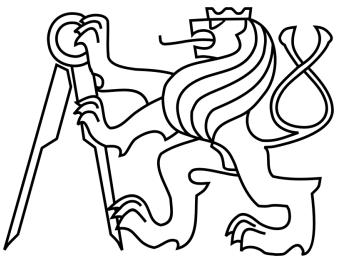
Czech Technical University in Prague
Faculty of Electrical Engineering
Department of Cybernetics



BACHELOR THESIS

RRT-path method used for cooperative surveillance by group of helicopters

Author: Matěj Račinský

Thesis supervisor: Dr. Martin Saska

In Prague, May 2016

Author statement for the undergraduate thesis:

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

Prague, date _____ _____

signature

Název práce: Aplikace algoritmu RRT-path v úloze autonomního dohledu skupinou helikoptér

Autor: Matěj Račinský

Katedra (ústav): Katedra kybernetiky

Vedoucí bakalářské práce: Dr. Martin Saska

e-mail vedoucího: saska@labe.felk.cvut.cz

Abstrakt Tato práce se zabývá plánováním trasy roje složeného z bezpilotních helikoptér v úloze autonomního dohledu. Popisuji zde principy a implementaci algoritmu pro plánování trasy roje bezpilotního helikoptér za použití RRT-Path algoritmu. Algoritmus popsán v této práci kombinuje RRT-Path algoritmus s optimalizací pomocí Dubinsových křivek. Tato práce zahrnuje implementaci algoritmu v jazycích C++ a Python. Funkce vyvinutého systému byla ověřena experimenty, které jsou prezentovány v této práci.

Klíčová slova: RRT, RRT-Path, Dubins curves, UAV, swarm

Title: RRT-path method used for cooperative surveillance by group of helicopters

Author: Matěj Račinský

Department: Department of Cybernetics

Supervisor: Dr. Martin Saska

Supervisor's e-mail address: saska@labe.felk.cvut.cz

Abstract In this thesis we study deployment of a swarm consisting of Unmanned Aerial Vehicles (UAVs) in the task of autonomous surveillance using motion planning. This thesis describes principles and implementation of the algorithm for motion planing of swarm of UAVs using the RRT-Path algorithm. Algorithm described in this thesis combines the RRT-Path algorithm with optimization by Dubins Curves. The thesis includes the implementation in C++ and Python, functionality was verified by experiments presented in this thesis.

Keywords: RRT, RRT-Path, Dubins curves, UAV, swarm

CONTENTS

CHAPTER

ONE

INTRODUCTION

Unmanned Aerial Vehicle (UAV) is an aircraft intended to operate with no pilot on-board. The UAV is also known as drone and it has been gaining popularity in recent years in both academic circles and wide public. The term Micro Aerial Vehicle (MAV) is also used for very small UAV, typically intended for multi-robot scenarios. Drones are used in many applications, both military and civil, but unfortunately, the term "drone" has negative connotation and is mainly linked to military actions. The main complication of massive industrial use of drones is national regulation and legislation.

The drone may be controlled with various kinds of autonomy: either by a given degree of remote control from an operator, located on the ground or in another vehicle, or fully autonomously, by onboard computers [10].

For small drones, the quadrotor (also called quadcopter) design has become very popular and widely used. Quadrotor is a helicopter that is lifted and propelled by four horizontal rotors. Quadrotors are used in the Multi-Robot Systems group at CTU, which uses them in various multi-robot scenarios, such as formation flying [14, 15, 17, 18, 21], swarm robotics [13, 20], environment monitoring [19] and autonomous surveillance [16, 23], which is being solved also in this thesis. Example of a swarm of quadrotors can be seen in figure 1.1.

International Civil Aviation Organization (ICAO) [10] classifies UAVs into two groups: Remotely piloted aircraft - an aircraft where the pilot is not on board of the aircraft and control the aircraft from another location, and Autonomous aircraft - an unmanned aircraft that does not allow pilot intervention in the management of the flight.

Remotely piloted aircraft are teleoperated usually by a person with a remote radio controller. Some examples of uses of teleoperated aircraft are inspections of power lines [5], monitoring of agricultural areas [6], with proof of concept demonstrated in Château de Châtagneréaz vineyard [1], filming movies and acrobatic aerial footages [6], counting wildlife and searching people lost in wilderness [11] and crowd monitoring of large events including festivals and protests. UAV monitoring of crowds allow operators to see different parts of the surveyed scene, follow crowd/people and thus provide information that is not accessible with fixed infrastructure of immovable cameras. UAVs are handy in many use cases, where helicopters with human pilots are too costly or unwieldy.

Autonomous aircraft fly independently without an operator directly controlling the flight. The operator specifies a task for the aircraft to execute instead. The task difficulty depends on the level of autonomy of the aircraft. It can be anything from flying straight to a single point or

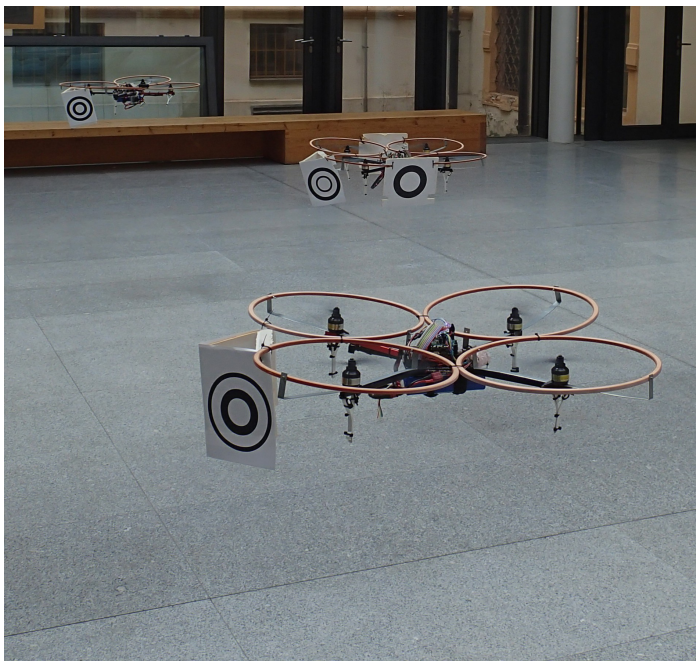


Figure 1.1.: Quadrotors assembled in the Multi-Robot Systems group at CTU

following a path consisting of multiple points, to perform a complex task of following a moving object and avoiding collisions.

Both start-up companies and large corporates compete in developing various kinds of autonomous aircraft for many different purposes. Currently there are 368 projects on kickstarter tinkering with drones. The Amazon plans to use autonomous drones for packages delivery in their program Prime Air [26]. Next to delivering goods, the other commonly discussed topic is localization of people or other objects in rescue missions taking place in dangerous or hard to access areas and autonomous surveillance of Areas of Interest (abbreviated as AoI).

Miniaturization, more manufacturers and lowering costs of UAVs and nature inspired algorithms led to the idea of creating UAV swarms. Vito Trianni [24] defines four criteria for robotic system to be considered a swarm robotic system.

1. The system should be relevant for the coordination and control of a large number of robots. This includes all approaches that aim for scalability.

2. The system should involve relatively few groups of homogeneous robots, each group comprising a large number of individuals, high redundancy is required within each group.

3. The system should consider tasks that can not be effectively solved by the single robot, due to individual limitations.

4. The system should involve robots that have local and limited sensing and communication abilities.

The approach presented in this thesis is not scalable, but it satisfies the other criteria. Swarm cannot be remotely controlled by one person and controlling each UAV in swarm by one operator would be really hard to coordinate and practically impossible to avoid collisions between UAVs,

so the only plausible way to manually control the swarm is to remotely control whole swarm as one drone. This thesis deals exclusively with fully autonomous aircraft where no UAV needs to be remotely controlled by an operator.

In the problem of controlling of swarm, collisions between UAVs must be avoided. That means that a method to keep them in a safe distance between each other has to be found. Second, a precise enough relative localization system is needed to keep track of the swarm formation shape and relative distances. And finally, a motion planning algorithm has to be developed in order to move the swarm from its initial position to a target position (Area of Interest in case of the surveillance scenario) in a certain environment which can contain a set of obstacles [7]. Keeping track of the position of UAV individuals in the swarm is essential to prevent collisions between them and keeping the swarm organized. The most obvious approach would be to equip each UAV with a Global Positioning System (GPS) chip to obtain absolute positions and use them to calculate relative distances between each other. However, UAVs are often deployed in areas where GPS performs poorly or is impossible to use (e.g. inside buildings), moreover, even in open spaces with a good GPS signal the accuracy offered by GPS (around 3 m) is not sufficient for control of compact swarms with possibly smaller relative distances.[12]

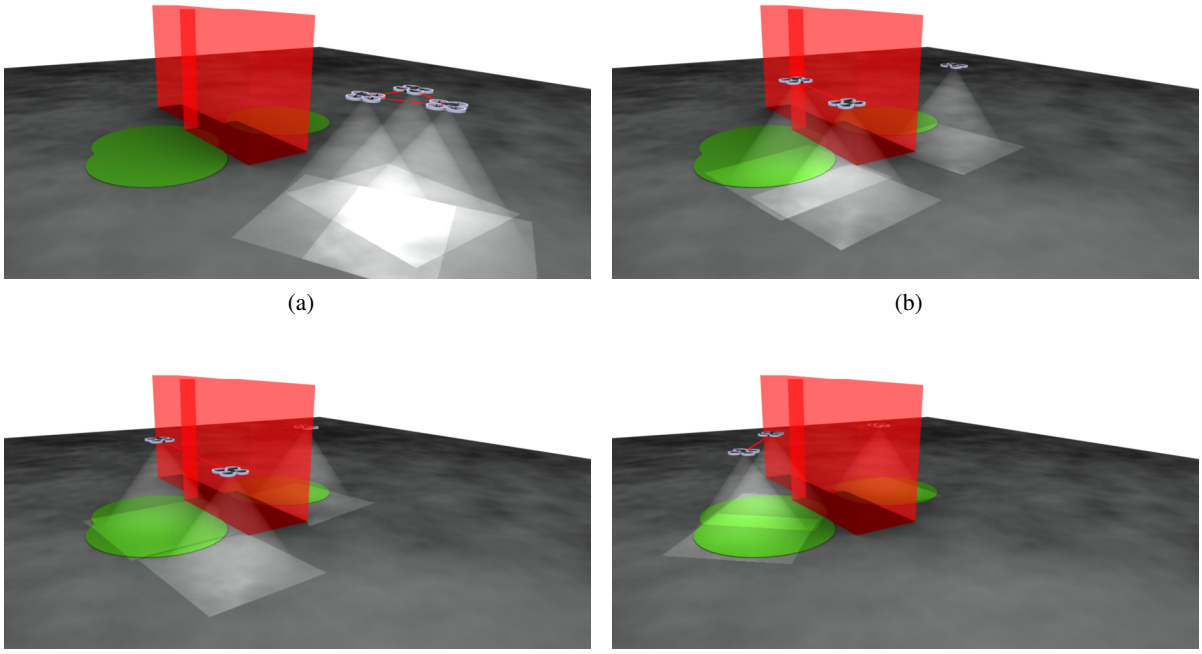
1.1. Objective

The objective of this thesis is to explore the possibility of using swarms of UAVs in the task of autonomous surveillance. The task is defined as finding a collision free trajectories from initial depot to an AoI or more AoIs and covering the largest possible part of AoI or AoIs. For details on the problem definition see [23].

This task can be done using several approaches. One approach is finding an optimal position in AoI for each individual UAV and then searching feasible and collision-free trajectory for each UAV. This approach has the problem that the feasible solution may not exist due to obstacles and relative localization constrains, as shown in [23]. A demonstration of such problem can be seen in the figure 1.2. Another approach is to find trajectory to any points above AoIs and then to optimize AoI coverage.

This thesis presents implementation of finding feasible trajectories using the RRT-Path algorithm [27–29] and optimization by Dubins curves [3]. The RRT-Path algorithm is based on the Rapidly-exploring Random Tree algorithm (RRT), described in [8]. To guarantee proper function of the RRT-Path algorithm for multi-UAV deployment, many support algorithms and systems must be implemented. The motion model of UAVs is implemented and collision avoidance algorithm is used for obstacles avoidance. In the next step, trajectories found by the RRT-Path algorithm are optimized using the Dubins curves, which provide optimal trajectory for motion model used in this thesis.

Part of this thesis is also verification of the planning algorithm by following trajectories by UAVs swarm in simulation software V-REP. The last part of this thesis is to design an interface with system [22], where trajectories can be loaded to real UAVs for experiments.



(a) (b)

(c) (d)

Figure 1.2.: Violation of the relative localization constraints when trajectories are planned after searching the optimal positions. Relative localization constraints are violated in b), c), d). Obstacles are visualized by red colour, AoIs by green colour. Source [23]

CHAPTER
TWO

ALGORITHM

The basic structure of the whole algorithm is shown in the figure 2.1.

The first module represents finding the guiding paths for the trajectory planning. It takes map as an input and returns the optimal path from initial positions to AoIs. The path planning uses an A* algorithm. All details regarding the path planning are explained in section 3.3. The trajectory planning consists of three sub-modules. The first sub-module is the RRT-Path trajectory planning which takes the map and the guiding path as the inputs and returns space-filling tree with some leaves in the AoIs. The next sub-module is RRT which continues to search the space above AoIs. All feasible trajectories are used as an input of the sub-module for finding the best coverage. This sub-module computes coverage of AoIs in each configuration and returns the trajectory to configuration with the best AoIs coverage. The quality of the coverage is determined by the cost function, which is described in chapter 5. The trajectories re-sampling module re-samples trajectories to higher sampling rate, which results in smoother trajectories. This module is here mainly due to constraints of real UAVs, which are mentioned in chapter 8. The last module is optimization of trajectories by Dubins curves. The optimization is explained in chapter 7.

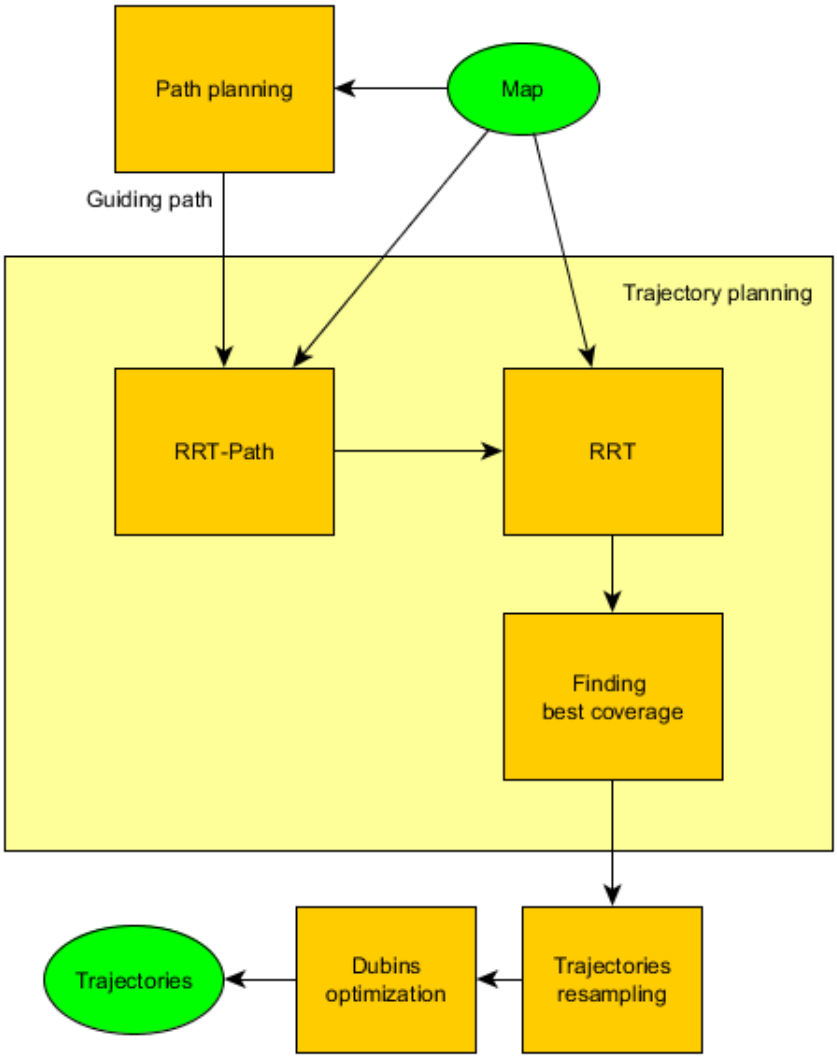


Figure 2.1.: The basis of whole algorithm

CHAPTER
THREE

RRT-PATH

In this chapter a brief introduction of the RRT-Path algorithm is covered. Firstly, we need to define the RRT algorithm which the RRT-Path is based on.

3.1. Rapidly Exploring Random Tree

Rapidly Exploring Random Tree (RRT), introduced by LaValle [8] in 1998, is non-deterministic algorithm for motion planning, used to search non-convex spaces by randomly-built space-filling tree. The RRT method builds a tree T rooted at q_{start}. In each iteration, a random sample q_{rand} is chosen from the configuration space C and the nearest node q_{near} in the tree to q_{rand} is found. The node q_{near} is expanded using a local planner to obtain a set of new configurations reachable from q_{near}. The nearest configuration towards q_{rand} is selected from this set and added to the tree. The edge from q_{near} to the newly added configuration contains control inputs used by the local planner to reach the new configuration. The algorithm terminates if the distance between a node in the tree and q_{goal} is less than d_{goal} or after I_{max} of planning iterations. [29] The RRT method is sketched in pseudocode 3.1. In the RRT algorithm, configurations on the third line have uniform distribution. The implementation of the expansion step is sketched in pseudocode 3.2. The expansion step includes motion planning with the use of a motion model. Forward motion model $\dot{q} = f\ (q, u)$is considered there and the expansion of node q is realized by applying several control inputs $u \in U$ to the model in order to obtain new configurations reachable from the node. The control inputs are applied over time Δt. New configurations are obtained by integration of the motion model, which can be solved analytically in the case of simple systems like Car-like robots [21], or using numerical integration like Euler integration of Runge-Kutta methods for complex systems. As UAVs can be driven by continuous values, the set U has to be discretized in order to allow RRT to expand the node $q_{near}$to a reasonable number of candidate configurations. [29]

3.2. RRT-Path

RRT-Path, introduced by Vonásek [27, 29] in 2015, is an improved version of RRT featuring preprocessing of configuration space. RRT-Path enables UAVs to manoeuvre around obstacles and find way in narrow passages. RRT-Path also finds goal much faster [12]. RRT-Path uses the

Algorithm 3.1 the RRT algorithm source [29]

Input: Configurations q_{alert} and q_{goal}, maximum number of iterations I_{max}, maximum distance to goal d_{goal}, configuration space C

Output: Trajectory P or failure

```
T.add (q_start) // create new tree and add initial configuration q in it
for iteration :=1:I_max do
    q_rand := getRandomConfiguration(C)
    q_near := nearest node in tree T to q
    expandTree(q_rand,q_near)
    d = distance from tree T to q_goal
    if d < d_goal then
        P = extract trajectory from q_start to q_rand
        return P
    end
end
return failure // no solution was found within I_max iterations
```

Algorithm 3.2 expandTree(q_{rand}, q_{near}): Expansion procedure of the RRT algorithm source [29]

Input: Random configuration $q_{rand} \in C$, configuration tree T, its nearest node in the tree $q_{near} \in T$

Output: Extended tree T

```
R = ∅ // set of configurations reachable from q_near together with control inputs
foreach u∈U do
    q = q_near + ∫_0^Δt f (q_near, u) dt
    if q is feasible then
        R = R ∪ {(q, u)}
    end
end
if R ≠ ∅ then
    (q_new, u) = select a conguration from R closest to q near ;
    T.addNode(q_new)
    T.addEdge(q_near, q_new, Δt, u)
end
```

guiding path during building the space-filling tree. Before running the RRT algorithm, the guiding path from q_{start} to q_{goal} is found and sampled. One of inputs to the RRT-Path algorithm is the probability $p\,(guided)$. In the main loop of the algorithm, obtaining of the random configuration is modified. Instead of random configuration with uniform distribution, configuration around the $q_i$is selected with probability $p\,(guided)$.

Let G be the guiding path and $\left(q_{start}, q_1, q_2, ..., q_{goal}\right) \in G$ the points of the guiding path, where $q_i \in C_{free}$ and $i \in (start, 1, 2, ..., goal)$. In the beginning, $q_i := q_1$, so random point is selected from the area around the point q_1 with probability $p\,(guided)$. When the leaves of the searching tree reach distance lower than $r_{dist}$to the q_i, then the next point of the guiding path will be used instead of q_i, so $q_i := q_{i+1}$. This continues until q_{goal} is reached, which means the RRT-Path algorithm ends.

3.3. Guiding path

The guiding path is obtained by transferring the map to a graph representation and then the path is found using the graph-search algorithms. The map can be transferred to the graph representation by using the Voronoi diagram, a visibility graph or by discretization to a grid representation.

Then the path can be found by using Dijkstra algorithm or A* algorithm. In this thesis, the A* algorithm has been used because of its ability to find optimal path and easy calculation of heuristic function in Euclidean space.

The classic cost function of node q_i in A* algorithm is $f\,(q_i) = g\,(q_i) + h\,(q_i)$. The $g\,(q_i)$ is sum of costs of all edges in shortest path between nodes and q_{start} and q_i. The $h\,(q_i)$ is heuristic estimate of distance between q_i and q_{goal}. In Euclidean space, it is calculated as $h\,(q_i) = \left\|q_i - q_{goal}\right\|$. In this thesis, a node q_i in the graph has a function $o\,(q_i)$ representing its proximity to the nearest obstacle. This function expresses obstacles avoidance. $o\,(\cdot)$ function used in this thesis is following $o\,(q_i) = 30 s_{direct}\,(q_i) + 15 s_{diagonal}\,(q_i)$ where $s_{direct}\,(q_i)$ is count of obstacles in direct neighbours of the node q_i and $s_{diagonal}\,(q_i)$ is count of obstacles in diagonal neighbours of the node q_i. Another example of such function is $o\,(q_i) = \frac{const}{\|q_i - nearest\,obstacle\|}$, where $const$ is weight of the $o\,(q_i)$ and determines how much obstacles should be avoided. The map with nodes evaluation can be seen in the figure 3.1. For every two nodes $q_i, q_j \in C_{free}$, the shortest path $p_{min}\left(q_i, q_j\right)$ of all paths is used as metric for the map. The length of path $p\left(q_{start}, q_{goal}\right)$ in graph is calculated as

$$length\left(p\left(q_{start}, q_{goal}\right)\right) = \sum_{q_i = q_{start}}^{q_{goal-1}} d\,(q_i, q_{i+1}) + \sum_{q_i = q_{start+1}}^{q_{goal-1}} o\,(q_i) \tag{3.1}$$

, where the $d\,(\cdot)$ is euclidean distance between neighbouring nodes. $o\,(q_{start})$ and $o\left(q_{goal}\right)$ are not calculated because the metric needs to satisfy the condition $d(q_i, q_j) = 0 \Leftrightarrow q_i = q_j$. Then all conditions for the metric are satisfied. Because the $o\,(\cdot)$ is considered only in metric calculation and not in the heuristic estimate, the heuristic is admissible and thus it finds the optimal path.

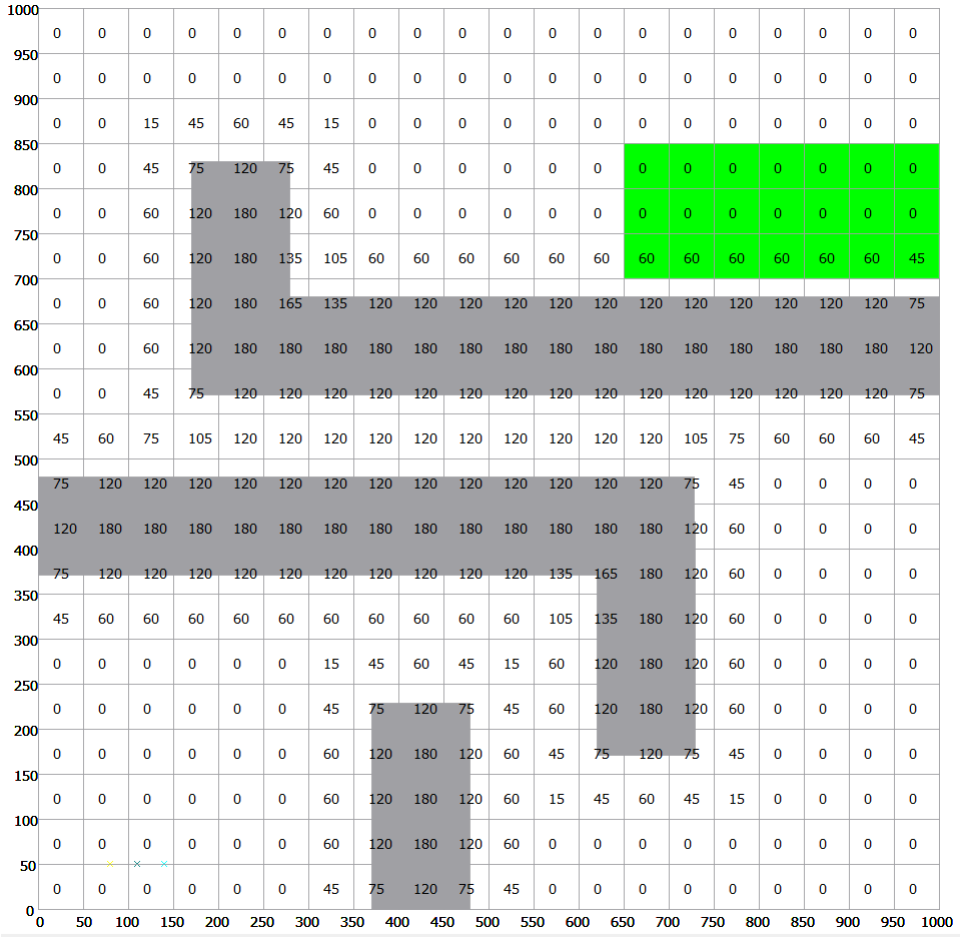


Figure 3.1.: Example of the a map with nodes evaluated by $o(\cdot)$ function.

CHAPTER
FOUR

GROUPING OF GOALS FOR THE GUIDING PATH

During the processing of the map (method MapProcessor::getEndNodes in codebase), all AoIs are grouped to one big AoI, which is the smallest rectangle covering all AoIs.

If this modification is enabled, only one goal is used for all AoIs instead of one goal for every AoI (a node in the middle of AoI rectangle is considered to be the goal node). The whole swarm has only one guiding path, so the grouping prevents the swarm from splitting. The relative localization is the main reason to have only one big swarm instead of more smaller swarms (or individual UAVs in case of the same count of AoI and UAVs), due to increased stability and reduced possibility of relative collisions as discussed in [23]. When obstacle is in the middle, nearest node which is not occupied by an obstacle is used as middle of goals group. The middle is used as the target for the guiding path.

This approach has the following advantage. When individual AoIs are near to a global goal of the whole group, as seen in 4.2, then the whole swarm follows one guiding path without any splitting. The grouping makes the RRT-Path run faster and also the advantage of relative localization is included.

Maps with goals and obstacles are shown in figures 4.1 and 4.2.

The disadvantage of this method emerges when individual AoIs have a bigger distance from each other than can be covered by UAVs, that are keeping the relative distances required by the relative localization constraints specified in [4]. Then this approach may fail, because the goal of the RRT-Path is very distant from AoIs, as can be seen in figure 4.1 and the main part of the path is found by RRT algorithm. The RRT-Path is able to find the trajectories much faster than the RRT algorithm and thus finding trajectories by the RRT algorithm is much slower.

Nevertheless such distribution of AoIs can be simply detected by measuring distances between the rectangle centre and the AoIs and the swarm can be split since all sub-swarms will operate in different parts of the environment and therefore in safe relative distances. The proposal of detection is mentioned in section 13.1. The method proposed in this thesis can be used independently for all sub-swarms. This thesis is not focused on the distribution of AoIs to independent swarms and therefore all situations presented here enable feasible grouping of AoIs for a single swarm.

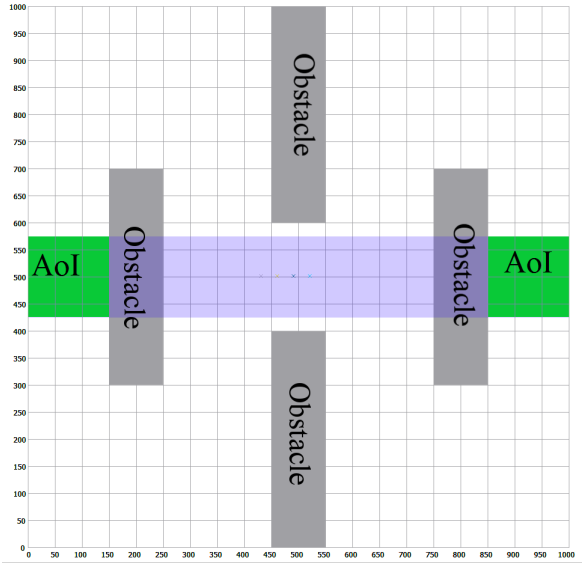


Figure 4.1.: A map with the goals unsuitable for grouping. Group is marked with blue rectangle. Obstacles have grey colour and AoIs have green colour.

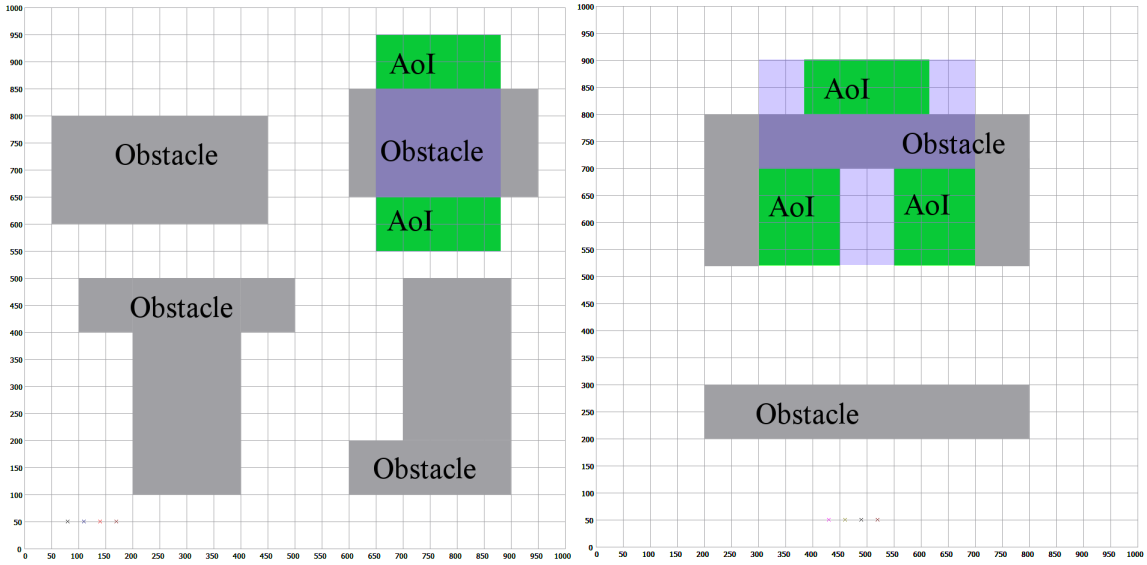


Figure 4.2.: Maps with the goals suitable for grouping. Groups are marked with blue rectangles. Obstacles have grey colout and AoIs have green colour.

CHAPTER
FIVE

AREAS OF INTEREST COVERAGE

Covering Areas of Interest (AoIs) with UAVs is the key part of the task of autonomous cooperative surveillance. In task of surveillance, UAVs observe space below them by on-board camera, as shown in figure 5.1. AoIs and areas seen by UAVs are represented by rectangles in this thesis for simplicity and fast computation.

Coverage of AoIs is an optimization problem. This optimization problem can be solved by finding minimum of cost function. The value of cost function should objectively reflect the quality of coverage. The lower value should represent better coverage. In this thesis, cost function $f(q_i)$ represents quality of coverage in configuration q_i, where configuration q_i is set of UAV positions. The cost function $f(\cdot)$represents size of AoIs not covered by UAVs, which is equivalent to information not seen by UAVs. That means if $f(q_1) < f(q_2)$ configuration q_1 covers bigger part of AoIs than configuration q_2. The environment is discretized to square grid represented by matrix $A \in R^2$. One of parameters for coverage optimization is size of one square of grid a $[map\,units]$. Each element $A_{j,k} \in R$ represents square with size a. Before computing areas seen by UAVs, $A_{j,k} := A_{max}$ if it contains AoI, and $A_{j,k} := 0$ otherwise. $A_{max} = 100$ is used in experiments, but any value which does not cause overflow or underflow in float representation will suffice. Then the following formula is applied to every element of the world representing matrix A:

$$A_{j,k} := A_{j,k} \cdot l^m, \tag{5.1}$$

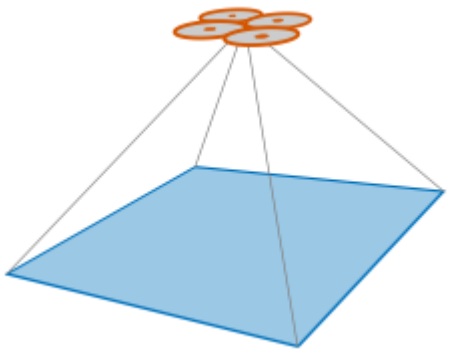


Figure 5.1.: The area viewed by UAV on-board camera, source [12]

where variable l should indicate amount of information not seen by UAV even if this UAV covers the element $A_{j,k}$. This is because quality of image obtained from on-board camera depends on many factors, such as time of day, weather conditions, flight altitude, camera chip resolution, lens parameters, stabilization, frame rate and so on. Due to these factors, it is convenient to let more UAVs observe same area. In this implementation, $l := 0.5$, but if the flight altitude will be considered in the optimization algorithm, higher flight altitude would lead worse image recording, so $l(flightAltitude_1) > l\,(flightAltitude_2)$ for $flightAltitude_1 > flightAltitude_2$. Variable m represents number of UAVs seeing the area of element $A_{j,k}$. Example of AoI with one UAV observing part of it can be seen in figure 5.2. As we can see, parts of AoI not seen by a UAV have value $A_{j,k} = A_{max} = 100$ and parts of AoI seen by UAV have value $A_{j,k} = A_{max} \cdot l^m = 100 \cdot 0.5^1 = 50$. In some applications, one UAV may observe everything in its observable area and does not need another UAV observing same area. In this case, $l := 0$, so the cost of the element $A_{j,k}$ will not change when more UAV will observe it, because 0^m is same for every $m \in N$. The l^m where $1 > l > 0$ and $m > 1$ evaluates the configuration where more UAVs observe the same area as better than one UAV observing the area, but worse than each UAV observing its own area.

If UAV arrives above any AoI, the guiding path is not used for this UAV anymore and instead of it, RRT algorithm continues by random searching over the AoI. Random node for RRT algorithm is selected only from AoI beneath the UAV, not from whole environment. The RRT algorithm is stopped when maximum number of nodes is reached. After stopping, all states found by RRT algorithm where UAVs are above AoIs are evaluated by the coverage cost function. Every node of the RRT algorithm contains configurations of all UAVs and during the RRT, but every UAV has its own random point selected during the RRT. During the RRT algorithm, the feasibility and relative localization constraints are being checked. State with the lowest cost function is used as result of the path finding algorithm and used as input for Dubins curves optimization.

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	100	100	100	100	100	100	100	100	100	0	0	0	0	0	0
0	0	0	0	100	100	100	100	100	100	100	100	100	0	0	0	0	0	0
0	0	0	0	50	50	50	50	50	50	100	100	100	0	0	0	0	0	0
0	0	0	0	50	50	50	50	50	50	100	100	100	0	0	0	0	0	0
0	0	0	0	50	50	50	50	50	50	100	100	100	0	0	0	0	0	0
0	0	0	0	50	50	50	50	50	50	100	100	100	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

(a) 1 UAV

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	100	100	100	100	100	100	100	100	100	0	0	0	0	0	0
0	0	0	0	100	100	50	50	50	50	50	50	50	0	0	0	0	0	0
0	0	0	0	50	50	25	25	25	25	50	50	50	0	0	0	0	0	0
0	0	0	0	50	50	25	25	25	25	50	50	50	0	0	0	0	0	0
0	0	0	0	50	50	25	25	25	25	50	50	50	0	0	0	0	0	0
0	0	0	0	50	50	25	25	25	25	50	50	50	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

(b) 2 UAVs

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	50	50	50	50	50	50	50	100	100	0	0	0	0	0	0
0	0	0	0	50	50	25	25	25	25	25	50	50	0	0	0	0	0	0
0	0	0	0	25	25	12.5	12.5	12.5	12.5	25	50	50	0	0	0	0	0	0
0	0	0	0	25	25	12.5	12.5	12.5	12.5	25	50	50	0	0	0	0	0	0
0	0	0	0	50	50	25	25	25	25	50	50	50	0	0	0	0	0	0
0	0	0	0	50	50	25	25	25	25	50	50	50	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

(c) 3 UAVs

Figure 5.2.: AoI matrix with one UAVs above it represented by matrix with every cell evaluated by formula 5.1. The AoI is marked with green colour and the areas observed by the UAVs is marked with blue colour. Area covered by more UAVs has smaller value as implies the formula.

CHAPTER

SIX

UAV SWARM PROPERTIES

6.1. Motion model

The RRT-Path algorithm is universal and works with any motion model, which allows us to find paths feasible for swarm of UAVs. The motion model in RRT-Path is important for obtaining configurations of UAVs in the next state. In order to obtain smooth trajectories that are easy to follow, the trajectory planning in space of circles and lines is chosen. For this purpose, car-like model was chosen. Differential equations of motion model in 3D from [21] are

$$\begin{aligned} \dot{x}(t) &= v(t)\sin\varphi(t) \\ \dot{y}(t) &= v(t)\cos(t) \\ \dot{z}(t) &= w(t) \\ \dot{\varphi}(t) &= K(t)\,v(t) \end{aligned} \tag{6.1}$$

where $x(t), y(t), z(t)$ are coordinates of UAV, $\varphi(t)$ represents heading of UAV, $v(t)$ is forward velocity, $K(t)$ is curvature, $w(t)$ is ascent velocity. Vector $\begin{bmatrix} K(t) & w(t) & v(t) \end{bmatrix}$ represents the input vector of motion model. Difference equations are used for calculation of next state in RRT-Path and RRT algorithms. When inputs are held constant in each time interval between two time steps, difference equations are

$$\begin{aligned} x(k+1) &= \begin{cases} x(k) + \frac{1}{K(k+1)}\left(\sin\left(\varphi(k) + K(k+1)\,v(k+1)\,\Delta t(k+1)\right) - \sin(\varphi(k))\right) \\ if\ K(k+1) \neq 0 \\ x(k) + v(k+1)\cos(\varphi(k))\,\Delta t(k+1) \\ if\ K(k+1) = 0 \end{cases} \\ y(k+1) &= \begin{cases} y(k) - \frac{1}{K(k+1)}\left(\cos\left(\varphi(k) + K(k+1)\,v(k+1)\,\Delta t(k+1)\right) - \cos(\varphi(k))\right) \\ if\ K(k+1) \neq 0 \\ y(k) + v(k+1)\sin(\varphi(k))\,\Delta t(k+1) \\ if\ K(k+1) = 0 \end{cases} \\ z(k+1) &= z(k) + w(k+1)\,\Delta t(k+1) \\ \varphi(k+1) &= \varphi(k) + K(k+1)\,v(k+1)\,\Delta t(k+1) \end{aligned} \tag{6.2}$$

6.2. Relative localization

In compact swarms, every UAV has to be aware of its neighbours in order to remain together in one swarm and reduce the possibility of collision. In this thesis, constraints of relative localization are implemented by setting the minimal distance d_{min} and maximal distance d_{max}. The minimal distance is set to avoid collisions and because UAVs push air beneath them when they fly. Large amount of air needs to be pushed below UAV in order to keep it flying. Because of this, UAVs can not fly too close to each other because of air currents they produce. UAVs in minimal distance do not affect each other by air currents. By real experiments, the minimal distance was identified as 2 meters. The maximal distance needs to be set to respect the sensors range. UAVs in the Multi-Robot Systems group at CTU use on-board vision based localization system, published in [4]. In [4] a maximal range of sensors is specified based on pattern size and camera resolution and UAVs in bigger distance than this maximal range can not be seen. In experiments in this thesis, the maximal distance was assumed to be 5 meters.

Each UAV must have n or more neighbours in distance d, where $d_{min} < d < d_{max}$ in each step of the motion planning. Default setting $n = 2$ was used in presented experiments. Computation of this relative localization constraint is fast, but this constraint is unusable for more than 5 UAVs. Swarm of 6 or more UAVs can split to 2 or more groups and still fit this relative localization constraint. The following algorithm, which consists of two steps and uses graph representation, is proposed to check whether all UAVs are in the same swarm. Every UAV is considered as a node and every pair of UAVs with distance d, where $d_{min} < d < d_{max}$, are connected by an edge. If the graph has only one connected component, all UAVs are in one swarm. Otherwise, the graph has more connected components, which implies UAVs are split into more swarms.

In the first step of the algorithm, the adjacency matrix A is built. Each UAV represents one column and one row in this matrix. The adjacency matrix A is built as

$$A_{i,j} = \begin{cases} 1 & if\ d_{min} < d_{i,j} < d_{max} \\ 0 & otherwise \end{cases} \tag{6.3}$$

where i, j are indices of matrix A and $d_{i,j}$is distance between i-th and j-th UAV. In the second step, the graph represented by A is traversed by depth-first algorithm, starting at $A_{1,1}$. If all nodes are visited during the traversing, the graph has only one connected component and all UAVs are in one swarm. The swarming method can be enabled or disabled in configuration.

CHAPTER
SEVEN

DUBINS CURVES

Dubins curves, also called Dubins manoeuvrers or Dubins path, were published by Lester Eli Dubins in 1957 [3]. Length of Dubins path is optimal path for car-like motion model and can be efficiently applied also for control UAVs if the robot moves at constant forward speed. The important constraint is the maximum steering angle ϕ_{max}, which results in a minimum turning radiusρ_{min}. As the car travels, consider the length of the curve in $\mathcal{W} = \mathbb{R}^2$ traced out by a pencil attached to the centre of the car. The task is to minimize the length of this curve as the car travels between any q_I and q_G. Due to ρ_{min}, this can be considered as a bounded-curvature shortest-path problem. If $\rho_{min} = 0$, then there is no curvature bound, and the shortest path follows a straight line in $\mathbb{R}^2$. In terms of a cost function, the criterion to optimize is

$$L(\tilde{q}(t), \tilde{u}(t)) = \int_0^{t_F} \sqrt{\dot{x}(t)^2 + \dot{y}(t)^2} dt, \tag{7.1}$$

where t_F is the time at which q_G is reached, and a configuration is denoted as $q(t) = (x(t), y(t), \varphi(t))$, $\tilde{q}(t)$ denotes the function $\tilde{q} : [0, t] \rightarrow X$, which is called the state trajectory (or state history). Similarly, $\tilde{u}(t)$ denotes the action trajectory (or action history). If q_G is not reached, then it is assumed that $L(\tilde{q}, \tilde{u}) = \infty$. [9]

When considering constraints of inputs (actions) for motion model, the system can be simplified to

$$\begin{aligned} \dot{x}(t) &= \cos\varphi(t) \\ \dot{y}(t) &= \sin\varphi(t) \\ \dot{\varphi}(t) &= u(t) \end{aligned} \tag{7.2}$$

in which u is chosen from the interval $U = \{-\tan\phi_{max}, 0, \tan\phi_{max}\}$. As we can see, the simplified system is identical to equations 6.1 with $v = 1$.

It was shown in [3] that between any two configurations, the shortest path for the Dubins car can always be expressed as a combination of no more than three motion primitives. Each motion primitive applies a constant action over an interval of time. This interval of time is not constant and it may differ for each primitive during the path. Furthermore, the only actions that are needed to traverse the shortest paths are $u \in \{-\tan\phi_{max}, 0, \tan\phi_{max}\}$. The primitives and their associated symbols are shown in table 7.1. The S primitive drives the car straight ahead. The L and R primitives turn as sharply as possible to the left and right, respectively. Using these symbols, each possible kind of shortest path can be determined as a sequence of three symbols

Symbol	Steering u
L	$-\tan \phi_{max}$
S	0
R	$\tan \phi_{max}$

Table 7.1.: The three motion primitives from which all optimal curves for the Dubins car can be constructed.

in the order in which the primitives are applied. Let such a sequence be called a word. There is no need to have two consecutive primitives of the same kind because they can be merged into one. Under this observation, ten possible words of length three are possible. Dubins showed that only these six words are possibly optimal:

$$\{LRL,\ RLR,\ LSL,\ LSR,\ RSL,\ RSR\}. \tag{7.3}$$

The shortest path between any two configurations can always be characterized by one of these words, which are called the Dubins curves.

7.1. Trajectories optimization using Dubins curves

Because of the fact that Dubins curves provide us an optimal path, they can be used to optimize the trajectory found with the RRT-Path algorithm. For only one UAV, the proposed optimization algorithm works as follows. Two random points of trajectory are chosen and Dubins curves are calculated between them. If calculated curves do not collide with the obstacles, they are used instead of the original trajectory between the points. This step is repeated until the whole trajectory can not be shortened more after e.g. 2000 iterations and thus is sub-optimal.

In a real situation, we do not know whether found trajectory is optimal or not, so we need to determine conditions for stopping the optimization. The optimization is stopped if the trajectory is not shortened after many (e. g. 150) iterations or optimization is too slow and trajectory is shortened only by small distances (e. g. shortening by 5% per 1000 iterations).

For a swarm, the situation is complicated because of relative localization constraints, i.e. minimal and maximal distances between individual UAVs. Dubins curves must be sampled in the same frequency as the trajectory found by the RRT-Path algorithm. The motion model in the RRT-Path algorithm uses constant control input for a fixed time interval T. The frequency of sampling is $\frac{1}{T}$ or its integral multiply when a trajectory is being re-sampled. Each point has to be validated for feasibility in terms of minimal and maximal distance from another UAVs. So the curves can be used only when all trajectories between minimal and maximal distance of relative localization.

Due to using randomly chosen points, the optimization is stochastic and non-deterministic.

7.1.1. One UAV demonstration

In figure 7.1 a trajectory of one UAV found by the RRT-Path algorithm in map with one obstacle marked by a dark grey rectangle is depicted. Obstacle amplification is marked by a light grey rectangle. In figure 7.2 optimal trajectory found using Dubins curves is shown. The resulting trajectory consists of many Dubins curves and it was obtained by algorithm mentioned above.

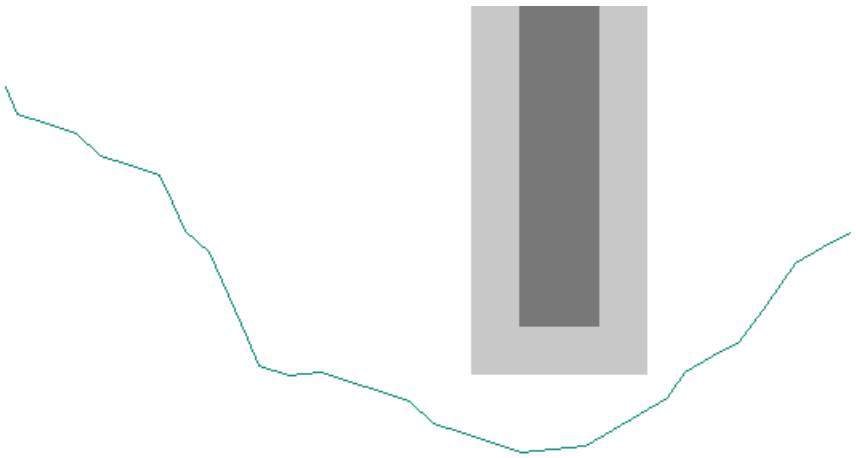


Figure 7.1.: Trajectory of one UAV found by the RRT-Path algorithm before Dubins curves optimization

Random points have been replaced by Dubins curves and after many iterations, e.g. 2000, the optimal trajectory was found.

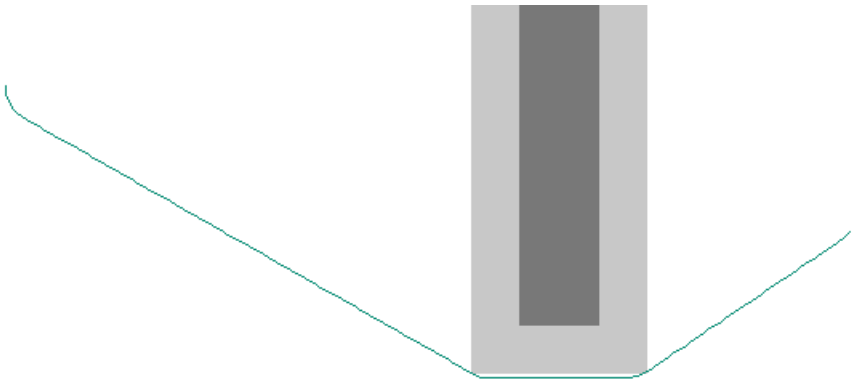


Figure 7.2.: Trajectory of one UAV found by the RRT-Path algorithm after Dubins curves optimization.

CHAPTER
EIGHT

TRAJECTORY RE-SAMPLING

Motion model in the RRT-Path algorithm uses constant control input in time interval from 0.5 to 1 second. Smaller interval for constant input causes RRT-Path algorithm to run for too long. When using too short constant input interval, the tree has too many nodes, grows slowly and runs out of memory much faster than longer interval. An interval longer than 1 second makes UAVs unable to manoeuvre between smaller obstacles. Thus range from 0.5 to 1 second was experimentally chosen as best interval. Using x seconds long constant input interval also means $\frac{1}{x}Hz$ frequency of points in resulting trajectory in the output of the algorithm. So the range from 0.5 to 1 second implies resulting frequency is in range 1Hz to 2Hz.

Real UAVs in Multi-Robot Systems group at CTU use frequency 70Hz for providing target points to UAVs and trajectories with lower frequency are linear interpolated to have frequency 70Hz. That means frequency 2Hz is too low for real usage because a trajectory generated with this frequency would not be smooth enough.

Change of frequency before the RRT-Path algorithm makes the algorithm unable to run efficiently in bigger maps, so this approach does not solve the problem.

Another solution is to re-sample the trajectory after Dubins curves. But this method failed because after Dubins optimization, the curves had different length and different constant input durations.

The best solution for this issue is re-sampling of trajectory generated by RRT-Path algorithm before it is optimized by Dubins curves. This solution also has big advantage in Dubins curves optimization because it results in shorter final trajectory as will be shown in the experiments in the chapter 12.

CHAPTER
NINE

COVERING MORE AOIS WITH ONE SWARM

Some maps have distribution of obstacles and AoIs where algorithm stated above fails. These maps can be seen in figure 9.1. Standard algorithm which uses only one guiding path always leads swarm to only one Area of Interest and the second area remains completely uncovered.

In case of using relative localization where every UAV needs only 1 to 2 neighbours, UAVs can create chain and reach to more distant targets or targets divided by obstacles which UAVs can not reach when moving as one swarm using standard RRT-Path algorithm.

Following modifications must be done if we want to cover all AoIs in figure 9.1.

UAVs are split to two groups, and every group has its own guiding path to one AoI. Relative localization keeps all UAVs in one swarm by its constraints described in section 6.2. With this setting, the swarm behaves like chain, because it is "pulled" to opposite sides by different guiding paths, but it is also connected by relative localization, so it does not split into more smaller swarms. Successful coverage by using the chain behaviour can be seen in figure 9.2.

Unfortunately, this approach does not work in all maps and configurations as can be seen in figure 9.3. The map on the right shows typical example of getting stuck in local minimum, where all UAVs preferred covering only one AoI over covering both AoIs, but the map on the left shows different issue. When we have symmetric map and UAVs do not have starting position in middle of this map, one side of the chain needs to be "pulled" with more power than the other side to cover both AoIs and encircle the obstacle between UAVs starting position and AoIs. This leads to configuration where only one side of chain covers AoI and the other side of chain does not reach the second AoI. This bigger pulling power, which would probably solve this issue, can not be simulated in RRT-Path algorithm. These problems demonstrates the fact this approach is not robust and needs manual preprocessing of UAVs starting positions. The limitation of this approach is also the fact it works only for two AoIs.

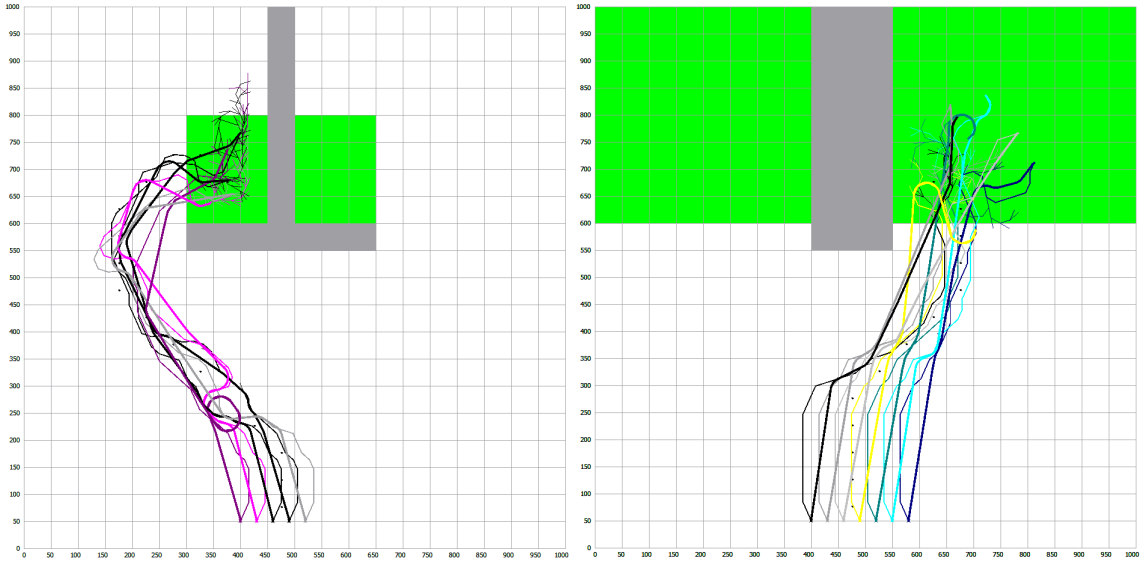


Figure 9.1.: Maps with only one covered Area of Interest

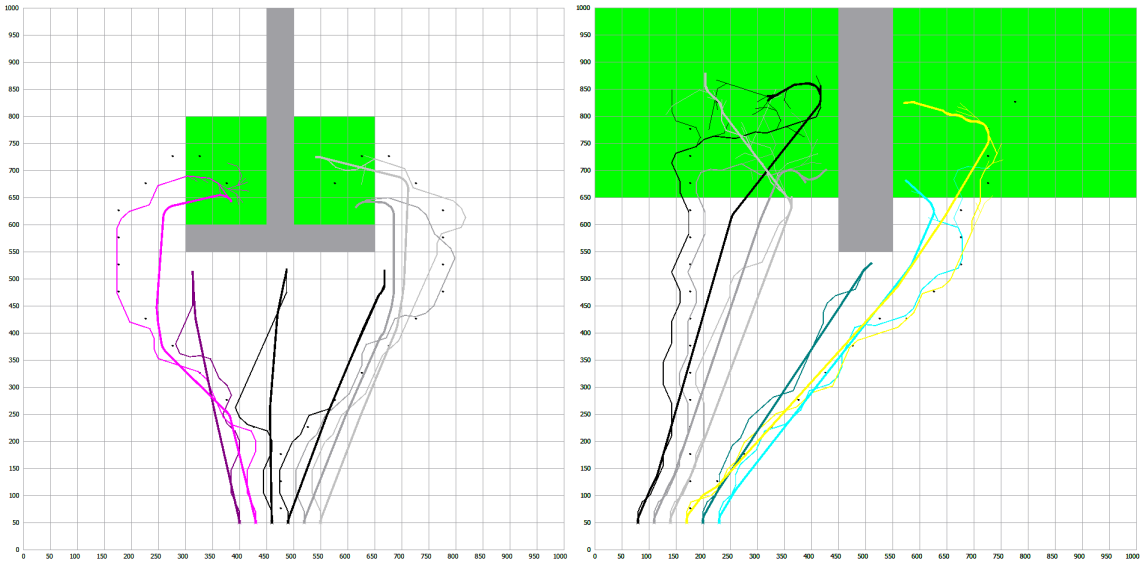


Figure 9.2.: Maps with successful chaining behaviour

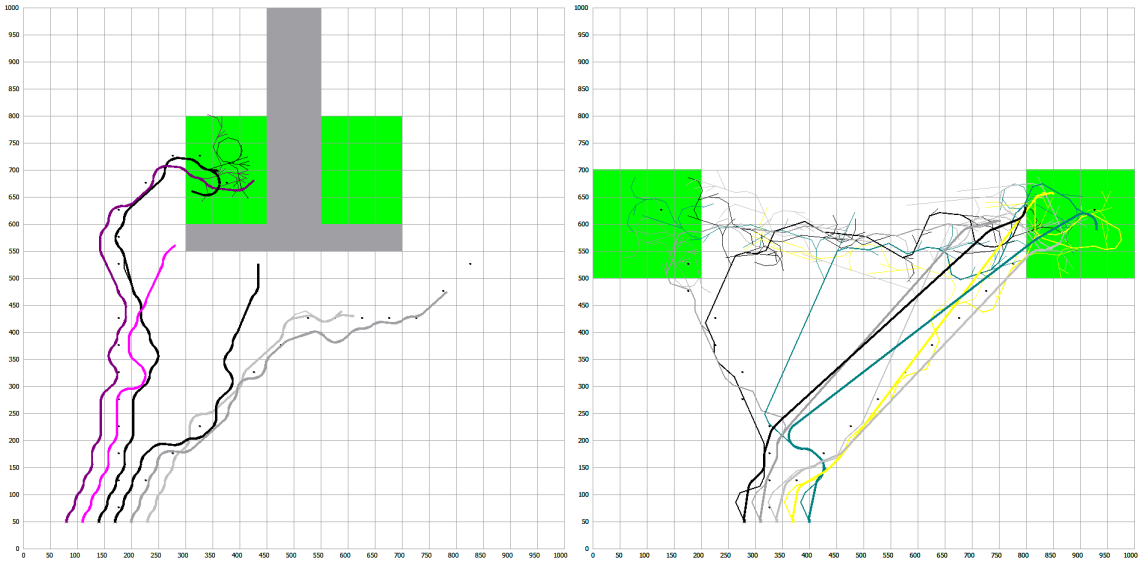


Figure 9.3.: Maps with unsuccessful chaining behaviour

CHAPTER

TEN

V-REP SIMULATIONS

V-REP is an acronym for Virtual robot experimentation platform [2], a simulator developed by Coppelia Robotics, providing an advanced environment for testing and simulations of robots of all types. The V-REP environment is free and open-souce for educational purposes. The environment takes in account certain physical laws like gravity, inertia or friction, which enables to truthfully verify applicability for deployment of UAVs in the real world. V-REP has many build-in models, but user can also create his own robot. V-REP enables to control robots over API and has API clients for C, C++, Python, Java, Lua, Matlab, Octave and Urbi.

10.1. UAV control and trajectory simulation

Python is convenient for fast prototyping and has native functions for easy JSON parsing, which made it good choice for simulations of generated trajectories in V-REP.

UAVs in V-REP can be controlled over remote API only by changing location of their target. Then UAV tries to reach the location of its target. Unfortunately, when using default UAV VREP controller, UAVs only follow location, with speed proportional to distance. UAVs do not try to reach target and simultaneously to have zero speed when reaching their target, which causes overshoot. This fact leads to another disadvantage of such UAV controller. In long and straight corridors the UAV increases its speed, which causes overshoot when trajectory changes its direction because the UAV is not able to slow down and follow its trajectory in turn. These overshoots were many times bigger than size of UAVs, so they could not be ignored and had to be fixed. During first, naive implementation, position of next state of the trajectory was set as target position for UAV, but due to overshoot and large distances between states UAVs failed to follow the trajectory. Another implementation uses linear interpolated trajectory between UAV and its next state position. The calculated target is placed in the line between UAV and next state position, in the constant distance to UAV as

$$\mathbf{X}\left(k+1\right)_{target} = \mathbf{X}\left(k\right)_{UAV} + \frac{\left(\mathbf{X}\left(k\right)_{ns} - \mathbf{X}\left(k\right)_{UAV}\right)}{\left\|\mathbf{X}\left(k\right)_{ns} - \mathbf{X}\left(k\right)_{UAV}\right\|} \cdot const \tag{10.1}$$

where $\mathbf{X}\left(k\right)_{UAV}$ is UAV position in the k-th iteration of the simulation, $\mathbf{X}\left(k\right)_{ns}$ is position of next state in planned trajectory in k-th iteration, $\mathbf{X}\left(k+1\right)_{target}$ is position of UAV target in the $k+1$-th iteration and $const$ is constant experimentally tuned, so the UAV does not move too fast

nor too slow. Too fast movements cause overshoot and too slow movements cause the simulation to run for needlessly long time.

But as mentioned earlier, even this approach do not work well. In long passages, where trajectory did not turn, UAVs increased their velocity and inertia, which made them harder to turn. The problem of overshooting is shown in figure 10.1. The overshoot is at the end of long passage in the map 10.2. Red and violet balls represent positions of next states in trajectory and green balls represent UAV targets. In the first image, we can see UAVs leaving the narrow passage. As you can see in second and third image, positions of next state are in same place, but because of constant distance of target and UAV, the target is dragged by UAVs inertia.

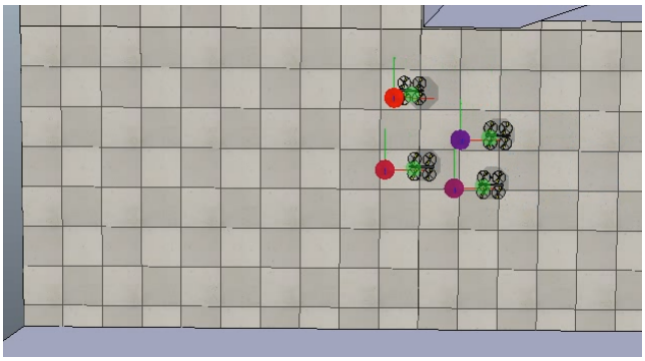
This has been fixed by not updating the position of the target when distance between UAV and the next state is bigger than in previous iteration and the position of the next state is still the same, as

$$\mathbf{X}(k+1)_{target} = \begin{cases} \mathbf{X}(k)_{UAV} + \frac{(\mathbf{X}(k)_{ns} - \mathbf{X}(k)_{UAV})}{\|\mathbf{X}(k)_{ns} - \mathbf{X}(k)_{UAV}\|} \cdot const \\ if \ \|\mathbf{X}(k)_{ns} - \mathbf{X}(k)_{UAV}\| < \|\mathbf{X}(k-1)_{ns} - \mathbf{X}(k-1)_{UAV}\| \\ \wedge \mathbf{X}(k)_{ns} = \mathbf{X}(k-1)_{ns} \\ \mathbf{X}(k)_{target} \\ else \end{cases} \tag{10.2}$$

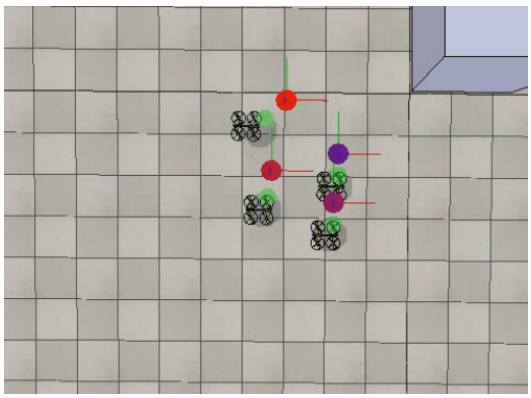
This prevents the target from dragging by UAV with big inertia.

10.2. Simulations

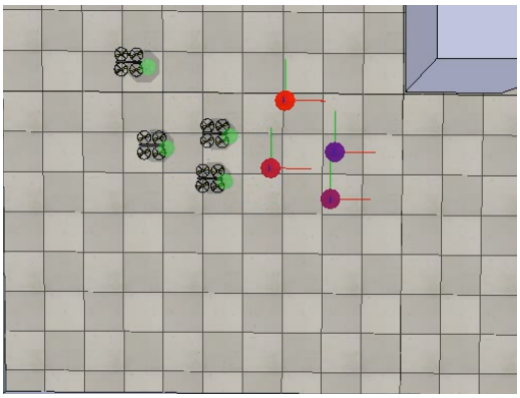
Trajectories obtained from the algorithm proposed in this thesis were simulated and verified in the V-REP environment. Screenshots from the simulation are in figure 10.2. All videos with simulations can be seen in enclosed DVD. Screenshots of trajectories verification on map 12.3 are shown in figure 10.2. Screenshots of trajectories verification on map 12.9 are shown in figure 10.3.



(a)

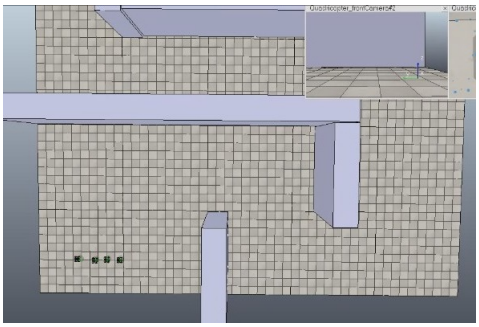


(b)

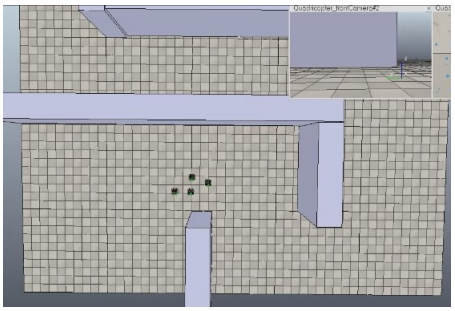


(c)

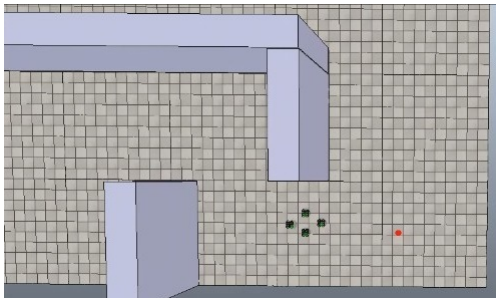
Figure 10.1.: UAV overshoot. In a) we see UAVs flying from the right to the left. Every UAV follows its target. Red and violet balls represent positions of next states in trajectory and green balls represent UAV targets. In the b) the planned trajectory has a curve and next states start heading upwards. In c) all UAVs have too big inertia to follow its targets precisely and they overshoot. The targets are dragged by UAVs and they become more distant from the next state.



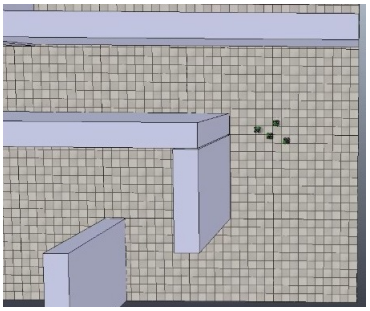
(a) t = 0 s



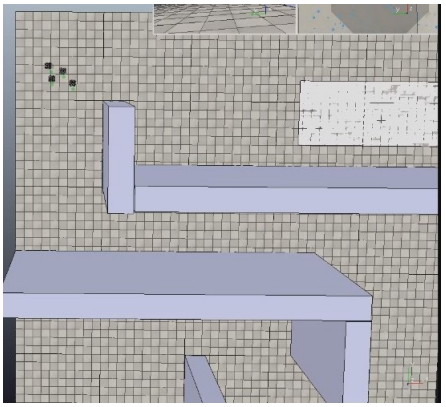
(b) t = 25 s



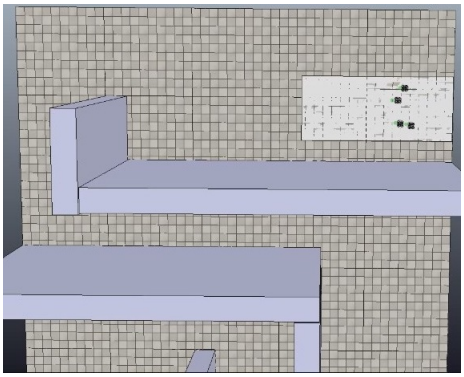
(c) t = 45 s



(d) t = 71 s

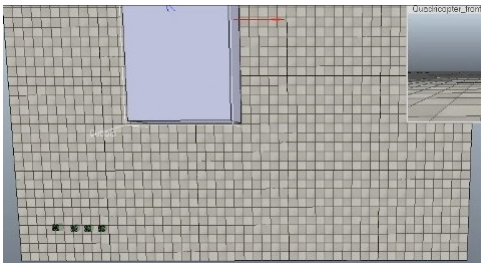


(e) t = 133 s

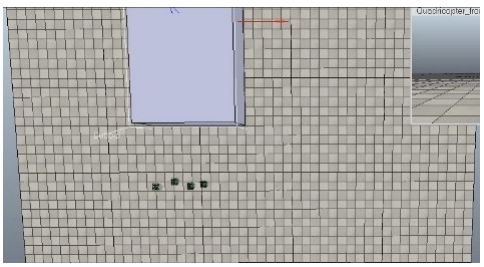


(f) t = 192 s

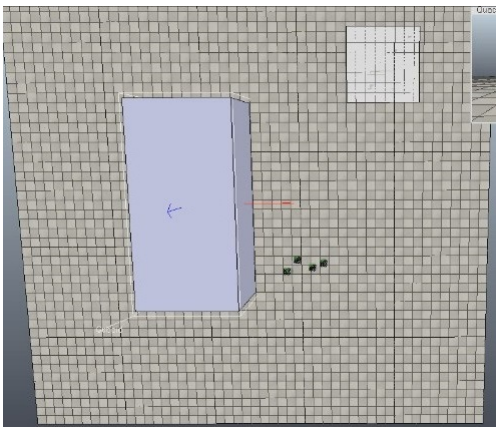
Figure 10.2.: VREP simulation of trajectories in complex map. All UAVs proceeded successfully from the initial position to the AoI, marked with white rectangle. Video with the simulation can be seen in enclosed DVD in the file videos/map0_new.avi. Map used in this simulation is also used in experiments 12.1 and 12.2.1



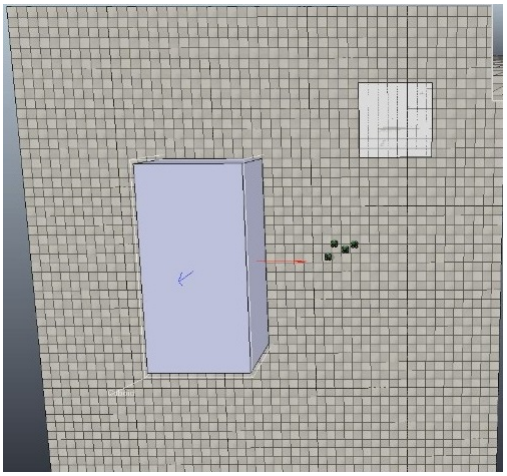
(a) t = 0 s



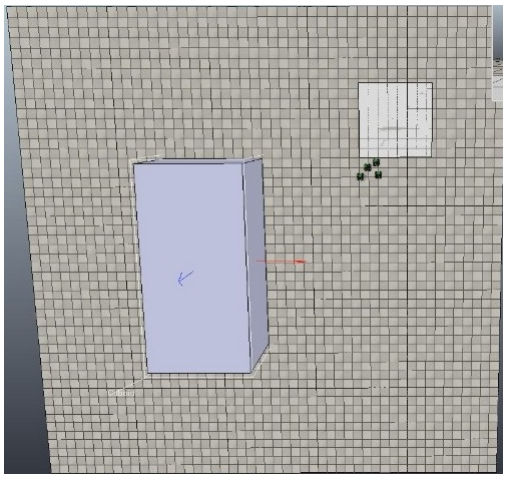
(b) t = 21 s



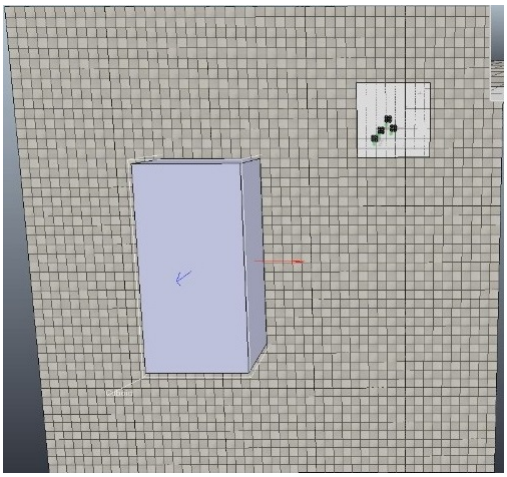
(c) t = 36 s



(d) t = 40 s



(e) t = 45 s



(f) t = 50 s

Figure 10.3.: VREP simulation of trajectories in complex map. All UAVs proceeded successfully from the initial position to the AoI, marked with white rectangle. Video with the simulation can be seen in enclosed DVD in the file videos/map3_new.avi. Map used in this simulation is also used in experiments 12.1 and 12.2.1

CHAPTER
ELEVEN

IMPLEMENTATION

This part will cover implementation of the algorithm, which was used for simulations. Whole codebase in C++ can be found at Github repository `https://github.com/racinmat/AutonomousSurveillanceBachelorThesis`. Apart from the C++ program, I also created some CLI scripts in PHP, for drawing map and trajectories from the JSON representation, batch running of Dubins curves optimization and concatenation of the experiment results before plotting them in Matlab. These can be seen at`https://github.com/racinmat/UAVUtils`. V-REP simulations were made by communicating with V-REP through remote API, the client is written in Python and can be seen ar`https://github.com/racinmat/VRepPathBuilder`. Trajectories are persisted in JSON format. JSON is more compact than XML and can be easily parsed by all widely used programming languages. Trajectories are also persisted to CSV format, so they can be loaded to MATLAB and then into real UAVs.

11.1. External libraries

Some external libraries are used in the implementation. Every used external library is mentioned here. Boost libraries, downloaded from `http://www.boost.org/`, are used for smart pointers and matrix operations, libraries for Dubins curves are from Master Thesis by Petr Váňa [25]. Generating of JSON from C++ object is done via Json spirit library. Another external library is V_Collide from The University of North Carolina at Chapel Hill. The source code of the V_Collide library can be found at `http://gamma.cs.unc.edu/V-COLLIDE/`.

Because V-Collide sources were written in 1997 and because I used C++11 compiler to compile my source codes, I had to rewrite part of this library for compatibility and to make public API easier to use. Modifications can be seen at `https://github.com/racinmat/VCollide2`.

Last used external library is QT, which was used to create platform independent GUI.

11.2. Code structure and services

A brief UML scheme demonstrating dependency diagram of codebase is shown in figure 11.1. To keep diagram simple, only services are displayed. Other classes, which are not services, were left out for readability. Diagram was generated using software StarUML.

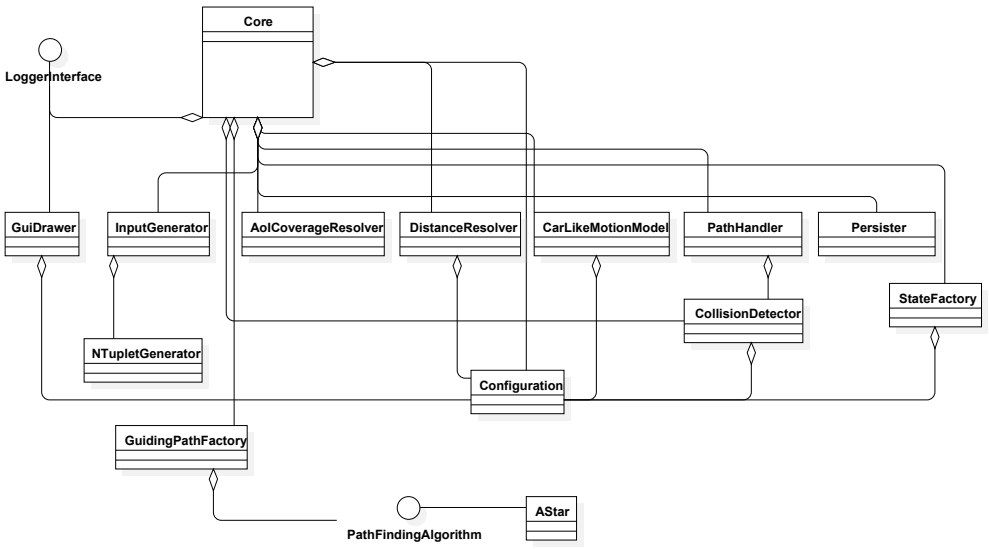


Figure 11.1.: Dependency diagram

Core class holds core of whole Application and has all other classes as dependencies, as is shown in the image 11.1.

As mentioned in the chapter 2, Configuration is DTO for all configuration variables, but to keep reasonable amount of classes, Configuration is also service, which delegates all configuration changes from GUI to Core class. Configuration and GuiDrawer implementation LoggerInterface are the only connections between Core and GUI.

State factory creates State classes according to Factory pattern. State class represents state in RRT-Path algorithm. State has coordinates and rotations for all UAVs. Persister persists found trajectories to JSON using Json Spirit library. PathHandler serves as utils class for manipulations with trajectories (vector of State classes). CarLikeMotionModel holds motion model algorithm. InputGenerator is used to generate inputs to motion model. NTupletGenerator only generates variation with repeating for given input. DistanceResolver counts distances between two states and length of trajectory. AoICoverageResolver determines cost function for states,where all UAVs are in AoIs. GuidingPathFactory is wrapper for PathFindingAlgorithm interface and is used by Core to find guiding path. Implementation of PathFindingAlgorithm is AStart class.

11.3. Utility scripts

All graphs in experiments with re-sampling and Dubins optimization were generated with usage of PHP scripts. Script runDubinsOptimization.php runs sequentially resampling with given frequencies multiple times. This script can be run many times at once with different configuration, which brings advantage of parallel run without need to deals with threads. Script processDubinsOptimizatinData.php merges all CSV result files to one big matrix, with number of rows equal to maximal number of iterations and number of columns equal number of runs of the optimization. For example, as seen in experiment 12.2.1 for frequency 1 Hz it is matrix with size 2095x100. This can be loaded directly to matlab so the graph can be generated. Script drawPaths.php generates map to png image. E. g. map generated by this script can be seen in 12.3.

CHAPTER
TWELVE

EXPERIMENTS

12.1. RRT-Path

In this experiment, a trajectory in a map shown in the figure 12.1 is searched by using the RRT-Path algorithm. UAVs start in the lower left corner. In this figure, we can also see the trajectory found by the RRT-Path algorithm. The algorithm follows the guiding path quite precisely, and whole searching tree for each UAV has small branching factor. Only the AoI is covered by branches of searching trees. The algorithm is run 1000 times and a feasible solution was found in every run. The results can be seen in the figure 12.2, where the mean value and standard deviation can be seen for each iteration of all 1000 runs. The figure 12.2, describes distance to nearest neighbour. This experiment has been run with 3 UAVs. In every iteration of the algorithm, each UAV has a nearest neighbour. Distances between each UAV and its nearest neighbour is shown in the figure 12.2. This information is important for the relative localization. The minimal and maximal distance are 24 and 180 map units, in this experiment. During the whole flight, UAVs fly very close to each other. But in the end, they fly away from each other, because they can cover bigger part of AoI when they have bigger distance between each other.

12.2. Influence of re-sampling on Dubins curves optimization

To demonstrate the optimization, few maps were selected to be used in re-sampling and optimization experiments. The RRT-Path algorithm found trajectories for UAVs. These trajectories were re-sampled and optimized 100 times to obtain relevant results because of using random numbers during the optimization and avoiding getting stuck in local optima. Due to time and memory consumption, each optimization is stopped after 200 iterations where optimization did not shorten the trajectory or when speed of trajectory shortening was slower than 5% of original trajectory length per 1000 iterations. The algorithm also stops when consumed memory exceeds 1900 MB. This is right before shutting of program by operating system, because 32bit processes are not allowed to use more than 2 GB of RAM. In next sections, maps with obstacles, AoIs and trajectories will be shown. Obstacles are grey rectangles, AoI is green rectangle and each UAV has trajectory marked with different colour. For measuring of influence of re-sampling of trajectory to Dubins curves optimization, following frequencies were selected: 1 Hz (initial

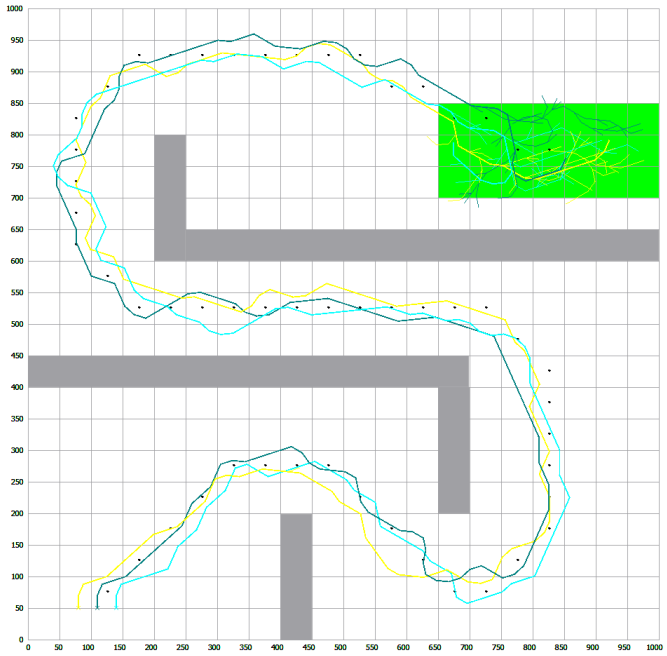


Figure 12.1.: Map used for experiment . Low branching factor can be found along the whole guiding path. When UAVs arrive above AoI, branching factor is higher.

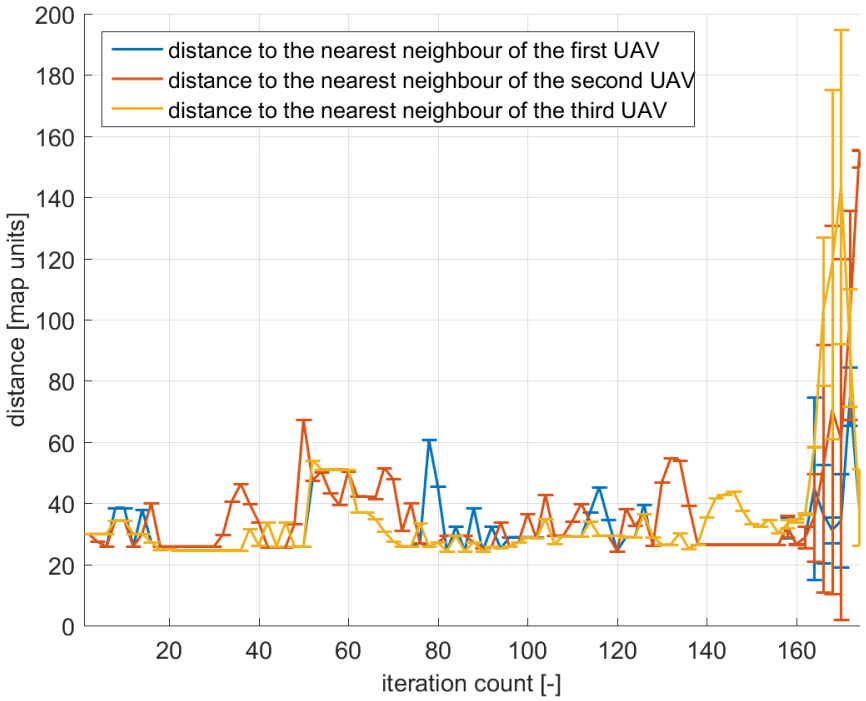


Figure 12.2.: Distances between nearest UAVs in swarm and distance to AoI in experiment 12.1. Relative distance is small along the whole guiding path. Only the covering of AoI lets UAVs get more distant from each other.

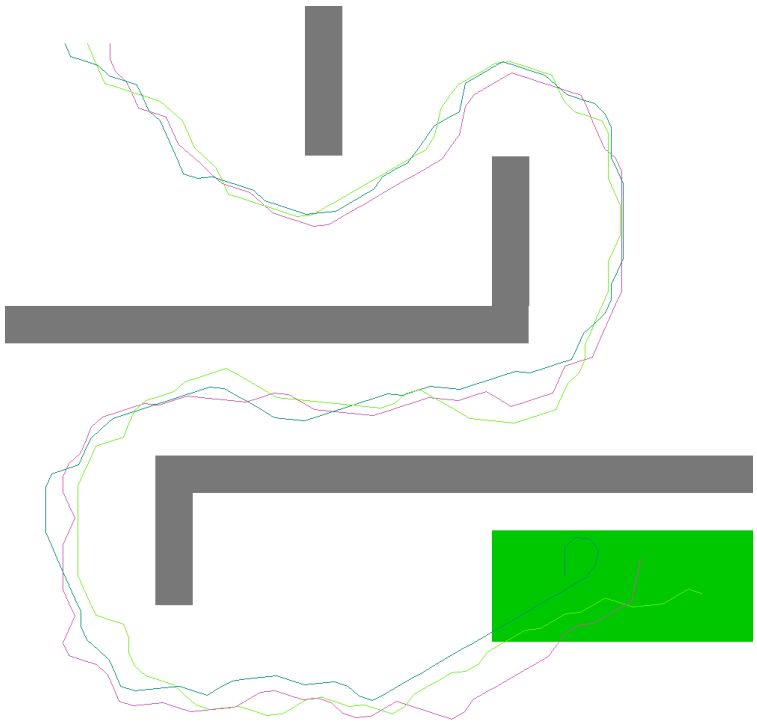


Figure 12.3.: Trajectory before Dubins curves optimization.

frequency used in RRT-Path algorithm), 2 Hz, 4 Hz, 6 Hz, 8 Hz, 10 Hz, 12 Hz, 14 Hz, 16 Hz, 18 Hz, 20 Hz.

12.2.1. First experiment

The map with trajectories found by the RRT-Path algorithm can be seen in 12.3.

The best result of Dubins curves optimization (re-sampling of 20Hz) is shown in 12.4. As we can see, trajectories are much shorter than trajectories before optimization in 12.3. At the beginning of trajectories, in the left upper corner of the picture, we can see much smoother curves than before optimization. This is due to re-sampling to frequency 20Hz, which smooths trajectories.

In real flight, it is undesirable to have trajectories close to obstacles, so obstacles are amplified before optimization. This can be seen in 12.4 where UAVs keep certain distance from the obstacles.

The table 12.1 shows average, minimal and maximal length of all trajectories from 100 optimizations after the re-sampling and optimization.

The results are also shown in graph 12.5. In the graph we can see that the initial frequency 1 Hz has worst results and the frequency 20 Hz has the best results. We can also see that in frequency 14 Hz and higher, all 100 iterations had same results, the minimum, maximum and mean value are the same. But the second best frequency in terms of minimal, maximal and mean value is 6 Hz and even the worst optimization in 6 Hz has smaller total distance than 8 to 18 Hz.

Depending on re-sampling frequency, the courses of optimization are also different.

In 12.6, 12.7 and 12.8 we can see mean values and standard deviations for different frequen-

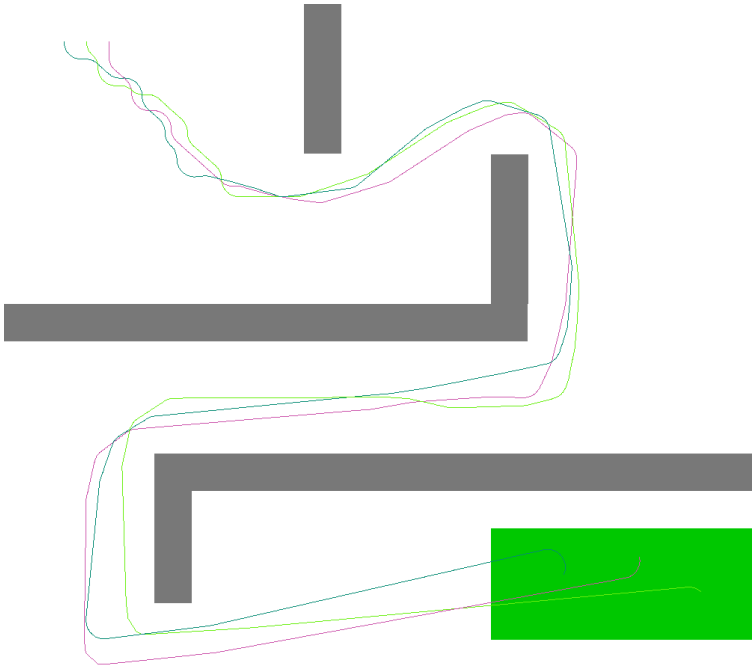


Figure 12.4.: Trajectory after Dubins curves optimization. In the left upper corner the trajectories are not optimized enough. This happened because the optimization got stuck in local minimum. The local minimum was caused by relative localization constraints, which complicates the optimization. The crossing trajectories are very difficult to optimize due to relative localization constraints.

Frequency [Hz]	Minimal distance [m]	Maximal distance [m]	Average distance [m]
1	8582.18	8849.7	8721.2904
2	8311.65	8548.81	8430.23
4	8366.88	8393.09	8379.985
6	8248.9	8275.7	8262.3
8	8249.88	8378.51	8314.195
10	8286.22	8472.2	8379.21
12	8302.51	8309.2	8307.6613
14	8303.18	8303.18	8303.18
16	8363.92	8363.92	8363.92
18	8510.32	8510.32	8510.32
20	8194.22	8194.22	8194.22

Table 12.1.: Re-sampling and optimization results of the experiment 12.2.1. The initial frequency has

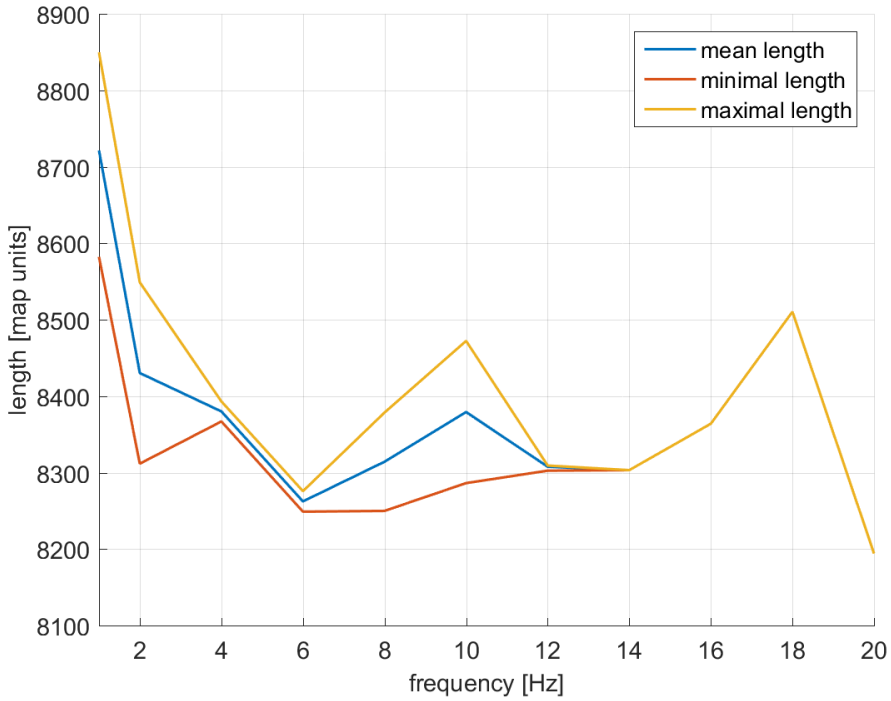


Figure 12.5.: Re-sampling and optimization results graph. This is visualization of data from table 12.1.

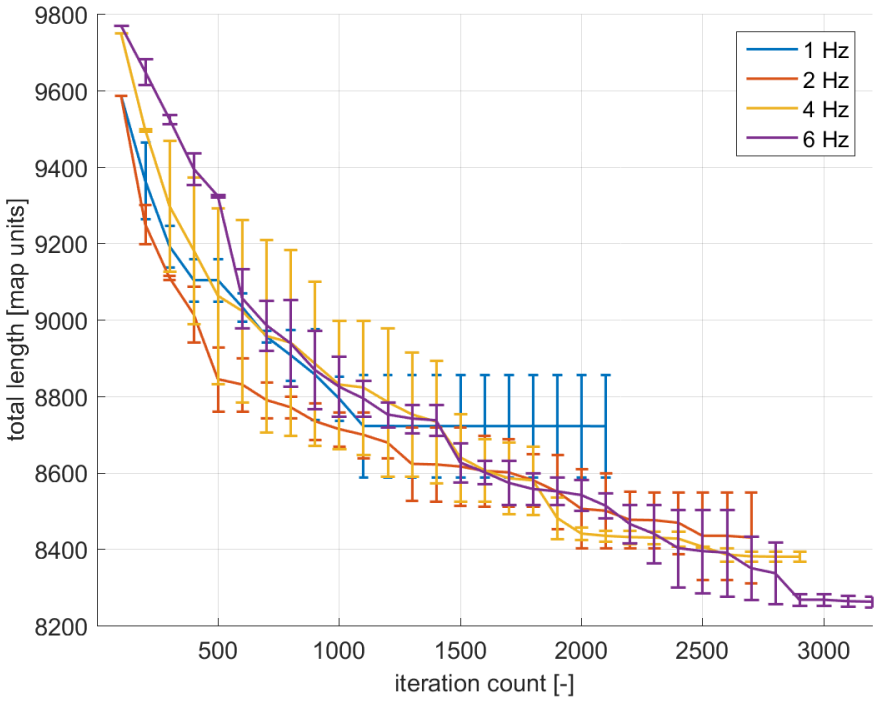


Figure 12.6.: rogress of length of paths during the optimization of trajectory in figure 12.3 for 2 Hz, 4 Hz, 6 Hz

cies, divided into three graphs for better readability. The vertical lines are error bars, they show standard deviation during the optimization. Because the error bars would be too dense if they were shown for each iteration, only every 100th iteration is shown in the graphs. For comparison, frequency 1 Hz is also shown in each graph, the initial frequency before re-sampling. As we can see, frequencies 14, 16, 18 and 20 Hz have almost zero standard deviation and converge to lower value than the initial frequency. High standard deviation can be seen for frequency 10 Hz. That means the optimization got stuck in local optimum and was not able to shorten any trajectory for many iterations.

12.2.2. Second experiment

The best result of Dubins curves optimization (re-sampling of 4Hz) is shown in figure 12.10. As we can see, trajectories are much shorter than trajectories before optimization in figure 12.9, as in experiment 1, but curves in the upper part of figure still were not optimized. This was caused by optimization algorithm getting stuck in local optimum. As shown in figure 12.4, optimizing trajectory for one UAV does not get stuck in local optimum. The local optimum is caused by relative localization constraints and crossing paths. The algorithm tried to optimize other parts and ended due to stopping conditions mentioned above. Light grey colour represents obstacle amplification.

The results are shown in figure 12.11. Contrary to the graph from first experiment 12.2.1, all frequencies have same minimal distance, maximal distance and mean distance. This shows us interesting fact. For same frequency, all 100 runs got stuck in same local optimum, but every

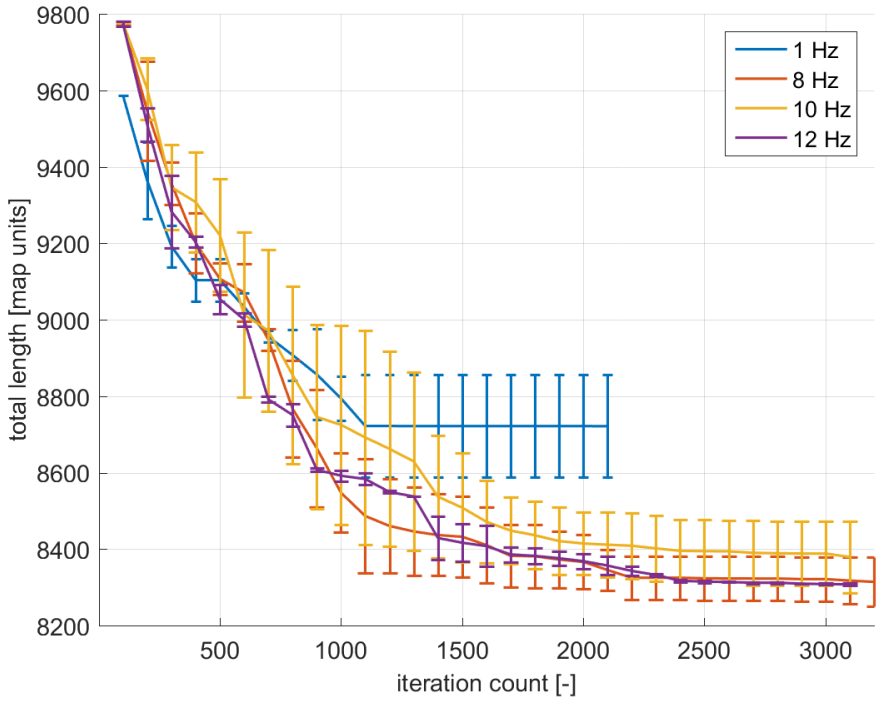


Figure 12.7.: Progress of length of paths during the optimization of trajectory in figure 12.3 for 8 Hz, 10 Hz, 12 Hz

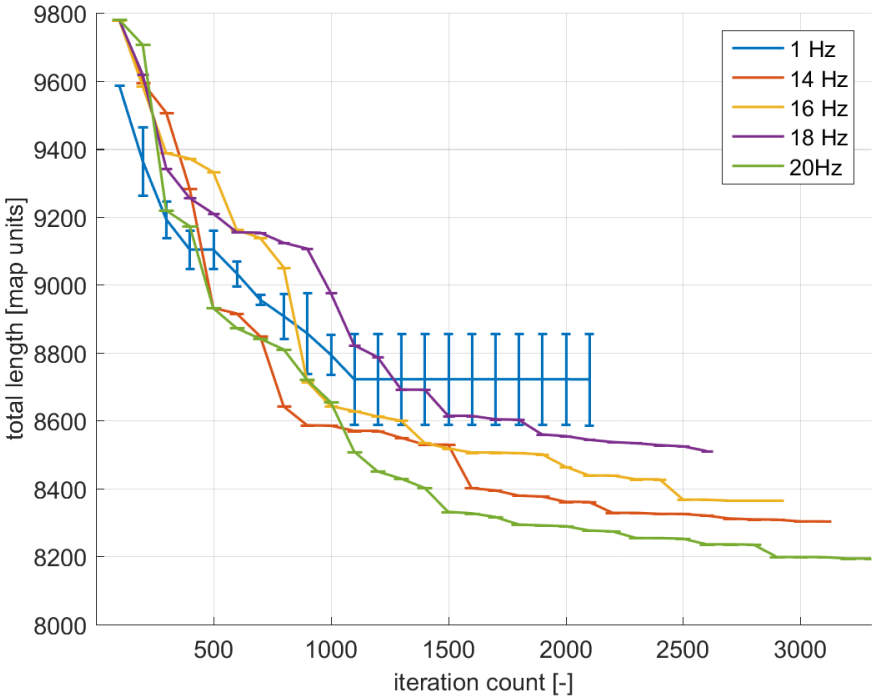


Figure 12.8.: Progress of length of paths during the optimization of trajectory in figure 12.3 for 14 Hz, 16 Hz, 18 Hz, 20 Hz

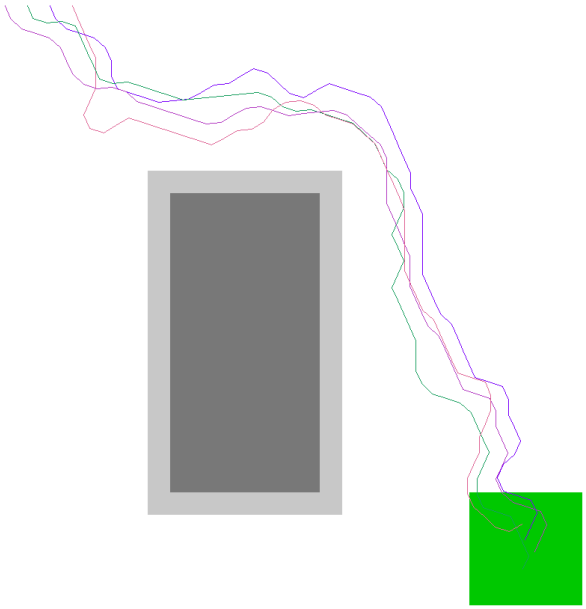


Figure 12.9.: Trajectory in experiment 12.2.2 before Dubins curves optimization

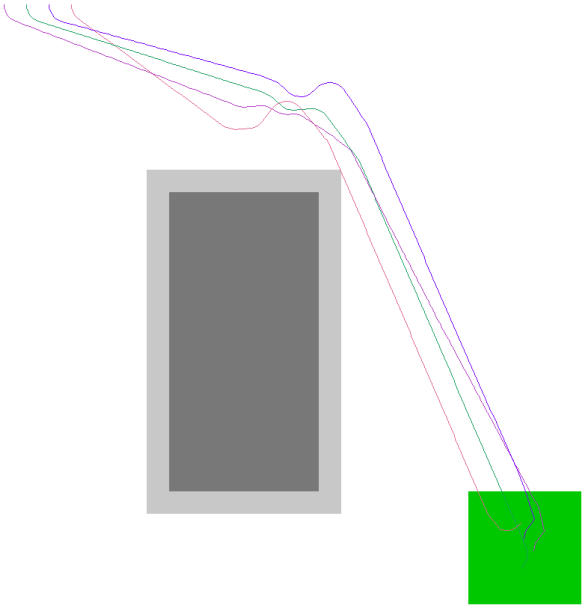


Figure 12.10.: Trajectory in experiment 12.2.2 after Dubins curves optimization

frequency has different local optimum where the algorithm can stuck. Trajectory in this experiment is much smaller, which leads to zero standard deviation and difference between minimal and maximal distance between optimization results in one frequency. The difference between maximal and minimal results is bigger when optimizing longer and more complicated trajectory. As we can also see from graphs, we can not predict optimal re-sampling frequency from trajectory.

Depending on re-sampling frequency, the progresses of length of paths during the optimization are also different.

In figures 12.12, 12.13 and 12.14, we can see mean values and standard deviations for different frequencies, divided into three graphs for better readability. The vertical lines are error bars, they show standard deviation during the optimization. The error bars are shown only in every 10th iteration n graphs for better readibility. For comparison, frequency 1 Hz, the initial frequency before re-sampling, is also shown on each graph. As we can see, in comparison to experiment 1, standard deviations are zero, so the optimization algorithm exhibits deterministic behaviour even if this optimization method is stochastic.

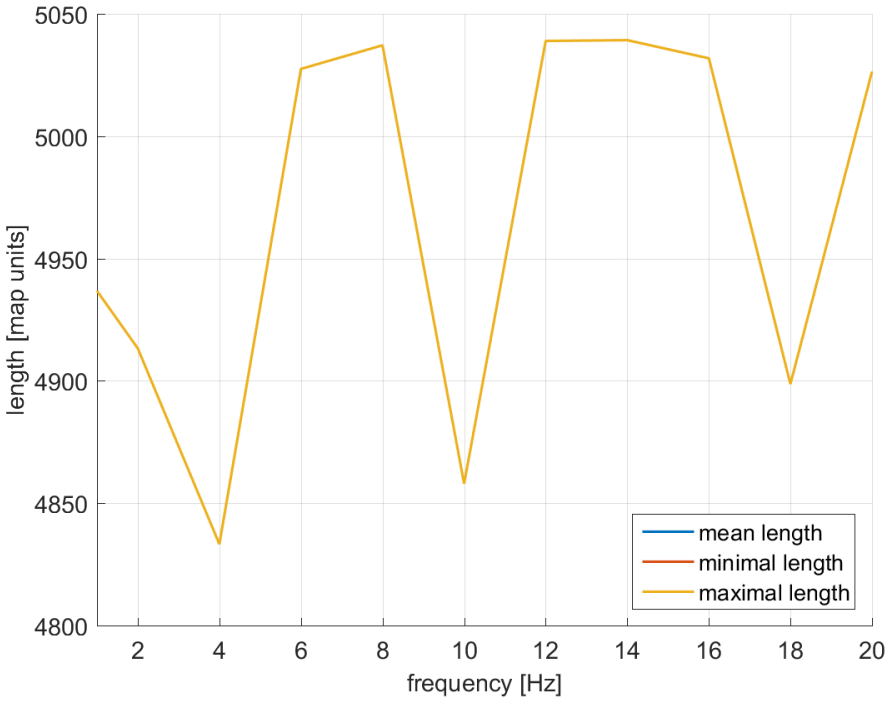


Figure 12.11.: Re-sampling and optimization results graph of experiment 12.2.2. Mean distance, maximal distance and minimal distance are overlapping in this graph. This is caused by zero standard deviation of results at the end of optimization.

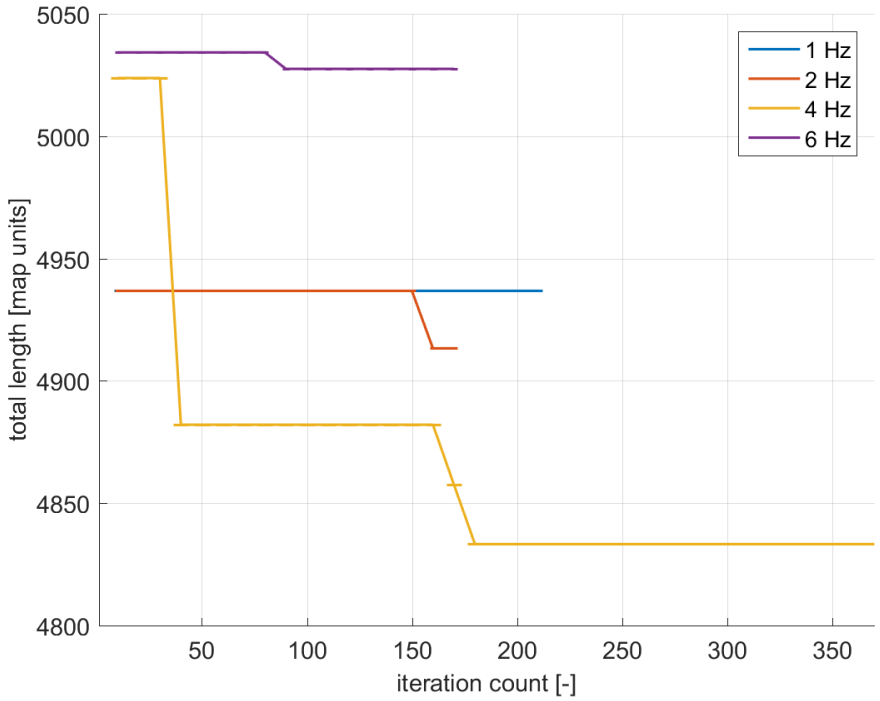


Figure 12.12.: Progress of length of paths during the optimization of trajectory in figure 12.9 for 2 Hz, 4 Hz, 6 Hz. As shown in figure, the optimization got stuck in local optima very soon for frequency 6 Hz.

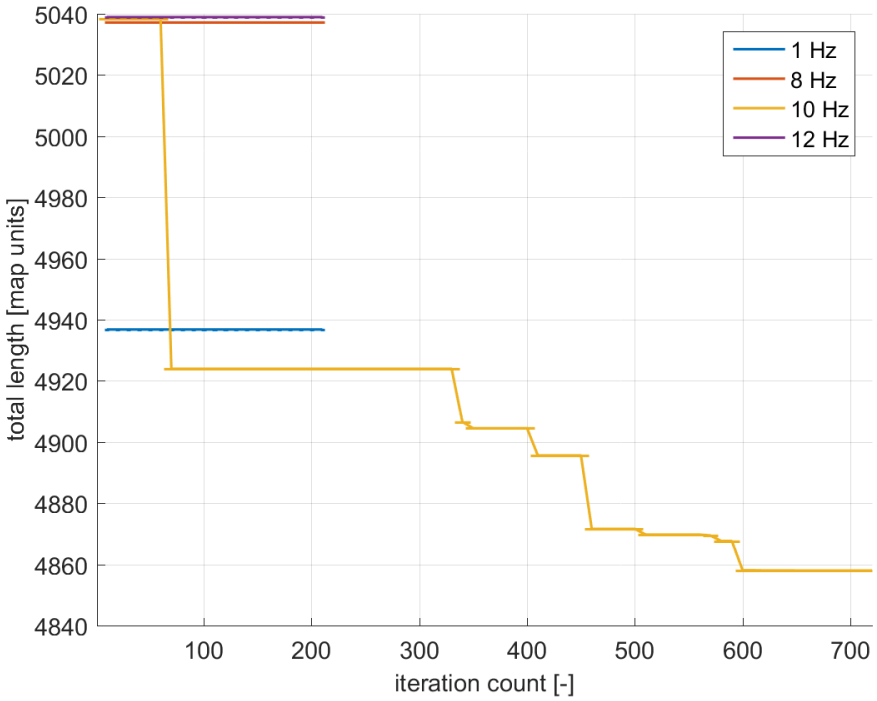


Figure 12.13.: Progress of length of paths during the optimization of trajectory in figure 12.9 for 8 Hz, 10 Hz, 12 Hz.

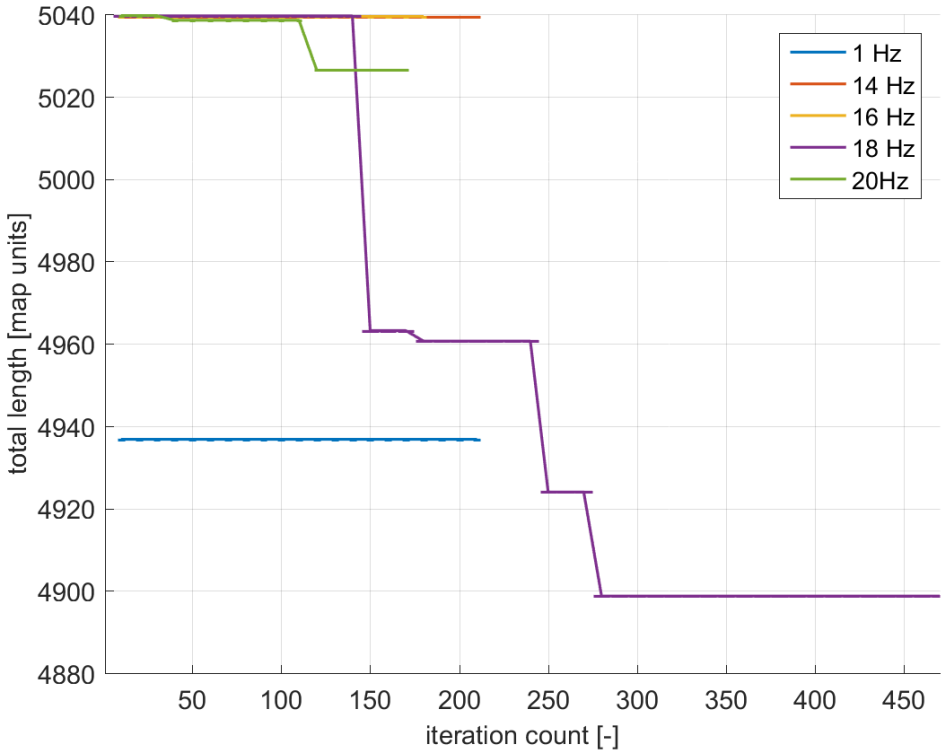


Figure 12.14.: Progress of length of paths during the optimization of trajectory in figure 12.9 for 14 Hz, 16 Hz, 18 Hz, 20 Hz.

CHAPTER
THIRTEEN

CONCLUSION

This thesis contributes to the problems of autonomous surveillance with a novel approach by using a sampling-based algorithm for trajectory planning of a swarm of UAVs and consequence optimizing of obtained trajectories by Dubins curves. This thesis also examined the possibility of re-sampling of trajectories and influence of re-sampling on the optimization process.

The goal of this thesis was to design and implement a method based on the RRT-Path algorithm for motion planning of a swarm of cooperating unmanned aerial vehicles in the task of autonomous surveillance, optimize it by Dubins curves and simulate the flight of the swarm in V-REP platform. The algorithm provides trajectories from a depot station to optimal positions in Areas of Interest in two internationally standardized formats (CSV and JSON). An easy computable cost function was created and implemented.

The task of autonomous surveillance was approached by separating the it to more smaller subtasks. The first subtask is finding a path to a position, in which all UAVs are located above Areas of Interest. The path finding is realized by using A* algorithm to obtain the optimal guiding path. The RRT-Path algorithm uses this path to guide the space-filling tree. The second subtask is optimization of AoIs coverage using the RRT algorithm. The next subtask is re-sampling of trajectories and optimizing trajectories by Dubins manoeuvrers, which provide us shorter trajectories than trajectories found only by using the RRT-Path algorithm. The effectiveness of Dubins curves optimization depends on re-sampling frequency, but unfortunately, the best frequency is heavily dependent on the map where trajectories are searched.

The presumed use of autonomous surveillance is monitoring of a parking lot near supermarket, shopping centre, company campus, etc. The advantage of an autonomous system over static cameras is the fact the UAV swarm can operatively change the distribution of individual UAVs according to the position of cars and minimize dead angles by moving UAVs. Static cameras usually fail to capture the identity of a criminal due to large distance or a bad angle, while UAVs can change their positions to capture the image of criminal in higher quality or from better angle and even provide simultaneous record from more cameras. Not only parking lots can be monitored, another possibility is monitoring of people during large events or large scale monitoring of agricultural areas. UAVs also scale much easier than static cameras, because UAVs can fly to their destination without human help, but static cameras must be installed by human and fast scalability is important factor in massive events and situations where size of Area of Interest change over time. Contrarily to GPS localized UAVs, swarm described in this thesis can even operate in hard to access areas, dense urban areas, inside buildings and other places with

no GPS signal, thanks to the relative localization system. The relative localization also enables the UAVs to fly in compact swarm, which could not be achieved by using GPS due to its errors and inaccuracy.

13.1. Future work

The approach described in chapter 9 could be modified to be more robust and usable in more maps with more than 2 AoIs. Generalization of this approach would require not crating a chain, which is good only for covering two AoIs, but tree structure, where each leaf covers one AoI. I will now propose principle of the generalized algorithm.

In the first step, we need to decide, which AoIs can be covered by swarm. Some maps can have distant AoIs where all AoIs could not be covered at the same time. This requires finding shortest tree T which connects middles of all AoIs. When sum of all edges in this tree d_{total} is bigger than $d_{max} = (number\ of\ uavs - 1) \cdot maximal\ distance\ between\ neighbouring\ uavs$, which is maximal length of chain created by UAVs, some AoI and branch leading to its middle will be excluded. This results in shortening the value d_{total}. This process will be repeated while d_{total} will be bigger than maximal d_{max}. When $d_{total} < d_{max}$, we obtain set of AoIs which can be covered by swarm of UAVs. In the next step, UAVs will be split to multiple groups connected by relative localization constraints, one group for each AoI. Then the guiding paths will be planned for each groups to reach corresponding branches of T. Then the RRT-Path algorithm will take place. The final step will be optimization by Dubins curves, described in this thesis. The exclusion of some AoIs could be used to split AoIs to more groups. Then each group will be covered by its own sub-swarm and trajectories will be planned independently.

Another proposal of future work is implementing better controller for the UAV in VREP environment. Controller which is able to control both position and speed of UAV would solve problems described in the chapter 10.

BIBLIOGRAPHY

[1] Heard on the grapevine: drones to transform viticulture, July 2015. URL https://droneapps.co/case-study-drones-to-transform-viticulture/.

[2] V-rep, May 2016. URL http://www.coppeliarobotics.com/.

[3] Lester Eli Dubins. On curves of minimal length with a constraint on average curvature, and with prescribed initial and terminal positions and tangents. *American Journal of Mathematics*, 79(3):497–516, July 1957. URL http://www.jstor.org/stable/2372560?origin=crossref&seq=1#page_scan_tab_contents.

[4] Jan Faigl, Tomáš Krajník, Jan Chudoba, and Martin Saska. Low-cost embedded system for relative localization in robotic swarms. *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, pages 993 – 998, May 2013.

[5] Roosevelt A. Fernandes. Monitoring system for power lines and right-of-way using remotely piloted drone., April 1989. URL http://www.google.com/patents/US4818990.

[6] Brian Fung. Why drone makers have declared war on the word 'drone', August 2013. URL https://www.washingtonpost.com/news/the-switch/wp/2013/08/16/why-drone-makers-have-declared-war-on-the-word-drone/.

[7] L. E. Kavraki, P. Svestka, J. C. Latombe, and M. H. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Transactions on Robotics and Automation*, 12:566–580, August 1996.

[8] Steven M. LaValle. Rapidly-exploring random trees: A new tool for path planning. Technical report, Department of Computer Science, Iowa State University, 1998. URL http://msl.cs.uiuc.edu/~lavalle/papers/Lav98c.pdf.

[9] Steven M. LaValle. *Planning Algoritms*. Cambridge University Press, 2006. URL http://planning.cs.uiuc.edu/.

[10] International Civil Aviation Organization. *Unmanned Aircraft Systems (UAS)*. International Civil Aviation Organization, April 2011.

[11] Andrea Peterson. The switch states are competing to be the silicon valley of drones, August 2013.

[12] Matěj Petrlík. Planning of swarm deployment for autonomous surveillance. Bachelor's thesis, Czech technical university in Prague Faculty of Electrical Engineering Department of Cybernetics, 2015.

[13] M. Saska. MAV-swarms: unmanned aerial vehicles stabilized along a given path using onboard relative localization. In *Proceedings of 2015 International Conference on Unmanned Aircraft Systems (ICUAS)*, 2015.

[14] M. Saska, V. Vonasek, T. Krajnik, and L. Preucil. Coordination and Navigation of Heterogeneous UAVs-UGVs Teams Localized by a Hawk-Eye Approach. In *Proceedings of 2012 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2012.

[15] M. Saska, T. Krajnik, V. Vonasek, P. Vanek, and L. Preucil. Navigation, Localization and Stabilization of Formations of Unmanned Aerial and Ground Vehicles. In *Proceedings of 2013 International Conference on Unmanned Aircraft Systems (ICUAS)*, 2013.

[16] M. Saska, J. Chudoba, L. Preucil, J. Thomas, G. Loianno, A. Tresnak, V. Vonasek, and V. Kumar. Autonomous Deployment of Swarms of Micro-Aerial Vehicles in Cooperative Surveillance. In *Proceedings of 2014 International Conference on Unmanned Aircraft Systems (ICUAS)*, 2014.

[17] M. Saska, Z. Kasl, and L. Preucil. Motion Planning and Control of Formations of Micro Aerial Vehicles. In *Proceedings of The 19th World Congress of the International Federation of Automatic Control (IFAC)*. IFAC, 2014.

[18] M. Saska, T. Krajnik, V. Vonasek, Z. Kasl, V. Spurny, and L. Preucil. Fault-Tolerant Formation Driving Mechanism Designed for Heterogeneous MAVs-UGVs Groups. *Journal of Intelligent and Robotic Systems*, 73(1-4):603–622, 2014.

[19] M. Saska, J. Langr, and L. Preucil. Plume Tracking by a Self-stabilized Group of Micro Aerial Vehicles. In *Modelling and Simulation for Autonomous Systems*, 2014.

[20] M. Saska, J. Vakula, and L. Preucil. Swarms of Micro Aerial Vehicles Stabilized Under a Visual Relative Localization. In *Proceedings of 2014 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2014.

[21] M. Saska, V. Vonasek, T. Krajnik, and L. Preucil. Coordination and Navigation of Heterogeneous MAV–UGV Formations Localized by a ‘hawk-eye’-like Approach Under a Model Predictive Control Scheme. *International Journal of Robotics Research*, 33(10):1393–1412, 2014.

[22] M. Saska, T. Baca, J. Thomas, J. Chudoba, L. Preucil, T. Krajnik, J. Faigl, G. Loianno, and V. Kumar. System for deployment of groups of unmanned micro aerial vehicles in GPS-denied environments using onboard visual relative localization. *Autonomous Robots. First online.*, 2016.

[23] M. Saska, V. Vonásek, J. Chudoba, J. Thomas, G. Loianno, and V. Kumar. Swarm distribution and deployment for cooperative surveillance by micro-aerial vehicles. *Journal of Intelligent & Robotic Systems. First online.*, 2016.

[24] V. Trianni. *Evolutionary Swarm Robotics: Evolving Self-Organising Behaviours in Groups of Autonomous Robots*. Studies in Computational Intelligence. Springer Berlin Heidelberg, 2008. ISBN 9783540776123. URL https://books.google.cz/books?id=sg9rCQAAQBAJ.

[25] Petr Váňa. Path planning for non-holonomic vehicle in surveillance missions. Master's thesis, Czech Technical University in Prague, 2015. URL https://dspace.cvut.cz/bitstream/handle/10467/61814/F3-DP-2015-Vana-Petr-thesis.pdf.

[26] Chris Velazco. Amazon is experimenting with autonomous flying delivery drones, December 2013.

[27] V. Vonasek, J. Faigl, T. Krajnik, and L. Preucil. RRT-Path: a guided Rapidly exploring Random Tree. In *Robot Motion and Control 2009*, pages 307–316, Heidelberg, 2009. Springer. ISBN 978-1-84882-984-8.

[28] V. Vonasek, M. Saska, L. Winkler, and L. Preucil. High-level motion planning for cpg-driven modular robots. *Robotics and Autonomous Systems*, 68:116 – 128, 2015.

[29] Vojtěch Vonásek. *A guided approach to sampling-based motion planning*. PhD thesis, Czech Technical University in Prague Faculty of electrical engineering Department of cybernetics, August 2015.

APPENDIX
A

CONTENTS OF THE ENCLOSED CD

SwarmDeployment	Main application
SwarmDeployment	Source code of application in C++
Win32	Compiled binaries for 32bit Windows
UtilityScripts	PHP utility scripts
VRepPathBuilder	Source code of VREP simulation program
bibliography	Used bibliography
videos	Videos of VREP simulations
readme.txt	Brief manual for binaries
bachelor_thesis.pdf	Electronic version of bachelor thesis