CZECH TECHNICAL UNIVERSITY IN PRAGUE

Faculty of Electrical Engineering

BACHELOR'S THESIS

Jaroslav Janoš

Inspection Planning for Firefighting with Unmanned Aerial Vehicle

Department of Cybernetics

Thesis supervisor: **Ing. Robert Pěnička**

Prague – 05/2020

Author statement for undergraduate thesis

BACHELOR‘S THESIS ASSIGNMENT

I. Personal and study details

Student's name:	**Janoš Jaroslav**	Personal ID number:	**474435**
Faculty / Institute:	**Faculty of Electrical Engineering**		
Department / Institute:	**Department of Cybernetics**		
Study program:	**Cybernetics and Robotics**		

II. Bachelor’s thesis details

Bachelor’s thesis title in English:

Inspection Planning for Firefighting with Unmanned Aerial Vehicle

Bachelor’s thesis title in Czech:

Plánování inspekce budovy s požárem pomocí autonomního bezpilotního prostředku

Guidelines:

1. Get familiar with the task of inspection planning in 3D environments and compare similar approaches for inspection of urban structures.
2. Implement a suitable approach for the visible surface determination problem in an environment with obstacles.
3. Propose a motion planning method for urban structure inspection, which maximizes the covered area considering the constraints given by an UAV.
4. Experimentally evaluate in simulator the proposed inspection planning method for fire detection.

Bibliography / sources:

[1] R. Penicka, J. Faigl, and M. Saska, “Physical Orienteering Problem for Unmanned Aerial Vehicle Data Collection Planning in Environments With Obstacles”, IEEE Robotics and Automation Letters, vol. 4, no. 3, pp. 3005-3012, 2019.
[2] B. Englot and F. Hover, “Inspection planning for sensor coverage of 3D marine structures”, in 2010 IEEE/RSJ International Conference on Intelligent Robots and Systems, 2010, pp. 4412-4417.
[3] P. Janousek and J. Faigl, “Speeding up coverage queries in 3D multi-goal path planning”, in 2013 IEEE International Conference on Robotics and Automation, 2013, pp. 5082-5087.
[4] A. Randa, T. Taha, L. Seneviratne, J. Dias, G. Cai, P. Z. Peng, and D. F. Lin, “Aircraft Inspection Using Unmanned Aerial Vehicles”, in International Micro Air Vehicle Competition and Conference 2016, 2016, pp. 43-49.

Name and workplace of bachelor’s thesis supervisor:

Ing. Robert Pěnička, Multi-robot Systems, FEE

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

Date of bachelor’s thesis assignment: **07.01.2020** Deadline for bachelor thesis submission: **22.05.2020**

Assignment valid until: **30.09.2021**

Ing. Robert Pěnička	doc. Ing. Tomáš Svoboda, Ph.D.	prof. Mgr. Petr Páta, Ph.D.
Supervisor’s signature	Head of department’s signature	Dean’s signature

III. Assignment receipt

.

Date of assignment receipt Student’s signature

Acknowledgements

I would like to express my deepest gratitude to my advisor, Ing. Robert Pěnička, for all the materials and source codes he gave me, as well as advices and guidance throughout the research and writing of the thesis.

I would also like to thank the Multi-robot Systems Group: the real-time simulation of the proposed algorithm could have not been done without their previous work and knowledge.

Last but not least, I would like to thank my family and my friends for their mental support during not only this research, but also during my whole studies.

Abstract

The aim of this work is to propose a method for planning of an inspection path for a fire detection in an urban area by an unmanned aerial vehicle. Assuming that the fire spreads on the objects located near the surfaces, e. g. floors, the main objective of the resulting inspection is to analyze as large area of these surfaces as possible. However, the path planning has to consider a limited length of the trajectory and a limited visibility of the scene. The stated challenge was divided into two consecutive steps: a generation of positions in the configuration space of the used vehicle, and the multi-goal path planning over the generated positions. The generation was tested using two different approaches. Firstly, by a tetrahedral decomposition of the space, and secondly by a random sampling based on the probabilistic roadmap methods. The visibility of the surfaces is then determined by back-face culling, or ray tracing. The path planning builds upon an existing solution for the physical orienteering problem. A new approach to the system of rewards in the orienteering problem was proposed, where the sum of collected rewards is replaced by a union over the visible surfaces corresponding to the visited positions.

Keywords: coverage path planning, building inspection, unmanned aerial vehicle, ray tracing, back-face culling, probabilistic roadmap planner, orienteering problem

Abstrakt

Cílem této práce je vytvořit vhodnou metodu pro plánování inspekční trasy pro detekci požáru v budovách či podobných objektech autonomním vzdušným prostředkem. Hlavním předpokladem této metody je, že oheň se šíří na objektech umístěných poblíž povrchů jako jsou podlahy – úkolem výsledné inspekce je tedy analyzovat co největší část těchto povrchů. Při plánování je však nutné brát v úvahu omezenost maximální délky trajektorie i limitovanou viditelnost objektů. Uvedený problém byl rozdělen na dvě navazující části: nalezení pozic v konfiguračním prostoru použitého robotu a plánování pro více cílových bodů s nalezenými pozicemi. Pro hledání pozic byly vyzkoušeny dva přístupy: rozklad prostoru na čtyřstěny a také náhodné vzorkování prostoru, založené na metodách probabilistic roadmap. Viditelné části povrchu byly poté určeny algoritmem back-face culling, nebo metodou ray tracing (sledování paprsků). Samotné plánování trasy pak staví na již existujícím řešení pro physical orienteering problem. Byl navržen nový přístup k systému odměn pro orienteering problem, kdy je suma nasbíraných odměn nahrazena sjednocením viditelných povrchů, odpovídajících navštíveným pozicím.

Klíčová slova: coverage path planning, inspekce budov, bezpilotní vzdušný prostředek, metoda sledování paprsků, back-face culling, probabilistic roadmap, orienteering problem

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The progress in the robotic research allows their deployment in settings where it was not possible just few years ago. One of these rapidly progressing robots are unmanned aerial vehicles (UAVs). Besides its usage in the industry, a robot may be used nowadays in exploratory and rescue operations at places where the life threatening risk for people is too high. Firefighting is one of these situations. In 2019 there were more than 20,000 fires in the Czech Republic alone, most of them in buildings or similar objects [7]. It is a prompt intervention which plays especially significant role for firefighting, together with an early localization of the fire and its scope.

This paper has been inspired by one of the challenges of MBZIRC 2020[1] international competition, where the task was to localize as well as extinguish a set of fires in a high-rise building. This task was executed by a team of three UAVs together with one unmanned ground vehicle (UGV). The first part, the localization of the fire sources, might be generally divided into two distinct processes:

1. sensoric detection of the fire itself, by e. g. thermal camera, and
2. motion planning of a robot, whose task is the localization of the fire sources.

The main objective of this paper is to solve the second subchallenge: to propose a method, which resolves the motion planning problem for one UAV.

This method must take into account an area of visible surfaces (e. g. floors, walls or ceilings) in an inspected building (possibly also a housing estate), since the fire spreads on objects near the surfaces, or even directly on these surfaces, as the present author assumes. Maximization of the visible area should therefore increase the probability of the fire detection. Another constraint is the requirement for minimal cost of the inspection path and other limitations, as described in the detailed specification (see Section 1.1).

This thesis is organized thus: firstly, the current state of the art of the coverage path planning is summarized, namely in 2D as well as in the 3D environments. The exact approaches are compared with heuristic algorithms and methods for the visible surface determination are then shortly introduced. In the method overview part, the challenge is then divided

[1]Mohamed Bin Zayed International Robotics Challenge is an international robotics competition, which aims on a demonstration of the current state of the art in robotics, `https://www.mbzirc.com/challenge/2020`

into two subchallenges, the generation of so called “viewpoints” and the path planning. The viewpoints generation consists of finding suitable positions of the robot in the configuration space and in obtaining corresponding visible surfaces in the inspected building. This is solved by two methods; first based on a division of the space into smaller reference cells, and second based on a random sampling. These points are then filtered and linked together by a path planning algorithm, as explained in the next chapter. The whole method is tested on 24 appropriate scenarios in terms of coverage or path utilization in two urban environments, and finally the validity and correctness are checked in a real-time simulator.

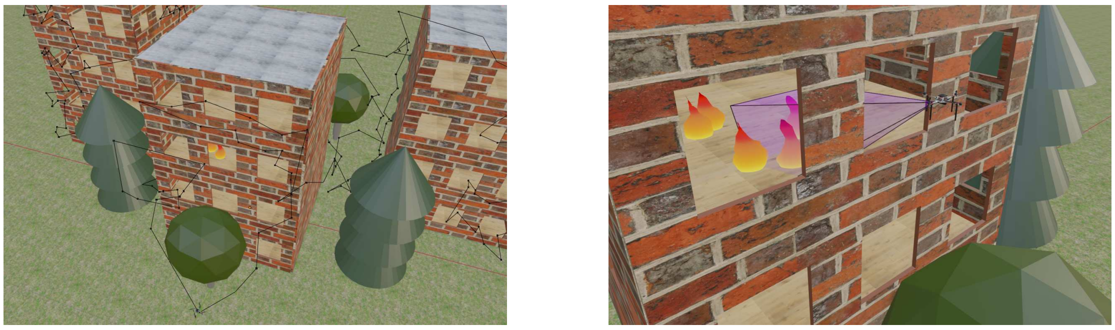


Figure 1.1: Illustration of the method’s function in a housing estate – the left figure depicts an example of a final inspection path, the right figure illustrates the sensor’s field of view during the fire detection

1.1 Specification

What is meant by the term “unmanned aerial vehicle”? Pajares [8] counts UAV among “unmanned aerial system” (UAS), which is an *“auto platform or remotely controlled platform through a remote station together with a communication system, including the corresponding protocol”*. That means, that the UAV is not only the aircraft itself, but also the full system with control and other auxiliary elements. This category consists of military fixed-wing UAVs as well as smaller (quad-, hexa-, octo-) copters with an ability to hold fixed position while flying. Even the smaller vehicles are able nowadays to carry a wide variety of sensors. These sensors might be used for both navigation purposes, e. g. GPS, Inertial Navigation Sensors or MEMS gyroscopes and accelerometers, or data collection purposes, e. g. cameras, thermal cameras, or smoke detectors (with regard to the firefighting). Apart from these, the UAVs can also perform many activities with actuators and other tools including extinguishers, fire blankets etc. These, altogether with the capacity of used batteries and weight of the aircraft itself, have a great impact on the maximal time of flight.

The vehicle endurance must be considered as a factor, thus the deciding parameter is a travel budget T_{max} (i. e. the maximum length of the inspection path), which must not be exceeded. Violation of this rule might even lead to a destruction of the UAV, since it might for example perform an emergency landing right into the flames. The complete visibility of the surfaces is then not assured, which means, that the purpose of the desired inspection should not be in a search for the first signs of an impending fire, but rather in a localization of an existing fire of a greater size for any subsequent rescue operations. Another restriction resulting from the target application is the limited space of operation. Since the insides of the building might be, due to the flames and smoke, hazardous even for a robot to operate, the path must be located entirely outside the analysed buildings and the UAV must also avoid other obstacles.

As many other free objects in 3D space, UAVs have theoretically 6 degrees of freedom (DOF) possibly with other redundant 3 DOF added by the used camera (if it is not fixed). However, presented methods are specialized and verified in 5 DOF (position in 3D, yaw and pitch), as the “roll” position might be in practice hard to achieve and would increase the difficulty of the problem. The extension on 6 DOF is possible anyway, when specified in the input for the method.

The main objective of the considered path planning is therefore to find a suitable path within the limited travel budget for an inspection of a building with fire so that the visible area of the analysed surfaces during the inspection is as large as possible. This path leads from a given starting point not only to a given ending point, but should also cross some of the generated viewpoints, thus the planning is multi-goal. Non real-time path planning of the inspection before its deployment is assumed, as well as a known model of the environment (i. e. blueprint of the building and its surroundings). Additionally, the object of interest must be precisely specified and must be sampled on an equally spaced and sufficiently dense point grid. Each point then represents a small subarea of the object of interest (e. g. all the floors of the inspected building). Another important input is the 3D model of an UAV. The output is the best trajectory represented by a list of points with their correct order, together with a list of visible points of interest.

Chapter 2

Related works

The path planning problem, sometimes referred to as motion planning, is defined as a method to *"find a collision-free motion between an initial and a final configuration within a specified environment"* [9]. The term "motion" can be then viewed as a sequence of valid configurations of the given robot. However, as Choset [10] points out, these methods do not address problems and applications such as lawn mowing, painting or harvesting, as well as inspections of various structures. Methods solving these tasks are known as *Coverage Path Planning algorithms* (CPP). They add an additional constraint to the previous definition, namely the path must pass over all points of an area or a volume (considering 3D environment).

The CPP problem might be also viewed as the *Covering Salesman Problem* (CSP), which is a variant of the generally known *Travelling Salesman Problem* (TSP). In CSP, an agent must visit a neighbourhood of each city, contrary to TSP, where each city must be visited directly [10, 11]. Some works also refer to a *prize collecting travelling salesman problem*, or *prize collecting rural postman problem* respectively, in which an agent needs to find a minimum-cost closed walk accross a graph with a prize located on each edge. However, this prize is collected only on the first traversal [6, 12]. All of these problems are proven to be NP-hard [10, 12].

Bearing in mind the problem of a visible surface determination, one can also refer to the *Art Gallery Problem* (AGP), where the main objective is to find a minimum number of guards, who need to be placed in an n-vertex polygon, so that all points in the interior are visible [13]. An extension with the relaxation of form of mobile guards is the *Watchman Route Problem* (WRP). This problem aims to find an optimal watchman route, for which a polygon is covered in a way, that the optimal route is the shortest possible route [13]. The AGP is NP-hard for polygonal maps, and so is the WRP [5].

As all of the aforementioned problems originate in coverage path planning methods, a basic overview of principles used in 3D as well in 2D environments will be presented in the next section.

2.1 Methods of coverage path planning

The CPP problems might be classified as either *online*, or *offline* (based upon *a priori* knowledge of the target environment). Although the offline models might be unrealistic in many scenarios, they form a foundation for online algorithms, which utilize real-time sensor measurements and adapt the resulting path accordingly [11]. These are also known as *sensor-based coverage algorithms* [10]. The main task of this work is based on offline methods, on that account offline methods are discussed primarily.

According to Galceron et al. [11], CPP problems might be additionally classified as *heuristic* or *complete* depending on whether they provably guarantee complete coverage of a free space. This property might be on the one hand very important in applications like mine detection, on the other hand they require more sensory and computational power [10], which must be taken into account when cosidering robots with limited travel budget. Therefore, the cost-per-quality of coverage might be better with randomized (i. e. heuristic) approaches [10].

2.1.1 Exact cellular decomposition

Exact cellular decomposition methods split the free space of the target environment into smaller obstacle-free units called *cells*. This way, the task shrinks to visiting each of these cells by using a graph-based search and covering them with a basic back-and-forth motion [10, 11, 9, 14, 15].

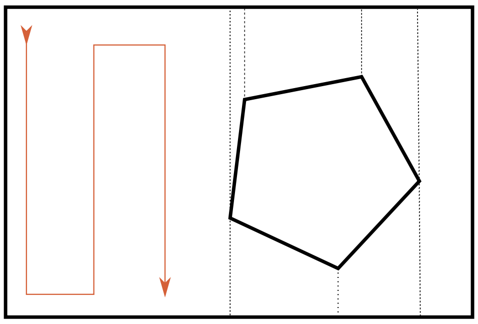


Figure 2.1: Trapezoidal decomposition of an rectangular environment with a demonstration of the back-and-forth motion

The typical and simple technique is the trapezoidal decomposition, depicted in Figure 2.1, where the environment is split into cells of the trapezoidal shape. Such configuration allows usage of the aforementioned back-and-forth motion to cover each cell [16]. The complete coverage is ensured by an exhaustive walk through the adjacency graph [10]. The application is for example in the agronomy, for the navigation of autonomous tractors or other mobile robots in fields with various shapes [17].

Moreover, some of these cells might be joined into bigger units, as shown in Figure 2.2. This problem addresses Boustrophedon decomposition, where a line segment is swept through the environment, while the boundaries of the cells are determined by the current number of intersections of the line segment with the particular obstacle [1]. This method can be also generalized using other shapes than the line segment. Such decomposition is proposed by the authors of [15] and is called *Morse-based cellular decomposition*. The mentioned shape of the sweep segment is decribed by a so called *Morse function*. By choosing a different Morse function, more complex cell patterns and decompositions might be obtained, e. g. Spiral, Spike and Squarel Patterns or Brushfire decomposition. As [11] points out, these might be useful for robots with various kinematic constraints.

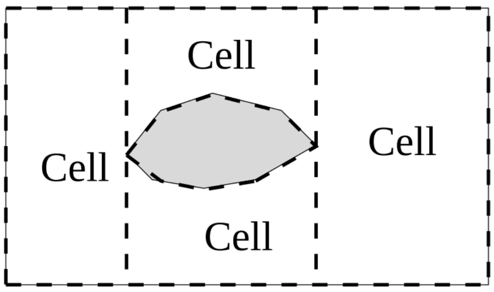
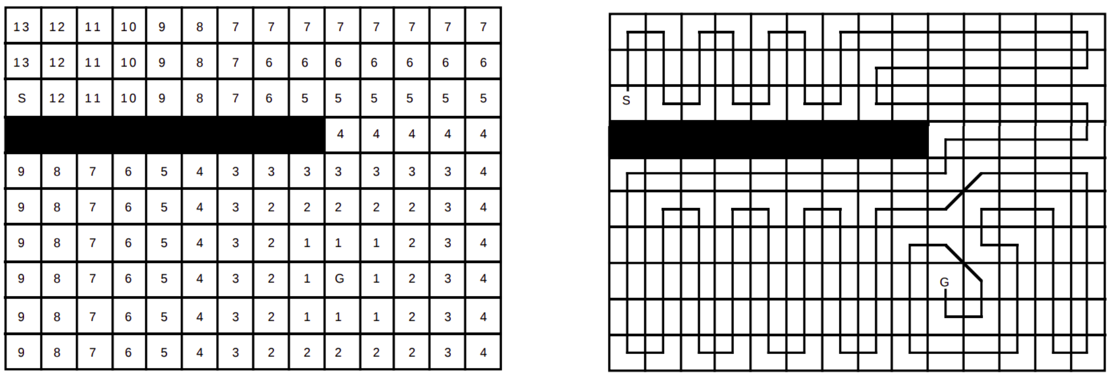


Figure 2.2: Boustrophedon decomposition [1]

2.1.2 Grid-based methods

Grid-based methods decompose the free space into small cells of the same size and shape. Furthermore, a value representing obstacle presence (might be also a real number representing the probability) is assigned to each cell. Despite the fact, that the complete coverage is not guaranteed and is dependent on the resolution of cells, these methods are the most commonly used in coverage problems [11]. The shape of the cells is usually rectangular, some authors propose a substitution by a triangular mesh, which offers higher resolution and thus better coverage [11]. The size of the cells often represents a footprint of the used robot or its end effector [10]. An example of a grid-based coverage is depicted in Figure 2.3.

The coverage path might be then determined by conventional state-space search methods. One of them is wavefront algorithm, which computes the distance of the cells from the goal state. During the execution robot performs a “pseudo-gradient ascent”, when it starts with the furthest cells from the goal and continues to the closer ones [2]. The authors of [18] further exploit this offline approach to an online *Iterate Wavefront* algorithm, which is, however, outperformed by the proposed *Delayed Greedy-Scan* algorithm.



13	12	11	10	9	8	7	7	7	7	7	7	7	7
13	12	11	10	9	8	7	6	6	6	6	6	6	6
S	12	11	10	9	8	7	6	5	5	5	5	5	5
■	■	■	■	■	■	■	■	■	4	4	4	4	4
9	8	7	6	5	4	3	3	3	3	3	3	3	4
9	8	7	6	5	4	3	2	2	2	2	2	3	4
9	8	7	6	5	4	3	2	1	1	1	2	3	4
9	8	7	6	5	4	3	2	1	G	1	2	3	4
9	8	7	6	5	4	3	2	1	1	1	2	3	4
9	8	7	6	5	4	3	2	2	2	2	2	3	4

Figure 2.3: Complete coverage by the grid-based distant wavefront algorithm [2]

Another online algorithm, called *Spanning Tree Covering*, constructs a systematic spiral path generated from a spanning tree of the partial grid map which is being incrementally built using onboard sensors [3]. Its function is shown in Figure 2.4.

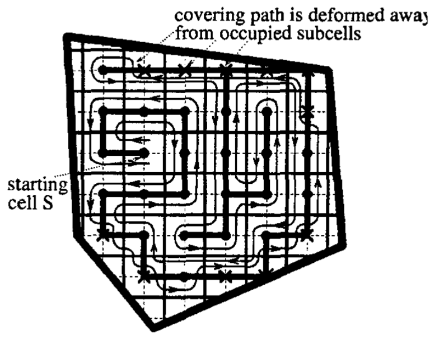


Figure 2.4: Execution of the *Spanning Tree Covering* algorithm [3]

Promising solution based on neural networks is also presented in [19, 20]. Each cell is then associated to a single neuron. As the algorithm is used for floor cleaning, each neuron represents the "cleanness" of the concerned cell, which is globally spread and "attracts" the robot.

2.1.3 3D coverage problem

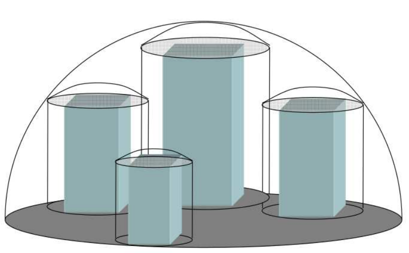
However, some problems cannot be simplified and solved by 2D methods, therefore more complex 3D methods must be considered. Many of them are though based on the same concept of cellular decomposition, problems with configuration spaces of higher dimensionality (five degrees of freedom or more) rely mostly on sampling-based algorithms.

Decomposition-based algorithms

Torres et al. [21] modify described 2D methods for usage by UAVs with a fixed camera, so that the camera footprint on the analysed surface is determined only by the height of the flight and camera parameters such as field of view (FOV) and aspect ratio. The main task is an energy optimization by decreasing the number of turns as well as the choice of a suitable start and end point of the considered back-and-forth motion. The authors propose calculation of an optimal line sweep direction and "coverage alternatives" to address this problem. Complex polygonal areas are decomposed to subpolygons, similar to the Boustrophedon decomposition technique.

Very often is the main area of application an underwater inspection or seabed mapping which is relative to an aerial application with the dimensionality of the configuration space. Conventional 2D methods are not feasible as they do not reflect complex structure of the ocean floor with caves or islands of different shapes. One of the solutions, presented in [22], is to cover these so called inlets and islands as soon as they are discovered and then continue with the conventional back-and-forth motion. It is moreover ensured, that every part of the surface is covered exactly once. This algorithm however does not deal with caves.

Another field of interest is inspection and coverage of urban structures such as buildings or even whole cities. In [4] the authors propose approximation of buildings by hemispherical and cylindrical coverage models, as shown in Figure 2.5a. Such approximation enables application of the same approach to buildings of different shapes and sizes. The trajectory of the model consists of circular paths in different heights according to robot's camera parameters, and the transition between them is determined with respect to the minimum of the time of flight. A complete trajectory for a cylindrical model can be seen in Figure 2.5b. However, such principle cannot be used for the solution of the main task of this work, as the models and trajectory do not reflect the visibility of the objects inside and focus only on the outter shell of the concerned urban structure.


(a) Coverage model for a general urban environment

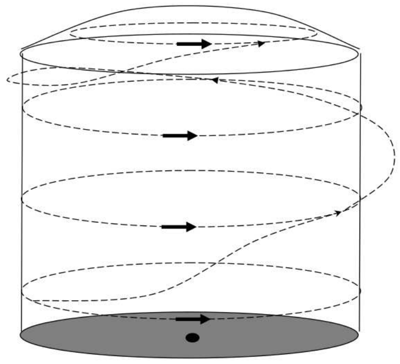

(b) Generated UAV trajectory for a cylindrical coverage model

Figure 2.5: Approximation of urban structures and the corresponding trajectory of UAV [4]

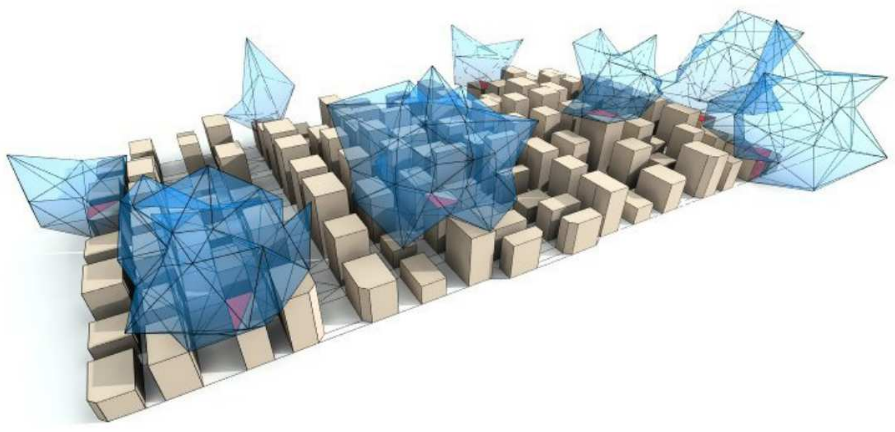


Figure 2.6: Example of a coverage space [5]

The urban environment is also the main subject of Janoušek and Faigl [5]. In their work, a supporting structure for visibility queries – set of covering spaces – has been proposed. For each object of interest a coverage space constisting of polyhedra is built and used to determine an inspection path by a technique of *self-organizing maps*. The planning algorithm is basically an unsupervised learning procedure for fully connected neural network. The coverage spaces, depicted in Figure 2.6, and their construction represent a promising and fast way to calculate visibility queries using fewer operations which is an inseparable part of the building scanning process. They are therefore discussed in a greater detail in Section 2.4.

Sampling-based algorithms

A different approach to the whole coverage problem choose Englot and Hover [12]. The base idea behind their algorithm for autonomous inspection of complex ship hull parts[1] is a division on the art gallery problem and the prize-collecting rural postman problem. Conventional modular approaches would be infeasible in such constrained environments, as the robot cannot move through the spaces between component structures [11]. The art gallery problem is solved by a quasi-random sampling of the configuration space so that the whole discretized model of the hull is swept by the bathymetry sensor volume. In contrast to [5], the *ray tracing algorithm* for visibility queries has been implemented, thus this method will be also thoroughly described in Section 2.4. These random points in the free space are then connected to a minimum-cost closed walk. The authors of [23] use almost the same principle, but for the inspection of aircrafts using UAVs. Moreover, they improve the presented algorithm by an heuristic algorithm for the path planning with *"increased possibility of obtaining an optimal path as the discretization resolution increases"* [23].

Similar method to the ray tracing is proposed in [24]. The authors utilize photon mapping algorithm for solution of the WRP. This technique offers more robust, physics-accurate approach as it takes into account media's effect on the propagation of photons as well as type of used sensor's electromagnetic radiation.

The division of the problem on AGP part and TSP part might, however, pose a problem for a path planning of a robot with differential constraints as some generated viewing points might not be reachable from other points in the free space [11, 25]. To address this problem, the authors of [25] propose an inspection algorithm that does not separate the problem and despite uses the random sampling techniques described above. This algorithm incrementally constructs a tree of feasible configurations and checks the newly created trajectory for optimality. The algorithm asymptotically converges to the optimal inspection trajectory with probability one.

2.2 Motion planning

The motion planning part of [12] is based on *Probabilistic RoadMap* (PRM) planner, since it solves the problem of collision avoidance in the environments with obstacles. The target environment of the method proposed in this work is in this manner very similar, therefore the methods of PRM are in the following section shortly introduced.

The PRM planner is a planning method proposed by Kavraki et al. [26] suitable especially for robots with many degrees of freedom, where decomposition methods often fail. The operation might be divided into two phases: *learning phase* and *query phase*. In the learning

[1]This approach is combined with the previously described back-and-forth motion in 3D environment for non-complex parts of the ship hull.

phase a free configuration space of the robot is sampled and generated configurations are connected with some neighbouring configurations using a *local planner*. This way a graph with nodes and edges representing the configurations and the computed connections is created. The following query phase uses the generated graph to answer queries about a free path between two configurations of the robot. Other expansion phases, whose principle is almost the same as the learning phase, might follow.

However, there are many possible implementations of the described methods, especially in the learning phase. They differ in used sampling methods or in base principle of the local planner. In [27], the authors present different approaches to the sampling. They distinguish two basic types: *uniform* and *advanced* sampling. The uniform sampling techniques include random sampling, grid-based sampling with increasing density of a grid, Halton points set or cell-based sampling, which is basically a random sampling in smaller cells that leads to a better spreading of the samples throughout the configuration space. The overall best results have Halton point sets [27]. Each technique has even so its advantages in specific environments. The advanced sampling should be used in more complex environments with narrow passages or large obstacles, as the techniques (gaussian, obstacle based, bridge test, etc.) produce samples with higher density among the occupied areas. As authors of [27] also suggest, the choice of the local planner for collision checking and determination of the free path might be also crucial. One approach to the local planning is *incremental planning* which takes small steps on the path and examines possible collisions for each step; the other one is *binary planning*, using the principle of binary search applied on the incremental approach. A combination of the previous ones is known as *line planning*, which checks the origin first and then proceeds with the mentioned binary method.

Karaman and Frazzoli [28] further exploit and compare several methods of choice of neighbour nodes in the learning phase. Generally, all of the given solutions, including the basic one, mark as the neighbour of c a node fulfilling certain criteria. Either, the distance from the node to c must not exceed given threshold d, or it must be between k nearest nodes of c. Furthermore, this node must not be in the same, already connected, component. However, according to [28], a fixed k or d might lead to probabilistic incompleteness and such algorithm is not asymptotically optimal. Therefore, the authors come with *PRM* algorithm*, where the threshold d is dependent on the actual number of nodes in the graph n as well as on the dimensionality of the configuration space l:

$$d = \gamma_{PRM} \left(\frac{\log n}{n} \right)^{\frac{1}{l}}, \tag{2.1}$$

$$\gamma_{PRM} > 2 \left(1 + \frac{1}{l} \right)^{\frac{1}{l}} \left(\frac{\mu(\chi_{free})}{\zeta_l} \right)^{\frac{1}{l}}, \tag{2.2}$$

where $\mu(\chi_{free})$ denotes Lebesque measure of the obstacle-free space and ζ_l is the volume of the unit ball in the l-dimensional Euclidean space [28]. More suitable is another version of the algorithm, the *k-nearest PRM**, where the number of nearest neighbours k is computed in similar but simpler manner:

$$k = k_{PRM} \log n, \tag{2.3}$$

$$k_{PRM} > e \left(1 + \frac{1}{l} \right). \tag{2.4}$$

Note that *"k_{PRM} is a constant that depends only on l, and ... $k_{PRM} = 2e$ is always a valid choice"* [28]. The authors further prove the asymptotical optimality and overall better performance, compared to the other approaches.

The other commonly used sampling-based algorithm is *Rapidly exploring Random Trees* (RRT), which is, in contrast to the PRM planner, only a single-query planner and thus is not suitable for the offline multi-goal planning [29, 28], which is the concern of this work. There are many other variants of both PRM and RRT, but their reconnaissance is not the subject of this work and may be found e. g. in [29].

2.3 Orienteering problem

In previous CPP algorithms, a complete coverage of an object of interest was the main concern of the proposed algorithms. However, some applications, such as fire detection, do not require this feature, as the size of the examined phenomenon exceeds size of the smallest part of the target environment. Another important feature, especially for mobile battery-powered robots as UAVs, might be the time of flight constraint, also known as *travel budget*. Such cases are then rather classified as *Orienteering Problems* (OP). In the OP, a set of vertices, each with assigned reward, is given, while the starting and goal nodes are fixed. The objective is to maximize the sum of collected rewards from visited vertices, so that the trajectory does not surpass the given travel budget [30]. In the case of this work, the rewards represent the area of the desired surfaces to be scanned (e. g. building floors or walls) from such vertices.

According to [30], several authors propose for the solution of OP exact algorithms such as branch-and-bound or branch-and-cut. However, these are able to solve problems with up to 500 vertices, for more complex tasks a heuristic approach must be used. Most of the heuristic approaches are based on the 2-OPT heuristic for TSP. The best result has been achieved by a five step heuristic introduced by Chao et al. [31]. At first, an ellipse over all points satisfying the given travel budget and, secondly, a greedy algorithm are used to find initial feasible paths. The most promising path P is then improved by performing a two point exchange, where a point from P is exchanged with some point from other path, so that the insertion is as cheap as possible and the total reward is increased or at worst only slightly decreased. The next improvement step, called one-point movement, consists of a placement of a point from the other path into P, under the same criteria as in the previous part. The iteration ends with the 2-OPT procedure and a final optimization, where the "least efficient" points are removed. As the authors of [30] state, only one approach involving a multi-objective variable neighbourhood search algorithm outperformed this heuristic, which has been validated by a newer survey [32].

Variable Neighbourhood Search (VNS) is a widely applicable approach to the design of heuristics for solution of various problems as combinatorial problems, a central problem of discrete location theory and others proposed by Hansen and Mladenović [33]. The basic VNS algorithm comprises of *shake* step, where a random point x' in the neighbourhood of an initial solution x is generated and *local search* step, where a local optimum x'' is found by state space search methods starting in x'. The solutions x and x'' are then compared and the better one is used for next iterations.

The orienteering problem for data collection in urban-like environments with obstacles is the main subject of work presented by Pěnička et al. [6]. The described problem is denoted as the *Physical Orienteering Problem* (POP). In their VNS-PRM* based solution of this problem, they combine into a single optimization problem the VNS heuristic methods for the solution of the orienteering problem with the methods of PRM* assuring the avoidance of obstacles. These methods are tightly coupled, as the VNS methods "links" together the paths between nodes found by the planner to maximize the sum of the collected rewards, and PRM* methods incrementally build a roadmap according to a feedback from the VNS methods. The new PRM* points are sampled in hyperellipsoids between neighbouring nodes, the exact number of added points is based on the possible reward gain as well as the actual density of points in the particular area. This approach decreases the computational complexity, since the roadmap is densely sampled only in promising areas. The resuls were also verified in 3D urban-like environments, as depicted in Figure 2.7.

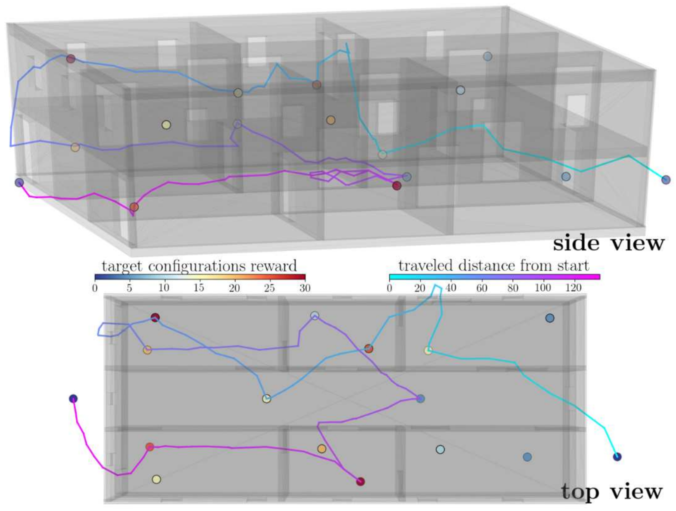


Figure 2.7: Example solution for the POP in a building [6]

2.4 Visible surface determination

Many of the aforementioned methods in Section 2.1.3 solve a common problem of computer graphics – determination of visible surfaces of a target object. These methods are primarily used for a correct and efficient projection of 3D objects on a 2D image, taking into account the occlusions of the scene, caused by the obstacles in the field of view. This principle is quite relative to the principle of all camera-like sensors, including the ones reffered in this thesis. In this case, these methods are therefore used for the determination of the visible parts of the surfaces (e. g. building floors) in the field of view of the used sensor, because the view may be also blocked by the obstacles or the building itself.

2.4.1 Ray tracing

The most commonly method used for the visibility problem is *ray tracing* [12, 23, 24]. The ray is actually an approximation of a photon's path. Rather than trace a photon from the source of the light with a high probability, that it does not even contribute to the image and ends up out of the scope (which is known as *forward ray tracing*), it is determined for each pixel in the image buffer which photon contributed to its final form, what its source was and how its trajectory was. This method is called *backward ray tracing*. Such approach would be, however, in most cases too expensive and purposeless, as all what is needed to know is the last object of the photon's reflection. When referring to the ray tracing, it is therefore searched only for the "first object hit by a ray" coming *de facto* from its destination [34].

In the real implementation, every object at the scene is tested for an intersection with the ray, while looking for the closest one [35]. As the intersection with the non-trivial objects might be computationally demanding, the object's surface should be in the case of the visible surface detection triangulated. The triangulation then simplifies the problem on a ray-triangle intersection. The authors of [36] compare algorithms to address this problem: Badouel's algorithm [37] builds on the idea of barycentric coordinates and expresses with them an intersection of the ray and the plane defined by the concerned triangle. Then three basic comparations lead to the result. Moeller-Trumbore's algorithm [38] exploits Badouel's algorithm by a transformation of the triangle into a basis with the triangle's edges as base vectors. This step reduces the storage time and space, because it is not neccessary to store the normal of the plane. In contrast to these methods, Segura and Feito [36] introduce an algorithm which does not directly calculate the intersection point – that means another improvement in the computational time.

2.4.2 Other approaches

Janoušek and Faigl [5] solve the visibility problem by sufficiently dense discretization of the free space to polyhedra using the Delaunay tetrahedralization implemented e. g. in `TetGen` [39]. For each polyhedron P and each object of interest O, a smallest possible set S of facets of P is found, so that every ray starting in P with a direction to any point in O intersects only the facets in S. This technique is similar to the *back-face culling* algorithm [35]. In the environment with obstacles, a graph of transitive dependency is built. The algorithm is further described in Section 3.1. The authors promise better performance than the aforementioned ray tracing technique [5]. However, as the subsequent implementation and testing showed, the tetrahedralization of complex structures is quite demanding and the process itself is not working on the target environments presumed by this work. The details of these issues are discussed in Sections 3.1.2 and 3.1.3.

Chapter 3

Generation of viewpoints

As it was already stated in the previous chapter, there are several ways to solve the assigned challenge of finding the inspection path. Were the main concern be the complete visibility, the author of this thesis would have chosen an approach similar to methods described in [12], [23] or [5]. However, none of the mentioned considers a limited length of the final path, i. e. the travel budget. To the best of author's knowledge, the constraint of the travel budget has never been described in any similar coverage path planning method. The closest relative problem, which takes into account the travel budget, is the orienteering problem, that motivates the formulation of this particular inspection planning.

The solution is divided into two subsequent steps. The first one is similar to the AGP, i. e. visibility determination problem, whose output is a set of points, hereafter dubbed as viewpoints, with a subreward associated with each of them. This subreward represents the number of visible subareas, i. e. number of parts of the divided object of interest located in the FOV of the used sensor and not occluded by obstacles. The subareas, hereafter dubbed as points of interest, are moreover weighted by a priority multiplier. The second step is the path planning over the set of found (or generated) viewpoints.

The viewpoints represent an approximation of the function, which assigns the number of visible points of interest to any position in the configuration space of the used UAV. The more points are sampled, the more accurate the approximation is. However, the computational demands increase at the same time. Without such approximation, the considered methods for path planning would not be feasible. Two approaches concerning the generation of viewpoints are proposed: the back-face-culling based approach from [5] and the random-sampling based technique, with the use of the ray tracing algorithms.

3.1 Space-decomposition approach

3.1.1 Algorithm overview

According to Section 1.1, a space of obstacles M and a set of points of interest N are given. Each obstacle $m \in M$ consists of a set of vertices V_M and a set of corresponding facets F_M. Considering additionally a working space $W \subseteq \mathbb{R}^3$, a free space $R \subseteq W$ is then defined as:

$$R = \{w \in W \mid w \notin M\}. \tag{3.1}$$

The space R, however, does not correspond with a space, in which the UAV is allowed to move. This space is hereafter marked as O. The motion is namely restricted not only by the analysed objects and other obstacles, but also by their interior. Here arises an important question: how to define an interior of an object? Possible solutions include any concave hulls, which are not uniquely defined, so that additional parameters for each object would be required, and they are quite difficult to compute. Because of these reasons a simple convex hull is used[1]:

$$O = \{w \in W \mid w \notin \text{convex hull of } M\} \subseteq R. \tag{3.2}$$

For the correct function of the following back-face culling algorithm, it is necessary to subdivide the spaces R, and O correspondingly, to smaller subspaces. For this purpose the 3D Delaunay triangulation[2] was chosen, similar to [5], which produces tetrahedral meshes T_R and T_O. It is then possible to exactly define the analysed challenge:
For each tetrahedron o in O find a set of points Q from N so that every point is fully visible from o.

To verify the full visibility the back-face culling method proposed in [5] was adapted: given $n \in N$ and $t \in T_R$, the algorithm returns the smallest possible set of facets S so, that every ray starting in t and heading to n crosses only facets in S. Any facet from S must not be then transitively dependent on any obstacle. To determine all facets of S, an out-pointing normal of each facet $f \in F_t$ is tested by a cross product with a line from n to opposite vertex $p \in V_t$ of f. If the result is greater than or equal to 0, the facet is inserted into S. This method is hereafter denoted as `criticalFacets(n, t)`.

The next challenge is how to connect the points in N with R, i. e. how to find the starting tetrahedra $t_0 \in R$, covering some point $n \in N$, in order to start the construction of a graph of transitive dependency. The proposed solution is to find a set of facets, whose points lie in the same level as n (and thus ensuring equality of one coordinate). Then it is performed a test determining, whether the point lies in a triangle (considering only the two dimensions left), using the barycentric coordinates [40]: point n lies in a triangle defined by vertices v_1, v_2 and v_3, when equation

$$n = v_1 + (v_2 - v_1) \cdot s + (v_3 - v_1) \cdot t \tag{3.3}$$

is valid, while $s \geq 0$, $t \geq 0$ and $1 - s - t \geq 0$.

The algorithm flow continues as depicted in Algorithm 1: a graph G of transitive dependency is built, which means, that every descendant of an inserted tetrahedron on the "path" heading to the point of interest must not border with any obstacle. Set T_{close} therefore represents tetrahedra transitively dependent on some obstacle and set T_{free} represents all tetrahedra, which have not been processed yet. The output of an auxiliary function `neighbors(t)` is a set of all tetrahedra, which share a facet with the given tetrahedron t. After the construction of $G(T_G, H)$ (Algorithm 2) it is decided, if the tetrahedron can be inserted into the graph of all viewpoints $Q(T_Q, K)$ (Algorithm 3). The inserted tetrahedron must be part of O and must meet the camera visibility range constraint ρ for every visible point of interest.

[1]The convex hull is an ideal solution for classic cuboid and sphere-like buildings, but it may too harshly restrict solutions for curvy or ring-shaped buildings.

[2]For the 3D Delaunay triangulation an external library will be used, therefore the principle and algorithm of triangulation will not be described in any greater detail. More information can be found e. g. in [39].

Algorithm 1 Overview of the space decomposition method

```
    Input $N$ – set of points of interest
    Input $M$ – space of obstacles
    Input $\rho$ – maximal visibility range
    Output $Q(T_Q, K)$ – graph of viewpoints and corresponding visible points of interest

 1: $R \leftarrow \{w \in W \mid w \notin M\}$
 2: $O \leftarrow \{w \in W \mid w \notin$ convex hull of $M\}$
 3: $T \leftarrow$ tetrahedralization of $R$
 4: for $\forall n \in N$ do
 5:     $T_{close} \leftarrow \emptyset$
 6:     $T_{visible} \leftarrow \emptyset$
 7:     $K \leftarrow \emptyset$
 8:     $t_0 \leftarrow$ starting tetrahedron for $n$, $t_0 \in T$
 9:     $T_{free} \leftarrow T \setminus \{t_0\}$
10:     while $\exists t \in T_{free} : t \in$ NEIGHBORS$(e) \wedge e \in T_{visible}$ do
11:         $G(T_G, H) \leftarrow$ GETDEPENDENCY$(n, t, T_{free}, T_{close})$
12:         $Q(T_Q, K) \leftarrow Q(T_Q, K) \cup$ PROCESSVISIBILITY$(n, T_{close}, G(T_G, H))$
13:     end while
14: end for
15: return $Q(T_Q, K)$
```

3.1.2 Implementation

The first version of the program was designed to test the basic functionality of the base algorithm and the used external libraries, therefore many of the proposed steps have been executed manually. However, even the basic tests failed, as will be described later in Section 3.1.3, thus no further improvements have been performed. The main program is written in `C++`, version `C++17` and was compiled using `g++` compiler. The auxiliary scripts for plotting were coded in `Python 3.6`.

The program works as follows: first it reads input file with three categories of objects: obstacle, object of interest and surrounding free space. Note, that the objects of interest and free-space objects must be uniquely named in the input file and these names must be passed in the call to the program along with the file path. The objects must be triangulated, for the correct function of the tetrahedralization process. The aforementioned input file must be in Wavefront OBJ format, a format for *"...defining a 3D geometry for the surface of one or more objects."* [41]. Files in this format are easy to parse, as every line starts with one or two characters defining the type of a record, and is followed by the data in specified format. However, only three types of records are supported in the designed implementation: `v` as a vertex, `f` as a facet and `o` as an object (the defining header for each 3D object).

After the parsing process, the free-space object is tetrahedralized using an external library `TetGen` [39], which corresponds to the implementation of [5]. Although other available libraries for tetrahedralization has been taken into account, e. g. `CGAL`, `TetGen` offers lightweight and simple command-line-like interface and, apart from tetrahedralized object, produces also other usable outputs. For example the lists of the neighbouring tetrahedra. Such outputs spare computational time of the following methods.

The implementation of the construction of covering spaces corresponds more or less to

Algorithm 2 Construction of dependency graph

```
Input $n$ – point of interest
Input $t$ – tetrahedron to process
Input $T_{free}$ – set of free tetrahedra
Input $T_{close}$ – set of blocking tetrahedra
Output $G(T_G, H)$ – graph of dependency

1: function getDependency($n$, $t$, $T_{free}$, $T_{close}$)
2:     $T_{tmp} \leftarrow \{e \mid e \in T_{free},\ e \in$ neighbors$(t)\}$
3:     $H \leftarrow \emptyset$
4:     $T_G \leftarrow \emptyset$
5:     while $T_{tmp} \cap T_{free} \neq \emptyset$ do
6:         $e_i \leftarrow$ pop from $T_{tmp}$ such that $e_i \in T_{free}$
7:         $T_{free} \leftarrow T_{free} \setminus \{e_i\}$
8:         $F_{nbr} \leftarrow$ criticalFacets$(n,\ e_i)$
9:         if $\exists f \in F_{nbr} : f$ abuts on an obstacle then
10:            $T_G \leftarrow T_G \cup \{e_i\}$
11:            $T_{close} \leftarrow T_{close} \cup \{e_i\}$
12:        else
13:            for $\forall e_{nbr} \in \{e \mid f \in e,\ f \in F_{nbr}\}$ do
14:                $T_{tmp} \leftarrow T_{tmp} \cup (T_{free} \cap \{e_{nbr}\})$
15:                $T_G \leftarrow T_G \cup \{e_{nbr}\}$
16:                $H \leftarrow H \cup \{(e_i,\ e_{nbr})\}$
17:            end for
18:        end if
19:    end while
20: return $G(T_G,\ H)$
21: end function
```

the Algorithms 2 and 3. The visibility of the points of interest from a tetrahedron is nevertheless represented by Boolean values, which could theoretically improve the memory usage – for the larger sets of the points of interest an identification number might be otherwise represented by 64bit data types, as opposed to `vector<bool>`, where only 1 bit is required. Despite this fact, please note, that this algorithm might be quite memory-consuming, especially for dense meshes and great sets of points of interest.

The output of this primitive testing version is not the set of outter tetrahedra with a list of visible points of interest, i. e. the graph Q, but only the so called coverage spaces – the coverage space of a point of interest n is a set of tetrahedra, from which the n is fully visible.

3.1.3 Basic testing of the proposed algorithm

In order to test the time complexity and correctness of the designed algorithms, two primitive testing environments were created. The environment 1 (see Figure 3.1a) consists of three cubes, the top of the middle one is marked as an object of interest. The environment 2 represents typical target environment: a cuboidal building with 4 floors marked as objects of interest. Both of the scenarios have been run on `Intel Core-i7 6500U` with 16 GB DDR3L 1600 MHz RAM, 8 GB swap partition on `Samsung 840 EVO` Solid state drive and `Ubuntu 18.04 LTS` operating system.

Algorithm 3 Addition of viewpoints to graph

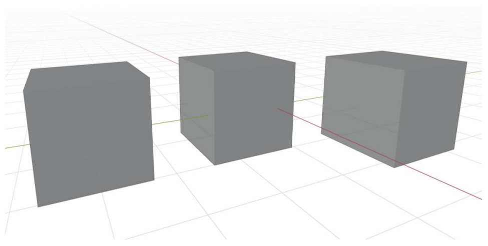
```
    Input $n$ – point of interest
    Input $T_{close}$ – set of points of interest
    Input $G(T_G, H)$ – graph of dependency
    Output $Q(T_Q, K)$ – graph of viewpoints

function PROCESSVISIBILITY($n$, $T_{close}$, $G(T_G, H)$)
    $T_Q \leftarrow \emptyset$
    $K \leftarrow \emptyset$
    $T_{tmp} \leftarrow T_{close}$
    while $\exists t_{act} \in T_{tmp}$ do
        $T_{close} \leftarrow T_{close} \cup \{t_{act}\}$
        $T_{dep} \leftarrow \{e \mid e \notin T_{close} \wedge (t_{act}, e) \in H\}$
        $T_{tmp} \leftarrow T_{tmp} \cup T_{dep}$
    end while
    for $\forall t_{vis} \in T_G \setminus T_{close}$ do
        $T_{visible} \leftarrow T_{visible} \cup \{t_{vis}\}$
        if $t_{vis} \subset O \wedge \|t_{vis}, n\| < \rho$ then
            $T_Q \leftarrow T_Q \cup \{t_{vis}, n\}$
            $K \leftarrow K \cup \{(t_{vis}, n)\}$
        end if
    end for
return $Q(T_Q, K)$
end function
```

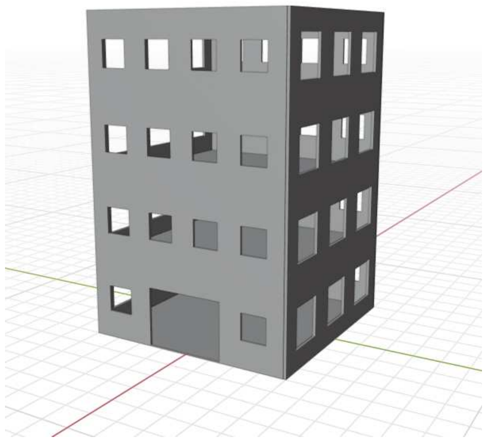
The first runs revealed serious issues in the environment 2, where the input objects could not be tetrahedralized by `TetGen` library. The cause lied in the model of the building, where single floors and walls overlapped themselves, and thus did not fullfil the requirements on *Piecewise Linear Complexes* [39]. Even the Boolean "Union" operation did not solve the issue and so every wall and floor object must have been shrunk. In this way, however, small gaps were created, which led to a generation of hundreds of thousands small tetrahedra. `TetGen` furthermore does not support any constraints of the minimum size of a single tetrahedron. The duration of the tetrahedralization process alone is dependent on the specified options, e. g. maximal size or quality of tetrahedra, of `TetGen`. The process lasted for this simple model of the building from 2 to 15 minutes according to the different configurations of `TetGen`, which is not suitable for any complex environments.

The next issue discovered in the environment 2 was the memory consumption. Since the number of previously generated tetrahedra was in order of millions, the size of the "visibility matrix" was in order of gigabytes. A possible solution is to store a list of identification numbers of visible points for each tetrahedron and moreover to limit the number of such tetrahedra.

The most weighty issue concerns the algorithm correctness. During the construction of the transitive dependency graph the collision with an obstacle is wrongly propagated. The method marks as "closed" every tetrahedron, even if only one of its critical facets is incident with the obstacle or another "closed" tetrahedron, although not all the incoming lines of sight are blocked. This case is illustrated in Figure 3.2b, where at first T_4 is (rightfully) marked as "closed", since not all the points in T_4 have full visibility on the area of interest. This is (again rightfully) propagated to T_3 and T_2, but erroneously to T_1, whose points have the full visibility on the point of interest. The error is propagated further and creates a "cap"

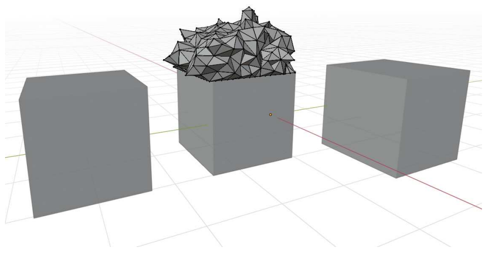


(a) Environment 1: basic model for a check of algorithm correctness

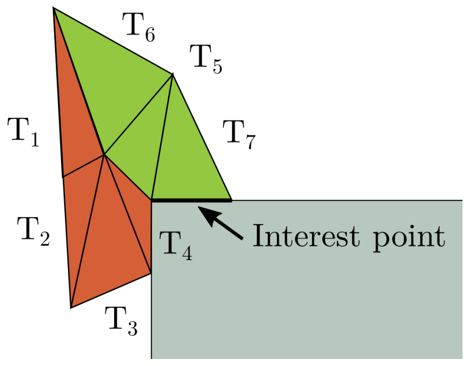


(b) Environment 2: simple model of a building

Figure 3.1: Environments for testing of the space-decomposition method



(a) Boundaries of the constructed coverage space for the environment 1



(b) Illustration scheme of the wrong propagation

Figure 3.2: Wrong transitive dependency propagation – reality and scheme

above the analysed area of interest which is the result of testing environment 1, depicted in Figure 3.2a.

A possible solution is to test each tetrahedron separately with e. g. ray tracing which increases nevertheless the time complexity and is incoherent with the statement that this method is faster than the ray tracing. The algorithm might work only in the urban open-space environments as presented in [5], or the authors probably wanted to propose only the structure (space divided by tetrahedralization on smaller subspaces with ability to hold visibility information), as the title of [5] suggests.

Because of the described time complexity issues of tetrahedralization, as well as the limits of the proposed algorithm, and possible additional harsh constraints on the supported input objects, the author of this thesis decided to leave this approach and fully focus on the sampling-based methods.

3.2 Sampling-based approach

3.2.1 Algorithm overview

In the same manner as in the space-decomposition approach, the working space is denoted as W, the free space as R, the operating space of the UAV as O and the set of all points of interest as N. Contrary to it, not only the visibility range, but also all other constraints of the given sensor, namely aspect ratio and the field of view (FOV), are considered from the beginning. Therefore, the configuration space C of the UAV was additionally defined:

$$C = O \times (-\pi,\, \pi\rangle^3, \tag{3.4}$$

where the first 3 dimensions represent the position of the UAV and the other 3 stand for the orientation of the sensor in the yaw-pitch-roll representation. As described in Section 1.1, the usage of roll position was not practically tested, even though it is theoretically considered.

The examined challenge might be then defined as follows:
Sample the configuration space C and obtain a set of viewpoints V, so that every v in V has a clear line of sight with at least some[3] *points from N.*

The basic overview of a solution is given in Algorithm 4.

Algorithm 4 Overview of the sampling based method

```
    Input $N$ – set of points of interest
    Input $M$ – space of obstacles
    Input $F$ – viewing frustum (defined by aspect ratio, range and FOV of the sensor)
    Input $P$ – number of target points
    Output $Q(T_Q,\, K)$ – graph of viewpoints and corresponding visible points of interest

 1: $R \leftarrow \{w \in W \mid w \notin M\}$
 2: $O \leftarrow \{w \in W \mid w \notin$ convex hull of $M\}$
 3: $C \leftarrow$ free configuration space of the UAV
 4: $I \leftarrow 0$
 5: while $I < P$ do
 6:     $t \leftarrow$ random configuration of the robot and sensor, $t \in C$
 7:     $V \leftarrow$ GETVISIBLEINTERESTS$(t,\, N,\, F)$
 8:     if $|V| \geq$THRESHOLD then
 9:         $I \leftarrow I + 1$
10:         $T_Q \leftarrow T_Q \cup \{t\}$
11:         for $\forall v \in V$ do
12:             $T_Q \leftarrow T_Q \cup \{v\}$
13:             $K \leftarrow K \cup \{(t,\, v)\}$
14:         end for
15:     end if
16: end while
17: return $Q(T_Q,\, K)$
```

First of all, the space C is randomly sampled. This procedure is divided into two consecutive steps: obtaining a valid random position of the UAV in O, followed by sampling a rotation of the sensor. For the position of the UAV, any strategy presented in Section 2.1.3

[3]The exact threshold is dependent on the total number of the points of interest.

might be used, as the whole procedure is based on the aforementioned PRM* algorithm. The "pure random" strategy with an uniform distribution was chosen for the purposes of this work, as the implementation is simple and the performance is sufficient for the target environments. Again, the sampled point must not collide with any of the given objects nor be placed in their interior. By the term "interior" are meant the convex hulls of the obstacles, as described in Section 3.1.

For the rotation of the camera, the same principle might be applied. However, each of the obtained viewpoints should have a clear line of sight with some points of interest and uniform sampling would produce too many invalid viewpoints. On that account the distribution of every base angle is *normal* with mean and dispersion based on the sensor's capabilities and placement. The only exception is the case of the yaw angle, where the mean is dependent on the center of the closest building. This might potentionally eliminate some viewpoints with a clear line of sight with some unique points of interest, nonetheless it increases the probability of validity of the viewpoint.

The next step is to obtain a set of all visible points from the sampled viewpoint, represented in Algorithm 4 with the method `getVisibleInterests`. For this purpose, two algorithms are employed. The major one is the Moeller-Trumbore ray-triangle intersection algorithm introduced in Section 2.4. It is based on aforementioned barycentric coordinates, discussed in Section 3.1. Assuming, that the ray R and triangle T intersect, it is possible to write an equation for the intersection point P:

$$P = s \cdot v_1 + t \cdot v_2 + u \cdot v_3, \tag{3.5}$$

$$s + t + u = 1, \tag{3.6}$$

where v_1, v_2 and v_3 are vertices of T and $s,\ t,\ u \in \mathbb{R}_0^+$.
As it is also possible to define P with parametric equation as $P = O_R + w \cdot d_R$, where O_R is the origin of R and d_R is the direction vector of R, the Equation (3.5) may be rewritten as:

$$O_R + w \cdot d_R = (1 - t - u) \cdot v_1 + t \cdot v_2 + u \cdot v_3, \tag{3.7}$$

$$O_R - v_1 = t \cdot (v_2 - v_1) + u \cdot (v_3 - v_1) - w \cdot d_R. \tag{3.8}$$

When the Equation (3.7) has at least one solution, R intersects T.
The validity of the Equation (3.7) is tested for every ray, coming from the viewpoint and heading to each of the points of interest, with every object.

This process might be, however, computationally demanding, especially in environments with many points of interest. To speed it up, all the points of interest are preprocessed and filtered by the second algorithm based on frustum culling presented in [23]. Every point of interest p is therefore transformed to the coordinate system of the deployed sensor (with the viewpoint as an origin):

$$p_{sen} = \mathbf{T}_0^{sen} p, \tag{3.9}$$

where $\mathbf{T}_0^{sen}$ is a 4-by-4 transformation matrix, transforming from the space O to the space of the sensor *sen*:

$$\mathbf{T}_{sen}^0 = \begin{bmatrix} \mathbf{R} & \mathbf{v} \\ \mathbf{0} & 1 \end{bmatrix}, \tag{3.10}$$

$$\mathbf{T}_0^{sen} = \begin{bmatrix} \mathbf{R}^T & -\mathbf{R}^T\mathbf{v} \\ \mathbf{0} & 1 \end{bmatrix}, \tag{3.11}$$

where **R** is a 3-by-3 rotation matrix determined by the orientation of the sensor and **v** is a vector of the viewpoint's coordinates.
The position of the transformed point p_{sen} is then checked for the visibility within the given range and FOV.

The viewpoints generation process always takes place before the path planning, nevertheless it is also possible to generate new viewpoints during the path planning. On the one hand, this approach may extend the set of visible points and thus expand the visible surface too, on the other hand it increases demand on the computational power and slows down the process of the path planning. The new viewpoints should be, therefore, generated with as high probability of an addition to the existing path as possible. This concerns especially more distant viewpoints, since the total length of the path is limited and such addition would probably trespass the given threshold. Because of this, the "pure random" sampling strategy is not suitable: new viewpoints should be generated as close to the existing path as possible. To satisfy this demand, two sampling strategies are proposed, "Ellipse" and "Sphere".

During the Ellipse sampling, new viewpoints are generated only in a hyperellipsoid around the trajectory between two random viewpoints, neighbouring to each other in the existing path. These points lie on the surface of the hyperellipsoid, length of one of its axis is thus the distance between the concerned viewpoints. Similar principle use Pěnička et al. [6] for the roadmap expansion.

The Sphere sampling should be more adaptable for the target environment, as the valid viewpoints appear only near to windows and other transparent objects. New viewpoints are thus generated only in a sphere, which center is a random viewpoint in the existing path. The circumference of the sphere was experimentally estimated as a quarter of the average length of the trajectory between two neighbouring viewpoints.

Exact number of generated viewpoints in the initial phase as well as in the extension phase cannot be generally determined and is highly dependent on every single target environment, especially on its size and complexity. Therefore it is, along with the performance of the proposed sampling strategies, tested later in Chapter 5.

3.2.2 Implementation

The program is written in `C++`, version `C++17` and was compiled using `clang++` compiler. The main input is a configuration file in `XML` format[4], in version 1.0. This file contains links to the input models in OBJ format mentioned in Section 3.1.2, information about the sensor capabilities and orientation, sampling range and configuration etc. Complete overview and structure might be found in an enclosed `README` file.

The parsing of all models happens in the same way as in the previous approach. The only difference are the requirements. Every input file may contain only 2 types of objects: obstacles and objects of interest. The objects of interest must be uniquely named and these names must be explicitly specified in the configuration file. Every object of interest must be equally divided on smaller subobjects without any requirement on shape of each of them, since only the points are taken into account. Every other object is automatically considered as an obstacle and must be therefore triangulated. Contrary to the tetrahedra approach, objects

[4]`https://www.w3.org/XML/Schema`

might overlap themselves. During this process a convex hull of the concerned model is created, using `CGAL` external library[5].

After the parsing procedure, the viewpoints are generated according to the stated algorithm. For the generation of pseudo-random numbers, 48-bit RANLUX generator is used. This generator is based on the lagged Fibonacci algorithm, which promises to be faster than other pseudo-random algorithms [42]. In the beginning, it is seeded by the actual time from the *high-resolution clock*. The initial sampling is done using uniform distribution and cartesian coordinates. For Ellipse and Sphere sampling strategies the spherical coordinates with uniform distribution of corresponding angles were implemented. The position of every generated point is then compared with the convex hull using the aforenamed `CGAL` library and potential collisions are checked using `RAPID` library[6]. If the viewpoint is not valid which means that it is located inside the convex hull or collides with it, the process of sampling is repeated. Number of such repetitions in not limited in the beginning of problem solution which implies that the exact number of desired viewpoints is sampled. However, the number of repetitions is limited in the extension phase during the path planning, where the generation and validation are repeated for at worst 50 times per one desired viewpoint to be sampled.

The set of the visible points of interest is expanded using the author's own implementation of the said algorithms. For the raytracing algorithm external libraries like `CGAL` were considered and tested but they failed in the closed urban environments. Other algorithms were quite difficult to implement and would pointlessly increase the size of the resulting program. The performance of the own implementation of Moeller-Trumbore algorithm is sufficient, compared to the complexity of the following path planning algorithms. The size of the viewpoint's set of the visible points of interest is compared to the thresholds stated in the input file. It is possible to specify two different thresholds, one for the initial sampling and the other one for the sampling throughout the path planning.

[5]`https://doc.cgal.org/latest/Convex_hull_3/index.html`

[6]RAPID (Rapid and Accurate Polygon Interference Detection) is an algorithm for collision detection using oriented bounded boxes [43]. This library was adapted from [6].

Chapter 4

Path planning

For the reasons presented in the previous sections, especially because of the limited travel budget, methods solving the Orienteering Problem (OP) were applied to the current challenge of finding an inspection path. Moreover, the proposed algorithm is built on the implementation of the VNS-PRM* approach to the Physical Orienteering Problem, presented in [6] which is correspondingly modified and extended to fulfill the specification of the considered inspection planning.

The most significant modification is made to the system of rewards collected by an UAV in OP. Usually, the total reward is the sum of collected rewards from individual viewpoints as they are fully additive. However, in this case the individual rewards represent extents of the visible surface and multiple observations of the same subarea are equal to a single observation. This is relative to the described prize-collecting rural postman problem (see Chapter 2), where a reward is collected only on the first passage through a reward point. Measures addressing this issue are proposed in the next section.

4.1 Algorithm overview

The path planning procedure is tied with the preceding viewpoints generation and as such, the same terminology is used. Therefore the working space is denoted as W, the free space as R, the operating space of the UAV as O, the set of all points of interest as N and the free configuration space of the UAV as C, with the same relations. Furthermore, another input is the set of generated viewpoints $V \subseteq C$ along with the information about the visibility of the points of interest. Hereafter, a set N_q of all visible points of interest from $q \in V$, is the output of the reward function:

$$r : r(q) = N_q. \tag{4.1}$$

Another input information are the start and goal configurations, q_S, $q_G \in C$. The path is then defined as an ordered sequence of feasible configurations, $\tau \subseteq C$, while $\tau_1 = q_S$ and $\tau_k = g_G$, where $k = |\tau|$. To correctly define the considered cost function, a function $A : C \to O$ must be declared. Function A only omits the orientation of the sensor and preserves the information about a position of the UAV in O. This also means, that the cost of transitions between sensor states as well as UAV's orientations in the space is not considered. The cost function is then declared as:

$$c : O^2 \to \mathbb{R}. \tag{4.2}$$

As it presents the length of the path between two configurations q and q', the author of this thesis defines it as:

$$c(q, q') = \|A(q),\, A(q')\|_2, \tag{4.3}$$

where $\|\cdot\|_2$ is the Euclidean distance. To ensure a collision-free path, function o denotes, whether the path between two configurations collides with any obstacle:

$$o : C^2 \to \{0, 1\}. \tag{4.4}$$

The value is either 0 for a collision-free path, or otherwise 1. The whole method of finding an inspection path might be then defined as an optimization task:

$$\begin{aligned} \max \quad & \left| \bigcup_{q \in \tau} r(q) \right|, \\ \text{s.t.} \quad & \sum_{i=1}^{k-1} c(\tau_i, \tau_{i+1}) \leq T_{max}, \\ & \sum_{i=1}^{k-1} o(\tau_i, \tau_{i+1}) = 0, \\ & \tau_1 = q_S,\ \tau_k = q_G. \end{aligned} \tag{4.5}$$

According to Equation (4.5), the aim of the method is to maximize the total area of visible surfaces, i. e. the sum of the points of interest. The sum of the costs for single trajectories between the neighbouring configurations in the sequence τ must not exceed the given travel budget. The sequence must start in the given starting configuration and must finish in the given goal configuration. Furthermore, the whole path must be collision-free, that is implied by the sum of collisions represented by the function o, which must be equal to zero.

An overview of an algorithm solving of the described problem is depicted in Algorithm 5, including the preceding methods of the viewpoints generation, `initialViewpointSampling` and `expansionViewpointSampling`.

The solution is based on a combination of VNS and PRM* methods, in the same manner as the solution in [6]. The VNS method focuses on the searching for a sequence of viewpoints with the highest reward and with a cost smaller than the given travel budget. Contrary to it, PRM* methods rather seek for the shortest collision-free paths between viewpoints, for a so called roadmap. They also ensure avoiding any obstacles. This roadmap is continuously expanded with a feedback from VNS about the quality of particular locations in the environment. The PRM* methods thus reduce the cost of the actual path sequence as well as the cost of an addition of other promising viewpoints.

The fundamental principles of PRM were presented in Section 2.1.3. The used methods are summarized in Algorithm 6. During the initialization (corresponding to the *learning* *phase*), the configuration space C is uniformly sampled using the “pure random” strategy and these points are then connected with their nearest neighbours[1], in case that this connection is collision-free. For this purpose, an *incremental* local planner is implemented. Its advantage is simplicity and small amount of “overheading” code, contrary to a binary local planner, which has been also tested. Due to the significantly longer computational time in every tested configuration, this idea was abandoned. The distances between single PRM points are determined using the classic Dijkstra’s algorithm.

The roadmap is further expanded by every iteration (see Algorithm 5). The only difference between the initial and the extension sampling is the strategy, since during the extension, new points are sampled in hyperellipsoids around the path between two neighbouring viewpoints. Unlike the viewpoints generation, the desired number of points to expand is split among every feasible trajectory between any pair of viewpoints. The number of added points is based on the current sampling density and the possible reward increase on the particular trajectory. This technique is fully adapted from [6]. The final path is furthermore smoothed (method `PRMSmoothing`), which means, that all redundant PRM points are removed from every path between any two viewpoints, while the particular path remains collision-free. The `PRMSmoothing` might be performed during every iteration as well. This might reduce the total cost of the actual best path, on the other hand it significantly extends the computational time of every iteration. Effectiveness of this technique is shown in Chapter 5.

The VNS part starts with `initialGreedySolution`, which adds viewpoints to the path on the basis of their “added length” versus “added reward” ratio. Its output is not only a path sequence τ, but also a general vector of viewpoints μ. The vector μ includes all viewpoints, which structure is fixed during the whole algorithm execution. Considering $\tau = \{q_S, q_2, q_3, \dots, q_{k-1}, q_G\}$, then:

$$\mu = \{q_2, q_3, \dots, q_{k-1}, q_k, \dots, q_{|V|}\}. \tag{4.6}$$

The position of particular viewpoints changes throughout the iterations, but first $k-2$ viewpoints of μ always form the path sequence τ. The main VNS method, `shake` and `localSearch`, whose theoretical principle was described in Section 2.3, are then executed in a loop, as depicted in Algorithm 5. The loop is broken when the number of iterations trespasses given threshold, the path is not improved for a certain number of iterations, or the alloted time runs out. The Algorithm 7 further clarifies the application of VNS on the particular challenge, or the physical orienteering problem respectively [6].

The *shake* procedure consists of two alternating behaviours, *Path move* and *Path exchange*. In the first case, a random coherent sequence of viewpoints from μ is moved forward, and *de facto* inserted into τ. In the second case, two random sequences from μ are exchanged between themselves. In both cases, the number of viewpoints in τ is eventually adjusted so, that it fits the given travel budget.

The *local search* procedure works in a very similar manner, too. Its alternating submethods, *One point move* and *One point exchange*, take one random viewpoint from μ and move it to τ, or exchange it with another random viewpoint in τ respectively. These steps are executed $|V|^2$ times, where V is the set of all viewpoints. The changes, which do not im-

[1]The exact number of neighbours is given by the used PRM* principle and is further explained in Section 2.1.3.

Algorithm 5 Inspection planning algorithm

```
Input $N$ – set of points of interest
Input $M$ – space of obstacles
Input $P_{init,\,exp}$ – desired number of viewpoints in initial and expansion phase
Input $H_{init,\,exp}$ – desired number of PRM points in initial and expansion phase
Input $F$ – viewing frustum (camera configuration)
Input $T_{max}$ – travel budget
Output $\tau$ – inspection path

1: $V \leftarrow$ initialViewpointSampling($N$, $M$, $F$, $P_{init}$)
2: $G \leftarrow$ initialPRMSampling($M$, $H_{init}$)
3: $Q \leftarrow$ find the best paths between viewpoints in $V$ using $G$
4: $\tau\ \mu \leftarrow$ initialGreedySolution($Q$, $T_{max}$)
5: while stopping condition not met do
6:     $I \leftarrow 1$
7:     while $I \leq 2$ do
8:         $\tau'$, $\mu' \leftarrow$ shake($\tau$, $I$, $\mu$)
9:         $\tau''$, $\mu'' \leftarrow$ localSearch($\tau'$, $I$, $\mu'$, $|V|$)
10:        if (length($\tau''$) $\leq T_{max}$ and reward($\tau''$) $\geq$ reward($\tau$)) or
11:        (reward($\tau''$) = reward($\tau$) and length($\tau''$) < length($\tau$)) then
12:            $\tau \leftarrow \tau''$
13:            $\mu \leftarrow \mu''$
14:            $I \leftarrow 1$
15:        else
16:            $I \leftarrow I + 1$
17:        end if
18:    end while
19:    $V \leftarrow V \cup$ expansionViewpointSampling($N$, $M$, $F$, $P_{exp}$)
20:    $G \leftarrow$ expansionPRMSampling($M$, $H_{exp}$)
21:    $Q \leftarrow$ update best paths between viewpoints in $V$ using $G$
22: end while
23: $\tau \leftarrow$ PRMSmoothing($\tau$)
24: return $\tau$
```

Algorithm 6 Methods of PRM*

```
Input $M$ – space of obstacles
Input $H$ – number of PRM points to sample
Input/Output $G(D, E)$ – (existing) roadmap

1: $R \leftarrow \{w \in W \mid w \notin M\}$
2: $O \leftarrow \{w \in W \mid w \notin$ convex hull of $M\}$
3: $C \leftarrow$ free configuration space of the UAV
4: function PRMSampling($M$, $H$)
5:     $I \leftarrow 0$
6:     $D_{new} \leftarrow \emptyset$
7:     while $I < H$ do
8:         $p \leftarrow$ random sample (uniform/in hyperellipsoid), $p \in C$
9:         $D \leftarrow D \cup \{p\}$
10:        $D_{new} \leftarrow D_{new} \cup \{p\}$
11:    end while
12:    $G \leftarrow$ createRoadmap($G$, $D_{new}$)
13: return $G$
14: end function
15:
16: function createRoadmap($G$, $D_{new}$)
17:    for $\forall d \in D_{new}$ do
18:        $J \leftarrow$ kNearest($d$)
19:        for $\forall j \in J$ do
20:            if collisionFree($d$, $j$) then
21:                $E \leftarrow E \cup \{(d, j)\}$
22:            end if
23:        end for
24:    end for
25: return $G$
26: end function
```

prove the current path, are discarded. The travel budget is not considered, due to the PRM extension strategy, as described in [6].

Algorithm 7 Methods of VNS

```
Input $I$ – variant of the method, $I \in \{1, 2\}$
Input $|V|$ – number of generated viewpoints
Input/Output $\mu$ – partly ordered vector of feasible viewpoints
Input/Output $\tau$ – currently best path

1: function SHAKE($\tau$, $I$, $\mu$)
2:     if $I = 1$ then                                                  ▷ Path move
3:         $X \leftarrow$ random sequence of viewpoints from $\mu$
4:         $P \leftarrow$ random position in $\tau$, outside of $X$
5:         $\tau$, $\mu \leftarrow$ move $X$ to $P$
6:     else                                                             ▷ Path exchange
7:         $X \leftarrow$ random sequence of viewpoints from $\mu$
8:         $Y \leftarrow$ random sequence of viewpoints from $\mu$, $X \cap Y = \emptyset$
9:         $\tau$, $\mu \leftarrow$ exchange $X$ with $Y$
10:    end if
11:    $\tau \leftarrow$ optimize $\tau$ to fit the budget $T_{max}$
12: return $\tau$, $\mu$
13: end function
14:
15: function LOCALSEARCH($\tau$, $I$, $\mu$, $|V|$)
16:     for $|V|^2$ do
17:         if $I = 1$ then                                             ▷ One point move
18:             $X \leftarrow$ random viewpoint from $\mu$
19:             $P \leftarrow$ random position in $\tau$, must not be the position of $X$
20:             $\tau'$, $\mu' \leftarrow$ move $X$ to $P$
21:         else                                                        ▷ One point exchange
22:             $X \leftarrow$ random viewpoint from $\mu$
23:             $Y \leftarrow$ random viewpoint from $\tau$, $X \neq Y$
24:             $\tau'$, $\mu' \leftarrow$ exchange $X$ with $Y$
25:         end if
26:         if REWARD($\tau'$) $\geq$ REWARD($\tau$) or
27:         (REWARD($\tau'$) $=$ REWARD($\tau$) and LENGTH($\tau'$) $<$ LENGTH($\tau$)) then
28:             $\tau \leftarrow \tau'$
29:             $\mu \leftarrow \mu'$
30:         end if
31:     end for
32: return $\tau$, $\mu$
33: end function
```

A solution of the reward challenge is indicated even in the problem definition, Equation (4.5): the summation of rewards must be replaced by a union over the sets of points of interest, hereafter referred as "reward sets", corresponding to the viewpoints contained in the path τ. To prioritize some objects of interest, the points of interest have moreover certain "weight", specified by the user input. The areas visible during the transitions between single viewpoints are not considered. The function `reward` (in Algorithm 7 and Algorithm 5) returns then the "weighted cardinality" of the global reward set (corresponding to the whole path τ). The complexity of a viewpoint addition to the path is $2(n_1 + n_2)$, where n_1 is the number of the points of interest in the global reward set and n_2 is the number of points of interest in the reward set of the viewpoints[2]. Contrary to it, the complexity of a removal of any viewpoint and its reward set is due to the *inclusion-exclusion principle* rather $2n_1^2$, since a union over the remaining reward sets must be performed.

For this reason the present author wanted to propose a heuristic for a rough assumption of viewpoint's “quality”. The method is inspired by the *Conditional Random Fields* used in image segmentation [44]. The main assumption is, that near pixels in the image with similar color should belong to the same object in the image. In the same manner, it is assumed, that near viewpoints with similar orientation of the sensor should have a visibility of the same points of interest. On the one hand, no reward sets are required in this process, on the other hand, it does not reflect sensor's parameters or the boundaries of the objects of interest. Moreover, the implementation would be difficult and the computational demands presumably even higher than in the case of the global reward set.

To reduce the complexity of a removal, a counter was added for every point of interest in the global reward set. The complexity of a viewpoint addition is still $2(n_1 + n_2)$, whereas the complexity of a viewpoint removal is $2(n_1 + n_2)$, assuming that the complexity of a creation, a destruction, an increase and a decrease of the counter is constant. On the other hand, a memory complexity has increased from n to $2n$.

4.2 Implementation

The implementation of the path planning algorithm is built upon the previous implementation of viewpoints generation and as such, it shares its structures and object models. It also expands the configuration file with input parameters like travel budget, PRM points counts and other PRM settings, stopping conditions, starting and goal points, or output settings. Once again, the majority of the added code is based upon the implementation of [6].

After the viewpoints generation, the program continues with the initial PRM sampling. This sampling uses ranges and collision models from the sampling of viewpoints. Every newly sampled viewpoint is connected with k nearest neighbours, according to the PRM* strategy. These neighbours are found using `FLANN` external library[3] and collisions on the corresponding paths are checked using `CGAL` and `RAPID` as in Section 3.2.2. Single queries are solved using Dijkstra's algorithm with an adjusted heap and switchable PRM smoothing.

The program keeps distances of all paths among viewpoints, these are updated after every PRM and viewpoints expansion. The early tests have shown, that this update is a bottleneck of the whole program, therefore it has been parallelized. The number of assigned threads can be specified during the compilation by definition of the `NUM_THREADS` define.

The VNS methods were implemented roughly in the same manner as depicted in Algorithm 7. The random sequences and the viewpoints are generated using the same pseudo-random engine as in Section 3.2.2. The path operations are optimized by memorizing of all

[2]Considering only the addition of the corresponding reward set, which is ordered as well as the global reward set.

[3]FLANN is a library for nearest neighbor search using approximate solution, which has proven to be good enough for majority of cases [45].

distances between the visited viewpoints from the beginning to the end, as well as in reverse, from the end to the beginning of the path.

Finally, the system of rewards was implemented in two ways:

1. Performing union over the reward sets, using an ordered `set` from the standard library, and the `set_union` function from `algorithm` library (i.e. the straightforward approach).
2. Keeping a counter for every point of interest, using `map` structure from the standard library.

Even though the complexity of the second approach is not $2(n_1 + n_2)$, as advertised in the theoretical part, but $n_2 \log n_1$, it was still in average about 10 times faster than the first straightforward approach.

The output files of the whole algorithm comprise:

1. a set of all sampled viewpoints,
2. a set of all sampled PRM points,
3. a set of all visible points of interest,
4. the final inspection path, and
5. a plain file containing an information about the total reward and the length of the path, as well as the total number of elapsed iterations.

All sets are in the OBJ format, the final path is in OBJ format (only points positions) and in SSV format[4] (points positions along with the corresponding yaw angles). The SSV format is used by the UAV for trajectory following either in simulation or with a real drone.

[4]SSV stands for space-separated values

Chapter 5

Results

The proposed solution and its features were tested on two urban environments. The results of the evaluation should answer these questions:

- *Is the extension of the viewpoints during the path planning efficient?*
- *With an adequate travel budget, is the area of the visible surfaces sufficient?*
- *Does the proposed method of prioritization of certain objects of interest work?*
- *Which one of the proposed sampling strategies for viewpoints extension does perform better?*
- *How much does the PRM smoothing method improve the algorithm's utilization of the given travel budget?*

The first of the testing environments is a classical cuboidal building (Figure 5.1), which is already used in Section 3.1.3. Its 4 floors with dimensions 8×8 m are equally divided on 25,092 points of interest with the weight 1 and are visible from every side. Each point of interest represents then an area of 0.01 m^2.

The second environment is represented with a small housing estate, with three buildings and other obstacles in form of trees. It is depicted in Figure 5.2, along with the starting point of an UAV, since it is, contrary to the first environment, a decisive factor for the path planning algorithm. Each of the buildings corresponds to the building in the environment 1, therefore the subdivision of the floor area is the same. The side buildings are marked as priority, with weight 3. The motion of an UAV is further restricted to the area between the buildings, the right wall of the right back building and the left wall of the left back building are therefore inaccessible.

To answer the stated hypothesis, 24 testing configurations were formed. They are shortly introduced in the following sections, along with the comparison of the results and their evaluation. The full overview of configurations and their raw results are available in the Appendix, Full results of testing.

Each of the configurations was executed 5 times, the algorithm was stopped after one of the following stopping conditions had been met: 1,000 iterations, 80 iterations without any

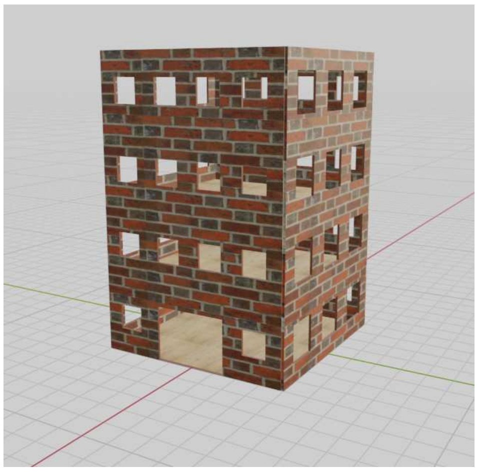
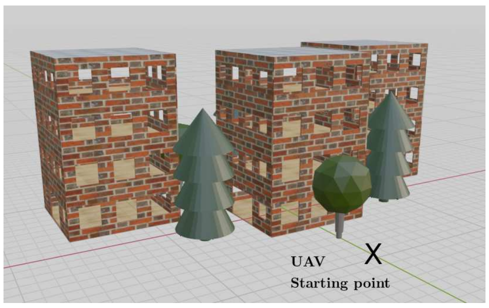
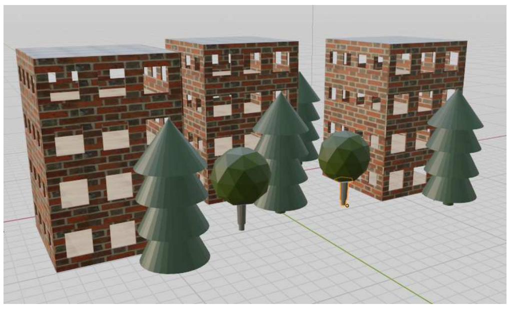


Figure 5.1: Environment 1 – single 4-floor building



(a) Front view with the starting point of UAV



(b) Back view

Figure 5.2: Environment 2 – a housing estate

improvement (except for the path length reduction due to the PRM extension), or exceeded time of execution (3 hours).

The UAV is substituted by a simple cylindrical model with radius 0.5 m and height 0.7 m. The camera's parameters correspond to the `FLIR Lepton 3.5` thermal camera with a resolution 160×120 and the FOV 57°. The minimum and maximum range were determined by the resolution of the camera and the size of a sample in the previously presented models as 0.3 m and 5.5 m respectively. The main criterion is a reliable fire detection, with a small probability of false alarms. Therefore the fire must be detected by all pixels in a 3×3 frame.

To ensure equal conditions, all the configurations were run on one computer with an `Intel Core i7-6500U` processor, 16 GB DDR3L 1600 MHz RAM, 8 GB swap partition on `Samsung 840 EVO` Solid state drive and `Ubuntu 18.04 LTS` operating system.

5.1 Viewpoints extension

The proposed method of the viewpoints generation was tested in both environments, with travel budgets 50 m, 100 m and 200 m for the environment 1 and 100 m, 200 m and 400 m for the environment 2. The choice is random, partly based on the primal basic tests. The evaluated extension strategies, a ratio of viewpoints generated in the initial phase referred as "base" and viewpoints generated in the extension phase referred as "extension", are 200/0, 100/5 and 50/10 for the environment 1 and 300/0 and 100/10 for the environment 2. The number prior to the slash marks base viewpoints, the number after the slash marks extension viewpoints. The choice is again random, with regards to the computational complexity for higher numbers of extension viewpoints. The extension sampling strategy for all the configurations is Sphere, PRM sampling rate is 5000/50 for the environment 1 and 5000/150 for the environment 2, using similar marking as for the viewpoints generation.

Table 5.1: Comparison of the results for evaluated viewpoints extension strategies for the environment 1

Budget [m]	Base/ext.	Average reward [-]	Average path length [m]	Average count of iterations [-]
50	200/0	9898.0 ± 236.6	47.3 ± 0.1	124.6 ± 15.1
50	100/5	13634.8 ± 701.4	47.8 ± 0.2	567.0 ± 26.2
50	50/10	13460.2 ± 421.1	48.1 ± 0.5	358.0 ± 12.2
100	200/0	16463.6 ± 327.5	96.6 ± 1.3	119.6 ± 17.7
100	100/5	21553.2 ± 402.4	98.9 ± 1.0	478.4 ± 24.4
100	50/10	20465.4 ± 608.5	98.9 ± 1.1	351.2 ± 23.9
200	200/0	21206.4 ± 136.9	194.9 ± 2.2	275.6 ± 35.0
200	100/5	24158.4 ± 135.4	192.6 ± 2.1	540.4 ± 30.1
200	50/10	22853.6 ± 241.2	195.5 ± 2.7	407.4 ± 22.4

The comparison of the average rewards from 5 tests, along with the average path length and average count of elapsed iterations, is shown in Table 5.1 and for a better illustration also in Figure 5.3. The number after ± sign denotes the standard deviation. As the graph shows, the extension sampling strategies clearly outperform 200/0 strategy in every travel

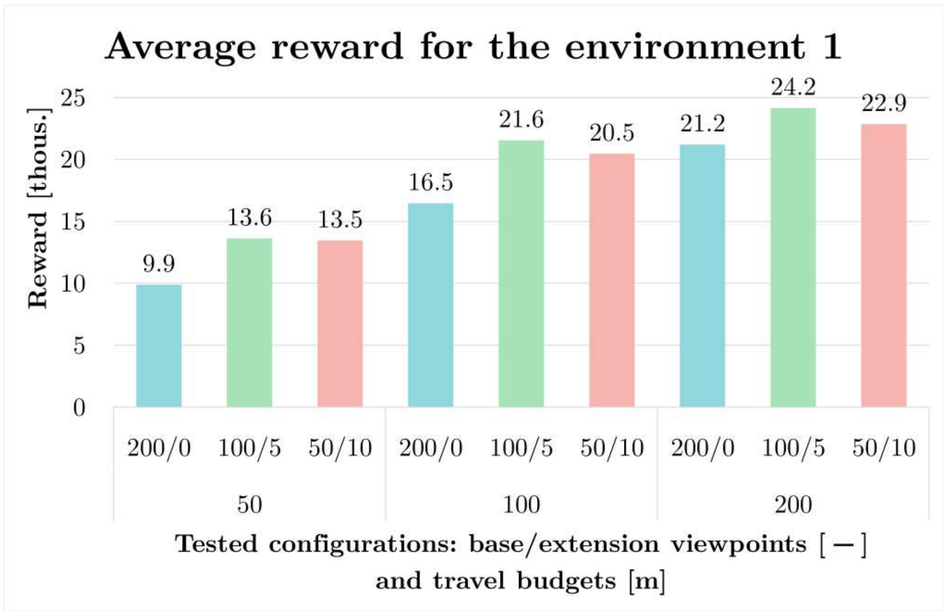


Figure 5.3: Average reward for the environment 1, based on the budget and the viewpoints extension strategy

budget. However, a higher number of extension viewpoints does not ensure better results, as it is computationally demanding and lowers the total number of elapsed iterations during the 3-hours execution span (the maximum time of execution was the stopping condition in all cases, except for configurations with the strategy 200/0).

Table 5.2: Comparison of the results for evaluated viewpoints extension strategies for the environment 2

Budget [m]	Base/ext.	Average reward [-]	Average path length [m]	Average count of iterations [-]
100	300/0	46270.8 ± 1030.1	96.8 ± 0.8	164.2 ± 25.6
100	100/10	58642.6 ± 1376.8	96.3 ± 0.3	219.6 ± 5.2
200	300/0	91774.8 ± 1260.0	198.5 ± 1.1	209.0 ± 31.8
200	100/10	100512.6 ± 1087.6	191.8 ± 0.1	151.0 ± 1.1
400	300/0	115645.6 ± 1324.7	387.1 ± 3.0	256.8 ± 27.9
400	100/10	114513.6 ± 2660.7	388.9 ± 3.8	142.8 ± 1.6

The results for the environment 2 are in the same manner as for the environment 1 depicted in Table 5.2 and in Figure 5.4. They confirm the conclusion from the environment 1, that the extension strategies do perform better than the initial-sampling-only strategies. The difference is not so significant, which is probably caused by the complexity of the environment itself (as it contains more objects than the environment 1) and by a low number of elapsed iterations with a minimal deviation.

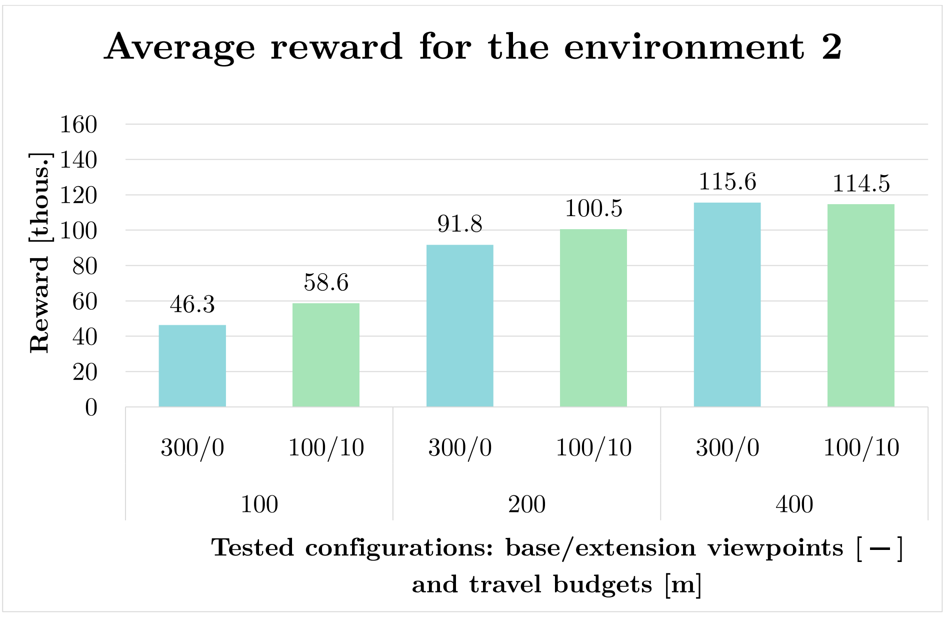


Figure 5.4: Average reward for the environment 2, based on the budget and the viewpoints extension strategy

5.2 Relative visibility of the area of interest

The same testing configurations and even the same data, depicted in Tables 5.1 and 5.2, were also used to evaluate the relative coverage, i. e. relative visibility, of the areas of interest in both environments.

The results for the environment 1 in Figure 5.5 show, that the coverage is almost complete for the travel budget 200 m and the extension sampling strategy 100/5. The best achieved result for the environment 1 is even 97.84 % of the total count of the points of interest, according to Full results of testing in Appendix.

However, the results for the environment 2 are not so promising. Surprising is the small gap between configurations with the travel budget 200 m a 400 m, which is probably caused by the low number of elapsed iterations, or a small number of base viewpoints for such a spacious environment.

5.3 Prioritization of objects of interest

As it was already mentioned above, the objects in the environment 2 have different weights (rewards) on their points of interest. To check the correct functionality, the paths of the best runs for the configurations from Section 5.1 were further analysed for the distribution of the visible points of interest among the single buildings.

The results are depicted in Figure 5.7. From the total number of visible points of interest during the inspection, the share of the visible points in the objects with a higher priority is higher than in the objects with a lower priority, while the share in the objects with an equal priority is almost equal. This is valid for all tested configurations. For the 100 m budget only

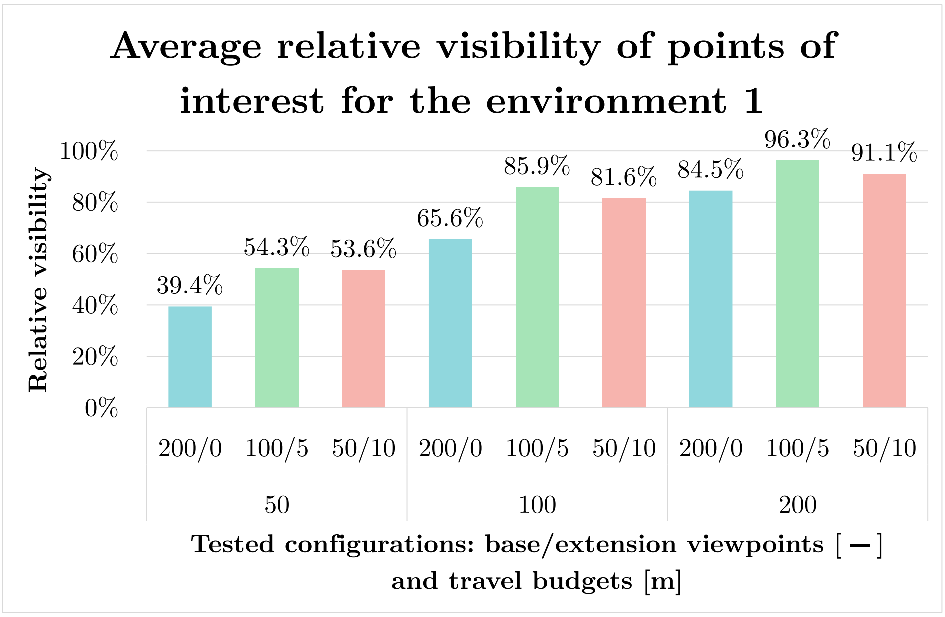


Figure 5.5: Average percentage of the visible points of interest for the environment 1

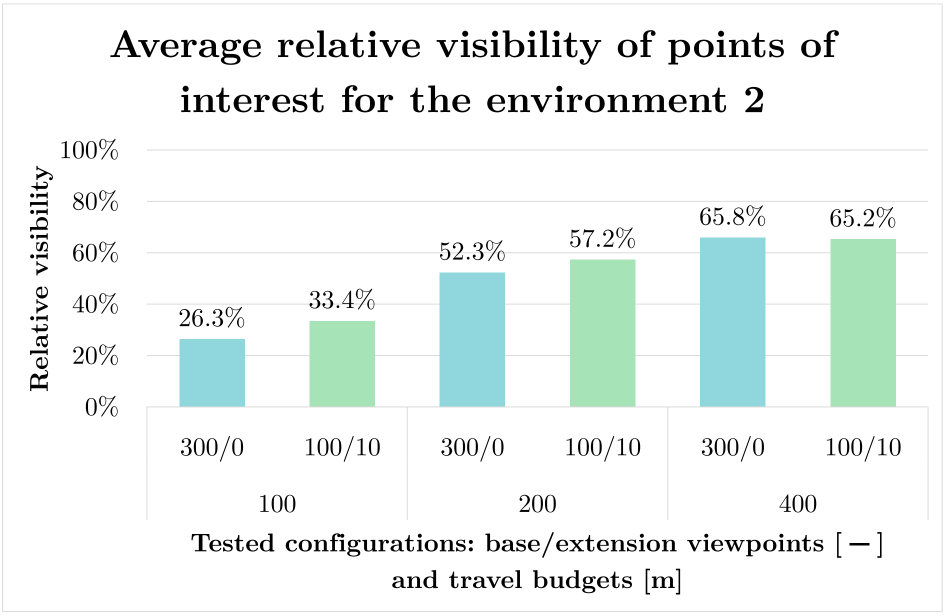


Figure 5.6: Average percentage of the visible points of interest for the environment 2

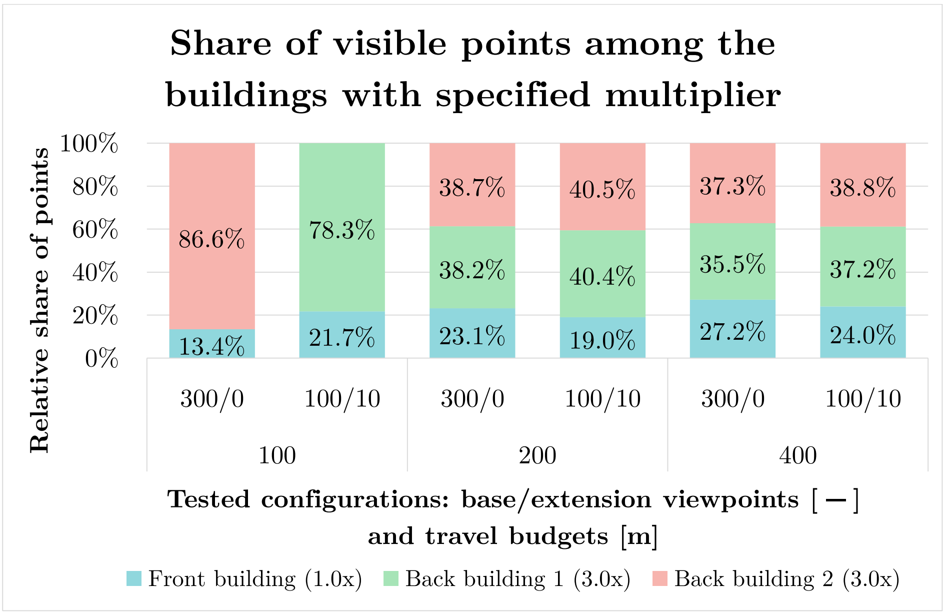


Figure 5.7: Share of the visible points of interest per object of interest in the environment 2

one high-priority object of interest is visited, the points of interest of the low-priority building are "collected" along the path. One of the high-priority buildings is chosen randomly. The configurations with sampling strategy 100/10 perform better also in this case, except for the configuration with the travel budget 100 m, where the share is higher. However, the total number of visible points of interest is higher too.

5.4 Viewpoints extension sampling strategies

To compare the proposed viewpoints extension strategies, Sphere and Ellipse, some of the previous configurations with Sphere extension strategy were repeated with the Ellipse strategy. This concerns configurations with the travel budget 200 m for the environment 1 and the configuration with the same budget for the environment 2. Other parameters remain the same.

Table 5.3: Comparison of the average results for different extension sampling strategies for the environment 1

Base/ext.	Sampling strategy	Average reward [-]	Average count of iterations [-]	Average count of extended viewpoints [-]
100/5	Sphere	24158.4 ± 135.4	540.4 ± 30.1	373.6 ± 19.5
100/5	Ellipse	24270.8 ± 59.7	554.0 ± 26.5	405.4 ± 16.9
50/10	Sphere	22853.6 ± 241.2	407.4 ± 22.4	752.4 ± 52.6
50/10	Ellipse	23671.2 ± 213.3	487.8 ± 18.3	678.4 ± 36.4

The results for the environment 1 are presented in Table 5.3 and the comparison of the average rewards alone is depicted in Figure 5.8. As the data shows, the Ellipse strategy

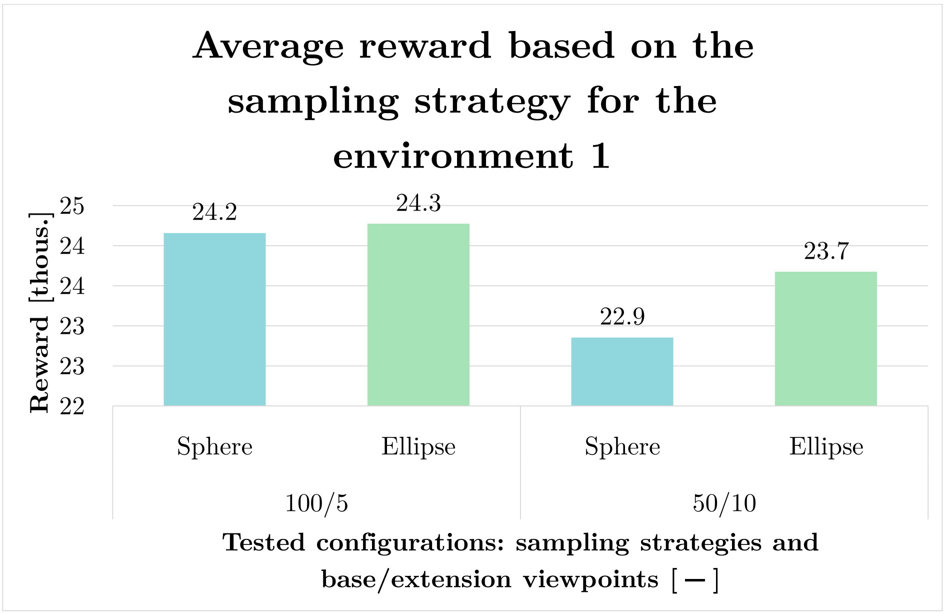


Figure 5.8: Average reward based on the viewpoints extension strategy for the environment 1

outperforms the Sphere strategy both in the total reward as well as in the count of extended points (the results for the extension rate 50/10 are due to the great deviation rather unclear).

Table 5.4: Comparison of the average results for different extension sampling strategies for the environment 2

Base/ext.	Sampling strategy	Average reward [-]	Average count of iterations [-]	Average count of extended viewpoints [-]
100/10	Sphere	100512.6 ± 1087.6	151.0 ± 1.1	1030.8 ± 14.1
100/10	Ellipse	95082.4 ± 953.9	228.4 ± 15.0	490.8 ± 8.0

Contrary to results for the environment 1, the Sphere strategy shows better results in the environment 2 than the Ellipse strategy, in the manner of the reward as well as in the count of the extended viewpoints. This behaviour is probably caused by the rules for the diameter of the sampling sphere, namely a quarter of the average path length between two neighbouring viewpoints in the final path. This average is bigger for the spatious environment 2 and better covers window and door areas, than the sphere in the environment 1. The Ellipse strategy produces too many invalid points for the environment 2, but covers the whole area of doors and windows in the environment 1.

5.5 PRM smoothing

Finally, the PRM techniques and their impact on the reward were examined. Current configurations with the PRM settings 5000/50, hereafter referred as “Normal conditions”, were compared with PRM settings 5000/500, hereafter referred as “Tenfold extension”. Afterwards, “Normal conditions” were compared with the PRM settings 5000/50 with the proposed

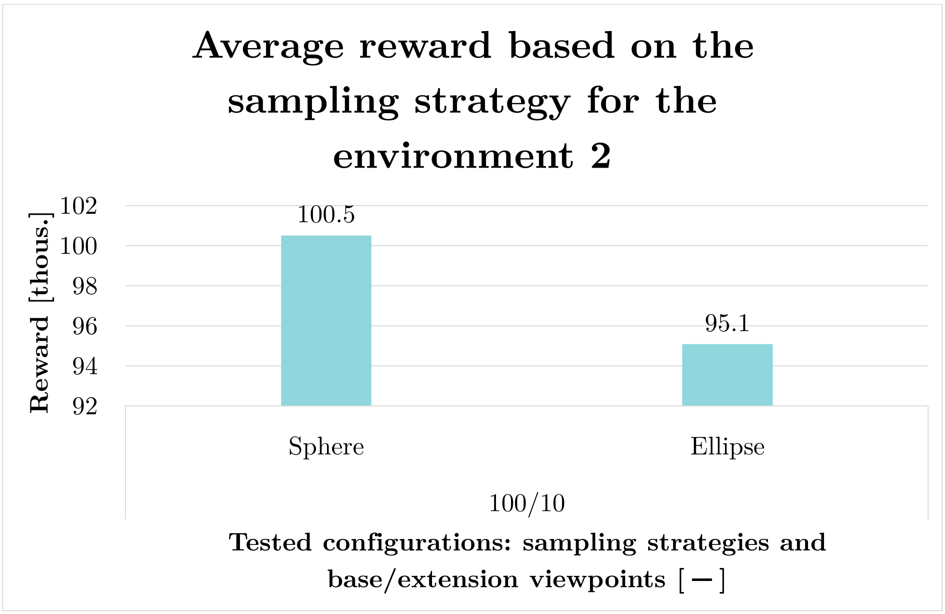


Figure 5.9: Average reward based on the viewpoints extension strategy for the environment 2

concurrent PRM smoothing (the smoothing is thus executed after every iteration). Similarly to the viewpoints generation, the number prior to the slash marks PRM points sampled in the initial learning phase, the number after the slash marks then PRM points sampled in a single extension phase. The tests were performed only in the environment 1, since the PRM smoothing is computationally demanding and the results for the environment 2, with such limited time of execution, would not be satisfying. The travel budget is set to 100 m, the viewpoints extension strategies are 200/0, 100/5 and 50/10 with the viewpoints extension strategy Sphere.

Table 5.5: Comparison of the average results for different PRM extension strategies

PRM* conditions	Base/ext.	Average reward [-]	Average path length [m]	Average count of iterations [-]
Normal conditions	200/0	16463.6 ± 327.5	96.6 ± 1.3	119.6 ± 17.7
Normal conditions	100/5	21553.2 ± 2576.4	98.9 ± 1.5	478.4 ± 181.1
Normal conditions	50/10	20465.4 ± 2091.4	98.9 ± 1.6	351.2 ± 118.2
Tenfold extension	200/0	16713.6 ± 250.7	97.9 ± 0.7	334.2 ± 37.0
Tenfold extension	100/5	20782.2 ± 2056.4	95.9 ± 1.0	277.8 ± 28.6
Tenfold extension	50/10	18426.6 ± 862.5	96.4 ± 1.0	218.4 ± 58.0
PRM smoothing	200/0	16697.4 ± 381.9	99.8 ± 0.1	197.2 ± 50.7
PRM smoothing	100/5	20387.0 ± 1970.7	99.7 ± 0.2	157.4 ± 21.1
PRM smoothing	50/10	18505.6 ± 939.2	99.9 ± 0.1	104.6 ± 46.5

The results are presented in Table 5.5 and further illustrated in Figures 5.10 and 5.11. The graph in Figure 5.10 clearly shows better utilization of the travel budget for the concurrent PRM smoothing, contrary to the tenfold extension rate, where the superfluous number of PRM points even extends the length between viewpoints and the majority of points is removed

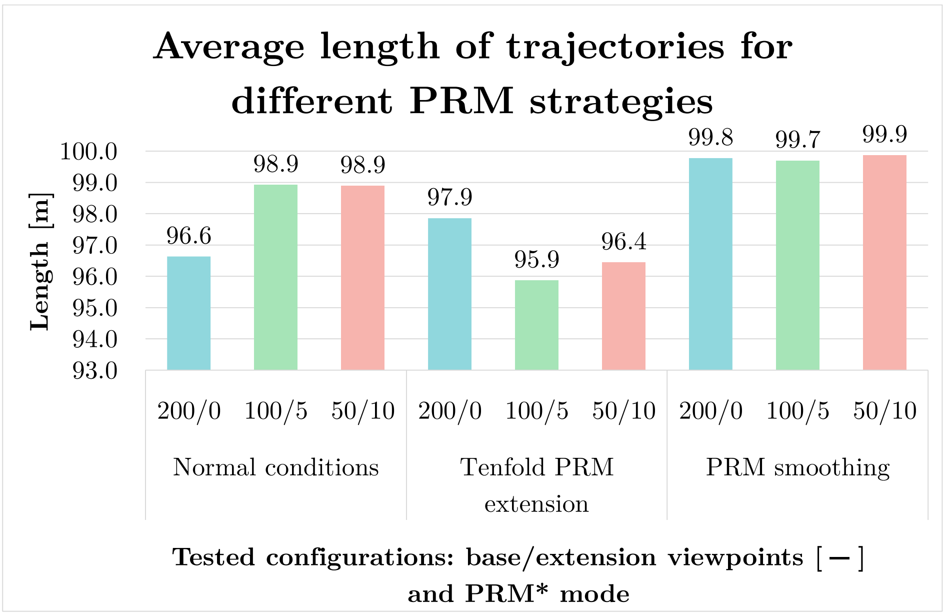


Figure 5.10: Average path length based on the PRM conditions

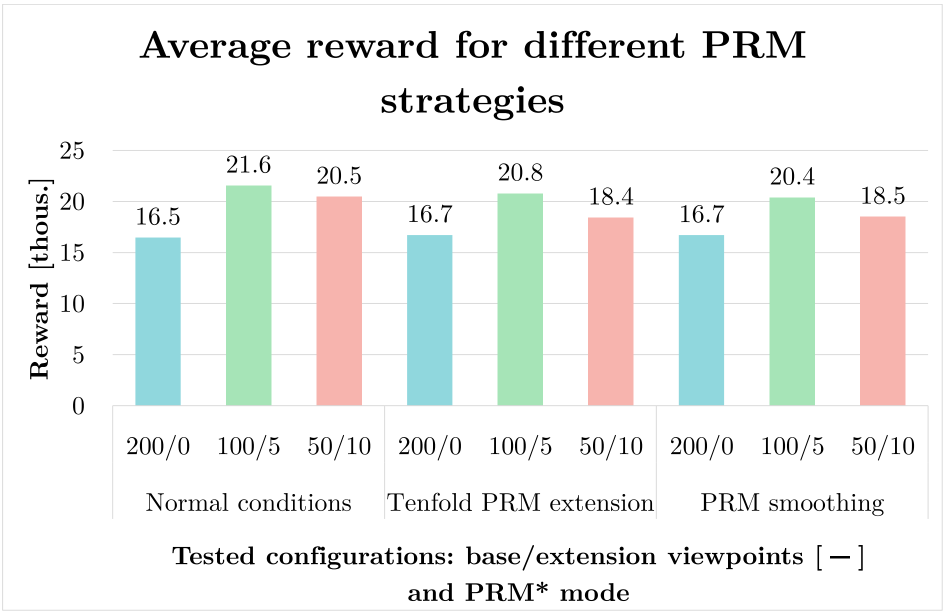


Figure 5.11: Average reward based on the PRM conditions

during the final PRM smoothing. The only exception is the viewpoints extension rate 200/0, which is probably rather random, moreover the difference is not so significant (bearing in mind the standard deviation). However, this improvement does not reflect in the total number of the visible points, according to Figure 5.11. The best rewards are achieved in normal PRM conditions, the worst ones by using the concurrent PRM smoothing. This is caused by the aforementioned computational demands of the PRM smoothing, as shown in the count of elapsed iterations in Table 5.5. On the other hand the dispersion is so great, that some of the results are not so convincing.

Chapter 6

Simulation

The proposed solution was also evaluated using the real-time simulator `Gazebo`[1] and the output OBJ files were visually checked using the `Blender` graphic editor. The main purpose was to verify the validity and the collisionlessness of the generated path as well as the conformity of the visible areas (algorithm output versus simulation output).

The configuration of the used UAV is almost the same as the UAV used in the Fire challenge of the MBZIRC 2020 competition. It is based on the `Tarot 650` base, with 4 propellers, the `mvBlueFOX` camera, `Intel Realsense D435 Depth Camera` (capable of taking depth, infrared and RGB images), `FLIR Lepton 3.5` thermal camera module and other sensors, i.e. Garmin GPS module, Garmin range finder, RPLidar, etc. It does not carry any water gun, fire blanket, or any other extinguisher, because the main objective is only the fire localization part of the firefighting task. The camera is fixed, which means, that the change of the yaw angle is performed by the rotation of the whole UAV. Other angles were not considered (they were fixed). The output of the thermal camera was substituted by the output of the infrared camera, with parameters (the visibility range and FOV) matching the used thermal camera. The UAV is controlled by the Robotic Operating System with an extension and additional firmware created by the Multi-Robot Systems group. The function of the used system is described e. g. in [46] or [47].

The simulated environments are the same as those in Chapter 5. At first, there was an assumption, that the best paths from the referred section will be used also for the simulation, early tests showed, however, that the safety gap between the UAV and any obstacle is too small, which together with the jitter in the GPS signal often caused a collision. Although the position estimator was later switched from classic GPS to RTK GPS[2], the model has been enlarged and some of the configurations were run again – namely configuration with the viewpoints extension rate 100/5 and the travel budget 100 m for the environment 1 and with the viewpoints extension rate 100/10 and the travel budget 100 m for the environment 2.

The majority of the tests was only visual, the output videos for both environments are available on the attached CD, see CD Content in Appendix, or on YouTube[3]. The flow of the incoming messages was unfortunately too high for the used ROSBag recording utility and

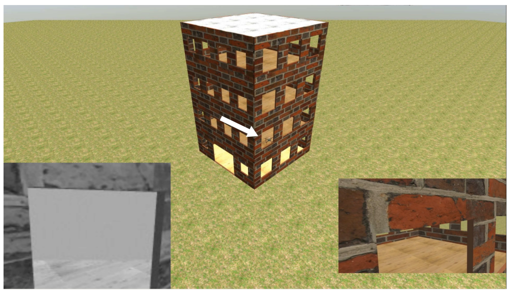
[1]`http://gazebosim.org/`

[2]Real Time Kinematics (RTK) is a technique for better position estimation, as it receives a correction signal besides the signal from the satellites [48].

[3]*Inspection Planning for Firefighting with UAV:* `https://youtu.be/e-Lysl2o-Mg`

installed solid state drive, therefore the quality of the videos is low. To the best of author's knowledge, there is no better way for video capturing from `Gazebo` simulator as well as from the simulated onboard cameras.

The evaluation of the generated path for the environment 1 was at first performed in `Blender`, Figure 6.1b, then the flight was simulated in `Gazebo`, Figure 6.1a. The path does not collide with the building. However, its validity is questionable, as the ideal strategy would be to visit only the opposing middle windows of each floor. The problem may lie in the insufficient initial sampling, or in the time of execution – maybe after few more iterations the path would "fall" in this global optimum.


(a) Screenshot from the *Gazebo* simulation with the camera outputs

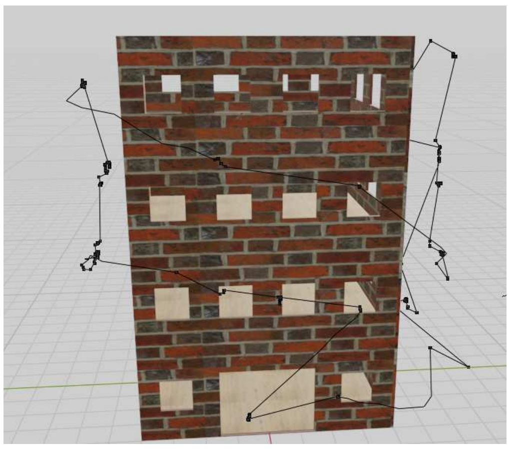

(b) Evaluation of the generated path in *Blender* editor

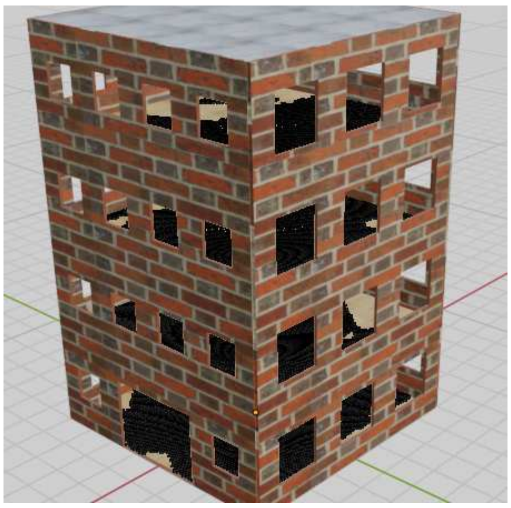
Figure 6.1: Path evaluation for the environment 1

The visible area of interest corresponds more or less with the output of the algorithm which is partly depicted in Figure 6.2. The coverage of all floors is sufficient and appropriate for the fire detection, with regards to the travel budget.

In the same manner, the visual checks were also performed on the generated path for the environment 2, as can be seen in Figure 6.3. The path does not collide with any of the obstacles, and is adjusted accordingly to them. This is highlighted in Figure 6.4a, which is also a screenshot from the simulation. Even though the path is collision-free, the simulation revealed an issue with the system of cost. The UAV exploits the "overlay" of the middle and the left building to capture visible points of interest from both of them, which causes constant rotations between every corresponding viewpoint. It is unrealistic that these rotations would consume no energy. However, this might be avoided by using a revolving camera with a small power consumption, or by redefining the cost system of the algorithm by adding a small price for any rotation.

The environment 2, contrary to the enviroment 1, also better illustrates the function of the viewpoints sampling algorithm. In Figure 6.4b can be seen that viewpoints are generated in groups, near the existing viewpoints, so that the additional distance to the path length would be minimal.

An overview of the visible areas of interest in the environment 2 is depicted in Figure 6.5. The coverage is rather unsatisfactory, for both buildings, but the main cause lies in the low


(a) Overview of the visible points (depicted in black)

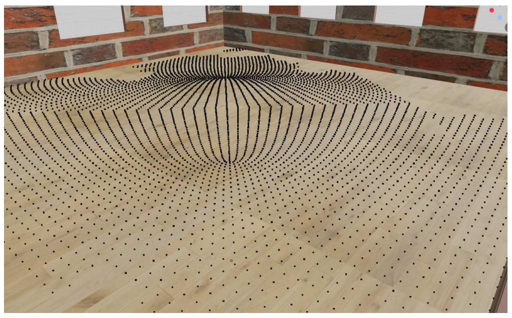
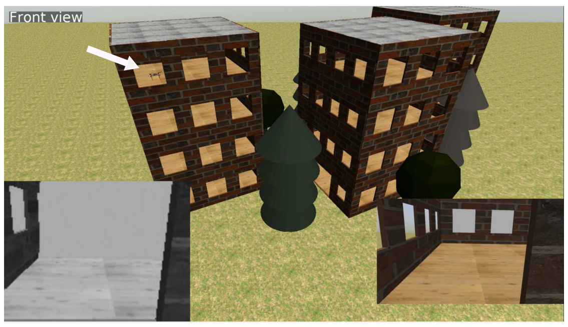

(b) Detail of the visible points for the second floor

Figure 6.2: Visible points of interest in the environment 1



(a) Screenshot from the *Gazebo* simulation with camera outputs

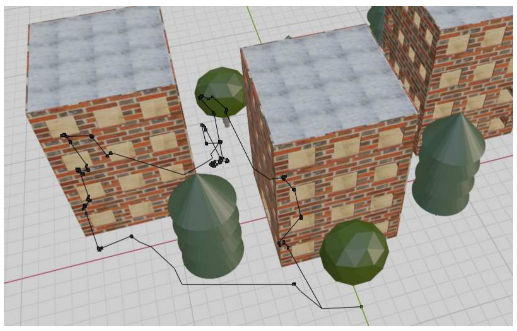
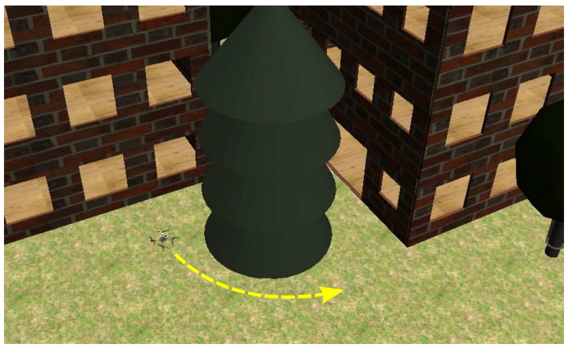

(b) Evaluation of the generated path in *Blender* editor

Figure 6.3: Path evaluation for the environment 2


(a) Collision avoidance in the environment 2

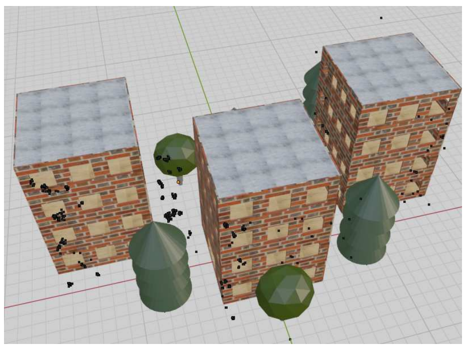
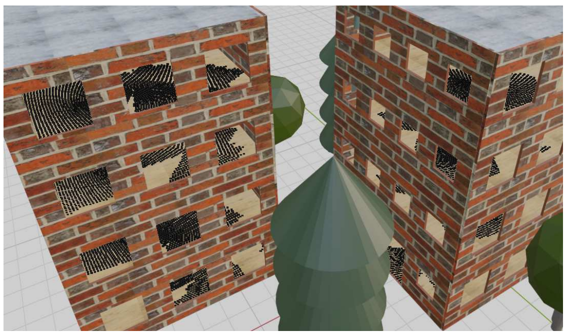

(b) All sampled viewpoints in the environment 2

Figure 6.4: Collision avoidance and viewpoints sampling in the environment 2


(a) Overview of the visible points (depicted in black)

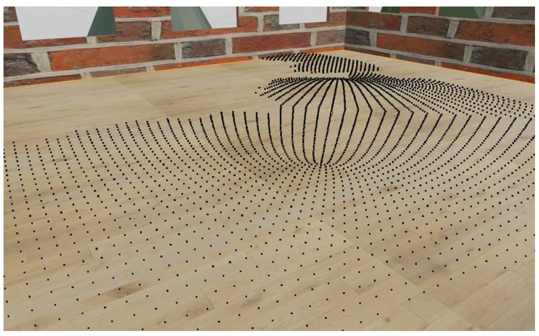
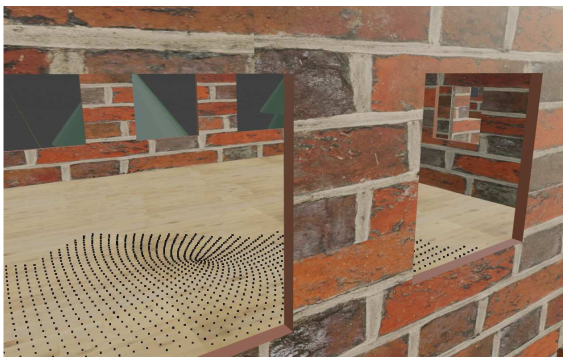

(b) Detail of the visible points for the second floor of the left-most building

Figure 6.5: Visible points of interest in the environment 2


(a) Output of the algorithm, visualized in *Blender* editor

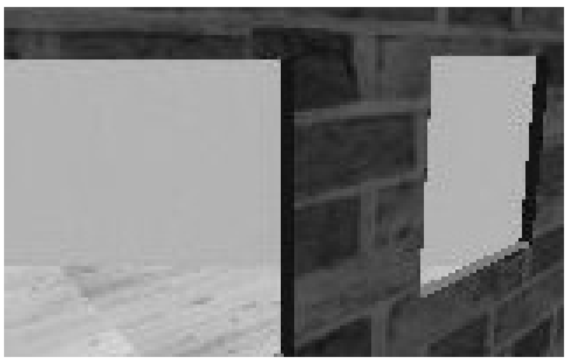

(b) Corresponding image from the onboard infrared camera

Figure 6.6: Comparison of the visible points of interest from the algorithm's output and the simulation

travel budget. The coverage is better in the left building, with the higher weight of points of interest.

The accuracy of the algorithm output, compared to the shot from the infrared camera, is depicted in Figure 6.6.

Chapter 7

Conclusion

The main objective of this thesis was to propose a suitable algorithm for planning of a building inspection path for firefighting. During the inspection flight, the UAV should explore as large area of objects of interest as possible and thus maximize the probability of the fire detection and of its precise localization. The trajectory is further constrained by a travel budget and also by the inaccesibility of the insides of the objects of interest. The path is computed on the basis of given models and parameters of the installed camera such as visibility range, aspect ratio, and field of view.

The assigned challenge was given into a context of similar well-known problems, whose methods of solution were shortly introduced. This concerns the visibility determination methods, e. g. ray tracing, as well as the path planning methods in 2D and 3D environments, e. g. PRM planner. The described techniques were then applied on the assigned challenge, which was divided into two separate problems: the generation phase of the so called viewpoints, and their appropriate connection (the path planning phase).

The generation phase was solved by two different approaches: the space decomposition method and the sampling based method. The first of the named is based on a decomposition of the space on so called tetrahedra and the determination of the visible surface by the back-face culling technique. The sampling based method is inspired by the PRM* algorithm, thus it randomly picks points in the configuration space of the UAV. The sampling is executed in the beginning as well as during the path planning phase. For the extension sampling two strategies were proposed: “Sphere” and “Ellipse”. The visible surface determination is performed by a custom implementation of the ray tracing.

The path planning phase builds upon the existing VNS-PRM* solution for the Physical Orienteering Problem, therefore it combines the VNS methods for the solution of the Orienteering Problem with PRM* methods for the path planning which were improved by the smoothing of the final path. However, the main difference is in the system of rewards, where a simple addition is replaced by a union over the surfaces visible from the corresponding viewpoints which are part of the resulting path.

The final algorithm and its proposed methods were subsequently verified on 24 configurations in two urban environments, representing a single building as well as a housing estate. The results showed that the generated paths meet the stated constraints and sufficiently cover the analysed objects of interest when given a sufficient computing time and resources. The extension of the viewpoints during the path planning phase was proved to perform better than the initial-only samplings. The rates of sampled viewpoints should be chosen with regards to the target environment and the computational complexity. However, none of the proposed sampling strategies clearly outperformed the other one. In the same manner as the rates for the viewpoints generation, the rates of sampled points for the PRM planner should also be chosen with regards to the computational complexity. Any superfluous points decrease the number of elapsed iterations, and thus decrease the final area of the visible surface. The proposed PRM smoothing improves on the one hand the utilization of the given travel budget, on the other hand, this improvement significantly increases the computational complexity and thus any increases of reward were not proved.

Finally, the generated outputs were visually checked in a graphic editor as well as in the `Gazebo` real-time simulator. These tests revealed an issue in the used cost system, where changes of the UAV's orientation were not penalized.

Any future research based on this work could focus on the improvement of the cost system, e. g. adding a small penalization to the transitions in the UAV's orientation. Another suitable area for further development include the sampling strategies. The proposed "Sphere" stategy might be redefined to extend the visibility better, other strategies might be designed as well. To improve the overall performance, the path planning methods might be further optimized by parallelization, or by any other suitable techniques and substitutions.

This work successfully presented one of the possible approaches to the coverage path planning problems with regards to the travel budget and other constraints. Together with a suitable detection algorithm, it might be used for the surveillance of specific building or other urban objects. The field of application is not strictly restricted to the fire detection, the only condition is a relatively greater size of the examined phenomenon than the smallest visible unit of the object. Therefore, it can be used e. g. for the detection and localization of people in rescue operations.

Bibliography

[1] H. Choset and P. Pignon, "Coverage path planning," *Field and Service Robotics*, pp. 203–209, 1998.

[2] A. Zelinsky, R. Jarvis, J. C. Byrne, and S. Yuta, "Planning paths of complete coverage of an unstructured environment by a mobile robot," in *In Proceedings of International Conference on Advanced Robotics*, 1993, pp. 533–538.

[3] Y. Gabriely and E. Rimon, "Spiral-stc," *Proceedings IEEE International Conference on Robotics and Automation*, pp. 954–960, 2002.

[4] P. Cheng, J. Keller, and V. Kumar, "Time-optimal uav trajectory planning for 3d urban structure coverage," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2008, pp. 2750–2757.

[5] P. Janousek and J. Faigl, "Speeding up coverage queries in 3d multi-goal path planning," in *IEEE International Conference on Robotics and Automation*, 2013, pp. 5082–5087.

[6] R. Penicka, J. Faigl, and M. Saska, "Physical orienteering problem for unmanned aerial vehicle data collection planning in environments with obstacles," *IEEE Robotics and Automation Letters*, vol. 4, no. 3, pp. 3005–3012, 2019.

[7] Ministerstvo vnitra, generální ředitelství Hasičského záchranného sboru České republiky, "Statistická ročenka 2019," Praha, 2020. [Online]. Available: https://www.hzscr.cz/soubor/statisticka-rocenka-2019.aspx

[8] G. Pajares, "Overview and current status of remote sensing applications based on unmanned aerial vehicles (uavs)," *Photogrammetric Engineering & Remote Sensing*, vol. 81, no. 4, pp. 281–330, 2015-04-01.

[9] A. Gasparetto, P. Boscariol, A. Lanzutti, and R. Vidoni, "Path planning and trajectory planning algorithms," *Motion and Operation Planning of Robotic Systems*, vol. 2015, pp. 3–27, 2015.

[10] H. Choset, "Coverage for robotics – a survey of recent results," *Annals of Mathematics and Artificial Intelligence*, vol. 31, no. 1/4, pp. 113–126, 2001.

[11] E. Galceran and M. Carreras, "A survey on coverage path planning for robotics," *Robotics and Autonomous Systems*, vol. 61, no. 12, pp. 1258–1276, 2013.

[12] B. Englot and F. Hover, “Inspection planning for sensor coverage of 3d marine structures,” in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2010, pp. 4412–4417.

[13] N. Chesnokov, “The art gallery problem,” pp. 1–13. [Online]. Available: http://www-math.mit.edu/~apost/courses/18.204_2018/Nicole_Chesnokov_paper.pdf

[14] V. Lumelsky, S. Mukhopadhyay, and K. Sun, “Dynamic path planning in sensor-based terrain acquisition,” *IEEE Transactions on Robotics and Automation*, vol. 6, no. 4, pp. 462–472.

[15] E. U. Acar, H. Choset, A. A. Rizzi, P. N. Atkar, and D. Hull, “Morse decompositions for coverage tasks,” *The International Journal of Robotics Research*, vol. 21, no. 4, pp. 331–344, 2016-07-02.

[16] F. P. Preparata and M. I. Shamos, *Computational Geometry: An Introduction*, 1st ed. New York: Springer Science & Business Media, 1985.

[17] T. Oksanen and A. Visala, “Coverage path planning algorithms for agricultural field machines,” *Journal of Field Robotics*, vol. 26, no. 8, pp. 651–668, 2009.

[18] V. Shivashankar, R. Jain, U. Kuter, and D. Nau, “Real-time planning for covering an initially-unknown spatial environment.” 2011.

[19] C. Luo and S. X. Yang, “A bioinspired neural network for real-time concurrent map building and complete coverage robot navigation in unknown environments,” *IEEE Transactions on Neural Networks*, vol. 19, no. 7, pp. 1279–1298, 2008.

[20] C. Luo, S. Yang, D. Stacey, and J. Jofriet, “A solution to vicinity problem of obstacles in complete coverage path planning,” *Proceedings IEEE International Conference on Robotics and Automation*, pp. 612–617, 2002.

[21] M. Torres, D. A. Pelta, J. L. Verdegay, and J. C. Torres, “Coverage path planning with unmanned aerial vehicles for 3d terrain reconstruction,” *Expert Systems with Applications*, vol. 55, pp. 441–451, 2016.

[22] S. Hert, S. Tiwari, and V. Lumelsky, “A terrain-covering algorithm for an auv,” *Underwater Robots*, pp. 17–45, 1996.

[23] A. Randa, T. Taha, L. Seneviratne, J. Dias, G. Cai, P. Z. Peng, and D. F. Lin, “Aircraft inspection using unmanned aerial vehicles,” in *International Micro Air Vehicle Competition and Conference*, 2016, pp. 43–49.

[24] B. Johnson, J. Isaacs, and H. Qi, “A comparative study of methods to solve the watchman route problem in a photon mapping-illuminated 3d virtual environment,” *IEEE Applied Imagery Pattern Recognition Workshop (AIPR)*, pp. 1–8, 2014.

[25] G. Papadopoulos, H. Kurniawati, and N. M. Patrikalakis, “Asymptotically optimal inspection planning using systems with differential constraints,” *IEEE International Conference on Robotics and Automation*, vol. 2013, pp. 4126–4133, 2013.

[26] 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, no. 4, pp. 566–580, 1996.

[27] R. Geraerts and M. Overmars, “Sampling techniques for probabilistic roadmap planners,” *Institute of Information and Computing Sciences.*

[28] S. Karaman and E. Frazzoli, “Sampling-based algorithms for optimal motion planning,” *The International Journal of Robotics Research*, vol. 30, no. 7, pp. 846–894, 2011.

[29] A. Short, Z. Pan, N. Larkin, and S. van Duin, “Recent progress on sampling based dynamic motion planning algorithms,” *IEEE International Conference on Advanced Intelligent Mechatronics (AIM)*, pp. 1305–1311, 2016.

[30] P. Vansteenwegen, W. Souffriau, and D. V. Oudheusden, “The orienteering problem,” *European Journal of Operational Research*, vol. 209, no. 1, pp. 1–10, 2011.

[31] I.-M. Chao, B. L. Golden, and E. A. Wasil, “A fast and effective heuristic for the orienteering problem,” *European Journal of Operational Research*, vol. 88, no. 3, pp. 475–489, 1996.

[32] A. Gunawan, H. C. Lau, and P. Vansteenwegen, “Orienteering problem,” *European Journal of Operational Research*, vol. 255, no. 2, pp. 315–332, 2016.

[33] P. Hansen and N. Mladenović, “Variable neighborhood search,” *European Journal of Operational Research*, vol. 130, no. 3, pp. 449–467, 2001.

[34] A. Glassner, “An overview of ray tracing,” in *An Introductiton to Ray Tracing*, 2nd ed. San Francisco: Morgan Kaufmann, 2002, pp. 1–26.

[35] J. Bell, “Visible surface determination,” Chicago, Illinois, 2019. [Online]. Available: https://www.cs.uic.edu/~jbell/CourseNotes/ComputerGraphics/VisibleSurfaces.html

[36] R. J. Segura and F. R. Feito, “Algorithms to test ray-triangle intersection. comparative study,” in *The 9-th International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision*, 2001, pp. 76–81.

[37] D. Badouel, “An efficient ray — polygon intersection,” *Graphics Gems*, pp. 390–393, 1990.

[38] T. Möller and B. Trumbore, “Fast, minimum storage ray-triangle intersection,” *Journal of Graphics Tools*, vol. 2, 2005.

[39] H. Si, “Tetgen,” Berlin, 2013. [Online]. Available: http://wias-berlin.de/software/tetgen/1.5/doc/manual/manual.pdf

[40] E. W. Weisstein, “Barycentric coordinates,” c1999-2020. [Online]. Available: http://mathworld.wolfram.com/BarycentricCoordinates.html

[41] “Wavefront obj file format,” 2020. [Online]. Available: https://www.loc.gov/preservation/digital/formats/fdd/fdd000507.shtml#notes

[42] "Pseudo-random number generation," 2020. [Online]. Available: https://en.cppreference.com/w/cpp/numeric/random

[43] M. Figueiredo, L. Marcelino, and T. Fernando, "A survey on collision detection techniques for virtual environments," *Proc. of V Symposium in Virtual Reality, Brasil*, vol. 307, 2002.

[44] P. Krähenbühl and V. Koltun, "Efficient inference in fully connected crfs with gaussian edge potentials," in *Advances in Neural Information Processing Systems 24*, J. Shawe-Taylor, R. S. Zemel, P. L. Bartlett, F. Pereira, and K. Q. Weinberger, Eds. Curran Associates, Inc., 2011, pp. 109–117.

[45] M. Muja and D. Lowe, "Flann - fast library for approximate nearest neighbors," *Computer Science Department, University of British Columbia*, 2013. [Online]. Available: https://www.cs.ubc.ca/research/flann/uploads/FLANN/flann_manual-1.8.4.pdf

[46] T. Baca, D. Hert, G. Loianno, M. Saska, and V. Kumar, "Model predictive trajectory tracking and collision avoidance for reliable outdoor deployment of unmanned aerial vehicles," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2018, pp. 6753–6760.

[47] V. Spurný, T. Báča, M. Saska, R. Pěnička, T. Krajník, J. Thomas, D. Thakur, G. Loianno, and V. Kumar, "Cooperative autonomous search, grasping, and delivering in a treasure hunt scenario by a team of unmanned aerial vehicles," *Journal of Field Robotics*, 10 2018.

[48] *An Introduction to GNSS*, Second edition ed. Calgary, Alberta, Canada: NovAtel Inc., 2015.

Appendices

Full results of testing

Below are the raw results of the testing phase, with computed average values and highlighted best tests. The number of final parameters may differ for various configurations, as they are used for evaluation of different aspects of the algorithm.

Table 1: List of all tested configurations

	Environ-ment	Viewpoints strategy (base/extension)	Extension strategy	PRM points generation (base/extension)	Concurrent path smoothing?	Budget [m]
1	1	200/0	–	5000/50	no	50
2	1	200/0	–	5000/50	no	100
3	1	200/0	–	5000/50	no	200
4	1	100/5	Sphere	5000/50	no	50
5	1	100/5	Sphere	5000/50	no	100
6	1	100/5	Sphere	5000/50	no	200
7	1	50/10	Sphere	5000/50	no	50
8	1	50/10	Sphere	5000/50	no	100
9	1	50/10	Sphere	5000/50	no	200
10	1	100/5	Ellipse	5000/50	no	200
11	1	50/10	Ellipse	5000/50	no	200
12	1	200/0	–	5000/500	no	100
13	1	100/5	Sphere	5000/500	no	100
14	1	50/10	Sphere	5000/500	no	100
15	1	200/0	–	5000/50	yes	100
16	1	100/5	Sphere	5000/50	yes	100
17	1	50/10	Sphere	5000/50	yes	100
18	2	300/0	–	5000/150	no	100
19	2	300/0	–	5000/150	no	200
20	2	300/0	–	5000/150	no	300
21	2	100/10	Sphere	5000/150	no	100
22	2	100/10	Sphere	5000/150	no	200
23	2	100/10	Sphere	5000/150	no	300
24	2	100/10	Ellipse	5000/150	no	200

Table 2: Results for the configuration 1

	Reward [-]	Length [m]	Iterations [-]
1	9932	47.319	106
2	9940	46.834	104
3	9716	47.562	110
4	9217	47.447	119
5	**10685**	**47.322**	**184**
Avg.	9898	47.297	125

Table 3: Results for the configuration 2

	Reward [-]	Length [m]	Iterations [-]
1	**17260**	**99.723**	**122**
2	16233	93.966	184
3	15792	94.600	90
4	17224	99.892	118
5	15809	94.977	84
Avg.	16463.6	96.631	120

Table 4: Results for the configuration 3

	Reward [-]	Length [m]	Iterations [-]
1	20938	188.880	340
2	**21719**	**196.349**	**269**
3	21009	199.095	145
4	21181	199.719	294
5	21185	190.352	330
Avg.	21206.4	194.879	276

Table 5: Results for the configuration 4

	Reward [-]	Length [m]	Iterations [-]
1	13335	47.391	604
2	**15584**	**48.138**	**551**
3	12516	47.988	626
4	14879	47.520	475
5	11860	48.183	579
Avg.	13634.8	47.844	567

Table 6: Results for the configuration 5

	Reward [-]	Length [m]	Iterations [-]
1	21763	99.927	410
2	20688	99.991	440
3	22297	99.895	478
4	20537	99.950	530
5	**22481**	**94.849**	**534**
Avg.	21553.2	98.922	478

Table 7: Results for the configuration 6

	Reward [-]	Length [m]	Iterations [-]	Inserted ext. viewpoints [-]
1	23995	198.130	647	383
2	24388	196.729	539	374
3	**24551**	**189.183**	**465**	**331**
4	23811	187.799	508	339
5	24047	190.988	543	441
Avg.	24158.4	192.566	540	374

Table 8: Results for the configuration 7

	Reward [-]	Length [m]	Iterations [-]
1	**14504**	**47.440**	**367**
2	14008	47.361	341
3	12188	49.992	395
4	13794	47.966	364
5	12807	47.750	323
Avg.	13460.2	48.102	358

Table 9: Results for the configuration 8

	Reward [-]	Length [m]	Iterations [-]
1	20975	99.968	325
2	**22137**	**99.999**	**289**
3	18586	94.603	351
4	19692	99.914	434
5	20937	99.968	357
Avg.	20465.4	98.890	351

Table 10: Results for the configuration 9

	Reward [-]	Length [m]	Iterations [-]	Inserted ext. viewpoints [-]
1	22204	199.835	363	872
2	23011	199.953	377	795
3	22667	199.783	380	839
4	22718	189.673	434	648
5	**23668**	**188.245**	**483**	**608**
Avg.	22853.6	195.498	407	752

Table 11: Results for the configuration 10

	Reward [-]	Length [m]	Iterations [-]	Inserted ext. viewpoints [-]
1	**24395**	**193.023**	**554**	**428**
2	24201	197.980	653	387
3	24371	194.463	546	444
4	24314	194.006	511	419
5	24073	193.384	506	349
Avg.	24270.8	194.571	554	405

Table 12: Results for the configuration 11

	Reward [-]	Length [m]	Iterations [-]	Inserted ext. viewpoints [-]
1	**24028**	**193.624**	**445**	**738**
2	22938	192.984	499	600
3	23940	199.876	504	692
4	23436	193.744	449	772
5	24014	199.749	542	590
Avg.	23671.2	195.995	488	678

Table 13: Results for the configuration 12

	Reward [-]	Length [m]	Iterations [-]
1	15929	95.794	296
2	**17298**	**99.892**	**433**
3	16436	97.217	386
4	16719	97.192	338
5	17186	99.204	218
Avg.	16713.6	97.860	334

Table 14: Results for the configuration 13

	Reward [-]	Length [m]	Iterations [-]
1	20421	95.962	283
2	**21688**	**95.811**	**263**
3	20677	95.202	271
4	19949	96.692	291
5	21176	95.675	281
Avg.	20782.2	95.868	278

Table 15: Results for the configuration 14

	Reward [-]	Length [m]	Iterations [-]
1	18285	95.519	230
2	18382	95.280	214
3	**18784**	**96.261**	**218**
4	18196	95.822	224
5	18486	99.338	206
Avg.	18426.6	96.444	218

Table 16: Results for the configuration 15

	Reward [-]	Length [m]	Iterations [-]
1	15306	99.825	82
2	17320	99.946	218
3	16668	99.387	83
4	16726	99.896	263
5	**17467**	**99.830**	**340**
Avg.	16697.4	99.777	197

Table 17: Results for the configuration 16

	Reward [-]	Length [m]	Iterations [-]
1	20208	99.972	165
2	21262	99.092	159
3	**22007**	**99.449**	**173**
4	17906	99.989	131
5	20552	99.983	159
Avg.	20387	99.697	157

Table 18: Results for the configuration 17

	Reward [-]	Length [m]	Iterations [-]
1	**19174**	**99.909**	**106**
2	18931	99.991	99
3	17720	99.987	104
4	18332	99.649	121
5	18371	99.821	93
Avg.	18505.6	99.871	105

Table 19: Results for the configuration 18

	Reward [-]	Length [m]	Iterations [-]	Share of visible points of int. [-]
1	46137	96.665	157	
2	42452	94.880	82	
3	**48552**	**96.882**	**230**	**2379:0:15391**
4	47210	99.607	205	
5	47003	95.814	147	
Avg.	46270.8	96.769	164	

Table 20: Results for the configuration 19

	Reward [-]	Length [m]	Iterations [-]	Share of visible points of int. [-]
1	93449	199.838	117	
2	**93627**	**193.981**	**160**	**8535:14096:14268**
3	93427	199.508	288	
4	87020	199.592	265	
5	91351	199.483	215	
Avg.	91774.8	198.480	209	

Table 21: Results for the configuration 20

	Reward [-]	Length [m]	Iterations [-]	Share of visible points of int. [-]
1	114239	398.804	293	
2	112332	382.381	149	
3	115041	386.521	302	
4	116359	383.000	281	
5	**120257**	**384.867**	**259**	**13340:17384:18255**
Avg.	115645.6	387.115	257	

Table 22: Results for the configuration 21

	Reward [-]	Length [m]	Iterations [-]	Share of visible points of int. [-]
1	58488	96.146	204	
2	**62459**	**96.301**	**213**	**5279:19054:0**
3	54947	96.043	220	
4	56461	95.323	233	
5	60858	97.441	228	
Avg.	58642.6	96.251	220	

Table 23: Results for the configuration 22

	Reward [-]	Length [m]	Iterations [-]	Share of visible points of int. [-]
1	**104114**	**191.816**	**150**	**7562:16070:16114**
2	101445	191.885	148	
3	100153	191.468	150	
4	97727	191.578	153	
5	99124	192.009	154	
Avg.	100512.6	191.751	151	

Table 24: Results for the configuration 23

	Reward [-]	Length [m]	Iterations [-]	Share of visible points of int. [-]
1	108535	397.122	146	
2	**121979**	**382.254**	**140**	**11627:18001:18783**
3	109952	385.110	146	
4	112546	399.046	144	
5	119556	380.994	138	
Avg.	114513.6	388.905	143	

Table 25: Results for the configuration 24

	Reward [-]	Length [m]	Iterations [-]	Inserted ext. viewpoints [-]
1	96479	194.660	260	494
2	93546	195.270	269	500
3	**97337**	**195.459**	**195**	**467**
4	92213	193.934	210	480
5	95837	195.111	208	513
Avg.	95082.4	194.887	228	491

CD Content

In Table 26 are listed names of all root directories on CD.

Directory name	**Description**
final-method_sources	source code of the final algorithm
space-decomposition_sources	source codes for the space decomposition
simulation	simulation videos for both environments
thesis	the thesis in pdf format
thesis_sources	latex source codes

Table 26: CD Content