementations use sensor data to construct a map, but neither of them uses the map to locate. This is the main difference from a SLAM implementation.

The answer to the last report question inherently reasons around the overall goal of the report, namely to investigate methods for mapping the environment in which a UAV swarm acts, utilizing relative position estimates based on relative distance measurements. A general conclusion is that there are many different methods for separately approaching mapping and localization problems using UAV swarms, leading the way to even more potential strategies for combining promising solutions. Results of tests conducted during the work of this thesis present two viable strategies for combining both types of systems, but many more could be explored.

6.2 Further work

A takeaway from real-world tests of the positioning systems is that the performance of the involved filters is radically worsened when the update frequency

is decreased. Therefore, it could be a good idea to develop an improved motion model, more resilient to large time deltas and noise in velocity estimates.

The positioning filter using MDS coordinates as measurements did not work as expected and the propositions for the improvement of that filter could be investigated. The method of taking the positions of all UAVs into account when rotating and flipping to counter ambiguities could potentially keep the filter from divergence after the first convergence.

For a SLAM implementation, the map must be used to localize each drone in the swarm. Both methods for constructing the map provide sufficient information for a proper SLAM implementation, that is an implementation where the LIDAR is used not only to insert information in the map but also used to help the localization in the so-far constructed map. It would therefore be a challenging and interesting task to extend the position estimating with LIDAR data.

The resolution of a map has a major impact on the time needed for constructing a merged map, where a comparison between each cell is necessary. [13] did not only explore how the incorporation of uncertain sensors could improve the mapping algorithm, but also how the resolution of the map could be determined. The grid map can be modified to use dynamic resolution. The cells in the grid are then locally merged or split. Splitting cells improves the resolution in the nearby area and should be done when two sensor measurements disagree on the classification of a cell in the grid. This subject could be investigated further.

A final idea for further work is to test the various mapping systems with the best-performing relative localization system. Hardware-related issues and lack of time came in the way during the course of this work. It could be an interesting task to implement functionality for combining the systems on hardware UAVs.

Appendix

A

Extended positioning system for a swarm of three UAVs

The notations related to the EKF of the extended positioning system for the case of a swarm with three UAVs, where all available pairwise distances are used, is described in Table A.1.

Table A.1: *Notations related to the Kalman filtering process used to estimate the relative positions and yaw angles of agents 2 and 3 in the HF of agent 1.*

Notations
$$X = \begin{bmatrix} \mathbf{p_{12}} \\ \mathbf{p_{13}} \end{bmatrix} = \begin{bmatrix} x_{12} & y_{12} & \psi_{12} & x_{13} & y_{13} & \psi_{13} \end{bmatrix}^T$$
$$U = \begin{bmatrix} v_1^T & r_1 & v_2^T & r_2 & v_3^T & r_3 \end{bmatrix}^T$$ $$v_i = \begin{bmatrix} v_i^x & v_i^y \end{bmatrix}^T$$
$$\dot{X} = \begin{bmatrix} \cos(\psi_{12})v_2^x - \sin(\psi_{12})v_2^y - v_1^x + y_{12}r_1 \\ \sin(\psi_{12})v_2^x + \cos(\psi_{12})v_2^y - v_1^y - x_{12}r_1 \\ r_2 - r_1 \\ \cos(\psi_{13})v_3^x - \sin(\psi_{13})v_3^y - v_1^x + y_{13}r_1 \\ \sin(\psi_{13})v_3^x + \cos(\psi_{13})v_3^y - v_1^y - x_{13}r_1 \\ r_3 - r_1 \end{bmatrix}$$

$$f_1(i) = -\sin(\psi_{1i})v_i^x - \cos(\psi_{1i})v_i^y,\ f_2(i) = \cos(\psi_{1i})v_i^x - \sin(\psi_{1i})v_i^y$$

$$A = \begin{bmatrix} 1 & r_1\Delta t & f_1(2)\Delta t & 0 & 0 & 0 \\ -r_1\Delta t & 1 & f_2(2)\Delta t & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & r_1\Delta t & f_1(3)\Delta t \\ 0 & 0 & 0 & -r_1\Delta t & 1 & f_2(3)\Delta t \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

$$B = \begin{bmatrix} -1 & 0 & y_{12} & \cos(\psi_{12}) & -\sin(\psi_{12}) & 0 & 0 & 0 & 0 \\ 0 & -1 & -x_{12} & \sin(\psi_{12}) & \cos(\psi_{12}) & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 & 0 & 1 & 0 & 0 & 0 \\ -1 & 0 & y_{13} & 0 & 0 & 0 & \cos(\psi_{13}) & -\sin(\psi_{13}) & 0 \\ 0 & -1 & -x_{13} & 0 & 0 & 0 & \sin(\psi_{13}) & \cos(\psi_{13}) & 0 \\ 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

$$z = \begin{bmatrix} d_{12} & d_{13} & d_{23} \end{bmatrix}^T$$

$$h(X) = \begin{bmatrix} \sqrt{x_{12}^2 + y_{12}^2} \\ \sqrt{x_{13}^2 + y_{13}^2} \\ \sqrt{(x_{13} - x_{12})^2 + (y_{13} - y_{12})^2} \end{bmatrix}$$

$$H = \begin{bmatrix} x_{12}/z_1 & y_{12}/z_1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & x_{13}/z_2 & y_{13}/z_2 & 0 \\ -(x_{13} - x_{12})/z_3 & -(y_{13} - y_{12})/z_3 & 0 & (x_{13} - x_{12})/z_3 & (y_{13} - y_{12})/z_3 & 0 \end{bmatrix}$$

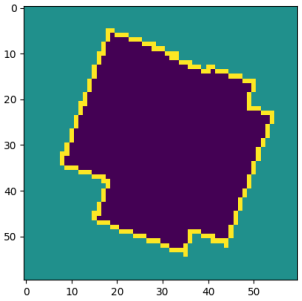
$$R = \begin{bmatrix} \sigma_d^2 & 0 & 0 \\ 0 & \sigma_d^2 & 0 \\ 0 & 0 & \sigma_d^2 \end{bmatrix}$$

$$Q = \begin{bmatrix} \sigma_{v_{xy}}^2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & \sigma_{v_{xy}}^2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \sigma_{v_r}^2 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \sigma_{v_{xy}}^2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \sigma_{v_{xy}}^2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \sigma_{v_r}^2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & \sigma_{v_{xy}}^2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & \sigma_{v_{xy}}^2 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \sigma_{v_r}^2 \end{bmatrix}$$

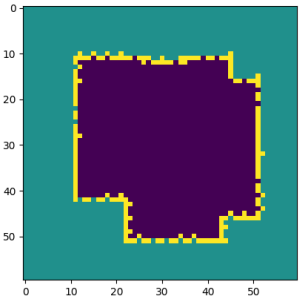
B

Information loss due to rotation of grids

The rotation of a grid is performed as described in Section 5.1.2, but each rotation leads to some informational loss due to truncation of grid indices. To show this, the original map illustrated in Figure 3.4a is first rotated 20 degrees clockwise, illustrated in Figure B.1a, and then 20 degrees anti-clockwise, illustrated in Figure B.1b.



(a) *The example map in Figure 3.4a rotated 20 degrees clockwise.*



(b) *The map rotated back to its original orientation.*

Figure B.1: *Illustration of how a rotation of a grid leads to some informational loss for the map. The left map is rotated 20 degrees, and the right map is rotated back to its original orientation.*

Bibliography

[1] Bitcraze AB. Crazyflie 2.1, 2022. URL https://www.bitcraze.io/products/crazyflie-2-1/.

[2] Bitcraze AB. State estimation, 2022. URL https://www.bitcraze.io/documentation/repository/crazyflie-firmware/master/functional-areas/sensor-to-control/state_estimators/.

[3] Bitcraze AB. The coordinate system of the crazyflie 2.X, 2022. URL https://www.bitcraze.io/documentation/system/platform/cf2-coordinate-system/.

[4] Bitcraze AB. Flow Deck v2, 2022. URL https://www.bitcraze.io/products/flow-deck-v2/.

[5] Bitcraze AB. Loco Positioning System, 2022. URL https://www.bitcraze.io/documentation/system/positioning/loco-positioning-system/.

[6] Bitcraze AB. Loco Positioning System, 2022. URL https://www.bitcraze.io/documentation/system/positioning/accuracy-loco/.

[7] Bitcraze AB. Loco Positioning System, 2022. URL https://www.bitcraze.io/documentation/repository/crazyflie-firmware/master/functional-areas/loco-positioning-system/.

[8] Bitcraze AB. Multi-ranger deck, 2022. URL https://www.bitcraze.io/products/multi-ranger-deck/.

[9] Andreas Birk and Stefano Carpin. Merging occupancy grid maps from multiple robots. *IEEE Proceedings, special issue on Multi-Robot Systems*, 94(7): 1384–1397, 2006. doi: 10.1109/JPROC.2006.876965.

[10] J. E. Bresenham. Algorithm for computer control of a digital plotter. *IBM Systems Journal*, 4(1):25–30, 1965. doi: 10.1147/sj.41.0025.

[11] Rui Chen, Bin Yang, and Wei Zhang. Distributed and collaborative localization for swarming uavs. *IEEE Internet of Things Journal*, 8(6):5062–5074, 2021. doi: 10.1109/JIOT.2020.3037192.

[12] Ruwan Egodagamage and Mihran Tuceryan. A Collaborative Augmented Reality Framework Based on Distributed Visual Slam. In *2017 International Conference on Cyberworlds (CW)*, pages 25–32, 2017. doi: 10.1109/CW.2017.47.

[13] Victor Terra Ferrão, Cássio Dener Noronha Vinhal, and Gelson da Cruz. An Occupancy Grid Map Merging Algorithm Invariant to Scale, Rotation and Translation. In *2017 Brazilian Conference on Intelligent Systems (BRACIS)*, pages 246–251, 2017. doi: 10.1109/BRACIS.2017.69.

[14] Jeffrey Humpherys, Preston Redd, and Jeremy West. A Fresh Look at the Kalman Filter. *SIAM Review*, 54(4):801–823, 2012. doi: 10.1137/100799666. URL https://doi.org/10.1137/100799666.

[15] Timon Idema. Rotating Reference Frames, 2020. URL https://batch.libretexts.org/print/Finished/phys-17360/Full.pdf.

[16] Daniek Joubert, Willie Brink, and Ben Herbst. Pose uncertainty in occupancy grids through Monte Carlo integration. In *2013 16th International Conference on Advanced Robotics (ICAR)*, pages 1–6, 2013. doi: 10.1109/ICAR.2013.6766589.

[17] S.J. Julier and J.K. Uhlmann. Unscented filtering and nonlinear estimation. *Proceedings of the IEEE*, 92(3):401–422, 2004. doi: 10.1109/JPROC.2003.823141.

[18] Marco Karrer, Patrik Schmuck, and Margarita Chli. CVI-SLAM—Collaborative Visual-Inertial SLAM. *IEEE Robotics and Automation Letters*, 3(4):2762–2769, 2018. doi: 10.1109/LRA.2018.2837226.

[19] Tony Lacey. *Tutorial: The Kalman Filter*, chapter 11, pages 133–140. MIT, 1998.

[20] Pierre-Yves Lajoie, Benjamin Ramtoula, Yun Chang, Luca Carlone, and Giovanni Beltrame. DOOR-SLAM: Distributed, Online, and Outlier Resilient SLAM for Robotic Teams. *CoRR*, abs/1909.12198, 2019. URL http://arxiv.org/abs/1909.12198.

[21] Shushuai Li, Mario Coppola, Christophe De Wagter, and Guido C. H. E. de Croon. An autonomous swarm of micro flying robots with range-based relative localization. *CoRR*, abs/2003.05853, 2020. URL https://arxiv.org/abs/2003.05853.

[22] Wenchao Li, Beth Jelfs, Allison Kealy, Xuezhi Wang, and Bill Moran. Cooperative Localization Using Distance Measurements for Mobile Nodes. *Sensors*, 21(4), 2021. ISSN 1424-8220. doi: 10.3390/s21041507. URL https://www.mdpi.com/1424-8220/21/4/1507.

[23] Mathworks. What is SLAM (simultaneous localization and mapping), 1 2022. URL https://se.mathworks.com/discovery/slam.html.

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

[25] Dries Neirynck, Eric Luk, and Michael McLaughlin. An alternative double-sided two-way ranging method. In *2016 13th Workshop on Positioning, Navigation and Communications (WPNC)*, pages 1–4, 2016. doi: 10.1109/WPNC.2016.7822844.

[26] Matteo Palieri, Benjamin Morrell, Abhishek Thakur, Kamak Ebadi, Jeremy Nash, Arghya Chatterjee, Christoforos Kanellakis, Luca Carlone, Cataldo Guaragnella, and Ali-Akbar Agha-Mohammadi. LOCUS: A Multi-Sensor Lidar-Centric Solution for High-Precision Odometry and 3D Mapping in Real-Time. *CoRR*, abs/2012.14447, 2020. URL https://arxiv.org/abs/2012.14447.

[27] Yan Pei, Swarnendu Biswas, Donald Fussell, and Keshav Pingali. An Elementary Introduction to Kalman Filtering. *Communications of the ACM*, 62, 10 2017. doi: 10.1145/3363294.

[28] Patrik Schmuck and Margarita Chli. Multi-UAV collaborative monocular SLAM. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3863–3870, 2017. doi: 10.1109/ICRA.2017.7989445.

[29] Jiawen Shen, Shizhuang Wang, Yawei Zhai, and Xingqun Zhan. Co-operative relative navigation for multi-UAV systems by exploiting GNSS and peer-to-peer ranging measurements. *IET Radar, Sonar & Navigation*, 15(1):21–36, 2021. doi: https://doi.org/10.1049/rsn2.12023. URL https://ietresearch.onlinelibrary.wiley.com/doi/abs/10.1049/rsn2.12023.

[30] Marco Taboga. Covariance matrix, Lectures on probability theory and mathematical statistics, 2021. URL https://www.statlect.com/fundamentals-of-probability/covariance-matrix.

[31] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic robotics*. MIT Press, Cambridge, Mass., 2005. ISBN 0262201623 9780262201629. URL http://www.amazon.de/gp/product/0262201623/102-8479661-9831324?v=glance&n=283155&n=507846&s=books&v=glance.

[32] Viktor Tuul. Online Collaborative Radio-enhanced Visual-inertial SLAM. Master's thesis, Kungliga Tekniska Högskolan, 6 2019.

[33] Florian Wickelmaier. An introduction to MDS, May 2004. URL https://www.hongfeili.com/files/paper100/paper4.pdf.

[34] Richard Wilkinson. Multivariate statistics, 2022. URL https://rich-d-wilkinson.github.io/MATH3030/6-mds.html.

[35] Shuien Yu, Chunyun Fu, Amirali K. Gostar, and Minghui Hu. A Review on Map-Merging Methods for Typical Map Types in Multiple-Ground-Robot SLAM Solutions. *Sensors*, 20(23), 2020. ISSN 1424-8220. doi: 10.3390/s20236988. URL https://www.mdpi.com/1424-8220/20/23/6988.