Bachelor Thesis

Czech
Technical
University
in Prague

F3

Faculty of Electrical Engineering
Department of Cybernetics

# Spanning Tree Coverage Algorithm on Large Spaces for Multi-UAV Systems

Jan Chleboun

Supervisor: Tiago Pereira do Nascimento
Field of study: Cybernetics and Robotics
May 2022

## Acknowledgements

I would like to thank my supervisor, Prof. Tiago Pereira do Nascimento, Ph.D., for regular consultations and all the great advice. I would also like to express my gratitude for help with hardware experiments to Ing. Daniel Heřt, Ing. Jiří Horyna, M.Sc. Parakh Manoj Gupta, and M.Sc. Akash Chaudhary. My thanks also go to the whole Multi-Robot Systems Group for making the thesis possible and for providing me the hardware necessary to realize the real world experiments. I am also very grateful for the unlimited support and advice from my friends (especially Daniel Kubišta and Jiří Jirák) and my family.

## Declaration

## Abstract

In this work, the Improved Artificially Weighted Spanning Tree Coverage (IAWSTC) and the Cycle Growing With Event Partitioning (CGWEP) algorithms are proposed. These novel approaches are suitable for the exploration of environments with cluttered regions, where unexpected obstacles can appear. In order to solve these problems, a smoothing algorithm together with an online replanner to avoid unexpected detected obstacles is proposed. Experiments are carried out in simulation and on real drones. The algorithms are then compared and conclusions are deduced.

**Keywords:** exploration, spanning tree, UAV, trajectory planning, trajectory smoothing

**Supervisor:** Tiago Pereira do Nascimento

## Abstrakt

V této práci jsou navrženy algoritmy Improved Artificially Weighted Spanning Tree Coverage (IAWSTC) a Cycle Growing With Event Partitioning (CGWEP). Tyto nové přístupy jsou vhodné pro prohledávání prostorů se známými překážkami, kde se mohou objevit také nečekané překážky. Pro vyřešení tohoto problému je také navržen algoritmus pro vyhlazování trajektorií a algoritmus přeplánování při detekci neočekávaných překážek. Byly provedeny simulační experimenty spolu s experimenty na opravdových dronech. Algoritmy jsou porovnány a zhodnoceny.

**Klíčová slova:** průzkum, dron, plánování trajektorií, vyhlazování trajektorií

**Překlad názvu:** Algoritmus pro průzkum velkých prostorů pomocí skupiny kooperujících dronů

# Contents

# Figures

## Tables

# Chapter 1

# Introduction

In recent years Unmanned Aerial Vehicles (UAVs) also known as *drones* are gaining popularity due to their ability to perform various complex tasks. Such tasks can be, for example, transport of small objects [10], historical building documentation [24], surveillance [9], asset inspection [7], and so on. A typical UAV is a small lightweight helicopter with four rotors, such as the one in Figure 1.1. The biggest strength of an UAV is its agility, size, and price, whereas the biggest weakness is short battery life (approximately 20 minutes). In most applications, groups consisting of multiple UAVs are used, which is possible due to the affordability of such robots.

**Figure 1.1:** UAVs used by the MRS group [1]

For every application where the UAVs move, a path planning algorithm is needed that generates the individual trajectories. Such an algorithm is

[1]Source: `http://mrs.felk.cvut.cz/projects/gacr-swarm-ii`

the main focus of this work. This work focuses on a decentralised version of the algorithm, which ensures equal distribution of computational load and no dependency on a central unit. The algorithm consists of two parts: first, the explored environment is split into subregions corresponding to individual drones, and second, a trajectory for each drone is created such that its corresponding subregion is covered. For the environment partitioning, the approach from [8] was used as a starting point and modified for better performance.

The trajectories can be generated in 3D space or in 2D space, meaning that the UAV's height remains fixed. This thesis focusses on the latter approach, as it can be advantageous in large areas such as meadows or deserts because neglecting the third dimension significantly saves computational resources. This approach is suitable for tasks such as object localisation or search and rescue missions, where the terrain is scanned from a fixed height. A widely used path planning algorithm is the Spanning Tree Coverage (STC) algorithm [12], [8]. This popular algorithm works really well when certain conditions are met, however, there are some disadvantages of this approach which are overlooked quite often. In this work, the algorithm was implemented, an alternative approach was designed, and the two algorithms were compared.

When initial suitable trajectories are found, it is common that they contain rough turns, which makes it hard for the UAV to track the trajectory accurately. For this reason, a smoothing algorithm is used. There are plenty trajectory smoothing algorithms already, but in this work a new method is proposed based on the least squares solution of a system of linear equations, whose biggest advantage is its simplicity.

## 1.1 Objective

The main objective of this thesis is to develop an algorithm for the exploration of large environments with known obstacle maps. This thesis aims to:

1. Implement the Artificially Weighted Spanning Tree Coverage (AWSTC) algorithm from [8]
2. Propose an improved version of the AWSTC algorithm with less redundancy of coverage
3. Design an alternative algorithm for environment exploration and compare it to the AWSTC algorithm

4. Propose a new trajectory smoothing method
5. Implement real time detection of unexpected obstacles and replanning
6. Perform simulation and real world experiments

## 1.2 Contributions

The contributions of this thesis are:

1. The Improved Artificially Weighted Spanning Tree Coverage (IAWSTC) algorithm, which is the improved version of the AWSTC algorithm from [8], with less redundancy and with the ability to handle unexpected obstacles
2. The Cycle Growing With Environment Partitioning (CGWEP) algorithm, which can be used as an alternative to the IAWSTC or AWSTC algorithms
3. The Trajectory Smoothing Using Least Squares Solution (TSULSS) approach, which can be used for fast smoothing of long trajectories

# Chapter 2

# Related Works

There are many works that focus on area coverage using multiple UAVs. In [8] the environment is divided into cells as follows:

$$C_u = \{c_i |\ c_i \in C, c_i \cap A = \varnothing, c_i \cup O = \varnothing\},$$

where $A$ is the area with obstacles $O$ sampled with equal cells $C$. The coverage problem is then treated as an optimisation problem

$$\begin{aligned} &\min \max \|P_i\|, \\ &\text{s.t.} \bigcup_{i=1}^{N} P_i = C_u, \end{aligned}$$

where $P_i$ is the spanning tree constructed for the ith agent and N is the number of agents. The AWSTC algorithm is then proposed to find a suitable solution. All cells are iteratively assigned score by each agent, and the agents take turns and pick cells. After this, the individual spanning trees are reorganized and converted into trajectories, which are then smoothed using Bézier curves. In this thesis, their work was implemented, modified in order to reduce the redundancy of the resulting trajectories, and an alternative algorithm was proposed.

Another approach is presented in [23], where the task is solved using *game theory* optimizing energetic efficiency of communication between agents. The problem is decomposed into two subproblems: coverage maximization and power control. These subproblems are then solved using a spatial adaptive play-based algorithm. Unlike the distributed algorithms considered in this thesis, the coverage algorithm uses a command centre, which means that it is centralised.

In [15], the environment is represented as a polygon, then decomposed into multiple subpolygons where a simple trajectory with a minimal number of turns can be easily generated and, finally, the individual trajectories are connected. Only one UAV is used in [15], but the same approach could be applied for multiple UAVs. In [15], neither static nor dynamic obstacles are considered. Another approach to exploring polygonal environments, which generates very similar trajectories, is the Geometric Vector Algorithm proposed in [16].

A similar approach to the one used in this thesis for constructing spanning trees for the coverage problem can be found in [1], however, there is no further mention of the trajectory generation from the spanning trees found this way.

In [17] two approaches with similar results are proposed - The Coverage with Route Clustering algorithm represents the free area as a graph using *boustrophedon cellular decomposition algorithm* and then formulates the path finding problem as *MinMax k-Chinese postman problem*. The other proposed algorithm, named Coverage with Area Clustering decomposes the problem into coverage problems of multiple similar subareas.

A more practical point of view is adopted in [2], where real-world factors such as maximum flight time, setup time, and battery state of UAVs are taken into account. The coverage problem is solved in two steps. A graph, where the vertices represent geographic coordinates, is constructed in a way that a single UAV would cover the area optimally in terms of cover time. After this a mixed integer linear programming problem is solved.

In [20] the coverage problem is solved to perform post-earthquake mapping. The operation bases are also taken into account, which means that the algorithm must also decide which bases to open and assign each drone an operation base. The problem is described by a minimax objective function.

In [9] the task of finding objects of interest in a specified area is tackled. This surveillance problem is treated as a multivehicle variant of the Dubins travelling salesman problem with neighbours and solved with unsupervised learning. This approach is computationally inexpensive and is further improved by reparameterization with the use of Bézier curves.

# Chapter 3

# Problem Formulation

The main goal of this chapter is to describe the problem in depth and to lay out the necessary terminology which is needed for accurate descriptions of the algorithms. Please note that some of the definitions arised solely for the purposes of this work and should not be mistaken for official terminology in use.

## 3.1 Definitions

**Definition 3.1.** **Agent** is an instance of a robot (UAV) taking part in the task.

**Definition 3.2.** **Cell** is a square cutout of 2D space. It can be divided into four **subcells** with equal dimensions. The **coordinates** of a *cell* are the coordinates of its center. A *cell* is considered **explored** after the *agent* passes through its center or through the centres of all its *subcells*. A *cell* is **unexplored** if it is not *explored*.

**Definition 3.3.** **Region** is a continuous (meaning no *cell* is isolated) set of *cells*. Its subsets are **subregions**.

**Definition 3.4.** **Environment** is the area where the task is performed. The goal is to explore the environment fully.

**Definition 3.5.** **Environment representation** is a *region* which fully describes the *environment*.

**Definition 3.6.** **Obstacle** is a representation of an object in physical space that prevents the UAV from entering a certain area. An *agent* can never visit a *cell* which contains an *obstacle*. A *cell* with an *obstacle* is considered *explored* at the moment the agent obtains the first information about the *obstacle*.

**Definition 3.7. Trajectory** is a sequence of points in space. When the drone **tracks** a trajectory, it flies through each point the *trajectory* consists of.

**Definition 3.8. Euclidean distance** of *cells* $C_1, C_2$ is a metric defined as:

$$d_{C_1,C_2} = \sqrt{(x_{C_1} - x_{C_2})^2 + (y_{C_1} - y_{C_2})^2}, \tag{3.1}$$

where $x_{C_1}, y_{C_1}$ are the coordinates of the first *cell* and $x_{C_2}, y_{C_2}$ are the coordinates of the second *cell*.

**Definition 3.9. Manhattan distance** of *cells* $C_1, C_2$ is a metric defined as:

$$D_{C_1,C_2} = |x_{C_1} - x_{C_2}| + |y_{C_1} - y_{C_2}|, \tag{3.2}$$

where $x_{C_1}, y_{C_1}$ are the coordinates of the first *cell* and $x_{C_2}, y_{C_2}$ are the coordinates of the second *cell*.

**Definition 3.10.** Cells $C_1, C_2$ are **neighbour cells** if

$$D_{C_1,C_2} = \mathrm{CS}, \tag{3.3}$$

where $D_{C_1,C_2}$ is the *Manhattan distance* of cells $C_1, C_2$ and CS is the length of a side of a *cell*.

**Definition 3.11.** Cells $C_1, C_2$ are **diagonal neigbour cells** if

$$d_{C_1,C_2} \leq \mathrm{CD}, \tag{3.4}$$

where $d_{C_1,C_2}$ is the *Euclidean distance* of cells $C_1, C_2$ and CD is the length of a diagonal of a *cell*. Each *neighbour cell* is also a *diagonal neighbour cell*, but the opposite does not apply.

**Definition 3.12. Exploration status** of a cell is the information whether the *cell* is *explored* or *unexplored* and whether it contains an *obstacle*.

**Definition 3.13. Merging** is a process in which the *environment representation* is divided into blocks and then each block is replaced by a single cell which shares the *exploration status* with the original block. Each block consists of four cells with the same *exploration status*, where each pair of cells are *neighbour cells*. *Merging* can be performed only on certain *environments*.

**Definition 3.14. Surrounding cycle** is an oriented graph consisting of *cells* and edges. An edge can connect *cell* $C_1$ and *cell* $C_2$ only if $C_1$ and $C_2$ are *neighbour cells*. The most important property of the *surrounding cycle* of a region $S_i$ is that no *unexplored* cell $C_j \in S_i$ can be outside of the *surrounding cycle*.

**Definition 3.15. Left hand rule** is a way of navigation inside an *environment representation* where we consider a virtual agent $A_v$ which is represented by its position in the *environment representation* and by its *heading* $h_{A_v} \in \{\mathrm{left}, \mathrm{right}, \mathrm{up}, \mathrm{down}\}$. The *left hand rule* is used when the $A_v$ decides what action to perform. The priority of actions depends on the current *heading* of $A_v$. The action with highest priority among all feasible actions is chosen. We can see the action priority table for the *left hand rule* in Table 3.1.

**Table 3.1:** Left hand rule action priority table

| Heading of $A_v$ | Action with priority 4 | Action with priority 3 | Action with priority 2 | Action with priority 1 |
|---|---|---|---|---|
| right | up | right | down | left |
| down | right | down | left | up |
| left | down | left | up | right |
| up | left | up | right | down |

**Definition 3.16. Redundancy ratio** describes the redundancy of coverage. It is calculated as follows:

$$\zeta_r = \frac{\sum\limits_{i=1}^{N} \operatorname{card}(S_i)}{\operatorname{card}(E_{free})}, \tag{3.5}$$

where N is the number of agents, $\operatorname{card}(S_i)$ is the number of cells in the $i$-th region $S_i$ and $\operatorname{card}(E_{free})$ is the number of cells from the *environment representation* which do not contain an obstacle.

**Definition 3.17. Equality ratio** describes how much the sizes of subregions corresponding to the individual agents differ. It is calculated as follows:

$$\zeta_e = \frac{\max\limits_{i\in\{1,\dots,N\}} \operatorname{card}(S_i)}{\frac{1}{N}\operatorname{card}(E_{free})}, \tag{3.6}$$

where N is the number of agents, $\operatorname{card}(S_i)$ is the number of cells in i-th region $S_i$ and $\operatorname{card}(E_{free})$ is the number of cells from the *environment representation* that do not contain an obstacle.

**Definition 3.18. Length ratio** describes the redundancy of a trajectory. It is calculated as follows:

$$\zeta_e = \frac{\operatorname{length}(\tau)}{\operatorname{card}(S_i)\, CS}, \tag{3.7}$$

where $\operatorname{length}(\tau)$ is the length of trajectory $\tau$, CS is the cell size and $\operatorname{card}(S_i)$ is the number of cells in i-th region $S_i$.

**Definition 3.19. Curvature ratio** describes the curvature of a trajectory. It is calculated as follows:

$$\zeta_c = \frac{\sum\limits_{i=1}^{M-2} \left|\arccos\left(\frac{v_i \cdot v_{i+1}}{|v_i|\,|v_{i+1}|}\right)\right|}{\operatorname{length}(\tau)}, \tag{3.8}$$

where length$(\tau)$ is the length of the trajectory $\tau$, M is the number of points $P_1, \ldots, P_M \in \tau$ and $v_i, v_{i+1}$ are calculated as follows:

$$v_i = P_{i+1} - P_i, \tag{3.9}$$
$$v_{i+1} = P_{i+2} - P_{i+1}. \tag{3.10}$$

To extend this definition for closed trajectories, we append points $P_{M+1} = P_1$ and $P_{M+2} = P_2$ to the end of the trajectory, but we exclude those auxiliary points from the length calculation.

**Definition 3.20. Length increase ratio** describes how much the length of a trajectory increases due to trajectory smoothing. It is calculated as follows:

$$\zeta_i = \frac{\operatorname{length}(\tau_s)}{\operatorname{length}(\tau)}, \tag{3.11}$$

where length$(\tau)$ is the length of the original trajectory $\tau$ and length$(\tau_s)$ is the length of the smoothed trajectory $\tau_s$.

**Definition 3.21. Turn intensity** describes the smoothness of turns in a trajectory. The greater the turn intensity, the sharper the turns are. It is calculated as follows:

$$\zeta_t = \frac{\sqrt{\sum_{i=1}^{M-2} \left(\arccos\left(\frac{v_i \cdot v_{i+1}}{|v_i|\,|v_{i+1}|}\right)\right)^2}}{M-2}, \tag{3.12}$$

where M is the number of points $P_1, \ldots, P_M \in \tau$ and $v_i, v_{i+1}$ are calculated as follows:

$$v_i = P_{i+1} - P_i \tag{3.13}$$
$$v_{i+1} = P_{i+2} - P_{i+1} \tag{3.14}$$

To extend this definition for closed trajectories, we append points $P_{M+1} = P_1$ and $P_{M+2} = P_2$ to the end of the trajectory.

## 3.2 Environment Representation

For the purpose of this work, the *environment representation* from [8] was adopted. The height is fixed, meaning that in 3D the environment is a plane with a constant vertical coordinate. This plane is divided into *cells*.

*Path planning* in this *environment representation* consists of two steps.

- Finding *regions* $S_i$, where $i \in \{1, 2, \ldots, N\}$ and $N$ is the number of *agents*, such that

  $$\bigcup_{i=1}^{N} S_i = E_{free},$$

  where $E_{free}$ is a subset of the *environment representation* consisting of *cells* $C_j, j \in \{1, 2, \ldots, M\}$, where $M$ is the number of *cells* that do not contain an *obstacle*.

- Generating a *trajectory* for each subregion $S_i$ such that all the *cells* in $S_i$ are *explored* and the last *cell* of the trajectory is equal to the first *cell* of the *trajectory*. The latter condition is suitable for applications such as surveillance, where the *trajectory* is tracked multiple times consecutively.

# Chapter 4

# Exploration

The main goal of the exploration process is to divide the *environment representation* to *subregions* corresponding to individual *agents* and generate *trajectories* such that each *subregion* is fully covered. Preferably, the differences of lengths of the resulting trajectories should be minimal, which ensures efficient coverage.

## 4.1 Environment Partitioning

In the environment partitioning part of the AWSTC algorithm from [8] the *agents* take turns, iteratively evaluate all suitable *cells*, and then pick the one with the highest calculated score and mark it as *explored.* This process ends when all *cells* in the area of interest are marked as *explored.* The original evaluation formula (4.1) taken over from [8] consists of three terms $E^a_j, E^b_{j,k}, E^c_j$. Here $E_{i,j}$ is the score of the $j$th cell evaluated by the $i$th robot and N is the number of robots used for the task:

$$E_{i,j} = E^a_j + \sum_{k=1,k\neq i}^{N} E^b_{j,k} + E^c_j. \tag{4.1}$$

The first term $E^a_j$ describes the distance of the evaluated cell from the centre of inertia of the unexplored area. This ensures that cells on the border of the unexplored area have higher scores. It is calculated as follows:

$$E^a_j = \sigma_a \left(|x_i - x_{ce}| + |y_i - y_{ce}|\right), \tag{4.2}$$

where $\sigma_a > 0$ is a constant, $x_i, y_i$ are the coordinates of the evaluated cell and $x_{ce}, y_{ce}$ are the coordinates of the centre of inertia of the environment.

The second term $E^b_{j,k}$ describes the distance of the $j$th cell from the region $S_k$ belonging to the kth agent. This term increases the score of cells that are far away from the other UAVs. It is calculated as follows [1]:

$$E^b_{j,k} = \sigma_b \min_{[x_i,y_i]\in S_k} \left(|x_i - x_j| + |y_i - y_j|\right), \tag{4.3}$$

where $\sigma_b > 0$ is a constant, $x_j, y_j$ are the coordinates of the evaluated cell and $x_i, y_i$ are the coordinates of the cells belonging to the region $S_k$ belonging to the $k$th agent.

The third term $E^c_j$ describes the exploration status (*explored* / *unexplored*) of the $j$th cell. It decreases the score of already explored cells. It is calculated as follows:

$$E^c_j = \begin{cases} 0 & \text{for } \textit{unexplored} \text{ cells} \\ -\mathcal{E}_M & \text{for } \textit{explored} \text{ cells,} \end{cases} \tag{4.4}$$

where $\mathcal{E}_M > 0$ is a large constant.

The Environment partitioning algorithm is described again in Algorithm 1.

### 4.1.1 Proposed Approach

In order to improve the behaviour of the AWSTC algorithm, the formula (4.1) used for evaluation of the feasible cells was modified to:

$$E_{i,j} = E^a_j + \sum_{k=1,k\neq i}^{N} E^b_{j,k} + E^c_{j,m} + E^d_{i,j}. \tag{4.5}$$

[1] In [8] this term is defined vaguely using a term $d_{i,k}$ that the authors define as '$d_{i,k}$ is the Euclidean distance between the $i$th cell and the $k$th agent', but the formula they use corresponds to the Manhattan distance, and it is also not clear what they meant by 'distance between cell and agent'. Therefore, we assumed it is the Manhattan distance between the evaluated cell and $k$th agent's region and that they accidentally used $d^2$ instead of $d$ in the formulas, but our interpretation might not be what the authors originally meant. However, the behaviour of the algorithm with such corrections matched the behaviour documented in the article much more closely than the other interpretations that we tried.

**Algorithm 1** The environment partitioning algorithm

```
Divide the environment into cells $C_j$
for each cell $C_j$, $j \in [1, 2, \dots, M]$ do
    $C_j \leftarrow$ unexplored
end for
Initialize subregions $S_i$
for each subregion $S_i$, $i \in [1, N]$ do
    $S_i \leftarrow$ $i$th UAV's starting point $\mathrm{SP}_i$
    $\mathrm{SP}_i \leftarrow$ explored
end for
while unexplored, $C_i$ exists do
    for each $\mathrm{UAV}_i$, $i \in [1, N]$ do
        Evaluate each cell with one or more neighbour cells belonging to the region $S_i$ with formula (4.1)
        find $C_{best} = \arg_j \max E_{i,j}$
        Add $C_{best}$ to $S_i$
        $C_{best} \leftarrow$ explored
    end for
end while
```

The term $E^c_{j,m}$ is a modification of the term $E^c_j$ from the original formula defined as follows:

$$E^c_{j,m} = \begin{cases} 0 & \text{for } \textit{unexplored} \text{ cells} \\ -\mathcal{E}_M \, D_{j,u} & \text{for } \textit{explored} \text{ cells,} \end{cases} \tag{4.6}$$

where $\mathcal{E}_M > 0$ is a large constant and $D_{j,u}$ is the *Manhattan distance* of the *cell* $C_j$ from the closest *unexplored cell*. To save computational resources, only the *unexplored* cells, where $E^c_{j,m} = 0$ are evaluated at first. The *explored* cells are evaluated only when there is no *unexplored cell* among the evaluated cells, which means that $D_{j,u}$ needs to be calculated infrequently.

The term $E^d_{i,j}$ describes how many *diagonal neighbour cells* of the *cell j* are from the same region. The purpose of this term is to increase the score of cells that form clusters, which is convenient for the Cycle Growing (CG) algorithm (Section 4.2.2) used in the trajectory generation process (Section 4.2). It is defined as follows:

$$E^d_{i,j} = \sigma_d \, \mathrm{NC}_{i,j}, \tag{4.7}$$

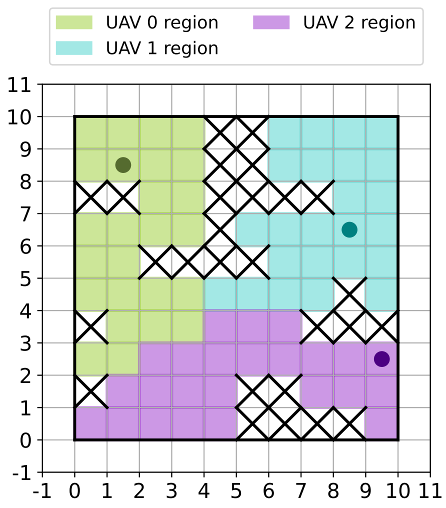
where $\sigma_d > 0$ is a constant and $\mathrm{NC}_{i,j}$ is the number of diagonal neighbour cells of cell $C_j$ that belong to the region $S_i$ corresponding to the $i$th agent.

Another improvement is the Algorithm 2, which is used to minimise the intersections of individual *regions* while maintaining similar *cell* counts. During this process, every *cell* $C_j$ that belongs to more than one *region* is removed from every *region* $S_i$ where the absence of $C_j$ does not violate the continuity of $S_i$. If $C_j$ does not belong to any *region* afterwards, it is marked as *unexplored*. After this, the *agents* take turns, evaluate each *unexplored cell* with 4.5 and pick the *cell* with the highest score. On each turn, the *agent* that gets to pick a *cell* is the *agent* whose corresponding *region* has the smallest number of *cells*. This whole process can be repeated until no solvable conflicts emerge, but in my implementation three iterations are performed, as it is enough to solve the majority of conflicts and it does not increase the computational demand much.

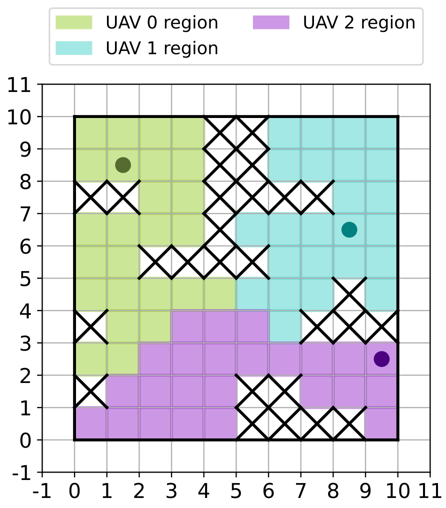
**Algorithm 2** Algorithm for region intersection reduction

```
for each cell $C_j$, $j \in [1, M]$ do
    if $\mathrm{NR}_{C_j} > 1$, where $\mathrm{NR}_{C_j}$ is the number of regions $S_i, i \in [1, N]$ which satisfy $C_j \in S_i$ then
        for each region $S_i$, $C_j \in S_i$ do
            if $S_i \setminus C_j$ is continuous then
                remove $C_j$ from $S_i$
            end if
        end for
        if $\mathrm{NR}_{C_j} = 0$ then
            $C_j \leftarrow$ unexplored
        end if
    end if
end for
while unexplored $C_j$ exists do
    $i \leftarrow$ agent with least cells in its region $S_i$
    Evaluate each cell with one or more neighbour cells belonging to region $S_i$ with formula 4.5 as viewed by agent $i$
    find $C_{best} = \arg_j \max E_{i,j}$
    Add $C_{best}$ to $S_i$
    $C_{best} \leftarrow$ explored
end while
```

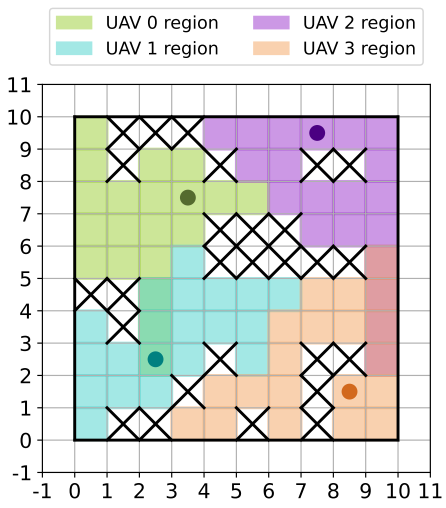
The numerical values of the constants $\sigma_a, \sigma_b, \sigma_d, \mathcal{E}_M$ can be found in Section 7.2. The outcome of the Environment partitioning part of the algorithm is shown in Figure 4.1. We can see that on some worlds the modified algorithm behaves almost identically to the original version. However, on some worlds there is a significant improvement in terms of redundancy.



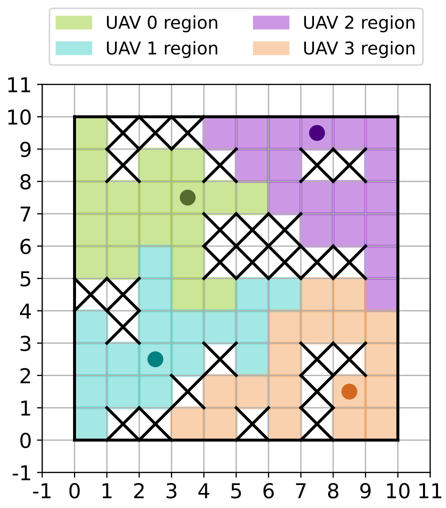
**(a) :** Original algorithm



**(b) :** Improved algorithm on world from (a)



**(c) :** Original algorithm



**(d) :** Improved algorithm on world from (c)

**Figure 4.1:** The original and improved environment partitioning - in (a) and (b) both versions perform almost identically on the same world, however, in (d) the improved version produces subregions with no overlaps as opposed to the original version (c) on the same world.

## 4.2 Trajectory Generation

When a corresponding subregion is assigned to each drone, the trajectories are planned in a way that each UAV explores each cell of its subregion. For this, the STC algorithm is used in [8].

### 4.2.1 Spanning Tree Coverage Algorithm

The main idea of the STC algorithm is to construct a tree $T_i$ which represents the relevant subregion $S_i$ corresponding to $i$th agent, divide every cell into four smaller subcells and then generate a trajectory such that each subcell is visited exactly once.

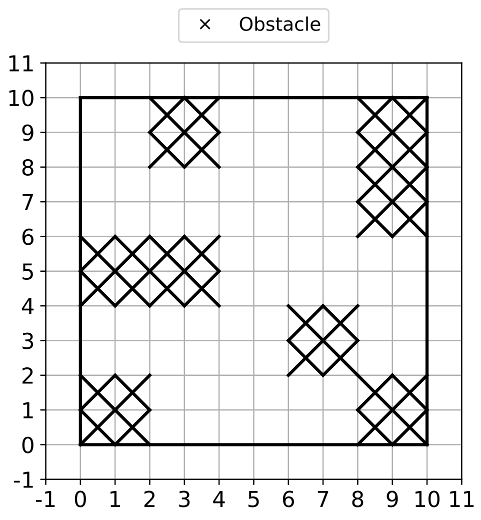
This approach performs optimally in terms of length of the resulting trajectory if *merging* (see Definition 3.13) takes place before the environment partitioning phase 4.1. Unfortunately this can be done only on some special worlds (see Figure 4.2). If such preprocessing cannot be applied, the length of the resulting trajectories increases dramatically, because instead of visiting only the center of each cell, the drone visits centers of each subcell.

To prevent this, the *environment* can be approximated using the *2l-size grid approximation* as in [6], but this means that some of the free cells from the *environment representation* will be considered obstacles and therefore will not be explored. This can even cause situations like the one in Figure 4.3 where no trajectory is generated, because the approximation results in a world full of obstacles. To cope with this problem, the rendition of the algorithm used in this work performs a test of the environment at the start of the runtime and the cells are merged only if no problematic blocks (which consist of both free cells and cells with obstacles) are found (see Algorithm 3). This means that on some worlds the STC algorithm performs optimally in terms of length of resulting trajectories, but on some worlds the trajectories are far longer than they need to be, which leaves space for improvement.

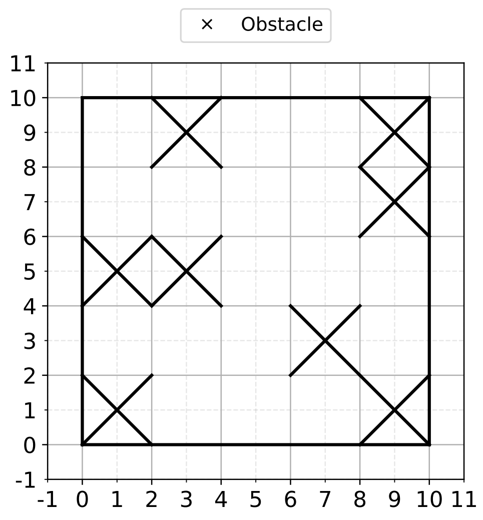
**Algorithm 3** The merging

```
if no 2x2 block of cells $B_j$ exists such that $C_{j,1}, C_{j,2} \in B_j$ and $C_{j,1} =$ obstacle, $C_{j,2} \neq$ obstacle then
    for each block of cells $B_j$ do
        replace $B_j$ with cell $C_j$
        exploration status of $C_j \leftarrow$ exploration status of $B_j$
    end for
end if
```

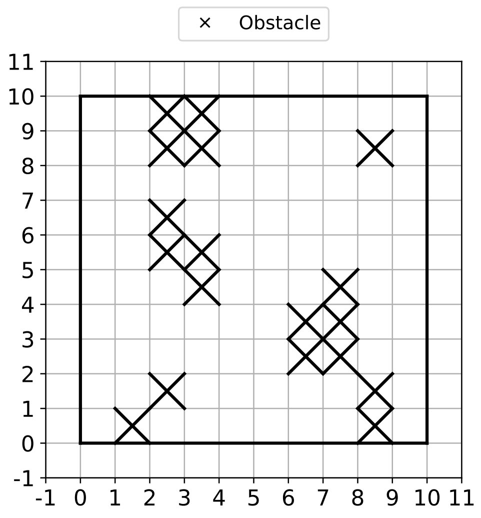
For a subregion $S_i$ the Spanning Tree $T_i$ is constructed as follows: At first the orientation of $S_i$ is determined. If $S_i$ is horizontal, then clusters of adjacent cells belonging to the same row are connected into horizontal branches and then the individual branches are connected with the leftmost cell. If $S_i$ is vertical, the procedure is the same, but the branches are formed from adjacent cells belonging to the same column, and they are connected via the lowermost cell.



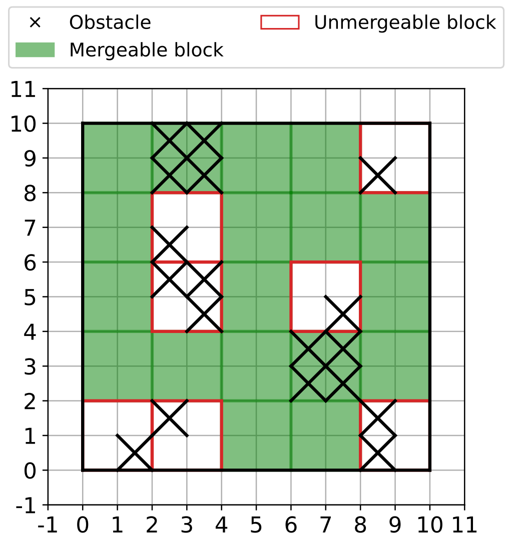
**(a) :** World suitable for merging



**(b) :** The result of merging on world (a)



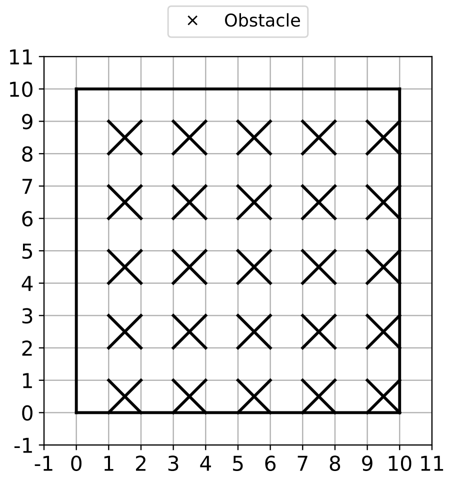
**(c) :** World where merging cannot be done



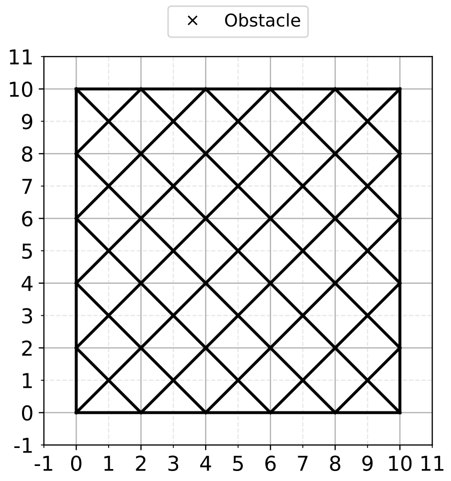
**(d) :** Problematic blocks of cells on world (c)

**Figure 4.2:** The merging

After the Spanning Tree is created, each cell is split into four subcells and a trajectory which passes through the centre of each subcell is generated. The outcomes of the STC algorithm can be seen in Figure 4.4. The algorithm is described in Algorithm 4.



**(a) :** Before approximation



**(b) :** After approximation

**Figure 4.3:** World which cannot be preprocessed with *2l-grid approximation*, because it would result in a world full of obstacles

## 4.2.2 Cycle Growing Algorithm

As we can see in Figure 4.4b, the trajectory generated by the STC algorithm is unnecessarily long when *merging* cannot be done. To cope with this, a new algorithm called the CG algorithm, which performs better on some worlds in terms of length of the resulting trajectories, is proposed.

In the CG algorithm the *surrounding cycle* $C_s$ of the *region* $S_i$ is found at first. A simple state machine, which starts in the left upper corner and navigates through the *region* using the *left hand rule* until the starting position is reached, is used for this (see Algorithm 5).

The pairs of cells are then appended to $C_s$ in a way that ensures that the continuity of the cycle is not violated. Both cells of each pair must be *neighbour cells*, must not contain an *obstacle*, none of them can already be in $C_s$, and both must have at least one cell from $C_s$ among its neighbours. If $C_s$ is *horizontal*, the horizontal pairs are added first and the vertical pairs afterwards. For *vertical* $C_s$ it is the opposite. This process of $C_s$ growing is explained in Algorithm 6.

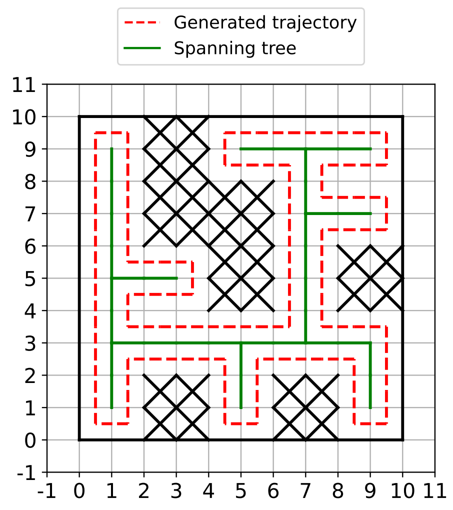
After this, all cells which have only *obstacles* or cells belonging to $C_s$ among its *neighbour cells* are added to the cycle and marked as *priority edges*. This is described in Algorithm 7.

**Algorithm 4** The Spanning Tree Coverage algorithm

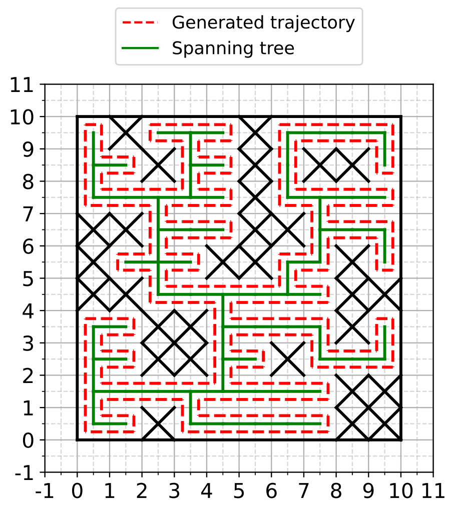
```
perform *merging* (run Algorithm 3)
initialize set xcoords and ycoords
**for** each cell $C_j \in S_i$ **do**
    add x coordinate of $C_j$ to xcoords
    add y coordinate of $C_j$ to ycoords
**end for**
**if** card(xcoords) $\geq$ card(ycoords) **then**
    orientation = horizontal
**else**
    orientation = vertical
**end if**
**if** orientation = horizontal **then**
    **for** each coord$_y \in$ ycoords **do**
        connect adjacent cells $C_j$ with the y coordinate $C_{j,y} =$ coord$_y$ into a branch BR$_j$
    **end for**
    **for** each branch BR$_j$ **do**
        connect to BR$_{j-1}$ and BR$_{j+1}$ with the leftmost cells
    **end for**
**else**
    **for** each coord$_x \in$ xcoords **do**
        connect adjacent cells $C_j$ with the x coordinate $C_{j,x} =$ coord$_x$ into a branch BR$_j$
    **end for**
    **for** each branch BR$_j$ **do**
        connect to BR$_{j-1}$ and BR$_{j+1}$ with the lowermost cells
    **end for**
**end if**
```

The algorithm then runs recursively on each unexplored continuous *subregion* of $S_i$ and the resulting cycles are connected. The whole CG algorithm is described in Algorithm 8.

The resulting directed graph is then converted into trajectory using another state machine which navigates through the graph using the *left hand rule* with the exception that edges which are marked as *priority edges* are assigned higher priorities than they would otherwise get from the *left hand rule*. Conversion of the generated cycle to a trajectory is described in Algorithm 9. The four phases of the CG algorithm are visualised in Figure 4.5. When this algorithm is combined with the Environment partitioning algorithm 4.1, the resulting CGWEP algorithm can be used as an alternative to the AWSTC algorithm from [8].



**(a) :** World where *merging* can be performed



**(b) :** World where *merging* cannot be performed

**Figure 4.4:** The results of the Spanning Tree Coverage algorithm

**Algorithm 5** Surrounding cycle retrieval algorithm

```
Find the upper left corner of unexplored area $C_{start}$
position ← $C_{start}$
heading ← right
finished ← $false$
initialize the cycle
while not finished do
    Add position to the cycle with the appropriate edge
    evaluate the neighbour cells of position with left hand rule and choose the cell $C_{best}$ and action $A_{best}$ with the highest priority
    position ← $C_{best}$
    heading ← action
    if position = $C_{start}$ then
        Add appropriate edge to enclose the cycle
        finished ← $true$
    end if
end while
```

## 4.3 Resulting Path Planning Algorithms

When we combine the Environment partitioning algorithm (Section 4.1) and the STC algorithm (Section 4.2.1), we obtain the AWSTC algorithm described in [8]. If we combine the STC algorithm with the modified version of the Environment partitioning algorithm (see Section 4.1.1), we obtain the IAWSTC algorithm. The combination of the improved Environment

**Algorithm 6** Growing of the *surrounding cycle*

```
if $C_s$ = horizontal then
    Process the horizontal pairs first and then the vertical pairs of neighbour cells in the following part of the algorithm
else
    Process the vertical pairs first and then the horizontal pairs of neighbour cells in the following part of the algorithm
end if
for each horizontal/vertical pair of neighbour cells $C_{j,1}, C_{j,2}$ do
    if $C_{j,1}, C_{j,2} \notin C_s$ and $C_{j,1}, C_{j,2} \neq obstacle$ then
        if NCS($C_{j,1}$)$\geq 1$ and NCS($C_{j,2}$)$\geq 1$, where NCS($C_j$) is the number of neighbour cells of cell $C_j$ which belong to surrounding cycle $C_s$ then
            Add $C_{j,1}, C_{j,2}$ to $C_s$ and add/remove appropriate edges
            Check the conditions and eventually add to $C_s$ also the pair of cells next to $C_{j,1}, C_{j,2}$.
        end if
    end if
end for
```

**Algorithm 7** The addition of priority edges

```
for each cell $C_j \notin C_s$ do
    neighbours $\leftarrow$ all neighbour cells of $C_j$
    u $\leftarrow 0$
    for each cell $C_n \in$ neighbours do
        if $C_n = obstacle$ or $C_n \in C_s$ then
            u $\leftarrow$ u $+ 1$
        end if
    end for
    if u = len(neighbours), where len(vec) is a function which returns the number of elements in vec then
        Add $C_j$ to $C_s$
        add $C_j$ to priority edges
    end if
end for
```

partitioning algorithm and the CG algorithm (Section 4.2.2) results in the CGWEP algorithm. We can see the execution of the IAWSTC and CGWEP algorithms for three UAVs in Figure 4.6.

**Algorithm 8** The Cycle Growing algorithm

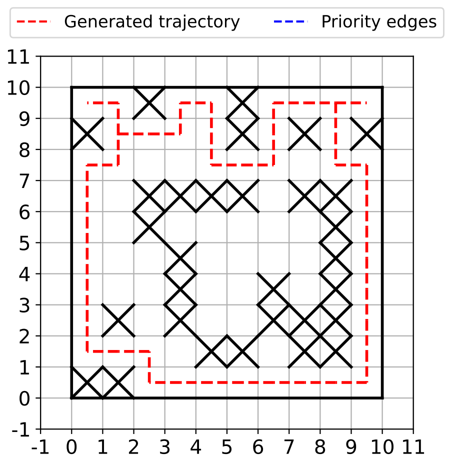
```
initialize $C_s$
Run Algorithm 5
Run Algorithm 6
Run Algorithm 7
for each cell $C_j \in C_s$ do
    $C_j \leftarrow$ explored
end for
if unexplored cell exists then
    run the whole Algorithm 8 on the unexplored subregion.
    connect the result to $C_s$
end if
```

**Algorithm 9** Cycle to trajectory conversion

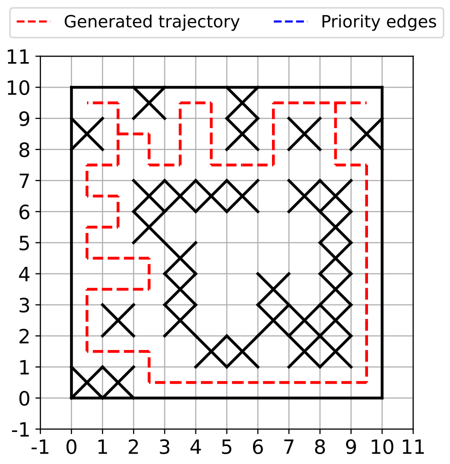
```
Initialize trajectory Traj
Find the upper left corner of the cycle $C_s$, $C_{start}$
position $\leftarrow C_{start}$
heading $\leftarrow$ right
finished $\leftarrow$ false
while not finished do
    Add position to Traj
    if edge pe from position in priority edges then
        remove pe from priority edges
        e $\leftarrow$ pe
    else
        e $\leftarrow$ choose a cell from edges of position using the left hand rule
    end if
    position $\leftarrow$ goal(e)
    update heading
    if position $= C_{start}$ then
        finished $\leftarrow$ true
    end if
end while
```

## 4.4 Time Complexity

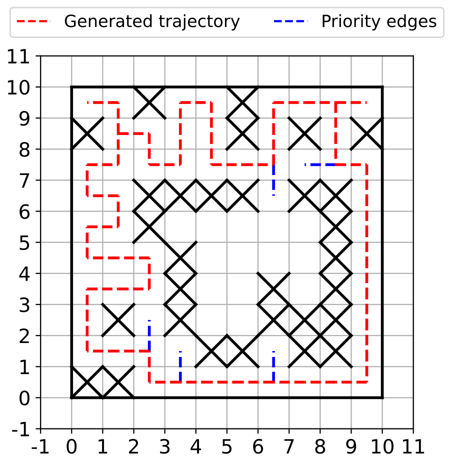
The time complexity of the original and improved versions of the Environment partitioning algorithm (Algorithm 1) is $\mathcal{O}(n^2)$. The time complexity of the STC algorithm (Algorithm 4) is $\mathcal{O}(n)$ and the complexity of the CG algorithm (Algorithm 8) is also $\mathcal{O}(n)$. Therefore, the resulting combination is $\mathcal{O}(n^2)$ for both the original AWSTC algorithm from [8] and the proposed IAWSTC and CGWEP algorithms.



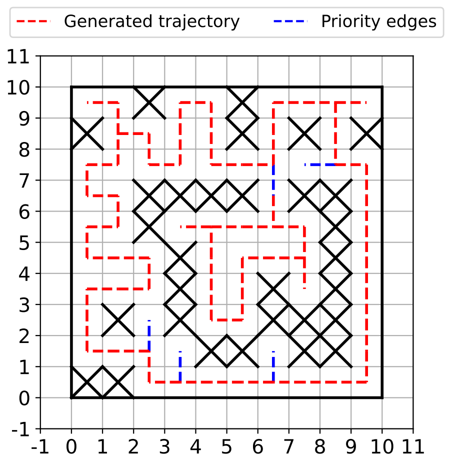
**(a) :** Step 1 - The surrounding cycle



**(b) :** Step 2 - Extended cycle

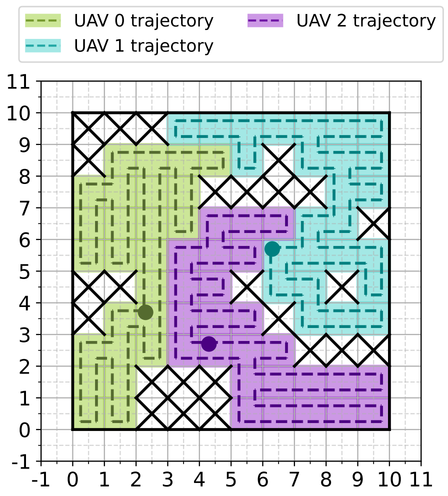


**(c) :** Step 3 - Priority edges

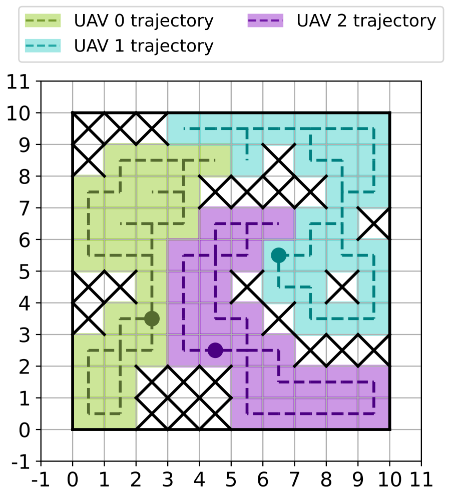


**(d) :** Step 4 - The recursive run on unexplored subregion

**Figure 4.5:** The phases of the Cycle Growing algorithm



**(a) :** The IAWSTC algorithm



**(b) :** The CGWEP algorithm

**Figure 4.6:** The outcomes of IAWSTC and CGWEP algorithms for three drones.

# Chapter 5

# Trajectory Smoothing

The trajectories generated by the algorithms described in Section 4.2 contain many rough turns. These turns can be smoothed to make it easier for the UAV's controller to track the generated trajectory. A fundamental yet effective optimisation based approach was chosen for the smoothing. It may seem that the smoothing is unnecessary, because in case the turn is too sharp, the controller deviates from the trajectory reference, which results in a smooth turn. However, when this happens, we lose control of the actual path, and some cells may remain unexplored, as the UAV does not pass through their centres. Therefore, it is better to smooth the generated trajectories.

There are various approaches to trajectory smoothing that can be generally divided into three main categories - *interpolation based methods*, *methods using special curves*, and *optimisation methods* [22]. The *interpolation based methods* use *polynomial interpolation* [5], *Bézier curves* [26], *cubic splines* [18], *B-splines* [3], or *NURBS curves* [13]. The *methods using special curves* use *Dubin's curves* [19], *clothoids* [4], *hypocycloids* [21], and other special curves [22]. The *optimisation methods* treat the trajectory smoothing as an optimisation problem, where energy, time of execution, and other criteria are minimized. In [27] trajectories are viewed as elastic bands where shape and speed are iteratively improved by solving convex optimisation problems. In [14] smoothness is achieved by minimizing the integral of squared acceleration using a numerical solution of a set of nonlinear equations obtained by the calculus of variation. A novel approach based on empirical mode decomposition, where the high frequencies of changes in $x$ and $y$ coordinates are discarded for better smoothness, is presented in [25]. Another unconventional approach is introduced in [11], where neural networks are used to perform the trajectory smoothing. For an extensive overview of state of the art trajectory smoothing methods, see [22].

## 5.1 Trajectory Smoothing Using Least Squares Solution

The main criterion for the trajectory smoothing algorithm was simplicity and low computational complexity, because it is used for long trajectories and the computations are carried out by the UAV's control unit during flight. Another important condition is that the trajectory after smoothing should still go through each of the points of the original trajectory, otherwise the agent would not fulfil the definition of exploration laid down in Definition 3.2. For those reasons, the TSULSS approach, which meets all the criteria mentioned previously, was developed.

### 5.1.1 The Mathematical Principle

The main idea of the proposed TSULSS approach is to insert $p$ equally distant points between each two points of the original trajectory (see Figure 5.1) and then minimise a function, which describes the curvature of the whole new trajectory $\tau$. The ideal function for this would be the sum of squared angles between pairs of consecutive vectors $v_i, v_{i+1} \in \tau$. However, this function can not be formulated as least squares solution of a system of linear equations. Therefore, the vectors $v_i, v_{i+1}$ were decomposed into three consecutive points $P_i, P_{i+1}, P_{i+2}$ and the distance $d_{mid,i}$ of the middle point $P_{i+1}$ from the centre of mass of the three points was used to describe the angle between the two vectors $v_i, v_{i+1}$. The distance $d_{mid,i}$ is visualised in Figure 5.2 and the relation of the distance $d_{mid,i}$ and the angle between $v_i, v_{i+1}$ can be seen in Figure 5.3.

The function for the description of the curvature of $\tau$ is the sum of squared distances $d_{mid,i}$ for each triad of consecutive points from the trajectory $\tau$:

$$f(\tau) = \sum_{i=0}^{N-2} \left[ \left( \frac{P_{i,x} + P_{i+1,x} + P_{i+2,x}}{3} - P_{i+1,x} \right)^2 + \left( \frac{P_{i,y} + P_{i+1,y} + P_{i+2,y}}{3} - P_{i+1,y} \right)^2 \right]. \quad (5.1)$$

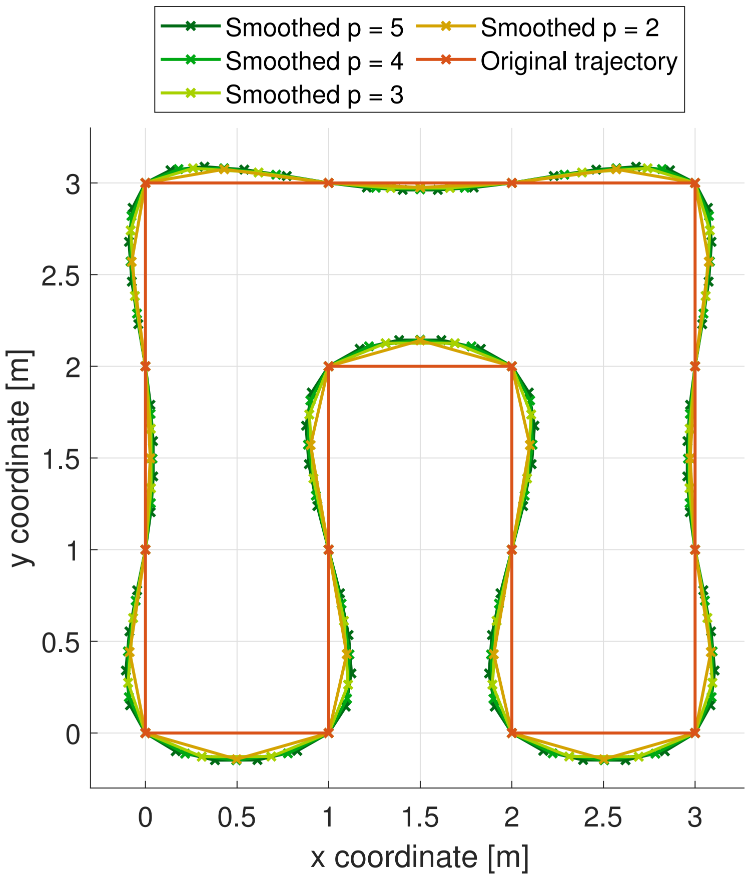


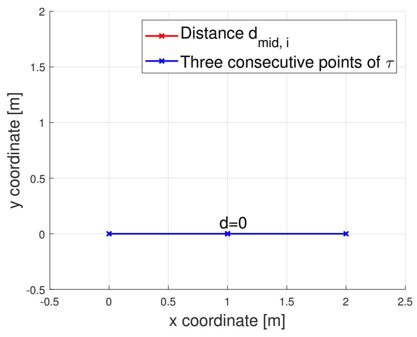
**Figure 5.1:** The meaning of the $p$ parameter - with higher values of $p$, more points are inserted into the trajectory, which leads to smoother turns.

Equation (5.1) describes the curvature of a trajectory $\tau$, but we must introduce the shift of individual points, because that is what needs to be computed. Therefore, each point $P_i$ is replaced by $P_i + \lambda_i\,\delta_i$, where

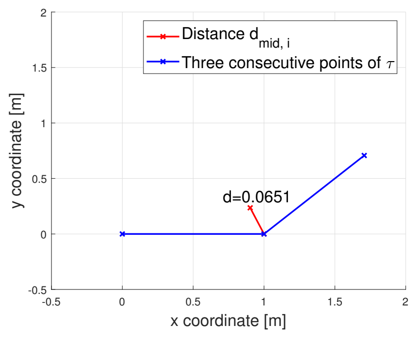
$$\lambda_i = \begin{cases} 0 & P_i \text{ belongs to the original trajectory} \\ 1 & P_i \text{ was added during the first phase of trajectory smoothing,} \end{cases}$$

and $\delta_i$ is the change of the $i$-th point of the trajectory. The term $\lambda_i$ ensures that the points of the original trajectory will not be changed during the smoothing process, and therefore, the UAV will *explore* all the *cells*. The resulting function is:

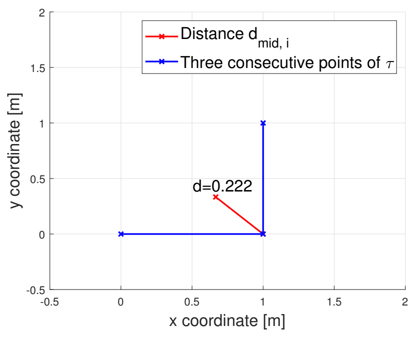
$$f(\tau) = \sum_{i=0}^{N-2}\left[\left(\frac{P_{i,x} + \lambda_i\,\delta_{i,x}}{3} - 2\,\frac{P_{i+1,x} + \lambda_{i+1}\,\delta_{i+1,x}}{3} + \frac{P_{i+2,x} + \lambda_{i+2}\,\delta_{i+2,x}}{3}\right)^2 \right.$$
$$\left. + \left(\frac{P_{i,y} + \lambda_i\,\delta_{i,y}}{3} + 2\,\frac{P_{i+1,y} + \lambda_{i+1}\,\delta_{i+1,y}}{3} + \frac{P_{i+2,y} + \lambda_{i+2}\,\delta_{i+2,y}}{3}\right)^2\right]. \quad (5.2)$$



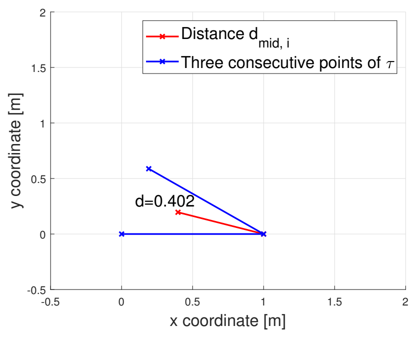
**(a) :** No turn



**(b) :** $< 90°$ turn



**(c) :** $90°$ turn



**(d) :** $> 90°$ turn

**Figure 5.2:** Distance $d_{mid,i}$ for various turns

For enclosed trajectories (where $P_1 = P_N$) one more term must be added to round off the connection of the start and the end. The final function with this term is:

$$f_f(\tau) = f(\tau) + \left( \frac{P_{N-1,x} + \lambda_{N-1}\,\delta_{N-1,x}}{3} - 2\,\frac{P_{1,x} + \lambda_1\,\delta_{1,x}}{3} + \frac{P_{2,x}\lambda_2\,\delta_{2,x}}{3} \right)^2 + \left( \frac{P_{N-1,y} + \lambda_{N-1}\,\delta_{N-1,y}}{3} - 2\,\frac{P_{1,y} + \lambda_1\,\delta_{1,y}}{3} + \frac{P_{2,y}\lambda_2\,\delta_{2,y}}{3} \right)^2 . \tag{5.3}$$

This equation can be rewritten in matrix form for better clarity:

$$f_f(\tau) = \|\mathbf{A}\,\mathbf{x} - \mathbf{b}\|^2, \tag{5.4}$$

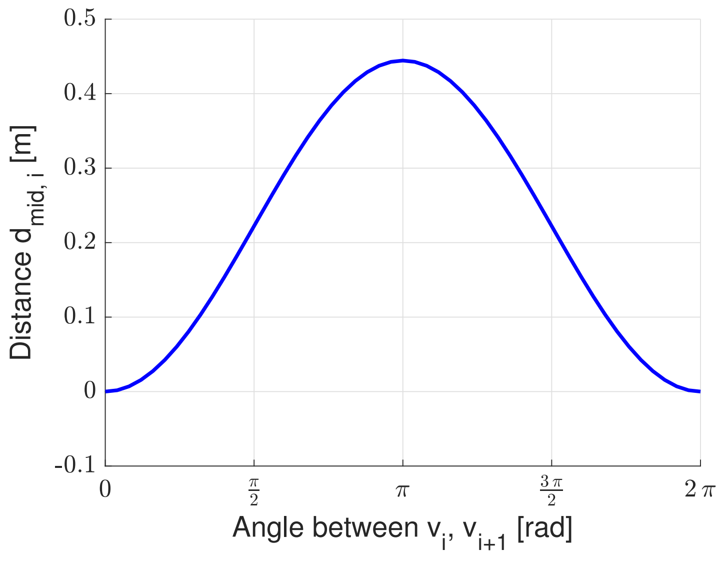


**Figure 5.3:** The relation of distance $d_{mid,i}$ and angle between $v_i, v_{i+1}$

where

$$
\mathbf{A} = \begin{bmatrix}
\frac{\lambda_1}{3} & 0 & -\frac{2\,\lambda_2}{3} & 0 & \frac{\lambda_3}{3} & 0 & 0 & \dots & \dots & \dots & \dots & \dots & 0 \\
0 & \frac{\lambda_1}{3} & 0 & -\frac{2\,\lambda_2}{3} & 0 & \frac{\lambda_3}{3} & 0 & \dots & \dots & \dots & \dots & \dots & 0 \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
0 & \dots & \dots & \dots & \dots & \dots & 0 & \frac{\lambda_{N-2}}{3} & 0 & -\frac{2\,\lambda_{N-1}}{3} & 0 & \frac{\lambda_N}{3} & 0 \\
0 & \dots & \dots & \dots & \dots & \dots & \dots & 0 & \frac{\lambda_{N-2}}{3} & 0 & -\frac{2\,\lambda_{N-1}}{3} & 0 & \frac{\lambda_N}{3} \\
-\frac{2\,\lambda_1}{3} & 0 & \frac{\lambda_2}{3} & 0 & \dots & \dots & \dots & \dots & 0 & \frac{\lambda_{N-1}}{3} & 0 & \dots & 0 \\
0 & -\frac{2\,\lambda_1}{3} & 0 & \frac{\lambda_2}{3} & 0 & \dots & \dots & \dots & \dots & 0 & \frac{\lambda_{N-1}}{3} & 0 & 0
\end{bmatrix},
$$

$$
\mathbf{x} = \begin{bmatrix} \delta_{1,x} & \delta_{1,y} & \delta_{2,x} & \delta_{2,y} & \delta_{3,x} & \delta_{3,y} & \dots & \delta_{N,x} & \delta_{N,y} \end{bmatrix}^T,
$$

$$\mathbf{b} = \begin{bmatrix} -\frac{1}{3}P_{1,x} + \frac{2}{3}P_{2,x} - \frac{1}{3}P_{3,x} \\ -\frac{1}{3}P_{1,y} + \frac{2}{3}P_{2,y} - \frac{1}{3}P_{3,y} \\ \vdots \\ -\frac{1}{3}P_{N-2,x} + \frac{2}{3}P_{N-1,x} - \frac{1}{3}P_{N,x} \\ -\frac{1}{3}P_{N-2,y} + \frac{2}{3}P_{N-1,y} - \frac{1}{3}P_{N,y} \end{bmatrix}.$$

The optimisation problem is then formulated as:

$$\min_{x \in \mathcal{R}^{2N}} \|\mathbf{A}\,\mathbf{x} - \mathbf{b}\|^2. \tag{5.5}$$

For better control over the resulting trajectories, the Tikhonov regularization was added to penalize significant changes in the trajectory, introducing the parameter $\mu$, which controls how much the smoothed trajectory can deviate from the original trajectory. The effect of the parameter $\mu$ can be seen in Figure 5.4. The final optimisation problem with Tikhonov regularization is:

$$\min_{x \in \mathcal{R}^{2N}} \left( \|\mathbf{A}\,\mathbf{x} - \mathbf{b}\|^2 + \mu\|\mathbf{x}\|^2 \right). \tag{5.6}$$

This problem can be solved by singular value decomposition, QR decomposition, or normal equations. In this work the normal equation solution was chosen, because of its low computational complexity.

### 5.1.2 Performance

We can see the result of the TSULSS approach in Figure 5.5. The algorithm performs as expected and it is also possible to change the resulting trajectories with the parameters $p$ and $\mu$. The algorithm performs fast enough for real time applications for trajectories with approximately 100 points (before the insertion of $p$ points). Longer trajectories must be decomposed into shorter subtrajectories, which can be smoothed separately and then joined into one final smooth trajectory. When dividing the trajectory into subtrajectories, the split point should not be in the middle of a turn.

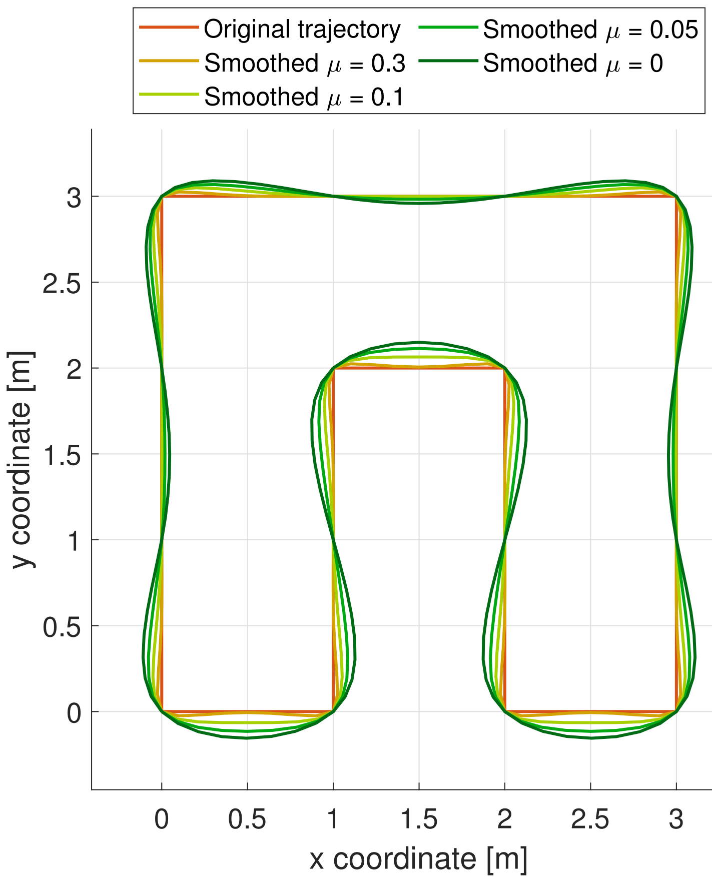


**Figure 5.4:** The meaning of the $\mu$ parameter - smoothed trajectories differ less from the original trajectory for higher values of $\mu$

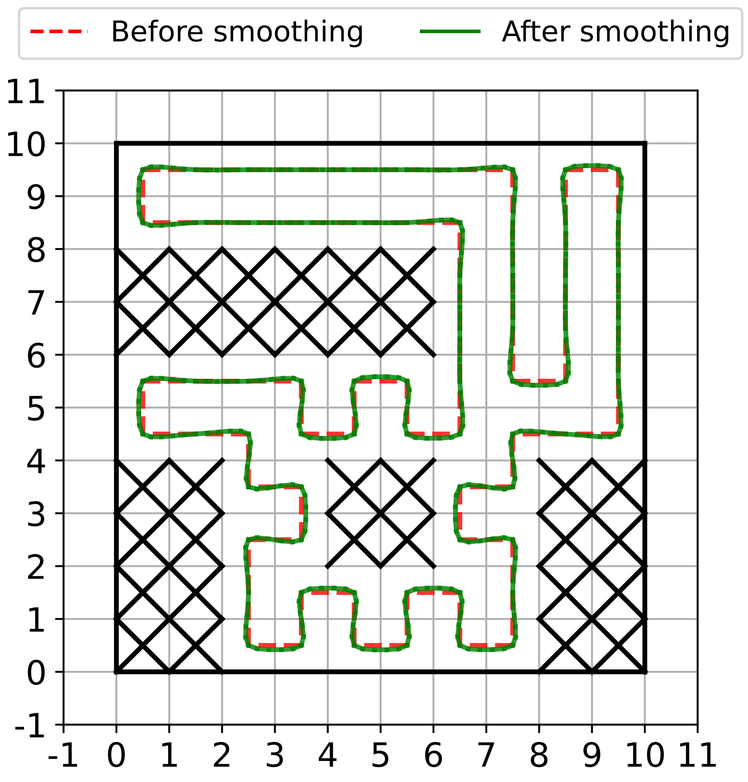


**Figure 5.5:** The result of trajectory smoothing

Chapter 6

# Online Obstacle Detection And Avoidance

After the trajectories are generated, each drone is commanded to its starting position. After this, they start tracking the trajectories, inform the other agents about the visited *cells*, and periodically check for obstacles colliding with their trajectory. Whenever an unexpected obstacle appears, which is not in the obstacle map, the UAV immediately stops and observes the obstacle for time $t_o$. If the position of the obstacle does not change during this time, it is classified as *Static obstacle*, otherwise it is classified as *Dynamic obstacle*. For an example of static and dynamic obstacles, see Figure 6.1.

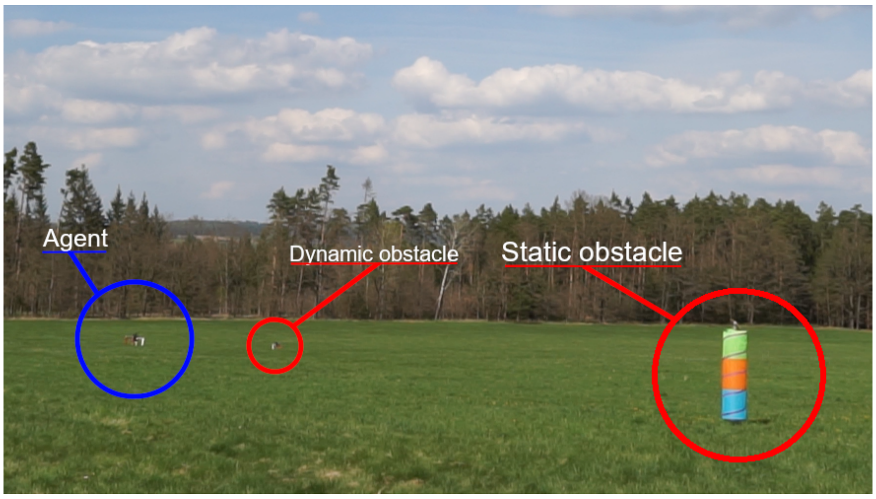


**Figure 6.1:** Static and dynamic obstacles

## 6.1 Dynamic Obstacles

*Dynamic obstacles* are obstacles, which move and therefore cannot be in the obstacle map. When a drone detects a dynamic obstacle colliding with it's trajectory, it suspends the trajectory execution and waits until the path is clear. Nothing is signaled to other UAVs. In this work, we assume dynamic obstacles aware of its surroundings which avoid crashing into the UAV. Therefore, it is sufficient for the drone to wait until the detected dynamic obstacle disappears.

## 6.2 Static Obstacles not Known Beforehand

When the detected obstacle is classified as *Static obstacle*, the agent sends the position of the obstacle to the other agents. The other agents add this new obstacle to their obstacle maps and suspend current trajectory execution. After this, the trajectories are *replanned*.

## 6.3 Replanning

During the replanning process all cells that were already visited by the agents are marked as *explored*, and the cells that were not yet visited and do not contain an obstacle are marked as *unexplored*. After this, the whole exploration algorithm is repeated. *Replanning* is also described in Algorithm 10.

**Algorithm 10** Replanning

```
Watch for obstacles
if detected obstacle then
    Wait for $t_o$
    if obstacle position changed then
        Wait for clear path
        Resume trajectory execution
    else
        Add obstacle to the obstacle map
        Signal obstacle position to other agents
        Other agents add obstacle to their obstacle maps and suspend trajectory execution
        Mark visited cells as explored and unvisited cells without obstacles as unexplored
        Repeat the exploration algorithm
    end if
end if
```

# Chapter 7

# Implementation

## 7.1 Used Programming Languages and Environments

The AWSTC, IAWSTC and CGWEP algorithms were implemented in Python and the Robotic Operating System (ROS)[1] using C++ for the simulation and hardware experiments.

The STC algorithm (Section 4.2.1), which is the trajectory generation part of the AWSTC algorithm, the CG algorithm (Section 4.2.2) which is the trajectory generation part of the CGWEP algorithm and the Environment partitioning algorithm (Section 4.1) along with it's modified version (see Section 4.1.1), which is the first phase of both IAWSTC and CGWEP algorithms, were implemented in Python only.

The Trajectory smoothing using least squares solution (Section 5.1) was developed in Matlab 2020a, then implemented in Python and also ROS using C++ for simulation and hardware experiments.

[1] `https://wiki.ros.org/`

## 7.2 Numerical Values of Constants and Parameters

The values of various constants and parameters used in the implementation of the algorithms can be seen in Table 7.1.

**Table 7.1:** Values of used constants and parameters

| | Name | Value |
|---|---|---|
| Environment partitioning | $\sigma_a$ | 1 |
| | $\sigma_b$ | 100 |
| | $\sigma_d$ | 10 |
| | $\mathcal{E}_M$ | 10000 |
| Trajectory smoothing | $p$ | 5 |
| | $\mu$ | 0.15 |
| Replanning | $t_o$ | 2 [s] |

# Chapter 8

# Quantitative Tests

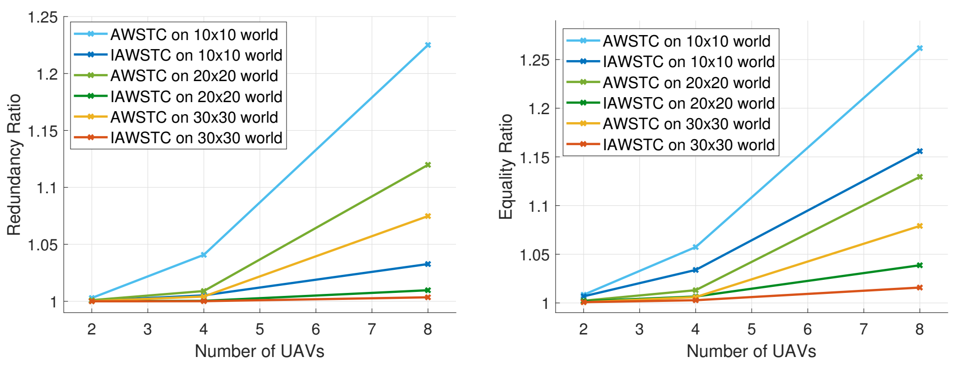
The individual parts of the algorithm were realized as standalone applications in Python, and multiple tests were carried out to determine the benefits of proposed algorithms and algorithm modifications.

## 8.1 Environment Partitioning Improvement

To prove that the modifications proposed in Section 4.1.1 improve the behaviour of the algorithm from [8], the original (AWSTC) and modified (IAWSTC) versions of the algorithm were launched on 1000 worlds with dimensions 10 x 10 cells, 500 worlds with dimensions 20 x 20 cells, and 200 worlds with dimensions 30 x 30 cells. All the worlds and starting points of the agents were randomly generated. The main criteria for the comparison were the redundancy ratio (see Definition 3.16) and the equality ratio (see Definition 3.17). The results are listed in Table 8.1 and are also visualised in Figure 8.1. We can clearly see that the modified version of the Environment Partitioning algorithm used in IAWSTC effectively improves both the redundancy ratio and the equality ratio.

**Table 8.1:** The quantitative tests of the environment partitioning algorithm

| World size | Number of agents | Redundancy ratio | | Equality ratio | |
|---|---|---|---|---|---|
| | | AWSTC | IAWSTC | AWSTC | IAWSTC |
| 10x10 | 2 | 1.0029 | 1.0005 | 1.0084 | 1.0068 |
| | 4 | 1.0408 | 1.0052 | 1.0574 | 1.0339 |
| | 8 | 1.225 | 1.0327 | 1.2617 | 1.1559 |
| 20x20 | 2 | 1.001 | 1.0001 | 1.0023 | 1.0018 |
| | 4 | 1.009 | 1.0004 | 1.0131 | 1.0066 |
| | 8 | 1.1198 | 1.0097 | 1.1295 | 1.0387 |
| 30x30 | 2 | 1.0001 | 1.0 | 1.0008 | 1.0008 |
| | 4 | 1.0042 | 1.0001 | 1.006 | 1.0028 |
| | 8 | 1.0748 | 1.0035 | 1.0791 | 1.0158 |



**(a) :** Redundancy ratio

**(b) :** Equality ratio

**Figure 8.1:** The dependency of redundancy ratio and equality ratio on number of drones and world size

## 8.2 Path Planning Algorithm Comparison

Quantitative tests were also carried out to provide a comparison between the two presented algorithms for trajectory generation (see Section 4.2). Both STC and CG algorithms were tested on 1000 randomly generated worlds with randomly generated dimensions, number of agents, and starting points. The experiments were performed on worlds, where merging can be done and therefore the STC algorithm acts optimally and also on worlds where merging cannot be done, separately. The measured criteria were the length ratio (see Definition 3.18) and the curvature ratio (see Definition 3.19). The results can be found in Table 8.2.

**Table 8.2:** The quantitative tests of the path planning algorithms

| | Algorithm | Length ratio | Curvature ratio |
|---|---|---|---|
| With merging | STC | 1.0 | 27.731 |
| | CG | 1.0014 | 34.0451 |
| Merging not possible | STC | 1.98 | 86.3356 |
| | CG | 1.1733 | 74.3884 |

We can deduce from the results of quantitative tests that on worlds, where merging can be done, the STC algorithm outperforms the CG algorithm, both in terms of length and in terms of curvature. This is no surprise as the STC algorithm is optimal in terms of trajectory length on such worlds and the way it is constructed guarantees low curvature. However, on the worlds where merging cannot be realized, the trajectories generated by the CG algorithm proved to be far shorter. The CG algorithm performed better also in terms of curvature on such worlds. Nevertheless, one valuable feature of the STC algorithm is that it provides an upper bound for the maximal turn size - for trajectory $\tau$ consisting of points $P_1, \dots, P_M$:

$$\max_{i \in \{1,\dots,M-2\}} \left| \arccos\left( \frac{v_i \cdot v_{i+1}}{|v_i|\,|v_{i+1}|} \right) \right| \leq \frac{\pi}{2}, \tag{8.1}$$

where

$$v_i = P_{i+1} - P_i, \tag{8.2}$$
$$v_{i+1} = P_{i+2} - P_{i+1}. \tag{8.3}$$

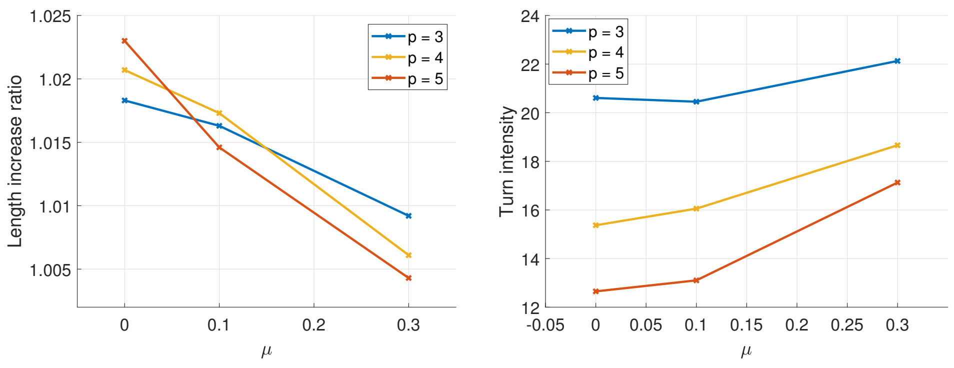
This does not hold true for the CG algorithm.

## 8.3 Trajectory Smoothing Evaluation

To examine the influence of the parameters $p$ and $\mu$ on the performance of the trajectory smoothing algorithm, various combinations of values of $p$ and $\mu$ were tested on 100 random worlds with random sizes $s \in \{$10 x 10, 11 x 11, ...,25 x 25$\}$ and random numbers of agents $N \in \{2, 3, \dots, 6\}$. The measured criteria were the length increase ratio (see Definition 3.20) and the turn intensity (see Definition 3.21). The results are listed in Table 8.3 and are also visualised in Figure 8.2. The results verified the assumption that the increase of $\mu$ parameter or $p$ parameter values lead to smoother trajectories with greater lengths. On the other hand, smaller values of these parameters lead to shorter trajectories with sharper turns.

**Table 8.3:** The quantitative tests of the trajectory smoothing algorithm

| $\mu$ parameter | $p$ parameter | Length increase ratio | Turn intensity |
|---|---|---|---|
| 0 | 3 | 1.0183 | 20.6074 |
| | 4 | 1.0207 | 15.3694 |
| | 5 | 1.0230 | 12.6498 |
| 0.1 | 3 | 1.0163 | 20.4525 |
| | 4 | 1.0173 | 16.0535 |
| | 5 | 1.0146 | 13.1025 |
| 0.3 | 3 | 1.0092 | 22.1290 |
| | 4 | 1.0061 | 18.6607 |
| | 5 | 1.0043 | 17.1271 |



**(a) :** Length **(b) :** Turn intensity

**Figure 8.2:** The dependency of length increase ratio and turn intensity on the $p$ and $\mu$ parameters

# Chapter 9

# Simulation Experiments

## 9.1 Environment

The simulation experiments were performed in the Gazebo simulator[1] (see Figure 9.1) on a laptop with Intel Core i7-8565U CPU with 8 GB RAM. Due to the limited computing power of the used hardware, the maximum of 3 drones could be used in the simulation experiments. However, this limitation was not an important issue, as three drones are sufficient to test the functionality of the algorithms.

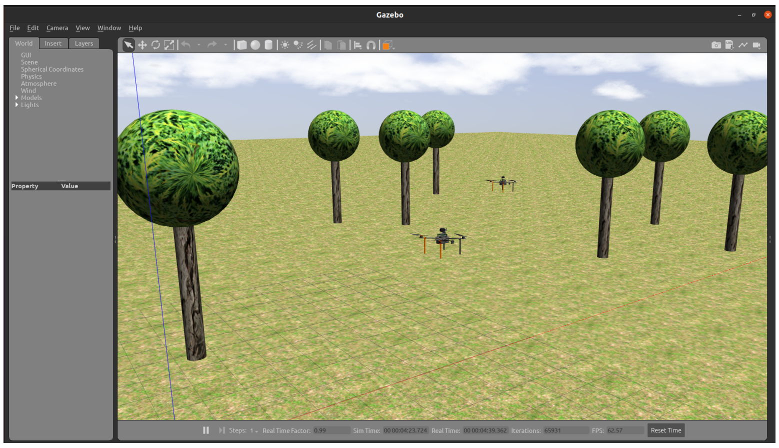


**Figure 9.1:** The simulation environment

[1] https://gazebosim.org/home

## 9.2 Results

Both algorithms were tested in the Exploration experiment (Section 9.2.1) and the Replanning experiment (Section 9.2.2) tested the online obstacle detection (Chapter 6) and replanning (Section 6.3). All the simulation experiments were successful.

### 9.2.1 Exploration Experiment

In this experiment, three UAVs explored a world with all the obstacles known beforehand. The dimensions of the world were 15 x 15 cells, which corresponds to 45 x 45 [m]. We can see the result of the IAWSTC algorithm in Figure 9.2 and the result of the CGWEP algorithm in Figure 9.3. It is evident, that the trajectories generated by the latter algorithm are shorter and also have smaller overlaps. Both figures are screenshots from a visualization tool called RViz[2] with additionally marked obstacles.

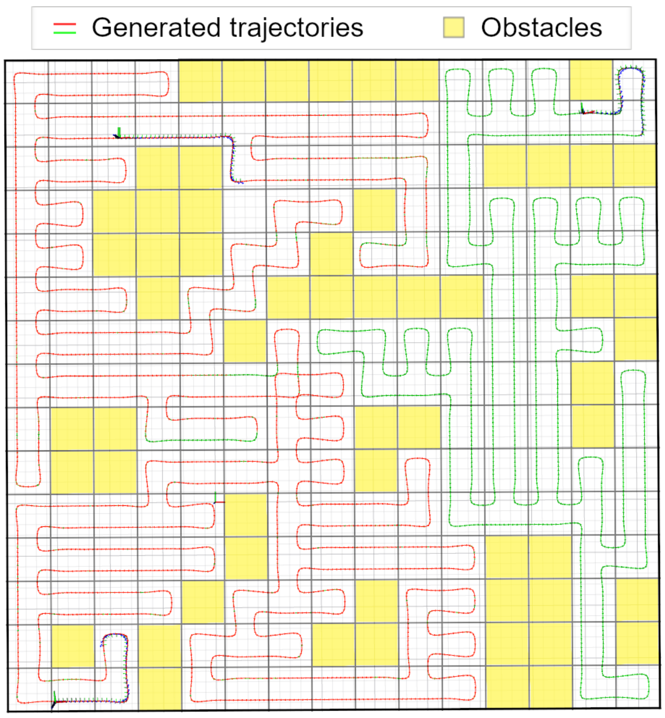


**Figure 9.2:** IAWSTC algorithm in simulation

[2]http://wiki.ros.org/rviz

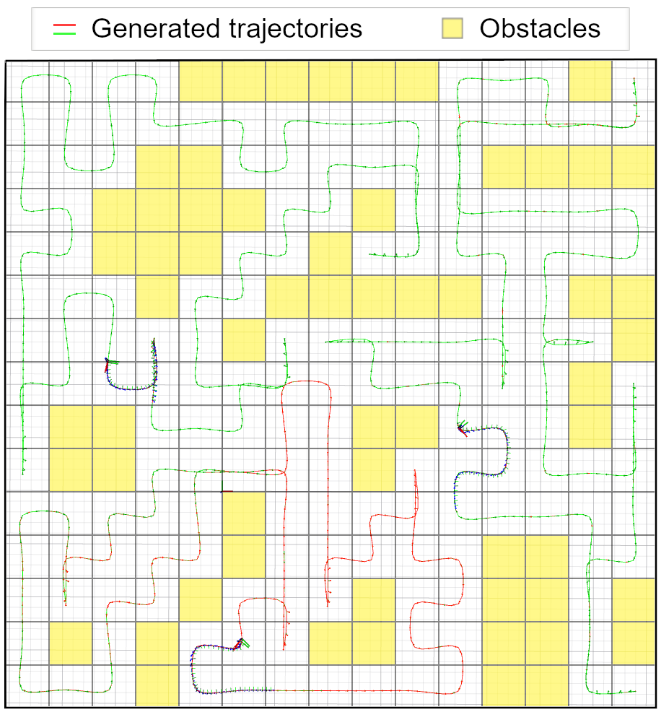


**Figure 9.3:** CGWEP Algorithm in simulation

## 9.2.2 Replanning

In the Replanning experiment an obstacle, which was not marked in the obstacle map, had been added. Two UAVs explored a world of 10 x 10 cells (30 x 30 [m]) using the CGWEP algorithm and when one of them noticed the unexpected obstacle, they both stopped and replanned their paths. We can see the original trajectories before replanning in Figure 9.4 and the modified trajectories after replanning in Figure 9.5.

[3]The recording of the Replaning experiment can be found here: `http://mrs.felk.cvut.cz/chleboun-2022-bp`

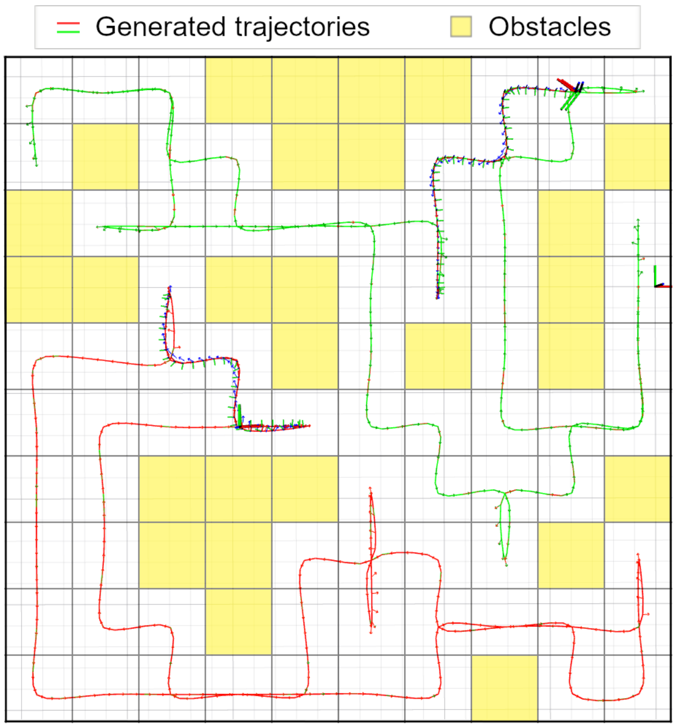


**Figure 9.4:** Replanning experiment[3]before unexpected obstacle detection

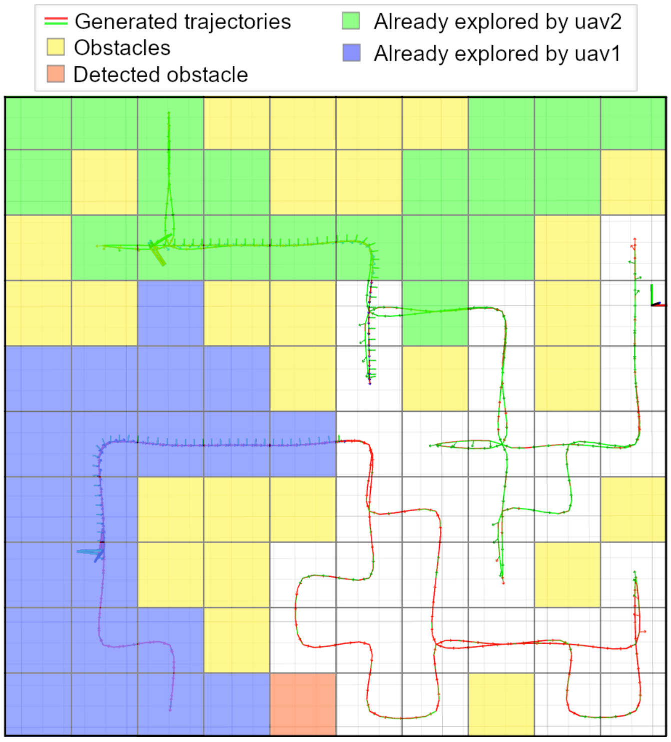


**Figure 9.5:** Replanning experiment after unexpected obstacle detection

# Chapter 10

# Hardware Experiments

## 10.1 Hardware description

Hardware experiments were performed on three UAVs of type DJI F450[1] with GPS receiver for localization, 2D LiDAR[2] for obstacle detection, Intel NUC i7[3] onboard computer and rangeFinder for height estimation. One of the drones (with propellers taken off) can be seen in Figure 10.1.

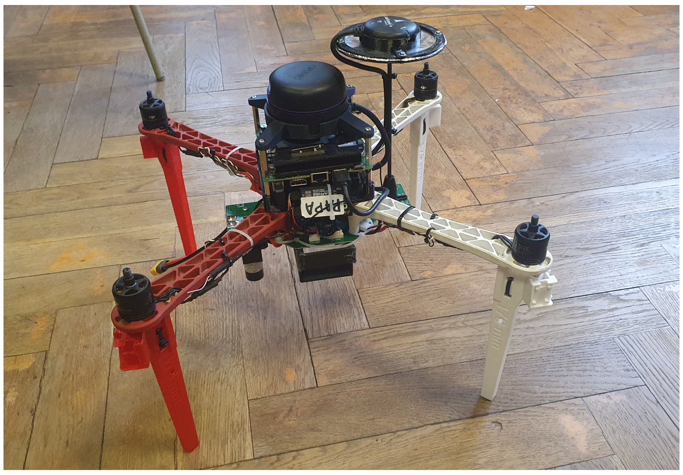


**Figure 10.1:** One of the UAVs used in the experiment

[1]https://www.dji.com/cz/flame-wheel-arf/feature

[2]https://www.slamtec.com/en/Lidar/A3

[3]https://www.intel.com/content/www/us/en/products/details/nuc/mini-pcs.html

The chosen environment was a meadow in the South Bohemian Region of the Czech Republic. The aerial view of the environment can be seen in Figure 10.2.

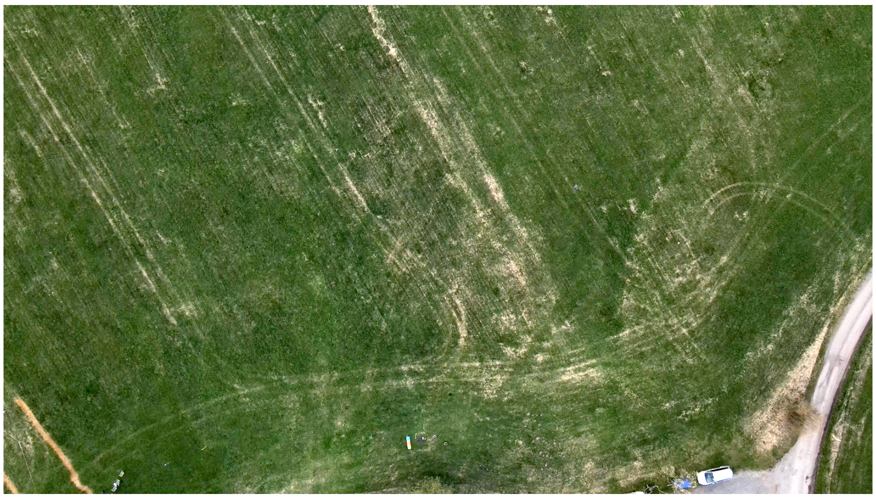


**Figure 10.2:** The aerial view of the environment

## 10.2 Results

In the experiment, the IAWSTC and CGWEP algorithms were tested on a 40 x 40 [m] region. The world representation dimensions were 10 x 10 cells. The cell size was set to 4 x 4 meters so that the collision prediction mechanisms of the MRS UAV system core would not affect the generated trajectories. The outcome of the IAWSTC algorithm can be seen in Figure 10.3 and the outcome of the CGWEP algorithm in Figure 10.3. Both experiments were successful.

[4]The recording of the IAWSTC algorithm hardware experiment can be found here: `http://mrs.felk.cvut.cz/chleboun-2022-bp`

[5]The recording of the CGWEP algorithm hardware experiment can be found here: `http://mrs.felk.cvut.cz/chleboun-2022-bp`

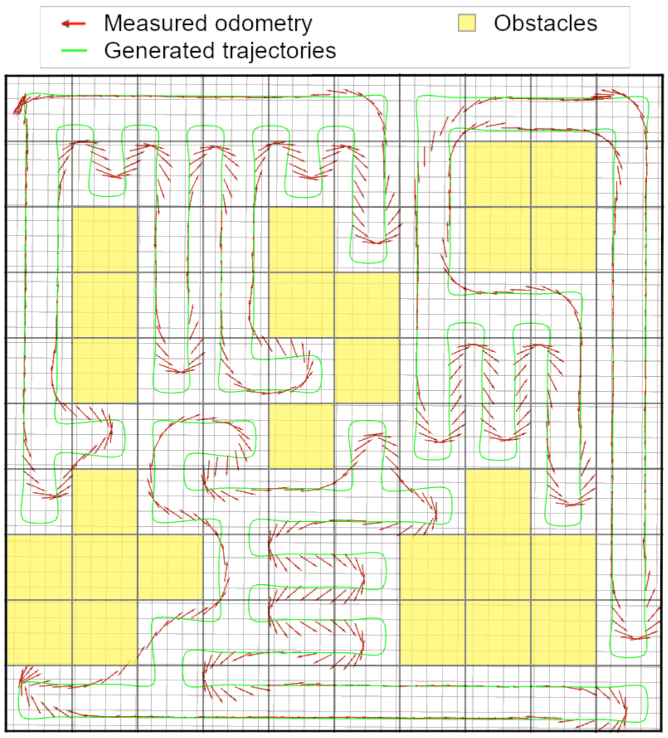


**Figure 10.3:** IAWSTC algorithm on real UAVs[4]

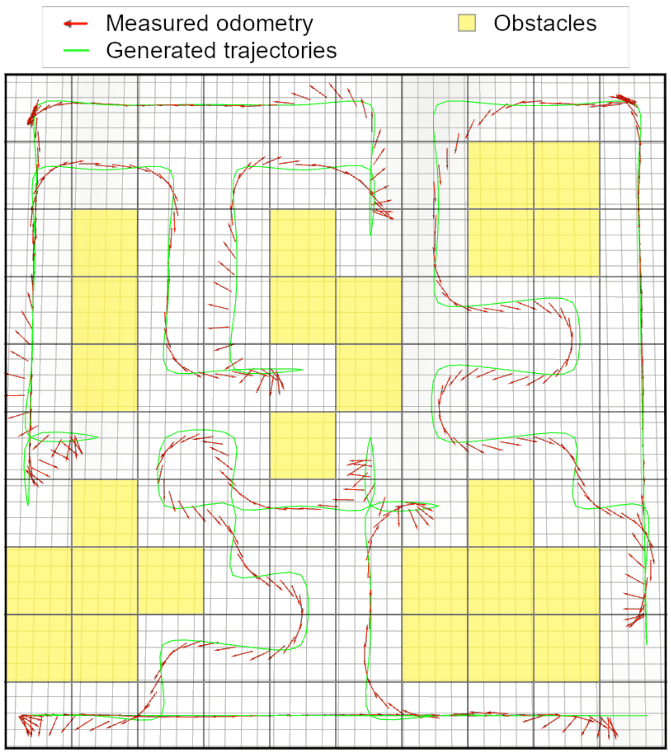


**Figure 10.4:** CGWEP Algorithm on real UAVs[5]

# Chapter 11

# Conclusions

## 11.1 Final Conclusion

In this thesis, the AWSTC algorithm from [8] was implemented and improvements were proposed to reduce the redundancy of coverage, resulting into the IAWSTC algorithm. An alternative approach, the CGWEP algorithm, was designed and compared to the IAWSTC algorithm. Quantitative tests were also carried out along with simulation and hardware experiments. The results verified that the IAWSTC algorithm produces subregions with less redundancy than the AWSTC algorithm, and also that the IAWSTC algorithm performs better than the CGWEP algorithm on the worlds where merging cannot be performed. On the other hand, on worlds where merging is not possible, the CGWEP algorithm proved to be better in terms of both length and curvature of the resulting trajectories. The implementations of both algorithms were enriched with obstacle detection for both fixed and moving obstacles and replanning. The resulting trajectories were further improved with a novelty trajectory smoothing approach. The implemented algorithms are suitable for tasks such as search and rescue missions or object localisation in cluttered environments.

## 11.2 Future Work Proposition

It would be interesting to extend the exploration algorithms to polygonal environments with polygonal obstacles. This would result in a more versatile exploration technique which could be used in more universal spaces. Additionally, an online version of the CG algorithm could be created for environments where the obstacle map is not known beforehand.

The TSULSS can be extended into three dimensions. Also, instead of using constant speed, the speed profile could be optimized across the generated trajectories. This remains an opportunity for future research.

# Appendix A

# Acronyms

|  |  |
|---|---|
| **AWSTC** | Artificially Weighted Spanning Tree Coverage |
| **CG** | Cycle Growing |
| **CGWEP** | Cycle Growing With Environment Partitioning |
| **IAWSTC** | Improved Artificially Weighted Spanning Tree Coverage |
| **ROS** | Robotic Operating System |
| **STC** | Spanning Tree Coverage |
| **TSULSS** | Trajectory Smoothing Using Least Squares Solution |
| **UAV** | Unmanned Aerial Vehicle |

# Appendix B

## Bibliography

[1] Agmon, N., Hazon, N., and Kaminka, G. Constructing Spanning Trees for Efficient Multi-robot Coverage. In *Proceedings IEEE International Conference on Robotics and Automation* (2006), pp. 1698–1703. Available at https://doi.org/10.1109/ROBOT.2006.1641951.

[2] Avellar, G. S. C., Pereira, G. A. S., Pimenta, L. C. A., and Iscold, P. Multi-UAV Routing for Area Coverage and Remote Sensing with Minimum Time. *Sensors 15*, 11 (2015), 27783–27803. Available at https://doi.org/10.3390/s151127783.

[3] Biagiotti, L., and Melchiorri, C. Online Trajectory Planning and Filtering for Robotic Applications via B-spline Smoothing Filters. In *Proceedings IEEE/RSJ International Conference on Intelligent Robots and Systems* (2013), pp. 5668–5673. Available at https://doi.org/10.1109/IROS.2013.6697177.

[4] Brezak, M., and Petrović, I. Path Smoothing Using Clothoids for Differential Drive Mobile Robots. *IFAC Proceedings Volumes 44*, 1 (2011), 1133–1138. Available at https://doi.org/10.3182/20110828-6-IT-1002.02944.

[5] Chang, S.-R. A G2 Continuous Path-smoothing Algorithm Using Modified Quadratic Polynomial Interpolation. *International Journal of Advanced Robotic Systems 11* (February 2014). Available at https://doi.org/10.5772%2F57340.

[6] Chunqing, G., Yingxin, K., Zhanwu, L., An, X., You, L., and Yizhe, C. Optimal Multirobot Coverage Path Planning: Ideal-Shaped Spanning Tree. *Mathematical Problems in Engineering* (2018), 1–10. Available at https://doi.org/10.1155/2018/3436429.

[7] Demkiv, L., Ruffo, M., Silano, G., Bednář, J., and Saska, M. An Application of Stereo Thermal Vision for Preliminary Inspection of Electrical Power Lines by MAVs. In *Aerial Robotic Systems Physically Interacting with the Environment (AIRPHARO)* (October 2021), IEEE, pp. 1–8. Available at `https://doi.org/10.1109/AIRPHARO52252.2021.9571025`.

[8] Dong, W., Liu, S., Ding, Y., Sheng, X., and Zhu, X. An Artificially Weighted Spanning Tree Coverage Algorithm for Decentralized Flying Robots. *IEEE Transactions on Automation Science and Engineering 17*, 4 (2020), 1689–1698. Available at `https://doi.org/10.1109/TASE.2020.2971324`.

[9] Faigl, J., Váňa, P., Pěnička, R., and Saska, M. Unsupervised Learning-based Flexible Framework for Surveillance Planning with Aerial Vehicles. *Journal of Field Robotics 36*, 1 (2019), 270–301. Available at `https://doi.org/10.1002/rob.21823`.

[10] Feng, K., Li, W., Ge, S., and Pan, F. Packages Delivery Based on Marker Detection for UAVs. In *Chinese Control and Decision Conference (CCDC)* (2020), pp. 2094–2099. Available at `https://doi.org/10.1109/CCDC49329.2020.9164677`.

[11] Fujii, S., and Pham, Q.-C. Realtime Trajectory Smoothing with Neural Nets. *CoRR abs/2111.02165* (2021). Available at `https://doi.org/10.48550/arXiv.2111.02165`.

[12] Gabriely, Y., and Rimon, E. Spanning-tree Based Coverage of Continuous Areas by a Mobile Robot. In *Proceedings ICRA. IEEE International Conference on Robotics and Automation (Cat. No.01CH37164)* (2001), vol. 2, pp. 1927–1933. Available at `https://doi.org/10.1109/ROBOT.2001.932890`.

[13] Hashemian, A., Hosseini, S., and Nabavi, s. N. Kinematically Smoothing Trajectories by NURBS Reparameterization – An Innovative Approach. *Advanced Robotics 31* (November 2017), 1296–1312. Available at `https://doi.org/10.1080/01691864.2017.1396923`.

[14] Hussein, A., and Elnagar, A. On Smooth and Safe Trajectory Planning in 2D Environments. In *Proceedings of International Conference on Robotics and Automation* (1997), vol. 4, pp. 3118–3123. Available at `https://doi.org/10.1109/ROBOT.1997.606762`.

[15] Jiao, Y.-S., Wang, X.-M., Chen, H., and Li, Y. Research on the Coverage Path Planning of UAVs for Polygon Areas. In *5th IEEE Conference on Industrial Electronics and Applications* (2010), pp. 1467–1472. Available at `https://doi.org/10.1109/ICIEA.2010.5514816`.

[16] Kang, Z., Ling, H., Zhu, T., and Luo, H. Coverage Flight Path Planning for Multi-rotor UAV in Convex Polygon Area. In *Chinese Control And Decision Conference (CCDC)* (2019), pp. 1930–1937. Available at `https://doi.org/10.1109/CCDC.2019.8833382`.

[17] Karapetyan, N., Benson, K., McKinney, C., Taslakian, P., and Rekleitis, I. M. Efficient Multi-Robot Coverage of a Known Environment. *CoRR abs/1808.02541* (2018). Available at `https://doi.org/10.48550/arXiv.1808.02541`.

[18] Li, X., Gao, X., Zhang, W., and Hao, L. Smooth and Collision-free Trajectory Generation in Cluttered Environments Using Cubic B-spline Form. *Mechanism and Machine Theory 169* (2022), 104606. Available at `https://doi.org/10.1016/j.mechmachtheory.2021.104606`.

[19] Lin, Y., and Saripalli, S. Path Planning Using 3D Dubins Curve for Unmanned Aerial Vehicles. In *International Conference on Unmanned Aircraft Systems (ICUAS)* (2014), pp. 296–304. Available at `https://doi.org/10.1109/ICUAS.2014.6842268`.

[20] Nedjati, A., Izbirak, G., Vizvari, B., and Arkat, J. Complete Coverage Path Planning for a Multi-UAV Response System in Post-Earthquake Assessment. *Robotics 5*, 4 (2016). Available at `https://doi.org/10.3390/robotics5040026`.

[21] Ravankar, A., Ravankar, A. A., Kobayashi, Y., and Emaru, T. SHP: Smooth Hypocycloidal Paths with Collision-Free and Decoupled Multi-Robot Path Planning. *International Journal of Advanced Robotic Systems 13*, 3 (2016), 133. Available at `https://doi.org/10.5772/63458`.

[22] Ravankar, A., Ravankar, A. A., Kobayashi, Y., Hoshino, Y., and Peng, C.-C. Path Smoothing Techniques in Robot Navigation: State-of-the-Art, Current and Future Challenges. *Sensors 18*, 9 (2018), 3170. Available at `https://doi.org/10.3390/s18093170`.

[23] Ruan, L., Wang, J., Chen, J., Xu, Y., Yang, Y., Jiang, H., Zhang, Y., and Xu, Y. Energy-efficient Multi-UAV Coverage Deployment in UAV Networks: A Game-theoretic Framework. *China Communications 15*, 10 (2018), 194–209. Available at `https://doi.org/10.1109/CC.2018.8485481`.

[24] Smrčka, D., Báča, T., Nascimento, T., and Saska, M. Admittance Force-Based UAV-Wall Stabilization and Press Exertion for Documentation and Inspection of Historical Buildings. In *International Conference on Unmanned Aircraft Systems (ICUAS)* (June 2021), IEEE, pp. 552–559. Available at `https://doi.org/10.1109/ICUAS51884.2021.9476873`.

[25] Yuan, H. A Novel Trajectory Smoothing Algorithm Based on Empirical Mode Decomposition. In *Fifth International Conference on Image and*

*Graphics* (2009), pp. 223–226. Available at https://doi.org/10.1109/ICIG.2009.75.

[26] Zhou, F., Song, B., and Tian, G. Bézier Curve Based Smooth Path Planning for Mobile Robot. *Journal of Information and Computational Science 8* (December 2011), 2441–2450. Available at https://www.researchgate.net/publication/285739464_Bezier_curve_based_smooth_path_planning_for_mobile_robot.

[27] Zhu, Z., Schmerling, E., and Pavone, M. A Convex Optimization Approach to Smooth Trajectories for Motion Planning with Car-Like Robots. *CoRR abs/1506.01085* (2015). Available at https://doi.org/10.48550/arXiv.1506.01085.

# BACHELOR‘S THESIS ASSIGNMENT

## I. Personal and study details

| | | | |
|---|---|---|---|
| Student's name: | **Chleboun Jan** | Personal ID number: | **492323** |
| Faculty / Institute: | **Faculty of Electrical Engineering** | | |
| Department / Institute: | **Department of Cybernetics** | | |
| Study program: | **Cybernetics and Robotics** | | |

## II. Bachelor’s thesis details

Bachelor’s thesis title in English:

**Spanning Tree Coverage Algorithm on Large Spaces for Multi-UAV Systems**

Bachelor’s thesis title in Czech:

**Algoritmus pro průzkum velkých prostorů pomocí skupiny kooperujících dronů**

Guidelines:

Improve algorithm [1] by optimization on the cell size distribution and the path planning covering a single cel per run, decreasing the optimization and computational costs. For avoiding sharp turns, propose an adequate smoothing. The final implementation must experimentally verified on a real robot and compared with exisiting approaches.
Furthermore, the student must consider the following modifications and tasks:
1. The student must consider fixed and moving obstacles.
2. Experiments with real robots must be performed.
3. An optimization smoother for the planned path must applied.
4. Comparisons with state-of-the-art algorithm (reference [1]) must be performed.

Bibliography / sources:

[1] Dong, W., Liu, S., Ding, Y., Sheng, X., \& Zhu, X. (2020). An Artificially Weighted Spanning Tree Coverage Algorithm for Decentralized Flying Robots. IEEE Transactions on Automation Science and Engineering, 2020
[2] N. Agmon, N. Hazon and G. A. Kaminka, "Constructing spanning trees for efficient multi-robot coverage," Proceedings 2006 IEEE International Conference on Robotics and Automation, 2006

Name and workplace of bachelor’s thesis supervisor:

**Tiago Pereira Do Nascimento, Ph.D. Multi-robot Systems FEE**

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

Date of bachelor’s thesis assignment: **21.01.2022** Deadline for bachelor thesis submission: **20.05.2022**

Assignment valid until: **30.09.2023**

| | | |
|---|---|---|
| Tiago Pereira Do Nascimento, Ph.D. | prof. Ing. Tomáš Svoboda, Ph.D. | prof. Mgr. Petr Páta, Ph.D. |
| Supervisor’s signature | Head of department’s signature | Dean’s signature |

## III. Assignment receipt

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

.

| | |
|---|---|
| Date of assignment receipt | Student’s signature |