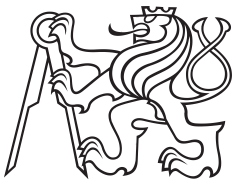
Bachelor Project



Czech Technical University in Prague

F3

Faculty of Electrical Engineering
Department of Cybernetics

Map Merging for UAV Swarms

Jan Maděra

Supervisor: Ing. Matěj Petrlík
Field of study: Open Informatics
Subfield: Computer and Information Science
August 2020

BACHELOR'S THESIS ASSIGNMENT

I. Personal and study details

Student's name: **Maděra Jan** Personal ID number: **469836**

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

Department / Institute: **Department of Cybernetics**

Study program: **Open Informatics**

Branch of study: **Computer and Information Science**

II. Bachelor's thesis details

Bachelor's thesis title in English:

Map Merging for UAV Swarms

Bachelor's thesis title in Czech:

Slučování map pro roje dronů

Guidelines:

The focus of this thesis is to implement an algorithm for merging local occupancy grid maps [1] of UAV agents. The input of the implemented technique will be the map built by the Hector SLAM [2] algorithm. The developed system should be applicable in real-world UAV teams. The following tasks will be solved:
• Become familiar with the Hector SLAM [2] algorithm and its occupancy grid map [1] representation.
• Perform a survey of available map merging techniques suitable for building a global map shared by a team of UAVs. Recommended starting point are publications [3, 4, 5].
• Implement an occupancy grid map merging method of choice.
• Integrate the implemented method into the MRS group software architecture based on ROS middleware.
• Provide an on-demand merged map output with a 2D pose estimate of each agent.
• Verify the functionality in the Gazebo simulator, assess the capabilities and limitations. Discuss the scalability with respect to the number of agents.
• Prepare an experiment with a group of UAVs that shows the applicability in the real-world environment of choice.

Bibliography / sources:

[1] Thrun, Sebastian, and Arno Bücken. "Integrating grid-based and topological maps for mobile robot navigation." Proceedings of the National Conference on Artificial Intelligence. 1996.
[2] S. Kohlbrecher, O. von Stryk, J. Meyer and U. Klingauf, "A flexible and scalable SLAM system with full 3D motion estimation," 2011 IEEE International Symposium on Safety, Security, and Rescue Robotics, Kyoto, 2011.
[3] A. Birk and S. Carpin, "Merging Occupancy Grid Maps From Multiple Robots," in Proceedings of the IEEE, July 2006.
[4] H. Li, M. Tsukada, F. Nashashibi and M. Parent, "Multivehicle Cooperative Local Mapping: A Methodology Based on Occupancy Grid Map Merging," in IEEE Transactions on Intelligent Transportation Systems, Oct. 2014.
[5] Carpin, S. Fast and accurate map merging for multi-robot systems. Auton Robot 25, 305–316 (2008).

Name and workplace of bachelor's thesis supervisor:

Ing. Matěj Petrlík, Multi-robot Systems, FEE

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

Date of bachelor's thesis assignment: **08.05.2020** Deadline for bachelor thesis submission: **14.08.2020**

Assignment valid until: **30.09.2020**

Ing. Matěj Petrlík	doc. 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

Acknowledgements

I would like to thank my supervisor Ing. Matěj Petrlík for his patience, guidance and valuable insights. I thank my family for their encouraging support throughout my studies. I am also grateful to all people from the MRS group who helped me to understand the workflow with the MRS group software architecture. Last but not least, I thank my girlfriend for her endless support.

Declaration

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, 14. August 2020

Abstract

In this thesis, we present an implementation of a genetic-algorithm-based method for merging occupancy grid maps. This method finds the relative pose of agents by iteratively evolving a population of relative poses using a fitness function to evaluate each individual pose. We apply the method on two datasets, which shows that the approach can merge maps with relatively high noise and initial poses of agents being up to 20 meters apart.

With known local poses of agents in their respective maps and with the common merged map, we can approximate the pose of agents relative to each other. Another benefit of the map merging is that we can significantly reduce the time needed for exploration tasks by using multiple agents with a shared map at once.

Keywords: genetic algorithm, LIDAR, map merging, occupancy grid map, relative pose estimation, ROS, simultaneous localization and mapping, unmanned aerial vehicle

Supervisor: Ing. Matěj Petrlík

Abstrakt

V této práci představujeme implementaci slučování mřížkových map obsazenosti za použití metody založené na genetickém algoritmu. Tato metoda je schopna najít relativní pózy agentů iterativním vývojem populace relativních póz za použití fitness funkce vyhodnocující pózu každého jednotlivce. Metoda je aplikována na dvě datové sady. Aplikace metody ukazuje, že tento postup dokáže sloučit mapy s relativně velkým zašuměním a s počátečními pózami agentů vzdálených až 20 metrů.

Se známými lokálními pózami agentů v jejich mapách a se společnou sloučenou mapou můžeme odhadnout relativní vzájemné pózy všech ostatních agentů. Dalším přínosem slučování map je možnost významně zredukovat čas potřebný k prozkoumání okolí použitím více agentů se sdílenou mapou najednou.

Klíčová slova: bezpilotní letoun, genetický algoritmus, LIDAR, mřížková mapa obsazenosti, odhad relativní pózy, ROS, simultánní lokalizace a mapování, slučování map

Překlad názvu: Slučování map pro roje dronů

Contents

Figures

Tables

Abbreviations and Notation

List of notation

In Table 1 is listed notation used in this thesis. Note that we denote vectors with bold lowercase letters (e.g. $\mathbf{p}$), matrices (and map vectors) with capital letters (e.g. M_i).

Symbol	Meaning
plain lowercase letter, e.g. c	scalar
bold lowercase letter, e.g. $\mathbf{p}$	column vector
uppercase letter, e.g. M	matrix or map vector
$\mathbf{p}^T, M^T$	vector and matrix transpose
$\oplus$	Compounding operator
$\mathbf{\Delta}, F$	Objective function (heuristic function, fitness function)
ai	acceptance index
M_i	Generic occupancy grid map with index i
md	Manhattan distance function
Occ	Occupancy threshold function
$\mathbf{p}$	Relative pose
$\hat{\mathbf{p}}$	Optimal relative pose

Table 1: Lists of mathematical notation

List of abbreviations

In Table 2 are listed abbreviations used in this thesis.

Abbreviation	Meaning
2D	Two-dimensional
3D	Three-dimensional
6DOF	Six degrees of freedom
API	Application programming interface
DHT	Discretized Hough transform
GNSS	Global navigation satellite system
GVD	Generalized voronoi diagram
IMU	Inertial measurement unit
ICP	Iterative closest point
LIDAR	Light detection and ranging
MAV	Micro aerial vehicles
MRS	Multi-robot systems
MRSLAM	Multi-robot simultaneous localization and mapping
OGM	Occupancy grid map
OV	Occupancy value
PC	Personal computer
ROS	Robot operating system
SLAM	Simultaneous localization and mapping
SOM	Self-organizing map
UAV	Unmanned aerial vehicle
UGV	Unmanned ground vehicle
V2V RP	Vehicle to vehicle relative pose

Table 2: Lists of abbreviations

Chapter 1

Introduction

Autonomous unmanned aerial vehicles (**UAVs**) have become a popular part of multi-robot systems (**MRS**). Although an autonomous UAV can have many shapes (airplane, helicopter) and features, when we refer to UAV in this thesis, we usually mean quad-copter similar to the one presented in figure 1.1. There are many valid reasons for their increased frequency of usage in a lot of real-world scenarios. The first reason can be their high flexibility. UAVs can be used in many unique scenarios. It is relatively affordable to have a small drone or a swarm of micro aerial vehicles (**MAVs**) for reconnaissance and flying in tight spaces. On the other hand, some UAVs can be as large as a couple of meters. Another reason can be better precision of available sensors and rising onboard computational power, which allows the use of more computationally intensive algorithms.

Groups of UAVs are being used in many challenging scenarios. Examples of possible deployment are search and rescue operations where UAVs aid human workers in mapping hazardous environments and gathering information about the position of essential subjects. In letter [25], only single UAV was used, but the simultaneous use of multiple cooperating UAVs could speed up the exploration. Many technological challenges and scenarios where MRS could be used are outlined in a visionary project [29]. The last example is the documentation of dark and human-restricted areas of historical monuments [26, 18]. In both cited letters, all MAVs rely on ground-based devices that provide a common reference frame, precise localization, and mapping of their surroundings to ensure their safety. Deployment of ground-based localization devices can be time-consuming and expensive in large scenarios, or even impossible in some hard to access places like caves and mines.

The localization precision requirements of some missions can be satisfied only by relying on the additional localization devices. In some other missions like search and rescue, the fast deployment is much more significant than extreme precision. Without ground-based stations measuring the UAV's position, UAVs' swarm has to rely on their sensors to determine the relative position of each other and map their surroundings. Furthermore, to utilize a group of UAVs for faster completion of missions, each UAV provides its

Figure 1.1: F450 quad-copter equipped with 2D Slamtec RPLIDAR A3 360° which was used in this thesis real-world experiments.

map, also called a local map, to other swarm members. These available local maps can then be merged to produce one common global map. The additional information about the environment from other members of the swarm contained in a global map can then be used for better path planning and optimal exploration of unexplored parts of the environment. This thesis addresses problem of map merging for UAV swarms with an estimation of UAV position relative to each other.

1.1 Simultaneous localization and mapping

1.1.1 UAV SLAM

To autonomously complete a UAV mission and simultaneously ensure its safety and especially the safety of surrounding subjects which can be in many cases of a (historical) value many times larger than the value of a UAV (as in [26]), the drone has to have precise sensors which sense any obstacle that could disturb an execution of the UAV mission and sensors which determine its position. However, that is not enough. Among other things, these sensed data from multiple sensors have to be fused into meaningful information about UAV state, pose, and surrounding obstacles based on which the UAV can choose the best solution of a given task while minimizing risks. The process of incrementally constructing a consistent map given the data from sensors while at the same time, localizing agents in the environment is called simultaneous localization and mapping (**SLAM**). Accurately solving SLAM

for a robot is one of the most challenging problems in field robotics by itself, as is roughly stated in [34].

1.1.2 UAV swarm SLAM

SLAM is a complex problem for a single UAV. Trying to localize swarm of UAV relative to each other and simultaneously merging maps of individual drones adds another layer of complexity.

The problem of merging maps can be described in the sense of frames of reference. If we provide a common frame to all UAVs, then merging maps is trivial. In this situation, we know the accurate relative pose of each UAV. We just need to transform the map of each UAV by its relative pose with respect to the UAV that is currently merging maps. This is needed in order to align the local maps of UAVs correctly. If there is no external device providing a common reference frame, or UAVs do not provide it themselves, each individually localized UAV has different non-coincident reference frame. Without a common frame, UAVs cannot cooperate, and while deployed to the same workspace, they are risking collision unless they are localized relative to each other. Localization can be provided using only sensors and hardware right on the UAVs like markers [23], blinking UV LEDs [37] or computer vision [36]. Another possible way to provide an approximate common frame of reference is to use some type of global navigation satellite system (**GNSS**) like GPS or GLONASS. Nevertheless, GNSS's main disadvantage is that it can be used only in areas with a mostly unobstructed view of satellites. Consequently, to determine UAV position in indoor missions or even operations in dense forest or dense urban areas with high buildings where the GNSS signal is insufficient, a different type of localization like the ones using markers, UV LEDs, or computer vision has to be used.

However, what if any of the mentioned methods for relative localization is not available? If direct localization of other UAVs is impossible, the transformation (relative pose) between individual reference frames has to be estimated indirectly. One way to estimate the frame is by building a single global map from local maps of UAVs. That way, if we have prior knowledge of UAVs location in local maps and merge the local maps, we can also extract the UAV global position from the created global map. This thesis assumes **LIDAR** (Light Detection and Ranging) as the primary sensor for collecting data of single UAV, which are then processed in SLAM problem-solving algorithm called HectorSLAM [17] to build a local map. There are some possible solutions to the problem of merging maps described in chapter 2. We selected one of the solutions and implemented it into Multi-robot Systems Group software architecture [1].

[1] Documentation of the MRS Group software architecture with links to open source code on Github and MRS Group's main website can be found on `https://ctu-mrs.github.io/`

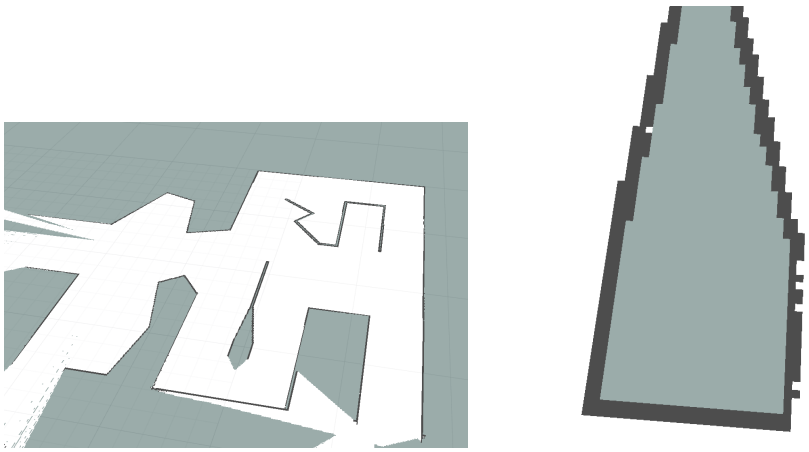


Figure 1.2: (**left**) A visualization of a occupancy grid map generated by RVIZ, the 3D visualization tool for Robot Operating System (ROS). Black cells have occupancy value 100, white are unoccupied cells with occupancy value 0, and the rest of cells have unknown occupancy state with value -1. (**right**) Detail of the occupancy grid map with visible boundaries of cells.

1.1.3 Types of map representation

Until now, we used maps as a general concept of understanding the environment. However, mobile robotics use many different representations of maps for different applications. Let us introduce at least three commonly used map representations.

One of the most frequently used map representations is ***occupancy grid map*** first proposed as an occupancy map in [8]. This representation of the map will be described in more depth because we use it to implement map merging. It is a metric representation in which each object or its part has assigned coordinates in the map space. It represents a map in most cases as a two-dimensional (2D), a discrete evenly-spaced grid of cells. Each cell has an occupancy value indicating a belief that an obstacle is located at the place of the cell in the real environment. For example, in Robot Operating System [27], the cell inside the occupancy grid map representation can acquire value -1 if its state is unknown and value in the interval $[0, 100]$. The value is 0 if the cell is believed to be unoccupied. The higher the value is, the firmer the belief that the cell is occupied. An example of an occupancy grid map is shown in figure 1.2. Sensors like LIDAR, SONAR, or stereovision can be used to generate the occupancy grid map.

Topological map is another type of map representation. It usually divides the map into regions that represent parts of the environment. These regions can be represented as nodes in a graph with edges representing paths between the regions. This representation needs significantly less memory and computational power than the occupancy grid representation of the same environment. Nevertheless, this map representation has many drawbacks. The simplification of the environment and the loss of important information

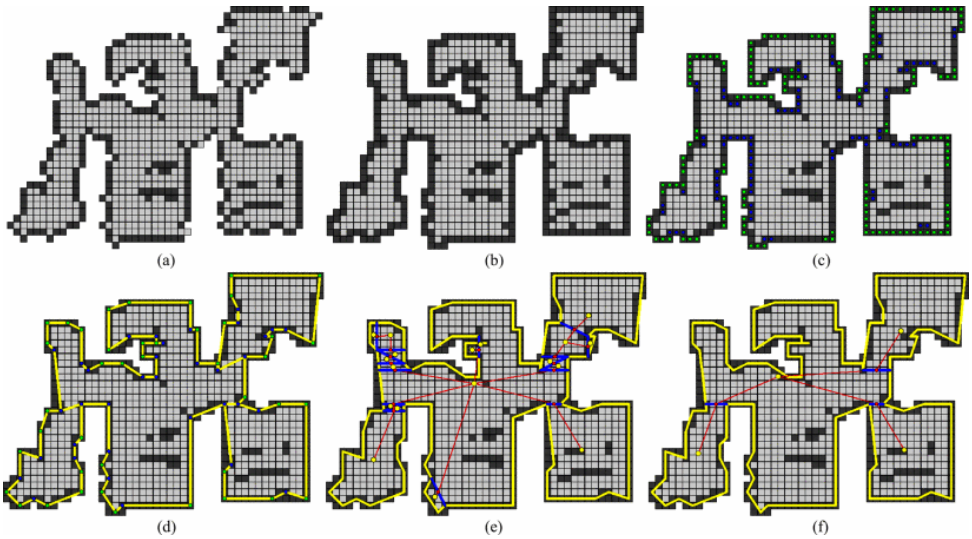


Figure 1.3: Example of topological map representation. Topological map is in this case generated from occupancy grid-map. In (e) the yellow circles are nodes in centers of rooms and red lines are edges of the topological map connecting neighboring nodes. Reprinted from [13, Figure 5]

about the environment are the main ones. The difference between the occupancy grid map and topological map can be seen in [35]. An example of a topological map generated on top of the occupancy grid map can be seen in figure 1.3.

3D Point cloud is a three-dimensional (3D) representation of an environment. It uses a set of points to represent the boundary of a 3D object. As in the case of the occupancy grid, it can be generated by the LIDAR scanner. Example of 3-D point cloud map merging can be seen in figure 1.4 or in the article [3].

1.2 Thesis structure

In this thesis, we present related work to the topic of map merging for UAV swarms in section 1.3, then a description of the used solver of SLAM problem called HectorSLAM in section 1.4. Chapter 2 presents survey of available map merging techniques suitable for building a global map shared by a team of UAVs. In chapter 3, we introduce the implementation of the method for merging local occupancy grid maps based on genetic algorithms. The experiments performed on the implementation and analysis of its performance are detailed in chapter 4. Possible future work is discussed in chapter 5.1. Finally, the thesis is concluded in chapter 5.

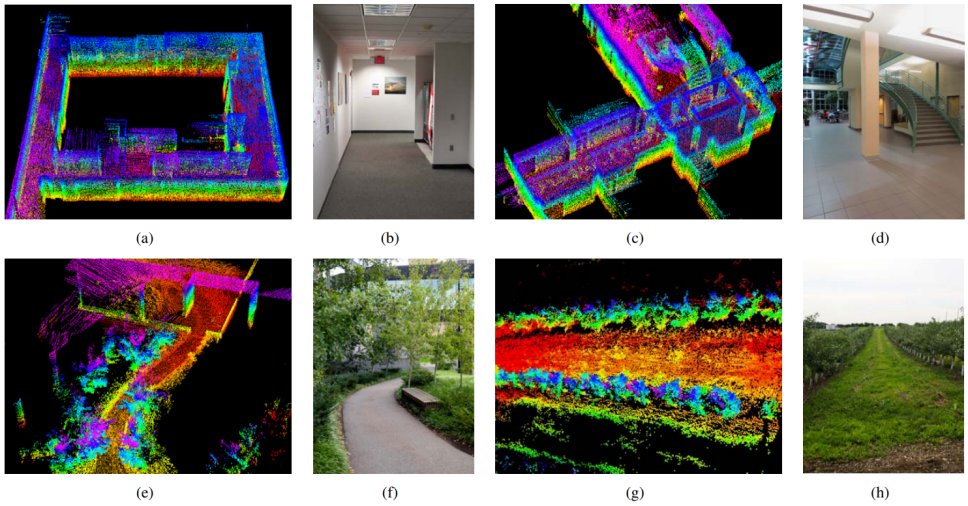


Figure 1.4: Examples of point cloud map representations with photos of relevant real world environments to the left of each point cloud. Reprinted from [39, Figure 10]

1.3 Related work

1.3.1 SLAM

SLAM is still an open problem in mobile robotics with significant interest among the research community. Basics of construction of occupancy grid map representation as well as an interesting use of grid-based map combined with not so precise but more memory efficient topological map is detailed in [35].

A survey of SLAM [12] describes many SLAM algorithms that could be possibly used. According to the survey, the SLAM solvers can be divided into two main subgroups. The first group uses camera sensors as the primary source of the necessary information about the surrounding environment used for localization and mapping. These approaches are labeled as **visual SLAM solvers**. The use of cameras and inertial measurement units (**IMUs**) brings increased robustness of the algorithms, mainly during an aggressive motion of the camera. The most of SLAM problem solvers can be categorized into two main approaches. These are *filtering-based approaches* and *optimization-based approaches*. The paper [22] is an example of a filtering-based approach that uses an extended Kalman filter-based algorithm called A Multi-State Constraint Kalman Filter for vision-aided inertial navigation. As an example of an optimization-based SLAM can serve an article [10], which uses a semi-direct visual odometry approach to create a map of the environment and estimate the motion of the vehicle and thus localize itself in its surrounding. Among the other worth mentioning visual SLAM approaches are the ones using deep learning, which can perceive not only semantic entities but also

geometric features and odometry at once, as is shown in [14]. As authors of the survey state, the visual SLAM approaches generally face problems like poor adaptability to changing illumination conditions, high dynamic environment, and low texture environment. They also have problems with fast motion, vigorous rotation, and image deformation while using a rolling shutter instead of a global shutter.

The second group labeled as **LIDAR SLAM solvers** can be further divided into subgroups of *2D SLAM* using, for example, the occupancy grid map described in section 1.1.3 as an output map representation, and *3D SLAM* using, for example, 3D point cloud representation of the environment. The survey [12] provides examples of 2D SLAM systems. One of them is Cartographer [11], which is SLAM system developed by Google, and the HectorSLAM [17] which is described in the section 1.4. Example of 3D SLAM solver can be IMLS-SLAM [5] using only point cloud generated by 3D LIDAR without IMU, Lidar Odometry and Mapping in Real-time (LOAM) [39] which uses point cloud from 3D LIDAR to separately compute odometry and generate the map. Compared to visual SLAM solvers using cameras, the LIDAR SLAM solvers are affected much less by illumination condition than visual solvers. For example, in our experiment with LIDAR in a dark forest with dense tree crowns, we had only small inaccuracies in detecting spots of surfaces directly illuminated by the Sun. On the other hand, LIDAR can be much more expensive than a camera due to its high-end technology and the need for expensive sensors to precisely measure the distance to objects. The survey further states that many researchers focus on the fusion of LIDAR and vision, balancing LIDAR reliability and versatility of vision.

Other types of LIDAR SLAM solvers were not exactly mentioned in the survey. One of these approaches uses the normal distribution transform [1] to solve SLAM with data from LIDAR. These methods divide the 2D plane into cells similar to an occupancy grid. To each cell, they then assign a normal distribution, and as a result, they get continuously differentiable probability density, which they can match with another scan. The SLAM technique described in [24] is using correlation for scan matching. This method does not use heuristics but instead try to implement probabilistically-motivated scan-matching.

1.3.2 Map merging

The map merging methods can be divided into many categories by different criteria. According to the article [20], there are two basic categories of map merging methods. The first category estimates the relative pose of two vehicles directly based on localization sensors, such as GPS, and then merges the maps according to this relative pose. These methods are called **Vehicle-Pose-Estimation-Based Methods**. The second category does not rely on accurate estimation of vehicle relative pose, but instead, these methods try to merge maps directly based on the maps' consistency measurements. These are

Map-Consistency-Based Methods. We deal only with the latter category of methods in this thesis as we do not use any localization sensors that could directly and precisely localize the two vehicles. The Map-Consistency-Based merging methods can be further divided by the type of map representation used, as shown in section 1.1.3. We use specifically the occupancy grid map representation in map merging, so this related work will cover methods using the same representation as we do. We found many interesting ideas on merging multiple occupancy grid maps.

The article [2] uses a random walk as a search method for the optimal relative pose of maps. Next in [20] H. Li et al. present a general framework for map merging based on genetic algorithm and uses objective function based on occupancy likelihood, which has many benefits oppose to [2]. The described method in [20] is the one we chose to implement. [30] is an exciting review of the state of the art of mobile MRS research, including easy to understand description of main problems with localization of agents relative positions and other mapping-specific issues.

Other interesting articles are describing merging methods or possible improvements to merging that we will also further explain in chapter 2. The map merging approach [4] uses Hough transform to find straight line-like features in both occupancy grid maps, which are then used to align maps during the merging process. The method [32] applies neural networks to the map merging problem. In the conclusion of the article, authors stated: "This is the first known application of neural network theory to solve the multiple robot SLAM." The paper [31] further improve the neural network method in [32]. It uses a similar method to find the relative rotation of two maps, but it approaches the problem of searching for translation differently. The authors used a probabilistic generalized Voronoi diagram to find the relative translation. This approach also accounts for uncertainty in the relative map pose rotation. The article [6] describes how Reinforcement Learning can be used to decide when is the best time to merge maps of two agents upon their rendezvous instead of immediate merging. The merging time management upon rendezvous is not a part of this thesis and could be explored in future work.

1.4 HectorSLAM

In our implementation of occupancy grid map merging, we used an open-source SLAM solver called HectorSLAM [17], which is available as a ROS package [16]. Its functionality is further described in the article [17]. This SLAM solver is aiming for usability not only on unmanned ground vehicles (**UGV**) but also on UAVs. The authors achieved this by estimating motion in full six degrees of freedom (**6DOF**). As is shown in figure 1.5, HectorSLAM provides not only the occupancy grid map but also a 2D pose estimate to 3D navigation subsystem, where it aids IMU and other optional sensors

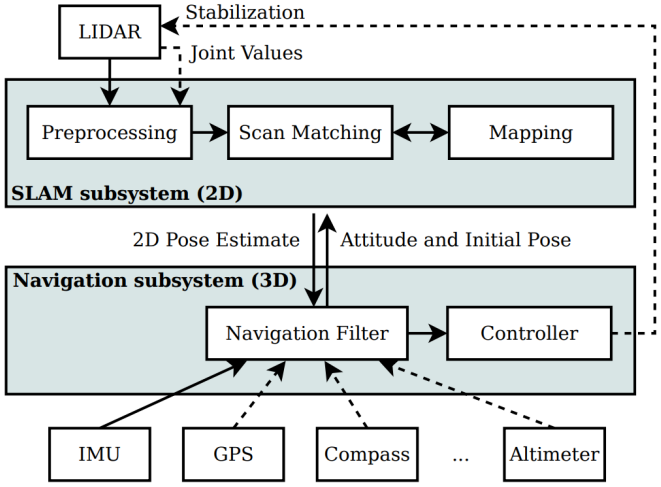


Figure 1.5: Overview of the mapping and navigation system (dashed lines depict optional information). Reprinted from [17, Figure 1]

information in 3D pose estimation. The navigation subsystem and SLAM subsystem are updated individually, so the 3D pose estimation is not slowed down by the SLAM subsystem and can be used for navigation of the agent. This solver is used only for the estimation of the 3D pose in real-time, not for the optimization of a pose graph generated in the past. This means that it cannot be used in scenarios with large loops being closed.

SLAM subsystem

Given the joint values, attitude, and initial pose (see fig. 1.5), the data from the LIDAR scan are converted into point cloud of scan endpoints. The data from LIDAR can be downsampled and filtered. For example, the authors state that if the z coordinates of endpoints are not within a given threshold of the intended scan plane, they are filtered out by the HectorSLAM. This filtering is useful for our case as we use UAV, which can sometimes be in attitude in which it could scan ground instead of a wall or other obstacles.

Multi-level Map representation

HectorSLAM uses Gauss-Newton method for optimization of scan alignment inspired by work [21]. Gauss-Newton method can get stuck in a local minimum. To diminish this problem, authors use multiple occupancy grids with a descending resolution where every coarser map has half the resolution of the preceding map with higher resolution. Each map with the different resolution is kept in memory and updated every time the alignment process generates the pose estimates. The process of aligning the new scan with the current occupancy grid map starts with estimating a pose in the map with the lowest

resolution. The resulting estimated pose is iteratively used in maps with higher resolution to refine the estimate and achieve higher precision.

Navigation filter

HectorSLAM uses an Extended Kalman Filter to estimate the pose in 6DOF. The article [17] also states that scan matching provides only 2D position and orientation in the plane. An additional height sensor is needed for a full 3D estimation of the pose.

1.5 Scalability

As is described in the article [2], the definition and implementation of the pairwise map merging method can be easily extended to solve the multiple-map merging problem. The merging of two maps M_1 and M_2 gives us a new map, which we will call $\tilde{M}_2$. Let us use $\times$ in this section as a symbol for the merging operation of two overlapping maps. Let the m be the number of local maps to be merged into one global map. The first two local maps can be merged as follows:

$$\tilde{M}_2 = M_1 \times M_2. \tag{1.1}$$

We can then gradually merge a newly created map $\tilde{M}_n$ with next map M_{n+1}:

$$\tilde{M}_{n+1} = \tilde{M}_n \times M_{n+1}, \tag{1.2}$$

until we merge all m maps to one global map. The map $\tilde{M}_m$ is the resulting map of merging. By iteratively repeating this pairwise merging step on all maps, we can get the final merged map. Therefore we only need to implement the pairwise merging process, which can then be easily scaled to merge any number of maps. By easily scaled, we mean that merging three or more maps can be solved by merging maps in pairs, as described above. This serial pairwise merging can be further optimized. We could, for example, distribute the process on more than one UAV instead of merging all maps on one UAV. We could also represent each UAV as a node in a graph and connect the UAVs so that they form a directed chordless cycle. Each UAV would then need to merge only its map with the map of the next UAV connected by the cycle edge. The optimization of the sequence in which the multiple-merging process will be carried out is a difficult task that can be further explored in future work. We will mainly discuss the pairwise merging in this thesis.

1.6 Contribution

The contribution of the thesis is:

- Survey of available map merging techniques suitable for building a global map by merging local occupancy grid maps of multiple UAVs.
- Occupancy grid map merging method implementation with a 3D pose estimate of UAVs relative to each other. Our method does not make any geometric assumptions about the environment.
- Integration of the map merging method into the MRS group software architecture based on Robot Operating System (**ROS**) middleware [27].

Chapter 2

Map merging methods survey

This chapter defines map merging and describes methods for merging occupancy grid maps (**OGM**) obtained from HectorSLAM solver.

2.1 Map merging definition

First, let us formally define what we mean by a 2D map and unify the definition of map merging terminology for all methods in this chapter.

We assume that the occupancy grid map M is a matrix with r rows and c columns. Each cell $M(i, j)$ contains belief value (also called occupancy value), which indicates if the cell is occupied, unoccupied, or its state is unknown. We will use the same values for belief state as in the ROS definition of occupancy grid[1]. The values in range $[0, 100]$ indicate the probability of the cell occupancy. The value 100 represents the belief that the cell is occupied, 0 represents the belief that the cell is unoccupied, and the value -1 represents an unknown state. The used HectorSLAM discards probabilistic information in the cells and encodes the occupancy belief of each cell only with the three values -1,0 and 100 for "unknown," "unoccupied," and "occupied" cell.

Indices i and j of each cell $M(i, j)$ also indicate the spatial location of the cell because each OGM representation in ROS contains its map resolution as a length of side of each uniform square cell in meters. We can get a cell's position in a local map by multiplying the indices with the map resolution, which gives us distance from map origin in meters.

To merge two occupancy grid maps M_1 and M_2, we need to find the best rigid transformation T given some evaluation function. We assume that the deficiencies of the OGM like spatial distortion or map scale are insignificant, so that non-rigid transformations like scaling or shearing do not have to be used. The transformation T rotate the OGM M_2 by a given angle θ and translate it by given translation (t_x, t_y) so that it overlaps with M_1. This transformation

[1] definition of occupancy grid can be found in the documentation of ROS API: http://docs.ros.org/melodic/api/nav_msgs/html/msg/OccupancyGrid.html

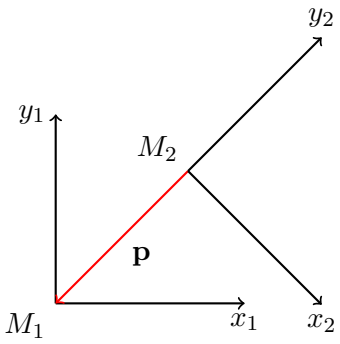


Figure 2.1: We are looking for a relative pose $\mathbf{p}$ by which we can transform the occupancy grid map M_2 so that it overlaps with the M_1.

can be represented by a relative pose $\mathbf{p} = [t_x, t_y, \theta]^T$ of the OGM M_2 in the M_1 as is shown in the figure 2.1. Thanks to easy conversion between relative pose $\mathbf{p}$ and transformation T, the terms will be used interchangeably to suit the context throughout this thesis. We can achieve transformation of the whole grid map by transforming each cell by given relative pose $\mathbf{p}$. This transformation of map can be defined by multiplication with transformation matrix as in paper [2]:

$$T_{t_x,t_y,\theta}(x,y) = \begin{bmatrix} \cos\theta & -\sin\theta & t_x \\ \sin\theta & \cos\theta & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}. \tag{2.1}$$

This notation is a little tricky because the multiplication of 3 by 3 matrix with three component vector gives us again three component vector. We would like this transformation of two component point to return again two component vector. Because of this reason, we will use the notation defined in the PhD thesis [38] called the compounding notation in which the transformation of coordinates (x, y) by translation (t_x, t_y) and rotation θ is defined as follows:

$$\begin{bmatrix} t_x \\ t_y \\ \theta \end{bmatrix} \oplus \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} x\cos\theta - y\sin\theta + t_x \\ x\sin\theta + y\cos\theta + t_y \end{bmatrix}. \tag{2.2}$$

The symbol $\oplus$ is used for compounding operation. We will follow the compounding notation for the rest of this thesis. The notation will be further expanded in the implementation chapter 3.

Suppose we have occupancy grid map $M = \{(\mathbf{c_1}, o_1), (\mathbf{c_2}, o_2), \ldots, (\mathbf{c_n}, o_n)\}$, where $(\mathbf{c_i}, o_i)$ is i-th cell of the map M with position $\mathbf{c_i}$ in the grid and the occupancy state o_i (note that bold lowercase letters like $\mathbf{c_i}$ denotes vector). To transform the map M by pose $\mathbf{p} = [t_x, t_y, \theta]^T$ we can use defined compounding operation $\oplus$ like this:

$$\mathbf{p} \oplus M = \{(\mathbf{p} \oplus \mathbf{c_1}, o_1), (\mathbf{p} \oplus \mathbf{c_2}, o_2), \ldots, (\mathbf{p} \oplus \mathbf{c_n}, o_n)\}. \tag{2.3}$$

Now that we have defined how to transform map by pose with the compounding operation, we can define the general goal of occupancy grid merging of two maps. Methods for merging OGM usually consist of the objective function, which represents a consistency of maps and an optimization technique whose task is to search for optimal map alignment that maximizes the objective function.

Definition 2.1. Given objective function of map merging Δ and two OGMs M_1 and M_2 we search for optimal relative pose $\hat{\mathbf{p}}$ such that:

$$\hat{\mathbf{p}} = \arg\max_{\mathbf{p}}(\Delta(M_1, \mathbf{p} \oplus M_2)). \tag{2.4}$$

The objective function Δ takes as a parameter two occupancy grid maps and returns a value indicating how well the maps overlap. The higher the value, the better they overlap. Each merging method differs from others in its objective function Δ, and the method for optimizing the relative pose $\mathbf{p}$. The objective function is in some merging methods called heuristic function as it provides information on whether the search for the optimal relative pose is going towards the goal.

In the rest of this chapter, we will introduce several map merging methods with their objective functions and methods for optimizing relative pose. As was explained in section 1.5 the pairwise map merging can be easily scaled to map merging with a higher number of maps. Therefore, we discuss only the pairwise map merging problem in the described methods.

2.2 Random Walk method

The first considered method in article [2] uses a random walk to explore a space of relative poses and a heuristic function with combination of the Manhattan distance and a "locking" parameter.

Heuristic function

The heuristic function of this method consists of two parts:

$$\Delta(M_1, M_2) = \psi(M_1, M_2) + c_{\text{lock}} \cdot (\text{dis}(M_1, M_2) - \text{agr}(M_1, M_2)). \tag{2.5}$$

The first part which is named ψ computes the Manhattan distance (md) between cells with same occupancy value (**OV**):

$$\psi(M_1, M_2) = \sum_{c \in C} d(M_1, M_2, c) + d(M_2, M_1, c), \tag{2.6}$$

$$d(M_1, M_2, c) = \frac{\sum_{M_1[\mathbf{p_1}]=c} \min\{\text{md}(\mathbf{p_1}, \mathbf{p_2}) | M_2[\mathbf{p_2}] = c\}}{\#_c(M_1)}, \tag{2.7}$$

where d function sums all minimal Manhattan distances from cells with OV (occupancy value) c in M_1 to cells with the same OV c in M_2 and divides this sum by number of cells with OV c in M_1. The function ψ can be computed in linear time. The Manhattan distance is the distance between two points given by absolute differences of their coordinates.

The agr(M_1, M_2) and dis(M_1, M_2) in the second part of heuristic function (2.5) is the number of cells in M_1 and M_2 where there is agreement (respectively, disagreement). The higher value of constant $c_{\text{lock}} \geq 0$ allows for smaller amounts of overlap needed for maps to be correctly merged, but it slows down the convergence to correct merging. The author explained the heuristic function simply: "Only ψ provides meaningful gradients for the motion planning, whereas dis(M_1, M_2) - agr(M_1, M_2) only 'locks' the two maps in place as soon as the identical regions are aligned." The one disadvantage of this heuristic function is that we have to choose the constant c_{lock}, which can be hard to tune for this method to be fast and robust in different environments with different conditions. All given equations are explained in detail in the article [2].

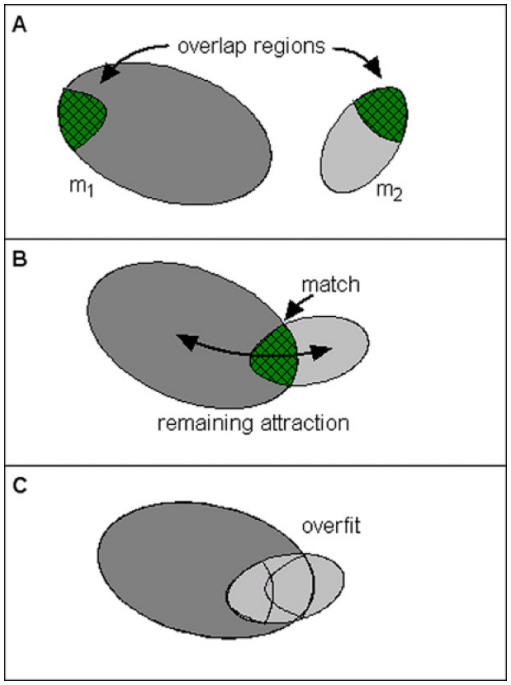


Figure 2.2: The heuristic function consists of two parts. The ψ generates a kind of a attraction force between the maps. The second part with the scaling factor c_{lock} helps the two maps to "lock" in an aligned state and prevents the method from overfitting. Reprinted from [2, figure 3]

Optimization method

Let us now describe the random walk algorithm used in this merging method. It is given some starting relative pose $\mathbf{p_s}$. First it computes the heuristic

value $c_k = \Delta(M_1, \mathbf{p_s} \oplus M_2)$. Then in a loop, the algorithm generates a new relative pose $\mathbf{p_t}$ using a Gaussian distribution as described in the article and computes its heuristic value c_t as with the $\mathbf{p_s}$. If the heuristic value of newly generated pose is better then the current best pose ($c_t > c_k$), then the newly generated pose becomes the best pose ($\mathbf{p_s} = \mathbf{p_t}, c_k = c_t$). If the newly generated pose does not have a better heuristic value than the current value (it does not improve the overlapping), it still has a slight chance to be accepted and assigned to $\mathbf{p_s}$. The reason for accepting even pose with worse overlapping, which is a behavior similar to simulated annealing, is to avoid getting trapped in local minima as in the case of the hill-climbing optimization.

Interesting ideas

Authors suggest how to **identify failure** of merging method when there is no overlapping or when the method did not have enough time to find global optimum. To rule out cases where the merging of M_1 and M_2 failed, they came up with **acceptance indicator** ai:

$$ai(M_1, M_2) = \frac{agr(M_1, M_2)}{agr(M_1, M_2) + dis(M_1, M_2)}. \tag{2.8}$$

Only if this acceptance indicator is close to 1.0, it indicates that merging was successful. It is shown in experiment, that the successful attempts lead to ai with value well above 0.98, while failed attempts with the best ai had value well below 0.9.

An interesting idea mentioned in the article (in III. A. Overview) is that the search for the transformation of one map to align and merge with another map correctly is similar to the docking problem of a protein and a ligand in computational biology. This idea could be inspected in future research.

2.3 Hough Transform method

The next method [4] uses Discretized Hough transform (**DHT**). It is a method for the detection of primitive shapes. In our case, it is used to detect line-like objects in the occupancy grid map. This method can detect more complex shapes like circles or ellipses, but authors focus only on the detection of lines due to their intention of using this method primarily for mapping building interiors where linear features like walls are abundant. This algorithm returns a set of possible transformations with the respective objective function values. This choice of returning multiple possible transformations instead of the "best" one is due to the possibility of the transformation with the highest objective value being the wrong transformation that does not align maps correctly. We can use a more suitable objective function to choose the correct transformation.

The main idea behind DHT is that a line has its polar representation $x\cos\theta + y\sin\theta = \rho$. If we discretize θ to uniform steps in the interval $[0, 2\pi)$ and choose the interval of ρ to accommodate expected line segment length, we can then represent DHT by a matrix $HT_{\mathcal{M}}$ with ρ_S rows and θ_S columns. First, the occupancy grid has to be converted to a binary image before DHT. The conversion can be done by setting occupied cells to 1 and other unknown or free cells to 0. The DHT is explained in [7], and we will try to explain this computation simply. The values of the matrix $HT_{\mathcal{M}}$ are computed by first setting all elements to zero. Then for every pixel (x_0, y_0) in binary image with value 1, compute the ρ value in the polar representation $x\cos\theta + y\sin\theta = \rho$ for all θ_S discrete values and increment the cell in the DHT matrix with indexes closest to computed values of ρ and θ.

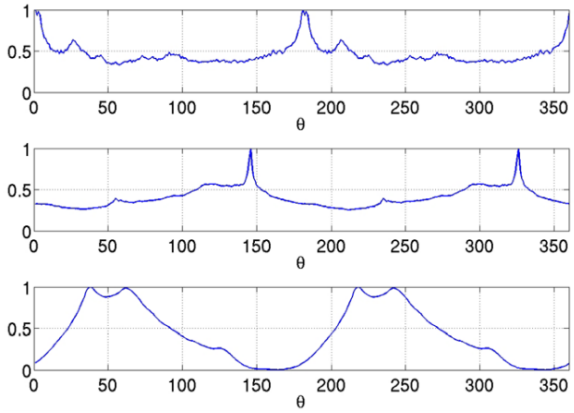


Figure 2.3: Two Hough spectra in the two top panels with their cross correlation in the third one normalized to the range 0-1. Reprinted from [4, figure 3]

We can then compute a Hough Spectrum $HS_{\mathcal{M}}$ of both occupancy grids being merged. It is computed from their $HT_{\mathcal{M}}$ matrix created by the DHT as a signal with sampling period $2\pi/\theta_S$:

$$HS_{\mathcal{M}}(k) = \sum_{i=1}^{\rho_S} \mathrm{HT}_{\mathcal{M}}(i,k)^2 \qquad 1 \leq k \leq \theta_S. \tag{2.9}$$

We can also compute circular cross correlation $CC_{\mathcal{M}_1\mathcal{M}_2}$ of two Hough spectra with same sampling period:

$$CC_{\mathcal{M}_1\mathcal{M}_2}(k) = \sum_{i=1}^{\theta_S} \mathrm{HS}_{\mathcal{M}_1}(i)\mathrm{HS}_{\mathcal{M}_2}(i+k) \qquad 1 \leq k \leq \theta_S. \tag{2.10}$$

Example of the circular cross correlation is shown in figure 2.3. We can then extract multiple local maxima from the $CC_{\mathcal{M}_1\mathcal{M}_2}$, which are associated with a possible optimal rotation for the correct overlap of two occupancy grids.

With possible rotations extracted, we still need to find candidates for optimal translation (t_x, t_y). We can get t_x and t_y separately by extracting two more Hough spectra from binary images of the first map and the second

map rotated by each extracted rotations and then computing circular cross-correlation for X-spectra and Y-spectra defined in the article. The algorithm for computing hypothesis in the article then chooses the translation with the best cross-correlation. For all possible optimal rotation it finds best translation and returns possible transformations $\mathbf{p_1}, \mathbf{p_2}, \ldots, \mathbf{p_n}$ with their respective objective function values $\Delta_1, \Delta_2, \ldots, \Delta_n$.

This approach works well for merging maps with many linear features like walls of buildings. Authors also state that if the maps being merged contain at least one long wall, the algorithm will be capable of merging the maps. The article shows an experiment with robots mapping outdoor areas with less linear features where the maps were still successfully merged, but experiments with no linear features were not shown.

2.4 Genetic-Algorithm-Based methodology

The method in [20] is unique in a way that it uses the strategy of the evolutionary Genetic Algorithm as an optimization technique to search for optimal relative pose $\hat{\mathbf{p}}$.

Heuristic function

The objective function $\Delta(M_1, M_2)$ measuring the consistency of two maps, in this case, differs from the other two presented approaches. It uses only the occupied cells (in an article called local maximum occupied cells) to measure consistency. Authors explain this choice with the idea that occupied cells most likely correspond to objects on the map. An object (occupied cells) in both maps increases the Δ only if it tends to be close to its corresponding image in the other map. As authors say, it should also be affected less by inconsistencies of maps caused by the used SLAM algorithm than methods computing consistency of all cells in a map.

The used heuristic function is defined like this:

$$\Delta(M_1, \mathbf{p} \oplus M_2) = \sum_{\mathbf{c} \in C(M_2)} \psi_{M_1}(\mathbf{p}, \mathbf{c}), \tag{2.11}$$

$$\psi_{M_1}(\mathbf{p}, \mathbf{c}) = \begin{cases} M_1(\mathbf{p} \oplus \mathbf{c}) & \text{if } \mathbf{p} \oplus \mathbf{c} \in \mathrm{Occ}(M_1) \\ 0 & \text{otherwise} \end{cases}. \tag{2.12}$$

The $\mathrm{Occ}(M_1)$ is a function that returns all grid cells with occupancy value higher than some chosen threshold. This function could be used with SLAM solvers that return the occupancy grid with a probabilistic representation of cells state, compared to HectorSLAM that uses only three values to represent the state of the cell. Authors, for example, chose this threshold for occupied cells to be 0.6. The whole process of computing this heuristic function could

be simply described as this. Each occupied cell in map M_2 is transformed by pose $\mathbf{p}$. If the cell in map M_1 on coordinates of the transformed cell is occupied (above the Occ threshold), add its value to the total sum of Δ value.

Optimization method

This method is inspired by genetic algorithm (**GA**) heuristic search methodology [28]. A biological analogy inspires the idea of GA. Natural selection favors the strongest individuals in a population as they are more likely to survive and transfer their genes to the next generation. The new generation inherits the genetic traits of their parents that were naturally selected. The traits can be inherited by genetic operations like *mutation* and *crossover*.

The population in our case consists of relative poses between which we are looking for the optimal pose $\hat{\mathbf{p}}$ which maximizes the objective function $\Delta(M_1, \hat{\mathbf{p}} \oplus M_2)$. The objective function serves as an evaluation of the poses population. The evaluated population can be sorted into strong and weak groups of individuals, and the process of natural selection can create the next generation. The genetic operations work with the position and orientation of pose. The *mutation* takes one individual and creates a new one by making a little change of position and orientation values. This method uses two designs of crossover operation. Both crossovers take two poses from the previous generation and create one new pose. First, *crossover_I* takes the position of first pose and orientation of the second pose and mixes them to create a new pose. The second crossover method *crossover_II* makes a random linear combination of the position and orientation of the two given poses. The last option of making a new individual is by initializing it to a random value, which serves as a means of preserving diversity in the population.

Interesting ideas

The article also described how the UAV_1 which is merging the maps could estimate the position of the UAV_2 in the merged map with the found relative pose $\mathbf{p}$ of the two maps M_1 and M_2 in combination with the local pose of the second UAV in its local map M_2. Authors also outlined how this knowledge of vehicle to vehicle relative pose (**V2V RP**) can be used with cameras for multi-vehicle perception association. In other words, each vehicle can see through the cameras of other vehicles and even "see through" these vehicles, as was illustrated in the experiments. This problem of "seeing through" other vehicles is called cooperative augmented reality, which two of the authors of this article explained in [19]. We chose the method described in this section for our implementation of the map merging algorithm. It will be described in more detail in chapter 3.

Figure 2.4: The occupancy grid map merging can also be used for estimation of vehicle to vehicle relative pose. With this pose known, we can for example use other vehicles perception to "see" through them. Reprinted from [19, figure 5]

2.5 Neural Networks-Based approach

The method in [32] uses neural networks to downscale the occupancy grid map (OGM) representation into cluster points to speed up the process of map merging. We start by describing the optimization method first.

Optimization method

This method consists of multiple steps, shown in figure 2.5(top left). First, a map segmentation is performed on both occupancy grids, depicted in figure 2.5(bottom). This segmentation divides occupied cells into groups of cells that are close together. This is done, so the computation of objective function does not have to check every cell, and a faster heuristic can be used. Then a neural network called self-organizing map (SOM) is trained. Its structure can be seen in figure 2.5(top right). The input of the SOM is the 2D location of occupied cells. It is trained so that the weights of each neuron represent coordinates of clusters in the occupancy grid. These clusters represent features of a map in a much more compact way than the occupied cells do, and the search for the relative pose of maps can now be performed efficiently.

The third step is to find first the relative orientation. It is found by generating a 360-degree histogram of each cluster surface norms, which are explained in the article and then by tuning these histograms with the Radon transform, that is similar to the Hough transform with one difference. The Radon transform, as is explained in [33], is the projection of the image intensity along a radial line. Similar to the Hough transform method described in 2.3,

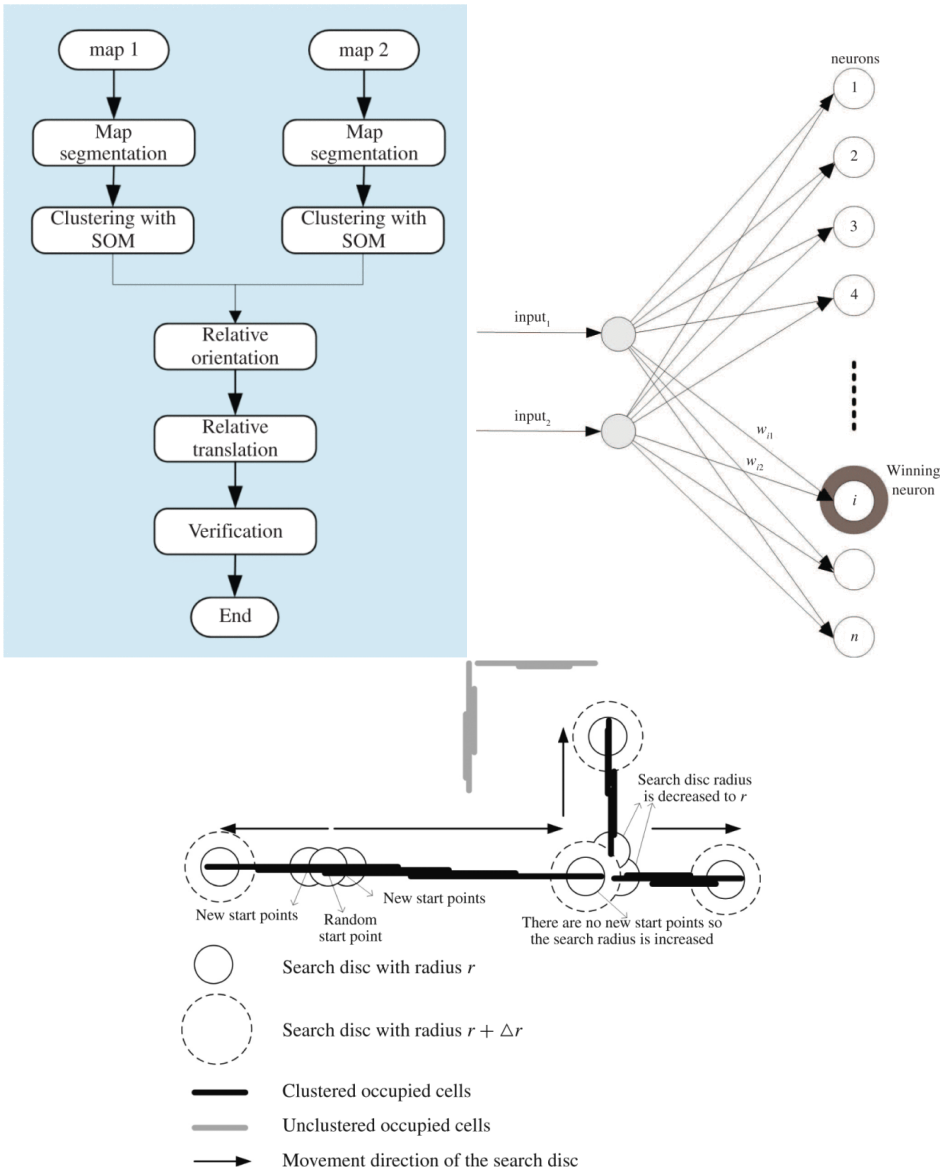


Figure 2.5: (top left) Flowchart of the proposed algorithm **(top right)** Structure of self-organizing map (SOM) neural network **(bottom)** A map segmentation process. Reprinted from [32, figures 1, 2, 3]

we get the orientation.

Now that we have the occupancy grids aligned, the points with the same orientation determined by their norm are probably the same in the real world. This information is used in an algorithm similar to the iterative closest point, which finds the relative translation. The search for the relative translation of this method can be accelerated, as is described in section 2.6.

Heuristic function

Similar to the Hough method 2.3 this method does not use any specific heuristic function during the search for relative pose orientation. It uses objective function only as a final verification of correct map merging or to choose the best candidate between found relative pose translations. The verification method of cluster convergence that is used during the selection of relative translation is computed like this:

$$J = \sum_{i=1}^{n} \|p_1^{\{i\}} - p_2^{\{i\}}\|. \tag{2.13}$$

Where p_1 and p_2 are points of two corresponding clusters in M_1 and M_2. The n is a total number of clusters. As authors explain "The J is the sum of the squared Euclidean distances between corresponding matching points."To ensure accurate merging of whole maps the same acceptance indicator as in the case of Random Walk method 2.2 is used:

$$ai(M_1, M_2) = \frac{agr(M_1, M_2)}{agr(M_1, M_2) + dis(M_1, M_2)}. \tag{2.14}$$

Interesting ideas

Downscaling the occupancy grid map into cluster points is an interesting way to speed up the map merging process. It can also recognize different shapes of features, not only lines like in the case of the Hough transform method.

2.6 Probabilistic Generalized Voronoi Diagram method

This method [31] uses Radon transform to find the relative rotation of two OGMs similar to the neural network-based approach in section 2.5. It also refers to the article [32] of the neural network-based approach as its inspiration for the search of relative rotation. The novelty of this approach is in the use of probabilistic Generalized Voronoi Diagram (**GVD**) to find the relative translation of two OGMs and the use of entropy filter for maps fusion.

Optimization method

As was said, this method uses Radon transform to find the relative rotation of two OGMs. It then, in contrast with other methods, incorporates uncertainty to the rotated map. This uncertainty of rotation is represented by the Gaussian distribution approximation of non-Gaussian distribution in the covariance matrix.

With maps aligned, the next step is to find the relative translation. The method first finds the Generalized Voronoi Diagram (GVD) using mathematical morphological operations on the binary representation of the occupancy grid map (OGM). The probabilistic version of GVD can be generated to incorporate the probabilistic information of OGM into the GVD. The probabilistic GVD is useful only in the case that OGM is represented by probabilistic occupancy state and not like in case of HectorSLAM only by "free", "occupied", and "unknown" state. The GVD skeleton is then divided into edges by applying a dilation mask on vertices of the skeleton. The vertices are cells with at least three edges of the skeleton coming out of these cells. The dilation mask goal is to delete enough surrounding occupied cells of vertices so that the edges have gaps big enough between each other. Each edge is represented as a matrix with dimensions large enough for edge to fit in, as shown in figure 2.6.

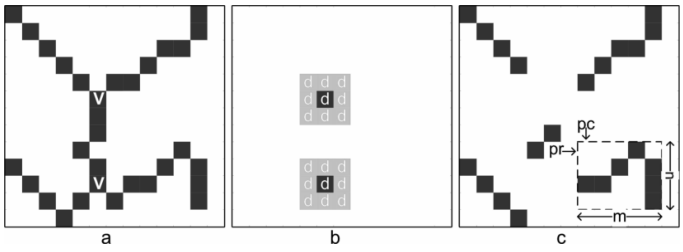


Figure 2.6: **a)** A GVD with vertices marked by "v" **b)** Dilation mask on vertices **c)** Edges with representation of matrix marked by dashed line. Reprinted from [31, figure 3]

These probabilistic edge matrices of two maps are then used in a matching process which finds the best relative translation between the maps. The matching process uses a 2D cross-correlation. After finding a relative translation, the uncertainty of this translation is incorporated, as in the case of rotation uncertainty. The last step after transforming the map is to fuse both maps.

The fusion is done by first combining and filtering probabilities of both maps into one fused map M_{fused}. Then the entropy filter H is applied to the M_{fused}. For cell $M(i,j)$ with probability $p(M(i,j)) = p_{ij}$:

$$H(M(i,j)) = -p_{ij}\log p_{ij} - (1-p_{ij})\log(1-p_{ij}). \tag{2.15}$$

The difference of both map entropies called *mutual information* is computed:

$$I_{ij} = H(M_1(i,j)) - H(M_{\text{fused}}(i,j)). \tag{2.16}$$

With values from M_{fused} applied only when the mutual information is positive, the final map M_{final} cells look like this:

$$M_{\text{final}}(i,j) = \begin{cases} M_{\text{fused}}(i,j) & I_{ij} \geq 0 \\ M_1(i,j) & I_{ij} < 0 \end{cases}. \tag{2.17}$$

Interesting ideas

The use of uncertainty is an interesting way of making the merging process more robust. The experiments described in the article also show that this method is at least eight times faster then the random walk method described in section 2.2 with comparable accuracy of map merging.

Chapter 3

Implementation

This chapter describes the chosen method in more detail, as well as the reason for its selection. Furthermore, we describe the ROS middleware and the integration of the implemented method into the MRS group software architecture based on the ROS. Finally, we present a diagram of the whole map merging process pipeline.

3.1 Method of map merging

As we have shown in the short survey of methods in chapter 2, there are many completely different approaches to the problem of map merging. Each of them has some drawbacks and some benefits. We will now inspect which method will best suit our problem domain.

3.1.1 Requirements on the method

We choose to use HectorSLAM described in section 1.4 as a SLAM algorithm for occupancy grid mapping. The inputs to our map merging method are the OGMs (occupancy grid maps) generated by HectorSLAM on each UAV of the swarm and their local pose in the map, which is also generated by the HectorSLAM navigation subsystem. The output on each UAV should be global OGM and the relative poses of other UAVs relative to the current UAV.

The method should also be usable even with some UAVs failing. Failure of one robot should not ruin the overall mission. Because of this, any type of leader/follower algorithm should not be used. The computation of map merging should be decentralized with each robot computing its own global map. This is possible with all of the described methods.

3.1.2 Choice of method

We have described five suitable methods for map merging, which are:

1. Random Walk method (RW)
2. Hough Transform method (HT)
3. Genetic-Algorithm-Based methodology (GA)
4. Neural Networks-Based approach (NN)
5. Probabilistic Generalized Voronoi Diagram method (PGVD)

As was said in section 2.6, the PGVD was at least eight times faster than the RW approach. Deciding between the two approaches, we should pick PGVD. Also, the HT method works well only in areas with linear features like walls. The other methods do not have a problem like this without any significant disadvantage compared with the HT method. This leaves us with the last three methods.

Both the PGVD method and the NN approach use Radon transform to find the relative rotation of two OGMs. For finding relative translation, the NN uses an algorithm similar to the iterative closest point (**ICP**) algorithm. The PGVD uses as the name suggests the probabilistic generalized Voronoi diagram as an alternative to the ICP. Unfortunately, the PGVD article does not offer a comparison of the merging speeds of the two approaches.

The GA methodology has one inconvenience, which is neither disadvantage nor advantage. If we have prior knowledge of approximate relative pose, the relative pose's search space will be much smaller than if we have no prior knowledge. Another way around if we do not have prior knowledge, this method will have to search in the space of all possible relative rotations and translations. The other methods do not work with prior knowledge, so there is no advantage in knowing the approximate relative pose in their case. In a situation where the merging is used more than once, the GA can continuously optimize the approximation of the relative pose by using the estimate from the previous map merging for initialization of relative pose population.

The best approach would be to try more than one method, but that would go far beyond the scope and the time constraints of this thesis. Concretely, we implemented a genetic-algorithm-based method described in section 2.4.

3.2 Genetic-algorithm-based methodology in detail

The genetic-algorithm-based method was described briefly in section 2.4. We will now describe it with all details needed for implementation.

3.2.1 Compounding notation

We described part of the compounding notation in the section 2.1. Let us now complete the description following the notation in [38]. Given the two poses $[x_1, y_1, \theta_1]^T$, $[x_2, y_2, \theta_2]^T$, and a 2D point $[x_p, y_p]^T$, the compounding notation is defined as follows:

$$\begin{bmatrix} x_1 \\ y_1 \\ \theta_1 \end{bmatrix} \oplus \begin{bmatrix} x_2 \\ y_2 \\ \theta_2 \end{bmatrix} = \begin{bmatrix} x_2 \cos\theta_1 - y_2 \sin\theta_1 + x_1 \\ x_2 \sin\theta_1 + y_2 \cos\theta_1 + y_1 \\ \theta_2 + \theta_1 \end{bmatrix}, \tag{3.1}$$

$$\begin{bmatrix} x_1 \\ y_1 \\ \theta_1 \end{bmatrix} \oplus \begin{bmatrix} x_p \\ y_p \end{bmatrix} = \begin{bmatrix} x_p \cos\theta_1 - y_p \sin\theta_1 + x_1 \\ x_p \sin\theta_1 + y_p \cos\theta_1 + y_1 \end{bmatrix}, \tag{3.2}$$

$$\text{inv}\left(\begin{bmatrix} x_1 \\ y_1 \\ \theta_1 \end{bmatrix}\right) = \begin{bmatrix} -x_1 \cos\theta_1 - y_1 \sin\theta_1 \\ x_1 \sin\theta_1 - y_1 \cos\theta_1 \\ -\theta_1 \end{bmatrix}. \tag{3.3}$$

The inv() is a inversion function which works like a point reflection with center in the origin. As was defined in the section 2.1, to transform the occupancy grid map $\mathbf{M} = \{(\mathbf{c_1}, o_1), (\mathbf{c_2}, o_2), \ldots, (\mathbf{c_n}, o_n)\}$ by pose $\mathbf{p} = [\theta, t_x, t_y]^T$ we can use defined compounding operation $\oplus$ like this:

$$\mathbf{p} \oplus \mathbf{M} = \{(\mathbf{p} \oplus \mathbf{c_1}, o_1), (\mathbf{p} \oplus \mathbf{c_2}, o_2), \ldots, (\mathbf{p} \oplus \mathbf{c_n}, o_n)\}. \tag{3.4}$$

3.2.2 Fitness function

We also defined the objective function in the section 2.4 which we will call in terminology of genetic algorithms the **fitness function** and use symbol F as in the article [20] instead of Δ:

$$F(M_1, \mathbf{p} \oplus M_2) = \sum_{\mathbf{c} \in C(M_2)} F_{M_1}(\mathbf{p}, \mathbf{c}), \tag{3.5}$$

$$F_{M_1}(\mathbf{p}, \mathbf{c}) = \begin{cases} M_1(\mathbf{p} \oplus \mathbf{c}) & \text{if } \mathbf{p} \oplus \mathbf{c} \in \text{Occ}(M_1) \\ 0 & \text{otherwise} \end{cases}. \tag{3.6}$$

3.2.3 Optimization of relative pose

An evolution algorithm is used for the optimization of a relative pose. This optimization method iteratively evolves a population of poses based on a given fitness function. The process of finding the relative pose is composed of the initialization and evolution, as is defined in the article [20].

We will first clarify what we mean by local and global pose. The local pose of the UAV is the pose on the local map. We will explain how the pose is represented in the ROS middleware because we are using it as our environment for the implementation of this algorithm. The local pose of an agent in the OGM (occupancy grid map) is defined as a translation and rotation from the initial pose of an agent in the OGM. This initial pose is often set to the center of the map with zero rotation corresponding to the direction of the x-axis. However, in some missions, we have prior knowledge that the agent will move only in a certain direction. Thus, it is convenient to set the initial pose so that the agent has enough map space in the direction it will explore. The relation between initial pose and local pose is illustrated in figure 3.1.

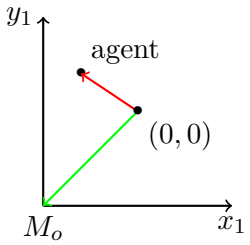


Figure 3.1: The local agent pose (red arrow) in the OGM is given by translation and rotation from the initial pose depicted by (0,0). When the initial pose is not in the occupancy grid map origin M_o the pose of the M_o (green arrow) in the ROS middleware has negative coordinates.

By the global pose of the UAV, we mean a pose in some higher frame of reference than the OGM of the UAV (for example, the pose in the global map of UAV swarm or pose received from the GPS in the frame of reference of the whole globe).

Initialization

Initialization consists of the computation of the initial pose $\mathbf{p}$ and the creation of the initial population.

1. Computation of initial pose $\mathbf{p_i}$:

   With *local* pose of UAV_1 and UAV_2 in map labeled as $\mathbf{L}_{\mathrm{UAV_1}}$, $\mathbf{L}_{\mathrm{UAV_2}}$ respectively and the *global* pose labeled as $\mathbf{G}_{\mathrm{UAV_1}}$, and $\mathbf{G}_{\mathrm{UAV_2}}$ the initial pose can be computed as follows:

   $$\mathbf{p_i} = \mathbf{L}_{\mathrm{UAV_1}} \oplus \mathrm{inv}(\mathbf{G}_{\mathrm{UAV_1}}) \oplus \mathbf{G}_{\mathrm{UAV_2}} \oplus \mathrm{inv}(\mathbf{L}_{\mathrm{UAV_2}}). \tag{3.7}$$

   Nevertheless, the approximation of global pose is usually unknown to us, as explained in the introduction. We have to initialize the relative pose of the two maps using a different approach.

Let us describe possible solutions for generating the initial approximation of the relative pose without knowledge of the UAVs global poses.

Without any knowledge of the UAVs relative pose the easiest solution to implement is to approximate the initial pose with zero vector:

$$\mathbf{p_i} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}. \tag{3.8}$$

But this yield inconsistent results in maps that have different sizes and in maps where the UAVs initial pose is in different location of map. For this reason we will initialize the relative pose $\mathbf{p_i}$ of the OGMs so that the initial poses of both UAVs are on the same spot in the map as is illustrated in the figure 3.2.

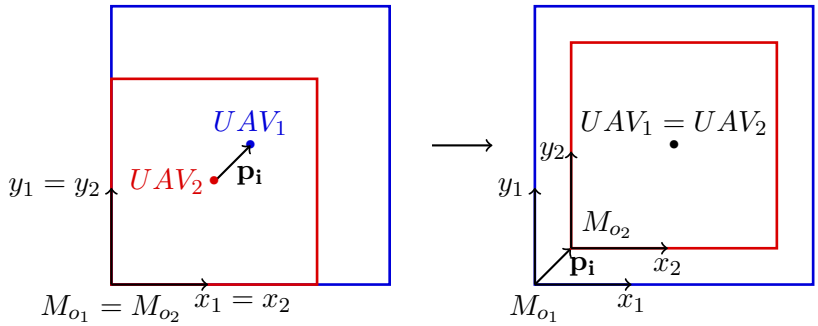


Figure 3.2: The $\mathbf{p_i}$ is the initial pose of the received red occupancy grid map in the reference system defined by the origin of the blue occupancy grid map.

2. Initialization of population:

   Authors randomly initialize the population of poses $\{\mathbf{p}_{(k)} | k = 1, 2, \ldots, n\}$ within an error range around the initial pose $\mathbf{p_i}$ with position error being ± 30 m and orientation error being $\pm 30°$. The maps can only be merged if there is an overlap between them, which means that the largest possible transformation of the map would be a translation by the length of the diagonal of the largest OGM being merged and rotation of $\pm 180°$. The error range in which the population is generated should be chosen according to the initial location of the UAVs. We set the position error to be a circle with a radius equal to half of the UAVs occupancy grid map longest dimension and with a center in the initial pose position. The orientation error is, in our case $\pm 180°$. This is a much larger error range than the one used in the experiments proposed in the article.

Iteration

The evolution process begins after the initialization of the first population. The evolution can be divided into epochs. Each epoch starts by acquiring

the current state of the occupancy grid maps. The epoch then iteratively performs the steps of evolving one generation of the population into the next generation. Each iteration performs the following steps to form the next generation:

1. Get likelihood of each individual in population by computing its fitness function as is explained in section 3.2.2.

2. Compute mean likelihood of the population:

$$F_{\text{mean}} = \frac{1}{n}\sum_{k=1}^{n} F(M_1, \mathbf{p}_{(k)} \oplus M_2), \tag{3.9}$$

and divide the population into a superior group which has higher or equal likelihood than the F_{mean} and inferior group with lower likelihood than the mean:

$$\{\mathbf{p}_{(\text{superior})}\} = \{\mathbf{p}_{(i)} | F(M_1, \mathbf{p}_{(i)} \oplus M_2) \geq F_{\text{mean}}\} \tag{3.10}$$

$$\{\mathbf{p}_{(\text{inferior})}\} = \{\mathbf{p}_{(j)} | F(M_1, \mathbf{p}_{(j)} \oplus M_2) < F_{\text{mean}}\} \tag{3.11}$$

This division is used in order to find superior poses which will probably merge map better and should be propagated into the next generation with no to little change. The inferior group will be replaced with newly generated poses. Now comes the genetic part of the genetic algorithm.

Genetic operations. Before we describe next steps of iteration let us first describe the used genetic operations. Authors of the article [20] use three operations called *mutation*, *reinitialization*, and two versions of *crossover*:

- **Mutation** changes the position and orientation of a pose by a minor margin. We for example use normal distribution to generate the change of orientation in degrees with 0° mean and standard deviation of 4° to generate change mostly between ±4°. The error in position is in our case generated by normal distribution with mean of 0 cells and standard deviation of 20 cells. This position change is given in length of square occupancy grid cells side so the pose position translate mostly between ±20 cells side lengths.

- **Crossover I.** gets position of one pose and orientation of other pose and combines them together to create a new pose:

$$\mathbf{p}_{\text{new}} = \textbf{crossover\_I}(\mathbf{p}_1, \mathbf{p}_2) = [x_{\mathbf{p}_1}, y_{\mathbf{p}_1}, \theta_{\mathbf{p}_2}]^T. \tag{3.12}$$

- **Crossover II.** generate random number λ in interval $[0, 1]$ and do a linear combination of two poses:

$$\mathbf{p}_{\text{new}} = \textbf{crossover\_II}(\mathbf{p}_1, \mathbf{p}_2) = \lambda\mathbf{p}_1 + (1 - \lambda)\mathbf{p}_2. \tag{3.13}$$

- **Reinitialization** generates individual pose as in the initialization of population by randomly mutating the initial pose $\mathbf{p}_i$ by an error defined in the initialization of population step.

Now that we know how the basic genetic operations work we can continue to the next step of iteration.

3. Mutate superior group:

   We create mutated individual out of each member of the elite group, evaluate its fitness function, and if the mutation has a higher likelihood value, we replace the individual with this new mutation. Authors mutate the individual with the highest likelihood value (best individual) 100 times and other individuals only once. We tried to enhance the algorithm's performance by mutating the other individuals in the superior group more than once based on their likelihood compared with the likelihood of the best individual. Given the likelihood of the *best individual* l_{max}, likelihood of the *worst individual* in the superior group l_{min} and likelihood of the individual from the superior group *currently* being mutated l_{cur} the *number of attempts* n is computed like this:

$$n = max\left(\left\lfloor 100\left(\frac{l_{cur} - l_{min}}{l_{max} - l_{min}}\right)^{10}\right\rfloor, 1\right). \tag{3.14}$$

   This gives us integer from interval $[1, 100]$. Individuals with higher likelihood will have an exponentially higher number of attempts for mutation. The problem of this "enhancement" is that as we converge to the correct solution of finding the best relative pose for map merging, the number of individuals with similar likelihood values as the best individual increase rapidly. This means that many individuals would get close to 100 attempts for mutation, and the optimization process of finding the best relative pose would slow down rapidly. Therefore, we returned to the method used in the article, which mutates only the best individual 100 times and other individuals only once.

4. Replace inferior group:

   The inferior group is replaced by performing randomly selected (in our case with uniform distribution) genetic operations on randomly selected individuals from the superior group. As was described, we can replace the individual in the inferior group by mutating an individual from the superior group, doing a crossover I. or crossover II. on two individuals from the superior group, or the individual can be reinitialized as in initialization of population.

We described the process of evolving one generation into the next generation. We will now describe how the evolution process works as a whole.

Evolution

The whole evolution process starts by initializing the initial population generation. Then the first epoch starts. Each epoch consists of several iterations. After each epoch, the algorithm publishes its results of merging consisting of two parts. The first part is the merged occupancy grid map. It is a result of merging the received OGM transformed by relative pose with the highest likelihood value and the map of the UAV on which the merging process is executed. The second part of the merging process result is the approximation of a relative pose between the two UAVs whose maps were merged. The map merging method continuously accepts the updated occupancy grids and UAVs local poses at the beginning of each epoch.

The choice of the number of iterations in each epoch depends on how the users of this algorithm want it to behave. The epoch starts by preparing its data for the iteration process, which causes a significant overhead. This involves, for example converting the two occupancy grid maps into an image and the process of extracting a set of all occupied cells in each OGM. This process is mirrored when the epoch ends as we have to convert the image of the map back into the OGM. So users of this algorithm have to choose how they want to balance the frequency of epochs and speed of convergence to the optimal relative pose. With a higher frequency of epochs, the frequency of updates of the UAVs relative pose is increased, which can be preferred, for example, in a situation where we want to get the approximation of relative pose fast, but it does not have to be so precise. On the other hand, with the lower frequency of epochs, the overhead is decreased, and the algorithm's performance is increased, which could lead to faster convergence to the optimal relative pose between the two occupancy grid maps. However, in this case, the updates of the relative pose would be less frequent.

The computational power of the used UAV for map merging should also be considered in the choice of the frequency of epochs.

The next important question is, how should we use the approximation of the relative pose of the two maps and the evolved generation from the last epoch in the next epoch? We consider two approaches. One described in the article [20] which uses the evolved generation from last epoch without any alteration. For the second approach, we choose to reinitialize the generation around the approximation of the relative pose from the last epoch.

3.2.4 Terminating condition

The terminating condition of the algorithm is not exactly specified in our implementation, and the process runs until the UAV interrupts the merging of maps by itself, for example, after landing due to finishing its mission or depleting its batteries. We implemented the computation of the acceptance index explained at the end of section 2.2, which could be used as a terminating condition for the map merging by selecting some high threshold which if

passed by acceptance index would end the process. We decided to not use the ai because unlike in article [4] the ai of two maps with correct pose in our experiments does not always have value well above 0.9 and therefore cannot be distinguished from incorrect map merging which has value ai in our case somewhere between 0.8 and 0.9.

3.3 On-demand merged map output with a 2D pose estimate

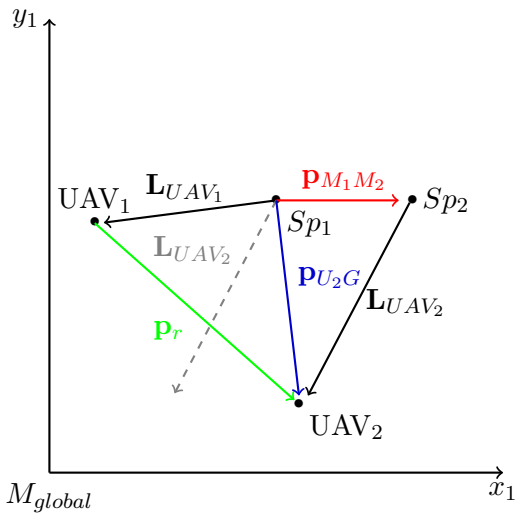


Figure 3.3: Depiction of the poses described in the section 3.3. The poses are depicted in the space of global map M_{global}. The Sp_1 and Sp_2 are starting positions of the UAV$_1$ and UAV$_2$ respectively.

As was described in section 3.2.3, the merging algorithm on one UAV returns the merged map and the pose estimate of the other UAV relative to the current UAV. The HectorSLAM provides the 3D local pose of UAV in its local OGM. This pose contains all 6 degrees of freedom, which are position in 3D xyz coordinate system and the roll, pitch, and yaw attitude of the UAV (represented by a quaternion). We labeled these local poses of UAVs $\mathbf{L}_{UAV_1}$ and $\mathbf{L}_{UAV_2}$. After finding the approximation of the relative pose $\mathbf{p}_{M_1M_2}$ of the two OGMs, we can estimate not only the 2D relative pose of the two UAVs but even the 3D relative pose $\mathbf{p}_r$ despite having few problems with this process. The problem in finding the 3D relative pose is that the $\mathbf{p}_{M_1M_2}$ is a 2D pose and we have defined only the 2D compounding notation. We simply represent this 2D pose by 3D pose with z coordinate set to zero and the roll as well as pitch set to zero. The 3D relative pose of the received UAV relative to the current UAV can be therefore computed similarly as the 2D relative pose which is computed like this:

$$\mathbf{p}_r = \mathrm{inv}(\mathbf{L}_{UAV_1}) \oplus \mathbf{p}_{M_1M_2} \oplus \mathbf{L}_{UAV_2}. \tag{3.15}$$

We simply translate the received UAV pose in the xy plane and rotate only its yaw orientation as in the case of 2D relative pose and let the z coordinate with roll and pitch unchanged. This approach can be used only if the UAVs height stored in z coordinate is the real height above the surface and not the height relative to the origin of the second UAVs map because the UAVs starting positions can be in different height. The figure 3.3 illustrate all poses defined in this section. We depicted only the 2D xy plane of each pose for illustrating purposes with the orientation of these poses equal to zero. Returning to the equation (3.15) the pose $\mathbf{p}_r$ of UAV_2 is relative to the UAV_1 which is executing the merging algorithm. To correctly transform this pose to be relative to the starting position of the UAV_1 in the frame of reference of the merged map we have to transform this relative pose by the local pose of the UAV_1. We will call this pose the $\mathbf{p}_{U_2G}$ as it is pose of the UAV_2 (U_2) relative to the starting position in global map (G) :

$$\mathbf{p}_{U_2G} = \mathbf{L}_{UAV_1} \oplus \mathbf{p}_r = \mathbf{L}_{UAV_1} \oplus \mathrm{inv}(\mathbf{L}_{UAV_1}) \oplus \mathbf{p}_{M_1M_2} \oplus \mathbf{L}_{UAV_2} = \tag{3.16}$$

$$= \mathbf{p}_{M_1M_2} \oplus \mathbf{L}_{UAV_2},$$

from which we can deduce that:

$$\mathbf{p}_{U_2G} = \mathbf{p}_{M_1M_2} \oplus \mathbf{L}_{UAV_2}. \tag{3.17}$$

3.4 ROS middleware

We choose to integrate the developed map merging algorithm into the Multi-robot Systems group software architecture, which is based on the ROS (Robot Operating System) middleware. The ROS is free, open-source software. It provides a structured communication layer above heterogeneous clusters of sensors, devices, and computing units.

The ROS implementation uses nodes, messages, topics, and services to organize its code in one large structure. Each system (e.g., a UAV system) consists of many nodes. The nodes are processes (e.g., map merging) that perform computation. Each node communicates with other nodes with messages. The nodes can be subscribed to topics publishing messages, and also, a node can publish its messages to topics. The topics are not the only way how the nodes can communicate. They can also use service which uses two types of messages called request and response. We can imagine the topics as a way to "broadcast" the node messages to anyone who wants to subscribe to it. On the other hand, the service communicates on the transaction-based approach of requesting some tasks to be accomplished and receiving the response to the request.

The whole system together forms a graph with computing nodes and edges being the topics and services.

3.5 Integration to MRS group software architecture

The Multi-robot Systems group uses its ROS based software to control and command its robots in many problems with few examples given in the introduction of this thesis. In our case of map merging, we use the MRS group software to extract data from LIDAR and height sensors, which are located on the f450 quadrocopter depicted in figure 1.1. The data from sensors are provided as messages in topics to which other nodes can subscribe.

The HectorSLAM is subscribed to both LIDAR and height sensor. It runs a Gauss-Newton method to optimize scan alignment and produce the occupancy grid map along with the estimate of the 3D pose generated using the Extended Kalman Filter on data from the height sensor and the 2D pose estimate generated in SLAM subsystem. The OGM and 3D pose estimates are each wrapped in an appropriate type of ROS message and continuously published as topics.

Our occupancy grid map merging node located on one of the UAVs subscribes to the topics of OGM and 3D pose of both UAVs. After each epoch of the map merging, the node sends one message with a merged occupancy grid map and one with an estimated pose of the map of the second UAV relative to the map of the merging UAV. After each iteration, our node also publishes messages with sets of occupied cells of both OGMs and a set of the population poses. These messages are used for visualization of the map merging process. The whole used pipeline is shown in figure 3.4.

We also implemented a node for UAVs relative pose approximation on top of the map merging node. This division to two nodes was made so that the map merging process does not interfere with the poses' approximation so that the poses can be estimated with much higher frequency. This second node subscribes to the local poses of UAVs and the map merging node's topic publishing relative pose of maps, as shown in figure 3.5. It also needs to subscribe to the origins of the occupancy grid map in case the maps have different sizes to compensate for the difference.

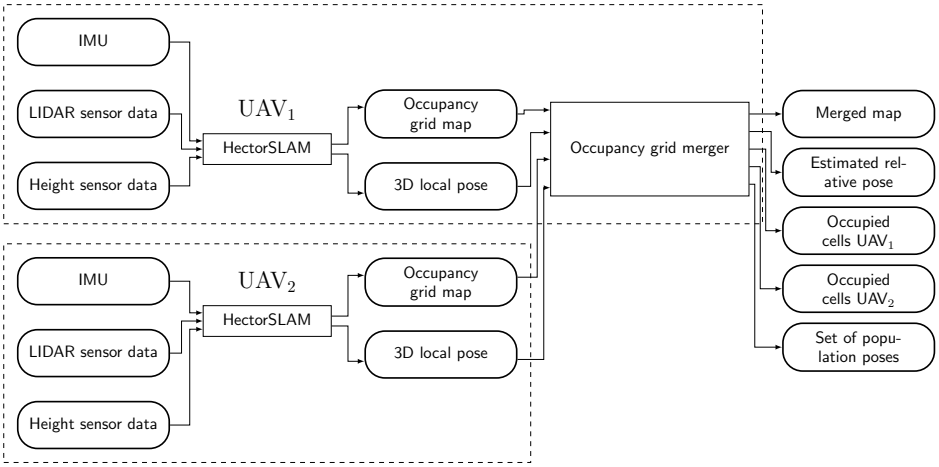


Figure 3.4: The pipeline of the occupancy grid map merging process. The round elements depict the topics. Rectangles depict the ROS nodes.

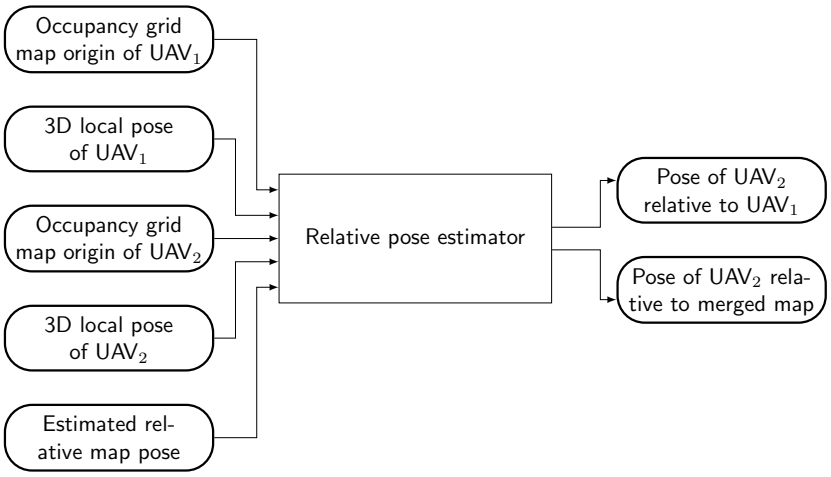


Figure 3.5: The pipeline of the relative pose estimation process. The round elements and rectangles have same meaning as in figure 3.4. Estimator subscribes to relative poses of both UAVs and to the estimation of pose of UAV_2 map relative to the UAV_1 map.

Chapter 4

Experiments

We tested our implementation of the merging method on both data generated by simulation and data from a real-world experiment. The simulation data were created using the *Gazebo* simulator [15]. For the real-world verification, we prepared and executed an experiment in a forest environment. In this chapter, we will present results from experiments generated in the simulation and the results of the experiment in the forest environment. We will show a rate of convergence of the fitness function and discuss the genetic algorithm's overall performance in each experiment. Last but not least, we will also discuss the limitations of our approach.

4.1 Datasets creation

Our map merging method is based on a genetic algorithm that uses a random function in the search process for best map alignment. Because of this randomness, it could, in some lucky few cases, find the alignment after the first iteration and sometimes be clueless for a large number of iterations even while merging the same OGMs as in the lucky case. Therefore, we recorded the experiments to be able to run the merging process on exactly the same data repeatedly. The *Rosbag* package [9] was used to record the data from all sensors of UAVs in real time and then it was used to playback the data on PC (personal computer) as many times as was needed in order to be able to test the robustness of the algorithm.

4.1.1 Simulation dataset creation

We used the Gazebo simulator to create testing environments, with the example depicted in figure 4.1. This map was mainly used to test the algorithm during the implementation process, but in the end, we also used it to test the final version capabilities. After that, we used a simulation model of F450 quad-copter equipped with 2D LIDAR and height sensors to fly through different parts of maps to create OGMs using the HectorSLAM. Each flight

was recorded by Rosbag to be later used for experiments. The real version of the F450 was later used in the real-world experiment. For experiments on the simulation environment showcased in this chapter, we used the maze environment shown in figure 4.1. We choose the initial arrangements of UAVs as follows. One arrangement with the initial positions on the opposite sides of the maze and the second with UAVs starting from the identical spot in the center of the environment.

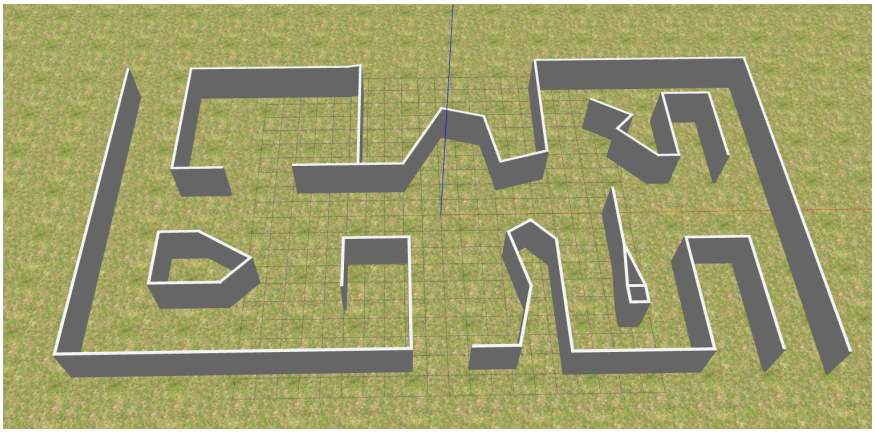


Figure 4.1: Map created in Gazebo simulator. It is approximately 34×17 meters large with walls 3 meters high.

4.1.2 Real-world dataset creation

We took the opportunity to make a real-world dataset on a testing camp organized by the MRS group. We have chosen a forest environment for testing our algorithm. Like the simulation dataset, we made many flights through different parts of the forest and recorded them with the Rosbag. We did in total around 30 flights from which only five flight recordings were usable for the experiments. The other 25 were either too short, the Rosbag did fail to record the flights, or the HectorSLAM did not work properly, and the UAV had to abort the flight. The forest environment is shown in figure 4.2.

4.2 Visualization of merging process

The map merging node (as shown in figure 3.4) also publishes topics purely meant for visualization purposes. These are topics with messages containing the position of occupied cells in both occupancy grids being merged and also set of poses of each generation of the evolution process. We also visualized the relative pose of UAVs published by the relative pose estimator node.

As our method has parameters that can (but do not have to) be tuned to a specific problem before it is used, the visualization topics are useful for

Figure 4.2: F450 equipped with 2D LIDAR and height sensor using HectorSLAM to navigate through a forest.

evaluating made changes. An example of these topics can be seen in figure 4.3.

4.3 Simulation experiments

We will now show two extreme cases of map merging scenarios. As a first experiment, we will show a scenario with two OGMs which are already aligned. The second experiment shows map merging with UAVs initial positions on opposite sides of the maze environment. The size of occupancy grid maps used in these experiments is 2048×2048 cells. Each cell represents a square with the length of the side being 4 centimeters. Therefore the maps are representing an area of size 81.92×81.92 meters.

4.3.1 Identical initial position

As was explained earlier, we created datasets by flying each UAV separately and recording their flights by the Rosbag. This allows us to have two identical initial UAV poses. By already aligned OGMs (map), we mean that the UAVs' initial poses are identical, and the relative pose between the maps depends only on their difference between the sizes of the maps. This difference is included in the relative pose before the evolution process even starts. Therefore, we can set the initial error ranges, which are also used for the reinitialization of individuals as close to zero as we like to.

We chose this approach to be able to test map merging with a relatively small overlap of the two maps. This overlap can be seen in figure 4.3 and with relative pose close to optimum it is roughly 10% of all the occupied cells.

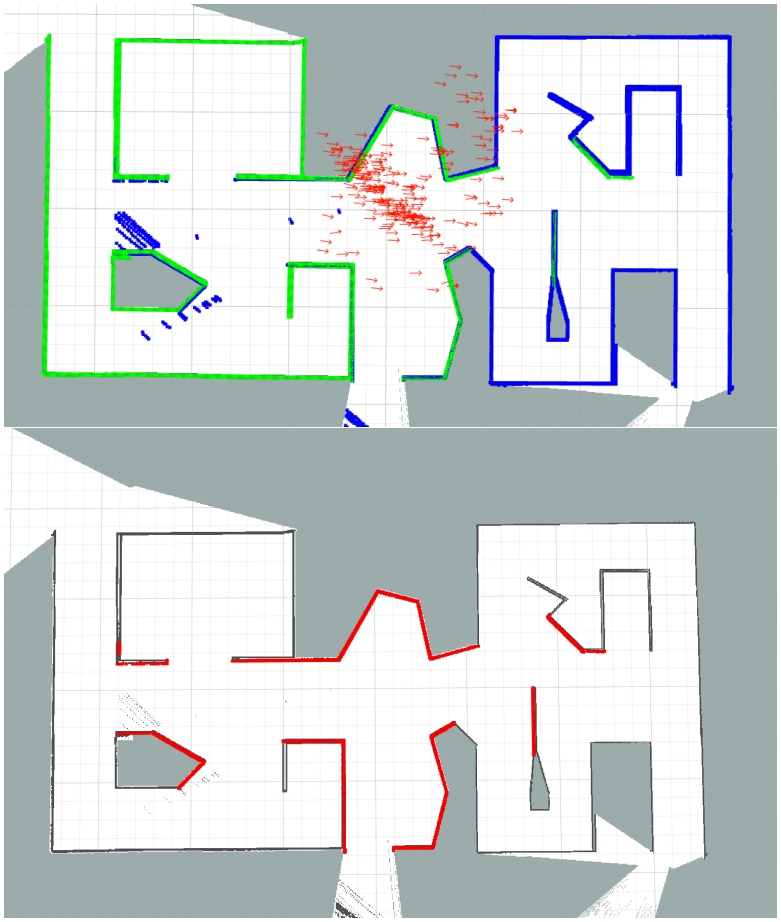


Figure 4.3: **(top)**Visualization of occupied cells (green and blue) of two OGMs (occupancy grid maps) merged together and also a current population of relative poses represented by red arrows pointing in the direction of pose orientation. **(bottom)** Visualization of overlapping occupied cells of both OGMs shown in red on top of the merged OGM.

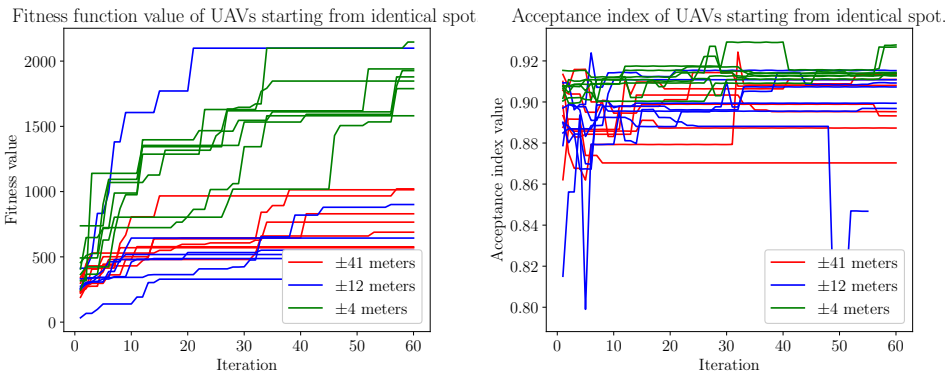


Figure 4.4: Development of fitness function and acceptance index through the process of genetic evolution.

We first started merging with error range set to 10% of the maps side length, which is, in this case, approximately ±4 meters. The algorithm was able to find the relative pose with a small error in units of decimeters in most cases at the end of the first iteration. We, therefore, increased the error range to 30% of maps side length. By increasing the error range from now on, we mean increasing the error of position and orientation explained in 3.2.3, which is used for the initialization of the population and reinitialization of the inferior group. By increasing the error range to a particular percent, we mean that the error range is as big as the percent of the map's side length.

As we found out while merging maps with increased error range, there are other significant local optima as only in one out of seven merging attempts did the algorithm converge to the correct relative pose, and in other six cases, it converged to similar relative pose as in the figure 4.5. We also tried to use the 100% error range (in this case, we have a map with a roughly 82 meters long side. This means an error of ±41 meters). None of the attempts made has lead to the correct solution.

The development of fitness function value, as well as the acceptance index value (described at the end of section 2.2) can be seen in figure 4.4. We can distinguish the successful merging attempts from the unsuccessful as those that converged to the correct solution have fitness value around 2000 after 60 iterations as opposed to the incorrect solutions.

The use of the acceptance index (ai) was a disappointment to us as we could not extract any meaningful information about the state of the map merging from its value. We checked many times and did not find any differences between our definition of the *ai* and the definition used in the source literature. We suspect that the ai value being only in the range between 0.80 and 0.93 could be caused by a high resolution of used maps and by the UAV falsely identifying the ground as an obstacle while taking off. However, we think that it should correlate with fitness value more then it does as both the ai and the fitness function serve as an indicator of correct map merging.

4.3.2 Different initial position

In the second case, we tested merging with two maps of fully explored mazes with UAVs initial position being on the opposite sides of the environment. The results can be seen in figure 4.7. None of the seven attempts shown in the graph could merge the occupancy grid maps even after 60 iterations. The attempt that was the closest to the correct solution is shown in figure 4.8. Merging maps with a larger error range ended with a similar result.

As the UAVs initial poses are roughly 36 meters apart, the used error range has to be at least 44% of the map. We, therefore, set the error of the translation to 45%. The authors of the genetic algorithm method used position error of ±30 meters which is only 6 meters less then in this case. They were able to merge their OGMs within the first few iterations, with the

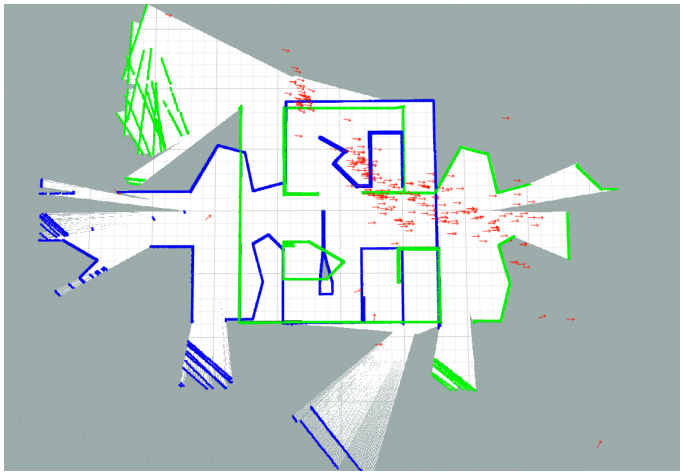


Figure 4.5: An incorrect convergence of map merging process after 60 iterations with initialization of poses population generation ranging the whole map (100% error range, and 360° orientation error).

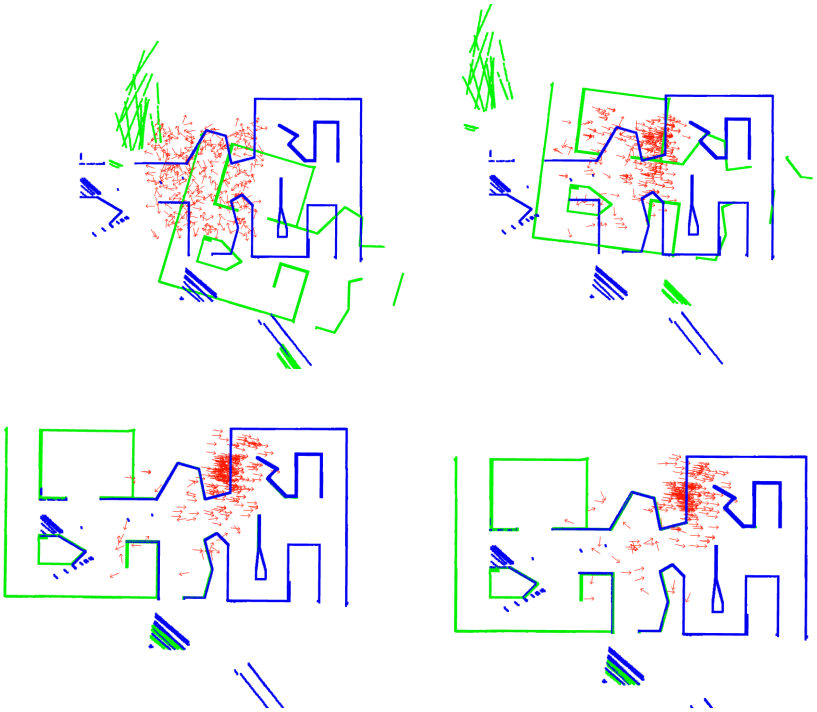


Figure 4.6: Process of map merging which converged to a correct solution to the merging problem. In the upper left state of the merging process are initial poses before first iteration. The upper right state is after 2 iterations. Lower left is state after 8 iterations. Lower right is state after 15 iterations.

majority finding the maps before the seventh iteration.

The inability to merge maps of the maze can be caused by the chosen parameters of evolution, with most significant being the size of the population, the choice of random distributions for individual mutations, or the number

of mutations of each individual per generation iteration. The incorrect map merging can also be caused by the chosen fitness function, which uses information only about the position of occupied cells and not the cells with other states.

The genetic-algorithm-based method of map merging with parameters used in these experiments works robustly and reliably only for smaller error ranges than used in this experiment. Therefore, the UAVs should take off in close proximity.

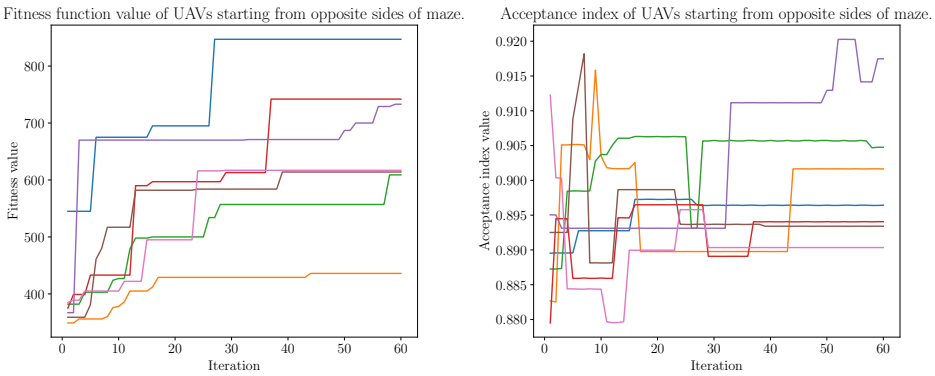


Figure 4.7: Graphs of map merging with initial poses of UAVs on the opposite sides of the maze environment. All attempts used roughly ±37 meters translation error range and ±81 degrees orientation error. None of the attempts to merge maps was successful.

Figure 4.8: The map merging result after 60 iterations with fully explored maze by both UAVs with initial poses on opposite sides of maze.

4.4 Real-world experiment

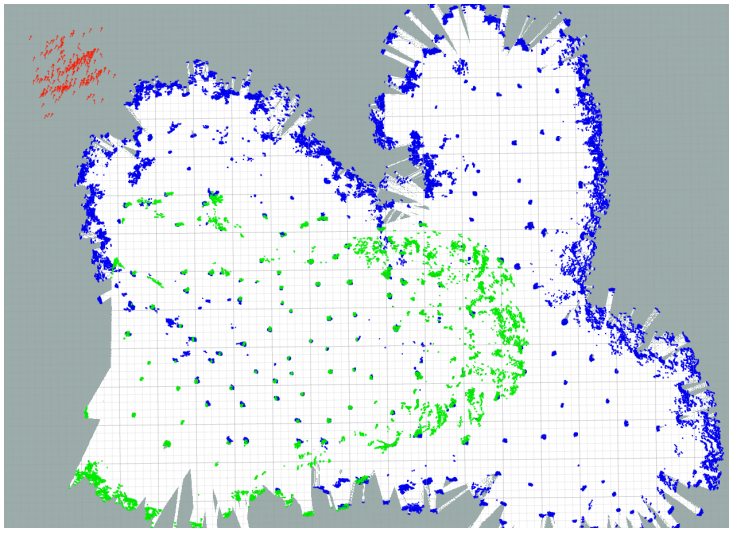


Figure 4.9: Successful merging attempt of two maps from the forest environment. Map of the merging UAV in the blue and received map in green. Note that the relative poses population is not around the initial position of UAV (middle of the map) but in the left upper corner. Different sizes of maps being merged cause this.

The first few flights made in the forest were using OGM with the same size as in the simulation cases, which is 2048×2048 cells with a resolution of a cell being 4 centimeters. During one flight, the UAV flew out of the OGM and crashed. Therefore, we increased the size of the map to 4096×4096 cells with the same resolution (which gives us a map roughly 164×164 meters large) so that we could map a larger area of the forest. Only later, we found out that only a few of the OGMs are overlapping. Between them, the one half had the original size, and the other half was the larger one. To test the map merging on the dataset from the forest, we made the map merging more generalized so that it can use a map of any size and with the initial pose of UAV anywhere in the OGM.

As can be seen in figure 4.9, the maps contain much noise. This is apparent mainly in the middle part where UAV with green OGM did have in a field of view some higher bushes. This was probably caused by a sudden decrease in the altitude of the UAV. We commanded it to maintain a height of 1.5 meters, but as the UAV was flying autonomously only with requests to fly to a given coordinate, the control unit could have some problem, which caused the momentary drop in altitude. We suspected that the noise could be a problem for the algorithm, but as it turns out, it successfully merged these two maps.

The maps used in this experiment have different sizes, as described above. As we know that UAVs in all datasets took off from the same area of roughly

five by five meters with a similar orientation, we can, therefore, set the error range as low as ±5 meters and orientation error as low as 30 degrees. We test our merging algorithm with three different error ranges. These are ranges with the error of 5%, 15%, and 30% of the map side size of the merging UAV. The UAV merging in our case has a map with 4096×4096 cells and a resolution of 4 centimeters. The errors are, therefore, ±8, ±25, and ±50 meters. The orientation error is in all cases set to 30% or ±54 degrees.

We started testing forest map merging with the error range of ±50 meters. As we expected, none of the map merging tests was successful. Next out of the seven tests with error range ±25, three were successful. These three successful cases can be clearly distinguished from the other four cases in figure 4.10. As observers of this merging process, we could see that the merging process converged to the correct solution in these three cases in iteration 29, 41, and 59, respectively. Our observation roughly agrees with the steep growth of fitness function of the best relative pose of the population from value around 1000 to value 3 or 4 times larger within a few iterations.

We further tested the map merging speed by decreasing the error range to ±8 meters. Note that the orientation error is still ±54 degrees, as stated above. With error this low, the algorithm converged to correct local optimum in all seven attempts. The best two attempts did even merge the two maps after 2 and 4 iterations, respectively. The slowest attempt to converge did find the correct relative pose after 29 iterations.

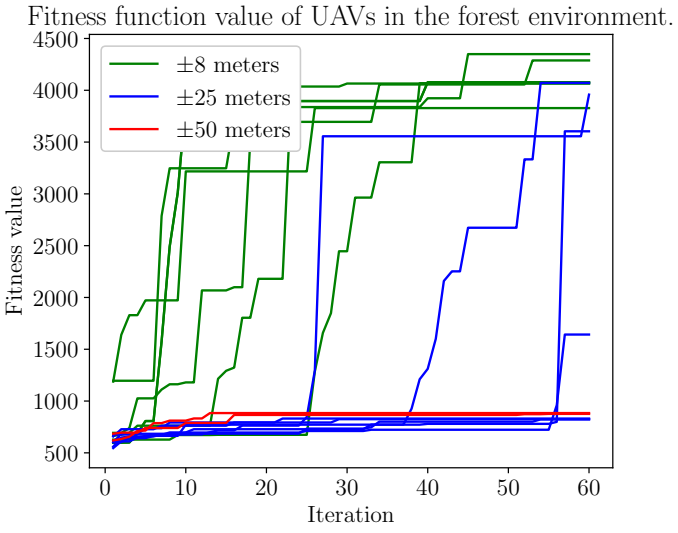


Figure 4.10: Graph of fitness function in forest environment. Notice the steep growth of the function around an iteration where the algorithm converged close to the correct relative pose.

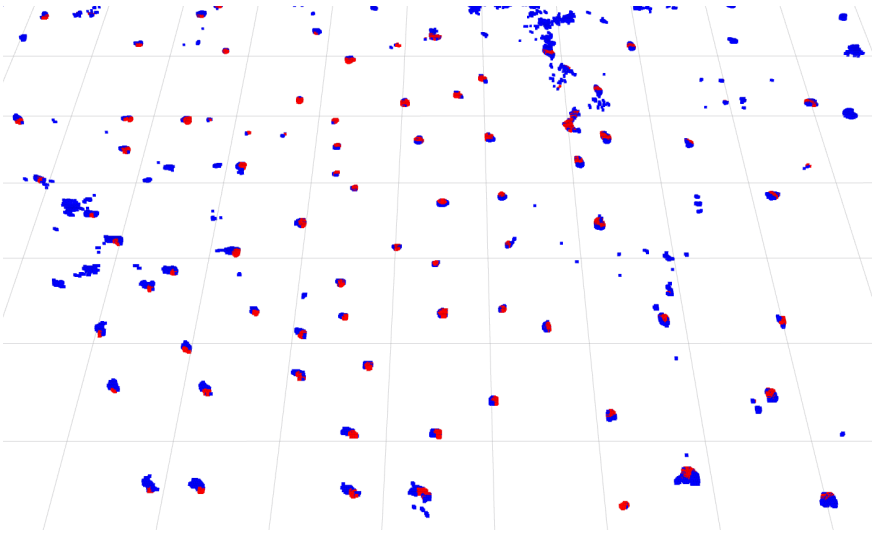


Figure 4.11: Detail of the same merged map as in figure 4.9 but with overlapping occupied cells in red on top of occupied cells in map of merging UAV which is in blue. Note that majority of trees trunks are overlapping.

4.5 Performance of the genetic algorithm

The time complexity of map merging algorithm iteration grows linearly with the number of occupied cells in the received occupancy grid map, as can be seen in figure 4.12. This figure contains data from the experiment with identical initial poses of two UAVs described in section 4.3.1. We proposed at the end of section 3.2.3 that the number of iterations in epoch (or duration of epoch) should be chosen by users of the algorithm to suit their needs. The short duration of the epoch should be chosen if the user needs more frequent updates of the relative pose but does not need it to converge as fast to the correct solution. The longer duration of epoch leads to faster convergence, but the relative pose update frequency is lower. Figure 4.13 shows the dependence of convergence and epoch duration on the chosen number of iterations.

We also tested the convergence of the algorithm which can be seen in figures 4.14 and 4.15. These figures show the error of translation and rotation of the relative pose with the highest value of the fitness function in the given iteration. The data were collected from the experiment with identical initial position described in section 4.3.1.

4.6 Limitations

As discussed in the experiments above, the algorithm is suitable for small error ranges that do not exceed 20 meters. The merging process does not always converge to the correct solution even after 60 iterations with a range higher than this.

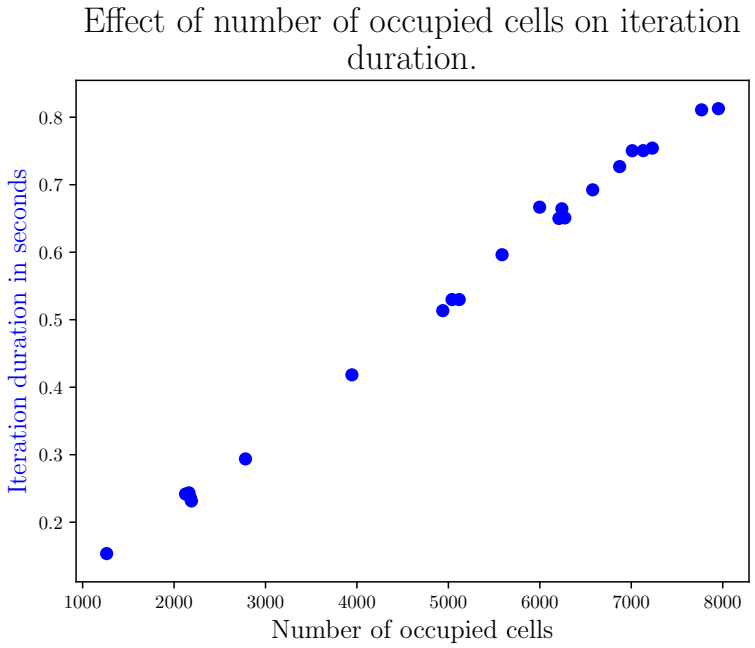


Figure 4.12: Time complexity of evolution iteration with dependence on number of occupied cells in received OGM.

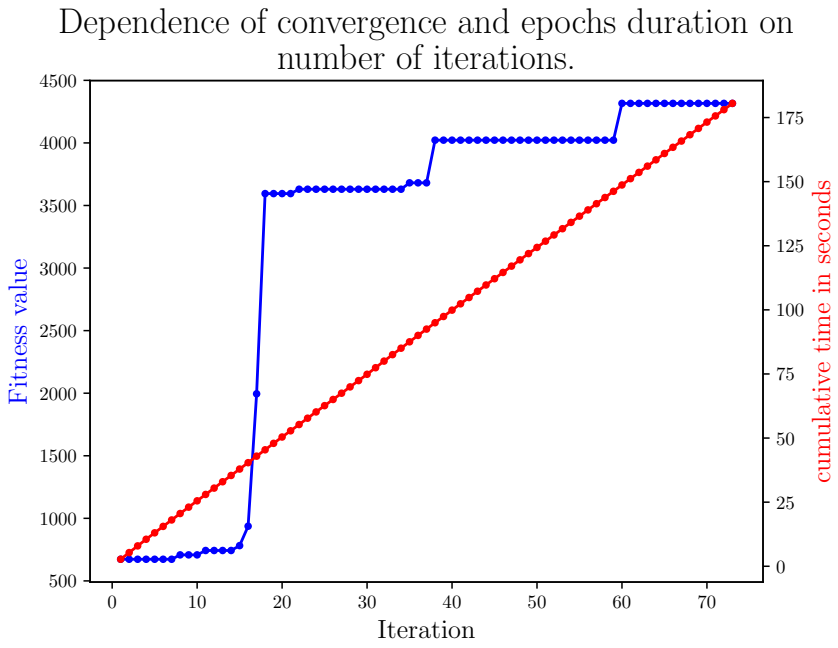


Figure 4.13: Data from simulation experiment on UAVs with identical initial pose. This graph illustrates the 3.2.3 choice of number of iterations of epoch.

The algorithm can also converge to a suboptimal local maximum of the fitness function, as can be seen in figure 4.5.

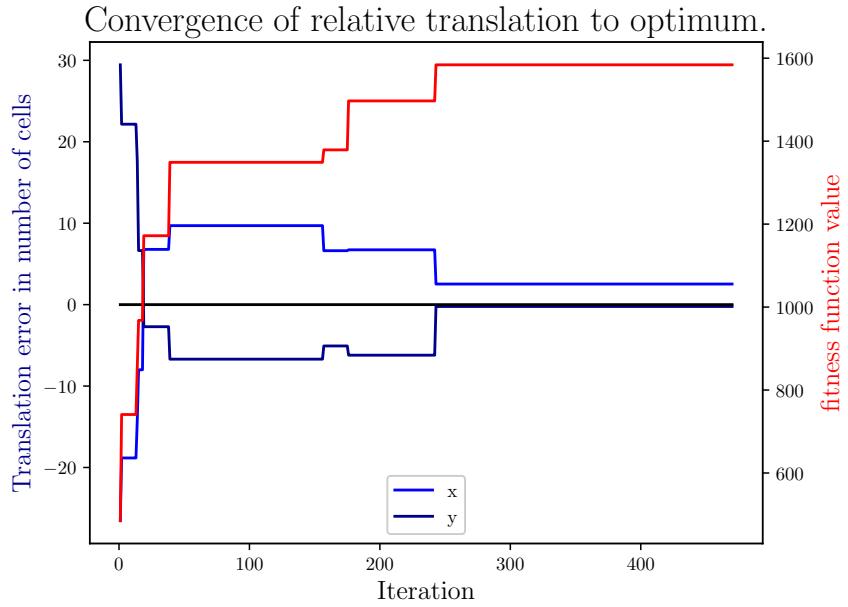


Figure 4.14: Convergence of relative translation on data from the first simulation experiment.

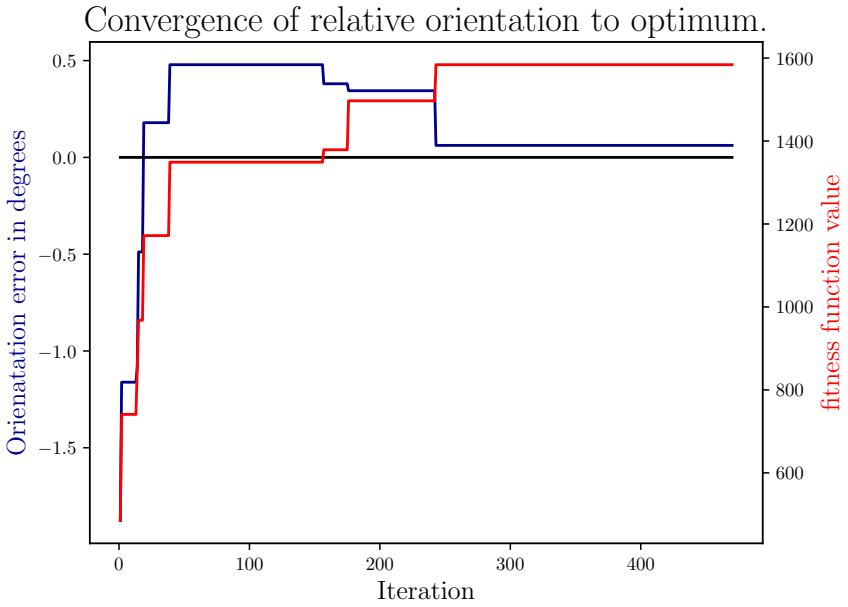


Figure 4.15: Convergence of relative rotation on data from the first simulation experiment.

Chapter 5

Conclusion

In this work, we have done a survey of five available occupancy grid map merging approaches. We have implemented a genetic-algorithm-based map merging methodology and verified it on our datasets created in simulation and real-world forest environments. Only onboard sensors were used to obtain the occupancy grid map of the surrounding and local pose of the UAV on the map. These sensors are a 2D rotating LIDAR and a height sensor. The proposed method does not assume any prior knowledge about the shape of obstacles in the surrounding environment and can be used indoors with many straight walls as well as in outdoors with diversely shaped obstacles. However, the method needs a rough estimate of the distance to the farthest UAV in the group.

We have also implemented an estimator of UAVs relative pose which, based on the accuracy of the map merging, can accurately estimate relative pose with much higher frequency than the map merging algorithm.

It was shown in the experimental chapter that the implemented method could merge maps with a distance of initial poses of two UAVs as long as 25 meters. It was also shown that it could be used in real-world scenarios even with inaccurate data with high noise.

According to the thesis assignment, the following tasks were successfully completed:

- We have become familiar with the HectorSLAM algorithm and the occupancy grid map representation, as shown in the introduction of this thesis.
- The survey of available map merging techniques suitable for building a global map shared by a team of UAVs was performed in chapter 2.
- The genetic-algorithm-based occupancy grid map merging method was implemented, as is described in chapter 3.
- The implemented method was integrated into the MRS group software architecture based on ROS middleware.

- The on-demand merged map output with a 2D estimate of the maps relative pose is provided by the occupancy grid merger ROS node shown in figure 3.4. On top of that, we have implemented a relative pose estimator ROS node, which provides a 3D relative pose estimate of UAVs as well as the 3D pose estimate of UAVs in a global merged map as is shown in figure 3.5. The implemented pairwise method of map merging can be easily used for multiple agents map merging by iteratively merging each received map with the implemented map merging method, as is described in section 1.5.

- We have verified the functionality in the simulation experiments created in the Gazebo simulator.

- We have prepared and executed an outdoor experiment in a forest. The collected data from the outdoor experiment was then used to test the implemented map merging algorithm.

5.1 Future work

During the implementation of our method, we have come across multiple possible improvements that could be incorporated into the merging process. Many of these exciting ideas were mentioned in chapter 2. We will outline some of the possible improvements in this section.

First, we would like to mention that the 2D occupancy grid navigation for flying vehicles is very restrictive as UAVs one of the main advantages compared with ground vehicles is the possibility of moving over an obstacle. By generating only a planar map of the surroundings, the UAV does not fully use its abilities. Therefore the occupancy grid map should be combined with more suiting 3D representation. A point cloud is a good alternative that could be inspected in the future work. We do not recommend fully replacing a 2D representation as it can store the basic information about the layout of obstacles in a large environment with smaller memory requirements, although without the information about the full 3D space.

Some methods are much more efficient in specific cases than the genetic-algorithm-based approach that we used. A process that would select the most suiting method for map merging based on the observation of surrounding should be implemented. This process would then, for example, select a method using Hough transform described in section 2.3 based on the observation that the UAV is in a building with straight walls.

We also had a problem with the increasing complexity of map merging with an increasing number of occupied cells. A downscaling of occupancy grid map into cluster points such as in 2.5 could be incorporated into the map merging process so that the algorithm can be used on merging problems with much larger maps with more occupied cells.

A rendezvous technique should be implemented. It helps the two UAVs to decide when is the best time to merge their maps by finding out when they are close together, and their maps overlap. The method mentioned in the introduction [6] could be a good introduction to this topic.

Lastly, in article [2] authors mention that the optimum of the heuristic function $\Delta(M_1, \mathbf{p} \oplus M_2)$ can be right next to the worst case. This is true, for example, if the map consists of spirals. To compete with this problem, we could use a multi-level map representation as in [17] as described in section 1.4.

Appendix A

Bibliography

[1] Peter Biber and Wolfgang Straßer. The normal distributions transform: A new approach to laser scan matching. volume 3, pages 2743 – 2748 vol.3, 11 2003.

[2] Andreas Birk and Stefano Carpin. Merging occupancy grid maps from multiple robots. *Proceedings of the IEEE*, 94:1384 – 1397, 08 2006.

[3] T. M. Bonanni, B. Della Corte, and G. Grisetti. 3-d map merging on pose graphs. *IEEE Robotics and Automation Letters*, 2(2):1031–1038, 2017.

[4] Stefano Carpin. Fast and accurate map merging for multi-robot systems. *Auton. Robots*, 25:305–316, 10 2008.

[5] Jean-Emmanuel Deschaud. IMLS-SLAM: scan-to-model matching based on 3d data. *CoRR*, abs/1802.08633, 2018.

[6] Pierre Dinnissen, Sidney Givigi, and Howard Schwartz. Map merging of multi-robot slam using reinforcement learning. pages 53–60, 10 2012.

[7] Richard O. Duda and Peter E. Hart. Use of the hough transformation to detect lines and curves in pictures. *Commun. ACM*, 15(1):11–15, January 1972.

[8] A. Elfes. Sonar-based real-world mapping and navigation. *IEEE Journal on Robotics and Automation*, 3(3):249–265, 1987.

[9] Tim Field, Jeremy Leibs, and James Bowman. Rosbag ROS package. `http://wiki.ros.org/rosbag`. Acessed: 2020-08-10.

[10] C. Forster, M. Pizzoli, and D. Scaramuzza. Svo: Fast semi-direct monocular visual odometry. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 15–22, 2014.

[11] W. Hess, D. Kohler, H. Rapp, and D. Andor. Real-time loop closure in 2d lidar slam. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1271–1278, 2016.

[12] Baichuan Huang, Jun Zhao, and Jingbin Liu. A survey of simultaneous localization and mapping with an envision in 6g wireless networks, 2019.

[13] K. Joo, T. Lee, S. Baek, and S. Oh. Generating topological map from occupancy grid-map using virtual door detection. In *IEEE Congress on Evolutionary Computation*, pages 1–6, 2010.

[14] Ue-Hwan Kim, Se-Ho Kim, and Jong-Hwan Kim. Simvodis: Simultaneous visual odometry, object detection, and instance segmentation, 2019.

[15] N. Koenig and A. Howard. Design and use paradigms for gazebo, an open-source multi-robot simulator. In *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566)*, volume 3, pages 2149–2154 vol.3, 2004.

[16] Stefan Kohlbercher and Johannes Meyer. hector_slam ROS package. `http://wiki.ros.org/hector_slam`. Acessed: 2020-07-16.

[17] Stefan Kohlbrecher, Oskar Von Stryk, Johannes Meyer, and Uwe Klingauf. A flexible and scalable SLAM system with full 3D motion estimation. *2011 IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR)*, pages 155–160, 2011.

[18] V. Krátký, P. Petráček, V. Spurný, and M. Saska. Autonomous reflectance transformation imaging by a team of unmanned aerial vehicles. *IEEE Robotics and Automation Letters*, 5(2):2302–2309, 2020.

[19] H. Li and F. Nashashibi. Multi-vehicle cooperative perception and augmented reality for driver assistance: A possibility to 'see' through front vehicle. In *2011 14th International IEEE Conference on Intelligent Transportation Systems (ITSC)*, pages 242–247, 2011.

[20] H. Li, M. Tsukada, F. Nashashibi, and M. Parent. Multivehicle cooperative local mapping: A methodology based on occupancy grid map merging. *IEEE Transactions on Intelligent Transportation Systems*, 15(5):2089–2100, 2014.

[21] Bruce Lucas and Takeo Kanade. An iterative image registration technique with an application to stereo vision (ijcai). volume 81, 04 1981.

[22] A. I. Mourikis and S. I. Roumeliotis. A multi-state constraint kalman filter for vision-aided inertial navigation. In *Proceedings 2007 IEEE International Conference on Robotics and Automation*, pages 3565–3572, 2007.

[23] Matias Nitsche, Tomáš Krajník, Petr Čížek, Marta Mejail, and Tom Duckett. Whycon: An efficent, marker-based localization system. 01 2015.

[24] E. B. Olson. Real-time correlative scan matching. In *2009 IEEE International Conference on Robotics and Automation*, pages 4387–4393, 2009.

[25] Matěj Petrlík, Tomáš Báča, Daniel Hert, Matouš Vrba, Tomáš Krajník, and Martin Saska. A robust UAV system for operations in a constrained environment. *IEEE Robotics and Automation Letters*, PP:1–1, 02 2020.

[26] P. Petráček, V. Krátký, and M. Saska. Dronument: System for reliable deployment of micro aerial vehicles in dark areas of large historical monuments. *IEEE Robotics and Automation Letters*, 5(2):2078–2085, 2020.

[27] Morgan Quigley, Ken Conley, Brian Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, and Andrew Ng. Ros: an open-source robot operating system. volume 3, 01 2009.

[28] Colin Reeves. *Genetic Algorithms*, volume 146, pages 109–139. 09 2010.

[29] Robert Richardson, Raul Fuentes, Tim Chapman, Michael Cook, James Scanlan, Zhibin Li, and David Flynn. *Robotic and Autonomous Systems for Resilient Infrastructure*. UK-RAS White Papers. UK-RAS Network, 06 2017.

[30] William Rone and Pinhas Ben-Tzvi. Mapping, localization and motion planning in mobile multi-robotic systems. *Robotica*, 31(1):1–23, 2013.

[31] S. Saeedi, L. Paull, M. Trentini, M. Seto, and H. Li. Efficient map merging using a probabilistic generalized voronoi diagram. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4419–4424, 2012.

[32] Sajad Saeedi, Liam Paull, Michael Trentini, and Howard Li. Neural network-based multiple robot simultaneous localization and mapping. *IEEE transactions on neural networks / a publication of the IEEE Neural Networks Council*, 22:2376–87, 12 2011.

[33] Sajad Saeedi, Michael Trentini, Mae Seto, and Howard Li. Multiple-robot simultaneous localization and mapping: A review. *Journal of Field Robotics*, 33(1):3–46, 2016.

[34] Chuck Thorpe and Hugh F. Durrant-Whyte. Field robots. In *ISRR*, pages 329–340, 2001.

[35] Sebastian Thrun and Arno Bü. Integrating grid-based and topological maps for mobile robot navigation. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence*, volume 2 of *AAAI'96*, page 944–950. AAAI Press, 1996.

[36] M. Vrba, D. Heřt, and M. Saska. Onboard marker-less detection and localization of non-cooperating drones for their safe interception by an autonomous aerial system. *IEEE Robotics and Automation Letters*, 4(4):3402–3409, 2019.

[37] V. Walter, N. Staub, A. Franchi, and M. Saska. Uvdar system for visual relative localization with application to leader–follower formations of multirotor uavs. *IEEE Robotics and Automation Letters*, 4(3):2637–2644, 2019.

[38] Chieh-Chih Wang. *Simultaneous Localization, Mapping and Moving Object Tracking*. PhD thesis, Carnegie Mellon University, Pittsburgh, PA, 04 2004.

[39] Ji Zhang and Sanjiv Singh. Loam: Lidar odometry and mapping in real-time. 07 2014.

Appendix B

CD Content

In Table B.1 are listed names of the root directories and files contained on the enclosed CD. The CD contains ROS nodes source code of map merging algorithm and relative pose approximation designed for multiple UAVs, the text of this thesis in a PDF format, and the latex source code of the thesis.

Directory	Description
/occupancy_grid_merger	ROS node for the occupancy grid map merging.
/relative_pose_estimator	ROS node for the on demand approximation of relative pose.
/experimetns	Experiments source code without the rosbag files.
/README.md	Instructions on how to use the ROS nodes.
/thesis.pdf	Text of this thesis in PDF format.
/thesis_source	Latex source code of this thesis.

Table B.1: CD content structure