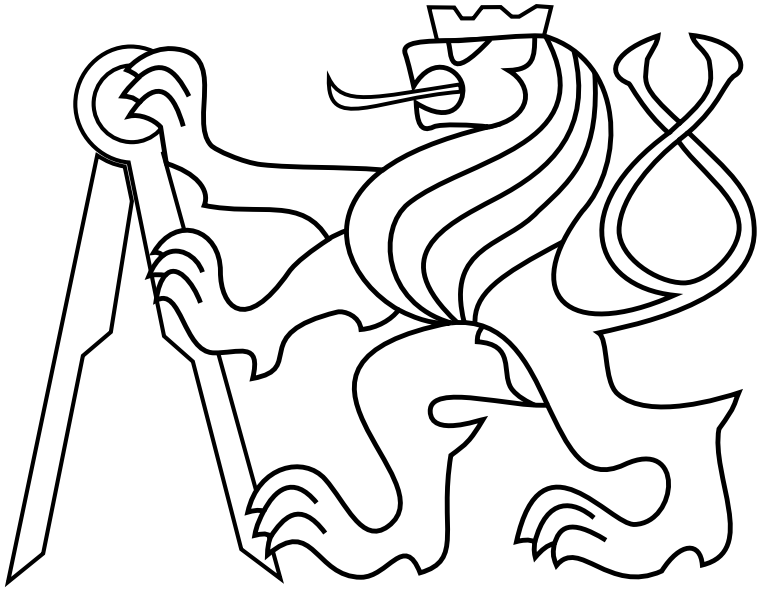
CZECH TECHNICAL UNIVERSITY IN PRAGUE

Faculty of Electrical Engineering

BACHELOR'S THESIS



Martin Křížek

Swarming of Unmanned Aerial Vehicles Using Indirect Information Exchange by Observation of the Workspace

May 2021

Department of Cybernetics

Thesis supervisor: **Ing. Jiří Horyna**

BACHELOR'S THESIS ASSIGNMENT

I. Personal and study details

Student's name: **Křížek Martin** Personal ID number: **483479**

Faculty / Institute: **Faculty of Electrical Engineering**

Department / Institute: **Department of Cybernetics**

Study program: **Cybernetics and Robotics**

II. Bachelor's thesis details

Bachelor's thesis title in English:

Swarming of Unmanned Aerial Vehicles Using Indirect Information Exchange by Observation of the Workspace

Bachelor's thesis title in Czech:

Rojové chování bezpilotních helikoptér používající nepřímou výměnu informace pozorováním pracovního prostoru

Guidelines:

The aim of this thesis is to design, implement and verify methods for swarming and relative localization of micro aerial vehicles (MAV) in forests. The thesis will be divided into the following tasks:

- To understand boids model of flocking of MAVs and the system of MRS group at CTU designed for stabilization of MAV groups.
- To design and implement a relative localization method for estimation of relative positions of neighboring MAVs based on local map sharing.
- To design an extension of the boids model to achieve reliable and faster flight through a forest environment.
- To integrate the system into the Robot Operating System (ROS) and verify its behavior with MAV models in realistic Gazebo simulator.
- To compare the obtained method with the current swarm system designed at MRS group for forest flying.
- If weather, COVID-19 restrictions, and platforms availability allows, to prepare a HW outdoor experiment.

Bibliography / sources:

[1] Quigley, Morgan, et al. "ROS: an open-source Robot Operating System." ICRA workshop on open source software. Vol. 3. No. 3.2. 2009.
[2] Vásárhelyi, Gábor, et al. "Optimized flocking of autonomous drones in confined environments." Science Robotics 3.20 (2018).
[3] Sobreira, Héber, et al. "Map-matching algorithms for robot self-localization: a comparison between perfect match, iterative closest point and normal distributions transform." Journal of Intelligent & Robotic Systems 93.3-4 (2019).

Name and workplace of bachelor's thesis supervisor:

Ing. Jiří Horyna, Multi-robot Systems, FEE

Name and workplace of second bachelor's thesis supervisor or consultant:

Date of bachelor's thesis assignment: **12.01.2021** Deadline for bachelor thesis submission: **21.05.2021**

Assignment valid until: **30.09.2022**

Ing. Jiří Horyna	prof. Ing. Tomáš Svoboda, Ph.D.	prof. Mgr. Petr Páta, Ph.D.
Supervisor's signature	Head of department's signature	Dean's signature

III. Assignment receipt

The student acknowledges that the bachelor's thesis is an individual work. The student must produce his thesis without the assistance of others, with the exception of provided consultations. Within the bachelor's thesis, the author must state the names of consultants and include a list of references.

. _____
Date of assignment receipt

_____
Student's signature

Author statement for undergraduate thesis:

I declare that the presented work was developed independently and that I have listed all sources of information used within it in accordance with the methodical instructions for observing the ethical principles in the preparation of university theses.

Prague, date..... .....

Signature

Acknowledgements

I would like to thank my thesis supervisor Ing. Jiří Horyna for his great advice on this thesis. My thanks go also to Bc. Afzal Ahmad, Ing. Martin Saska Dr. rer. nat., and other members of the Multi-Robot Systems group as this thesis is building on the system they created. Last but not least, many thanks belong to my friends and family for their ever-optimistic attitude.

Abstract

This thesis focuses on the design, implementation, and verification of a control system and relative localization approach for a swarm consisting of unmanned aerial vehicles in a forest environment. The core of the localization system is the ICP algorithm. The control system is based on Boids with modifications to adapt to the forest environment better. Implementation was verified in the realistic Gazebo simulator as well as in Matlab. The approach introduced in this thesis was also compared with the existing system for relative localization and navigation used in the Multi-Robot Systems group at Czech Technical University in Prague.

Keywords

UAV, drone, swarm, boids, obstacle avoidance, ICP

Abstrakt

Tato práce se soustředí na návrh, implementaci a ověření řídícího systému a systému pro relativní lokalizaci roje bezpilotních autonomních helikoptér v lesním prostředí. Základem lokalizačního systému je ICP algoritmus. Rojový řídící systém je inspirován Boidy a modifikován pro lepší interakci s reálným prostředím. Implementace byla ověřena v realistickém simulátoru Gazebo a pomocí Matlabu. Přístup, který je uveden v této práci, byl následně porovnán se současným systémem pro relativní lokalizaci a navigaci v lese, které používá skupina Multi-robotických systémů na ČVUT v Praze.

Klíčová slova

UAV, dron, roj, boids, uhýbání překážkám, ICP

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Robots in this thesis are called Unmanned Aerial Vehicles (UAV). These airborne vehicles do not carry a pilot on board. They can be controlled either by a human pilot remotely or they can fly autonomously. The UAVs used in this thesis are quadcopters. A quadcopter is a vehicle that is propelled by four propellers. Quadcopters or generally multicopters are being used on daily basis in many domains thanks to their multipurpose usage and relatively low cost. They can be used for area monitoring, search and rescue, military deployment or film making.

Figure 1.1: Swarm of UAVs navigating through a forest.

In this thesis, a group of UAVs is called a swarm (Figure 1.1). Individual UAV has limits as flight range that is influencing the area that a survey UAV can cover. Further, single UAV has limited thrust force for transportation of sizeable objects. Swarm is used to compensate for those limits and achieve more difficult tasks. Swarms of UAVs have many

applications. They can be used for search and rescue missions in hard to reach and difficult terrain to localize survivors [2] [3] [4]. Another use is in security and surveillance of certain areas [5] [6]. UAVs could measure the level of air pollution in cities [7] [8]. Swarms of UAVs could in the future help with fire localization or extinguishing [9] [10] [11]. Last but not least, swarms have potential in crop monitoring and agriculture [12] [13] [14], and much more.

Finding ways of safe navigation and localization of swarm members in a forest environment has many uses. The above mentioned search and rescue missions are one of the possible applications. Swarm system can be useful in cases where the terrain is difficult to reach by human rescuer and the area can not be observed from a high point because the trees obscure vision. There is also the possibility that system for navigation and localization in a forest environment will be generalized and used for localization and navigation in any obstacle-dense environment for safe pathfinding and localization.

This thesis is dealing with the task of navigation and localization of a swarm in a forest environment. A modified Boids-like system [15] [1] is used for control and navigation of the swarm. Indirect information exchange is used for the relative localization of nearby UAVs. Version of Iterative Closest Point (ICP) algorithm [16] is employed on shared laser scans to estimate positions of neighboring vehicles.

1.1 State of the art

The partial goal of this thesis is to improve the system introduced in [1]. In [1], a bio-inspired approach is chosen. UAVs do not communicate and they use only on-board sensors for relative localization purposes. LIDAR sensor is used for obstacle detection. The UVDAR system [17] is employed for relative localization without communication. The control mechanism is based on the idea of Boids from [15].

A similar way of swarm deployment is introduced in [18], where again a version of the bio-inspired Boids-like model is used for navigation. The system is fully autonomous without external localization or communication. As in the previous case, the UVDAR system [17] is used for the relative localization of neighboring agents.

Another swarming approach that was introduced in [19] is based on the behavior of a swarm of bees. Such a solution requires a minimum count of swarm agents thus it is not optimal for smaller groups of for example 5 UAVs. It is designed to be used for finding sources of light, heat, or radiation. The agents do not communicate nor have any memory. The process is fully decentralized and autonomous using only the on-board sensors.

Further, the authors of [20] also use principles of repulsion, velocity alignment, and collective and object collision avoidance. A combination of vectors produced is then used to steer the UAVs in the swarm. It has been validated on real hardware with a swarm of 30 drones and proved to be stable under realistic conditions.

There are already techniques for relative localization in areas without a global positioning system. One of them is the UVDAR localization system [17], [21]. The UAVs are equipped with ultraviolet LED markers. These markers emit light of frequencies that are found less in nature. The UAVs are also equipped with UV sensitive cameras with specialized bandpass filters. The UVDAR system relies only on the onboard sensors and estimates the position of neighboring UAVs by observing the positions of their respective markers in an image frame.

Another approach [22] uses Ultra-Wideband (UWB) technology for relative localization of UAVs. Each UAV first estimates its position relative to a static UAV. UWB ranging technology is then used to relatively localize neigboring crafts. UWB technology can be used not only to estimate the location but also the orientation of the device.

Last but not least, relative localization can be achieved using machine learning. The way of automatic annotation for learning datasets using the UVDAR system is presented in [23]. Convolutional neural networks for object detection can be trained using such a dataset for relative localization of UAVs.

1.2 Problem statement

The goal of this thesis was to design a system for swarm navigation in a forest environment with the following requirements:

1. Relative mutual localization of swarm members independent of any global positioning system.
2. Minimal inter-UAV communication. Only a local transmission between neighboring agents is used for relative localization computation.
3. Full autonomy. Given starting and goal position the group will coherently relocate.
4. Robustness of the designed system to situations such as short-term loss of network connection or failure of swarm individuals.
5. Scalability of the designed system.

UAV platform requirements for the problem are: rangefinder for height estimation, onboard computer, inertial measurement unit, and laser scanner. We suppose that the map of the environment is unknown. The UAVs are assumed to be capable of communicating with each other.

Chapter 2

Preliminaries

We will introduce crucial software systems and mathematical expressions necessary to understand the rest of this work. Namely, it is Robot Operating System (ROS), Gazebo realistic simulator, Boids swarm model, and used frames of reference.

2.1 Robot Operating System

ROS is a framework for robot software. It includes tools, libraries, and others. It helps to create robot behavior across a wide variety of robotic platforms. It is a middleware software for robot software development. The core of ROS are messages, services, topics, and nodes. Nodes[1] can subscribe and publish to a topic (Figure 2.1). A node represents a ROS process. Topics[2] can be thought of as channels of messages[3]. Any node can subscribe to a topic (accepting messages) as well as it can publish to a topic (sending messages). Services[4] are another type of connection between nodes. Services are used for one-on-one communication for action with defined beginning and end. Service interaction is comprising of request and reply. ROS also includes rviz[5]. Rviz is a visualization tool for visualizing robots in their environment with many configurable options.

2.2 Gazebo simulator

Gazebo simulator[6] is a 3D simulator with a robust physics engine for robotics. It is ideal for tests of algorithms in various modeled environments without hardware that

[1] http://wiki.ros.org/Nodes
[2] http://wiki.ros.org/Topics
[3] http://wiki.ros.org/Messages
[4] http://wiki.ros.org/Services
[5] http://wiki.ros.org/rviz
[6] http://gazebosim.org/

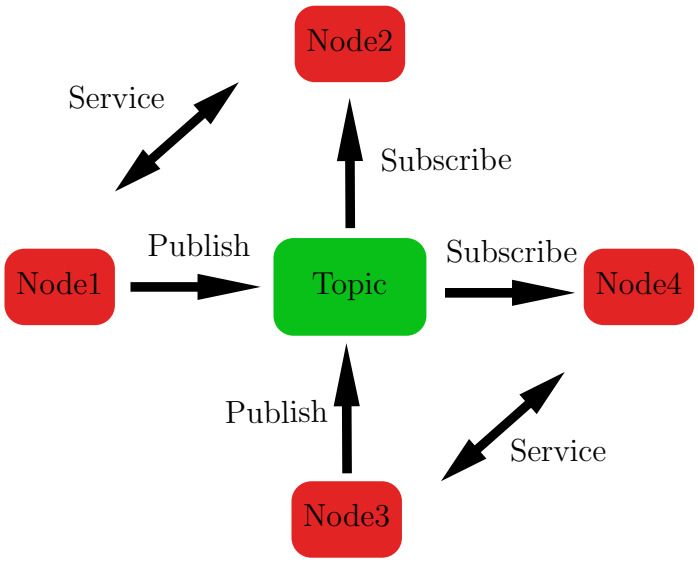


Figure 2.1: Visualization of available ways of communication between ROS nodes.

might not be to disposal at the moment. It also eliminates the danger of damaging robot hardware. Own robotic devices, as well as environments, can be created for testing purposes. Sensors can be designed to interact with the environment. It provides graphical components including textures, lightning and shadows as displayed in Figure 2.2. Gazebo is an open-source project. The majority of simulations in this thesis were carried out in Gazebo.

2.3 Boids

We present an overview of the former Boids controller [1] used within our group. The controller was based on the idea of Boids from [15].

The final control vector from [1] has the following form:

$$\vec{f} = \vec{p} + \vec{n} + \vec{c}, \tag{2.1}$$

where $\vec{f}$ represents the final control vector and $\vec{p}, \vec{n}, \vec{c}$ are the proximal, navigation, and collision vectors respectively.

the navigation vector to the goal location was constructed according to the following equations:

$$\vec{n} = k_n \phi(\mu_d) \vec{x}^n, \tag{2.2}$$

$$\phi(\mu_d) = \begin{cases} 1 - (\frac{\mu_d}{d_{max}})^2, & \text{if } 0 \leq \mu_d \leq d_{max} \\ 0, & \text{if } d_{max} < \mu_d \end{cases} \tag{2.3}$$

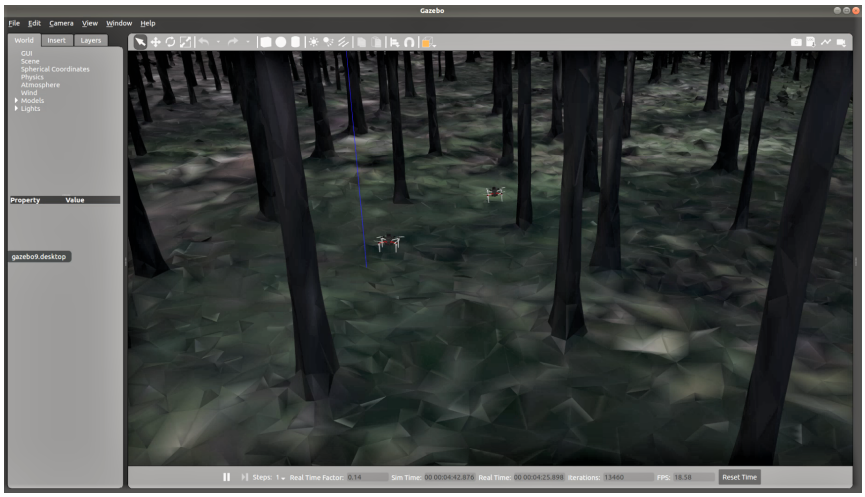


Figure 2.2: Running simulation within the Gazebo simulator.

where k_n is a gain of the navigation vector, $\vec{x}^n$ is the relative position of the closest point in the path, and $\phi(\mu_d)$ is used to regulate the distance of the agent from the rest of the group.

Obstacle collision avoidance vector had to follow these rules:

$$\vec{c} = \frac{1}{N_c} \sum_{i=1}^{N_o} \Phi(||\vec{x}_i^o||)\vec{x}_i^o, \tag{2.4}$$

$$\Phi(\vec{x}_i^o) = \begin{cases} k_o(\frac{1}{\sqrt{||\vec{x}_i^o||}} - \frac{1}{\sqrt{d_r}}), & \text{if } 0 \leq ||\vec{x}_i^o|| \leq d_r \\ 0, & \text{if } d_r < ||\vec{x}_i^o|| \end{cases} \tag{2.5}$$

where N_o is the total number of detected obstacles and $\vec{x}_i^o$ is the relative position of the currently detected i^{th} obstacle. N_c is the number of obstacles with a non-zero value for $\Phi(\vec{x}_i^o)$.

The used proximal control vector makes sure that agents are flying in a swarm formation but not too close to collide. Equations expressing the proximal control are:

$$\vec{p} = \frac{1}{N_a} \sum_{i=1}^{N_a+N_r} (a(||\vec{x}_i^a||)\vec{x}_i^a) + \frac{1}{N_r} \sum_{i=1}^{N_a+N_r} (r(||\vec{x}_i^a||)\vec{x}_i^a), \tag{2.6}$$

$$a(||\vec{x}_i^a||) = \begin{cases} 0, & \text{if } 0 \leq ||\vec{x}_i^a|| \leq d_0 \\ k_a(||\vec{x}_i^a|| - d_0)^2, & \text{if } d_0 < ||\vec{x}_i^a|| \leq d_f \\ \lambda_1 \tan^{-1}(||\vec{x}_i^a|| - 0.9d_f), & \text{if } d_f < ||\vec{x}_i^a|| \end{cases} \tag{2.7}$$

$$r(||\vec{x}_i^a||) = \begin{cases} -\lambda_2(\frac{1}{\sqrt{||\vec{x}_i^a||}} - \frac{1}{\sqrt{d_0}}), & \text{if } 0 < ||\vec{x}_i^a|| \leq d_c \\ -k_r(||\vec{x}_i^a|| - d_0)^2, & \text{if } d_c < ||\vec{x}_i^a|| \leq d_0 \\ 0, & \text{if } d_0 < ||\vec{x}_i^a|| \end{cases} \tag{2.8}$$

where $\vec{x}_i^a$ is the relative position vector of the currently observed i^{th} agent and $a(\vec{x}_i^a)$, $a(\vec{x}_i^a)$ are the attraction and repulsion functions respectively.

Modification of equations mentioned above is a part of this thesis and it is explained further in sections 4.2 and 4.3.

2.4 Frames of reference

Three types of reference frames are used in this thesis.

1. world frame - *world* coordinate frame has its center and orientation coinciding with the starting position and orientation of relevant UAV.
2. rotational frame - The *rotational* frame is a coordinate frame centered at the center of gravity of the relevant UAV and the xy-plane is parallel to xy-plane of the world frame. Orientation is the same as orientation of the UAV. This reference frame is used for most coordinate related operations.
3. body frame - *body* coordinate frame is similar to the *rotational* frame as it has its origin at the center of gravity of the UAV but its z-axis is parallel to the thrust force produced by the propellers.

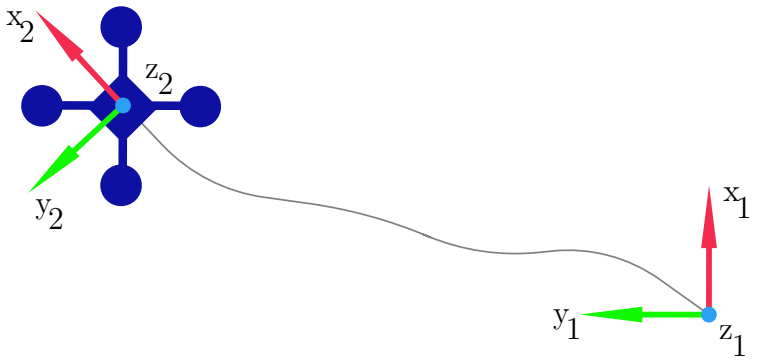


Figure 2.3: Coordinate system 1 describes the *world* frame and system 2 describes the *rotational* or *body* frame as those are identical in this situation.

Chapter 3

Localization of swarm neighbors

An obvious choice for a localization system for outdoor deployment of UAVs is a navigation satellite system, e.g. GPS (global positioning system). GPS performs well in a situation where the receiver has a high chance of successfully receiving and processing the signal. In cases like underground or tunnels, the GPS signal may be completely missing. In areas like a forest, the GPS signal may be insufficient or completely lacking. Based on prior experience the GPS itself doesn't provide enough accuracy to operate in forest-like areas. Therefore, this task requires localization that is independent of GPS.

For the agents to navigate using Boids-like mechanics it is necessary to relatively localize agents in their immediate neighborhood. That means there is a need for transformation between reference frames of nearby agents. Such transformation consists of translation and rotation. The proposed system is designed so that the swarm agents operate in 2D formation in a forest environment. Because of this, the agents have to work with translation in the x and y axes, and the rotation is considered around the z-axis that is perpendicular to the x and y axes. All these axes are expressed in the rotational frame (section 2.4) of the agent.

This work aims to introduce a relative localization system independent of any global localization system like GPS that should be scalable with minimal explicit inter-agent communication and sensory usage. First, the ICP (Iterative closest point) algorithm will be described and its usage in the proposed localization system will be explained. Next, the obstacle handling and representation will be discussed. Finally, the ICP algorithm will be improved using the FRMSD (Fractional root mean square distance).

3.1 ICP Algorithm

Iterative closest point is an algorithm used in various applications such as scan matching or path planning. Given two scans with an overlapping segment, the iterative closest

point is searching for the optimal way those scans overlap. UAVs are transmitting positions of near obstacles in the UAV's respective body frame (section 2.4) to other UAVs. Each UAV then compares received obstacle positions with detected obstacle positions and based on that estimates rotation and translation between rotational frames (section 2.4) of respective UAVs.

The ICP algorithm is represented by the pseudocode 1 where E is the error representing the amount of misalignment of the two scans, S and R are the points representing the source and the reference scans that ought to be compared, P are the paired points between S and R, $Tra \in \mathbb{R}^{2\times1}$ is the translation vector and $Rot \in \mathbb{R}^{2\times2}$ is the rotation matrix, *threshold* is acceptable error value and *iter* is the maximum number of iterations the algorithm can run.

Algorithm 1 Iterative Closest Point pseudocode

```
while E > threshold and i < iter do
    P ← pairing(S,R)
    [Tra,Rot] ← estimateTransform(P)
    S ← tranSource(S,Tra,Rot)
    E ← calcError(S,R)
    i++;
end while
```

The **pairing**(S,R) procedure is in most cases the biggest time consumer within the ICP algorithm. Source and Reference sets of points are inputs of the pairing procedure. The goal is to make pairs consisting always of one point from the Source set and one point from the Reference set. The points are paired so that for a given point a_S from the Source set a point a_R from the Reference set is chosen that is thought to be the correct placement of a_S in the Reference scan. To determine a correspondence between two points the Euclidean distance is mostly used. Comparing point distances is computationally demanding. The point a_R becomes paired with a_S if it is closest to a_S from the Reference set. In an ideal case, this process produces a set of pairs of truly corresponding points.

estimateTransform(P) is the crucial part of ICP. The paired points are used to compute the ideal translation and rotation needed to align both scans. Translation vector Tra is actually in many cases calculated even before the points are paired. It is done by calculating the center of mass of set S and R and shifting all the points in S by the vector

between the two centers of mass. This process can be described by the following equations:

$$\vec{c}_S = \frac{1}{N}\sum_{i=1}^{N}\vec{a}_i \qquad \vec{a}_i \in S, \tag{3.1}$$

$$\vec{c}_R = \frac{1}{N}\sum_{i=1}^{N}\vec{a}_i \qquad \vec{a}_i \in R, \tag{3.2}$$

$$Tra = \vec{c}_S - \vec{c}_R, \tag{3.3}$$

$$\vec{a'}_i = \vec{a}_i - Tra \qquad i = 1, 2, ..., N; \quad \vec{a}_i \in S; \quad \vec{a'}_i \in S', \tag{3.4}$$

where $\vec{c}_S$ and $\vec{c}_R$ are the centers of mass of set S and R respectively and N is the number of points in the set.

When the centers of mass are identical the goal is to minimize the distance between corresponding points only by rotating the S' set. There are various ways of computing the optimal rotation. In this thesis, the SVD (Singular value decomposition) (3.10) decomposition was used for this purpose. The method of computing the optimal rotation with the use of SVD can be summarized as:

$$\mathbf{R} = [\vec{a}_{r1}, \vec{a}_{r2}, ..., \vec{a}_{ri}], \tag{3.5}$$

$$\vec{a}_{ri} = [x_{ri}, y_{ri}]^T, \tag{3.6}$$

$$\mathbf{S'} = [\vec{a}_{s'1}, \vec{a}_{s'2}, ..., \vec{a}_{s'i}], \tag{3.7}$$

$$\vec{a}_{s'i} = [x_{s'i}, y_{s'i}]^T, \tag{3.8}$$

$$\mathbf{W} = \mathbf{R}\mathbf{S'}^T, \tag{3.9}$$

$$\mathbf{W} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T, \tag{3.10}$$

$$Rot = \mathbf{U}\mathbf{V}^T, \tag{3.11}$$

where $\mathbf{U}$ and $\mathbf{V}$ are matrices containing left-singular vectors and right-singular vectors of $\mathbf{W}$ respectively and the diagonal elements of $\mathbf{\Sigma}$ are the singular values of $\mathbf{W}$.

In some cases, there is the possibility that the determinant of the *Rot* matrix equals -1. Then the matrix is a reflection matrix and not a rotation matrix. To compensate for this issue the last column of the matrix $\mathbf{V}$ has to be multiplied by -1.

In **tranSource**(S,Tra,Rot) the rotation and translation from the previous step are applied to the Source set of points:

$$\vec{a}_i = Rot(\vec{a}_i - \vec{c}_S) + \vec{c}_R \qquad i = 1, 2, ..., N; \quad \vec{a}_i \in S. \tag{3.12}$$

Error of translation and rotation is calculated in **calcError**(S,R). A sum of squared differences between paired points after transformation can be used as a valid metric for representing the measure of alignment of given scans. The ICP algorithm stops after some given number of allowed iterations or when the error is small enough or when the error in step $k+1$ is higher than in the previous step k.

ICP algorithm is dependent on the first estimate. Thus, the error of translation and rotation at the initialization of the algorithm is not desirable to be significant. This condition doesn't have to apply in cases where corresponding points in both sets are already known but this is mostly not the case. If the difference in alignment is too significant at the beginning of the algorithm, the pairing of corresponding points cannot be carried out well enough and the algorithm is not likely to find the optimal solution.

3.2 Obstacles

This work describes navigation in a forest-like environment. As mentioned before the agents are considered to be operating in the 2D plane above the forest vegetation and beneath the tree crowns. This eliminates the danger of colliding with small branches that are hard to detect as well as improves obstacle detection. The obstacles in this environment can be represented as tall cylindrical structures. When assuming the 2D operational space the obstacles can be found as a 2D projection of the intersection of a cylinder and xy-plane of agents body frame (section 2.4). Further, the projection is replaced by a circle. Therefore, the circular shape of all obstacles can be assumed.

The RPLIDAR-A3 laser scanner was used as the sensor to detect surrounding obstacles. The sensor is sufficient enough considering the mentioned premises of navigation through 2D space. A point cloud is the output of the laser scanner consisting of up to 750 points 360 around the agent. This sensor is the only source of data for the ICP algorithm in the proposed solution. The number of points in the point cloud is a critical parameter for the most computationally demanding step of the ICP, the pairing of corresponding points. More points lead to significantly longer processing times. This is very important as this way of localization is supposed to be working with groups of agents. It follows that the algorithm will have to compare multiple scans. Shouldn't this process be fast it might not be able to operate in real-time. Therefore, the following approach of grouping points in the point cloud has been chosen.

Firstly, the points from the laser scanner point cloud have to be divided into groups. Each group consists of points that belong to the same obstacle. Assuming the obstacle shape to be a circle, points in a group will always be a section of this circle. The circle section shouldn't introduce any sudden large changes in depth of the scanned points as there are no sharp edges. Using this logic the points can be grouped accordingly as shown in the following pseudocode 2.

Grouping the points based on depth difference alone does not solve the problem because it only sorts the points but still keeps all of them. The next step is to represent every group by fewer points than it contains. It appears that every group can be well represented using a single point. Assuming the group of points is creating a circle section then circle fitting using the least squares method can be utilized. This method solves the problem in the rotational frame from section 2.4 by finding the best parameters x_0, y_0, r_0

Algorithm 2 Obstacle identification pseudocode

```
while not all points processed do
    repeat
        p ← next unprocessed point
        Obst.append(p)
        if (next_point == False) then break
        end if
        dist ← distance(p, next_point)
    until dist < 1
    Groups.append(Obst)
    Obst.clear()
end while
```

where x_0 and y_0 are the x and y coordinates of the circle center and r_0 is the radius of the given circle. Such a circle minimizes the error expressed as the distance between the circle and the points from the group. System of linear equations describing a circle can be expressed as:

$$2xx_0 + 2yy_0 + (r_0^2 - x_0^2 - y_0^2) = x^2 + y^2, \tag{3.13}$$

where $\vec{x}$ and $\vec{y}$ are the x and y coordinates respectively of points in a group. Equations in (3.13) can be written in a matrix equation

$$\mathbf{A}\vec{c} = \vec{b},$$

where matrix $\mathbf{A}$ contains x and y coordinates of the grouped points, vector $\vec{c}$ consists of the unknown variables x_0, y_0, and r_0, vector $\vec{b}$ includes the right side values.

$$\mathbf{A} = \begin{bmatrix} 2x_1 & 2y_1 & 1 \\ ... & ... & ... \\ 2x_n & 2y_n & 1 \end{bmatrix},$$
$$\vec{b} = \begin{bmatrix} x_1^2 + y_1^2 \\ ... \\ x_n^2 + y_n^2 \end{bmatrix},$$
$$\vec{c} = \begin{bmatrix} x_0 \\ y_0 \\ (r_0^2 - x_0^2 - y_0^2) \end{bmatrix}.$$

The following equation has to be solved in order to acquire the optimal fitting solution.

$$\mathbf{A}^T\mathbf{A}\vec{x} = \mathbf{A}^T\vec{b} \tag{3.14}$$
$$\vec{x} = (\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T\vec{b} \tag{3.15}$$

It is necessary to filter out groups of points that contain less than 3 points in total before applying the least squares method. The solution would not be unique if the number of points was lower than 3, since 3 parameters are unknown. In such a case, the group may be incorrectly represented.

The approach of circle fitting lowers the number of points to be paired and matched in ICP from an initial 750 to approximately 20 depending on the number of obstacles in the area of interest. Therefore significantly reducing the time needed to execute this computationally demanding step of the ICP. Moreover, this method also provides the same representation of the obstacle from different viewing angles as it computes the center of the obstacle and doesn't only consider the closest points on it. This makes the ICP algorithm independent of the angle of view. Thus it brings more reliable localization techniques. Using this method also introduces an improvement in transmission. When trying to minimize the amount of transmitted data the agents now have to share only lower tens (Figure 3.1) of obstacle coordinates instead of the previous entire 750 coordinate point cloud. In conclusion, the method of obstacle representation is essential for this localization system to work.

Figure 3.1: Red points are data from the laser scanner, whereas the green dot is the circle center estimation. Two agents are found in the center obtaining this obstacle laser scan.

3.3 FRMSD ICP

Usually, when employed for localization or odometry purposes, the ICP algorithm is used to align scans originating from the same agent. In the proposed system the ICP is

used to match two scans from different agents which introduces additional requirements. The main difference is that the size of the translation between the two scans is generally smaller when used on a single agent assuming a high enough scan rate. On the other hand, when used on scans originating from different agents, translation size is bigger because the agents find themselves several meters apart when the scans are taken. Thus the proposed method has to be robust and reliable for the comparison of two scans with diverse origins.

The proposed usage of ICP algorithm further brings a bigger number of outliers because the agents are further away and therefore their scans have less overlap. An outlier is a point in one of the scans that does not have a real corresponding point in the other scan. These are then paired wrongly with falsely corresponding points and thus negatively influencing the ICP algorithm. Numerous methods for outlier rejection exist. In this thesis, the Iterative Closest Point with Fractional Root Mean Square Distance (FRMSD) outlier rejection is used (Figure 3.2).

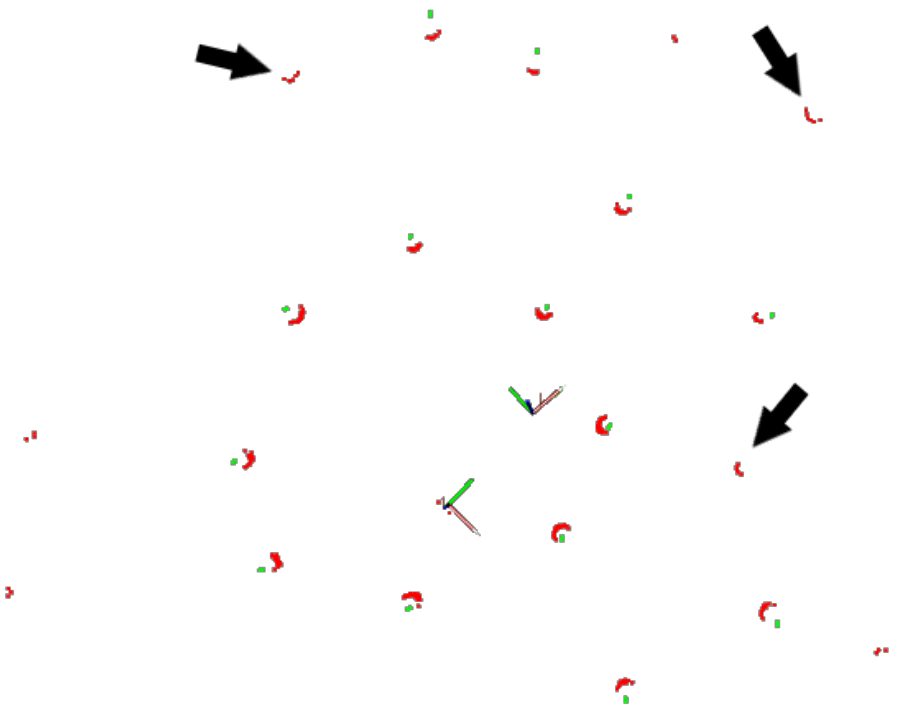


Figure 3.2: Black arrows point at obstacles, which represent outliers, that are visible only from one of the two agents and therefore are not marked by the green estimation. Outliers are not included in the scan comparison. The same approach is applied for neighbors which are evaluated as an obstacle by ICP.

Definition of FRMSD is as follows

$$FRMSD = \frac{1}{f^{\lambda}} \sqrt{\frac{1}{|D_f|} \sum_{p \in D_f} d^2}, \tag{3.16}$$

$$d = ||p - \mu(p)||, \tag{3.17}$$

where D is the own point set, p is a point from D, $\mu(p)$ marks the corresponding point in the received point set, $|D_f| = f|D|$, $f \in [0, 1]$ and $\lambda = 1.3$ (generally for 2D) as explained in [16].

Few additional steps must be added to implement the FRMSD into the ICP algorithm. Firstly, the set D is ordered starting with the point with the smallest distance to its corresponding point (3.17) and ending with the point with the biggest distance to its corresponding point. Then the first i points are chosen into the D_f for which $FRMSD_i$ is the smallest. It can be decided with more precision which pairs of points will be used in the calculation of the transformation because of the FRMSD extension to the ICP algorithm. Others will be discarded as pairs containing an outlier. The new set D_f is then used to compute the transformation.

The parameter λ can be modified during different stages of the algorithm based on the rate of convergence. It can be understood so that if the λ parameter is too small the FRMSD can classify even correctly corresponding points as outliers [16].

Algorithm 3 Outlier rejection FRMSD pseudocode

```
i = 0
best_i = 0
best_frmsd = 0
D ← QuickSort(D)
while i < |D| do
    i++
    D_f ← first i points from D
    if FRMSD(D_f) ¡ best_frmsd then
        best_i = i
        best_frmsd = FRMSD(D_f)
    end if
end while
D_f ← first best_i points from D
```

The FRMSD based pair filtering (pseudocode 3) is then repeated in the ICP algorithm every time after the pairs of corresponding points are made.

Chapter 4

Swarm Controller

This chapter describes the control mechanisms for swarm navigation through the forest. The swarm controller in this thesis is inspired by the work of Craig Reynolds on the topic of Boids [15]. The Boids control mechanism was first introduced as a simulation of animal motion such as bird swarming behavior or fish schools[1]. Boids are referring to the swarm particles. There are three basic steering behaviors for the particles: separation, alignment, and cohesion. The combination of those behaviors determines the motion of the particle. Extensions and variants of this method were already used for robotic swarm applications [24], [1]. This thesis introduces a Boid-based controller with obstacle avoidance and relative localization of neighboring swarm particles (section 3).

4.1 Control vector definition

We introduce a combining procedure needed to obtain the final control vector (pseudocode 4). The control vector has the meaning of UAV's desired velocity and it serves as a reference for the *SpeedTracker* [25]. First, the cohesion vector $\vec{v}_c$ and the separation vector $\vec{v}_s$ are computed. Then the navigation vector is calculated. Vectors $\vec{v}_c, \vec{v}_s, \vec{v}_n$ are summed into the vector $\vec{p}$ that is used to calculate the obstacle avoidance vector. Finally, $\vec{v}_c$, $\vec{v}_s$, $\vec{v}_o$ are used to recalculate the navigation vector. All the four vectors $\vec{v}_c, \vec{v}_s, \vec{v}_o, \vec{v}_n$ are then summed and the final control vector is scaled properly.

$$\vec{v} = k_c \cdot \vec{v}_c + k_s \cdot \vec{v}_s + k_o \cdot \vec{v}_o + k_n \cdot \vec{v}_n, \tag{4.1}$$

$$\vec{v}_f = \begin{cases} \frac{\vec{v}}{||\vec{v}||} \cdot k_f, & \text{if } ||\vec{v}_f|| > 1 \\ \vec{v} \cdot k_f, & \text{otherwise} \end{cases} \tag{4.2}$$

where $\vec{v}_f$ is the final control vector, the k_f is a parameter used to adjust the desired speed, k_c, k_s, k_o, and k_n are parameters to adjust their respective vectors.

[1] https://www.red3d.com/cwr/boids/

Algorithm 4 Control vector computation

while True **do**
  neighb_agents ← **getNeighbLocation**()
  obstacles ← **getObstacleLocation**()
  cohesion_vec ← **calcCohesionVec**(*neighb_agents*)
  separation_vec ← **calcSeparationVec**(*neighb_agents*)
  navigation_vec ← **calcNavigationVec**(*cohesion_vec, separation_vec*)
  p ← *cohesion_vec* + *separation_vec* + *avoidance_vec*
  avoidance_vec ← **calcAvoidanceVec**(*obsatcles, p*)
  navigation_vec ← **calcNavigationVec**(*cohesion_vec, separation_vec, avoidance_vec*)
  final_vec ← $k_c \cdot$*cohesion_vec* + $k_s \cdot$*separation_vec* + $k_o \cdot$*avoidance_vec* + $k_n \cdot$*navigation_vec*
  if $(||final\_vec|| > 1)$ **then**
    final_vec ← $\frac{final\_vec}{||final\_vec||}$
  end if
  final_vec ← $final\_vec \cdot k_f$
end while

4.2 Cohesion

The purpose of the cohesion vector is to hold the agents together so that they act as a group rather than individual units. To calculate the cohesion vector, all the neighboring agents' relative position vectors are summed and divided by the number that is by one higher than the count of neighboring agents. This will provide a "center of mass" of near agents.

$$\vec{c} = \frac{1}{N+1} \sum_{i=1}^{N} \vec{x}_i, \tag{4.3}$$

where $\vec{c}$ is the center of mass of neighbors, N is the number of neighbors, and $\vec{x}_i$ are relative coordinates of the *i-th* agent.

If there is an obstacle nearby of the UAV only the closest neighboring agent's coordinates are used to calculate the center of mass (Figure 4.1). This helps to prevent a situation where an agent is dragged into an obstacle by the group. It gives temporarily more flexibility to the agent while still moving with the group. This means that even if the agent is still being pulled into the obstacle it is advantageous to use only one neighbor for calculation. Thus, the center of mass of neighbors gets closer to the agent. When the center is closer than the cohesion vector becomes weak quicker (Figure 4.2) which makes the avoidance vector more influential.

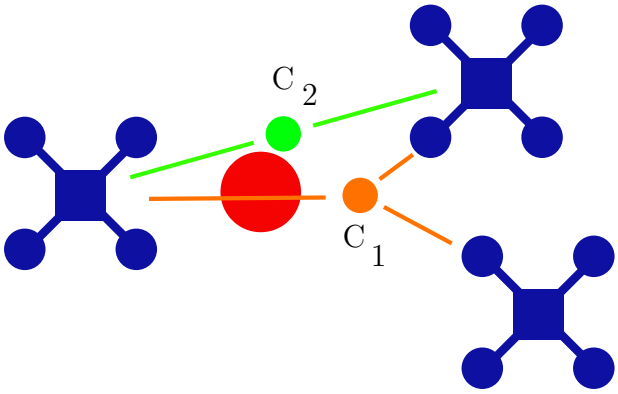


Figure 4.1: A situation where only the closest neighboring agent is used to calculate the center of mass. Agent on the left would be dragged straight into the obstacle (red) if the center c_1 (orange) was used. The center c_2 (green) provides more space for avoidance while still keeping up with the group.

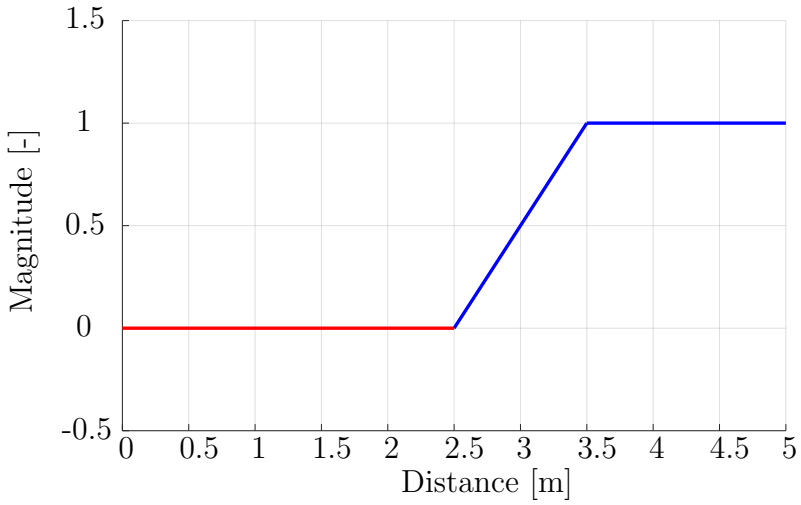


Figure 4.2: The relation between the distance of the center of mass and the magnitude of the cohesion vector. The blue part is where the vector is active and the red where the vector is unused.

The cohesion vector is then calculated using the following equations:

$$k = max(0, 1 - \frac{2.5}{||\vec{c}||}), \tag{4.4}$$

$$\vec{v}_c = k \cdot \vec{c}, \tag{4.5}$$

$$\vec{v}_c = \begin{cases} \frac{\vec{v}_c}{||\vec{v}_c||}, & \text{if } ||\vec{v}_c|| > 1 \\ \vec{v}_c, & \text{otherwise} \end{cases} \tag{4.6}$$

where the max function causes the cohesion vector $\vec{v}_c$ to be active only outside of a certain

radius. Thus, the vector $\vec{v}_c$ always acts as an attractive vector and never as a repulsive.

4.3 Separation

The separation vector has the collision avoidance function. It is supposed to prevent collisions between agents in the group. For this purpose, a distance-dependent approach has been chosen similar to [20]:

$$k_i = \begin{cases} -1, & \text{if } (1 - \frac{3}{||\vec{x}_i||}) < -1 \\ (1 - \frac{3}{||\vec{x}_i||}), & \text{otherwise} \end{cases} \tag{4.7}$$

$$\vec{v}_i = \begin{cases} \begin{bmatrix} 0 \\ 0 \end{bmatrix}, & \text{if } ||x_i|| > 2 \\ k_i \cdot \frac{\vec{x}_i}{||\vec{x}_i||}, & \text{otherwise} \end{cases} \tag{4.8}$$

$$\vec{v}_s = \frac{1}{N} \sum_{i=1}^{N} \vec{v}_i, \tag{4.9}$$

where $\vec{x}_i$ are coordinates of the i-th agent expressed in the rotational frame (section 2.4) and N is the number of all agents used to calculate the separation vector $\vec{v}_s$.

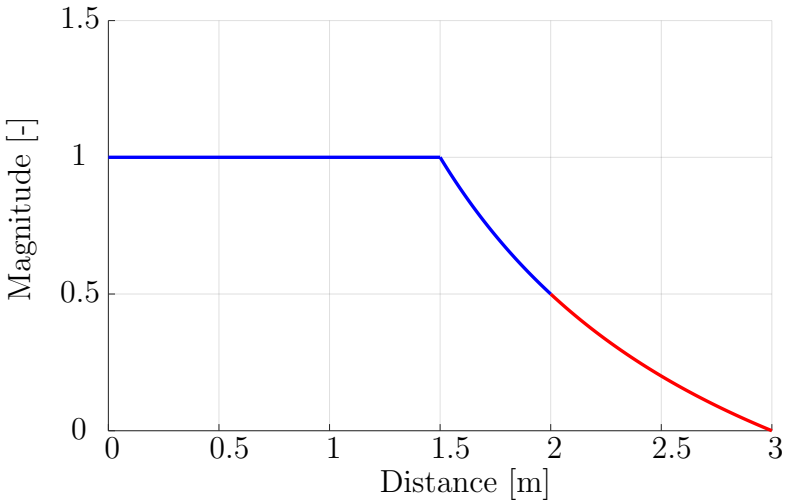


Figure 4.3: Relation between the distance of the neighboring agent and the magnitude of the separation vector computed for this agent. The blue part is where the vector is active and the red where the vector is unused.

The separation vector $\vec{v}_s$ is active only inside of a certain radius (Figure 4.3). The vector $\vec{v}_s$ always acts as a repulsive vector and never as an attractive. Moreover all neighbor

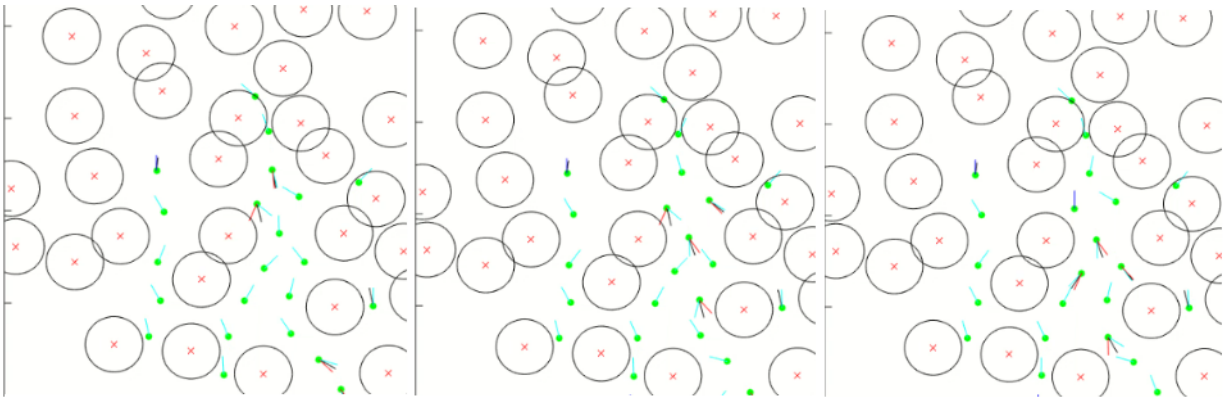


Figure 4.4: Pictures are chronologically from left to right. Red crosses are obstacles (with 1.5-meter black radius around them), green dots are agents. Red (separation) and black (final) lines depict the direction of given vectors that spread from the bottleneck.

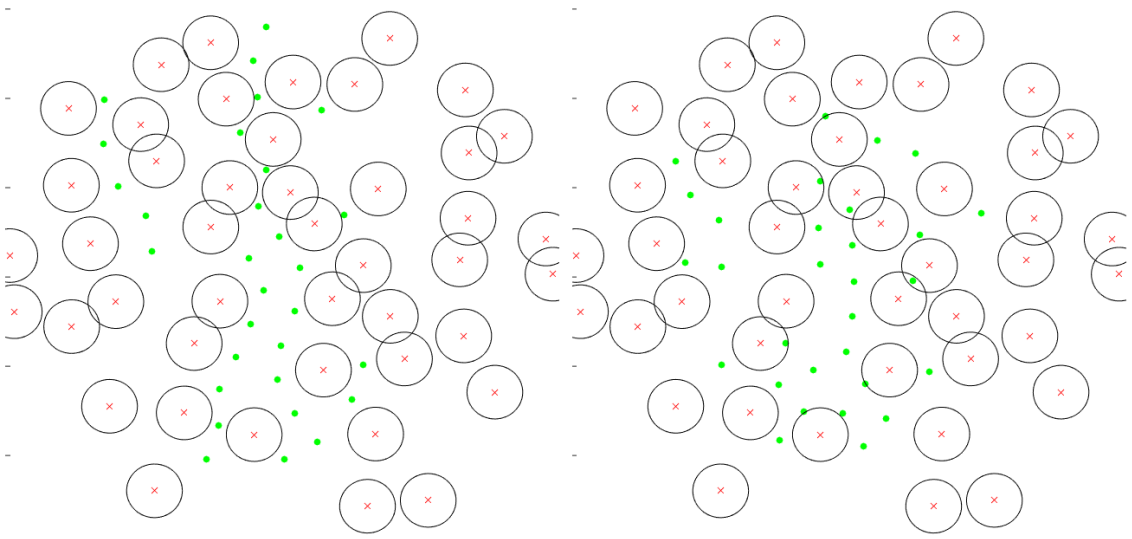


Figure 4.5: In the figure on the left, the agents are calculating the separation vector only for neighbors in front of them. In the right figure, agents are calculating the separation vector for all nearby neighbors. The left picture shows the emergent queuing behavior in the bottleneck whereas the right picture displays how the agents are forced closer to obstacles and the entire group seems more disorganized.

agents that are not situated in 180° circular section around the desired direction of movement are ignored when calculating $\vec{v}_s$. Large-scale simulations have shown that ignoring those agents actually prevents collision in some situations. A typical example is when the group is queuing into a bottleneck (Figure 4.5). If agents reacted with separation to all nearby neighbors it would cause pressure on the front of the group to move forward faster.

Therefore, it would provide less maneuvering space causing jam which forces agents into collisions with obstacles. Such behavior produces waves that spread from the front of the group to the back as shown in Figure 4.4.

4.4 Obstacle Avoidance

Another very important part of the control system is obstacle avoidance. The representation of obstacles is described in section 3.2. The obstacle avoidance is active in a certain radius around the agent (Figure 4.6) and considers always only the closest obstacle. The vector is orthogonal to the position vector of the obstacle [18]. It is calculated as a projection on orthogonal complement of the space spanned by the position vector:

$$\mathbf{P} = \mathbf{I} - \frac{\vec{y}}{||\vec{y}||}\left(\frac{\vec{y}}{||\vec{y}||}\right)^T, \tag{4.10}$$

$$\vec{v} = \mathbf{P}\vec{p}, \tag{4.11}$$

$$\vec{v}_o = \vec{v}\frac{1}{||\vec{v}||}\frac{2}{||\vec{y}||}, \tag{4.12}$$

$$\vec{v}_o = \begin{cases} \frac{\vec{v}_o}{||\vec{v}_o||}, & \text{if } ||\vec{v}_o|| > 1 \\ \vec{v}_o, & \text{otherwise} \end{cases} \tag{4.13}$$

where $\mathbf{P}$ is the projection matrix, $\vec{y}$ is the position vector of an obstacle, $\vec{v}_o$ is the obstacle avoidance vector and $\vec{p}$ is a vector representing the current desired direction of flight. Origin of $\vec{p}$ is described in section 4.1.

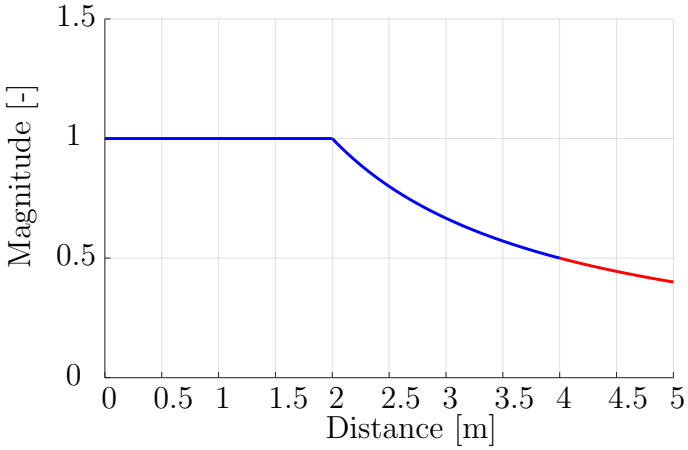


Figure 4.6: Relation between the distance of an obstacle and the magnitude of the obstacle avoidance vector. The blue part is where the vector is active and the red where the vector is unused.

Obstacles that are situated more than 135° from the current desired direction of movement are ignored when calculating $\vec{v}_o$ (Figure 4.7). This allows the agent not to be dragged too long around an obstacle and also discards obstacles that are not obstructing the path.

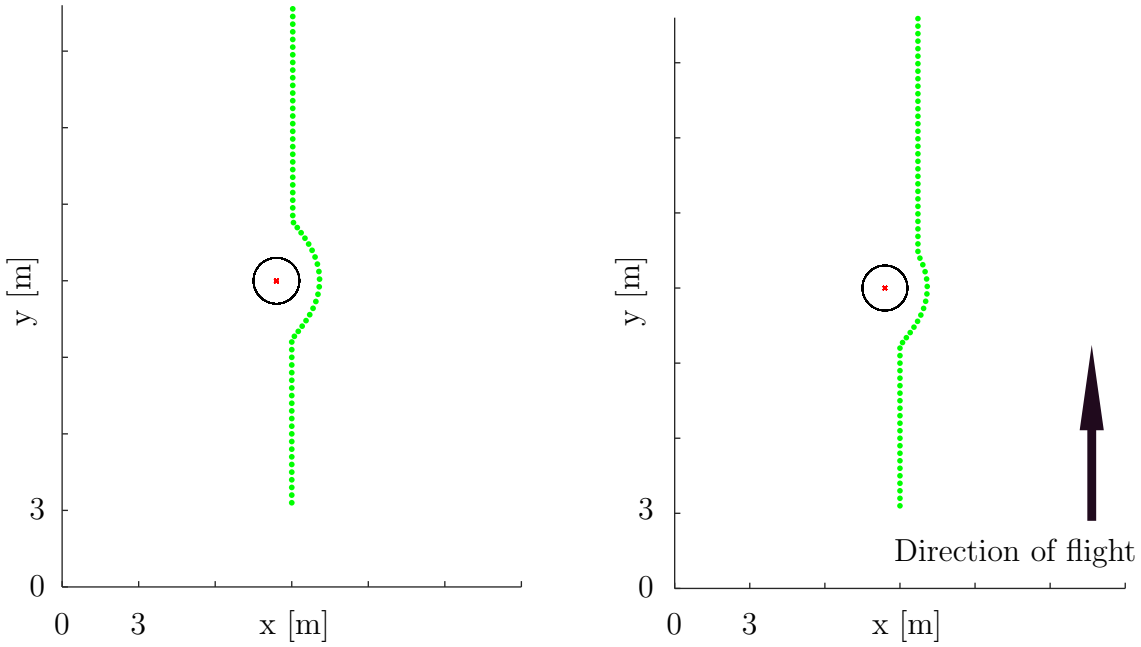


Figure 4.7: On the left is a path of an agent (green dots) considering past obstacle (red). On the right is the path when ignoring past obstacle. The path on the right is shorter without unnecessarily keeping close to the obstacle and turns. Also, it does not pull agents too close together when they approach the same obstacle from different sides.

4.5 Navigation and Alignment

If the group of UAVs has a certain location to reach and not only to hover around equilibrium it needs a way to navigate. Considering that the agents have their goal locations in the same direction it can be done in the following way. Heading alignment is ensured by computing the mean of all the neighbors' relative heading deviations and keeping up with the mean value (Figure 4.8).

$$\alpha = atan2\Big(\sum_{i=1}^{N} \sin\Theta_i, \sum_{i=1}^{N} \cos\Theta_i\Big), \tag{4.14}$$

where α is the mean, N is the number of considered agents, and Θ_i is the relative heading of i-th agent.

A virtual agent is added to ensure that the agents are moving towards the goal location. Goal location is defined in the world frame (section 2.4). The virtual agent has the same location as the agent itself and heading is set towards the goal position.

$$\Theta = atan2(g_y - p_y, g_x - p_x), \tag{4.15}$$

where Θ is the heading of the virtual agent, p_x, p_y, g_x and g_y are coordinates of the agent's position and final goal position respectively in the world frame. This is keeping the heading in the direction of the final goal position.

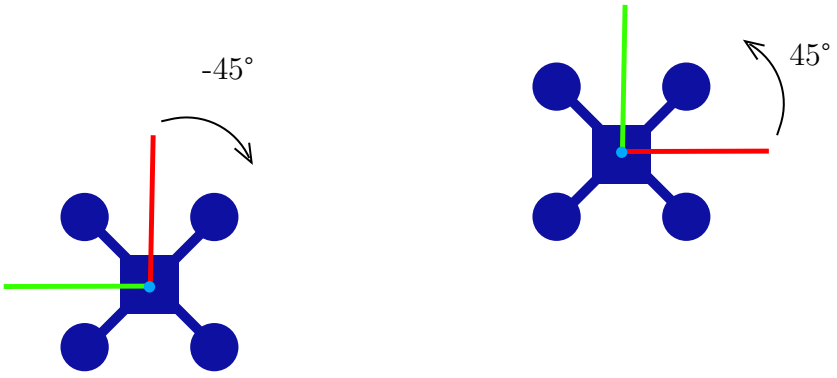


Figure 4.8: A situation where two agents have different relative heading. Green lines are x-axes of the agent's body frame, red lines are y-axes and light blue lines are z-axes. The arrows indicate the direction each agent will turn to keep the same relative heading.

Considering that the agents are maintaining similar relative heading. Then the navigation vector is a vector aligned with the x-axis of the body frame of the agent. This vector then propels the group in one common direction. The navigation vector is calculated based on all vectors introduced above. Magnitudes of all the previous vectors are summed and based on their combined magnitude the magnitude of the navigation vector is calculated (Figure 4.9).

$$k_n = ||\vec{v}_c|| + ||\vec{v}_s|| + ||\vec{v}_o||, \tag{4.16}$$

$$\vec{v}_n = \begin{cases} \begin{bmatrix} 0 \\ 0 \end{bmatrix}, & \text{if } k_n > 1 \\ \begin{bmatrix} 1 \\ 0 \end{bmatrix} \cdot (1 - k_n), & \text{otherwise} \end{cases} \tag{4.17}$$

where $\vec{v}_c, \vec{v}_s, \vec{v}_o$ is the cohesion, separation and obstacle avoidance vector respectively and $\vec{v}_n$ is the navigation vector.

The proposed approach of vector combining emphasizes the different priorities of individual vectors. The first priority is to prevent the group from falling apart and to

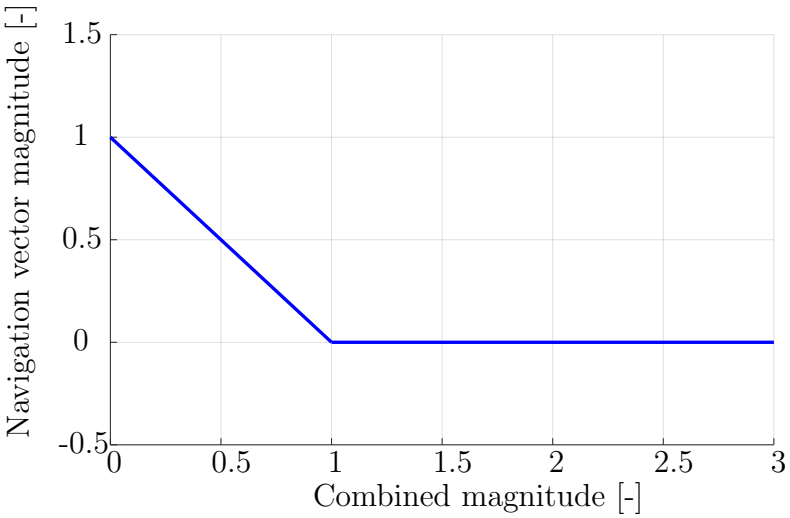


Figure 4.9: Relation between the combined magnitude of vectors used to compute the navigation vector and the magnitude of the navigation vector.

prevent collisions between the agents and obstacles. For example, the obstacle avoidance vector will have a greater magnitude when an agent needs to avoid a close obstacle. That will stop the navigation vector from demanding progress and provides more space for a steady avoidance maneuver. After the prioritized situation is resolved the navigation vector starts acting again.

4.6 Comparison

The differences between the control approach proposed in this thesis and the control approach specified in [1] are mentioned in this section.

First of all, there are major differences in the navigation vector (section 4.5). The navigation vector in [1] is a vector pointing to the closest point in the desired path of the UAV. This path is generated using a modified A* algorithm. This thesis proposes a navigation vector that is aligned with the x-axis of the body frame. Navigation is ensured using this vector together with heading alignment towards the goal location.

Another difference between the two approaches is scaling functions for the vectors. Scaling functions in [1] are designed to keep larger distances between UAVs within the swarm. The UVDAR system [17] is used for relative localization in [1]. The UVDAR system is more reliable for larger distances. The ICP localization system (section 3) works better with closer proximity as the laser scans overlap more. Therefore, this thesis proposes equations allowing smaller distances between agents.

The difference in separation vector is that all neighbors are considered when calculating the separation vector in [1]. On the contrary, only neighbors in front of the UAV are

used to calculate the separation vector in this thesis as explained in section 4.3.

The obstacle avoidance vector in [1] is computed considering all nearby obstacles. The obstacles behind the UAV are ignored when computing obstacle avoidance vector in this thesis as explained in section 4.4. Moreover, obstacle avoidance vectors in this thesis are tangential to the obstacles whereas in [1] they are pointing directly away from the obstacle.

The order of computing the control vectors (section 4.1) and the way of computing the navigation vector (section 4.5) along with considering fewer neighbors in the computation of the cohesion vector while too close to an obstacle (section 4.2) are also different from the ones used in [1]. The approach in this thesis was chosen to prioritize the obstacle avoidance vector making the flight safer.

Chapter 5

Database of agents

The ICP algorithm introduced in section 3.1 can be used to match two laser scans and determine the relative positions of their sources. A simple approach where every agent runs the ICP algorithm every time on every scan it receives is viable but has a weak scaling potential. In the worst-case scenario considering every agent is able to a receive scan from every other agent then every agent also has to run the ICP on every scan received. This approach is more computationally demanding for large-scale swarms. Therefore, we introduce the solution for reduction of computational demands of ICP algorithm together with process of neighbor information handling.

5.1 ICP First estimates

The main goal of the proposed approach is to make the localization system viable for large groups of agents by minimizing the time spent on computing the ICP algorithm. Two ways have been chosen to achieve this. Firstly, it is unnecessary to try to localize an agent that is too far to be localized by the ICP. Secondly, it is desirable to use the agent's last known position as a starting estimate for the ICP algorithm.

As mentioned in section 3.1, ICP requires a decent starting estimate to work properly. In this thesis, the problem of the first estimation is solved by iteratively calling ICP and comparing the obtained FRMSD. Thus, if there is not any available starting estimate to initiate the ICP algorithm, the ICP algorithm will be executed iteratively at every point of a grid that spans 6 meters to each direction from the agent. The grid consists of 441 points at which the ICP evaluates all relative headings (Figure 5.1). The best position is chosen as the first estimate. This is possible because the goal is assumed to be in a similar direction. Moreover, the sparse point cloud representation makes the ICP algorithm run fast enough (section 3.2).

Using only this way of estimating the initial position takes a relatively long time to compute. Therefore, the last known position is often used as the first estimate. We propose

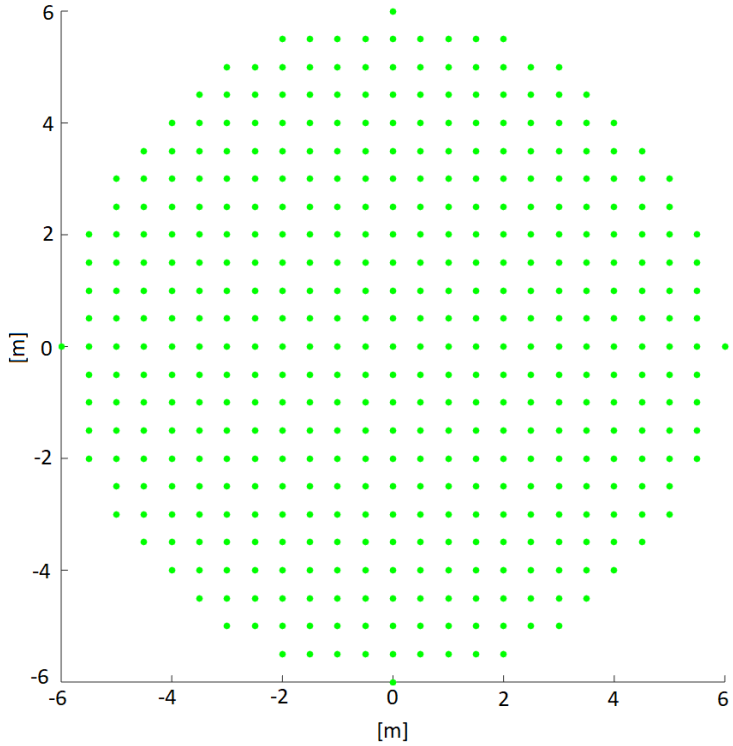


Figure 5.1: Green points around the agent are positions that are probed for the full starting estimate. The closest distance between two points is 0.5m.

to use 3 variations of the ICP to improve the computational demands of the localization system (Figure 5.2).

- Firstly, the initial ICP (iICP) is used when a new laser scan is received. It locates the source of the laser scan if possible and also creates an entry in the database. The iICP contains the full first estimate as described above and thus takes more time to be computed.
- Secondly, the short ICP (sICP). The sICP takes in the database entry and updates its values. It is not using the full starting estimate but already just the last known position. This makes it fast and is used most of the time.
- Lastly, the long ICP (lICP). Situations can occur when the position is incorrectly determined as a result of quick maneuver or other disturbing factors. In such cases, the sICP continues using the wrong position as the estimate which could lead to misplacement of the agent. It is difficult to determine when this happens. Therefore, regular checks take place after a given period of time to compensate for possible deviations. The lICP takes in the database entry and updates it using the full estimate every 5 seconds.

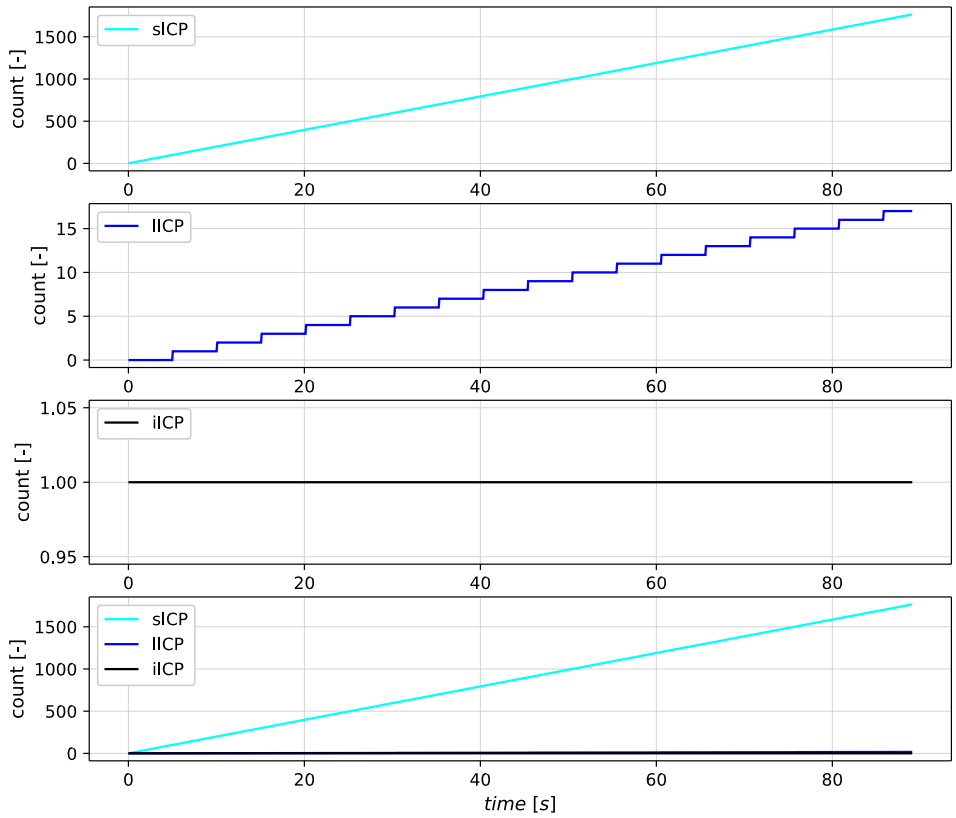


Figure 5.2: Data obtained from a short simulated flight of two agents.

A simulation was run with 2 UAVs using the Gazebo simulator (Figure 2.2). Graphs 5.2 show, which variant of ICP is used by one of the agents to keep track of the second one. The first graph shows the number of short ICPs used, the second one shows long ICPs and the third is the initial ICP. It shows that the iICP is called only once at the initialization and is never used again. The lICP is used every 5 seconds. The last graph presents all counts together for comparison. It is visible that the overwhelming majority of used ICPs are the least demanding sICP.

5.2 Agent entries

Above mentioned improvements require a certain way how an agent can distinguish between other individual agents in the group. However, it is preferred that all the agents in the group are interchangeable and not unique. This fact does not allow for any type of IDs

to be assigned to the individual agents because that would not make neighboring agents replaceable. Furthermore, the ID is another information that would have to be transmitted or communicated. Last but not least, adding a new agent to the group would be more complicated when merging two larger groups because IDs cannot be duplicated.

We propose a dynamic database consisting of entries about surrounding agents. The database is unique for each agent. Each agent makes an entry for every other agent that has been in contact with him. The entry holds important information about the agent. Unused entries can be discarded and new created. The laser scan is used as an ID for the entries. Considering the forest surrounding the group is unique then the laser scan is also unique for each agent. There can not be two agents exactly at the same location at the same time since the flocking operates in 2D and therefore there cannot be two agents with the same laser scan.

Algorithm 5 Database entry structure

```
struct uav {
e::Matrix2d rotation;
e::Vector2d translation;
ros::Time last_long;
vector<e::Vector2d> ID;
bool suspended;
ros::Time suspension_time;
ros::Time last_used;
bool chosen;
};
```

Pseudocode 5 shows the database entry for one agent. Each agent has such entry for every agent whose laser scan is received. Variables within the entry have the following purposes.

- **rotation** is 2D matrix describing the relative rotation between the headings of the two agents.
- **translation** is a 2D vector describing the relative translation between the two agents.
- **last_long** is a time information. It states the last time when ICP with the full starting estimate was used to update or create this entry.
- **ID** are coordinates representing the obstacles from the given laser scan.
- **suspended** is true if the agents location could not be determined the last time it was checked. It states whether the agent is in range of the ICP or not.
- **suspension_time** is a time information. It states the last time when the entry was classified as suspended.

- **last_used** is time information stating the last time the entry was updated in any way.
- **chosen** helps track active agents in the surroundings.

5.3 Entry handling

The database is changing with time. Entries are being updated, discarded, or added. Some are ready for use and some are suspended. The following algorithm 6 decides what happens with every entry.

Algorithm 6 Entry handling

```
for entry in database do
    entry.chosen = false
end for
for scan in received_scans do
    best_match = findBestMatch(scan, database)
    if best_match_frmsd <= 1 then
        database[best_match_index].chosen = true
    end if
    if best_match_frmsd > 1 then
        addNewEntry(scan, database, ref_scan)
        continue
    end if
    if best_match.suspended == true and best_match_suspension_time < 5 then
        best_match.last_used = timeNow()
        continue
    end if
    if best_match_lICP_time > 5 then
        lICP(best_match, ref_scan)
    else
        sICP(best_match, ref_scan)
    end if
    best_match.last_used = timeNow()
end for
```

First, the *.chosen* member of each entry is set to false as an initialization of the new iteration of the algorithm. Further, the following routine was repeated for every scan that was recently received by the agent.

findBestMatch(scan, database) takes in the received scan and the database. It applies ICP without the full starting estimate on the received scan and *.ID* member of

each entry. This step does not modify the database in any way. It finds the entry that together with the received scan achieves the lowest FRMSD.

The next step will be chosen based on the value of the last FRMSD of an entry. If the FRMSD is less or equal to 1 then the *.chosen* member of the entry is set to true.

addNewEntry(scan, database, ref_scan) is called if FRMSD is higher than 1, which indicates that the scan is out of range of the ICP. This value was acquired experimentally in the simulation. It takes in the received scan, database, and scan of the ego agent. It creates a new entry in the database for the received scan.

Next, the suspension status of the found entry is checked. If the entry is suspended for less than 5 seconds it will not be updated. This means that the agent is likely still out of range for the ICP.

lICP(best_match, ref_scan) or **sICP**(best_match, ref_scan) is used to update an entry if the entry is found with low enough FRMSD and is not suspended. The type of ICP is chosen based on the time since the last update with a full starting estimate.

In the end, the used entry is stamped with the time it was used. All entries that were used more than 0.5 seconds ago are discarded from the database.

Chapter 6

Simulation

The above mentioned components were put together and tested in simulations[1]. Simulations are carried out in realistic Gazebo simulator and using Matlab. Simulation scenario where the swarm has to navigate in a forest environment is testing both the localization of swarm neighbors presented in section 3 and section 5 as well as the proposed control approach (section 4). Gazebo is used for simulations that include UAV dynamics. Therefore, simulations in Gazebo are closer to the real world. Matlab is used for large-scale simulations that consist of tens of UAVs without dynamics. Matlab simulator is purely used to test the scalability of the proposed control system (section 4) and was designed specifically for the purposes of this thesis.

6.1 5 UAV - Gazebo

The first set of simulations was performed in the realistic Gazebo simulator. Five agents were used for this simulation. Their group goal was to navigate from around the point $(0, 0)$ to around the point $(30, 0)$ in a simulated forest environment. Data from one agent were used to present the results.

Graph 6.1 shows paths (dotted lines) of all agents in x, y coordinates. The blue path is the path of the agent that is specified by the triangle and whose data were used for presenting the results of the simulation. Black objects represent trees. The goal was to reach the point $(30, 0)$ as a group. The graph presents the swarm at three different points in time. The position of the group closest to the left represents the position after 17 seconds of the flight. The position in the middle depicts the group after 77 seconds from the start and the position on the right after 127 seconds. Trace (solid line) behind each agent shows its path during the previous 5 seconds.

The blue and the orange lines in the graph 6.2 represent the magnitudes of the navigation (section 4.5) and the obstacle avoidance (section 4.4) vector respectively. The

[1] http://mrs.felk.cvut.cz/krizek-2021-bp

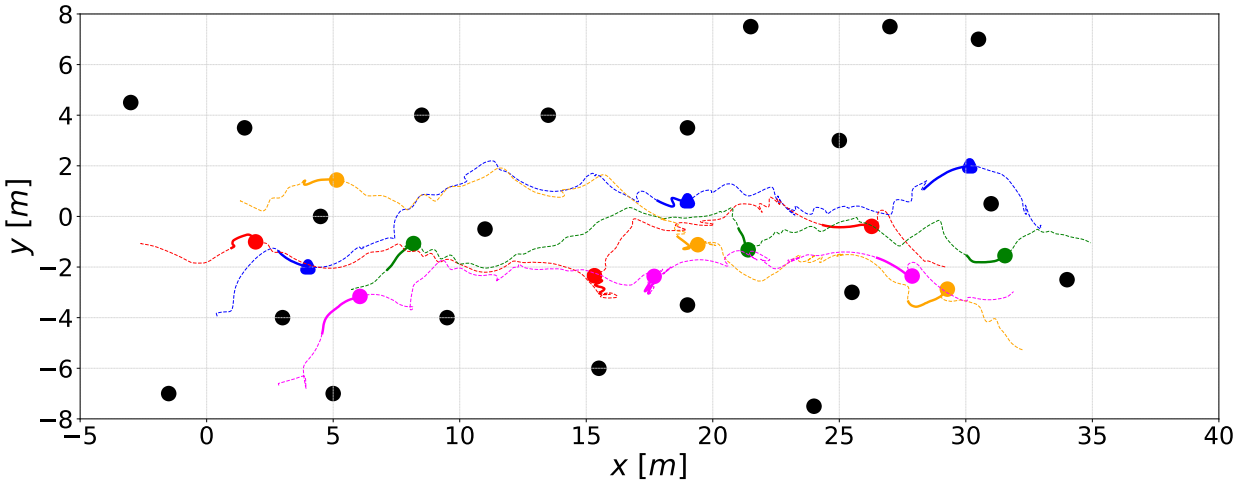


Figure 6.1: Agents (colored) depicted at three different times with their complete paths (dotted) during their navigation through obstacles (black).

navigation vector is filling in spots where the obstacle avoidance vector is less active as explained in section 4.5. This behavior produces almost mirror image and the differences are caused by the change in the activity of other vectors.

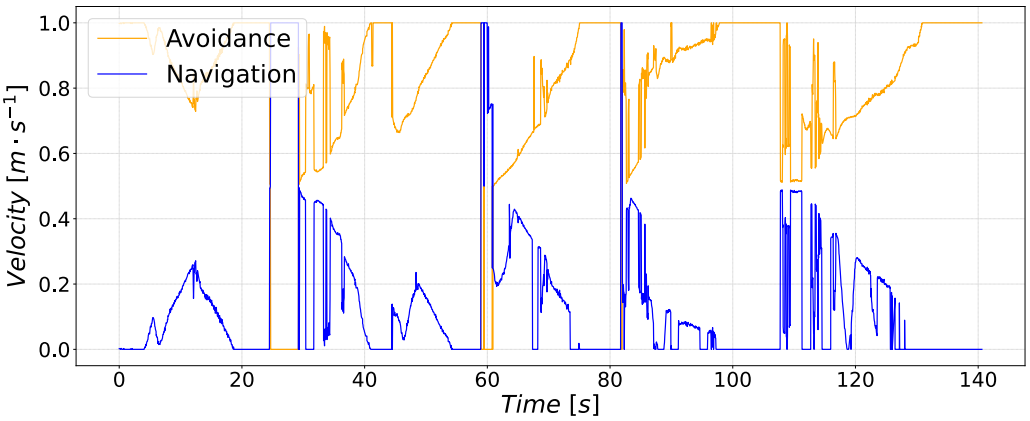


Figure 6.2: The navigation vector magnitude (blue) and obstacle avoidance vector magnitude (orange).

Connections between the presented data can be observed well around 127 *s* timestamp. In the graph with the closest agent distance (Figure 6.6) is a significant increase in distance to the closest neighbor. Evidence of this is also visible in the graph with the number of detected neighbors (Figure 6.7). The number of detected neighbors declines around this point which is indicating that the other agents are getting out of range of the ICP.

Followed by a quick increase in activity of the cohesion vector (Figure 6.3) that begins to force the agent back to the group. The situation is depicted in Figure 6.1 where the group on the right represents the situation at this timestamp. It is clear that the measured agent has increased its distance from the group and is the furthest from the center of mass and therefore is being pulled back to the group.

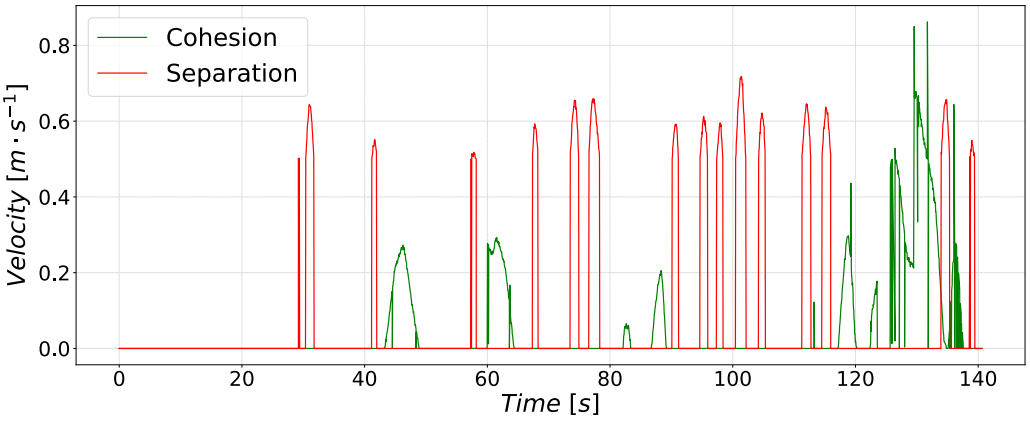


Figure 6.3: The cohesion vector magnitude (green) and separation vector magnitude (red). The two vectors do not work against each other but they alternate depending on the situation.

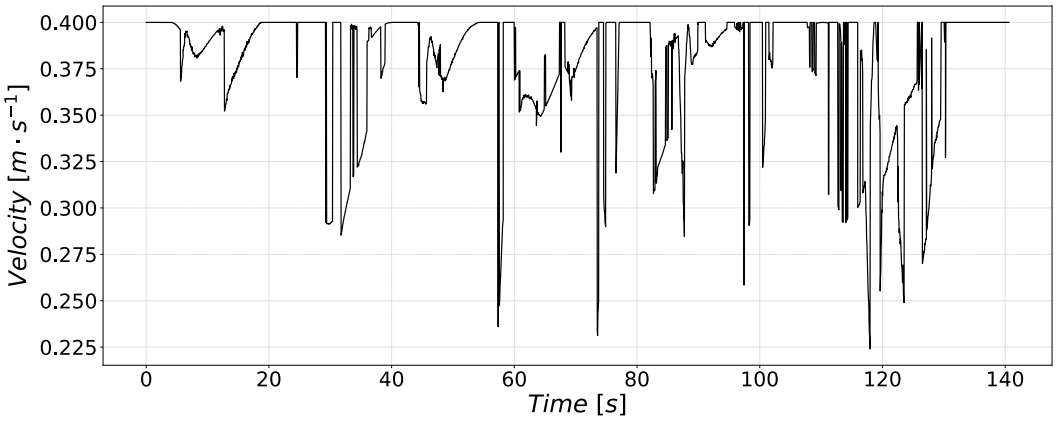


Figure 6.4: The final vector magnitude after combining all other components. The maximal magnitude of the final vector was set to 0.4.

Graph 6.5 shows that flight was without any agent-obstacle collision and the distance to obstacles was sufficiently big. Moreover, graph 6.6 shows that the flight was without any agent-agent collision and the distance between agents was sufficiently big as well. Peak

values in distance estimated by the ICP can be filtered by a low-pass filter. The average error between ground truth and ICP estimation is 0.052 m.

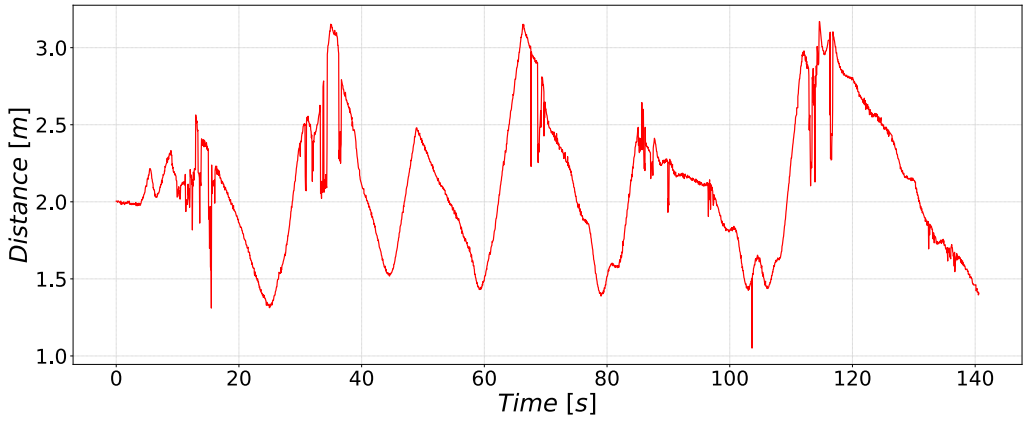


Figure 6.5: The shortest distance to the closest obstacle at any point in time.

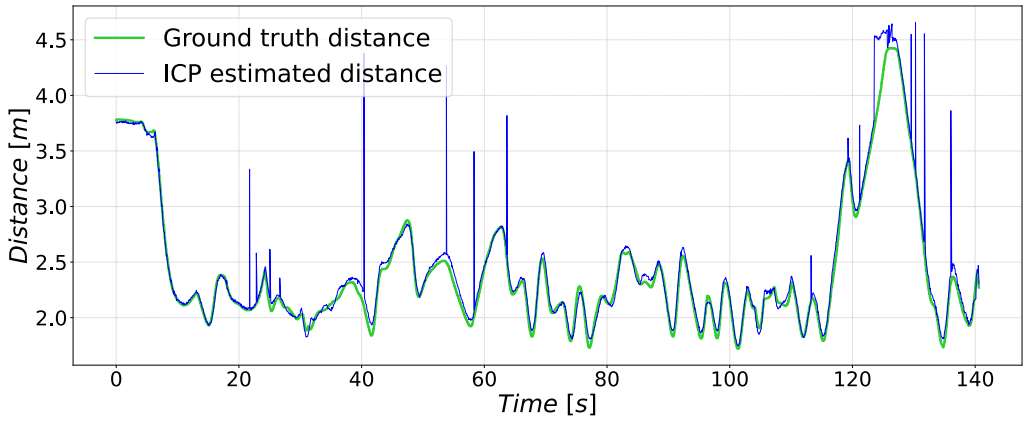


Figure 6.6: The shortest distance to the closest agent determined by the ICP (blue) and the ground truth (green).

Figure 6.7 presents the number of detected neighboring agents by the UAV during the flight. The correctness of this value depends not only on the laser scan of the ego agent but also on the laser scans provided by the other agents. Some neighboring agents might be balancing on the maximum range of the ICP. Therefore, quick changes between detected neighbors count can occur. But even if those short changes were all mistakes the one stable value is still active 89% of the time interval. This graph is the demonstration of entry handling introduced in section 5.3.

Distances agent-obstacle and agent-agent are important for the safety and reliability of the proposed system. Those distances from the simulation are presented in Figure 6.1

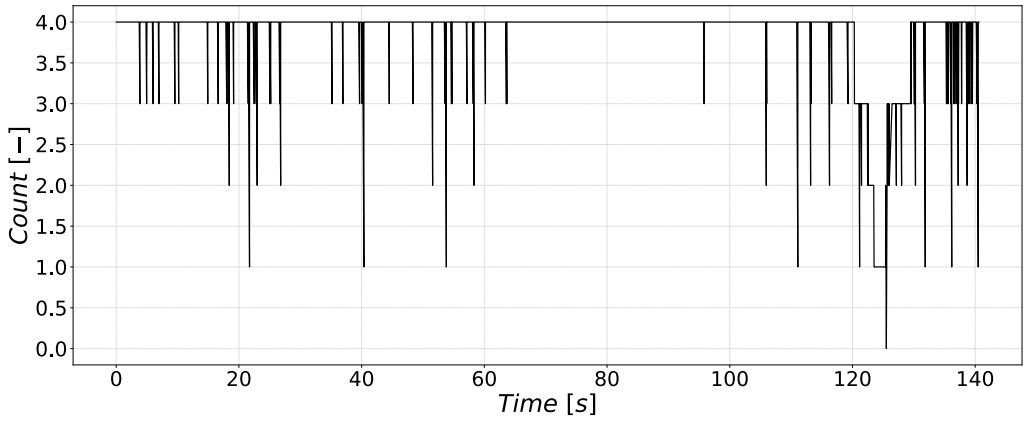


Figure 6.7: The number of detected neighbouring agents by the UAV during the flight.

along with their averages. Furthermore, the desired speed of the UAV can be seen in Table 6.4.

Agent	1	2	3	4	5	Average
Obstacle distance [m]	1.4	1.3	1.4	1.48	1.23	1.36
Agent distance [m]	1.35	1.52	1.57	1.45	1.4	1.45

Table 6.1: Individual minimal distances for each agent during the test flight.

6.2 7 UAV - Gazebo

Seven agents were used with the same goal as in the previous experiment in the second simulation. Moreover, another static UAV was placed close to the swarm to simulate system failure. The group behaved accordingly to the algorithm even when one UAV serves as a disturbance. Therefore, this case can be used as proof of decentralization and robustness of the system. Furthermore, interesting situation occurred during the same simulation. One of the agents was separated from the group shortly after takeoff but managed to join the group again later. The separation was caused by a strong need to avoid obstacle collision. System reliability within leaving and joining the group during the flight was demonstrated during this experiment (Figure 6.9).

Figure 6.9 consists of three graphs. The first one shows the number of located neighbors of the UAV2 at any time. The second figure shows the number of located neighbors of the UAV7 which is the agent that has separated from the group. The last figure presents the distance of the UAV7 to the closest neighbor according to the ICP (blue) and in comparison with the ground truth (green). The same time period is highlighted by the orange vertical lines in all figures. The first orange line marks the time when the separated agent

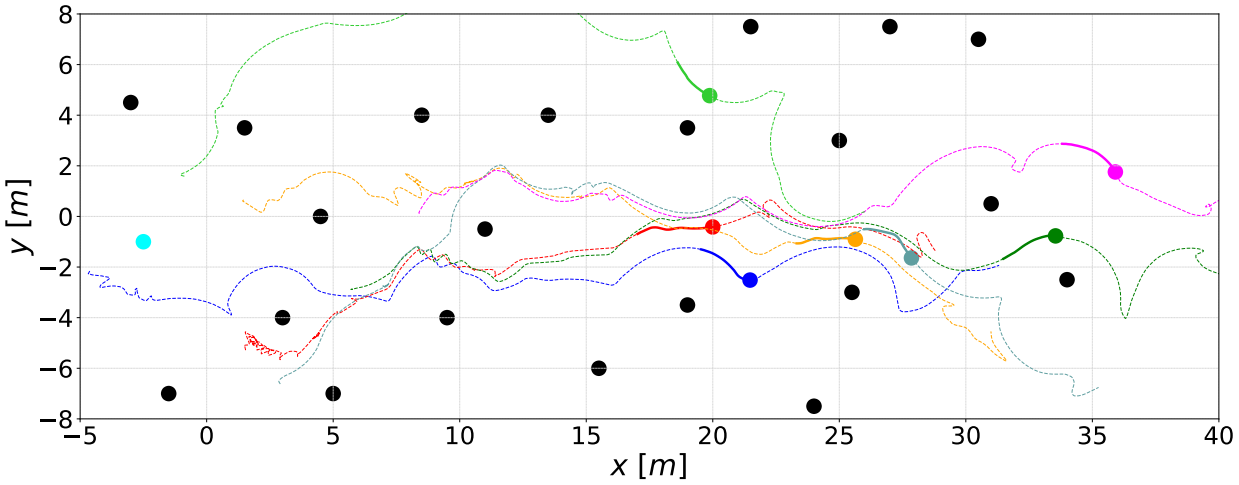


Figure 6.8: The swarm group of 7 UAVs and its path through the forest.

joins the group again and locates its first neighbor after being out of range. Since this moment the UAV7 tracks the position of the UAV2 with good precision according to the data in the third figure. The second orange line marks the time when the UAV2 localizes the UAV7. Red horizontal lines show the distance beyond which contact with the last neighbor before separation was lost (upper line) and when it was after separation regained (lower line). The contact was lost at around 6.7 m and regained at around 6.3 m. During the separation period, the number of neighbors of the UAV7 was a stable 0 which shows that all laser scans were correctly identified as out of range by the localization algorithm.

Paths of the individual agents as well as their locations at the time marked by the second orange vertical line in Figure 6.9 are shown in Figure 6.8. The cyan dot on the left represents the static UAV. The separated UAV (green) makes the first contact with UAV2 (red) as explained in Figure 6.9.

In simulations with a larger group of UAVs (e.g. 8 agents) an issue was identified. Each agent must consider all laser scans that are received. That means it takes more time to compute relative positions of neighbors for larger groups. Approximately every 5 seconds there was a peak in computational time as can be seen in the first graph of Figure 6.10. This time period corresponds with time intervals after which lICP is used to update location (Figure 5.2) and also the time after which the suspended laser scans are evaluated again using iICP. Therefore, it has been concluded that these peaks are caused by a high concentration of time-demanding algorithms (lICP and iICP) at one moment. This is further supported by the second graph in Figure 6.10 where the line representing the total number of lICP and iICP algorithms has a much steeper slope in those peaks. This issue causes delays because the controller waits with the command until all ICP algorithms are finished. The following approach was designed to solve the problem described in Figure

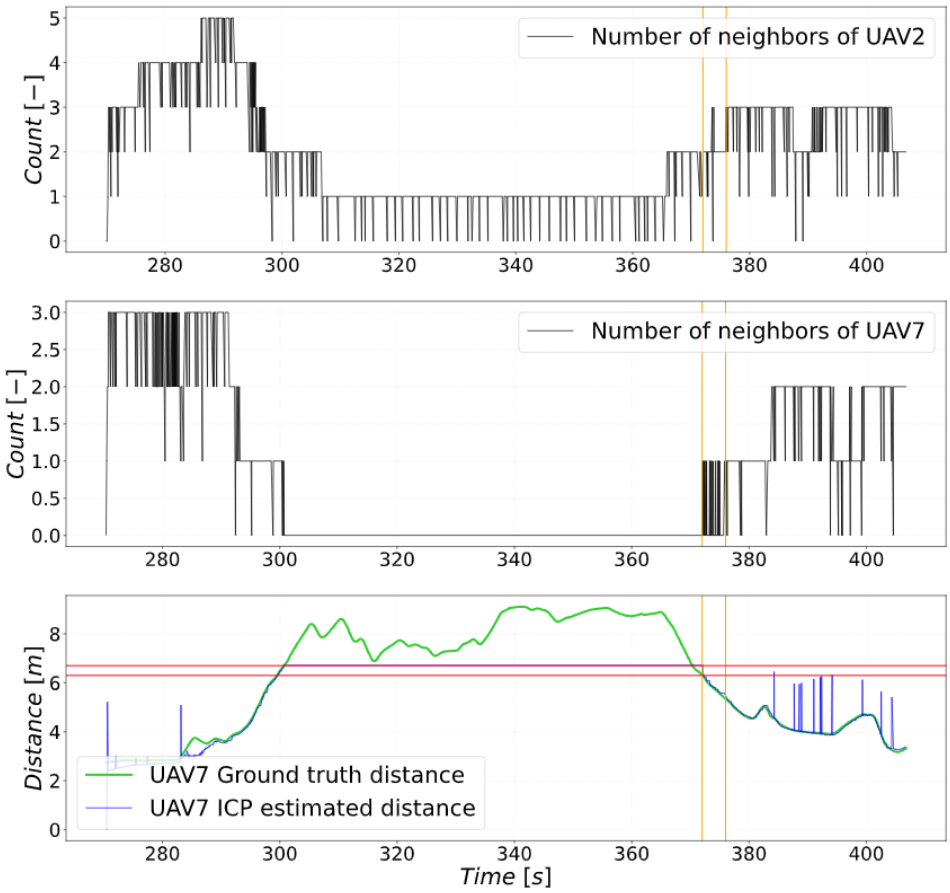


Figure 6.9: This figure describes the separation of UAV7 and its reunion with the UAV2 and the rest of the swarm.

6.10. The goal is to have concentrated computationally demanding algorithms spread out in time so that the periods with lower workload are utilized better. Only two lICP/iICP with the highest priority can be calculated in every step before the control command is published. The priority depends on time, when the location was lastly updated. lICP algorithms with lower priority are replaced by quicker sICP. Skipped lICPs are executed in one of the following computational steps, when they reach the top of the priority list. Also the maximum number of ICP iterations was reduced. Lastly, a failsafe was added that stops further computations if the step duration is already too long. However, in this case the failsafe has not even triggered. All of this was tested to confirm that it does not affect the precision of the localization system. The results are shown in Figure 6.11. The average mean value of step duration was 0.0362 s in Figure 6.10 and is 0.0317 s after the modifications. The computation time mean is similar but the variance is smaller. Therefore, the modifications are distributing the workload more uniformly. This is also supported by the second graph in Figure 6.11, where a similar number of lICP/iICP algorithms is computed in every step. The simulation introduced in this section was using these modifications.

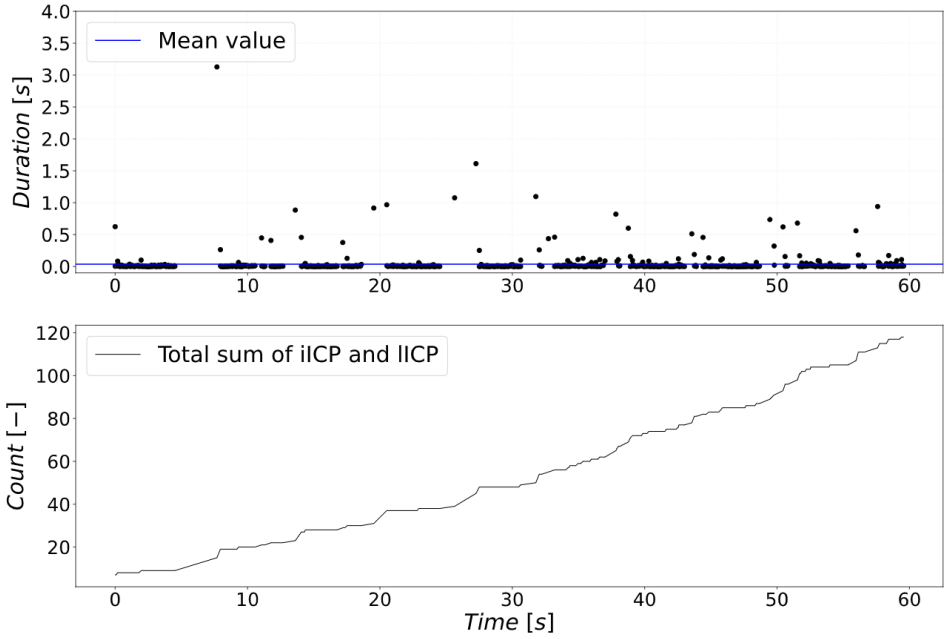


Figure 6.10: The upper graph shows sudden increases in the computational time needed to localize the neighbors. The lower graph shows that the higher demand for computational time is connected to the accumulation of processes.

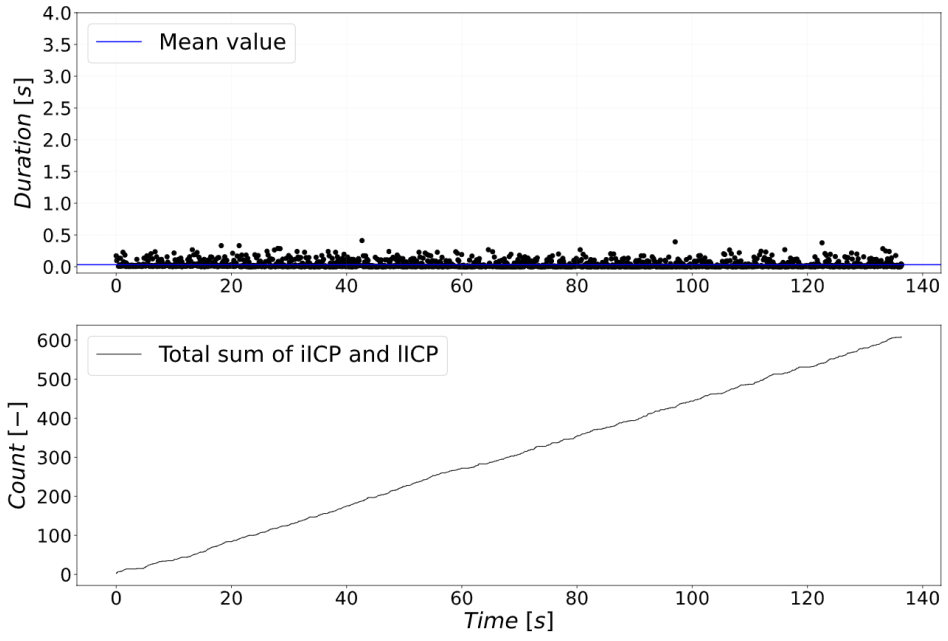


Figure 6.11: The situation after the issue from Figure 6.10 was resolved. The mean value of computational time did not change and the distribution is more uniform as the algorithms are being processed continuously.

6.3 5 UAV - Matlab

The transition of the proposed swarm system to the Matlab was verified by this experiment. It is supposed to prove that the Matlab simulator without UAV dynamics still reflects reality to a certain extend. The Matlab simulator will then be used for large-scale simulation.

Graph 6.14 shows the comparison between trajectories of swarm particles in Gazebo and Matlab within the same initial condition as in section 6.1. The time to goal was approximately 120s for Gazebo and 105s for Matlab. Means of agent to obstacle distances (Figure 6.12) were calculated using data from all 5 UAVs and they differ by approximately 0.3 m. Means of agent to agent distances (Figure 6.13) were calculated using data from all 5 UAVs and they differ by approximately 0.4m. The means from the Matlab simulation are higher in both cases.

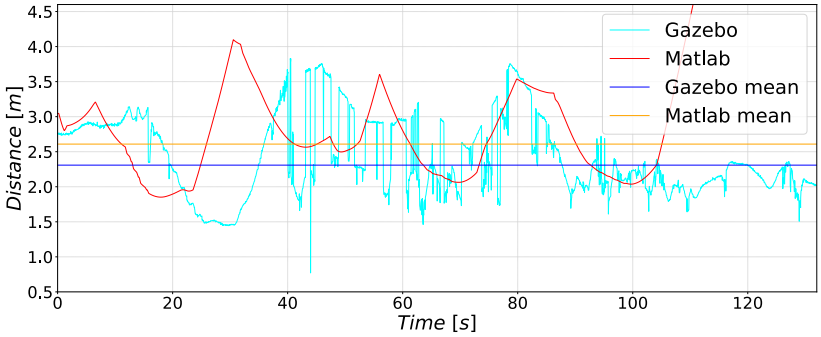


Figure 6.12: Agent to obstacle distances. Data are displayed only from one pair of UAVs.

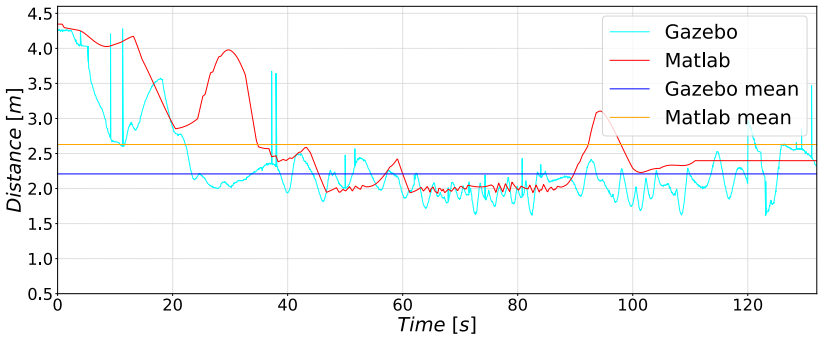


Figure 6.13: Agent to agent distances. Data are displayed only from one pair of UAVs.

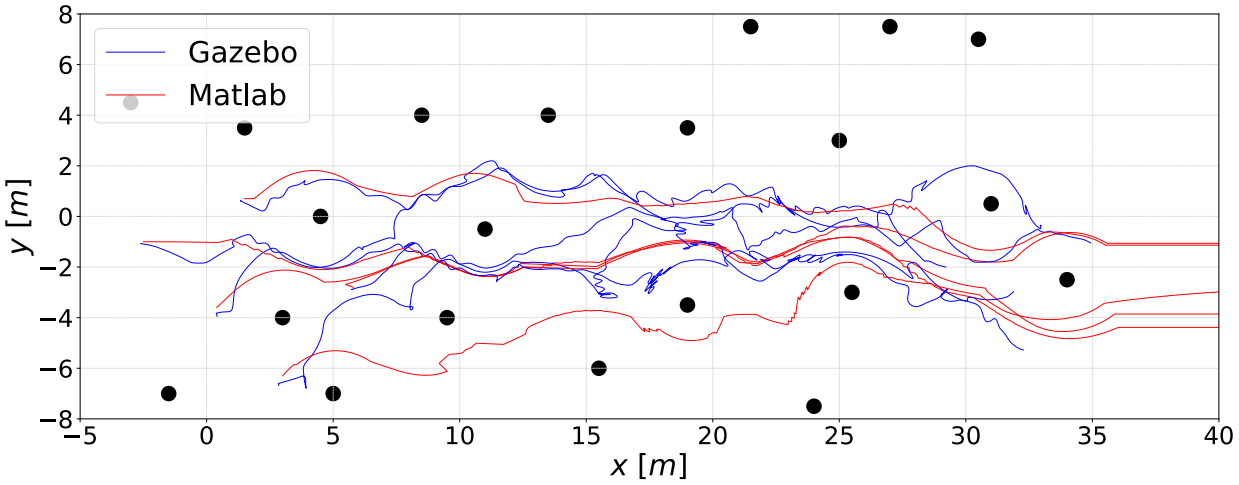


Figure 6.14: The comparison between trajectories from simulations in Gazebo and Matlab.

6.4 40 UAV - Matlab

An experiment with 40 UAVs in Matlab tests the scalability of the proposed system. Only the swarm controller (section 4) was tested in this simulation. The goal of the swarm was to navigate straight through a forest environment. The forest environment (Figure 6.15) was generated based on the tree density and distances between individual trees in the previous experiments to have comparable conditions.

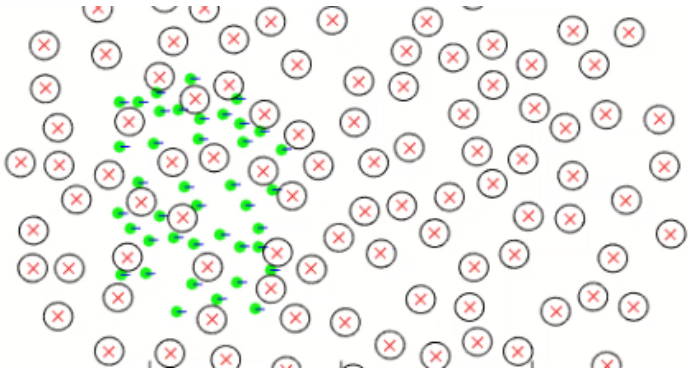


Figure 6.15: The swarm particles are green. Red crosses are centers of trees and black circles have a radius of 1.5 m. The direction of motion is from left to right.

Results are presented in Figure 6.16. One UAV was used to present the results of avoidance, navigation, separation, and cohesion vector. Distances to the closest obstacle and closest UAV are also presented on the same UAV. Moreover, distances are presented with their mean values calculated using the entire 40-UAV swarm. The minimal distance

to an obstacle was 1.06 m and the mean value is 2.71 m. The minimal distance to another agent was 1.31 m and the mean is 2.24 m.

The obstacle avoidance and the navigation vector are behaving similarly as in section 6.1. The UAV was positioned in the middle of the swarm for most of the time. Therefore, the separation vector is more frequent whereas the cohesion vector is inactive.

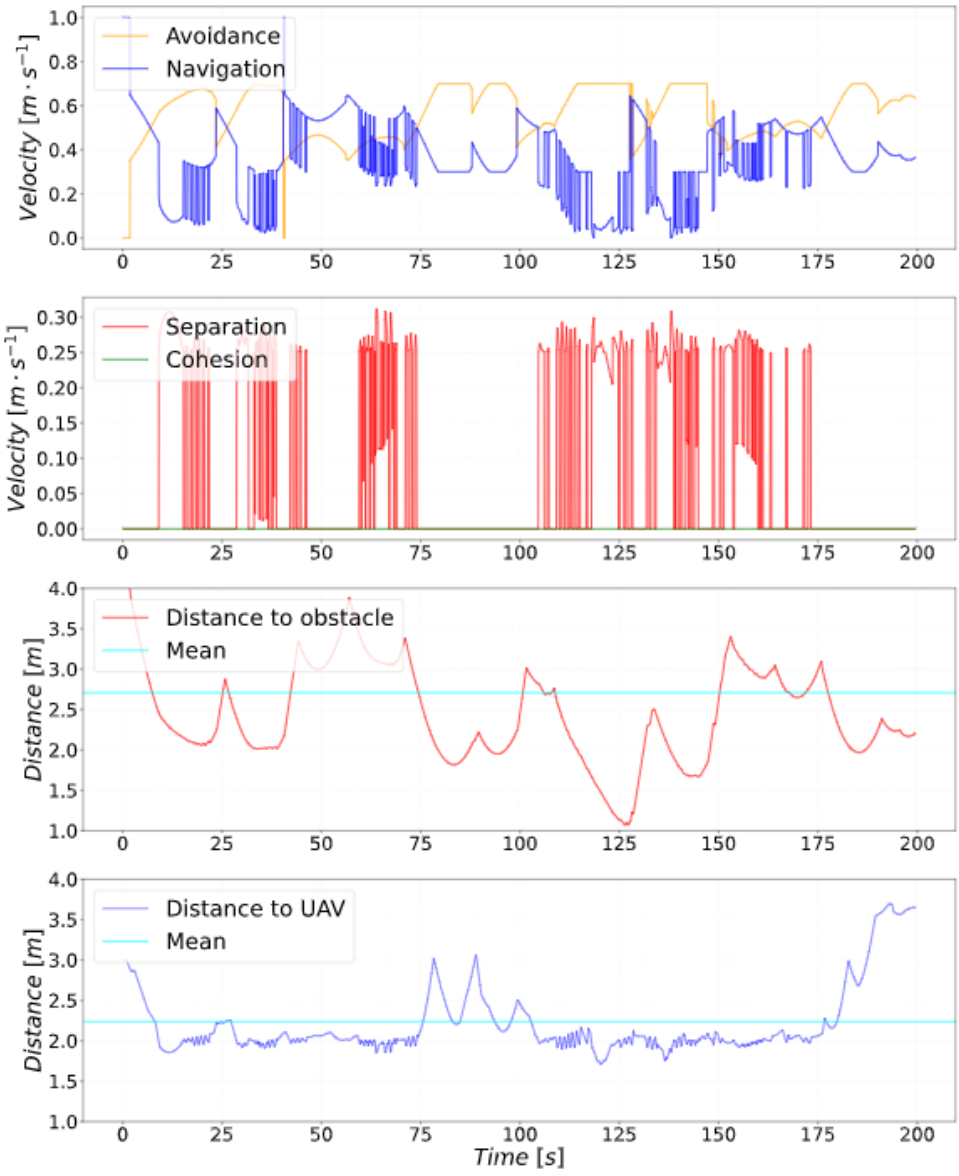


Figure 6.16: The first graph depicts magnitudes of obstacle avoidance and navigation vector. The second graph shows the magnitudes of the separation and cohesion vector. The last two graphs show minimal distance to obstacles and other UAVs respectively.

6.5 4 UAV - Gazebo comparison

The last experiment was performed to compare the system proposed by this thesis with the approach introduced in [1]. Results from [1] are shown in Figure 6.17. The experiment includes 4 UAVs. The UAV in the front is informed about the location of the goal. The same map and goal location was used with the system proposed in this thesis.

The UAVs stayed together as a group and successfully reached the goal location in both cases (Figure 6.17 and 6.18). The approach from [1] took 340 s to reach the goal. Swarm with the approach proposed in this thesis reached the goal in 147 s. In the end, the system proposed in this thesis was 2.3 times faster than the one introduced in [1].

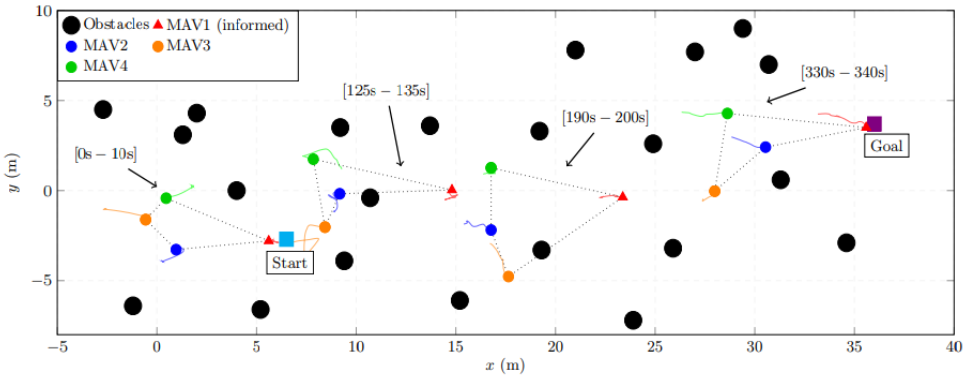


Figure 6.17: Flight of 4 UAVs using the system introduced in [1].

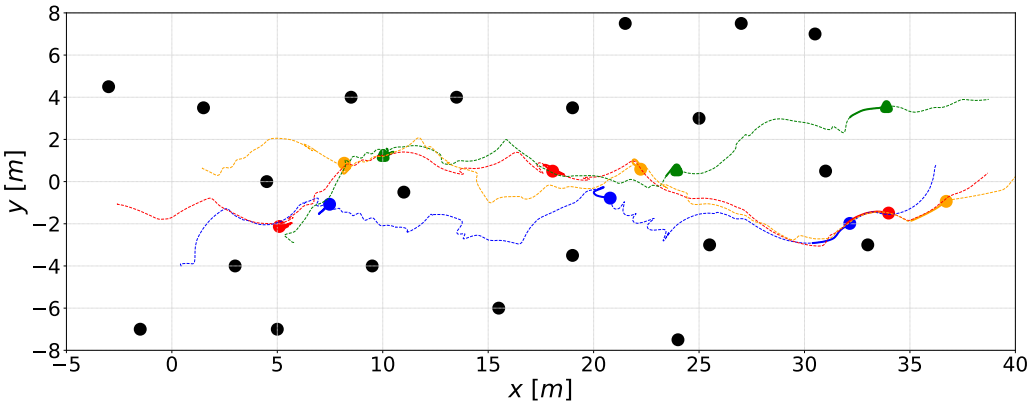


Figure 6.18: Flight of UAVs using the system proposed in this thesis. UAV marked by the triangle (green) is the informed UAV. Positions from the left are 35 s, 90 s, and 140 s since the start of the flight.

Chapter 7

Conclusion

This thesis introduced a system for swarming in a forest environment. A swarm controller and relative localization system were developed and tested. Simulations were carried out in the realistic Gazebo simulator and simulator implemented in Matlab to verify the functionality of the system. The swarm can safely and smoothly navigate between obstacles in a forest environment when using the approach proposed in this thesis. Following goals have been reached:

- System for relative localization using the ICP algorithm was introduced in Chapter 3.
- Swarm boids-like controller was developed in Chapter 4 allowing navigation of UAVs in a forest environment.
- Efficiency of the system for relative localization was further improved in Chapter 5.
- The complete system was implemented and integrated into Robot Operating System and the Multi-Robot Systems group system.
- Behavior of the system was successfully tested in Chapter 6 in realistic Gazebo simulator and Matlab.
- Developed approach was compared with the existing method for forest flying used in Multi-Robot Systems group.

The future work will be focused as follows. Prepare a HW outdoor experiment which we were unable to carry out because of the current situation. Further, merge the existing relative localization system used in the MRS group with the ICP variant as they can complement each other. The existing system would be used for larger distances and ICP for closer proximity.

Bibliography

[1] A. Ahmad, V. Walter, P. Petracek, M. Petrlik, T. Baca, D. Zaitlik, and M. Saska, "Autonomous aerial swarming in gnss-denied environments with high obstacle density," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*, 2021.

[2] G. Bevacqua, J. Cacace, A. Finzi, and V. Lippiello, "Mixed-Initiative Planning and Execution for Multiple Drones in Search and Rescue Missions," *Proceedings of the International Conference on Automated Planning and Scheduling*, vol. 25, no. 1, Apr. 2015. [Online]. Available: https://ojs.aaai.org/index.php/ICAPS/article/view/13700

[3] S. Mayer, L. Lischke, and P. W. Woźniak, "Drones for Search and Rescue," in *1st International Workshop on Human-Drone Interaction.* Glasgow, United Kingdom: Ecole Nationale de l'Aviation Civile [ENAC], May 2019. [Online]. Available: https://hal.archives-ouvertes.fr/hal-02128385

[4] J. Cacace, A. Finzi, and V. Lippiello, "Multimodal Interaction with Multiple Co-located Drones in Search and Rescue Missions," *arXiv:1605.07316 [cs]*, May 2016, arXiv: 1605.07316. [Online]. Available: http://arxiv.org/abs/1605.07316

[5] M. Saska, V. Vonásek, J. Chudoba, J. Thomas, G. Loianno, and V. Kumar, "Swarm Distribution and Deployment for Cooperative Surveillance by Micro-Aerial Vehicles," *Journal of Intelligent & Robotic Systems*, vol. 84, no. 1, pp. 469–492, Dec. 2016. [Online]. Available: https://doi.org/10.1007/s10846-016-0338-z

[6] F. Flammini, C. Pragliola, and G. Smarra, "Railway infrastructure monitoring by drones," in *2016 International Conference on Electrical Systems for Aircraft, Railway, Ship Propulsion and Road Vehicles International Transportation Electrification Conference (ESARS-ITEC)*, Nov. 2016, pp. 1–6.

[7] G. M. Bolla, M. Casagrande, A. Comazzetto, R. D. Moro, M. Destro, E. Fantin, G. Colombatti, A. Aboudan, and E. C. Lorenzini, "ARIA: Air Pollutants Monitoring Using UAVs," in *2018 5th IEEE International Workshop on Metrology for AeroSpace (MetroAeroSpace)*, Jun. 2018, pp. 225–229, iSSN: 2575-7490.

[8] R. Godall, E. O'tega, and O. Godswill, "Autonomous monitoring, analysis, and countering of air pollution using environmental drones," *Heliyon*, vol. 6, no. 1, p. e03252, Jan. 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2405844020300979

[9] M. Innocente and P. Grasso, "Self-organising swarms of firefighting drones: Harnessing the power of collective intelligence in decentralised multi-robot systems," *Journal of Computational Science*, vol. 34, pp. 80–101, May 2019. [Online]. Available: https://www.sciencedirect.com/science/article/abs/pii/S1877750318310238

[10] E. Ausonio, P. Bagnerini, and M. Ghio, "Drone Swarms in Fire Suppression Activities: A Conceptual Framework," *Drones*, vol. 5, no. 1, p. 17, Mar. 2021. [Online]. Available: https://www.mdpi.com/2504-446X/5/1/17

[11] B. Aydin, E. Selvi, J. Tao, and M. J. Starek, "Use of Fire-Extinguishing Balls for a Conceptual System of Drone-Assisted Wildfire Fighting," *Drones*, vol. 3, no. 1, p. 17, Mar. 2019. [Online]. Available: https://www.mdpi.com/2504-446X/3/1/17

[12] K. Spanaki, E. Karafili, U. Sivarajah, S. Despoudi, and Z. Irani, "Artificial intelligence and food security: swarm intelligence of AgriTech drones for smart AgriFood operations," *Production Planning & Control*, vol. 0, no. 0, pp. 1–19, Feb. 2021. [Online]. Available: https://doi.org/10.1080/09537287.2021.1882688

[13] P. Tripicchio, M. Satler, G. Dabisias, E. Ruffaldi, and C. A. Avizzano, "Towards Smart Farming and Sustainable Agriculture with Drones," in *2015 International Conference on Intelligent Environments*, Jul. 2015, pp. 140–143.

[14] V. Puri, A. Nayyar, and L. Raja, "Agriculture drones: A modern breakthrough in precision agriculture," *Journal of Statistics and Management Systems*, vol. 20, no. 4, pp. 507–518, Jul. 2017. [Online]. Available: https://doi.org/10.1080/09720510.2017.1395171

[15] C. W. Reynolds, "Flocks, herds and schools: A distributed behavioral model," in *Proceedings of the 14th annual conference on Computer graphics and interactive techniques*, ser. SIGGRAPH '87. New York, NY, USA: Association for Computing Machinery, Aug. 1987, pp. 25–34. [Online]. Available: https://doi.org/10.1145/37401.37406

[16] J. M. Phillips, R. Liu, and C. Tomasi, "Outlier Robust ICP for Minimizing Fractional RMSD," in *Sixth International Conference on 3-D Digital Imaging and Modeling (3DIM 2007)*, Aug. 2007, pp. 427–434, iSSN: 1550-6185.

[17] V. Walter, N. Staub, A. Franchi, and M. Saska, "UVDAR System for Visual Relative Localization With Application to Leader–Follower Formations of Multirotor UAVs," *IEEE Robotics and Automation Letters*, vol. 4, no. 3, pp. 2637–2644, Jul. 2019.

[18] P. Petráček, V. Walter, T. Báča, and M. Saska, "Bio-inspired compact swarms of unmanned aerial vehicles without communication and external localization," *Bioinspiration & Biomimetics*, vol. 16, no. 2, p. 026009, dec 2020. [Online]. Available: https://doi.org/10.1088/1748-3190/abc6b3

[19] T. Schmickl and H. Hamann, "Beeclust: A swarm algorithm derived from honeybees," *Bio-inspired Computing and Communication Networks. CRC Press (March 2011)*, 2011.

[20] G. Vásárhelyi, C. Virágh, G. Somorjai, T. Nepusz, A. E. Eiben, and T. Vicsek, "Optimized flocking of autonomous drones in confined environments," *Science Robotics*, vol. 3, no. 20, Jul. 2018. [Online]. Available: https://robotics.sciencemag.org/content/3/20/eaat3536

[21] V. Walter, M. Saska, and A. Franchi, "Fast Mutual Relative Localization of UAVs using Ultraviolet LED Markers," in *2018 International Conference on Unmanned Aircraft Systems (ICUAS)*, Jun. 2018, pp. 1217–1226, iSSN: 2575-7296.

[22] K. Guo, Z. Qiu, W. Meng, L. Xie, and R. Teo, "Ultra-wideband based cooperative relative localization algorithm and experiments for multiple unmanned aerial vehicles in GPS denied environments," *International Journal of Micro Air Vehicles*, vol. 9, no. 3, pp. 169–186, Sep. 2017. [Online]. Available: https://doi.org/10.1177/1756829317695564

[23] V. Walter, M. Vrba, and M. Saska, "On training datasets for machine learning-based visual relative localization of micro-scale UAVs," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*, May 2020, pp. 10 674–10 680, iSSN: 2577-087X.

[24] M. Saska, J. Vakula, and L. Přeućil, "Swarms of micro aerial vehicles stabilized under a visual relative localization," in *2014 IEEE International Conference on Robotics and Automation (ICRA)*, May 2014, pp. 3570–3575, iSSN: 1050-4729.

[25] T. Baca, M. Petrlik, M. Vrba, V. Spurny, R. Penicka, D. Hert, and M. Saska, "The MRS UAV System: Pushing the Frontiers of Reproducible Research, Real-world Deployment, and Education with Autonomous Unmanned Aerial Vehicles," *arXiv:2008.08050 [cs, eess]*, Jan. 2021, arXiv: 2008.08050. [Online]. Available: http://arxiv.org/abs/2008.08050

Appendices

List of abbreviations

In Table 1 are listed abbreviations used in this thesis.

Abbreviation	Meaning
UAV	Unmanned Aerial Vehicle
GPS	Global Positioning System
UVDAR	UltraViolet Direction and Ranging
UV	Ultraviolet
ROS	Robot Operating System
ICP	Iterative Closest Point
FRMSD	Fractional Root Mean Square Distance
UWB	Ultra-Wideband
SVD	Singular Value Decomposition
MRS	Multi-robot systems

Table 1: Lists of abbreviations