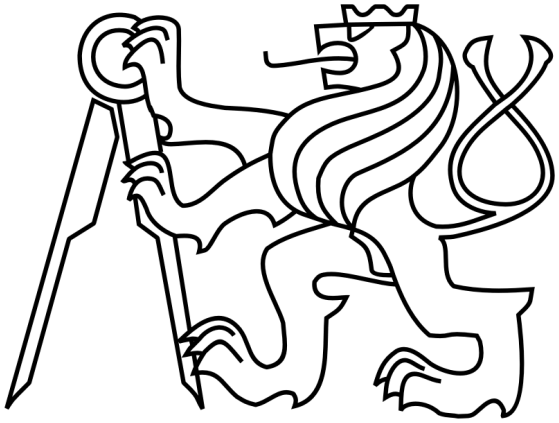
Czech technical university in Prague
Faculty of Electrical Engineering
Department of Cybernetics



Bachelor Thesis

Planning of swarm deployment for autonomous surveillance

Author: Matěj Petrlík

Thesis supervisor: Ing. Vojtěch Vonásek

In Prague on May 22, 2015

Czech Technical University in Prague
Faculty of Electrical Engineering

Department of Cybernetics

BACHELOR PROJECT ASSIGNMENT

Student: Matěj P e t r l í k

Study programme: Cybernetics and Robotics

Specialisation: Robotics

Title of Bachelor Project: Planning of Swarm Deployment for Autonomous Surveillance

Guidelines:

The goal of the thesis is to design and implement an algorithm for motion planning of a group of cooperating micro aerial vehicles (MAV) in the task of autonomous surveillance. The task of the motion planning is to compute sensing locations to observe desired areas of interest as well as trajectories from a depot station to these sensing locations. The motion models of MAVs [2] and motion constraints caused by the relative localization of MAVs [1] have to be considered in the motion planning. The provided motion plans do not need to be optimal.

1. Implement a motion planning algorithm based on Rapidly Exploring Random Trees (RRT) [4] for groups of MAVs moving in environments with obstacles. Design a fast heuristic function to enable RRT to find candidates of sensing locations together with the trajectories of MAVs.
2. Design an optimization method to improve sensing locations found in the previous step. Design a suitable cost function.
3. Experimentally verify the algorithms and compare them with the method described in [3].

Bibliography/Sources:

[1] Faigl, J. - Krajník, T. - Chudoba, J. - Přeučil, L. - Saska, M.: Low-Cost Embedded System for Relative Localization in Robotic Swarms. In ICRA2013: Proceedings of IEEE International Conference on Robotics and Automation, 2013.
[2] Lee, T. - Leoky, M. – McClamroch, N.H.: Geometric tracking control of a quadrotor UAV on SE(3), IEEE Conference on Decision and Control (CDC), 2010.
[3] Saska, M. - Chudoba, J. - Přeučil, L. - Thomas, J. - Loianno, G. - et al.: Autonomous Deployment of Swarms of Micro-Aerial Vehicles in Cooperative Surveillance. In Proceedings of International Conference on Unmanned Aircraft Systems (ICUAS), 2014.
[4] LaValle, Steven M.: Rapidly-Exploring Random Trees A New Tool for Path Planning, 1998.

Bachelor Project Supervisor: Ing. Vojtěch Vonásek

Valid until: the end of the summer semester of academic year 2015/2016

L.S.

doc. Dr. Ing. Jan Kybic
Head of Department

prof. Ing. Pavel Ripka, CSc.
Dean

Prague, February 12, 2015

České vysoké učení technické v Praze
Fakulta elektrotechnická

Katedra kybernetiky

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Student: Matěj P e t r l í k

Studijní program: Kybernetika a robotika (bakalářský)

Obor: Robotika

Název tématu: Plánování rozmístění roje helikoptér v úloze autonomního dohledu

Pokyny pro vypracování:

Cílem práce je navrhnout a implementovat algoritmus umožňující naplánovat rozmístění skupiny bezpilotních helikoptér vhodné pro úlohu autonomního dohledu. Součástí nalezeného řešení bude kromě finálního rozmístění skupiny také plán pohybu helikoptér z výchozí pozice do získaného rozmístění. Tento plán pohybu nemusí být optimální, ale musí splňovat omezení daná vizuální relativní lokalizací členů roje [1] a dynamická omezení pohybu helikoptér [2].
Úkoly:

1. Implementujte algoritmus plánování pohybu založený na metodě RRT (Rapidly exploring Random Tree) [4] pro formace helikoptér. Rozšiřte algoritmus o heuristiku tak, aby bylo možné kromě plánování pohybu hledat i vhodné rozmístění helikoptér nad požadovanými oblastmi zájmu.
2. Navrhněte optimalizační metodu pro vylepšení rozmístění helikoptér nalezených v předchozím bodě, navrhněte hodnotící funkci.
3. Navržené algoritmy porovnejte s metodou založenou na Particle Swarm Optimization [3].

Seznam odborné literatury:

[1] Faigl, J. - Krajník, T. - Chudoba, J. - Přeučil, L. - Saska, M.: Low-Cost Embedded System for Relative Localization in Robotic Swarms. In ICRA2013: Proceedings of IEEE International Conference on Robotics and Automation, 2013.
[2] Lee, T. - Leoky, M. – McClamroch, N.H.: Geometric tracking control of a quadrotor UAV on SE(3), IEEE Conference on Decision and Control (CDC), 2010.
[3] Saska, M. - Chudoba, J. - Přeučil, L. - Thomas, J. - Loianno, G. - et al.: Autonomous Deployment of Swarms of Micro-Aerial Vehicles in Cooperative Surveillance. In Proceedings of International Conference on Unmanned Aircraft Systems (ICUAS), 2014.
[4] LaValle, Steven M.: Rapidly-Exploring Random Trees A New Tool for Path Planning, 1998.

Vedoucí bakalářské práce: Ing. Vojtěch Vonásek

Platnost zadání: do konce letního semestru 2015/2016

L.S.

doc. Dr. Ing. Jan Kybic
vedoucí katedry

prof. Ing. Pavel Ripka, CSc.
děkan

V Praze dne 12. 2. 2015

Prohlášení autora práce

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

V Praze dne _____

_____
Podpis autora práce

Název práce: Plánování rozmístění roje helikoptér v úloze autonomního dohledu

Autor: Matěj Petrlík

Katedra (ústav): Katedra kybernetiky

Vedoucí bakalářské práce: Ing. Vojtěch Vonásek

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

Abstrakt V předložené práci studujeme nasazení roje složeného z bezpilotních helikoptér v úloze autonomního dohledu. Úloha spočívá v nalezení trajektorie do zadaných oblastí zájmu a v optimalizaci rozmístění roje nad oblastí zájmu za účelem pokrytí její co největší možné části, které umožňuje počet jedinců roje. Kvalita pokrytí oblasti zájmu je reprezentována navrženou hodnotící funkcí, založenou na diskretizaci konfiguračního prostoru. Metoda založená na takzvaných rychle rostoucích náhodných stromech (RRT) je vyvinuta za účelem nalezení trajektorie s omezeními danými překážkami a pravidly roje. Optimalizace cílového rozmístění je realizována algoritmem optimalizace roje částic (PSO), dále je prozkoumána možnost použít RRT jako optimalizační metodu. Výsledky všech navržených metod jsou analyzovány a experimentálně porovnány s referenční PSO metodou.

Klíčová slova: Autonomní dohled, Roje bezpilotních helikoptér, Plánování trajektorií, Rychle rostoucí náhodné stromy, Optimalizace roje částic

Title: Planning of swarm deployment for autonomous surveillance

Author: Matěj Petrlík

Department: Department of Cybernetics

Supervisor: Ing. Vojtěch Vonásek

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

Abstract In the presented work we study deployment of a swarm consisting of Unmanned Aerial Vehicles (UAVs) in the task of autonomous surveillance. The task consists of finding a trajectory to specified Areas of Interest (AoI) and optimizing the swarm distribution over them to cover the largest possible part of AoI allowed by the number of swarm individuals. The quality of AoI coverage is represented by a proposed cost function based on configuration space discretization. A method based of Rapidly-exploring Random Trees is developed to find a trajectory under constraints consisting of obstacles and swarm rules. The optimization of the final position is realized by Particle Swarm Optimization (PSO) algorithm, furthermore the possibility to use RRT as an optimization method is explored. Results of all proposed methods are analyzed and experimentally compared to a reference PSO-based method.

Keywords: Autonomous surveillance, Swarms of Unmanned Aerial Vehicles, Trajectory planning, Rapidly-exploring Random Trees, Particle Swarm Optimization

ACKNOWLEDGEMENTS

I would like to thank all the people that helped me with this bachelor thesis. Especially, I would like to thank my supervisor Ing. Vojtěch Vonásek for guiding me through all the problems I encountered and for giving me valuable insights. Further I would like to thank my family and all my friends for support.

CONTENTS

LIST OF FIGURES

CHAPTER

ONE

INTRODUCTION

1.1 Motivation

Unmanned Aerial Vehicle (UAV) - an aircraft which is intended to operate with no pilot on board, often referred to as a *drone*, has been gaining popularity in recent years in academic circles and wide public. The term *Micro Aerial Vehicle* and its abbreviation *MAV* is also used sometimes. They are becoming smaller, safer, able to carry more payload, easier to control, more robust, more flexible and most importantly cheaper. It is even possible to assemble own UAV from commonly available parts. A DIY quadrotor UAV is shown in Figure 1.1.

Figure 1.1: Example of a quadrotor UAV assembled at the Department of Cybernetics of CTU in Prague

International Civil Aviation Organization (ICAO) [1] classifies UAVs into two groups: *Remotely piloted aircraft* - an aircraft where the flying pilot is not on board the aircraft, 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. Typical uses of teleoperated aircraft are inspection of power lines [2], monitoring of agricultural areas [3] or 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 solid cameras.

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

Both large companies and small projects are competing in developing variety of different kinds of autonomous aircraft for various purposes. Most commonly discussed are drone deployments in goods delivery [4], localization of people or other objects in rescue missions taking place in dangerous or hard to access areas and above all autonomous surveillance of *Areas of Interest (AoI)*.

Smaller dimensions, better affordability and inspiration in nature, as presented in [5], led to the idea of forming UAV swarms. Any UAV group that is somehow coordinated can be considered a swarm. A group of remotely piloted aircraft is impossible, because one person cannot control more than one UAV and even if each UAV had its own operator, it would be difficult to avoid collisions between UAVs. Due to these reasons this thesis deals exclusively with autonomous aircraft, even when using the more general abbreviation UAV. An interesting example of collaborating swarm of UAVs can be found in [6], where a dynamic model of multiple quadrotors carrying cable-suspended payload is presented. Some of the benefits over UAV individuals are larger area covered by cameras in monitoring or surveillance missions, redundancy in case some individuals get broken, higher computing power and collective intelligence. On the other hand, when individual UAVs form swarms, some issues that need to be solved arise.

First of all, collisions between swarm members need to be avoided, which

means, that a way 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 motion planning algorithm has to be developed in order to move the swarm from initial position shown in Figure 1.3a to a target position in Figure 1.3b in 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 [8] 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) might not be sufficient.

A more suitable localization technique is described in [9]. The localization is performed by on-board cameras capturing space around UAVs. Each UAV carries a localization pattern consisting of a black circle (Figure 1.2) on a white background. The on-board system then calculates the distance from camera to the localization image from the knowledge of size and shape of the image.

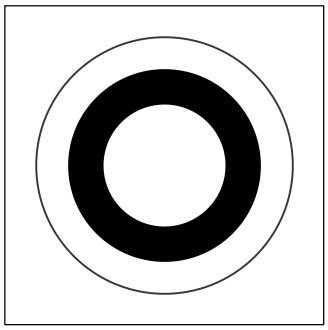
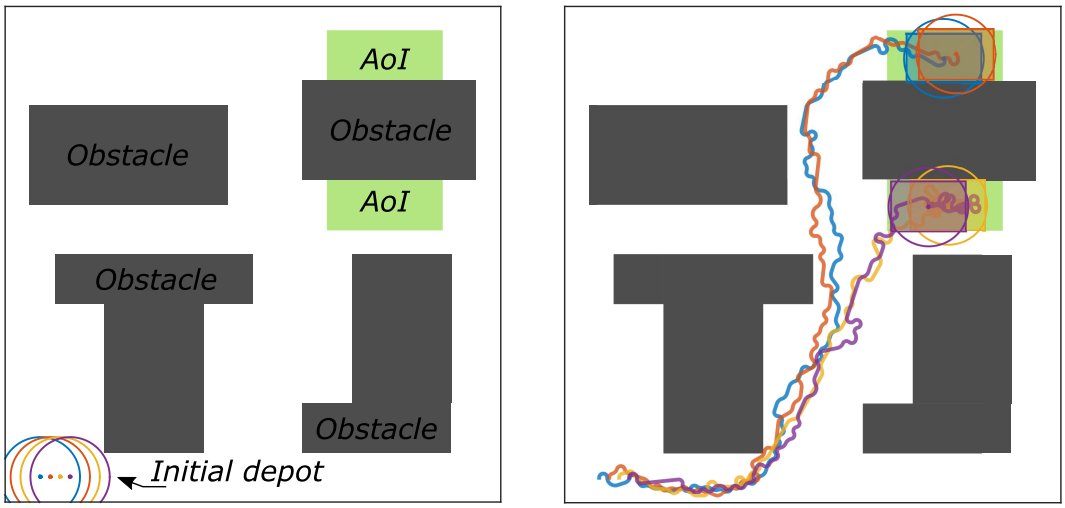


Figure 1.2: Circular localization pattern carried by each UAV

To avoid collisions, minimal relative distance must be greater than the distance at which two UAVs are able to stop when flying against each other at full speed, while maximal relative distance is constrained by the ability to recognize the localization pattern in the distance. The localization constraints present a valid reason to treat the problem as two dimensional, since the flight altitude has to be constant.



(a) UAVs located in their initial depot (charging station) with obstacles and AoI

(b) A possible solution - UAVs covering the AoI (not optimally) with trajectories from initial depot

Figure 1.3: An example of a surveillance problem with a solution

1.2 Objective

The objective of this thesis is to explore the possibility of using swarms of quadrotor UAVs in the task of autonomous surveillance. The task is defined as finding a collision free trajectory from an initial depot to an AoI and covering the largest possible part of AoI.

Several approaches exist in the task of autonomous surveillance. One approach is to find an optimal position of each individual UAV from the swarm over the AoI by an optimization algorithm in the first step and then find a trajectory from initial position of the swarm to the optimal position with the use of a trajectory planning algorithm in the second step. The problem of this static approach is that the optimal position might not be feasible, i.e., it might be impossible to find a trajectory to the position, due to obstacles or localization constraints. Second possibility is to find a trajectory and optimize the coverage at the same time, thus guaranteeing feasibility of the trajectory. The problem was already tackled by *Multi-robot Systems* group at *Czech Technical University in Prague* where a method [10] based on *Particle Swarm Optimization (PSO)* was developed. This method succeeds in finding a feasible trajectory to an optimally covered AoI, however, the trajectories produced are too complicated and excessively long, thus leaving less battery life of UAVs left for monitoring the AoI.

Principal idea of this thesis is to use an algorithm based on *Rapidly-exploring Random Trees* (*RRT*) and their improved versions for solving the core planning task. The properties of RRT promise to deal with the issue of unnecessarily complicated trajectories produced by PSO. The RRT-based method from this thesis will be compared to the PSO-based method to find its benefits and drawbacks.

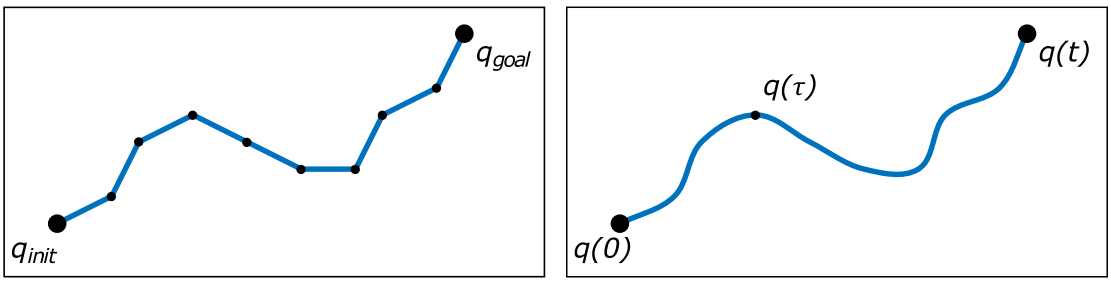
To successfully implement RRT, a number of support algorithms and systems are required. To simulate the behavior of the UAVs, a *motion model* must be chosen. One of the requirements of the planning task is collision avoidance realized by *v-collide* library [11]. A *localization system* must be present to keep the swarm together. A suitable *fitness function* will be needed to evaluate the positions of swarms. Different concepts of these supporting methods and their descriptions are mentioned in the next Chapter. The implementation details are described in Chapter 3. The comparison of RRT and PSO along with results of the experiments are presented in Chapter 4.

CHAPTER

TWO

STATE OF THE ART

In the field of motion planning there are some terms that need to be defined for correct description of the problems, their solutions and experiments. In this thesis, the term *motion planning* is used as a general term when referring to planning problems. When discussing specifically planning problems with only geometric constraints, i.e., connecting place A with place B, the term *path planning* (Figure 2.1a) is used, while the term *trajectory planning* (Figure 2.1b) refers to navigation of a robot with both geometric, kinematic and dynamic constraints, considering its *motion model.*



(a) Path planning (b) Trajectory planning

Figure 2.1: Path planning and trajectory planning

Path planning

In planning algorithms it is important to correctly define the *configuration space* $\mathcal{C}$ which is a set of possible transformations that could be applied to the robot. Defining configuration space offers a certain level of abstraction allowing motion

planning problems with different geometric and kinematic constraints to be solved by the same planning algorithms.

The world is defined in [12] as $\mathcal{W} = \mathbb{R}^2$ or $\mathcal{W} = \mathbb{R}^3$, which contains an obstacle region $\mathcal{O} \subset \mathcal{W}$. A rigid body *robot* is defined as $\mathcal{A} \subset \mathcal{W}$. A *configuration* $q \in \mathcal{C}$ represents a complete specification of state variables of $\mathcal{A}$. The minimal number of state variables is the Degree of Freedom (DoF) of the robot. For representing a configuration of a simple ground mobile robot like the one in Figure 2.2, it is sufficient to know its Cartesian coordinates and yaw angle which yields 3 state variables corresponding with 3 DoF of the robot, so $q = (x, y, \varphi)$.

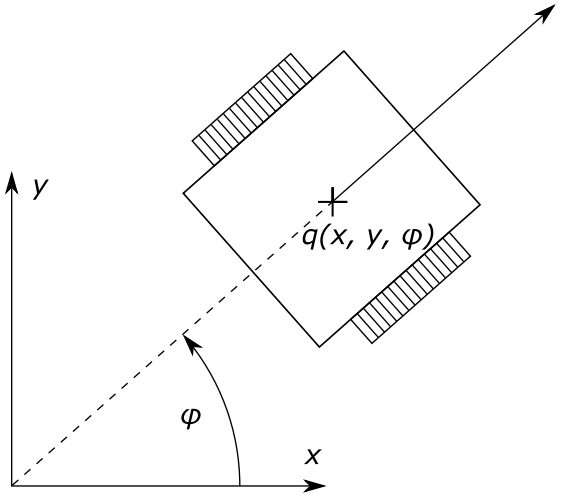


Figure 2.2: Simple ground robot with 3 state variables

The *obstacle region* $\mathcal{C}_{obs} \subset \mathcal{C}$ is defined as

$$\mathcal{C}_{obs} = \{q \in \mathcal{C} \mid \mathcal{A}(q) \cap \mathcal{O} \neq \emptyset\}, \tag{2.1}$$

which is the set of all configurations q, at which the robot body $\mathcal{A}(q)$ intersects the obstacle region $\mathcal{O}$. The remaining configurations are called *free space* and defined as $\mathcal{C}_{free} = \mathcal{C} \backslash \mathcal{C}_{obs}$

Let us define path planning problems as connecting initial configuration $q_{init} \in \mathcal{C}_{free}$ with goal configuration $q_{goal} \in \mathcal{C}_{free}$ through the free space $\mathcal{C}_{free}$ while avoiding configurations colliding with obstacles $\mathcal{C}_{obs}$. An example of simple path planning problem can be seen in Figure 2.3. The solution of a path planning problem is a sequence of configurations which are essentially points in a multidimensional space without any information about time or action inputs needed for the robot to get from one configuration to the next one.

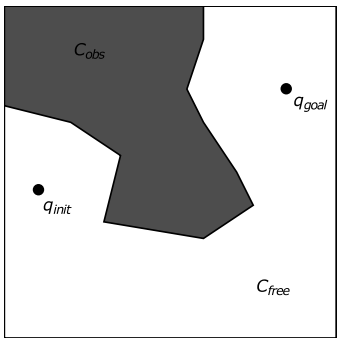


Figure 2.3: Simple path planning problem with free space and obstacle region

Trajectory planning

The purpose of trajectory planning is to find a trajectory consisting of motions considering the constraints of a real robot. A forward motion model

$$\dot{q} = f(q, u), \tag{2.2}$$

where $u \in \mathcal{U}$ is a control input from a set of possible inputs. The forward motion model describes how a configuration q changes after applying control input u. The set of all possible control inputs $\mathcal{U}$ depends on the motion model of robot.

Motion model is a mathematical model of a robot that represents its movement abilities, kinematic and dynamic constraints. Inputs of motion model are control inputs of actuators responsible for the movement of the robot. Output is the change of its state variables. In this thesis, two motion models were considered. Since this thesis deals with UAVs, the first motion model is a quadrotor motion model as quadrotors have a high freedom of movement allowing them to change direction quickly when following a trajectory. Second chosen motion model is a car-like motion model, even though it is primarily used for two dimensional problems and UAVs fly in 3D space.

Also the time information is added to each configuration in the trajectory so $q(\tau)$ must exist for each $\tau \in [0, t]$ (or each time step if τ is discrete), where t is the duration of the trajectory. Then the task of the trajectory planning is to find a trajectory, starting at $q(0)$ and ending at $q(t)$ such that the motion model $f(q, u)$ is respected. Such a trajectory can be represented e.g. by a sequence of control inputs u. This thesis is primarily focused on the trajectory planning, even though some path planning algorithms are also used when necessary. Trajectory planning is called planning under differential constraints, which are caused by dynamics

and kinematics of the robot.

Sampling-based algorithms

When dealing with motion planning problems a necessity to pick an algorithm that is best suited for a specific task arises. Important properties of motion planning algorithms are time needed to find a solution, ability to find the best path, avoiding local optima and high-dimensional problems solving. While many algorithms perform well in low-dimensional motion planning problems, their performance decreases significantly as the number of dimensions increases, producing unsatisfying results due to the fact, that their run time is exponentially dependent on the number of dimensions.

To solve high-dimensional path planning problems, an algorithm that is not exponentially dependent on the dimension is needed. Currently, the best algorithms suited for high-dimensional problems are *sampling-based algorithms*, which try to find the path using samples randomly drawn from the configuration space. Sampling-based algorithms are probabilistically complete, meaning the more time they are running, the more the probability to find a solution approaches 1. However, determining if there is no solution is impossible.

Probabilistic road map [7] is one of sampling-based algorithms consisting of two phases. The first phase is a *construction phase* which randomly samples n configurations $\mathcal{C}_{sampled} = (q_1, q_2, ..., q_n)$ called *milestones* from $\mathcal{C}_{free}$. Then each $q \in \mathcal{C}_{sampled}$ is connected to other near configurations by a local planner with collision avoidance to form a graph called *road map*. Usually, the local planner is realized by a straight-line planner, that connects two configurations by a line segment. The second phase is a *query phase*. After adding q_{init} and q_{goal} to the road map, a graph search algorithm is used to obtain the path shown in Figure 2.4.

Another sampling-based algorithm is *Rapidly-exploring Random Tree (RRT)*. It is designed to search high-dimensional spaces rapidly. RRTs are constructed incrementally from random samples of the configuration space in a way that the tree grows quickly towards the unsearched areas (Figure 2.5). This algorithm is particularly suited for trajectory planning problems involving obstacles and differential constraints, therefore its use in motion planning of UAV swarms is further explored in this thesis.

The major disadvantage of all sampling-based algorithms is the *narrow passage* problem depicted in Figure 2.6. With uniform sampling of the configura-

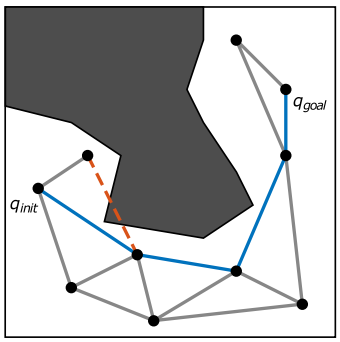


Figure 2.4: Probabilistic road map with highlighted blue path passing through $\mathcal{C}_{free}$ and a forbidden red dashed edge passing through $\mathcal{C}_{obs}$

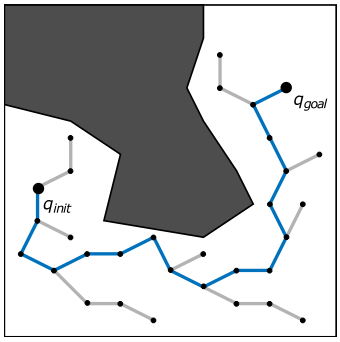


Figure 2.5: Example of a tree constructed by Rapidly-exploring Random Tree with a highlighted path

tion space the probability that a random sample will be drawn from the multi-dimensional volume of the narrow passage is rather low. This problem is usually dealt with by increasing the sampling probability in and around the narrow passage, however the narrow passage has to be detected first, e.g. based on the workspace knowledge. Location of narrow passage can be estimated based on medial axis [13, 14]. Another approach is to shift the random samples towards the free areas [15]. Although these techniques can significantly improve performance in narrow passages, they are useful only in low dimension space, since the importance of workspace knowledge decreases as the number of DoFs increases [16]. Analysis of RRT performance in an environment with narrow passages can be found in [17].

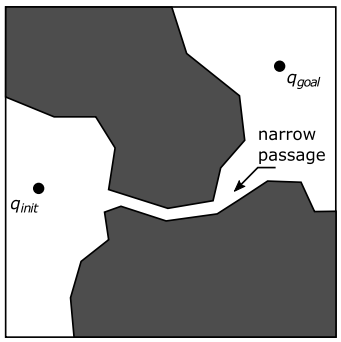


Figure 2.6: Narrow passage problem

2.1 RRT

Rapidly-exploring Random Tree is used for trajectory planning in this thesis therefore an explanation of its principles follows. The tree starts growing from its root (initial node q_{init}). In each iteration a *random sample* is drawn from the search space and a link between the random sample and the *nearest node* of the tree is attempted. The maximum length of the link is one of the parameters of RRT algorithm usually referred to as the growth factor. Furthermore a *motion model* can be implemented easily by picking the link from a pool of possible movements (inputs) generated by the motion model. If the link does not pass through any obstacle and satisfies all specified constraints, the point at the end of this link is added to the tree as a *new node*. The result of drawing the samples randomly from the search space is, that the tree expands towards unexplored areas quickly and eventually covers the whole search space after enough iterations are carried out. The *progressive expansion* of RRT can be seen in Figure 2.7

An RRT τ rooted at an initial configuration q_{init} with K iterations is constructed as shown in Algorithm 1

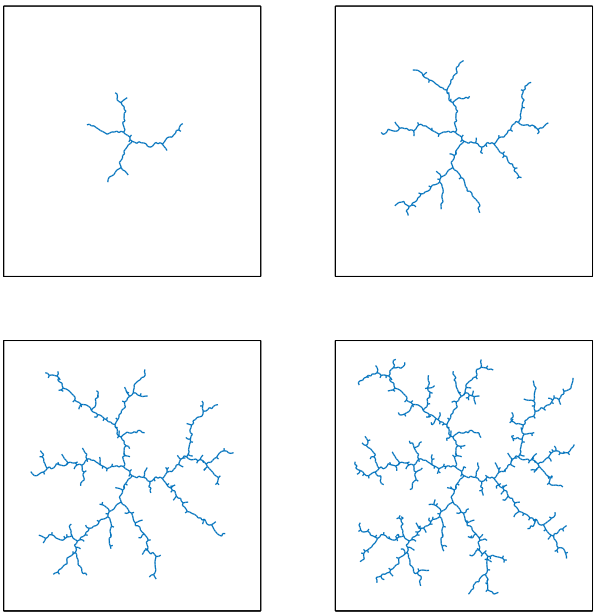


Figure 2.7: RRT after 100, 300, 600 and 1000 iterations

Algorithm 1: Construct RRT

```
1  τ.init(q_init);
2  for k ← 1 to K do
3      q_rand ← randomState();
4      q_near ← nearestNeigbor(q_rand, τ);
5      u ← selectInput(q_rand, q_near);          // input used for expansion
6      q_new ← newState(q_near, u);
7      if q_new is feasible then
8          τ.addNode(q_new);
9          τ.addEdge(q_near, q_new, u);
10 return τ
```

The first node of τ is q_{init}. In each iteration a random state q_{rand} is selected from the state space. Step 4 finds the closest node to q_{rand}. Step 5, called expansion, selects an input u that minimizes the distance from q_{near} to q_{rand}. Step 6 evaluates a potential new state q_{new}, which is added in step 8 as a vertex to τ if the state is feasible. An edge from q_{near} to q_{new} is also added, because the input u needed to reach q_{new} from q_{near} is not necessarily a straight line.

The expansion of an RRT is heavily biased towards unexplored portions of the

search space, which helps to avoid getting stuck at local optima. The distribution of nodes in an RRT approaches the sampling distribution, leading to consistent behavior so the direction of RRT growth is easily influenced. This feature can be exploited by biasing the sampling, effectively steering the tree growth towards desired areas. Time complexity of RRT is $O(n^2)$ and memory complexity $O(n)$ where n is the number of nodes in the tree. Time needed for an iteration of RRT is linearly dependent on the number of nodes in the tree as seen in Figure 2.8. The tree always remains connected, even though the number of edges is minimal. An RRT can be considered as a path planning module, which can be adapted and incorporated into a wide variety of planning systems easily.

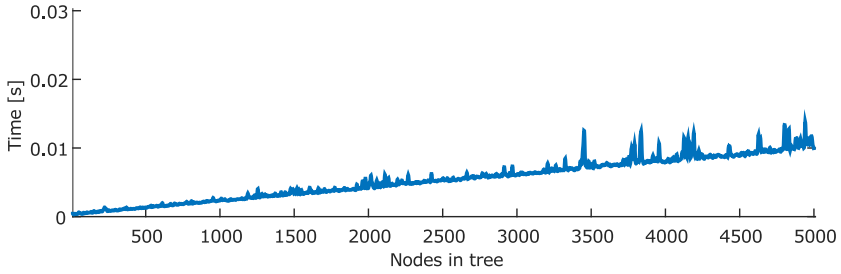
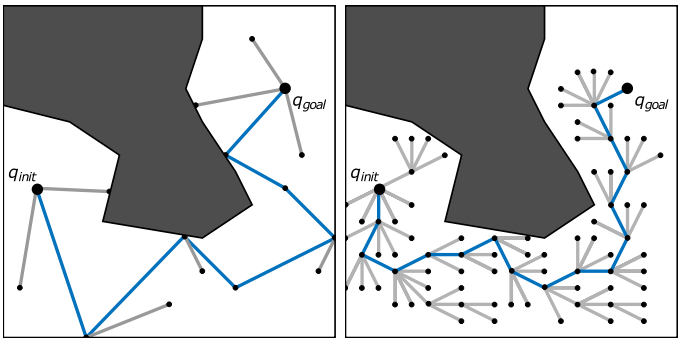


Figure 2.8: Linear dependency of iteration length on nodes in the tree

A few variations of the basic RRT algorithm exist that improve the behavior of the tree near narrow passages. First algorithm worth mentioning is *RRT-Connect* (Figure 2.9a), which is based on two ideas. The first idea is to extend the tree in the direction of q_{rand} until the tree reaches either q_{rand} or an obstacle. The second idea of RRT-Connect is that it builds two trees - one from q_{init} and one from q_{goal}. In [18] has been found, that performance can be improved by attempting to grow the trees towards each other. The advantage of RRT-Connect with two trees over basic RRT is a faster rate of convergence, however it cannot be used for problems with differential constraints due to the difficulty of connecting the trees while satisfying the differential constraints and as such is not applicable to motion planning problem in this thesis.

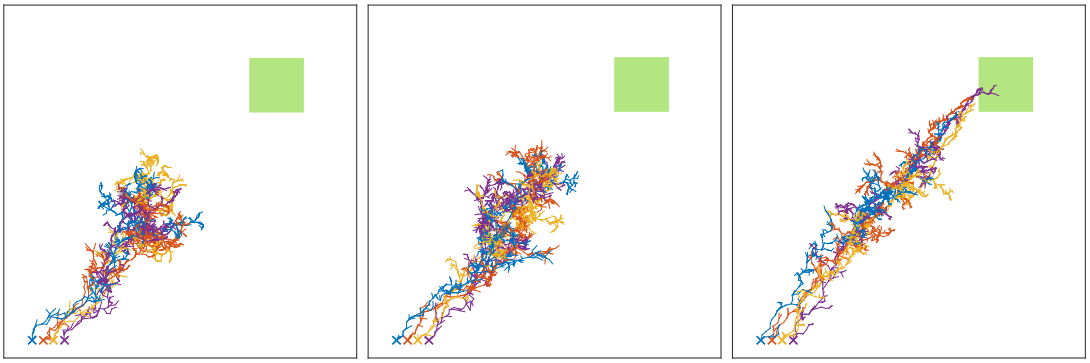
Another possible modification is *RRT-Blossom* (Figure 2.9b). Instead of expanding q_{near} with the control input sample that results in a q_{new} that is nearest to q_{rand}, it expands all control input samples that do not lead to regression. RRT-Blossom is further explained in [19].

The direction of growth of an RRT can be influenced by modifying the distribution of samples randomly drawn from the configuration space. A common



(a) RRT-Connect (b) RRT-Blossom

Figure 2.9: RRT modifications



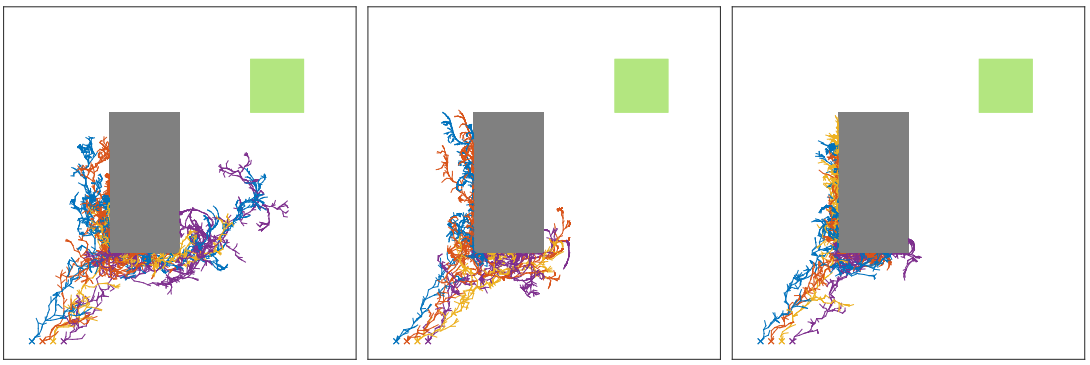
(a) Goal bias 0 % (b) Goal bias 1 % (c) Goal bias 10 %

Figure 2.10: Tree development after 1000 iterations with 4 UAVs

modification of the distribution is adding a goal bias - certain probability $p(goal)$ that the goal state q_{goal} will be used instead of a random sample q_{rand}. The benefit of this approach is a faster rate of convergence, as the tree growth is faster in the direction of q_{goal} (Figure 2.10), however the tree is more likely to get stuck near an obstacle (Figure 2.11) that prevents growth in the direction of q_{goal}. In an environment with less obstacles $p(goal)$ can be higher, while in a difficult environment with a lot of obstacles $p(goal)$ must be lower to allow RRT to find a way around them.

2.2 RRT-Path

RRT-Path is an improved version of RRT featuring preprocessing of the configuration space. Specifically the ability to maneuver around obstacles and through



(a) Goal bias 0 % (b) Goal bias 1 % (c) Goal bias 10 %

Figure 2.11: Tree development after 3000 iterations with 4 UAVs and an obstacle

narrow passages was improved by further modifying the goal bias.

Instead of using q_{goal} it takes points from a *guiding path* (Figure 2.12) which is a path from q_{init} to q_{goal} obtained from one of path planning algorithms. It is worth to note, that the guiding path is computed in the 2D/3D workspace as a geometric path. Such a path can be computed without considering the motion model. Geometric-based path planning methods based on *Visibility graph* or *Voronoi diagram* [12] can be used. Another approach is to discretize the workspace to a grid representation, where the path can be found using classic graph-search algorithms like A* or Dijkstra. In this thesis, the A* algorithm was used thanks to its desirable properties, notably optimality and performance.

Let $\mathcal{G}$ be the guiding path and $(q_{init}, q_1, q_2, ..., q_{goal}) \in \mathcal{G}$ the points of the guiding path, where $q_i \in \mathcal{C}_{free}$. In the beginning, area around the point q_1 is used instead of the random sample q_{rand} with probability $p(guided)$. If a new node of the tree is within a distance r_{dist}, then the next point of the guiding path will be used instead of q_{rand} with probability $p(guided)$. This step is repeated until q_{goal} is reached.

2.2.1 A*

To obtain the guiding path in RRT-Path, A* algorithm is used in this thesis. A* is a path-finding algorithm used to connect initial node n_{init} with a goal node n_{goal}. It belongs to the group of *best-first-search* algorithms. These algorithms expand the nodes that seem most promising in getting closer to reaching the goal node first. Drawbacks of best-first-search algorithms is that they are “greedy” which means,

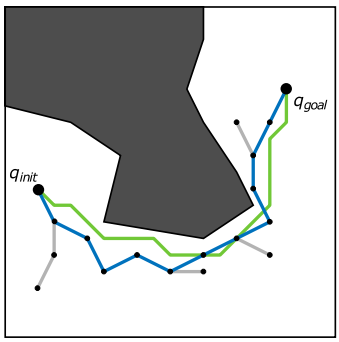


Figure 2.12: RRT-Path with green guiding path

that they are drawn to n_{goal} without taking into account the distance required to reach the currently expanded node from n_{init}, resulting in sub-optimality and non-completeness. However, A* uses a *cost function* $f(n) = g(n) + h(n)$, which is a sum of the cost from n_{init} to n and heuristic estimate of cost from n to n_{goal}, thus guaranteeing *optimality* with the condition, that its heuristic function is *admissible*.

The heuristic function $h(n)$ gives an estimate of the minimal cost from the node n to the node n_{goal}. A good heuristic function that ensures a correct behavior of A* has to satisfy the condition $\forall n \ : \ 0 \leq h(n) \leq h^*(n)$, where $h^*(n)$ is the real cost to the goal node. A heuristic function satisfying mentioned condition is called admissible and guarantees that A* finds the optimal solution. In common path-finding problems the *Euclidean distance* is often used as a heuristic function as it cannot be greater than the real distance. However, if the Euclidean distance is significantly shorter than the real distance, the speed of A* can be impaired.

A* is a complete algorithm so it will always find a solution if it exists. It is optimal assuming the heuristic function is admissible, moreover, it is also optimally efficient, therefore no optimal algorithm with the same heuristic will expand less nodes than A*. Time and memory complexity is $O(b^d)$ where b is the branching factor (how many neighbors each node has) and d is the number of nodes in the shortest path.

A* can also be used as a *distance transform* by running it with no stopping condition and n_{goal} as the initial node. Heuristic function in not used in this case so $f(n) = g(n)$. When A* iterates through all the nodes, their distance to n_{goal} will be known. In this thesis, the distance transform is used in the cost function of the *Particle Swarm Optimization* algorithm.

2.3 PSO

Particle Swarm Optimization [20] algorithm will be used to optimize the goal coverage. It belongs to the group of population-based algorithms. PSO is used to optimize multidimensional nonlinear problems while avoiding getting stuck in local optima. PSO behavior is inspired by the social behavior of bird flocks and fish schools. It relies on *swarm intelligence* of a population of particles that are iteratively guided through the configuration space by changes of their velocity vector.

Each PSO particle represents a candidate solution. The particles are initialized randomly and so are their velocities. Each particle keeps track of its position with the best *cost function*. The global best position of a particle in the whole population is remembered as well, thus representing the *social knowledge*. In each iteration the velocity vector $u_j(t)$ of every particle is updated to a combination of its previous velocity vector $u_j(t-1)$, its individual best position b_j and the global best position b_g. The rule to update the velocity vector of particle j is

$$u_j(t) = u_j(t-1) + c_1 r(b_j - p_j(t-1)) + c_2 r(b_g - p_j(t-1)), \tag{2.3}$$

where $p_j(t-1)$ is the previous position of particle j, c_1 and c_2 are learning factors, c_1 representing individuality and c_2 sociability. Random number r is from the uniform distribution between 0 and 1. The velocity vector update is followed by the position update

$$p_j(t) = p_j(t-1) + u_j(t). \tag{2.4}$$

CHAPTER

THREE

IMPLEMENTATION

3.1 Motion model

The need of a motion model comes from the demand of a feasible trajectory which is secured by building the trajectory from motions that a real UAV can follow. The real UAV is not a point in space, it has non-zero size and mass and its movement is limited by differential constraints. Two motion models were considered, first of them being a full quadrotor motion model whose outputs are moments of the UAV. When this representation is used in simulations, the behavior of the simulated UAV is close to the actual UAV. However, if another UAV with different parameters (mass, size, torque of rotors) is used, the motion model has to be updated accordingly. The second motion model provides no information about the UAV moments, therefore it is not dependent on the actual parameters of the UAV, making it more universal.

3.1.1 Quad-rotor motion model

A quadrotor consists of two pairs of rotors rotating in the opposite direction generating a thrust and torque normal to the plane determined by the four rotors. Thanks to this simple mechanical structure it offers great maneuverability including VTOL (Vertical Take-Off and Landing). The control of a quadrotor state originates in the amount of torque generated by each one of its rotors. When each of the rotors generate the same torque, the quadrotor will stay in place, however, if one of the rotors changes its torque, the quadrotor starts to move.

In Figure 3.1 one can see an inertial reference frame $\{\vec{i}_1, \vec{i}_2, \vec{i}_3\}$ and body-fixed frame $\{\vec{b}_1, \vec{b}_2, \vec{b}_3\}$ with origin in the center of mass of the quadrotor. In [21] is defined:

$m \in \mathbb{R}$	the total mass
$J \in \mathbb{R}^{3\times3}$	the inertia matrix with respect to the body-fixed frame
$R \in SO(3)$	the rotation matrix from the body-fixed frame to the inertial frame
$\Omega \in \mathbb{R}^3$	the angular velocity in the body-fixed frame
$x \in \mathbb{R}^3$	the location of the center of mass in the inertial frame
$v \in \mathbb{R}^3$	the velocity of the center of mass in the inertial frame
$d \in \mathbb{R}$	the distance from the center of mass to the center of each rotor in the $\vec{b_1}$, $\vec{b_2}$ plane
$f_i \in \mathbb{R}$	the thrust generated by the i-th propeller along the $-\vec{b_3}$ axis
$\tau_i = \mathbb{R}$	the torque generated by the i-th propeller about the $\vec{b_3}$ axis
$f \in \mathbb{R}$	the total thrust, i.e., $f = \sum_{i=1}^{4} f_i$
$M \in \mathbb{R}^3$	the total moment in the body-fixed frame

The equations of motion of this quadrotor are

$$\begin{aligned} m\dot{v} &= mge_3 - fRe_3 \\ \dot{R} &= R \\ J\dot{\Omega} + \Omega \times J\Omega &= M, \end{aligned} \tag{3.1}$$

where the hat map $\hat{\cdot} : \mathbb{R}^3 \to SO(3)$ is defined by the condition that $\hat{x}y = x \times y$ for all $x, y \in \mathbb{R}^3$. $SO(3)$ is a group of all rotations about the origin of Euclidean space $\mathbb{R}^3$.

This motion model offers precise control of the location and velocity (inputs) of an UAV by producing exact values of moments in any point in time, nevertheless these are not needed for a trajectory planning task and their computation requires too much processor time, thus slowing the simulation. The idea to use a simple, easy to compute motion model to be able to perform more iterations in the same time was born.

3.1.2 Car-like motion model

Car-like motion model is used for mobile robots moving in a two dimensional space, so why should it been considered for three dimensional motion planning of

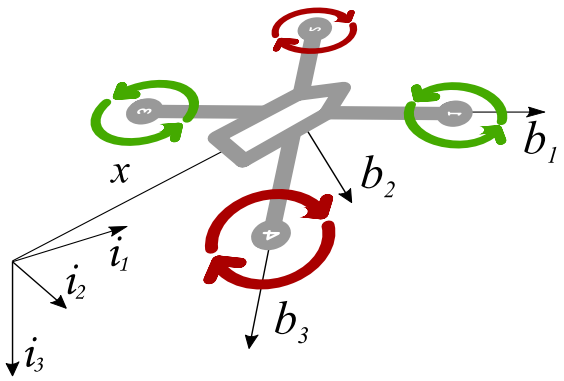
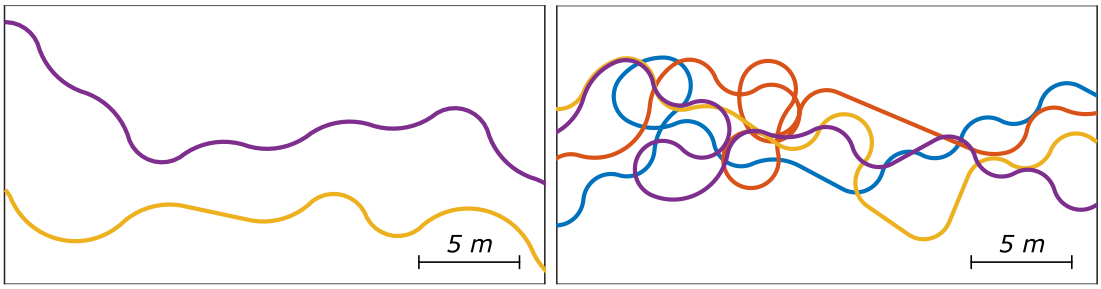


Figure 3.1: Motion model of a quadrotor with body-fixed frame and inertial reference frame

quadrotors? The answer lies in the fact, that the quadrotors are unable to fly in the space above each other due to the aerodynamic effects, moreover in this case they have to keep almost the same altitude [9] because of the relative localization system. These constraints put on the flight of UAVs present a valid reason to treat their motion planning as a two dimensional problem, where the flight altitude is specified by an experienced operator based on the character of monitored terrain and visibility conditions. Assuming that flight altitude is constant, the three dimensional planning problem is converted to a two dimensional one, so motion models of ground robots like the one in Figure 2.2 can be considered. The resulting trajectory is smooth with no sharp turns as seen in Figure 3.2, making it easy to follow the trajectory by a real UAV.



(a) A simple trajectory of two UAVs

(b) A trajectory of four UAVs circling on one spot for a moment due to localization constraints

Figure 3.2: Shape of the trajectories generated by the car-like motion model

A useful case of a motion model for ground robots is the one of a simple car. The car is a rigid body moving through space. Its configuration is denoted by

$q = (x, y, \varphi)$. The origin of the body frame is located in the center of the car, with the x axis pointing along the main axis of the car and y axis parallel to it. The configuration transition equation from [12] is

$$\begin{aligned} \dot{x} &= u_s \cos\varphi \\ \dot{y} &= u_s \sin\varphi \\ \dot{\varphi} &= \frac{u_s}{L} \tan u_\varphi, \end{aligned} \tag{3.2}$$

where u_s is a forward speed, u_φ is a steering angle and L is the size of the robot (Figure 3.3). The trajectory found with this motion model is made of smooth curves making it easy to follow by a real UAV. Now that the behavior of individual UAVs is defined, it remains to define the rules of the swarm.

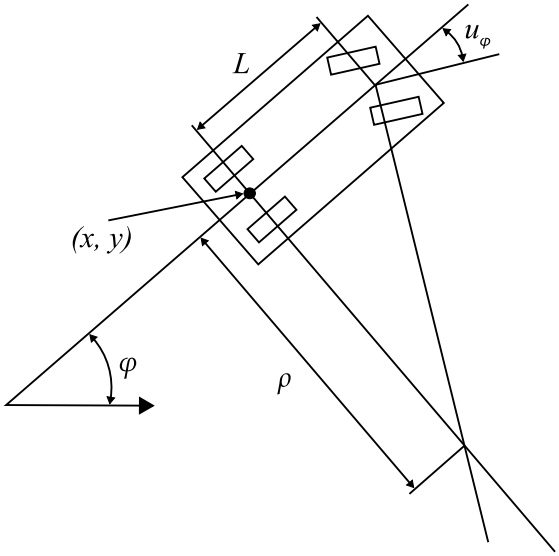


Figure 3.3: Car-like motion model

3.2 Relative localization

Individual members of the swarm have to be aware of their neighbors in order to remain in the swarm together with other UAVs. A relative localization technique from [9] is used in this thesis and is described in Chapter 2. After choosing appropriate localization technique it remains to define the swarm rules, which directly affect the formation and behavior of the swarm. To keep a swarm of n UAVs together it is necessary, that each UAV has $n-1$ neighbors in its localization

range at all times. In some cases, like multiple AoIs, that cannot be covered by a swarm without breaking the localization rules due to the distance between these areas, it might be beneficial to allow the swarm to split and cover more AoIs.

A weaker swarm constraint is, that each UAV has at least 1 neighbor in its localization range, effectively producing swarm splitting into pairs when appropriate. Of course, the condition of minimal distance must be satisfied for all neighbors to prevent collisions. This constraint is used throughout this work, thanks to the minimal limitation it represents for the planning algorithms, resulting in shorter time needed to expand a node, so more configurations are produced in the same time, while still maintaining the behavior of a swarm.

3.3 Cost function

All motion planning algorithms rely on evaluating reached states to decide whether the goal state is reached or which one of available nodes to expand. The value of a cost function should objectively reflect the quality of individual states. It is possible to use a simple binary cost function returning 1, when an UAV is above an AoI and 0 everywhere else. However, one needs to keep in mind, that the cost function is meant to be optimized to find the optimal coverage and optimizing a flat function with occasional peaks is not an easy task, therefore a need for a more sophisticated cost function arises.

The cost function $CF(q)$ proposed in this thesis measures the information not captured by on-board cameras of UAV swarm in configuration q, i.e., configuration q_1 is better than q_2 if $CF(q_1) < CF(q_2)$. The calculation is based on AoI and UAVs fields of view.

The world $\mathcal{W} = \mathbb{R}^2$ is mapped into an AoI Map matrix $A \in \mathbb{R}^{a \times b}$ whose element $A_{x,y}$ (a cell at position x and y respectively) has a size of $1\,m^2$ and its value is the amount of information in the area as seen in Figure 3.5. AoI is represented by value up to A_{max} and everything else including obstacles by value 0. This approach permits representing AoIs with higher or lower importance by assigning them a higher or lower values respectively.

The picture quality, shape and size of the area captured by the on-board camera of the UAV depends on many factors, some of them being time of the day, weather conditions, flight altitude, camera chip resolution, lens parameters, stabilization, frame rate, etc. To keep the cost function as universal as possible

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

Figure 3.4: AoI matrix with with a green AoI region, $A_{max} = 100$, and a blue region representing the area covered by an UAV

let us assume, that the area viewed by each UAV is a rectangle (Figure 3.5) whose dimensions are given by the image size of the on-board cameras.

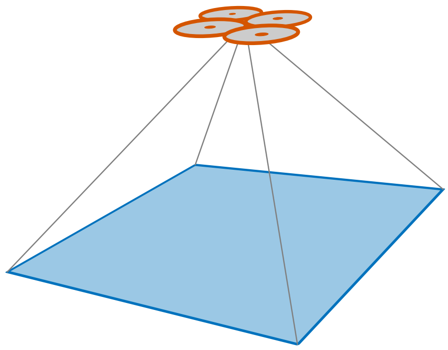


Figure 3.5: The area viewed by a camera mounted on each UAV

Another important decision is whether coverage of an area by multiple UAVs brings any additional information compared to a single UAV. The fact, that every UAV can view the same area from different angle, adding more information especially in areas with many objects, and the above mentioned factors influencing the picture quality favors the former case. However, a part of the information gained by more UAVs covering the same area is redundant, so covering larger area should be prioritized over covering the same area with more UAVs.

The designed cost function reflects the area coverage redundancy by dividing the value of covered part of A by 2 for each UAV covering the area:

$$CF(q) = \sum_{x=1}^{a} \sum_{y=1}^{b} \left(\frac{A_{x,y}}{2^n} \right), \tag{3.3}$$

where n is the number of UAVs covering area $A_{x,y}$.

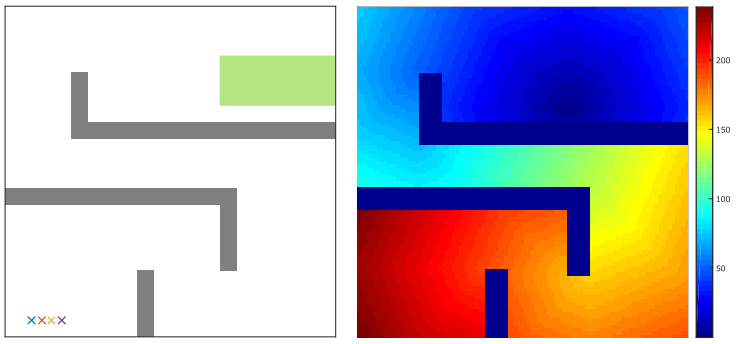
In article [10], a similar cost function was used with the UAV altitude taken into account:

$$CF(q) = \sum_{x=1}^{a} \sum_{y=1}^{b} min \left(0,\ A_{x,y} - \sum_{i \in Swarm(x,y)} \frac{S_{opt}}{S_i} A_{max} \right), \tag{3.4}$$

where S_i is area of the rectangle representing the part of the workspace that can be observed by the i-th UAV. S_{opt} is area of the region, which is observed by UAV flying at the altitude determined as the "optimal" altitude based on the particular application. However, after implementing PSO in this thesis, the results produced with this cost function were unsatisfying so an improvement is proposed in this thesis.

The problem with the cost function from 3.4 is, that it is hard to optimize, i.e., until a particle hits the AoI region, the movement of whole population is random, since the value of cost function is still the same. The proposed improvement is to add information about the distance from AoI obtained by the distance transform (Figure 3.6) mentioned in 2.2.1 to the cost function, which results in a more gradual optimization from the first iteration. The distance has to be weighted by α as it is only a helping information and must not influence optimization of the original cost function 3.3. The resulting cost function for PSO used in this thesis is

$$CF(q) = \sum_{x=1}^{a} \sum_{y=1}^{b} \left(\frac{A_{x,y}}{2^n} \right) + \alpha \cdot d(q,\ q_{goal}). \tag{3.5}$$



(a) Initial configuration space (b) Distance transform

Figure 3.6: Graphically represented discretization of configuration space with a distance transform, where red color means furthest from goal

3.4 RRT

The core functionality of this work is implemented by RRT. The simple structure of Algorithm 1 is preserved and details of individual steps will be explained. First of all, each configuration q contains positions of all UAVs in the swarm, therefore $q = (x_1, y_1, x_2, y_2, ..., x_n, y_n)$, where x_i, y_i are positions of i-th UAV. The variable z, representing UAV altitude, is not used, as the swarm is flying at a constant altitude.

On the first line, the tree is initialized by preparing an array of empty nodes and adding the initial node, also called root, to the tree. The nodes are containing configuration variables and other relevant variables: velocity, index of previous node for path reconstruction and inputs used to reach the node from previous node.

The main loop of RRT building starts on line 2. The loop runs until a desired number of iterations K is achieved. When the goal is to find a trajectory to AoI and optimization of the covered area is required, RRT growth is stopped after finding a node in which all the UAVs are within the boundaries of any AoI present on the map.

On the third line a random (or goal biased) sample q_{rand} from the configuration space is drawn. A node from the tree that is nearest to q_{rand} is selected as the nearest neighbor for expansion on line 4. The metric for measuring distance of nodes in this step is the Euclidean distance

$$d(p, q) = \sqrt{\sum_{i=1}^{n}(q_i - p_i)}. \tag{3.6}$$

As the distance $d(q, p)$ is used to compare distance of several configurations to a single one (q_{rand}), the comparison can be realized on the squared version of the distance, which is significantly faster.

Line 5 is the expansion step. A set of control inputs looks as follows:

$$\mathcal{U}_{carlike} = \{(u_s, u_\varphi)\}, \tag{3.7}$$

where $u_s \in \mathcal{U}_s$, where $\mathcal{U}_s$ is a set of K discrete values for input u_s in range from $u_{s,min}$ to $u_{s,max}$. Similarly for u_φ. If each input had values from range $\{-1, 0, 1\}$,

then a set of meaningful inputs is

$$\mathcal{U}_{carlike} = \{(-1,-1),\ (-1,0),\ (-1,1),\ (1,-1),\ (1,0),\ (1,1)\}. \tag{3.8}$$

The number of control inputs could be changed dynamically depending on a near obstacle heuristic, however it was found experimentally, that 5 samples of u_s and 9 samples of u_φ is enough for most problems. The control inputs are fed to the motion model, which returns a set of temporary configurations after applying the inputs. From this set, the configuration that is nearest to q_{rand} and satisfies all constraints (collision avoidance and relative localization) is added to the tree as q_{new} on line 7. If no input satisfies the constraints, then the next nearest node will be the new nearest neighbor. This mechanism helps to prevent getting stuck near an obstacle. The tree consists only from the nodes, the information about edges is replaced by inputs used for reaching the node.

As RRT is probabilistically complete, a trajectory is eventually found in an infinite time, however, in practical realization infinite time is not available, therefore the rate of convergence needs an improvement. To help RRT with finding trajectories to AoIs a fast heuristic based on a guiding path was implemented.

3.5 RRT-Path

RRT-Path uses a fast heuristic to preprocess the configuration space. The initial part of RRT-Path is the preparation of the guiding path. First, the whole configuration space is discretized into a matrix of cells. The size of the cell affects the ability to find a guiding path through a narrow passage. The smaller distance between obstacles, the higher the resolution should be. However, the exponential time complexity of A* has to be kept in mind. Each cell contains a value depending on the object located in it. Free space C_{free} is represented by 0, obstacles by 1, depot place (initial position of UAVs) by 2 and AoI 3 as in Figure 3.7. Second, A* is run on the map grid from a cell with value 2 to an area containing cells with value 3. If more than one AoI is present, then A* is run to all of them.

The next part of RRT-Path is running RRT with modified $randomState()$ function on line 3. Instead of drawing a random sample from $\mathcal{C}$, a sample in a *near neighborhood* of a point q_{path} from the guiding path is drawn with probability p_{guided}. The value of p_{guided} is a parameter for tuning, since no exact formula

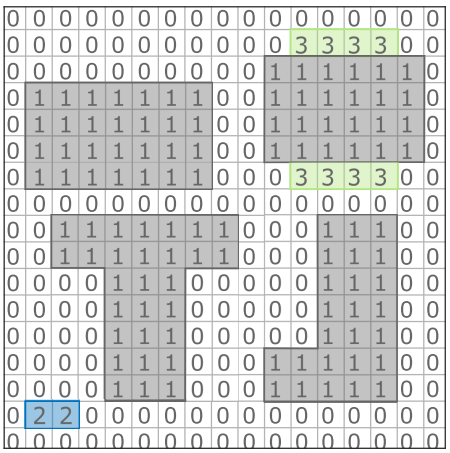
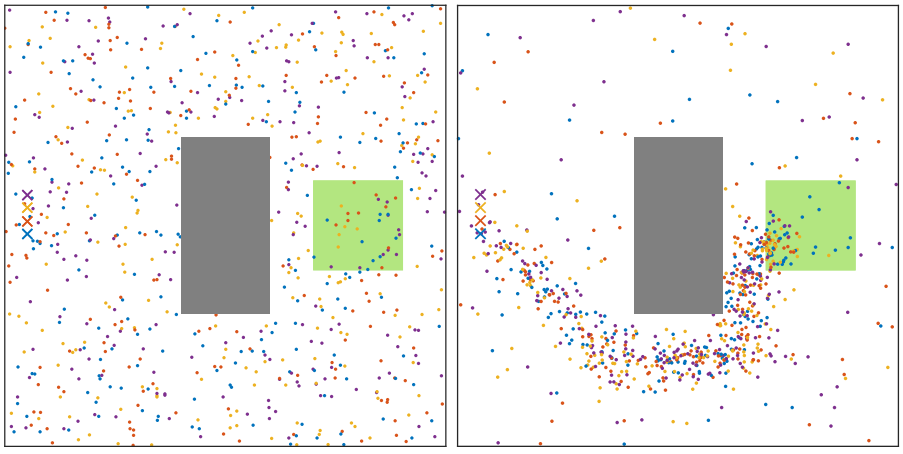


Figure 3.7: Discretized grid of configuration space

for it exists. Higher values result in a faster rate of convergence, but also higher probability of getting trapped in a highly constrained area. Usually $p_{guided} = 0.8$ presented satisfying results. Near neighborhood of a point is the area of an circle with center in the point and radius r_{near}. After an UAV reaches the near neighborhood of q_{path}, next point from the guiding path will be its q_{path}. The difference between non-biased RRT sampling and RRT-Path sampling is illustrated in Figure 3.8.



(a) RRT sampling distribution (b) RRT-Path sampling distribution

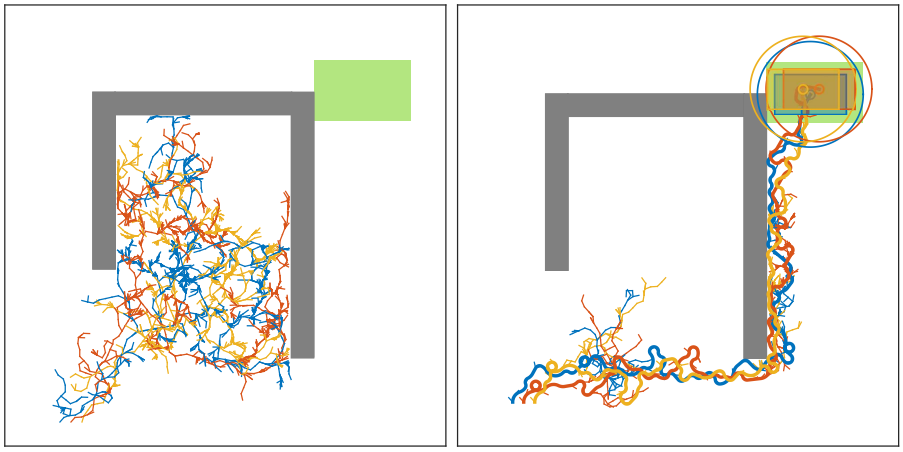
Figure 3.8: Sampling distribution of basic RRT and RRT-Path after 400 iterations

Algorithm 2: RRT-Path pseudocode

```
1  τ.init(q_init);
2  mapGrid ← discretizeMap();
3  guidingPath ← AStar(q_init, q_goal, mapGrid);
4  q_path ← guidingPath.first;
5  for k ← 1 to K do
6      if rand()<p_guided then
7          q_rand ← randomStateGuided(q_path, r_near);
8      else
9          q_rand ← randomState();
10     while feasible expansion not found do
11         q_near ← nearestNeigbor(q_rand, τ);
12         u ← selectInput(q_rand, q_near);
13         q_new ← newState(q_near, u);
14     τ.addNode(q_new);
15     if isWithinRadius(q_new, q_path, r_near) then
16         q_path ← guidingPath.next;
17     if goalReached ← checkNearGoal(q_new) then
18         break;
19 return τ
```

This approach encourages *swarm splitting* when more AoIs are present, as the only condition keeping it together is the relative localization. Other approach is, that q_{path} is common for all UAVs. Then, however, the swarm splits less often and some AoIs remain uncovered. The benefit is faster encountering of the first solution, although the solution is far from optimal. The benefits of these two approaches can be combined by specifying which UAVs should go to which AoIs in the beginning by the security operator, however, this may be a difficult task if several goals and AoIs exist. The swarm splitting tendency is further examined in Chapter 4.

The trade-off for faster convergence and improved behavior around narrow passages is that unlike RRT and PSO, RRT-Path requires the knowledge of the AoI location a priori, however, this is not a big problem as the area planned to be monitored is usually known. Another drawback is, that the guiding path is



(a) Trapped RRT (b) RRT-Path reached the AoI

Figure 3.9: RRT got trapped in an U-shaped obstacle, while RRT-Path successfully avoided the obstacle

found in a geometric approximation of the configuration space, where the UAV is just a point in a multidimensional space, therefore the guiding path might be impossible to follow by a swarm of real UAVs. The last complication are the two parameters, p_{guided} and r_{near}, that control the behavior of the algorithm. These parameters have to be tuned for each situation individually, because what worked well with one map and 4 quadrotors, might not work well with another map and 10 quadrotors. We found experimentally, that values $p_{guided} = 0.8$ and $r_{near} = 80$ work quite well for most maps. In Figure 3.9 a situation when the growth of the RRT tree gets stuck in an U shape obstacle can be seen, as well as the valuable behavior of RRT-Path resulting in a complete avoidance of the inside area of the U-shape.

3.6 Surveillance using PSO

Particle Swarm Optimization algorithm [10] was implemented for comparison with developed RRT method. Each particle j represents a configuration of the whole swarm, velocity vector $u_j(t)$ contains velocities of each UAV in the swarm obtained from the motion model. Whole PSO population as used in [10] can be seen in Figure 3.10.

The cost function 3.4 from [10] used to determine the quality of position of each particle was initially the cost function evaluating AoI coverage, however, it

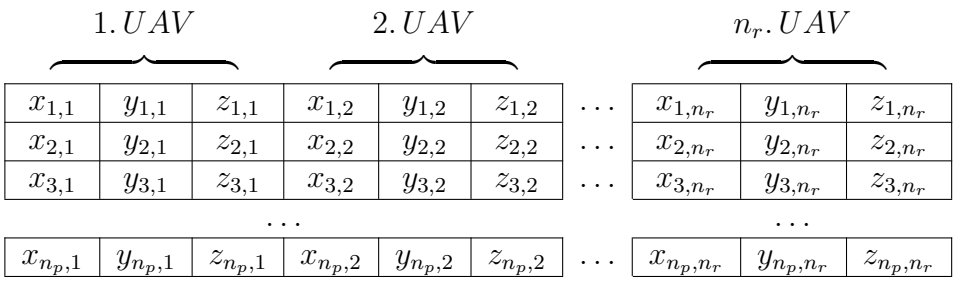


Figure 3.10: Scheme of the PSO population, each row representing one particle

was found to not work properly with PSO, as the cost function bears no information about the distance from AoI, so PSO particles were moving randomly over the scene until accidentally hitting the AoI area. This produced poor results with a few obstacles and with higher number of obstacles the solution was not found at all, because particles got stuck in the obstacles. Euler distance was another candidate for cost function, which improved convergence, but did not solve the problem with obstacles. Final solution was to use a distance transform, providing information about the distance to AoI while considering obstacles in the way. The distance transform based cost function of configuration q looks as follows:

$$CF(q) = d(q,\, q_{goal}), \tag{3.9}$$

where $d(q,\, q_{goal})$ is the distance between q and q_{goal} as returned by the distance transform.

Despite using the same motion model as in the case of RRT, PSO tends to produce longer trajectories, because UAVs change direction quickly as new global best positions are found.

3.7 Coverage optimization

After a trajectory to AoI is found, an optimization takes place to minimize the value of cost function, thus guaranteeing maximal possible area covered. Two methods were used for optimization in this thesis. Even though RRT is not a true optimization method, due to the fact, that it does not guarantee optimality, it was used to optimize the cost function, because in practical tasks optimality is often not required and a solution that is close to optimal is sufficient. The second method is PSO which is a true optimization method.

Optimization starts with UAVs located somewhere over the AoI, so the bounds of the scene can be shrunk to the edges delimiting the AoI. Now that the scene is much smaller, it is necessary to slow the movement of UAVs, so that they do not fly from one border to another. Both proposed methods use the same motion model, so the velocity can be modified by control input u_φ. AoIs can be of different size so instead of using constant value of control input, a fraction of the shorter edge of the AoI is used. That way a consistent behavior can be expected no matter the size of AoI.

After shrinking the scene and constraining the velocity, optimization can start. In each iteration, the cost function is checked and compared to the best cost function so far. Optimization can be stopped when enough iterations are carried out, or when a certain value of cost function is reached, however, if the cost function value cannot be reached, the algorithm would never finish.

The difference between optimization with the use of RRT and PSO is subject to experimental comparison in Chapter 4.

CHAPTER

FOUR

EXPERIMENTS AND RESULTS

4.1 Comparison of planning algorithms

The goal of this experiment is to compare three trajectory planning algorithms: RRT, RRT-Path and PSO. The chosen experiment is a simple task of autonomous surveillance in a large environment. In this environment, the AoI is located far from the depot with one obstacle between them. Therefore, the main challenge in this scenario is to compute trajectories of the whole UAV swarm to the AoI. The optimization part (finding optimal sensing locations of the individual UAVs) can be solved by "Final optimization" after the trajectory towards AoI is found. The experiment therefore primarily aims to investigate performance of the tested methods to find trajectories for a swarm of UAVs towards a goal area represented by the AoI. We can expect, that two tested methods (RRT and RRT-Path) are superior to PSO. The situation is pictured in Figure 4.1. For the sake of simplicity only two UAVs are deployed. All algorithms use the same motion model (car-like), relative localization and collision detection. PSO uses forty particles.

To obtain statistically relevant information, each algorithm was run 100 times. The algorithm run until the AoI was reached (positions of all UAVs were within the AoI boundaries) with a time limit of 1800 s. The monitored quality is the number of iterations needed to find a solution and the time spent. Examples of trajectories found by all tested algorithms can be seen in Figure 4.2.

In Table 4.1 experiment results are presented. RRT performed rather poorly needing 565 s to find a solution and failing to fulfill the time limit in more than a quarter of runs. On the other hand, RRT-Path found a solution in half a minute.

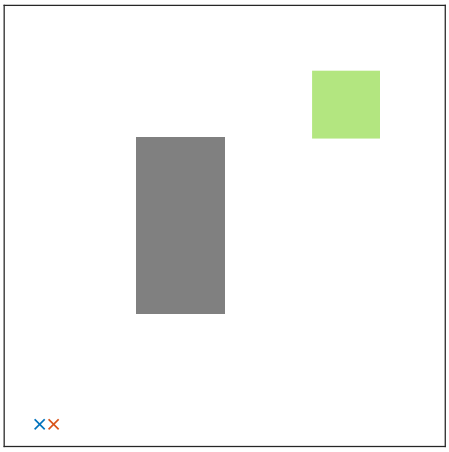


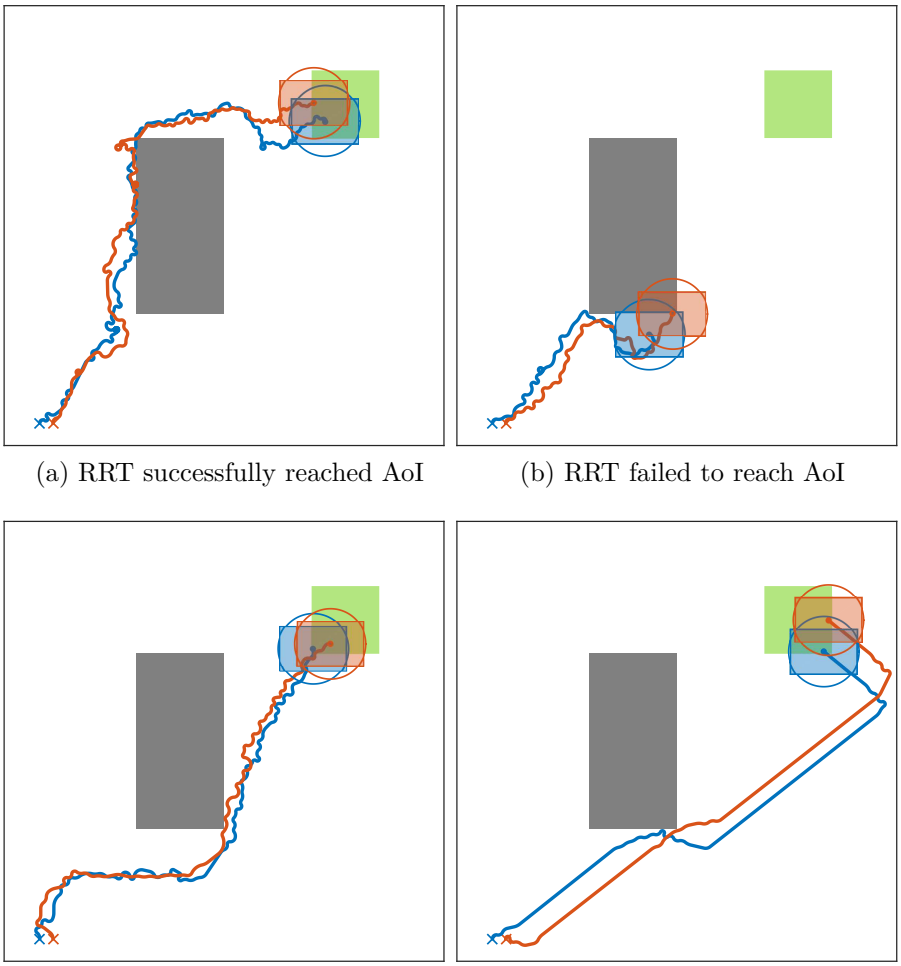
Figure 4.1: Environment with one goal and one obstacle

	RRT	RRT-Path	PSO
Solution found in % runs	73 %	100 %	100 %
Trajectory length [nodes]	119	98	86
Iterations	6150	860	53
Time spent [s]	565	31	83

Table 4.1: Table of trajectory planning results from 100 runs in a simple environment (trajectory length, iterations and time spent are median values)

PSO reached a solution in the least iterations, however, the time to do so was nearly three time higher than time of RRT-Path. This disproportion between iterations and time spent is caused by the number of particles whose position is updated in each iteration of PSO.

Poor performance of RRT is surprising. The task should fit the purpose of RRT, as there is no optimization involved so it is a regular planning problem. The result is so bad due to a combination of more factors. First, the location of AoI in the right corner is unsuitable for RRT, since the observed behavior is fast growth to the center of the map in the beginning and slow growth toward all the corners afterwards. If the AoI was located in the center of the map, RRT would reach it faster. Second, if the tree reaches an obstacle, the growth towards the area behind it is stopped, until the tree finds a way around the obstacle. Number of iterations needed to find a way around an obstacle depends on its shape and depot-obstacle and obstacle-AoI area ratio (Figure 4.3). The ratio affects the probability of a random sample drawn in front of the obstacle, thus contributing to finding a way around the obstacle.



(a) RRT successfully reached AoI

(b) RRT failed to reach AoI

(c) RRT-Path

(d) PSO

Figure 4.2: Trajectories found by tested algorithms with a time limit of 1800 s, with one example in which RRT got stuck in an obstacle and failed to reach the AoI in the time limit

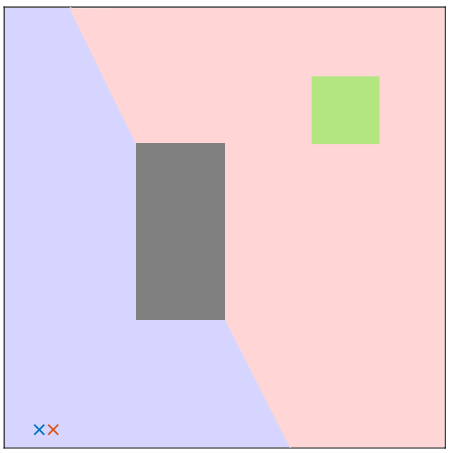


Figure 4.3: Random samples in the blue region contribute to maneuver around the obstacle when the tree gets stuck in the obstacle

It is worth mentioning, that this experiment is one of the simplest possible, nevertheless RRT failed to present satisfying results because of the reasons mention in previous paragraph, so it will not be examined in following trajectory planning experiments.

4.2 Comparison in a more complicated environment

Now that we know that RRT-Path and PSO work well in solving a trivial task with one obstacle, the ability to deal with a more constrained environment should be explored. The map used for this experiment is a maze from obstacles (Figure 4.4), so occasional failures due to the swarm getting stuck in an obstacle and long run times are expected. A swarm of four UAVs was used in this experiment to complicate the experiment a little more to fully examine the performance of both algorithms. Theoretically, RRT-Path should be able to find a solution faster thanks to its guiding path, however, RRT-Path is also more prone to getting stuck in an obstacle.

The results from the experiment presented in Table 4.2 are rather surprising. RRT-Path succeeded in finding a feasible trajectory according to expectations in 75 % cases, where in the remaining 25 % the swarm got stuck near an edge of an obstacle which is a common problem when the guiding path is passing too close to obstacles. The median time of 126 is an excellent result for a relatively complicated

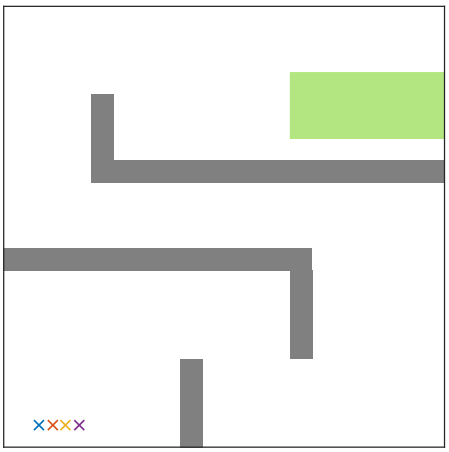
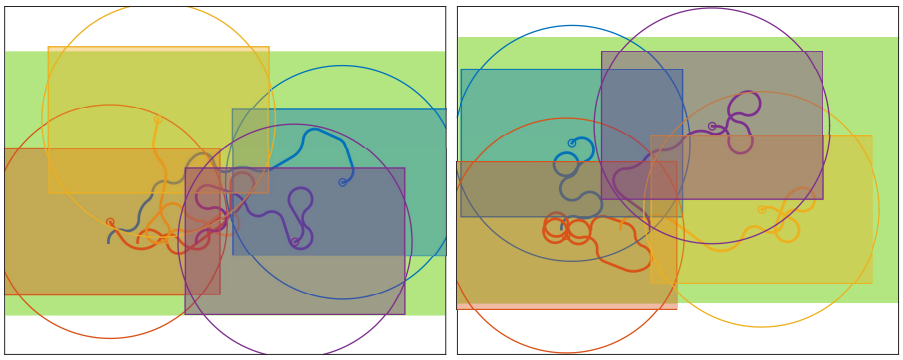


Figure 4.4: A more complicated environment in the form of a maze

	RRT-Path	PSO
Solution found in % runs	75 %	8 %
Trajectory length [nodes]	298	772
Iterations	1362	1000
Time spent [s]	126	962

Table 4.2: Table of trajectory planning results from 100 runs in a complicated environment (trajectory length, iterations and time spent are median values)



(a) RRT (b) PSO

Figure 4.5: Example of 85% AoI coverage

map. In contrast PSO totally failed at this map with only 8 % successful runs. The reason for this result is, that PSO has some tunable parameters and every situation requires different parameters. The settings in this case (20 particles, 1000 iterations, $c_1 = 2$, $c_2 = 2.5$) are not appropriate, even though they were working fine on other maps. The necessity to manually set PSO parameters is its main drawback, as they need to be found experimentally.

4.3 Coverage optimization

In this experiment, the final optimization of AoI coverage is analyzed. The goal is to verify the possibility of using RRT as an optimization method in the task of autonomous surveillance. PSO [20] is used as a reference optimization method for comparison. The first part of the autonomous surveillance (finding a trajectory to the AoI) is not considered in this experiment in order to compare only optimization methods without other influences. The trajectory planning ends when all UAVs are located inside the boundaries of AoI and this is where the optimization begins.

Optimization starts with a swarm of four UAVs within relative localization constraints distributed near a corner of a rectangular AoI 300 m long and 180 m wide. Measured values are run time, number of iterations and trajectory length when 85 % coverage of the AoI is reached, an example of such coverage can be seen in Figure 4.5. Each algorithm was run 100 times and statistically processed. The expected result is, that PSO will find UAV positions covering the required part of AoI in less time with long trajectories and that RRT will find short trajectories in longer time.

	RRT	PSO
Trajectory length [nodes]	14	93
Iterations	801	93
Time spent [s]	18	34

Table 4.3: Median values from 100 runs of optimization algorithms

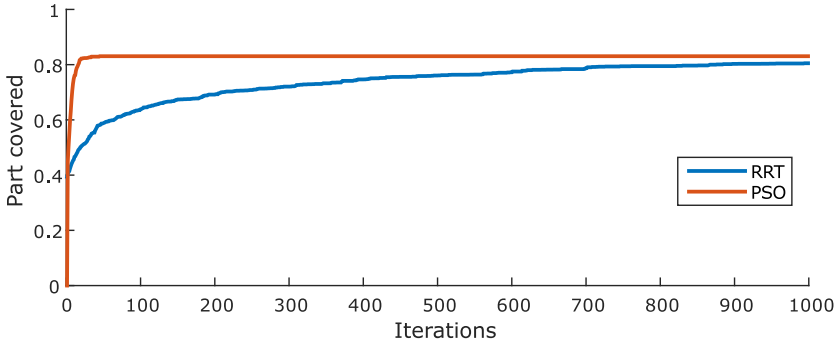


Figure 4.6: Convergence of the covered part of AoI

The results can be seen in Table 4.3. Trajectory length of RRT is nearly seven times shorter than trajectory of PSO as expected. However, surprising result is, that RRT also needed only half of the time of PSO to reach the required coverage, even though it needed nearly nine times more iterations (Figure 4.6). This interesting observation may come from the fact, that PSO has to manage twenty particles in a single iteration. RRT needs 0.0225 s to execute one iteration, while PSO needs 0.3656 s. However, if we divide the time PSO needs for one iteration by the number of particles, we get a time of 0.0183 s for one iteration of one particle. When compared in this way, PSO is a little faster, probably because of nearest neighbor search of RRT, since all other demanding processes (motion model, collision detection, relative localization, cost function) are common to both RRT and PSO.

The conclusion of this experiment is, that RRT can be successfully used in the task of autonomous surveillance to optimize the coverage of AoI, even though it is not a true optimization method. RRT outperforms PSO in the trajectory length and even run time.

4.4 Sampling method in RRT-Path

The method of drawing random samples around the guiding of RRT-Path has major impact on behavior of the algorithm. In addition to p_{guided} and r_{near}, the way how the point q_{guided} is chosen also influences the behavior. This experiment examines three ideas of choosing q_{guided}. The major difference can be observed in planning problems with more AoIs, as each AoI has its own guiding path from initial depot.

First idea (sampling A) is to change only the part of q_{guided} corresponding with the UAV that reached the point in guiding path to the first unreached point in a randomly chosen guiding path. That way every UAV has a different goal point from a different guiding path and the only constraint keeping the swarm together is relative localization. This sampling concept promises frequent splitting of the swarm when more AoIs must be covered, however, it is expected that the trajectory planning will need more time, since UAVs closer to the AoI have to wait for slower UAVs behind them.

Second sampling method (sampling B) proposes choosing q_{guided} from a randomly chosen guiding path, that is common for all UAVs from the swarm. All UAVs have the same goal point as the UAV closest to the goal, which should encourage faster movement of UAVs furthest from goal, thus improving the time in which the trajectory is found. The drawback of this attitude is, that when cornering around an obstacle the UAVs in the rear can get stuck at the obstacle trying to reach q_{guided} behind a corner of the obstacle.

Third possibility (sampling C) is to assign swarms of UAVs to guiding paths manually, consequently, the guiding path is predefined for each swarm. Number of UAVs going to each AoI can be controlled by an operator who evaluates the surveillance task and assigns swarms accordingly. Expected are low times of planning with AoIs covered as specified by the operator. This approach should offer the best results at the cost of more work for the operator, who might not be able to predict optimal swarm splitting.

The initial depot is located in the center of the map (Figure 4.7), with two AoIs located on edges of map boundaries and four obstacles blocking the way. The map should test the swarm splitting behavior and the ability to navigate UAVs around obstacles.

The results from 100 runs of this experiment are in Table 4.4. The experiment

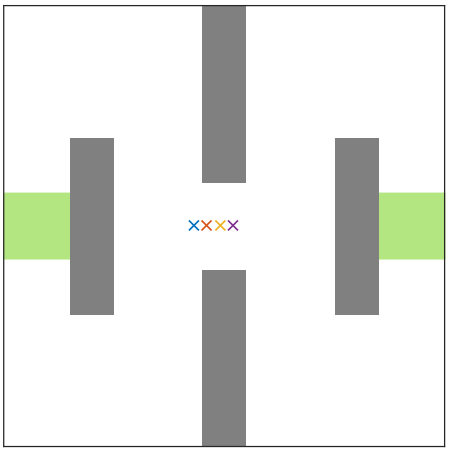


Figure 4.7: Experiment scene with four UAVs, two AoIs and four obstacles for sampling methods analysis

	Method A	Method B	Method C
Goals covered	1.64	1.31	1.98
Iterations	1647	258	132
Time spent [s]	117.42	7.32	4.28
Iterations per second	14.03	35.25	30.84
Trajectory length [nodes]	55	54	55

Table 4.4: Median values from 100 runs with different sampling methods (Goals covered is mean value)

produced expected results with sampling method C being the best in all monitored values. Average goals covered is 1.98 only because in one iteration, the algorithm failed to find a solution in the limit of 5000 iterations. Sampling method A managed to cover 1.64 goals in average in contrast to only 1.31 in sampling method B which proves the swarm splitting tendency of method A at the cost of 16 times longer run time. Values of iterations per second show, that sampling method A is the most time demanding, as other two method execute more than twice as much iterations per second. The higher time demand is caused by frequent violation of relative localization rules, when each UAV tries to fly along a different guiding path. The sampling method has no impact on the length of found trajectory as seen in the last row of table.

The outcome of this experiment is, that when manual assignment of UAVs to guiding paths is not possible or desired and more AoIs are present, having each UAV follow its own guiding point is recommended for best coverage.

CHAPTER

FIVE

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 examining the possibility to use sampling-based algorithms as an optimization method of coverage.

The goal of this thesis was to design and implement an algorithm based on Rapidly-exploring Random Trees for motion planning of a group of cooperating micro aerial vehicles in the task of autonomous surveillance and compare it to an already developed method from [10]. The algorithm should provide a trajectory from a depot station to an optimized sensing location. An easy to optimize cost function describing the quality of the sensing location had to be developed. Feasibility of the trajectory must be ensured by an appropriate motion model and relative localization system.

The task was approached by using a method consisting of two parts which are finding a trajectory to a position, in which all UAVs are located over the AoIs and following optimization of AoI coverage by an optimization method. The results from table 4.1 in 4.1 show, that basic RRT method is too time demanding compared to PSO method, therefore the basic version of RRT was improved by adding a fast heuristic in the form of A* algorithm to provide RRT with a guiding path which is a geometric trajectory of a simplified configuration space. This improved version of RRT called RRT-Path is much faster and provides a shorter trajectory, furthermore the splitting ability of the UAV swarm can be encouraged by choosing an appropriate way of sampling around the guiding path as examined in 4.4.

Two methods were developed to optimize the coverage represented by a cost function - RRT-based and PSO-based. The RRT-based method proved to be superior to the PSO-based one in both run time and trajectory path in experiment 4.3. A cost function was found by discretizing the configuration space into a matrix and assigning a value to each element of the matrix depending of the importance of the area. This cost function is not only easy to optimize and fast to calculate, above that it also offers the possibility to assign different levels of importance to each area. To provide feasibility of the final trajectory a car-like motion model was chosen over a motion model of an real UAV thanks to it simplicity and generality which is important for fast run time of algorithm and versatile application with different kinds of UAVs respectively.

The presumed use of autonomous surveillance is monitoring of a parking lot near a supermarket, shopping center or a company campus. The advantage of an autonomous system over static cameras is, that an UAV swarm can operatively change the distribution of its individuals according to the position of most cars. Moreover static cameras usually fail to capture the identity of a criminal due to a large distance or a bad angle, while UAVs can change their position accordingly 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 for a more efficient decision making about fertilizing, watering, harvesting etc. As opposed to GPS localized UAVs, our swarm can even operate in hard to access areas, dense urban areas or inside buildings thanks to the relative localization system based on visual pattern recognition.

Some ideas for future improvements emerged during the work on this thesis. The control unit of the UAV could be extended by a visual object detection system to be able to actively trigger an alarm during suspicious actions for immediate notification of interested parties. Also dynamic updates to the map could be made by the swarm if a new obstacle was discovered when flying to the AoI. Ideally only the location of AoIs would be specified relative to the initial depot and the map would be generated on the fly.

BIBLIOGRAPHY

[1] I. C. A. Organization, "Unmanned aircraft systems (UAS)," 2011.

[2] R. Fernandes, "Monitoring system for power lines and right-of-way using remotely piloted drone," Apr. 4 1989. US Patent 4,818,990.

[3] R. Sugiura, N. Noguchi, and K. Ishii, "Remote-sensing technology for vegetation monitoring using an unmanned helicopter," *Biosystems Engineering*, vol. 90, no. 4, pp. 369 – 379, 2005.

[4] D. Gross, "Amazon's drone delivery: How would it work," *CNN. Cable News Network*, vol. 2, 2013.

[5] E. Mathews, T. Graf, and K. Kulathunga, "Biologically inspired swarm robotic network ensuring coverage and connectivity," in *IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, pp. 84–90, Oct 2012.

[6] K. Sreenath and V. Kumar, "Dynamics, control and planning for cooperative manipulation of payloads suspended by cables from multiple quadrotor robots," in *Robotics: Science and Systems (RSS)*, 2013.

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

[8] V. Kumar, *Integration of inertial navigation system and global positioning system using Kalman filtering*. PhD thesis, INDIAN INSTITUTE OF TECHNOLOGY, BOMBAY MUMBAI, 2004.

[9] J. Faigl, T. Krajnik, J. Chudoba, L. Preucil, and M. Saska, "Low-cost embedded system for relative localization in robotic swarms," in *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 993–998, May 2013.

[10] 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 *International Conference on Unmanned Aircraft Systems (ICUAS)*, vol. 1, (Danvers), pp. 584–595, IEEE Computer society, 2014.

[11] J. D. Cohen, S. Gottschalk, T. Hudson, A. Pattekar, M. C. Lin, and D. Manocha, "V-collide library," Feb. 2 1998.

[12] S. LaValle, *Planning Algorithms*. Cambridge University Press, 2006.

[13] S. A. Wilmarth, N. M. Amato, and P. F. Stiller, "MAPRM: A probabilistic roadmap planner with sampling on the medial axis of the free space," in *IEEE International Conference on Robotics and Automation*, pp. 1024–1031, 1999.

[14] L. J. Guibas, C. Holleman, and L. E. Kavraki, "A probabilistic roadmap planner for flexible objects with a workspace medial-axis-based sampling approach," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, vol. 1, pp. 254–259, IEEE, 1999.

[15] N. M. Amato, O. B. Bayazit, L. K. Dale, C. Jones, and D. Vallejo, "Obprm: an obstacle-based prm for 3d workspaces," in *WAFR '98: Proceedings of the third workshop on the algorithmic foundations of robotics on Robotics : the algorithmic perspective*, (Natick, MA, USA), pp. 155–168, A. K. Peters, Ltd., 1998.

[16] H. Kurniawati and D. Hsu, "Workspace importance sampling for probabilistic roadmap planning," in *International Conference on Intelligent Robots and Systems*, September 2004.

[17] M. Clifton, G. Paul, N. Kwok, D. Liu, and D.-L. Wang, "Evaluating performance of multiple rrts," in *IEEE/ASME International Conference on Mechtronic and Embedded Systems and Applications*, pp. 564 –569, oct. 2008.

[18] J. Kuffner and S. LaValle, "RRT-Connect: An efficient approach to single-query path planning," in *IEEE International Conference onRobotics and Automation*, vol. 2, pp. 995–1001 vol.2, 2000.

[19] M. Kalisiak and M. van de Panne, "Rrt-blossom: Rrt with a local flood-fill behavior.," in *ICRA*, pp. 1237–1242, 2006.

[20] J. Kennedy, "Particle swarm optimization," in *Encyclopedia of Machine Learning*, pp. 760–766, Springer, 2010.

[21] T. Lee, M. Leoky, and N. McClamroch, "Geometric tracking control of a quadrotor UAV on SE(3)," in *IEEE Conference on Decision and Control*, pp. 5420–5425, Dec 2010.