

A Matter of Perspective – Three-dimensional Placement of Multiple Cameras to Maximize their Coverage

Von der Universität Bayreuth zur Erlangung des Grades eines
Doktors der Naturwissenschaften (Dr. rer. nat.) genehmigte
Abhandlung

Maria L. Hänel

aus Lichtenfels

| | |
|----------------------|--|
| 1. Gutachter: | Prof. Dr. Dominik Henrich, Universität Bayreuth |
| 2. Gutachter: | Prof. Dr. Jörg Rambau, Universität Bayreuth |
| 3. Gutachter: | Dr. Carola Schönlieb, University of Cambridge |
| Tag der Einreichung: | 20.1.2015 |
| Tag des Kolloquiums: | 24.4.2015 |

Version of October 13, 2015

Danke an all diejenigen, die ...

- ... mich in allen wichtigen Entscheidungen unterstützt und in allen Launen ertragen haben
- ... mich motiviert und hilfreich angeleitet haben,
- ... an mich geglaubt haben,
- ... mir ihre ehrliche Meinung gesagt haben,
- ... mit mir angeregt fachlich und nicht fachlich diskutiert haben,
- ... trotz des recht einseitigen Kontakts in letzter Zeit zu mir gestanden haben,
- ... immer ein offenes Ohr und eine offene Tür angeboten haben, wenn ich sie gebraucht habe,
- ... mir die anstrengende Zeit verkürzt und versüßt haben.

Abstract

In this thesis, the problem of optimally placing and orienting multiple cameras by a suitable algorithm is considered. The algorithm optimizes the positions and orientations of the cameras in a given three-dimensional environment to approximate a target, such as a human, most accurately, or to maximally cover the important regions of the environment. The more precise the approximation, the easier it is to protect the human. In general, the better the environment is covered in the relevant regions, the easier an approximation can be formed.

Such a method requires a *visibility analysis*, a simulation of the field of view of each camera. The visibility analysis is time consuming, since it can only be done geometrically. Previous work severely simplifies the optimization, e.g., by only considering a two-dimensional top view of the environment, by defining the possible mounting spots of the cameras in a discretized way, or by neglecting visual obstacles. In contrast to the existing work, the proposed algorithms place multiple cameras in a three-dimensional environment on a continuous domain considering static and dynamic visual obstacles.

The three algorithms are globally convergent and establish a feasible solution at any time after a short initialization phase. Several strategies are developed for decreasing the computation time of the methods: Some of the strategies decrease the number of objective function calls, some accelerate the visibility analysis, and some increase the convergence rate of the solver. In general, the number of function evaluations of the solvers is as low as in case of a local solver although it is applied to functions without gradient. It is practical for a variety of problems whose objective functions are non-convex, stair-cased, expensive, or given only as a black-box.

Furthermore, the computation is accelerated by the following strategies: The optimization methods are ready to incorporate *prior* information about good and bad placements, e.g., to prevent cameras from facing a wall of the room right from the beginning. Additionally, they can use the symmetry of a function and the fact that a function is substantially cheaper on subspaces of the domain for an acceleration. Lastly, one of the solvers has been proved to converge even when some of the function calls are computed in parallel. With the proposed architecture for camera placement, a system has been developed that efficiently generates provably good positions and orientations of cameras in three dimensions that a human might not think of. This is demonstrated on several synthetic and realistic examples.

In order to establish prior information about the objective function and its properties, the function is analyzed in the beginning of the thesis. The analysis shows the regions which are covered by the cameras or the regions that define the approximation of the target. Both regions have a polyhedral shape. Vertices, faces, and the contact of faces with specified points of the environment are classified depending on the camera parameters. This is particularly interesting to investigate the differentiability and stair-casing of the objective function.

Zusammenfassung

In dieser Arbeit wird eine Methode zur optimalen Platzierung und Orientierung mehrerer Kameras erstellt. Der Algorithmus optimiert die Positionen und Orientierungen der Kameras in einer vorgegebenen dreidimensionalen Umgebung, um ein Objekt, zum Beispiel eine Person, möglichst genau zu approximieren, oder die wichtigen Bereiche der Umgebung möglichst gut auszuleuchten. Denn je genauer die Approximation der Person ist, desto einfacher ist es, sie zu beschützen. Außerdem ist eine Approximation einfacher zu erstellen, je mehr von der Umgebung, in der sich das Objekt befindet, von den Kameras abgedeckt wird.

In einem solchen Programm wird eine Sichtbarkeitsanalyse benötigt, eine Analyse die zu einer gegebenen Umgebung den Sichtbereich einer Kamera bestimmt. Diese Analyse muss automatisch geometrisch berechnet werden und ist zeitaufwendig. In der existierenden Literatur wird die Sichtbarkeitsanalyse deshalb stark vereinfacht, beispielsweise wird nur die zweidimensionale Draufsicht auf einen Raum als Umgebung verwendet, Hindernisse, wie Mauern oder andere Menschen, werden nicht berücksichtigt oder es gibt nur diskrete Montagepunkte für die Kameras. Im Gegensatz zu existierenden Arbeiten, werden hier mehrere Kameras in einer dreidimensionalen Umgebung auf einem kontinuierlichen Definitionsbereich, also beispielsweise im ganzen Raum, unter Berücksichtigung von Hindernissen betrachtet.

Die drei erstellten Algorithmen sind zuverlässig, global konvergent und gelangen nach einer kurzen Initialisierungsphase zu einer zulässigen Lösung. Mehrere Strategien werden vorgestellt, um deren Berechnungszeit zu verkürzen: Die Reduktion der Funktionsaufrufe, die Beschleunigung der Sichtbarkeitsanalyse und die Beschleunigung der Konvergenz. Im Allgemeinen ist die Anzahl der Funktionsaufrufe der Methoden bis zur Terminierung vergleichbar mit lokalen Methoden, obwohl sie keinen analytischen Gradienten benötigen. Die Methoden sind für eine Vielzahl an Problemen interessant, deren Zielfunktion quantisiert, zeitaufwendig, nicht konvex oder nur als "Blackbox" gegeben ist. Mit der vorgestellten Architektur für Kameraplatzierung ist ein System entwickelt worden, das effizient gute Positionen und Orientierungen der Kameras in drei Dimensionen erstellt, an die der Mensch nicht unbedingt gedacht hätte. Das ist an mehreren synthetischen und praktischen Beispielen getestet worden.

Außerdem wurde die Berechnung durch die folgenden Strategien beschleunigt: Die Optimierungsmethoden können vorher bekannte Informationen über gute und schlechte Platzierungen berücksichtigen, beispielsweise um von vornherein auszuschließen, dass die Kameras an einer Wand zu dieser hinorientiert werden. Zusätzlich können die Methoden die Symmetrie einer Funktion und die Tatsache, dass die Funktion auf Unterräumen des Definitionsbereichs wesentlich einfacher zu berechnen ist, ausnutzen um die gesamte Berechnung zu beschleunigen. Zuletzt ist sogar eine der Optimierungsmethoden bewiesenermaßen konvergent, wenn die Zielfunktionsaufrufe parallel berechnet werden.

Um vorher bekannte Informationen und Eigenschaften von der Zielfunktion unseres Problems abzuleiten, wird die Funktion zu Beginn der Arbeit analysiert: Die Analyse beweist, dass die Gestalt der Bereiche, die durch die Kameras abgedeckt werden, oder der Bereiche, die die Approximation des Objektes definieren, ein entartetes Polyeder ist. Die Ecken, Flächen und das Aufeinandertreffen von Ecken und Flächen werden in Abhängigkeit der Kameraparameter klassifiziert. Diese Klassifizierungen sind besonders wichtig für die Untersuchung der Differenzierbarkeit und der Quantisierung der Zielfunktion.

Contents

| | | |
|----------|---|-----------|
| 1 | Introduction to Camera Placement | 1 |
| 1.1 | Camera Network Optimization | 3 |
| 1.1.1 | Parameters | 3 |
| 1.1.2 | Challenges | 5 |
| 1.1.3 | Aims | 7 |
| 1.2 | Problem Definition | 8 |
| 1.3 | Related Work | 10 |
| 1.3.1 | Camera Placement | 10 |
| 1.3.2 | Coverage and Visibility | 12 |
| 1.3.3 | Optimization | 13 |
| 1.4 | Overview | 15 |
| 2 | Deduction and Properties of an Objective Function for Camera Placement | 17 |
| 2.1 | Coverage of Multiple Cameras | 18 |
| 2.1.1 | Detectable Coverage | 19 |
| 2.1.2 | 3D Background-Subtraction Method | 23 |
| 2.1.3 | Conservative Approximation | 25 |
| 2.1.4 | Increased Reliability | 28 |
| 2.2 | Implementation of the Coverage of Multiple Cameras | 31 |
| 2.2.1 | Basic Implementation | 32 |
| 2.2.2 | Acceleration of the Evaluation | 35 |
| 2.2.3 | Sequential Evaluations | 36 |
| 2.3 | Geometry of the Coverage of Multiple Cameras | 39 |
| 2.3.1 | Shape | 39 |
| 2.3.2 | Volume | 43 |
| 2.3.3 | Faces and Vertices | 44 |
| 2.3.4 | Incidences due to Variable Camera Parameters | 49 |
| 2.4 | Properties of the Volume of the Coverage of Multiple Cameras | 55 |
| 2.4.1 | Continuity and Differentiability | 55 |
| 2.4.2 | Non-Convexity | 60 |
| 2.4.3 | Symmetry | 61 |
| 2.4.4 | Stair-casing | 63 |
| 2.5 | Summary | 67 |

| | | |
|----------|--|------------|
| 3 | Global Optimization of Costly, Non-differentiable, or Stair-cased Black-box Functions | 69 |
| 3.1 | Optimization Procedure utilizing a Radial Basis Function as a Response Surface Model | 70 |
| 3.1.1 | Exclusion Area Method with a Response Surface Model | 71 |
| 3.1.2 | Radial Basis Functions as a Response Surface Model | 74 |
| 3.1.3 | Update Rules and the Incorporation of Prior Information | 79 |
| 3.1.4 | Convergence | 84 |
| 3.2 | Optimization Procedure utilizing a Block Coordinate Ascent | 88 |
| 3.2.1 | Block Coordinate Ascent | 89 |
| 3.2.2 | Cost Reduction in Camera Placement | 90 |
| 3.2.3 | Challenges Concerning the Convergence | 91 |
| 3.2.4 | Block Coordinate Ascent for Non-differentiable, Non-separable Functions | 97 |
| 3.3 | Experimental Setup | 102 |
| 3.3.1 | Test Problems | 102 |
| 3.3.2 | Implementation Details | 105 |
| 3.3.3 | Focus of the Experiments and Test Parameter | 106 |
| 3.3.4 | Hardware Configuration | 109 |
| 3.4 | Experimental Results | 109 |
| 3.4.1 | Comparison to State of the Art | 110 |
| 3.4.2 | Efficiency and Accuracy | 112 |
| 3.4.3 | Dependency on Type and Slope of the Objective Function | 115 |
| 3.4.4 | Incorporation of Symmetry as Prior Information | 117 |
| 3.4.5 | Inner and Outer Termination Criteria | 118 |
| 3.5 | Summary | 121 |
| 4 | Application Examples | 125 |
| 4.1 | Hardware and Software Configuration | 126 |
| 4.2 | Robot Cell | 127 |
| 4.2.1 | Setup | 127 |
| 4.2.2 | Qualitative Results | 128 |
| 4.3 | Generic Room with a Human Walk | 130 |
| 4.3.1 | Setup | 130 |
| 4.3.2 | Qualitative Results | 131 |
| 4.4 | Quantitative Results | 134 |
| 4.5 | Summary | 136 |
| 5 | Conclusion | 139 |
| 5.1 | Summary | 139 |
| 5.2 | Contributions | 142 |
| 5.3 | Future Work | 145 |
| A | Bibliography | 147 |
| B | Symbols and Definitions | 159 |

Chapter 1

Introduction to Camera Placement

Cameras are deployed in private as well as in public space, in malls, in museums, or in the military. The purpose of the surveillance can vary between capturing a scene, documenting the course of an action, reconstructing, and tracking objects such as people. In modern production facilities, the requirement of a surveillance system goes beyond documenting the scene and has developed in direction of automation and self controlling.

Take the following example: In [75, 78, 80] humans and robots are supposed to share a common working area. Conventionally, humans use their visual sense in order to avoid collisions. However, in the context of fast working, heavy, or harmful machines a human's reaction speed can be insufficient. Such a scene is illustrated in Figure 1.1 to the left. In order to protect the human collaborator from any harm, safety strategies need to be developed: Potentially dangerous situations have to be detected early on and, as a consequence, the movement of the machine needs to be decelerated, stopped, or even redesigned. Using a camera as a sensor is attractive as it captures the human visually, in contrast to a contact sensor which needs to establish contact to the object at hand. Thus, a dangerous situation can be detected in time with enough space left between machine and human coworker.

Computer vision approaches are vital to automate surveillance tasks as the above and to plan locomotion. This is shown by various publications in such diverse fields as sidewalk and transport safety [92, 98], tracking [25], gesture recognition [28], or other surveillance tasks [49, 66], to name just a few. The significance of surveillance systems in our community can be illustrated in the context of toy industry quite vividly. Since very recently, several consumer electronic stores make sure that even the youngest of us boost sales of drones and remote controlled cars with attached cameras, e.g., the sales of Parrot's quadcopter [114]. In fact, the sales in this area have enough substance to keep the research permanently busy with new ideas. The significance of impressing others, e.g., by showing off with daring self portraits, is reaching a new height with the development of self-portraiting drones that follow you and return to you, [36, 76].

Nevertheless, the benefit of cameras in areas beyond toy industry is undeniable. If the physical integrity of humans is primary task of a surveillance system, then the system essentially needs to be failure-resistant. To this aim, usually more than one sensor is attached to walls or ceilings or, in case of dynamic systems, to drones. The images of these sensors are sent to one common or several distributed computing

cores. The mounting, communication, cores, and sensors are called a *sensor network*. The sensor network's task is to capture objects with each sensor, to process, to communicate, and to merge the images in order to reconstruct or track the objects, and to initiate responses based on the object's behavior. In the example of human-robot cooperation above, regions of the work cell where the human could be located are identified by a camera. Merging these associated regions means intersecting them and thus producing a silhouette of the human, illustrated in Figure 1.1 to the right. Generally, the more precise the silhouette of the human, the more exact is the robot's idea when to initiate a safety strategy, such as redesigning the robot's route.



Figure 1.1: Physical setup (left) and three-dimensional reconstruction of the human generated by [79] (right) ensuring the safety of humans when coexisting or cooperating with a robot in the same working area. The three-dimensional silhouette of the collaborator is constructed from the video streams of eight cameras in real-time.

Even the best safety strategy is obsolete if the *quality* of the sensor network is insufficient, e.g., if the surveillance area is poorly covered by the sensors in the relevant regions. This is why primarily the following questions need to be answered: How many sensors need to be applied? What are the maximum expenses that a sensor network may cost? Where, according to the surveillance area, do the sensors of a network need to be located in order to maximize the quality of the sensor network? How do the sensors need to be oriented or adjusted otherwise? Traditionally, heuristics are used to obtain a good placement of sensors, manually. Among these is the attachment “preferably in the corners of the room” when placing camera sensors – since they do not interfere with the action in the room, there, and a camera with a limited field of view is not limited by the walls in its back any further.

One way to get a more precise prediction about the quality of a specific setup is analysing the sensor coverage. Here, the *coverage* of a sensor is a particular region or set of objects in the room which can be observed by the sensor “in a certain way”. If these regions are too small or the objects are not sufficiently covered, the surveillance system may fail. When performed for camera sensors, analysing the coverage is also called *visibility analysis*, c.f. [101]. The coverage as well as the visibility needs to be deduced geometrically, taking into account the surveillance area, the placement of the sensors, and other obstacles in the scene. Then, for a particular set of network parameters, e.g., places of sensors, the quality can be estimated from the coverage by applying a suitable measure, as the volume of the regions in total, or a counting measure for objects.

Such an objective quality indicator is varied by changing the sensor parameters or the scenery. For

a chosen scene, the quality of a network allows to automatically *optimize* the sensor parameters and locations. A suitable optimization method deduces the next set of parameters of the network based on the quality of the current set of parameters. Section 1.1 describes parameters, challenges, and aims of sensor network optimization. Section 1.2 specifies these general terms in the context of the above motivation for camera networks. The related work is described in Section 1.3 and in Section 1.4 the content of the thesis is summarized.

1.1 Camera Network Optimization

Sensor networks are used in museums, in the industry, in the military, for natural sciences such as meteorology, archeology and robotics, in architecture, in sports, in virtual reality, and the list of applications goes on. In the industry, contact sensors help to establish the contact of workpiece and tool, but in the military, any kind of contact may be undesirable. In sports, several cameras are used to document the scene from several angles of the playing field. Conversely, in “eye in hand” applications in robotics, a single camera is mounted on a robot arm to get the next best view, so several cameras may not be preferable. Various types and aims of sensor networks exist but only few are designed equally. Thus, the design space of an according optimization is varying. In this section, the similarities for sensor network optimization are depicted in the context of the parameters of sensor optimization in general (Section 1.1.1), in the context of challenges of camera network optimization (Section 1.1.2), and in the context of aims of sensor network optimization (Section 1.1.3).

1.1.1 Parameters

The design space of wireless sensor networks is the combination of various factors described by [4, 128]: The costs or resources, the ease of deployment, heterogeneity of sensors, mobility, communication modality, network topology and transmission media, coverage, infrastructure, connectivity, network size, and life time. The authors of [143] add local processing, real-time performance, time synchronization, and data storage in their survey about visual sensor networks. However, the significant parameters for sensor network *optimization* can be summarized as follows:

Variables The variables of sensor network optimization are the parameters of the sensors that have an impact on the coverage in any kind. This includes the *number and model/type* of sensors in a network. For example when using a contact sensor, the coverage is basically the boundary of the sensor hardware. In contrast to this, a vision sensor covers the space between the sensor and the next wall. The variables also include the decision whether it is a passive sensor, like infrared, or an active sensor, such as radar. For active and passive vision sensors compare [1].

The *intrinsic parameters* of a sensor influence the coverage but not the outer hardware of the sensor, like a sensor’s range, image types, opening angle, and distortion. The *extrinsic parameters* of a sensor are parameters that influence the hardware as well, including position and orientation. In order to enlarge the range of the sensor network, the network can be distributed. In distributed network design, the sensors are grouped together and communicate with a group specific processor which further communicates with the nodes of other groups. Compare [119] for distributed computer vision algorithms.

The extrinsic and intrinsic parameters, the number and types of sensors, the positions and partitions of other hardware are used as variables for the optimization. In *sensor placement*, only a set of extrinsic parameters is optimized.

Input data The input data includes all relevant parameters that are not optimized, including the sensor model. Additionally, the environment plays an important role to hardware and software: In indoor environments, one is usually free to use whatever ceiling, walls, and floor he/she likes, and install artificial light or radiation if not sufficiently illuminated. Outdoor environments are more unpredictable due to weather changes like wind, clouds, rain, etc. In an optimization, the environment is usually fully or partially given as a CAD model. Exceptions exist: Networks which explore the environment build up a CAD model from the sensor images to find a collision free way for the agents with mounted sensors. Instead of a CAD model an actual part of the real world is provided for the optimization of these sensor networks.

When defining the environment, the *targets* of the sensor network, which are the objects that need to be covered, should be specified as well. *Obstacles*, defined as the remaining objects, influence the coverage of a sensor or the placement. Furthermore, the important regions that need to be covered by the sensors need to be specified, i.e. the surveillance area. Additional data which needs to be given in a sensor network optimization includes the possible sensor or hardware locations and other boundary conditions to hardware and software.

Type of domain The domain of the variables determines the type of optimization: For example, the parameters of the sensors can be chosen from a finite set, such as particular mounting spots for sensor places. In contrast to this, they could be varied in a continuous domain, e.g., when placing them alongside a complete ceiling. The optimization of these domains falls within the scope of *combinatorial* or *non-linear optimization*, cf. [106, 138] for an introduction to these subjects.

Additionally, sensor parameters can be optimized and fixed before starting the task of the network or, in contrast to this, dynamic sensors can be used whose parameters are adapted while covering the scene. The optimization of dynamic parameters falls within the scope of *optimal control*.

Constraints Constraints to the variables are either given by the environment, or by additional information. For example, there are places where no hardware, like sensors, processors, cables, etc., can be placed, such as the inside of a wall. Furthermore, the positions of wired sensors are much more restricted than the positions of wireless sensors. An example for additional information not given by the environment is the type of communication between sensors of a sensor network.

Objective function The objective function of the optimization states the quality of the network with respect to the task of the network. A network's *task* could be the search for, recognition, tracking, or reconstruction of objects, the exploration or modeling of areas, or the path planning and formation of robots, see the survey [24] for further examples. But faces could be recognized falsely, car tracks can be lost, or the reconstruction can overestimated. For distributed network, the communication plays an important role for the quality of a network. One could desire to decrease latency, reaction time, message loss, or to ease the deployment of sensors, cf. [87].

In general, however, research in sensor network optimization, and camera placement in particular, can be grouped roughly into three sections: The first aim is to *maximize the coverage* for a given

number of sensors and a given model of the environment, already mentioned in 1974 as Maximal Covering Location Problem [26]. The second goal is to *minimize the costs* of a network with respect to covering given regions of the surveillance area. In literature, this is commonly reduced to minimizing the number of sensors, already mentioned in 1987 as Art Gallery Problem [113]. The third goal is to *minimize the error* that may occur when fulfilling the task, e.g. the error of reconstruction or tracking. Note that only covered regions and objects can be reconstructed or tracked.

In the above example in Figure 1.1, an objective function can be the maximization of the distance between the robot and the human silhouette, as in [64], the minimization the error made in the reconstruction of the silhouette [163], or the minimization of the number of cameras. In all three cases, covering the important regions of a human's working area is essential.

Optimization method Optimizing such an objective function can either be done by a fast *heuristic* method or a provably convergent *solver*. In the first case, prior knowledge is used to adjust the parameters of the sensors and an experts needs to approve of their quality. The second method is an iteration in which the next set of parameters is automatically deduced from the quality of recent network parameters. The type of solver depends on the type of domain, as already mentioned above.

Clearly, the coverage of the sensor network plays an important role to all three types of objective function. The solver of such a problem needs a programmed version of the coverage of the sensor network in order to automate the placement, e.g., by simulation. This is where the following challenges arise.

1.1.2 Challenges

The practical implementation of sensor optimization is accompanied by a considerable amount of costs and problems. E.g., in the recent publication [97], the author uses simulated annealing as an optimization method to place cameras in a 2D environment and states that for “very high dimensional spaces (> 8), although the algorithm provided reasonably good solutions very quickly, it sometimes took several hours to jump to a better solution.” In order to get a feeling for the size of a problem in camera placement: Placing and orienting one camera in 2D requires three variables (x -, y - position, one orientation angle). Therefore, eight variables correspond to less than three cameras unless the orientation is not optimized. But what are the specific challenges of camera placement? According to [107], the following computational problems of the coverage can be encountered:

- Firstly, the coverage of one camera, let alone a network of cameras, can only be derived geometrically, meaning the visibility of every point, every object, or path needs to be checked for each camera. The simulation takes time and computational **costs**, e.g., for inverse ray tracing, cf. Section 2.2.
- Secondly, the coverages of several cameras sometimes overlap and their fusion resembles an **intersection or union of polyhedra**. Set operations on polyhedra are known to be a non-robust computation, i.e. when two polygons are tangentially contacted or are only intersected on one of their boundary edges, numerical errors can lead to topological inconsistencies, see [157]. In

order to cope with set operations, the most commonly used data structure for the coverage is the following: The surveillance area is discretized into an orthogonal grid, composed of small cubes of the room, called *voxels*. The coverage of one camera is then a collection of the voxels that are covered. This is also called **occupancy grid**.

The volume of the coverage can be derived by adding up the volume of the covered voxels. This results in a **quantized** objective function. Another frequently used quantized measure in camera optimization is the counting of covered objects or paths.

So the most important calculation in camera placement, the deduction of the coverage, is costly and either non-robust, in case of a polyhedral set operation, or quantized, in case of the occupancy grid or a finite number of objects. These are challenging properties of the coverage of a camera network. Measuring the quality of the coverage has additional challenges for camera network optimization, which are the following:

- The volume of the two-dimensional coverage of a visual sensor regarding its position is proved to be **non-convex, non-linear and only piecewise differentiable**, cf. [58, 97]. When allowing the camera to be placed at an edge or corner of the room, the measure can also be discontinuous.
- However, given the geometrical nature of the problem, only few approaches exist that provide an analytic formula of the measure of the coverage. The work [58] is an exception and provides the volume of the (unlimited) field of view of a visual sensor in 2D, but not in 3D, not with limited field of view, and for only a single camera. Useful properties, such as the convexity or differentiability of a measure of the coverage, are hardly analysed in literature. In the context of optimization, a function failing to provide specific details, such as a gradient, is henceforth called **black-box** function.
- Last, when utilizing a quantized measure, its range only consists of a finite number of function values, i.e. it is piecewise constant on the domain. This is henceforth called a **stair-cased function**.

Thus, in addition to costly function evaluations, the quality of a camera network may have one or more of the properties stair-casing, black-boxing, non-convexity, non-linearity, and piecewise differentiability, which need to be considered in an optimization.

To guarantee global optimality of camera placement, all local optima need to be found and checked. Two strategies can be pursued to find these local optima. A convex function on a convex domain has only local optima that are also global optima. So, the first strategy is identifying all convex parts of the function and then use a **local solver** to find the local optimum, e.g., by the sequential quadratic program. To increase the convergence rate, local solvers usually regard the gradient of the function. The second strategy disregards gradients and convexity and simply searches the domain almost everywhere. It is based on the fact that an arbitrary, continuous function needs to be sampled densely in the domain to find the global optimum, c.f. [153]. This is a fact that **stochastic solvers** take advantage of. For optimization, the piecewise constancy or differentiability, black-boxing, non-linearity, and expensiveness are challenging. The consequences are listed below.

- The word “dense” already hints that the convergence speed of local solvers is higher. The costs of the objective function are **multiplied by the number of objective function calls**. So normally, with a costly function such as ours, the choice would fall to a local solver.
- Unfortunately, a **gradient** of the objective cannot be derived when optimizing a black-box function. The numerical approximation of the gradient by the difference quotient is an alternative, but in an n -dimensional domain it needs $n + 1$ function evaluations. This multiplies the costs of the deduction of the camera coverage. In any case, the numerical gradient only converges to the real gradient of a function if this function is differentiable at this point, but the function at hand may be non-differentiable or may even be stair-cased.
- Without being convex, an arbitrary objective function can have **several local optima**. Usually, one would analyze the objective function for the parts of the domain on which the function is convex. But with a black-box function, such an analysis is difficult, as has already been stated, and the number of local optima cannot be identified.

Without gradient and convexity, the application of a local solver is hardly possible, but can we establish efficient camera network optimization that converges to the global optimum, nevertheless? This is the basic question we want to address in this work.

1.1.3 Aims

The aim of sensor network optimization is to adjust the intrinsic and extrinsic parameters of multiple sensors in order to increase the quality of the sensor network.

Problem 1.1.1 Let \mathbb{P} denote the multi-dimensional *parameter space of a sensor*. This set is kept abstract intentionally, but it may define the position and orientation of sensors, the intrinsic parameters of sensors, or similar properties. Then the *quality* q of a sensor network consisting of $N \in \mathbb{N}$ sensors is defined as a function $q : \mathbb{P}_1 \times \dots \times \mathbb{P}_N \rightarrow \mathbb{R}$. For the following problem a suitable solver needs to be found.

$$\text{Find: } \underset{x \in \mathbb{P}_1 \times \dots \times \mathbb{P}_N}{\operatorname{argmax}} q(x) \quad (1.1)$$

A suitable solver fulfills the following requirements:

Global convergence The objective function can have several local and global optima. Here, the global optimum is searched for. The aim is to establish a convergence of the solver to the optimum despite black-box, stair-cased objective functions.

Incorporation of prior information The Problem (1.1) suggests that some of the properties of the objective function can be deduced. For example, the positions of two equally built sensors can be switched without changing the objective value. A solver of Problem (1.1) needs to be ready to integrate such prior information.

Distribution A sensor network that can be split into groups of sensors and nodes is called *distributed network*, [142]. A group is called *self-organized* if a computing node within the group exists where the parameters of the group’s sensors are controlled, as done in [145]. As an advantage for distributed sensor networks, the optimization of self-organized groups can be computed in parallel on the groups’ computing nodes. However, the parallel optimization is not necessarily convergent to a local or global optimum. In this thesis, a solver needs to be found that can be computed in parallel but nevertheless converges to the optimum.

Anytime system In case of an algorithm which physically moves the sensors, collisions with other sensors, walls, or obstacles need to be prohibited, i.e. by defining a mounting area for the sensors. However, an optimization strategy may not find such positions in each iteration step. An *anytime system* is a system that returns a valid solution even if it is interrupted at any point in time between a short initialization phase and its termination. The solutions are iteratively improved with time. A valid solution is resembled by parameters of the sensor network that are feasible for sensor network optimization, e.g., the positions are chosen from the desired mounting area. A solver designed in this thesis should have this quality.

Efficiency The less time a solver needs, the more efficient it is. In order to reduce the time of sensor optimization, the costs of the coverage calculation and the total number of times in which the coverage is calculated need to be reduced.

After having introduced the parameters, aims, and challenges of sensor network optimization, the following section discusses the specific problem, which is the heart of the thesis.

1.2 Problem Definition

The quality of the sensor network can be defined by the costs of the sensor network, by the error that is made by the task, or simply by the maximization of the coverage. Common to all types of objective functions is the notion of coverage, as discussed in Section 1.1.1. Therefore, we will define the coverage independently on the type of sensors, on the type of optimization, and on the type of task.

The following sets are kept abstract, again intentionally. Let E denote the *environment* where the sensor network is used. The environment includes constraints to the network such as the geometrical arrangement of objects, cables, textures, etc. in a scene. The environment E is in the set of all possible scenes denoted by \mathbb{E} . Let \mathbb{A} be the *set of surveillance parts*, which are the items that are to be observed. This can include the points, objects, paths, etc. that are not to be missed. The parts under surveillance can be marked by different labels, for example, if the surveillance parts are voxels, one usually at least wants to distinguish between “detectable” and “undetectable” voxels. Other examples are the labels “changed” and “identical” in a change detection system. Let \mathbb{S} be the *set of sensor labels* that surveillance parts can be marked by.

Definition 1.2.1

Let \mathbb{P} denote the parameter space of a sensor and let $\mathbb{A}, \mathbb{E}, \mathbb{S}$ be as above.

1. A *sensor* σ is considered a function which maps a surveillance part $y \in \mathbb{A}$ with the parameters of $a \in \mathbb{P}$ and the constraints of the environment $E \in \mathbb{E}$ onto the set of sensor labels:

$$\begin{aligned} \sigma : \mathbb{E} \times \mathbb{P} \times \mathbb{A} &\rightarrow \mathbb{S} \\ (E, a, y) &\mapsto \sigma_{(E,a)}(y) \quad \text{or } \sigma_a(y) \text{ if } E \text{ is fixed} \end{aligned}$$

2. Let $E \in \mathbb{E}$ be a fixed environment and let $S \subset \mathbb{S}$ be a set of sensor labels, then the preimage $\sigma_a^{-1}(S)$ is called the *coverage* of the sensor with parameters $a \in \mathbb{P}$ and of the sensor labels in S .
3. Let us have $N \in \mathbb{N}$ sensors denoted by their various parameter vectors $a_1 \in \mathbb{P}_1, \dots, a_N \in \mathbb{P}_N$, then we call $x := (a_1, \dots, a_N)$ the *variables of the sensor network optimization* or simply the *variable vector*.
4. Choose a common set of sensor labels \mathbb{S} for all sensors. The sensors with the parameter vectors $a_1 \in \mathbb{P}_1, \dots, a_N \in \mathbb{P}_N$ and variable vector $x = (a_1, \dots, a_N)$ have a *fused coverage*:

$$C_x(S) := \begin{cases} \bigcap_{n=1, \dots, N} \sigma_{a_n}^{-1}(S) & \text{if a part is meant to be covered by all sensors, and} \\ \bigcup_{n=1, \dots, N} \sigma_{a_n}^{-1}(S) & \text{if a part is meant to be covered by at least one.} \end{cases}$$

There actually exist shades of the fused coverages “covered by all” and “covered by at least one sensor”. Additionally, either fused coverage can be transferred into the other. The modification including these shades is addressed in Section 2.1.4.

The sets \mathbb{A} , \mathbb{P} , and \mathbb{S} have been kept abstract in order to be able to adapt the coverage to various applications for sensor network optimization. To concretize the abstractly defined sets, consider the following examples. The first example states the parameter space of camera placement, the second one the set of surveillance parts. The third example depicts two types of sensor label sets, one for distinguishing detectable and undetectable regions of the environment, and the other one for distinguishing changed and identical regions.

Example 1.2.2

Assuming that the cameras are to be placed in a 3D scenery, let $\mathbb{L} \subset E \subset \mathbb{R}^3$ be the area of possible sensor locations. Then the position of a camera is in \mathbb{L} and its orientation can be denoted by (ψ, ϕ, ρ) with the yaw $\psi \in [-\pi, \pi]$, pitch $\phi \in [-\pi/2, \pi/2]$, and roll $\rho \in [-\pi, \pi]$. The notation is partly derived from flight navigation. Furthermore, let us assume, that all cameras share the same opening angle in direction of yaw and pitch of the cameras’ orientation, in order to simplify the notation. Now, we can specify the sets $\mathbb{P}_1 = \dots = \mathbb{P}_N := (\mathbb{L} \times [-\pi, \pi] \times [-\pi/2, \pi/2] \times [-\pi, \pi])$.

Example 1.2.3

In the particular case where the regions of a three-dimensional environment need to be covered, the set of surveillance parts is specified by $\mathbb{A} \subset \mathbb{R}^3$. In such a continuous case, the set of surveillance parts is then called *surveillance area*. Another example for a surveillance area are the regions of a simplified two-dimensional environment, such as the top view of a room. This stands in contrast to a quantized set of surveillance parts, such as the objects or paths in an environment.

Example 1.2.4

The set of sensor labels depicts what a sensor can and cannot observe. Cameras cannot see behind

walls, so the region behind a wall is undetectable and the region in front of the wall is detectable as long as no second wall is in front of it. Thus, in a camera network, the regions can be labeled by $\mathbb{S} := \{\text{detectable}, \text{undetectable}\}$. Let us assume that we have utilized a sensor which can distinguish between a target that has changed since we have had a look at it, last time, and a target that has not changed. Then, the set of sensor labels needs additional labels: $\mathbb{S} := \{\text{undetectable}, \text{changed}, \text{identical}\}$.

The coverage of a sensor network defined in this thesis can be measured using the volume λ of a solid. Depending on the labels in S the volume of the covered surveillance area is denoted by $\lambda(C_x(S))$. This measure can be used in an optimization in different ways: E.g., when maximizing the regions of the surveillance area that are detectable, the coverage is part of the objective function. When minimizing the costs with respect to keeping important regions covered, it is part of the constraints.

In this thesis, the coverage is used to maximize detectable regions, and to minimize the error of a reconstruction as in Figure 1.1: Therefore, let $x \in \mathbb{P}_1 \times \dots \times \mathbb{P}_N$ be the extrinsic parameters of the cameras in a network, i.e. the positions and orientations of cameras. The problem (1.1) is solved with the quality function

$$q(x) = \lambda(C_x(S)) \quad (1.2)$$

where $S \in \mathbb{S}$ is adaptable to the task.

In this thesis, the discussed sensors are cameras. The parameters of the cameras which are used as variables in the optimization are the position and orientation of the cameras.

1.3 Related Work

This section presents publications related to the methods developed in this thesis. The motivation for the survey is camera placement which is summarized in Section 1.3.1 for reconstruction purposes and in general. The objective function needs to be examined further. It is closely related to the field of visibility and computer graphics, the publications of which are covered in Section 1.3.2. The methods that solve a camera placement problem are optimization methods related to the fields in Section 1.3.3.

1.3.1 Camera Placement

In robotics alone, over 2000 research papers about vision have been published between 1986 and 2010, [24]. Thus, camera and general sensor planning is a vastly covered research area for multiple tasks. Heterogenous sensors have been utilized for example for tracking and detection [81, 127]. There have been surveys and classifications on heterogenous sensors and other hardware components, cf. [35, 145], about the communication [87], e.g., wireless communication [3, 23], and the environment, e.g., wide area surveillance [1].

Task specific camera placement

The optimization of cameras for a specific task is also vastly researched: The authors of [24] find that camera planning has been done for tasks such as inspection [149] and surveillance [141], grasping [99], tracking [8, 34], exploration and site modeling [120], object modeling [7] and reconstruction [104], recognition [33], path planning [9, 10] and multirobot formation [74]. The best view regarding radiation and illumination is depicted in [88]. Some publications regard dynamic sensors, that are adjusted while performing the task. This is called *visual servoing* [89]. For example, an eye-in-hand system [2, 7, 9, 10, 99] is a camera mounted on a robot arm while the robot moves. More than one sensor is, for example, adjusted on the fly in [8, 74], and in a distributed network in [142].

Following this thesis' motivation, one task specific goal is the *minimization of an error made when reconstructing* the position and measurements of the target's corners, curves, surfaces, objects, etc. The phrase Photogrammetric Network Design is often used for analysing the position of cameras by minimizing the reconstruction error of several (three-dimensional) points, for details cf. [67, 91, 108–111]. This is further developed by [132, 133] for applications in unknown environments (a CAD model is missing as well). The actual placement of several cameras to optimally localize an entire object which is not occluded is an assignment treated in [47]. One common simplification in this area is to reduce the domain of the camera's position and orientation, e.g., by the viewing sphere model given in [110] or the idea of situating all cameras on a plane and aiming them horizontally, cf. [47].

In this thesis, cameras are placed in a network to determine a visual hull of an entire object. In order to get the minimal error of the hull, [163] assumes that minimizing the occurring occlusions of solids also reduces and thus specifies their possible locations. His simplifications: The orientation of the camera is neglected as a variable since the camera is orientated towards the object, and obstacles are not regarded. Often, the challenge of minimizing the reconstruction error lies in obscuring obstacles which are placed in the environment, cf. [45, 46] and [63, 64]. These publications use a *background subtraction method* to identify obstacles and targets. This method is called *scan-line* method if the subtracted image is further simplified to a one-dimensional image [46]. But the methods regarding obstacles in [45, 46] only select the cameras from a predefined set of cameras, and [63, 64] optimize the distance of the hull to another object. Here, the visual hull is optimized in an efficient manner which allows a continuous domain and a three-dimensional environment.

More general camera placement - Art gallery problem and maximal set covering location problem

The minimal reconstruction error depends on the type of reconstruction and can seldomly be transferred to a different type of reconstruction. More general camera placements are the following:

Apart from minimizing such an error, for some camera networks the only information that is regarded is *whether* a part of the environment has been detected. One common goal in this context is to be able to observe all items of a given set (e.g., all surveillance parts) with a minimized number of cameras [12, 48, 60, 71, 95–97, 161]. This issue is called Art Gallery Problem especially when speaking of two-dimensional space, and has first been mentioned in 1987, cf. [113]. It has already been transferred to the three-dimensional case, cf. [90].

The reverse question is how to position and orientate a given number of cameras in order to maximize the observed parts of the environment. This problem is also known as Maximal Covering Location Problem since 1974, cf. [26]. Such parts can be surfaces, e.g., [70], and paths, as in [17, 50, 51], or the number of objects [95–97]. Distributed wireless sensor networks have been covered as well, e.g., by [151].

The closest publication to this thesis is [101] addressing the Backup Coverage Location Problem, whose surveillance parts of the environment are voxels and who considers overlaps of cameras. The author’s simplification is discretizing the environment, as well as the parameter space of the sensor network, and thus selecting the cameras out of a predefined subset of positions. In this thesis, the quality of the camera network with a given number of cameras is optimized on a continuous rather than a discretized domain.

1.3.2 Coverage and Visibility

In order to incorporate prior information into the optimization, the objective function needs to be analyzed before discretizing the surveillance area into voxel. Such an analysis of coverage and its quality can also be valuable for the maximization of continuous objectives as in [70, 101], for calculating the visual hull used by [64, 163], or when deriving continuous constraints to the art gallery problem as in [12, 48, 161]. Within their approach, these publications use an occupancy grid and disregard such an analysis. This is an indication that an exact visibility analysis as approached in this thesis has not been done, yet. Sutherland proposes to classify the algorithms for occlusion reasoning into list-priority methods, image space methods, and object space methods, [147]. The first two types of methods regard the discretization of the image plane in the output device (the pixel). Hence, a closer look on the object space methods is required.

The first algorithm ever developing the shape of the coverage can be seen in [126]: Every polygon edge is tested against every polyhedron whether it is occluded by solving a linear equation system. It works only for convex polyhedra. The algorithms of [5, 57, 85] test edges against edges. Thereby, the “quantitative invisibility” is an indicator of how many polygons occlude an edge from a given viewpoint. A subsequent algorithm [6] has also been developed surrounding the line segment by a halo, the latter occludes lines further away. More recent surveys of visibility and computer graphics are provided by [15, 31, 32, 40].

Many of the mentioned publications in [40] are not our concern, e.g., publications concerning curved objects, discretized object space, ray-, beam-, and cone tracing, z-buffer methods, radiosity computations etc. The works closest to the approach in this thesis are about clipping all polygons to the nearer polygons successively [158], about shadow regions [27], or umbra/penumbra [105], which are the boundary of a shadow on the wall. The authors of [164] make the latter algorithm less costly by walking along the edges and vertices of the blocker and illuminator simultaneously.

The classification of faces and vertices of the coverage can only be done if the coverage of a sensor has been deduced by an object space method or shadow method. However, none of the discussed publications has characterizing visual events, such as the non-existence of a face or a vertex. These visual events are important in this thesis since they influence the continuity, differentiability, and stair-casing of the volume of the fused coverage. However, in the context of shadows, the term discontinuity is used to express a change of the radiance on a surface. The authors of [68, 69] consider C^1 -shadow discontinuities for a single light source and [37, 38, 144, 150] extend their characterization to C^2 -discontinuities in 3D. Integrating the events into efficiently constructing accurate shades of a room full of objects is known as

discontinuity meshing. The authors of [41, 42] use a graph to store the events and [84] additionally use discretized objects for this purpose.

The continuity and differentiability of the volume of a camera's coverage has not been done until 2006. [58] set an example for an analysis of the volume of the field of view in 2D with the examination of the smoothness of the volume of an omni-directional camera in a polygonal environment. They prove that the volume is almost everywhere locally Lipschitz and they characterize the non-smooth behavior in order to optimize the position of the camera.

Being able to characterize non-smooth behavior in 3D in a similar way as [58] has done in 2D is addressed in this thesis. As will be seen, the coverage of several cameras with limited field of view in 3D is a more extensive case than the field of view of an omni-directional camera in 2D. In the context of shadows, a characterization of the places and orientations of several lights where visual events take place, corresponds to a characterization of network parameters, as is done in the first part of this thesis. However, none has considered places *and* orientations with several light sources that have a limited opening angle which is required for cameras with a limited field of view. Moreover, none has considered the non-differentiability of the error of a human approximation, as is done in this thesis.

1.3.3 Optimization

A non-linear, non-convex black-box function is to be maximized, globally. Strategies to solve optimization in general can be found in [106]. Methods optimizing a non-linear function have been developed for the use with and without gradient, for example the Nelder-Mead-Simplex [18, 102], or the Interior Point Filter Line Search [155]. Unfortunately, these methods have weaknesses when globally optimizing a high dimensional domain, or a black-box function.

Global algorithms are called *accurate methods* or *covering methods* by [153]. These strategies are also developed for non-linear programs without a gradient, e.g., the Differential Variation in addition to a Mutation Rule [130, 131] or the Ant Colony Algorithm [135–137]. However, strategies neglecting the gradient usually call the objective function more often, which can also be seen in the experiments in Section 3.4. We propose a method that calls the objective function less and can be computed in parallel.

Optimization of Response Surface Models

In order to increase the convergence speed of an optimization strategy, local approximations of the objective function have been used for a few decades. As soon as a good solution of the approximation is found, the real objective is evaluated at this point and the approximation is discarded. The Method of Moving Asymptotes and the Sequential Quadratic Program, based on [148] and [29, 77, 134], respectively, use such a local approximation. The approximation is convex and separable in the first case, and quadratic in the second, which makes the search for a good solution easier. However, in order to get such an approximation a gradient is needed. Additionally, the solutions for which the actual objective function has been evaluated are not stored. A *response surface model* or *surrogate* is an approximation of the whole objective function in which solutions can be updated and remembered. A taxonomy on global optimization methods based on response surfaces can be found in [73].

Let us assume that some samples have already been collected from the objective function by previous iteration steps. The methods to interpolate samples can be grouped into global and local methods as well: The local methods only utilize a subset of sample points to approximate a given function in the vicinity of a point $x \in \mathcal{D}$, e.g., B-Splines, Bézier curves, Hermite curves, or Catmull-Rom splines, cf. [52, 82] and citations therein. Thus, the function value of a point is only evaluated by the information of the neighboring samples. On scattered samples, i.e. samples that are not arranged on a grid, the Delaunay triangulation can be used to determine the vicinity of a point, [43]. In this thesis, a radial basis function interpolant from [115, 116] is used as a response surface model. Instead of a local method to interpolate samples, the radial basis function interpolants belong to the global methods which use the information of the whole sample set to approximate a function, such as polynomial interpolation or the natural cubic splines [139]. Additionally, the radial basis function interpolants are smooth functions, which a gradient is easily calculated of, that interpolate scattered samples of an n -dimensional domain without further triangulation. Other piecewise and global, multivariate approximations can be investigated in [21].

Radial basis function already have been studied half a century ago, e.g., where [39] and [55, 65] researched thin plate splines and multiquadric surfaces, respectively. The authors of [94] were the first to prove that multiquadratic surface interpolation is always solvable. Around the turn of the millennium, [116] developed a method to add samples subsequently to the function, like the Newton's subsequent interpolation method, which he already used as a response surface in optimization. A survey of radial basis functions can be found in [20, 21].

The authors of [16, 93, 121, 122], and [62] have developed optimization methods on radial basis functions as response surfaces. However, none has ever combined a surrogate solver with a Block Coordinate Ascent (BCA) as described in the next section in order to parallelize the method. Also none has adopted the symmetrical property of a function to the method. For publications which have combined a BCA and a surrogate solver compare the next section.

The acceleration of symmetrical functions has been studied before: The survey [118] summarizes that the strategies are adding *symmetry breaking* constraints or heuristics, modifying the search technique so that interchangeable values are removed, or discarding subsets of a discrete domain which is symmetrical to another subset. But adding symmetry breaking to a local search technique can also cause worse performance, cf. [117]. We are proposing to change the search technique so that already evaluated samples are added to the response surface model on all symmetrical parts of the domain. Thus, no symmetry breaking constraints need to be added and no solutions need to be discarded.

Block Coordinate Ascent

The parameters of one single camera are a subset of all parameters of the network. This subset is called *block* of parameters or variables. The optimization on such a block is an optimization on an affine subspace of the domain, just as a variant [129] of the Nelder-Mead-Simplex. The alternation between optimizations on different affine subspaces of the domain is generally called *variable decomposition method*, but more definite names for this method exist:

It is called *coordinate search method* when speaking of a one-dimensional subspace of the domain [106]. When speaking of an n -dimensional subspace, the following distinction can be made: The method is either named *block coordinate descent/ascent* if the subspace is parallel to the coordinate axes of one block,

or named *block-nonlinear Gauss-Seidel method* if the search direction is not parallel to the axes [61]. If the number of blocks is two, the method is called *alternating minimization/direction method* [11]. In the field of image analysis, the block coordinate ascent is also considered a *domain decomposition*. Such methods exist for sequential and parallel overlapping subspaces [53] and non-overlapping subspaces [54].

The convergence of these methods has been studied under various assumptions, e.g., [14] develops such methods for strictly convex, quadratic or separable functions. The authors of [13, 61] state the problem with block-coordinate descent/ascent methods: The next subspace does not necessarily lie in direction of the gradient. This can be changed by the gradient rule [86] in the differentiable case, and by the rules from [154] in non-differentiable but certain separable cases. The method in [103], which selects the block on which the gradient projection step is performed by a randomized rule, is developed further and parallelized by [125] for large data.

In this thesis, a block coordinate ascent (BCA) in combination with a radial basis function (RBF) as a response surface model is developed. A connection between BCA and RBF is found in the following publications: The authors of [61] use a block coordinate descent to fit a neuronal network based on a radial basis function to an actual function. However, they have not used the radial basis function of [115] with a polynomial term. Moreover, their aim is to fit the model, not to optimize the actual function.

The publications [100, 123] have combined a surrogate solver with a Block Coordinate Ascent under the assumption that not all the variables are equally relevant. They perturb the best point to generate a new sample point. The authors of [123] use a heuristic to choose the blocks, in which the variables are distorted by a Gaussian distribution. The authors of [100] combine the optimization on a cubic RBF with local search. Their global search method dynamically adjusts the number of decision variables being perturbed in each iteration, for large problems. In this thesis, the blocks that correspond to each camera are equally relevant and usually contain a similar number of variables. The global convergence of the method is shown.

1.4 Overview

Section 1.3.1 makes clear that there is a need for an approach that does not simplify the problem of camera placement, such as restricting the domain to a predefined finite set of camera positions. The continuous domain is to be searched for the best set of parameters for the whole camera network.

The aim of camera placement derived from Section 1.1.3 is to create a flexible algorithm that converges globally in an efficient way and improves the solution, subsequently, as an anytime system. For accelerating the whole iteration, additional demands on the optimization method are stated in the same section: The method has to be ready to be computed in parallel. Furthermore, prior information needs to be added easily.

Section 1.3.3 provides an overview on optimization methods suitable for one aim or the other, but never for all these aims. Furthermore, hardly any prior information is available for the volume of the reconstruction error or the coverage, cf. Section 1.3.2. These facts are the main motivation for the following contributions to camera network optimization:

- Establishing prior information about the geometry of the fused coverage in Sections 2.1 and 2.3. The detectable regions of the surveillance area, as well as the error of the visual hull reconstructed from a human target, as in Figure 1.1, can be expressed in terms of the fused coverage.
- Investigating an efficient way of computing the volume of the fused coverage as an objective function in Section 2.2;
- Establishing prior information about the volume of the fused coverage in Section 2.4;
- Incorporating prior information into an efficient, global solver for a stair-cased, black-box function in Chapter 3.1;
- Developing an efficient, global solver for a function that is less expensive when calling it on sub-spaces of the domain, and which is distributable, for optimizing a stair-cased, black-box function in Chapter 3.2;
- Demonstrating the efficiency of the proposed approaches on several synthetic functions in Sections 3.3 and 3.4 and on two practical examples in Chapter 4.

In the end, in Section 5.1, the thesis is summarized. The achievement of the aims and the contributions are addressed in Section 5.2 and in Section 5.3 open questions are presented.

Chapter 2

Deduction and Properties of an Objective Function for Camera Placement

The aim of the camera network of Figure 1.1 is to approximate the human collaborator in a scene to protect him or her. The camera network needs to be optimized in order to decrease the error of the approximation. In general, the wider the important regions of the room are covered by the cameras of the network, the more exact an approximation can be formed. But in order to improve the coverage or the approximation, an objective function needs to be specified, that measures the quality of the camera network. Moreover, its properties need to be investigated, such as continuity, differentiability, convexity, symmetry, and stair-casing. These properties will be used in Chapter 3 to develop a suitable optimization solver tailored to the objective function.

In this chapter, the fused coverage C of Definition 1.2.1 will play the key role since it can be modified to resemble both, the field of view of several cameras and the approximation error (Section 2.1). The optimization of the camera network usually calls the objective for various sensor network parameters $x \in \mathbb{P}_1 \times \dots \times \mathbb{P}_N$ until an optimum is found. The objective function chosen in this thesis is the volume of the fused coverage $q(x) = \lambda(C_x)$ from Equation (1.2). Its construction is illustrated in Figure 2.1.

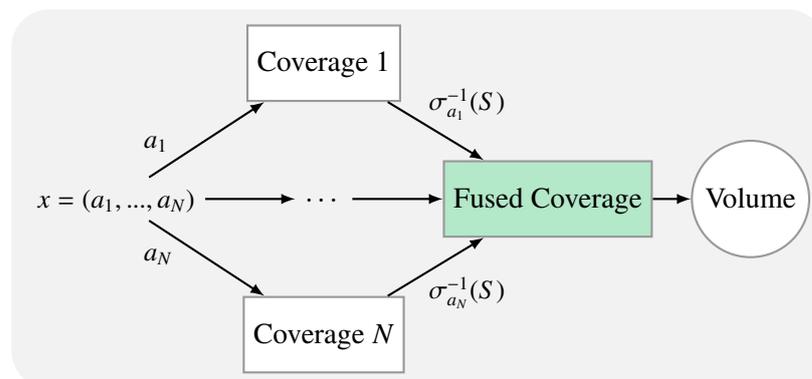


Figure 2.1: Illustration of a single objective function call $\lambda(C_x(S))$: Depending on the choice of the sensor labels $S \subset \mathbb{S}$, the coverage of each camera $\sigma_{a_n}^{-1}(S), n = 1, \dots, N$ needs to be simulated and fused (intersected or united). The fused coverage is measured by the volume.

First, we develop an accelerated simulation of the fused coverage. A possible implementation of $\sigma_{a_n}^{-1}(S)$ discretizes the environment into *voxels* and the image space of the cameras into *pixels* (Section 2.2.1). In this chapter, we discuss three acceleration methods:

- In Section 2.2.2, the computation of the coverage $\sigma_{a_n}^{-1}(S)$ of one camera is accelerated by suitable data structures and parallel computing.
- Subsequent objective function calls $\lambda(C_x(S))$ turn out to be less expensive for partly constant camera positions and orientations, in Section 2.2.3.
- Interchanging two cameras does neither alter the coverage nor the volume of the coverage of the camera network. This property is called *symmetry* and is stated in Section 2.4.3. A solver can use this property when caching recent function calls.

Second, in order to choose a suitable solver for such an objective function, this chapter investigates the mathematical properties of the quality $q(x)$ of the sensor network in a three-dimensional environment. The course of this investigation starts off disregarding the discretization into voxels and pixels. The contribution of this thesis to the field of research can be read in the context of a summary of the whole thesis in Section 5.2 and is the following:

- In Section 2.3.1, we will prove that the shape of the fused coverage is a polyhedral area, a polyhedron that can be disconnected and may be flat at some points. Its faces and vertices are necessary to analytically calculate the volume of the coverage (Section 2.3.2).
- We classify the vertices and faces of the fused coverage in Section 2.3.3 with respect to cameras' position and orientation as variables.
- The set of sensor network parameters where a face of the fused coverage meets a chosen point of the environment is classified as *incidence surface*. (Section 2.3.4) In contrast to a two-dimensional environment discussed in [58], the set is not a linear segment but a more general, non-planar surface in the network's parameter space.

There are two reasons for investigating these incidence surfaces with respect to the network parameters: The events where a face meets a vertex of the fused coverage are relevant for the deduction of continuity and differentiability of the coverage's volume (Section 2.4.1). The second reason is the *stair-casing* effect of the volume: The incidence surfaces of voxels partition the domain of the function into sets on which the objective function is piecewise constant (Section 2.4.4).

In order to formulate the objective function $q(x)$ and derive polyhedral and mathematical properties, we will need the notation of the set operations in Appendix B. It defines the symbols for a segment, ray, pyramid, open ϵ -ball, ϵ -sphere, and the closure, boundary, exterior, distance, and volume of a set.

2.1 Coverage of Multiple Cameras

An overview on how to formally define and refine the fused coverage C of Definition 1.2.1 is given in this section. By the choice of a suitable sensor label set $S \subset \mathbb{S}$, the fused coverage can be modified to

resemble both the field of view of several cameras and the approximation error of a human silhouette. The volume of this coverage $\lambda(C_x)$ is used as an objective function in Chapter 3.

In the course of this section, two examples for the fused coverage are given: The fused coverage is first used to resemble the field of view of several cameras (Section 2.1.1). Hereby, the variables of the objective function are also specified. In the second example, we modify the fused coverage to resemble a human approximation using only the sensor data of cameras, in Section 2.1.2. The section introduces a method to detect the human in the image space of a camera. This method can be modified to approximate the three-dimensional human in the environment instead of the image spaces of the cameras (Section 2.1.3).

After giving the examples, a general refinement of the fused coverage is discussed: In Definition 1.2.1, the fused coverage has been introduced as a region which is covered “by all cameras” or “by at least one camera”. On the one hand, the images of a single camera could be inaccurate, e.g., if the target is similar to the background, on the other hand, the term “covered by all cameras” allows the coverage to be failure safe, but is very restrictive. In Section 2.1.4, we incorporate shades of “covered by all cameras” and “covered by at least one” when introducing a reliability threshold $k \in \mathbb{N}$ for the number of cameras to the notion of the fused coverage. The refinement of the fused coverage does not only hold for visual sensors such as cameras, but also for non-visual sensors.

2.1.1 Detectable Coverage

In this section, the fused coverage exemplarily resembles the combined field of view of several cameras. The fused coverage is defined in Definition 1.2.1 as the union or intersection of the camera coverages. The coverage of a single camera is a preimage of the sensor map $\sigma : \mathbb{E} \times \mathbb{P} \times \mathbb{A} \rightarrow \mathbb{S}$. First, each of the following sets is specified: the set of the environments \mathbb{E} , the set of sensor network parameters \mathbb{P} that are utilized as variables in an optimization, the set of surveillance parts \mathbb{A} which contains the points that need to be monitored, and the set of sensor labels \mathbb{S} . In the end, we show how to assign a sensor label to each point in the surveillance area.

Environment and Parameter Space of a Single Camera

An environment $E \in \mathbb{E}$ with $\mathbb{E} \subset \mathcal{P}(\mathbb{R}^3)$ is the space where the camera will be placed, it contains the parts that need to be monitored and models visible restrictions like walls and furniture. In this thesis, the environment resembles the *closed empty space* of a room, which is not occupied by furniture or walls but only by transparent, non-solid matter. The model may seem contra-intuitive, however, there are two reasons for it: Firstly, a camera can just be placed in an empty space of a room. Secondly, the boundary between empty space and non-empty space of a room, i.e. informally the surfaces of furniture and walls, defines whether a point in the environment is visible from camera point of view.

At first, the empty space is modeled as a polyhedral area. Polyhedral Terrain Models (PTM) are often used to represent a terrain or topographic surface in a visibility analysis, [31, 101]. In [44] an informal definition of a polyhedron is given: A *convex polyhedron* is an intersection of finitely many closed half spaces. A *polyhedron* is the union of finitely many convex polyhedra. In this thesis, such a polyhedron is considered in the context of vertices, edges, and faces:

Definition 2.1.1

Let $n, m \in \mathbb{N}$ be integers with $n \geq m \geq 1$. Let $H_{(q,p)} \subset \mathbb{R}^n, q = 1, \dots, Q_1, p = 1, \dots, Q_2$ be half-spaces.

1. A finite concatenation $\mathcal{P} := \bigcup_{p=1, \dots, Q_1} \bigcap_{q=1, \dots, Q_2} H_{(q,p)}$ is called

(a) *mD-polyhedral area* in nD if an m -dimensional affine subspace $A \subset \mathbb{R}^n$ exists with $\mathcal{P} \subset A$.

(b) *mD-polyhedron* in nD if \mathcal{P} is an mD -polyhedral area and an m -manifold with boundary $\partial\mathcal{P}$.

2. Let $\mathcal{F}_i \subset \partial\mathcal{P}, i \in \{1, \dots, I\}, I \in \mathbb{N}$ be $\begin{cases} \text{points} & \text{in case } m = 1 \\ (m-1)\text{D-polyhedra} & \text{in case } m > 1 \end{cases}$

If $\partial\mathcal{P} = \bigcup_{i \in \{1, \dots, I\}} \mathcal{F}_i$ and if for all the pairs $(\mathcal{F}_i, \mathcal{F}_j), i, j = 1, \dots, I$ the intersection $\mathcal{F}_i \cap \mathcal{F}_j \subset A$ is maximally a $(m-2)D$ polyhedron then the \mathcal{F}_i are called *faces* of the mD -polyhedral area.

3. A 2D-polyhedron is called *polygon*, its faces are called *edges*. The face of an edge is called *vertex*.

4. The set of all 3D-polyhedral areas in 3D is defined as \mathbb{E} .

A polyhedron is a *m-manifold*, which means each point has a neighborhood that is homeomorphic to the Euclidean space of dimension m or a half-space of dimension m , again this means that it is nowhere flat or pinched [44] and neither is one of its faces. In contrast to the polyhedron, an mD -polyhedral area is not necessarily a manifold. A polyhedral area is not necessarily convex, it can have holes of dimension m and it can be distributed to several regions. The “ mD ” stands for its dimension in the space \mathbb{R}^n , which will only be needed for polyhedra with a lower dimension $m \leq n$, e.g., a polygon in a 3D space. The environment in which the cameras of the network are placed is 3D-polyhedral:

Notation 2.1.2

In this thesis, the *environment* E where the camera network is placed constitutes a 3D-polyhedral area: $E \in \mathbb{E}$.

In other words, an environment is a not necessarily convex polyhedron potentially containing three-dimensional holes, in fact not necessarily being connected, at all. Be aware that the boundary of the environment does not contain two-dimensional holes, it is a two-dimensional manifold without boundary. The cameras are placed in such an environment.

Definition 2.1.3

Let $E \in \mathbb{E}$ be an environment and let $p \in E$. Let $o, u \in \partial\mathbb{B}_1^3(0)$ be orthonormal unit vectors. The *camera parameters* in the environment E are defined as a 5-tupel $(p, o, u, \theta_u, \theta_{o \times u}) \in (E \times \partial\mathbb{B}_1^3(0) \times \partial\mathbb{B}_1^3(0) \times [0, \pi] \times [0, \pi])$ consisting of the following:

- The point p is called *camera position*, sometimes also *viewpoint*.
- The vector o is called *camera orientation*.
- The vector u is orthonormal to o . If u is a linear combination of o and $e_z = (0 \ 0 \ 1)^T$ it is called *view-up vector*.

- The scalars $\theta_u, \theta_{o \times u} \in [0, \pi]$ are called the *opening angles* in direction o with *opening vector* u and the vector orthogonal to o and u which is $(o \times u)$.
- The tuple $(\lambda_o, \lambda_u, \lambda_{o \times u})$ with $\lambda_o, \lambda_u, \lambda_{o \times u} \in \mathbb{R}$ is called *camera coordinates* of the point $x = p + \lambda_o o + \lambda_u u + \lambda_{o \times u} (o \times u)$.

Figure 2.2 (left) illustrates the camera coordinate system with the parameter vectors p, u and o of the camera. In this thesis, the position $p \in E$ and orientation $o, u \in \partial \mathbb{B}_1^3(0)$ of each camera are considered as variables to the objective function in an optimization. The opening angles $\theta_{o \times u} \in [0, \pi]$ and $\theta_u \in [0, \pi]$ are fixed. In case of u being the view-up vector, u points to the ceiling and can be derived by $u = \frac{1}{o_1 + o_2} (o_1 o_3 \quad o_2 o_3 \quad (o_1 + o_2)^2)^T$ with $(o_1 + o_2) \neq 0$ from the orientation $o = (o_1, o_2, o_3)^T$. Otherwise, the rotation between the view-up vector and u defines the rotation of the image plane of the camera about the orientation and can be determined by a single angle. The rotation angle of the image plane is called *roll*. In our investigations, however, we assume the roll to be fixed at 0 in favor of a clearer notation.

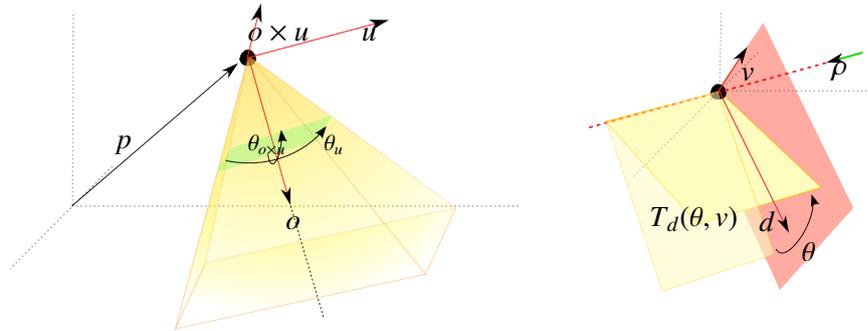


Figure 2.2: Left: Illustration of parameters of a camera in 3D: Position p , orientation o , and opening angles $\theta_u, \theta_{o \times u}$ for the opening vector u and $o \times u$; The camera frustum is an intersection of θ -spaces. Right: $T_d(\theta, v)$ is the θ -space with angle $\theta < \pi$, direction d , and opening v .

Thus, the investigated space of parameters of a single camera that serve as variables for an objective function is

$$\mathbb{P} \subset E \times \partial \mathbb{B}_1^3(0). \quad (2.1)$$

The camera parameters are now used to define the visibility as already promised in the last paragraph. Thereby, we use the term θ -space as a halfspace with an angle smaller than π illustrated in Figure 2.2 (right), to define the camera frustum in 3D.

Definition 2.1.4

Let $E \in \mathbb{E}$ be an environment. Let $a = (p, o) \in \mathbb{P}$ be camera parameters as in Inclusion (2.1).

1. A point $y \in \mathbb{R}^3$ is called *visible in E* from point $p \in E$ if the *line of sight* $[p, y]$ holds $[p, y] \subset E$.
2. $\mathbb{V}(p, E) := \{y \in E \mid y \text{ visible in } E \text{ from } p\} \subset E$ is called the *set of all visible points* from $p \in E$.

3. Let $\theta \in [0, \pi]$ be an angle. Furthermore, let $d, v \in \partial\mathbb{B}_1(0)$ be orthogonal to each other. And let $\rho_{(d,v)}$ be the projection onto the v - d -hyperplane. Then a θ -space is defined as

$$T_d(\theta, v) := \{x \in \mathbb{R}^3 \mid |\angle(\rho_{(d,v)}(x), d)| \leq \frac{\theta}{2}\}$$

d is called the *direction* and v the *opening* of the θ -space.

4. Let $\theta_u, \theta_{o \times u} \in [0, \pi]$ opening angles of the camera. The *frustum* of the camera with parameters a is the set

$$\mathbb{F}_a := \{y \in \mathbb{R}^3 \mid (y - p) \in (T_o(\theta_u, u) \cap T_o(\theta_{o \times u}, (o \times u)))\}$$

5. $\mathbb{V}_a(E) := \mathbb{V}(p, E) \cap \mathbb{F}_a$ is called the *field of view* of the camera with parameters a .

In literature, the set $\mathbb{V}_a(E)$ has several names, e.g., it is called “field of view” (FoV) by [2] or “viewshed” by [101]. Also further names like “camera beam” are thinkable. Nevertheless, it is not to be mistaken for the “depth of field”, giving the distance between the furthest and nearest objects that appear in acceptable sharp focus, [48]. For the set \mathbb{F}_a the name “camera cone” [70] exists next to the name “viewing frustum” [48]. The opening angles of a camera frustum are not limited to values smaller than π . However, if opening angles larger than π are used then \mathbb{F}_a is not an intersection but a union of θ -spaces.

Set of Sensor Labels and Surveillance Parts

Usually, not all the points in an environment need to be monitored by the sensor network. Additionally, the environment is sometimes rasterized into small cubes, called *voxels*. By using voxels the set of points that need to be covered can be discretized, e.g. the set encompasses the center or corners of each voxel. Let $\mathbb{A} \subset E$ denote the surveillance area of the environment, i.e. the points that need to be covered.

With a given position and orientation of the camera $(p, o) \in \mathbb{P}$ we are able to partition the environment as well as the surveillance area into detectable and undetectable regions. This is done by assigning a label of \mathbb{S} to each point in E by the mapping σ . The following definition shows which labels in \mathbb{S} need to be used.

Definition 2.1.5

Let $E \in \mathbb{E}$ be an environment and $\mathbb{A} \subset E$ be the surveillance area. Let $a = (p, o) \in \mathbb{P}$ be camera parameters as in Inclusion (2.1).

1. A point $y \in \mathbb{A}$ is called *out of range* of a camera with parameters a if $y \notin \mathbb{F}_a$.
2. A point $y \in \mathbb{A}$ is called *detectable* by a camera with parameters a if $y \in \mathbb{V}_a(E)$. Otherwise it is *undetectable* by the camera with parameter vector a .
3. A point $y \in \mathbb{A}$ is called *occluded* if $y \notin \mathbb{V}(p, E)$ and $y \in \mathbb{F}_a$.

Let the set of sensor labels be $\mathbb{S} := \{\text{detectable}, \text{occluded}, \text{out of range}\}$. With the given environment E and given camera parameters $a \in \mathbb{P}$ including the position of the camera p , we can classify whether a part $y \in \mathbb{A}$ is detectable, occluded, and out of range by the map $\sigma : E \times \mathbb{P} \times \mathbb{A} \rightarrow \mathbb{S}$. This map is illustrated in Figure 2.3.

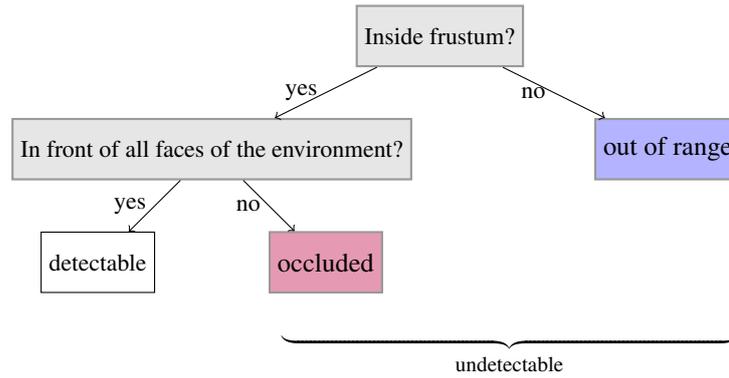


Figure 2.3: Tree of surveillance area division: The gray nodes categorize a point in the surveillance area according to the camera position and orientation. The colored nodes correspond to the labels in $S \subset \mathbb{S}$. Blue corresponds to a label which a point outside of the viewing frustum \mathbb{F}_a is marked with. A red labeled point is in \mathbb{F}_a but not in the set of visible points $\mathbb{V}(p, E)$. A point labeled in white is in the field of view of the camera.

Let $N \in \mathbb{N}$ be the number of cameras in the network and let $x = (a_1, \dots, a_N) \in \mathbb{P}_1 \times \dots \times \mathbb{P}_N$ be the variable vector of the camera network. In an optimization, the detectable regions of the surveillance area can be maximized in order to increase the efficiency of a camera network. One example of sensor network optimization which can be derived from Equation (1.1) and Definition 1.2.1 can be denoted by

$$\max_{x \in \mathbb{P}_1 \times \dots \times \mathbb{P}_N} \lambda(C_x(S)) \quad \text{with } S = \{\text{detectable}\}.$$

where the fused coverage C is defined by the preimage $\sigma^{-1}(S)$ of the sensor map as in Definition 1.2.1 and $\lambda(\cdot)$ denotes the volume of a set. The set $C_x(\{\text{detectable}\})$ is called the *detectable coverage*, further on. In addition to the detectable coverage, in the next section, the notion of the fused coverage is used to express the silhouette of a human.

2.1.2 3D Background-Subtraction Method

In this section, the fused coverage exemplarily resembles a human silhouette. For constructing a silhouette of a human by one camera, the *Background Subtraction Method* (BG subtraction) can be used. This is a method to detect changes in an environment. The method distinguishes between regions of the environment which have changed since the moment a reference image has been taken and regions that remained unchanged. Additionally, regions are considered which are out of range or occluded behind objects, like walls, doors, tables, and racks. Therefore, let the set of sensor labels of Definition 1.2.1 become

$$\mathbb{S} := \{\text{identical}, \text{changed}, \text{occluded}, \text{out of range}\}.$$

In order to explain the method further, it is necessary to distinguish the objects in an environment, [64]:

Definition 2.1.6

Let $E \in \mathbb{E}$ be an environment.

- A set of faces of E is called *object* if it is the boundary of a polyhedral area.

- Faces of E that do not change in time are called *static*, all other faces are called *dynamic*. The set of static faces will be denoted by $E_s \subset \partial E$ and the set of dynamic faces by $E_d \subset \partial E$.

In the following investigation, assume that E_s and E_d are static and dynamic objects. With this definition, the changes of an environment are directly depending on the dynamic objects. The method of background subtraction detects these dynamic objects in the following way: The camera takes an image of the environment in absence of any dynamic objects in the environment, this image is called *reference image* at time $t_0 \in \mathbb{R}$, say. Such an image is illustrated in Figure 2.4 (left). The image is rasterized into small squares called *pixel*. The boundary of the environment at this time is just consisting of static objects E_s . The pixel value of an image that is taken at any moment of time $t > t_0$ after t_0 is subtracted from the value of the pixel in the reference image at the same spot. The result of each pixel can be saved in an additional image, called *silhouette image* in [107]. In Figure 2.4, an image showing the additional dynamic objects (middle) and a silhouette image (right) are illustrated.

If a pixel value of the silhouette image is zero, the part of environment boundary which is mapped onto the pixel has not changed, otherwise a dynamic object has appeared. The pixel is called *background pixel* if it has the value zero, and *foreground* if it has not. The constant assumption is made that dynamic objects in the second picture differ from the static objects in the first picture (e.g., in color).



Figure 2.4: Illustration of the reference image (left) showing the static objects of an environment at time t_0 and an image showing the additional dynamic objects (middle) after $t_0 \in \mathbb{R}$ at time $t > t_0$. The pixel value of the image at time $t > t_0$ is subtracted from the value of the pixel in the reference image at the same spot. This results in the silhouette image (right) with pixel values larger (red) and equal zero (white).

In the context of BG subtraction, each point of the surveillance area can be marked as either one of the labels *identical*, *changed*, *occluded*, and *out of range*. Thereby, the camera sensor in the original sense becomes a sensor for classifying identical, changed, occluded and out of range regions of the surveillance area. Given the cameras position, a point $y \in \mathbb{A}$ can be classified as one of the labels *occluded* or *out of range* just as in Section 2.1.1 with one exception: The second choice, whether or not the point is *occluded*, only regards the static objects and not the dynamic objects.

Points that are inside the viewing frustum of the camera and in front of all static objects are detectable points. A detectable point is mapped onto a specific pixel by projecting it onto the image plane and assigning a specific pixel, e.g., by rounding. It is labeled the following way:

Definition 2.1.7

Let $\mathcal{I} \equiv \mathcal{I}(p, o) \subset E$ be the *image plane* of the camera and $\rho : \mathcal{I} \rightarrow \mathbb{N}^2$ denote the rasterization of the

image plane into pixels. Let the function $\pi : \mathbb{V}_a(E) \rightarrow \mathcal{I}$ denote the projection of a detectable point onto the image plane.

- A point $y \in \mathbb{V}_a(E)$ is called *identical* if $\rho(\pi(y))$ is a background pixel.
- Otherwise it is called *changed*.

The proposed *three-dimensional background-subtraction* method (3DBGS) uses the image space to determine regions with the labels *changed* and *identical* in the surveillance area. The procedure is illustrated in the Figures 2.5 (left).

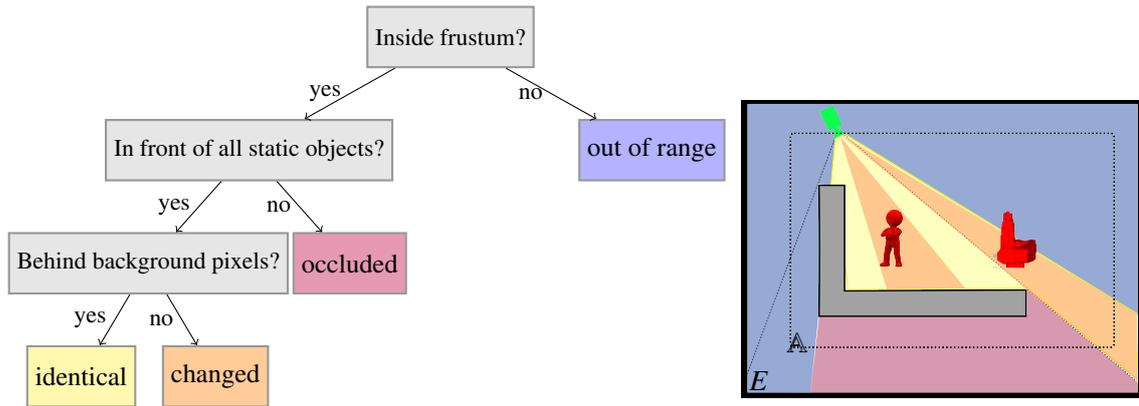


Figure 2.5: By the 3DBGS, the surveillance area is subdivided into *identical* (yellow), *changed* (orange), *out of range* (blue), and *occluded* (red) regions. This is illustrated by a tree (left) and a 2D environment including a pinhole camera model (right) with corresponding colors.

Left: The colors correspond to Figure 2.3. An orange marked point may have changed during surveillance at time $t > t_0$ and yellow labeled points definitely have not changed.

Right: Illustration of an environment E with static (gray) and dynamic (red) objects with a pinhole camera model (green). The surveillance area (contoured, dotted) is part of the environment $A \subset E$.

Assuming the camera is a pinhole camera, the 3DBGS labels the detectable points of the surveillance area as illustrated in Figure 2.5 (right). The identical parts (yellow) definitely do not include any dynamic objects: Let $p \in E$ be the position of the camera's pinhole. Let $x \in \mathcal{I}$ be a point on a background pixel in the image plane \mathcal{I} . The ray $[p, x)$ will not intersect any objects, dynamic objects in particular. Thus, the region projected by a background pixel will always be *identical* to the region at the moment t_0 . The remaining points are marked as *changed*.

The 3DBGS has been discussed for a single camera. It will be generalized for other types of sensors and more than one sensor in the next section.

2.1.3 Conservative Approximation

In the motivation of this thesis, the 3DBGS is used to approximate a human silhouette of multiple views. Previously, authors, such as [78] and [162], have proposed a 3DBGS for approximating an object in different applications, i.e. to protect humans and to count objects, respectively. Both approximations need

to be quite accurate in order not to invoke a false alarm or to miss an object when counting. The authors utilize several cameras to increase the accuracy of the approximation. The method is demonstrated in this section independently to the type of sensors.

In an optimization, the network parameters can be adjusted such that the human silhouette is approximated most accurately. In order to increase the accuracy of an approximation in this section a special type of approximation is used:

Definition 2.1.8

Let E be an environment and let $\mathbb{A} \subset E$ be a surveillance area.

- One of the objects $T \subset \partial E$ of the environment is designated to be a *target* and should be approximated. For this aim, $T \subset \mathbb{A}$ should hold.
- An object which is not a target is called *obstacle*.
- A set $\mathcal{A} \subset E$ is called *conservative approximation* for target T if $T \subset \mathcal{A}$.

A conservative approximation \mathcal{A} of a given target T is ideal for an optimization in which the approximation is to be made more accurate: If the approximation is minimized while keeping the approximation conservative, then the shape of \mathcal{A} assimilates the target's shape and never underestimates the target. Such a conservative approximation is now going to be derived by the 3DBGS which can only approximate dynamic targets. The objects of Table 2.1 occur in the environment:

| | target | obstacle |
|---------|--------|----------|
| static | | ✓ |
| dynamic | ✓ | ✓ |

Table 2.1: The target in this thesis is moving with time, thus, is a dynamic target.

Therefore, let the target be a dynamic object. Let a sensor be given by the parameter vector $a \in \mathbb{P}$ and the sensor map σ as in Definition 1.2.1. As the distinction in Figure 2.5 suggests, a chosen environment $E \in \mathbb{E}$ including static and dynamic objects can be divided into the following regions:

$$\mathbb{A}_{id}^{(a)} := \sigma_a^{-1}(\text{identical}), \quad \mathbb{A}_{ch}^{(a)} := \sigma_a^{-1}(\text{changed}), \quad \mathbb{A}_{or}^{(a)} := \sigma_a^{-1}(\text{out of range}), \quad \mathbb{A}_{oc}^{(a)} := \sigma_a^{-1}(\text{occluded}).$$

Since the target is a dynamic object, the important regions are the identical regions: The following lemma shows where dynamic objects can be found.

Lemma 2.1.9

Let $T \subset \mathbb{A}$ be a target of sensor σ with parameter vector $a \in \mathbb{P}$. Then $\mathbb{A} \setminus \mathbb{A}_{id}^{(a)} = \mathbb{A}_{ch}^{(a)} \cup \mathbb{A}_{or}^{(a)} \cup \mathbb{A}_{oc}^{(a)}$ is a conservative approximation of T .

Proof. After being preimages, the coverages of different sensor labels define a pairwise disjoint conjunction of \mathbb{A} , i.e.

$$\mathbb{A} = \mathbb{A}_{id}^{(a)} \cup \mathbb{A}_{ch}^{(a)} \cup \mathbb{A}_{or}^{(a)} \cup \mathbb{A}_{oc}^{(a)}, \quad \text{with } \emptyset = \mathbb{A}_{id}^{(a)} \cap \mathbb{A}_{ch}^{(a)} = \dots = \mathbb{A}_{or}^{(a)} \cap \mathbb{A}_{oc}^{(a)}$$

Let E_s be the static objects of E and let $y \in \sigma_{(E,a)}^{-1}(\{\text{identical}\}) = \mathbb{A}_{id}^{(a)}$. So, with Definition 2.1.7 the point $\rho(\pi(y))$ is in a background pixel which shows a part of the static objects E_s . Thus, either y is part of a static object $y \in E_s$ or it is not part of an object but only part of the *empty space* $y \in {}^iE$. However, if $y \in T$ neither is possible since $T \subset {}^\partial E \setminus E_s$, which leads to the conclusion that $T \subset \mathbb{A} \setminus \mathbb{A}_{id}^{(a)}$. \square

Therefore, as long as the target is approximated by both the coverages of changed and undetectable labels, such an approximation is conservative. Since this inclusion holds for one sensor with parameter vector $a \in \mathbb{P}$, the following inclusion holds for any target T if we consider a sensor network consisting of N sensors with camera parameter vectors a_n , $n = 1, \dots, N$. The set on the right side is already expressed by the fused coverage C from Definition 1.2.1 where a part of the surveillance area \mathbb{A} is covered by at least one camera.

Lemma 2.1.10

Let $T \subset \mathbb{A} \subset E$ be a target of $N \in \mathbb{N}$ sensors σ with parameter vector $(a_1, \dots, a_N) \in \mathbb{P}_1 \times \dots \times \mathbb{P}_N$.

$$T \subset \left(\mathbb{A} \setminus \mathbb{A}_{id}^{(a_1)} \cap \dots \cap \mathbb{A} \setminus \mathbb{A}_{id}^{(a_N)} \right) \equiv \mathbb{A} \setminus C_{(a_1, \dots, a_N)}(\{\text{identical}\})$$

This means the term on the right hand side is a conservative approximation of the target.

Such an approximation is more descriptive than the term in Lemma 2.1.9, since all the sensors are used. In Figure 2.6 (left), the 3DBGS is demonstrated with the pinhole camera model from Figure 2.5. The identical parts (yellow) definitely do not include any dynamic objects. This is also true for two cameras (middle), the more cameras the more parts of the surveillance area can be left out of the approximation of a target. The approximation (right, blue) is constructed of these points in the surveillance area that are marked as “not identical” from the point of view of all cameras.

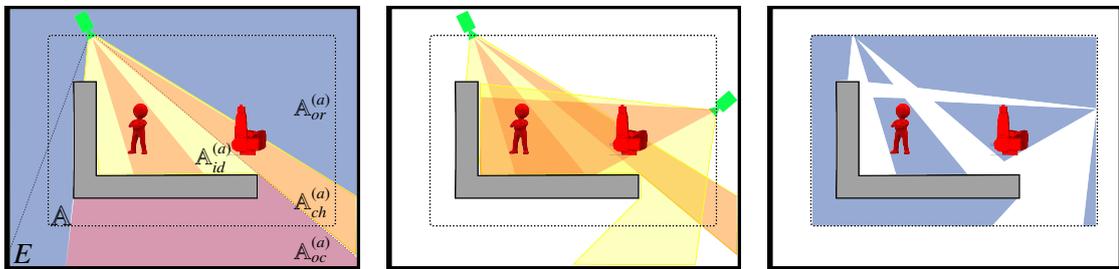


Figure 2.6: Left: Illustration of an environment E as in Figure 2.5. Middle: Two cameras instead of one decrease the size of the region where dynamic objects (red human and robot) can be located. Right: The conservative approximation (blue area) of the target (human) by two cameras is smaller than the approximation by one (not illustrated).

All the images show that the approximation is conservative even if obstacles are in the scene. E.g., if the human icon resembles the target then a dynamic obstacle (robot) and a static obstacle (gray wall) obscure the cameras. Nevertheless the approximation (blue) includes the target. Thus, with a 3DBGS discussed in this section $\mathbb{A} \setminus C_{(a_1, \dots, a_N)}(\{\text{identical}\})$ is a conservative approximation of the target. The set $C_{(a_1, \dots, a_N)}(\{\text{identical}\})$ is called the *identical coverage*, further on.

Lemma 2.1.10 is ideal for an optimization in which a conservative approximation \mathcal{A} of a given target T is to be made more accurate: For a conservative approximation holds $T \subset \mathcal{A}$. If the approximation

is minimized with respect to conservativity, then \mathcal{A} approximates the target's shape more and more accurately and never underestimates the target. The quality of the approximation can be optimized by this fused coverage

$$\min_{x \in \mathbb{P}_1 \times \dots \times \mathbb{P}_N} \lambda(\mathbb{A} \setminus C_{(a_1, \dots, a_N)}(S)) \quad \text{with } S = \{\text{identical}\}.$$

Note that this does not mean that the approximation converges to the target. In particular, this is not the case when dynamic obstacles are in the environment: The approximation encompasses points of dynamic obstacles which do not belong to the target. These points will not be excluded from the approximation during the above optimization. Nevertheless, the objective function is suitable for sensor network optimization: Given a moment in time, the volume of the dynamic obstacles does not depend on the sensor network parameters. An addend of the objective function that is constant in the variable vector does not change the optimization. The objective function has been used for camera network optimization for one-dimensional images of the cameras and a two-dimensional environment already, [47].

2.1.4 Increased Reliability

In the last section, the approximation is formed by the parts of the surveillance area which are marked as identical at least once. But the labels of one sensor could be inaccurate, e.g., if a target is similar to the static objects. Since this is a safety measurement, it would be better to form the approximation by a region which is marked by at least two or three sensors instead of one. We will discuss a version of the fused coverage, now, which is refined by the number of trusted sensors $k \in \mathbb{N}$, $k \leq N$ with a general set of sensor labels $S \subset \mathbb{S}$. Its properties will help to show that the maximization of identical regions yields the same result as the minimization of the approximation error.

For an environment E , a camera parameter vector $a \in \mathbb{P}$, and the surveillance area $\mathbb{A} \subset E$ the following notation is used:

$$\mathbb{A}_S^{(a)} := \sigma_{(E,a)}^{-1}(S) \equiv \{p \in \mathbb{A} \mid \sigma_{(E,a)}(p) \in S\}$$

In Definition 1.2.1, we have distinguished two types of fused coverages of N sensors, considering whether part is meant to be covered by *all* or *at least one* sensor. Now, we will establish a generally valid fused coverage C for the case that at least $k \in \mathbb{N}$ sensors need to cover a part.

Definition 2.1.11

Let \mathbb{A} be a fixed surveillance area in a fixed environment E and let $x = (a_1, \dots, a_N) \in \mathbb{P}_1 \times \dots \times \mathbb{P}_N$ be the parameters of a network. Let σ be the map of surveillance parts to labels, as in Definition 1.2.1. Let $k \in \mathbb{N}$ with $k \leq N$ be a given integer. Let $C_N(k) \subset S_N$ be the set of k -combinations of the N -tuple $(1, \dots, N)$ and order $\frac{N!}{k!(N-k)!}$. The elements of the set are determined by choosing k sensors out of N total without replacement disregarding the sequence of the tuple.

- The k -reliable coverage of the sensor network is defined by

$$C_{(x,E)}(k, S) \equiv C_x(k, S) = \bigcup_{\pi \in C_N(k)} \bigcap_{n=1}^k \mathbb{A}_S^{(a_{\pi(n)})} \subset \mathbb{A} \quad (2.2)$$

In the following sections, we will consider E chosen as constant, thus, it can be dropped in the notation. To ease on the notation we drop the parameter vector of the network x , too.

- The number k is called *reliability threshold*.
- Define the *threshold region* $C_=(k, S)$ of exact $k \leq N$ sensors as

$$C_=(k, S) := \begin{cases} C(k, S) \setminus C(k+1, S) & \text{if } k < N \\ C(k, S) & \text{if } k = N. \end{cases}$$

The bigger the reliability threshold k , the more sensor coverages $\mathbb{A}_S^{(a_{\pi(n)})}$ are intersected, and the more restrictive is the k -reliable coverage. In the case of the most restrictive solution $k = N$, this means that there is $\frac{N!}{N!(N-N)!} = 1$ possibility to choose N sensors out of N total. $C(N, S)$ then resembles the fused coverage of Definition 1.2.1 given that a part of the surveillance area needs to be covered *by all* sensors:

$$C(N, S) := \mathbb{A}_S^{(a_1)} \cap \dots \cap \mathbb{A}_S^{(a_N)} \quad (2.3)$$

Next, consider $N = 3$ sensors with a reliability threshold $k = 2$. A point of the surveillance area is in the 2-reliable coverage if it is marked by at least two sensors with any label of S . If two random sensors a_1 and a_2 are chosen only their overlapping coverage $\mathbb{A}_S^{(a_1)} \cap \mathbb{A}_S^{(a_2)}$ is part of the 2-reliable coverage. $C(2, S)$ encompasses all three possibilities to choose two sensors out of three, (a_1, a_2) , (a_2, a_3) , and (a_3, a_1) :

$$C(2, S) = (\mathbb{A}_S^{(a_1)} \cap \mathbb{A}_S^{(a_2)}) \cup (\mathbb{A}_S^{(a_2)} \cap \mathbb{A}_S^{(a_3)}) \cup (\mathbb{A}_S^{(a_3)} \cap \mathbb{A}_S^{(a_1)}).$$

In the case of $k = 1$, the 1-reliable coverage consists of the united sensor coverages $\mathbb{A}_S^{(a_{\pi(n)})}$. Thus, the k -reliable coverage contains all the points in \mathbb{A} which are marked by one of the labels in S at least k times. This is illustrated in Figure 2.7 for a network utilizing three cameras and the sensor labels *detectable* and *undetectable*. In the left image, three cameras and their detectable coverage of the environment are depicted. The yellow region resembles the 1-reliable coverage $C(1, S)$. In the middle, the 2-reliable coverage is illustrated as an additional blue area.

In contrast to the points that are marked with a label of S at least k times, the points which are marked *exactly* k times are contained in the threshold region $C_=(k, S)$. These regions are introduced in this section for the proof of the upcoming theorem. The threshold regions constitute pairwise disjoint sets that completely cover the whole surveillance area. In Figure 2.7 to the right the threshold regions $C_=(k, S)$ are illustrated that a surveillance area can be decomposed into.

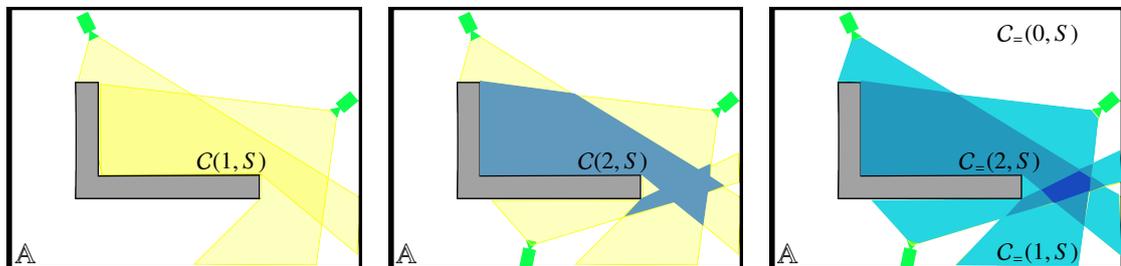


Figure 2.7: Illustration of three cameras (green) including their detectable coverage (left, yellow), the 2-reliable coverage (middle), and their threshold regions regarding the set of sensor labels $S = \{detectable\}$ (right) in the surveillance area \mathbb{A} .

Furthermore, $(N - k)$ threshold regions can be combined to a k -reliable coverage.

Lemma 2.1.12

Let the assumptions of Definition 2.1.11 hold. The coverage $C(k, S)$ can be constructed of $N - k + 1$ threshold regions:

$$\bigcup_{\kappa=k}^N C_{=}(\kappa, S) = C(k, S)$$

$$\text{Proof.} \quad \bigcup_{\kappa=k}^N C_{=}(\kappa, S) = \bigcup_{\kappa=k}^{N-1} \underbrace{C_{=}(\kappa, S)}_{C(\kappa, S) \setminus C(\kappa+1, S)} \cup C_{=}(N, S) = C(k, S) \setminus C(N, S) \cup C(N, S) = C(k, S)$$

With help of this observation, the coherence between the set of sensor labels $S \subset \mathbb{S}$ and its compliment and the coherence between the k -reliable coverage and its complement can be examined. If a part of the surveillance area is marked with one of the labels in S by at least $k > 0$ sensors, the remaining part is marked with the remaining labels $\mathbb{S} \setminus S$ at least $N - (k - 1)$ times:

Theorem 2.1.13

Let the assumptions of Definition 2.1.11 hold.

$$C(k, S) = \mathbb{A} \setminus C(N - (k - 1), \mathbb{S} \setminus S).$$

$$\begin{aligned} \text{Proof.} \quad C(k, S) &= \bigcup_{\kappa=k}^N C_{=}(\kappa, S) = \bigcup_{\kappa=0}^N C_{=}(\kappa, S) \setminus \bigcup_{\kappa=0}^{k-1} C_{=}(\kappa, S) \quad C_{=} \text{ pairwise disjoint} \\ &= \mathbb{A} \setminus \bigcup_{\kappa=0}^{k-1} C_{=}(\kappa, S) = \mathbb{A} \setminus \bigcup_{\kappa=N-(k-1)}^N C_{=}(\kappa, \mathbb{S} \setminus S) \quad \square \end{aligned}$$

The target can be conservatively approximated by successively excluding the cameras' identical coverage from the surveillance area with $\mathbb{A} \setminus C(1, \{\text{identical}\})$ of Lemma 2.1.10 (in our recent notation). The reliability threshold 1 in $\mathbb{A} \setminus C(1, \{\text{identical}\})$ means that *each* identical coverage is going to be excluded. This is advisable if all sensors can be trusted to label the parts of the surveillance area without error, i.e. no objects blend in with the background from any point of view. With Theorem 2.1.13 we know that the approximation is equivalent to:

$$\mathbb{A} \setminus C(1, \{\text{identical}\}) = C(N, \mathbb{S} \setminus \{\text{identical}\}) = C(N, \{\text{changed, out of range, occluded}\})$$

Excluding each camera's identical coverage amounts to the parts of the surveillance area that are commonly considered as *changed, out of range, or occluded* by all the cameras. Less obvious coherences are also established by the theorem above for reliability thresholds larger than 1. Additionally, the theorem can be used in an optimization of the volume of the k -reliable coverage: Instead of decreasing the volume of $C(k, S)$ in an optimization of the network, we could as well increase the volume of $C((N - (k - 1)), \mathbb{S} \setminus S)$ and vice versa.

Corollary 2.1.14

Let $T \subset \mathbb{A}$ be a dynamical target of a camera network with parameters $x \in \mathbb{P}_1 \times \dots \times \mathbb{P}_N$ using a 3DBGS. Let $k \in \mathbb{N}$, $k \leq N$ be a reliability threshold. Then the following is true:

- $C_x(k, \{\text{changed, out of range, occluded}\})$ is a conservative approximation of T .

- Let λ denote the volume of a set. The approximation error can be minimized by

$$\max_{x \in \mathbb{P}_1 \times \dots \times \mathbb{P}_N} \lambda(C_x(N - (k - 1), \{identical\})).$$

Proof. It holds

$$\begin{aligned} T & \underbrace{\subset}_{\text{Lemma 2.1.10}} \mathbb{A} \setminus C(1, \{identical\}) \underbrace{=}_{\text{Theorem 2.1.13}} C(N, \{changed, out of range, occluded\}) \\ & \underbrace{\subset}_{\text{Lemma 2.1.12}} C(k, \{changed, out of range, occluded\}). \end{aligned}$$

Thus, the latter is a conservative approximation. When minimizing a conservative approximation, the target is never underestimated and constitutes a lower bound to the volume of an approximation: With Theorem 2.1.13 it holds

$$\begin{aligned} \lambda(T) & \leq \lambda(C(k, \{changed, out of range, occluded\})) = \lambda(\mathbb{A} \setminus C(N - (k - 1)\{identical\})) \\ & = \lambda(\mathbb{A}) - \lambda(C(N - (k - 1), \{identical\})) \end{aligned}$$

Since $\lambda(\mathbb{A}) \geq \lambda(C(N - (k - 1), \{identical\}))$ minimizing $\lambda(C(k, \{changed, out of range, occluded\}))$ is the same as maximizing $\lambda(C(N - (k - 1), \{identical\}))$. \square

In order to decrease the error of the approximation, instead of minimizing the volume of the approximation $C(N, \{changed, out of range, undetectable\})$ we can as well maximize the volume of $C(1, \{identical\})$. Summarized, in this section a refined version of the fused coverage of Definition 1.2.1 regarding the number of trusted sensors has been developed with a general set of sensor labels $S \subset \mathbb{S}$. It has been transferred to a 3DBGS, a method which is used in a camera network to approximate a target conservatively. The important result for sensor network optimization: There is no difference in maximizing the volume of the identical coverage and minimizing the volume of the approximation of a target when abiding the threshold rule of Theorem 2.1.13.

In the upcoming chapters, the volume of the above k -reliable coverage $\lambda(C_x(k, S))$ is used as an objective function of problem (1.1). If the quality of the approximation needs to be improved then $S = \{identical\}$ is used, and otherwise if the detectable regions of the surveillance area need to be enlarged we set $S = \{detectable\}$.

2.2 Implementation of the Coverage of Multiple Cameras

The function $\lambda(C_x(k, S))$ is used as an objective function to camera network optimization in this thesis. It resembles the volume λ of the fused coverage C of $N \in \mathbb{N}$ cameras in a network with parameters $x \in \mathbb{P}_1 \times \dots \times \mathbb{P}_N$. C can be modified to resemble both, the total field of view of the cameras if $S = \{detectable\}$ and the approximation error of a human silhouette if $S = \{identical\}$. The number $k \leq N$ stands for the minimum number of cameras which cover a point in the coverage with a label of S .

The function $C_x(k, S)$ is not given as an analytic formula. For the optimization, the points inside a camera frustum that are not occluded by obstacles such as racks, walls, and doors need to be deduced geometrically. Such a deduction has to be done for all cameras taking into account the environment with obstacles, the surveillance area, the placement of the cameras, the sensor labels $S \subset \mathbb{S}$, and the reliability threshold k . The deduction of the k -reliable coverage of several cameras will be called *visibility analysis*,

as in case of the deduction of the detectable coverage in [101]. After the visibility analysis, for a particular state of network parameters, e.g., for particular positions and orientations of the cameras, the quality of the network can be estimated from the k -reliable coverage by applying a suitable measure, e.g., the volume of the covered regions in total.

In this section, some details are given how to implement the visibility analysis and the measurement of the volume as an objective function. In Section 2.2.1, the basic implementation of the k -reliable coverage together with an upper bound for the complexity of the implementation is stated. Then, in Section 2.2.2, we discuss some general ideas to accelerate the generation of the coverage. In Section 2.2.3, we develop two methods to accelerate the objective function when being evaluated several times in a row, such as done in an optimization.

2.2.1 Basic Implementation

The k -reliable coverage of several cameras is an intersection or union of all camera coverages. Thus, the visibility analysis requires to fuse *several polyhedral areas*. Set operations on polyhedra are known to be a non-robust computation: I.e. when two polygons are tangentially contacted or they intersect in only one edge, numerical errors can lead to topology inconsistencies, see [157].

In order to cope with set operations, the most commonly used data structure for the coverage is the following: The surveillance area is discretized into an orthogonal grid, composed of small cubes of the room, called *voxels*. The coverage of one camera is then a collection of the voxels that are covered. The data structure is also called *binary occupancy grid*, referring to occupied regions in robotic applications [152]. In order to derive the fused coverage of the whole camera network, a set operation such as the union or intersection needs to be applied over all camera specific occupancy grids. After the set operation, the quantized volume of the coverage is derived by adding up the volume of the covered voxels.

Therefore, we assume that the environment of the camera network is a collection of solid, static and dynamic objects. These solid objects are represented by a boundary representation, i.e. their boundary is made out of polygons called *faces*. Let the environment include $f_s, f_d \in \mathbb{N}$ static and dynamic faces with a constant number of faces $f = f_s + f_d$ in each frame during the surveillance time. Let the dynamic objects move on a trajectory which is discretized into $t - 1$ time intervals or rather $t \in \mathbb{N}$ time steps. Let the occupancy grid of each camera have $v \in \mathbb{N}$ voxels. Let each voxel be treated as a point in space, in particular a *voxel check* is defined as an operation applied to the voxel cube's center in the surveillance area. One could as well use the voxel's vertices or the edges, but to show the following procedure the center suffices. Let there be $N \in \mathbb{N}$ cameras in the network. Also, let O denote an upper bound of calculation steps within the simulation.

Consider an occupancy grid of a single camera's coverage. The question is how to mark each voxel with the right sensor label in S in an efficient manner. The voxel check for *out of range* is merely a localization of the voxel according to the boundary planes of the camera frustum. Additionally, one variant of marking the voxels of the detectable coverage, i.e. the voxels which are *not out of range*, is an inverse *ray tracing*: The half line starting at the focal point of a camera and a point in the image plane is called *ray*. Usually, the value of the image at such a point is determined by intersecting the ray and the faces of the environment. Conversely here, the sensor label of the voxel and not the value of the pixel is determined. This is accomplished by intersecting the segment between camera position and the

voxel with all the faces of the environment. In case of a non-empty intersection, the voxel is *occluded*. This test needs to be done for each camera, each voxel and each face in each timestep, so the complexity of the simulation has an asymptotic behavior of $\mathcal{O}(N \cdot v \cdot t \cdot f)$. If dynamic and static faces are stored in separate data structures this can be reduced to the following upper bound to the asymptotic behavior since static faces only have to be checked once over all timesteps:

$$\mathcal{O}(N \cdot v \cdot (f_s + t \cdot f_d))$$

We will now give a visibility analysis that synthesizes camera images, which is cheaper and can be used to derive the identical coverage as well as the detectable coverage. It is illustrated in Figure 2.8.

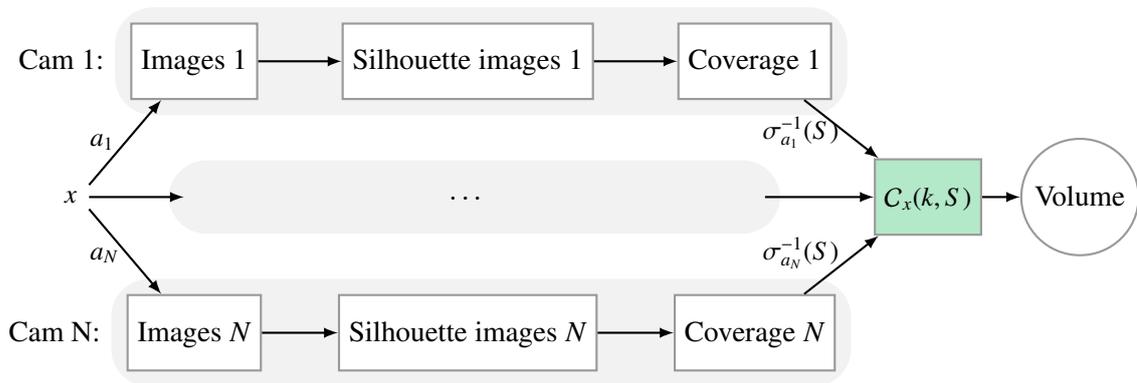


Figure 2.8: Illustration of a single function call $\lambda(C_x(k, S))$ with the method of image synthesis. Each camera synthesizes one (in case of the detectable coverage) or more images (in case of the identical coverage). These images are converted to silhouette images in 2D (in case of the identical coverage, Sections 2.1.2) and projected to each camera's coverage in 3D. The fused coverage $C_x(k, S)$ is measured by the volume. The calculations corresponding to each sensor (gray) are independent on another sensor's parameters.

Instead of tracing the ray between the camera position and each voxel, the idea is producing a depth image of the faces: The depth image is discretized into a number of $p \in \mathbb{N}$ little squares which are called *pixels*. The only difference to color images is that these images do not store a color value in each pixel, the pixels contain the distance between the camera position and the next face. The following procedure distinguishes whether or not a voxel is detectable: Project the voxel into the image plane and thereby determine the pixel it should be seen in, as by $\rho(\pi(\cdot))$ in Definition 2.1.7. If there is no pixel, the voxel is outside the viewing frustum and thus *out of range*. Otherwise, if there is a pixel, compare the distance that is stored in this pixel to the distance between voxel and camera. If the latter is larger then the voxel is *occluded*.

To derive the identical coverage, we produce more than one depth image. This is a way how to obtain the labels *identical* and *changed*: The first image is a background depth image showing only the static faces. An additional depth image per timestep including all the faces (static and dynamic) is produced. A background subtraction on the depth images yields a silhouette image per time step, as described in Section 2.1.2. If a voxel is detectable and is projected onto a background pixel then the voxel is labeled

by *identical* otherwise *changed*. An upper bound for the complexity of this approach is:

$$\mathcal{O}\left(\underbrace{N \cdot p \cdot (f_s + t \cdot f)}_{\text{Depth image}} + \underbrace{t \cdot N \cdot (p + v)}_{\text{BG subtr. and coverage}}\right) \quad (2.4)$$

as can be seen in detail in Table 2.2. The latter method is cheaper when using a large amount of voxels or a denser occupancy grid. For a sufficiently large amount of voxels the inverse ray tracing is more expensive than the pixel based visibility analysis.

$$N(p \cdot (f_s + t \cdot f) + t \cdot (p + v) - v \cdot (f_s + t \cdot f_d)) \leq 0 \Leftrightarrow p \cdot \frac{f_s + t \cdot f + t}{f_s + t \cdot f_d - t} \leq v$$

Thus, the coverage of a sensor can be constructed cheaper by synthesizing images than by inverse ray tracing. The fused coverage can now be derived by intersecting and uniting these camera coverages.

| | Description | Complexity |
|-----------------------|--|--|
| Faces to depth image | Straight forward approach to produce one reference depth image with pixels p including f_s static faces, and a few depth images with all faces for each camera. | $\mathcal{O}(N \cdot p \cdot (f_s + t \cdot f))$ |
| BG subtraction | A silhouette image is produced by subtracting each depth image from the reference depth image. | $\mathcal{O}(t \cdot N \cdot p)$ |
| Deriving the Coverage | The following will be done for each voxel in each timestep: | $\mathcal{O}(t \cdot v) \cdot \dots$ |
| | Project the voxel into the image plane of each camera. The corresponding pixel can be derived by scaling and rounding. | $\cdot \mathcal{O}(N)$ |
| | If a pixel including the projected voxel does not exist in the image plane then the voxel is <i>out of range</i> , otherwise the distance between the camera and the voxel is evaluated | $\cdot \mathcal{O}(N)$ |
| | If the distance is larger than the depth value of the reference image the voxel is <i>occluded</i> | $\cdot \mathcal{O}(N)$ |
| | If the voxel is neither <i>out of range</i> nor <i>occluded</i> it can be labeled as follows: If the pixel of the silhouette image is zero then the voxel is <i>identical</i> , otherwise <i>changed</i> | $\cdot \mathcal{O}(N)$ |

Table 2.2: Table of upper bounds of the calculation steps when deriving the coverage of one single camera. An occupancy grid is used as a data structure for the coverage. The occupancy grid is filled by synthesizing images of the environment.

The construction of the k -reliable coverage has been discussed in this section. Instead of an inverse ray tracing method, we will synthesize depth images and use an occupancy grid as data structure. This

method is improved in the upcoming sections.

2.2.2 Acceleration of the Evaluation

The complexity (2.4) has two parts, the complexity for synthesizing the images and the complexity for using a 3D-background subtraction method in order to obtain the camera coverage. There are several ways to accelerate this simulation.

The first part, the synthesis of the image, can be accelerated by utilizing a tree structure instead of an occupancy grid. An octree is a data structure that saves the faces in a hierarchical order: The faces that share one of the eight octants of a cell of the room are in the same subtree of the octree. Only the faces of suitable octree branches are checked for an intersection of the pixel ray. Thereby the average asymptotic behavior of taking the image improves.

Another way to accelerate the computation of the depth images is to parallelize the computations. In order to give a brief idea on the amplitude of an acceleration of the runtime, Figure 2.9 presents the results of a project [124] in which the depth image computation has been parallelized in different ways.

OpenMP OpenMP is a simple C/C++/Fortran compiler extension which allows to compute loop calls in parallel on several cores of the CPU without significantly having to rewrite existing source code, cf. [112]. The OpenMP has existed since the 1980s [22]. In the project [124], the generation of each depth image by each camera has been parallelized with OpenMP.

OpenGL Since the pixels of an image in each camera can be computed independently and the number of cores of a CPU is rarely sufficient for the number of pixel, a parallelization on a GPU is wanted. Since our aim is indeed a graphical application, we can also use a language more suitable to the original purpose of a graphics card: The Open Graphics Library (OpenGL) Application Programming Interface (API) “began as an initiative by SGI to create a single, vendor-independent API for the development of 2D and 3D graphics applications”, cf. [140], but is now maintained by the Khronos Group, as well as OpenCL. In the project [124], the primitives are not rendered for displaying their color on a screen, instead the stored values in the pixels are depth values.

In Figure 2.9 the runtime of taking an image is illustrated broken down to these two types of parallel computation. Be aware that the runtime is depicted in milliseconds on a logarithmic scale. The figure shows that if a graphics card is available, one should definitely use it to parallelize the computations. The OpenGL-parallelized synthesis only takes 0.021s.

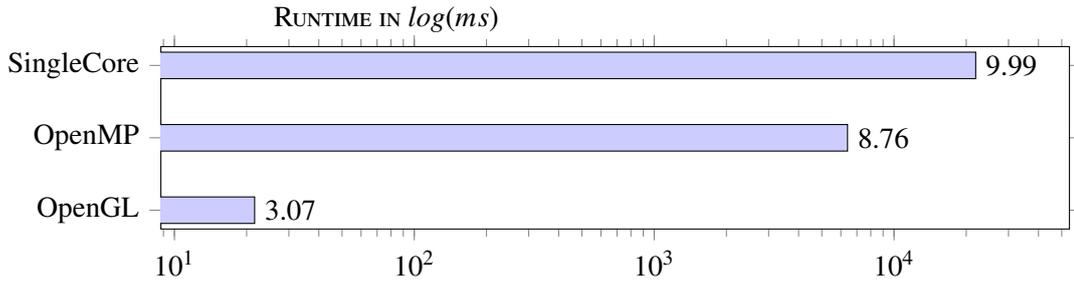


Figure 2.9: Bar plot of the runtime of synthesizing a depth image averaged over five cameras plotted on a logarithmic scale in milliseconds. The environment is rendered in an image of 320x240 pixel and had 304 static faces with 572 vertices and 104634 dynamic faces with 34878 vertices. Hardware: AMD FX-6100 (6 cores) processor, NVIDIA Geforce GTX 570 graphics card.

The runtime of synthesizing the image for the deduction of the coverage can be decreased by parallelization. This acceleration addresses only the first part of the complexity in (2.4). Acceleration approaches fall under the scope of Computer Graphics. The second part can be accelerated by multi-camera approximation approaches which fall under the scope of Computer Vision. The authors of [159], e.g., utilizes spatial and temporal coherency in order to decrease the average asymptotic behavior for the 3D background subtraction subtraction method (second term in the complexity (2.4)). The coherency states that neighboring voxels in space and time do not change their sensor label “a lot”.

An optimization consists of a solver calling the objective function several times. Each function call launches one visibility analysis. In our case the visibility analysis consists of the synthesis of depth images, the construction of each camera’s coverage, and construction of the fused coverage. The above approaches accelerate one single simulation of the k -reliable coverage. Accelerated *subsequent* calls of this simulation can be achieved by the following section.

2.2.3 Sequential Evaluations

In order to accelerate the optimization of camera network parameters we have decreased the runtime of a single objective function call. This was done by accelerating the construction of the coverage of a single sensor in the last two sections, Section 2.2.1 and 2.2.2. In an optimization, the objective function is called several times. In our application *subsequent* function calls can be accelerated by caching the camera coverage $\sigma_{a_n}^{-1}(S)$ for all cameras $n \in \{1, \dots, N\}$.

Caching the camera coverages is advantageous for the following reason: Consider the structure of a single function call $\lambda(C_x(k, S))$ when utilizing the depth image synthesis approach, cf. Figure 2.8. The coverage of each camera is independent on another camera’s position and orientation, which is illustrated in the figure as parallel gray boxes. When caching each camera’s coverage, we utilize their independence for accelerating two (or more) subsequent function calls. In order to show the acceleration, let i and $(i + 1)$ be two subsequent function calls. There are two ways of using the caching for the objective function developed in this thesis:

Recalculation Consider the coverages $\sigma_{a_n}^{-1}(S), n = 1, \dots, N$. If only a single camera $n \in \{1, \dots, N\}$ changed its position or orientation $a_n^{(i)} \rightarrow a_n^{(i+1)}$, then only the coverage $\sigma_{a_n}^{-1}(S)$ and not the

coverage of the remaining sensors $\bar{n} \neq n$ needs to be recalculated. Figure 2.10 shows an example of saving calculation time when repositioning a subset of cameras.

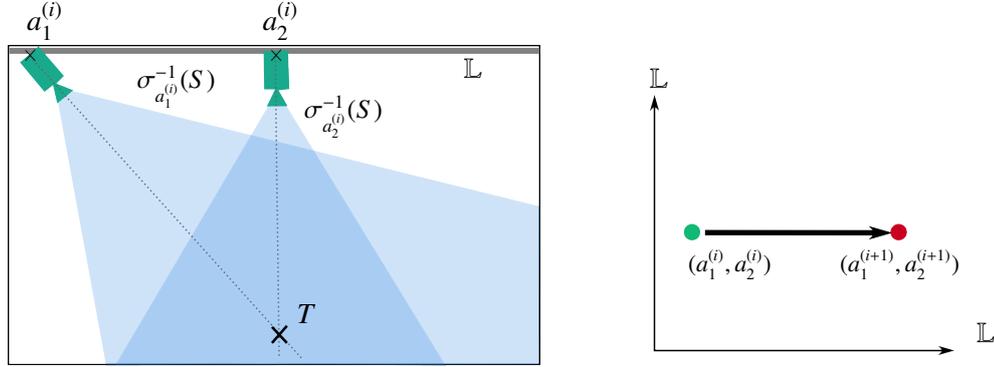


Figure 2.10: Example for the reduction of evaluation time when recalculating the coverages of two cameras. Left: The cameras are placed in \mathbb{L} (gray line) and oriented towards T . Thus, the domain of the objective function is $\mathcal{D} = \mathbb{L}^2$. Right: Illustration of repositioning the cameras from an initial state (green) to an end state (red) in the domain $\mathcal{D} = \mathbb{L}^2$. With $a_1^{(i)}$ being constant, the coverage $\sigma_{a_1^{(i)}}^{-1}(S)$ can be cached. Thus, the recalculation of the objective function

$$\lambda(C_{(a_1^{(i)}, a_2^{(i)})}(1, S)) = \lambda\left(\sigma_{a_1^{(i)}}^{-1}(S) \cup \sigma_{a_2^{(i)}}^{-1}(S)\right) \text{ is less expensive.}$$

Multiplication For all the cameras $n = 1, \dots, N$, consider the camera coverages $\sigma_{a_n^{c_n}}^{-1}(S)$ cached in both the objective function evaluations $c_n \in \{i, (i+1)\}$. Then, the *known* objective values can be multiplied in the following way: The objective function $\lambda(C_x(k, S))$ can be evaluated at all N -tuples $x = (a^{c_1}, \dots, a^{c_N})$ without constructing an additional camera coverage. For these 2^N possibilities, the objective function can be evaluated by applying set operations and measuring the volume, only. When multiplying the known objective values like this, the costly function is evaluated at a grid of solution points. Figure 2.11 shows an illustration of the multiplied variable vectors for which the objective value is known with this method.

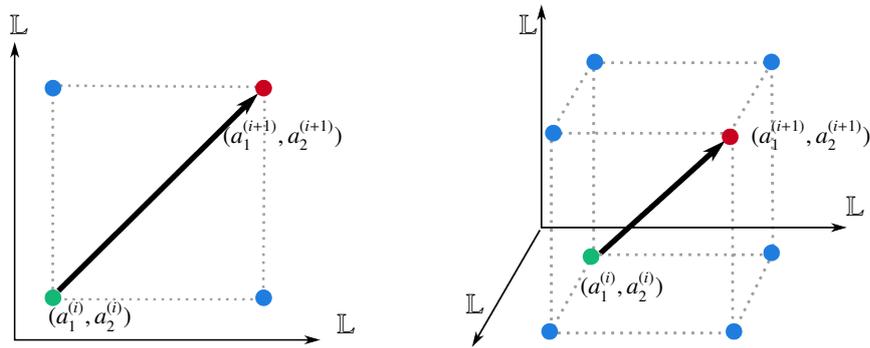


Figure 2.11: Illustration of the multiplied objective function values in the domain $\mathcal{D} = \mathbb{L}^2$ (left) and $\mathcal{D} = \mathbb{L}^3$ (right); The camera coverages of the initial state (i , green) and the end state ($(i+1)$, red) need to be evaluated and cached. The objective function value $\lambda(C_x(k, S))$ of the remaining states (blue) can be calculated by applying the set operation C to the camera coverages and measuring the volume λ .

These methods come into play when calling the objective function more often. The first method accelerates the visibility analysis when having changed only one camera's position or orientation. If the parameters of several cameras are changed the second method can be used. The latter method only affects the number of already evaluated objective function values. This is why the latter method can only be used if the function values are stored. The number of function values which are gained in one of these methods is depicted in the following lemma together with the amount of cost reduction in an optimization:

Lemma 2.2.1

Let the calculations of the coverage of one camera cost c_c (this indeed corresponds to the costs of Equation (2.4)). Let I be the number of times that the function $\lambda(C_x)$ is evaluated by the method of

Recalculation: *Let $\sigma_a^{-1}(S)$ be cached for the parameters $a = a_1^{(0)}, \dots, a_N^{(0)}$ at the initial function evaluation (0). By the method of recalculation, the total number of known function values is I . These function values can be gained with a cost reduction of $(IN - N + 1 + I) \cdot c_c$.*

Multiplication: *Let $\sigma_a^{-1}(S)$ be cached for the parameters $a = a_1^{(i)}, \dots, a_N^{(i)}$ at all the evaluations $i = 1, \dots, I$. By the method of multiplication, the total number of known function values is I^N . These function values can be gained with a cost reduction of $(I^N - I) \cdot N \cdot c_c$.*

Proof. Let a set operation cost c_u and measuring the volume cost c_v . One evaluation of the fused coverage of the cameras has a complexity of $N \cdot c_c + c_u + c_v$.

With the method of recalculation parameters we gain I function evaluations at the cost of $N \cdot c_c + c_u + c_v + (I - 1)(c_c + c_u + c_v)$. The same amount of iterations without this method would lead to the costs $I(N \cdot c_c + c_u + c_v)$. The cost reduction is therefore $(IN - N + 1 + I) \cdot c_c$.

The cost reduction of the second method can be deduced as follows: 2^N function evaluations normally can be gained with the costs $2^N \cdot (N \cdot c_c + c_u + c_v)$. With the method of multiplication we gain the same amount of evaluations with the complexity of $2 \cdot N \cdot c_c + 2^N \cdot (c_u + c_v)$. Thus, we save the expenses of $(2^N - 2)Nc_c$.

If we stored the sensor coverages of I iterations, the number of function evaluations would increase to I^N with the storage of $I \cdot N$, respectively. The saved costs account for $(I^N - I)Nc_c$. \square

Summarized, an optimization consists of a solver calling the objective function several times. In camera placement, each function call launches an expensive visibility analysis. The above methods accelerate the optimization in the following way: The first method saves costs by caching the sensor coverages and only recalculating a part of the visibility analysis. The second method generates multiple objective function values out of two actual visibility analysis'.

In the last sections, the visibility analysis has been constructed by synthesizing depth images. The objective function has been accelerated by accelerating the both the image synthesis and by accelerating subsequent function calls. Optimization solvers that can use these methods will be discussed later on, in Section 3.2.2.

2.3 Geometry of the Coverage of Multiple Cameras

We take a step back away from implementation details and have a look at the k -reliable coverage as it is defined in Section 2.1. In this section, the geometry of the detectable and identical coverage is addressed without being concerned with a voxel implementation. The shape of the k -reliable coverage and its properties will be used in Section 2.4 in order to investigate staircasing and continuity of the function we want to utilize as objective function for optimization.

The shape of the k -reliable coverage depends on the type of sensors that are used in the network. In contrast to the last section, let us assume non-distorted pinhole camera images for the next sections. For the shape of the identical coverage let us assume that a sensor label is assigned by a 3DBGS to a point in the surveillance area. Thus, the used sensor labels are “occluded”, “out of range”, “changed”, and “identical”, as in Section 2.1.4. In the course of this section, let the surveillance area $\mathbb{A} \subset E$ be a 3D-polyhedral area in the environment $E \in \mathbb{E}$, a polyhedron that can be disconnected and flat at some points. This is always true if $\mathbb{A} = E$. Additionally, assume that all the cameras have the same opening angles $\theta_u, \theta_{o \times u} \in (0, \pi]$ to ease the notation. Since the environment is three-dimensional, it is implied that by the term *polyhedral area* a 3D-polyhedral area is meant, and by *polyhedron* a 3D-polyhedron is meant. In order to distinguish the faces of the environment and the faces of the k -reliable coverage, the faces of the environment are called *polygons* henceforth.

In Section 2.3.1, we will prove that the shape of the k -reliable coverage is a polyhedral area. The faces and vertices of a polyhedral area are necessary to calculate its volume (Section 2.3.2). This is why the faces and vertices of the k -reliable coverage are all classified in Section 2.3.3. For deriving some properties of the objective function, such as stair-casing and differentiability (Section 2.4), the parameters of the camera network are relevant at which a face of the k -reliable coverage meets a vertex of the coverage or, in general, a point in the environment. These incidences are characterized in Section 2.3.4.

2.3.1 Shape

We will realize in this section that the k -reliable coverage $C(k, S)$ of several cameras with labels $S = \{\textit{detectable}\}$ (field of view of several cameras) or $S = \{\textit{identical}\}$ (error of approximation) is a polyhedral area of some sort if the surveillance area is a polyhedral area. In the course of the proof within this section, we also motivate the types of faces that can occur in the boundary of the k -reliable coverage. This is relevant for classifying the faces in the next section.

In order to deduce the shape of $C(k, S)$, the following Lemma is necessary which is a direct result of Definition 2.1.1.

Lemma 2.3.1

An intersection or union of polyhedral areas is again a polyhedral area.

Let us first tend to one camera $N = 1$ and only consider the separation into detectable and undetectable parts of the surveillance area. Later we will add the distinction between changed and identical regions and add several cameras. The coverage of one camera with parameters $a = (p, o) \in \mathbb{P}$ holds

$$\sigma_{(E,a)}^{-1}(\{\textit{detectable}\}) = \mathbb{V}(p, E) \cap \mathbb{F}_a \cap \mathbb{A}.$$

The coverage of one camera can only be a polyhedral area if the frustum \mathbb{F}_a and the set of all visible points $\mathbb{V}(p, E)$ are polyhedral areas. Figure 2.12 (left) illustrates the field of view of one single camera as a polyhedral area. For illustration purposes, the figure is limited to two dimensions although we are investigating the field of view of a camera in three dimensions.

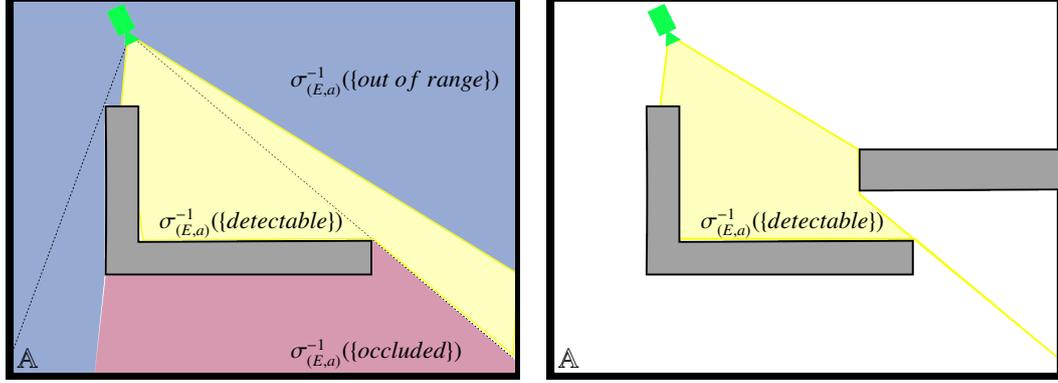


Figure 2.12: Comparison between different fields of view (yellow) of one single camera (green); Illustrated is the 2D case although the investigation considers the 3D case. Left: The field of view is a polygon since it includes only 2-manifold points. Right: The field of view is not a polygon due to the yellow ray in the bottom right corner, it is a polyhedral area.

Lemma 2.3.2 shows that the frustum \mathbb{F}_a is a polyhedron, but this cannot be said for the visible set $\mathbb{V}(p, E)$ with respect to all the camera parameters. The definition of a polyhedral area corresponds to a weakened polyhedron: The polyhedral area can be pinched and disconnected. Lemma 2.3.3 shows that the visible set is a polyhedral area. A field of view which is not manifold can be seen in Figure 2.12 (right).

Lemma 2.3.2

Let $a = (p, o) \in \mathbb{P}$ with $p \in E \setminus \partial E$. The shape of \mathbb{F}_a is a 3D-polyhedron.

Proof. The frustum \mathbb{F}_a is a 3D-polyhedron: A θ -space is an intersection of halfspaces. Since the opening angles are larger than 0, the frustum is a 3-manifold with boundary and thereby in \mathbb{R}^3 . Its boundary is illustrated in Figure 2.2. \square

The exact definition of the faces of one single camera's coverage follows in the next section. As anticipation: The boundary of the frustum is made out of four faces that are unbounded. We call these faces *opening faces* and two of them are illustrated in Figure 2.15 (left, purple lines). The faces are illustrated in 2D but they are relevant to this work in 3D. The next lemma addresses the shape of the visible set.

Lemma 2.3.3

Let E be an environment, $\mathbb{A} \subset E$ a surveillance area, and let $p \in E$ be the position of a camera. The shape of $\mathbb{V}(p, E)$ is a simply-connected 3D-polyhedral area.

Proof. For each segment $[p, y]$ from position p to a point $y \in \mathbb{V}(p, E)$ including $y = p$ holds $[p, y] \subset E$. Thus, the set $\mathbb{V}(p, E)$ is star-shaped and thereby simply connected. The visible set can be decomposed:

$$\mathbb{V}(p, E) = E \cap \complement \{y \in \mathbb{R}^3 \mid y \text{ not visible in } E \text{ from } p\} \quad (2.5)$$

Based on this Equation and the reasoning illustrated in Figure 2.13, $\mathbb{V}(p, E)$ is closed.

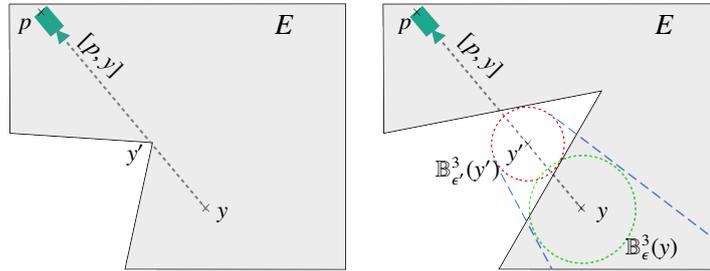


Figure 2.13: Reason for $\mathbb{V}(p, E)$ being closed. Let $y \in \partial\mathbb{V}(p, E)$. With Equation (2.5) holds: If $y \notin \mathbb{V}(p, E)$, then either $y \notin E$ and/or (as illustrated to the left) a point $y' \in [p, y]$ exists with $y' \notin E$ (Definition 2.1.4). But E is closed, so $y \in E$.

Furthermore, \mathring{E} is open, which leads to the fact that an open ball $\mathbb{B}_\epsilon^3(y') \subset \mathring{E}$ exists, as shown to the right (red). All points $y'' \in E$ with $[p, y''] \cap \mathbb{B}_\epsilon^3(y') \neq \emptyset$, lie on a cone behind the ball (blue) and hold $y'' \notin \mathbb{V}(p, E)$. y is at the center of the cone, just as y' is. Thus, an open ball $\mathbb{B}_\epsilon^3(y) \subset \mathring{\mathbb{V}}(p, E)$ exists (green), which means $y \notin \partial\mathbb{V}(p, E)$. This is a contradiction.

$\mathbb{V}(p, E) \subset E \subset \mathbb{R}^3$, but $\mathbb{V}(p, E)$ can be non-manifold (Counter example in Figure 2.12). However, a 3D-polyhedron which infringes the manifold property is a 3D-polyhedral area, thus the only property to be shown is that $\mathbb{V}(p, E)$ is a union of intersections of half spaces. The boundary of E is made of hyperplanes, the remaining boundary points $y \in \partial\mathbb{V}(p, E)$ are in $\partial\{y \in \mathbb{R}^3 \mid y \text{ not visible in } E \text{ from } p\}$. Their line of sight always touches an edge y'' of the environment as Figure 2.14 exemplarily shows in 2D. In 3D: In case p is not in the same one-dimensional subspace as y'' , y'' and p define a hyperplane, which is the boundary of one half-space. Otherwise, the one-dimensional subspace of y'' is the intersection of two half-spaces.

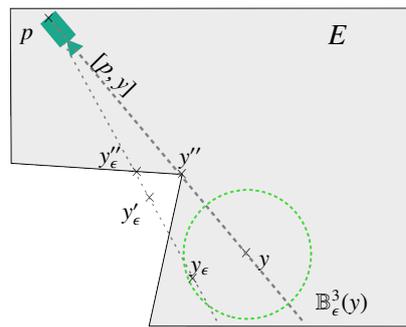


Figure 2.14: The line of sight of $y \in \partial\mathbb{V}(p, E) \setminus \partial E$ intersects an edge of the environment: The point holds $y \in \mathring{E}$ and for all $\epsilon > 0$ a point $y_\epsilon \in \mathbb{B}_\epsilon^3(y)$ exist with $y_\epsilon \notin \mathbb{V}(p, E)$. y_ϵ is not visible and, following this, a point $y'_\epsilon \in [p, y_\epsilon]$ exists with $y'_\epsilon \in \mathring{E}$. Since $p \in E$, a point $y''_\epsilon \in [p, y'_\epsilon]$ exists with $y''_\epsilon \in \partial E$. The sequence y''_ϵ lies on a plane and converges to a point $y'' \in [p, y]$ due to the theorem of intersecting lines. However, y is visible, and thus $[p, y] \subset E$. Thus, y' is at the boundary of a face of E . \square

What we have seen in this proof is that $\mathbb{V}(p, E)$ is a 3D-polyhedral area. A 3D-polyhedral area can be pinched, meaning it is not necessarily a 3-manifold. However, $\mathbb{V}(p, E)$ is at least simply connected. Furthermore, the polyhedral area has two types of faces: Firstly, faces that are defined by the polygons of the environment or surveillance area, and secondly, faces which separate occluded points of the environment from visible points. The influence on the faces of the identical coverage are again stated in the next section. There, the first type will be called *environmental face* and the latter *projection face*. Both are illustrated in Figure 2.15 (middle, blue and green lines). Again, the faces are illustrated in 2D but they are relevant to this work in 3D.

With \mathbb{F}_a and $\mathbb{V}(p, E)$ being polyhedral areas, the shape of the detectable region of a camera $\sigma_{(E,a)}^{-1}(\{\text{detectable}\})$ is a polyhedral area. Next, we will address the approximation of a target in the context of a single camera. What is the shape of the approximation error $\sigma_{(E,a)}^{-1}(\{\text{identical}\})$? With the Notation \mathbb{C} from Appendix B, the latter set can again be decomposed into

$$\sigma_{(E,a)}^{-1}(\{\text{identical}\}) = \sigma_{(E,a)}^{-1}(\{\text{detectable}\}) \cap \mathbb{C} \sigma_{(E,a)}^{-1}(\{\text{changed}\})$$

If $\sigma_{(E,a)}^{-1}(\{\text{changed}\})$ is a polyhedral area, the closed coverage $\mathbb{C} \sigma_{(E,a)}^{-1}(\{\text{identical}\})$ (Notation in Appendix B) is a polyhedral area as well. This is ensured in the following lemma:

Lemma 2.3.4

Let E be an environment, and let $a \in \mathbb{P}$. Let $E_d \subset \partial E$ be a dynamic object. Then, the shape of $\sigma_{(E,a)}^{-1}(\{\text{changed}\})$ is a polyhedral area.

Proof. Let \mathcal{I} and π be defined as in Definition 2.1.7. As described, all background pixels define a projected region that is identical to the environment at time t_0 and it holds:

$$y \in \sigma_{(E,a)}^{-1}(\{\text{identical}\}) \iff y \in \sigma_{(E,a)}^{-1}(\{\text{detectable}\}) \text{ and } x := \pi(y) \text{ is a background point}$$

Being a pinhole camera, the inclusion $[p, x] \cap \sigma_{(E,a)}^{-1}(\{\text{detectable}\}) \subset \sigma_{(E,a)}^{-1}(\{\text{identical}\})$ holds for all background points $x \in \mathcal{I}$. Consequently, $[p, x] \cap \sigma_{(E,a)}^{-1}(\{\text{detectable}\}) \subset \sigma_{(E,a)}^{-1}(\{\text{changed}\})$ for all foreground points $x \in \mathcal{I}$.

The foreground points are projections of the dynamic objects into the image plane. The dynamic faces form a polyhedral area $E_d \subset \partial E$, cf. Definition 2.1.6. Thus, the foreground points of the dynamic objects form a 2D-polyhedral area in silhouette image, as it is called in 2.1.2. The set $R := \{[p, x] \subset E \mid x \in \mathcal{I} \text{ is foreground point}\}$ is therefore a 3D-polyhedral area, and the set $R \cap \sigma_{(E,a)}^{-1}(\{\text{detectable}\}) \equiv \sigma_{(E,a)}^{-1}(\{\text{changed}\})$ as well. \square

What we have seen in this proof is that $\sigma_{(E,a)}^{-1}(\{\text{changed}\})$ is a 3D-polyhedral area. It can be pinched, it is not necessarily a 3-manifold. The faces of the coverage $\sigma_{(E,a)}^{-1}(\{\text{identical}\})$ that separate the changed and identical regions are called *silhouette faces*, further on. This type of face is illustrated in Figure 2.15 in 2D (right, orange lines), again with relevancy to the 3D case.

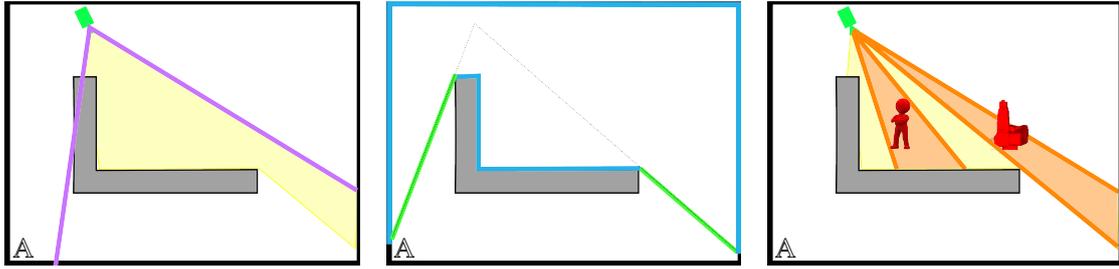


Figure 2.15: Example of Figure 2.6: Two-dimensional illustration of the four types of faces of the coverage $\sigma_{(E,a)}^{-1}(\{identical\})$ consisting of three types which are also faces of the field of view of the camera (first two pictures) and one additional type.

Left: The *opening faces* (purple lines) are faces of the boundary of the frustum of the field of view (yellow area). Middle: The *environmental faces* are parts of the polygons of the environment or surveillance area (blue lines). *Projection faces* separate occluded points of the environment and visible points (green lines). Right: *Silhouette faces* are faces of the coverage $\sigma_{(E,a)}^{-1}(\{identical\})$ that separate the changed and identical regions.

With this consideration we have gained the fact that the shape of the identical coverage $\sigma_{(E,a)}^{-1}(\{identical\})$ and the shape of the field of view is a polyhedral area. Next, we will consider a camera network with several cameras.

Theorem 2.3.5

Let E be an environment and let $E_s, E_d \subset E$ be polyhedral areas. Let $N, k \in \mathbb{N}$ and $k \leq N$. Let us utilize a camera network with N cameras and a 3DBGS as in Section 2.1.2, 2.1.3, and 2.1.4. And let $x := (a_1, \dots, a_N) \in \mathbb{P}_1 \times \dots \times \mathbb{P}_N$.

The identical ${}^c C_x(k, \{identical\})$ and the detectable coverage $C_x(k, \{detectable\})$ are both polyhedral areas.

Proof. By Definition 2.1.11 it holds $C_x(k, S) = \bigcup_{\pi \in C_N(k)} \bigcap_{n=1}^k \sigma_{a_{\pi(n)}}^{-1}(S)$ for a set $S \subset \mathbb{S}$ such as $S = \{detectable\}$ or $S = \{identical\}$. This is a concatenation in the sense of the polyhedral area of Definition 2.1.1 of polyhedral areas (Lemmas 2.3.2– 2.3.4). \square

With the above result the objective function of camera placement in this thesis is the volume of a polyhedral area. This is true for the objective function when maximizing the field of view of several cameras as well as minimizing the error of a human silhouette. The measurement of the volume of a polyhedral area will be discussed in the next section.

2.3.2 Volume

In order to investigate the properties (Section 2.4) of the objective function of sensor network optimization a measure of the k -reliable coverage must be chosen. In the last section, we have found that the shape of the k -reliable coverage is a polyhedral area. The volume of a polyhedral area is suitable for measuring both the identical and detectable coverage. In this section, vertices of a polyhedral area are used to obtain a formula for the volume of the k -reliable coverage.

There are several suitable representations for a polyhedral area: A computationally more convenient representation to check whether a point is inside a non-convex polyhedron is developed by [44], but for the moment we are more interested in a formula of the volume to examine the objective function's properties such as continuity or stair-casing.

In [160] a formula is given calculating the volume λ of a tetrahedron $[\mathbf{0}, T]$ which spans from $\mathbf{0} \in \mathbb{R}^3$ to a triangle T of the vertices $\mathcal{V}_1, \mathcal{V}_r$, and \mathcal{V}_{r+1} in 3D:

$$\lambda([\mathbf{0}, T]) = \frac{1}{6} \cdot \left((\mathcal{V}_1 - \mathbf{0}) \times (\mathcal{V}_r - \mathbf{0}) \right) \cdot (\mathcal{V}_{r+1} - \mathbf{0}) \quad (2.6)$$

In [83] it was shown that any convex or non-convex polygon \mathcal{F} with $R \in \mathbb{N}$ vertices $\mathcal{V}_1, \dots, \mathcal{V}_R$ can be separated into $(R - 2)$ triangles $T_r = (\mathcal{V}_1, \mathcal{V}_r, \mathcal{V}_{(r+1)})$, $r \neq 1, (r + 1) \neq 1$ if the segment $[\mathcal{V}_r, \mathcal{V}_{r+1}]$ forms a directed edge of the polygon. The spanning tetrahedron $[\mathbf{0}, T_r]$ of such a triangle can be calculated as above in 3D. As suggested in [160], the sum of the volumes of all these spanning tetrahedra is the volume of a pyramid $[\mathbf{0}, \mathcal{F}]$ spanning between $\mathbf{0}$ and \mathcal{F} .

Furthermore, [160] states that the volume of a polyhedron \mathcal{P} with $I \in \mathbb{N}$ faces \mathcal{F}_i can be calculated by the volumes of the spanning pyramids of all faces. In particular, the volumes of the spanning triangles with $R^{(i)} \in \mathbb{N}$ vertices $(\mathcal{V}_1^{(i)}, \mathcal{V}_r^{(i)}, \mathcal{V}_{(r+1)}^{(i)})$ of face \mathcal{F}_i are all added to a single volume:

$$\lambda(\mathcal{P}) = \sum_{i=1}^I \lambda([\mathbf{0}, \mathcal{F}_i]) = \frac{1}{6} \cdot \sum_{i=1}^I \sum_{r=2}^{R^{(i)}-1} \left((\mathcal{V}_1^{(i)} - \mathbf{0}) \times (\mathcal{V}_r^{(i)} - \mathbf{0}) \right) \cdot (\mathcal{V}_{r+1}^{(i)} - \mathbf{0}) \quad (2.7)$$

This procedure also comes to a valid result if the origin of the coordinate system does not lie inside the polyhedron: The volume of a tetrahedron spanned by a triangle in space in Equation (2.6) is signed: it is positive if the normal of the triangle points away from the origin $\mathbf{0}$. The normal of a triangle is directed away from the origin if the vertices are numbered clockwise when viewed from the origin. The normal of a triangle is the normal of the face. By such a signing the volume of a pyramid $\lambda(\mathbf{0}, \mathcal{F})$ spanned by a face \mathcal{F} is added if the normal of this face points away from the origin and subtracted if the normal points towards the origin. Since the normal of a face usually points to the exterior of a polyhedron, this procedure is valid for all kind of polyhedra.

By using the divergence theorem, [83] proves a similar formula in nD for a non-convex polyhedron given by a boundary representation. His idea is introducing a sign function $\text{sgn}(\mathcal{F})$ of a face that tells whether the volume of this particular tetrahedron $\lambda([\mathbf{0}, \mathcal{F}])$ is added to or subtracted from the overall volume of the polyhedron $\lambda(\mathcal{P})$. Since his proof is based on the divergence theorem (also Gauß's theorem) it is clear that the formula holds for bounded polyhedral areas, as we have introduced them in Definition 2.1.1, as well, if the complete boundary is given without two-dimensional holes. In particular, this means that a polyhedral area as in Figure 2.12 (right) has at least two faces in non-manifold points directed in opposite directions.

With Formula (2.7) the volume of the coverage, our objective function, can be expressed by the vertices and faces of the coverage. We will consider the types of vertices and faces next.

2.3.3 Faces and Vertices

As proved in the Theorem 2.3.5, the shape of the fused coverage is a polyhedral area. A polyhedral area is bounded by faces and its volume can be computed by its faces and vertices. In order to investigate

the properties of the volume of the fused coverage, we will therefore investigate the types of faces and vertices in this section. Within this section, remember the notation ∂ for the boundary, c for the closure, and \complement for the complement of a set (Appendix B).

The faces of the fused coverage $C_x(k, S)$ for a given set $S \subset \mathbb{S}$ depend on the faces of the coverage of one camera, since:

$$\partial C_x(k, S) = \partial \left(\bigcup_{o=1}^O \bigcap_{n=1}^k \sigma_{(E, \pi_o(a_n))}^{-1}(S) \right) \subset \bigcup_{o=1}^O \bigcap_{n=1}^k \partial \left(\sigma_{(E, \pi_o(a_n))}^{-1}(S) \right) \quad (2.8)$$

With this information the faces of the fused coverage can be reduced to the faces of the coverage of one camera. Our interest lies in the faces of the identical and detectable coverage, $^c \sigma_{(E,a)}^{-1}(\{\text{identical}\})$ and $\sigma_{(E,a)}^{-1}(\{\text{detectable}\})$ of one single camera, illustrated in Figure 2.15 for the 2D case. These faces in 2D or 3D are classified in Definition 2.3.6.

Definition 2.3.6

Let E be an environment, let $\mathbb{A} \subset E$ be a polyhedral area, let $a \in \mathbb{P}$, and let $\mathcal{F} \subset E$ be a face of the polyhedral area $\sigma_{(E,a)}^{-1}(\{\text{identical}\})$ or $\sigma_{(E,a)}^{-1}(\{\text{detectable}\})$. It is called:

(\mathcal{F}_E) *Environmental face* if it is part of the boundary of the environment or surveillance area: $\mathcal{F} \subset \partial E \cup \partial \mathbb{A}$

(\mathcal{F}_P) *Projection face* if it is part of the boundary of the set of visible points and not part of the boundary of the environment: $\mathcal{F} \subset \partial \mathbb{V}(p, E) \cap \complement \partial E$.

(\mathcal{F}_O) *Opening face* if it is part of the boundary of the camera frustum $\mathcal{F} \subset \partial \mathbb{F}_a$

(\mathcal{F}_S) *Silhouette face* if it is part of the boundary between changed and identical parts $\mathcal{F} \subset \partial \sigma_{(E,a)}^{-1}(\{\text{changed}\}) \cap \partial \sigma_{(E,a)}^{-1}(\{\text{identical}\})$. This type of face does not occur in the detectable coverage $\partial \sigma_{(E,a)}^{-1}(\{\text{detectable}\})$.

These are the faces of the identical and the detectable coverage of one camera. Hereby, the affine subspace of a face \mathcal{F}_E is camera independent whereas the affine subspaces of the faces \mathcal{F}_P , \mathcal{F}_O , and \mathcal{F}_S are camera specific, they change with the camera parameters. The following lemma states that the plane of projection, silhouette, and opening faces directly depends on the position of the camera, and the first two additionally on the environment:

Lemma 2.3.7

Let E be an environment, let $a = (p, o) \in \mathbb{P}$, and let $\mathcal{F} \subset E$ be a face of the polyhedral area $^c \sigma_{(E,a)}^{-1}(\{\text{identical}\})$. Let $A \subset \mathbb{R}^3$ be a 2-dimensional affine subspace with $\mathcal{F} \subset A$.

Then, the position of the camera holds $p \in A$ if \mathcal{F} is a projection face, opening face, or silhouette face.

If \mathcal{F} is a projection face: An edge $\mathcal{E} \subset \partial E$ of the static objects exists with $\mathcal{E} \cap \mathcal{F} \neq \emptyset$.

If \mathcal{F} is a silhouette face: An edge $\mathcal{E} \in \partial E$ in the dynamic objects exists with $\mathcal{E} \cap \mathcal{F} \neq \emptyset$.

We will call \mathcal{E} an *anchor*.

Proof. Opening faces: The frustum of the camera is defined by two θ -spaces in Definition 2.1.4. $\mathbb{F}_a := \{y \in E \mid (y - p) \in (T_o(\theta_u, u) \cap T_o(\theta_{o \times u}, (o \times u)))\}$ of the opening angles $\theta_u, \theta_{o \times u} \in [0, \pi]$ and the view-up vector $u \in \partial \mathbb{B}_1^3(0)$. By definition the position is element of the faces: $p \in \partial \mathbb{F}$.

Projection faces: The boundary of the polyhedral area $\mathbb{V}(p, E)$ separates the visible and not visible points which are included in the environment E . Visible points lie in front of all polygons of the static objects of environment from camera point of view. And points that are not visible are occluded by at least one polygon. Thus, a boundary point $y \in \partial \{z \in E \mid z \text{ visible from } p\} \cap \partial \{z \in E \mid z \text{ not visible from } p\}$ is on a ray defined by the position of the camera p and the boundary of such an obscuring polygon. The face of $\mathbb{V}(p, E)$ is then defined by the points that lie on this ray behind the polygon from the camera's point of view. Thereby, the position of the camera is part of this ray which is again part of the plane where the face is on.

Silhouette faces correspond to the projection faces on dynamic objects: The boundary of the silhouette of a dynamic object in the image plane $I \subset E$ is made of edges. These edges are projections from the edges of dynamic objects into the image plane. Therefore, the faces of the coverage $\sigma_{(E,a)}^{-1}(\{\text{identical}\})$ that separate the changed and identical regions are on planes defined by an edge of the dynamic object and the position of the camera p . \square

The name *anchor* is taken from the publication [58] which considers the visibility in the two-dimensional case (the environment is a polygon) with some differences: The authors do not classify vertex and face types since only two of each exist in the 2D case. An anchor in their sense is a vertex \mathcal{V}_a of the environment projected from the camera position p onto a face of the environment. This projection results in a vertex \mathcal{V}_f of the field of view. When moving p the vertex \mathcal{V}_f encounters a second vertex of the environment \mathcal{V}_e . The anchor of [58] is defined by this incidence. In the three-dimensional case, however, we do not use the incidence in the definition of the anchor. Here, an anchor is an edge \mathcal{E} of the environment (an edge in 3D corresponds to a vertex \mathcal{V}_a in 2D) projected from the camera position p onto a face of the environment. The incidence between the projected anchor and an additional vertex is described in the following paragraphs.

The authors of [58] make another interesting observation: The anchor needs to be *reflex*, which means that the angle inside the environment between the vertex' edges is strictly greater than π . In the three-dimensional case, the anchor of a projection or silhouette face (which is an edge) needs to be reflex in some points, as well. But in the upcoming sections, this property is merely additional information and not relevant for further investigations.

From Inclusion (2.8) we know that the type of faces of the k -reliable coverage corresponds to the above types of faces of the coverage of one camera. Say, the k -reliable coverage is a concatenation of the coverages of $N \in \mathbb{N}$ cameras. Then, its boundary consists of the environmental faces $\mathcal{F}_E^{(n)}$, projection faces $\mathcal{F}_P^{(n)}$, opening faces $\mathcal{F}_O^{(n)}$, and silhouette faces $\mathcal{F}_S^{(n)}$ of the camera coverages $\sigma_{(E,a_n)}^{-1}(\{\text{identical}\})$ for the camera parameters a_n belonging to one of the cameras $n = 1, \dots, N$. The affine subspaces of the faces $\mathcal{F}_P^{(n)}$, $\mathcal{F}_O^{(n)}$, and $\mathcal{F}_S^{(n)}$ change with the camera parameters $a_n \in \mathbb{P}_n$. The affine subspace of $\mathcal{F}_E^{(n)}$ is purely given by the environment and surveillance area and is constant for all the cameras.

The k -reliable coverage is a polyhedral area. A vertex of the coverage of cameras utilizing a background subtraction method as above is an intersection of at least three faces of the polyhedral area, independently on how many cameras are utilized in the network. A vertex can be an intersection of more than three

faces, but there are at least three faces. Thus, a vertex can be classified by a 3-tuple consisting of the faces $\mathcal{F}_E^{(n)}$, $\mathcal{F}_P^{(n)}$, $\mathcal{F}_O^{(n)}$, and $\mathcal{F}_S^{(n)}$. At the same time, the plane of a face $\mathcal{F}_E^{(n)}$ is camera independent and the planes of the faces $\mathcal{F}_P^{(n)}$, $\mathcal{F}_O^{(n)}$, and $\mathcal{F}_S^{(n)}$ depend on the camera parameters a_n .

We list the vertex types (3-tuples) in Figure 2.16 regarding the type of the faces and the number of cameras a vertex is defined by. The 3-tuple consists of three faces of the coverages of N cameras, but we have only distinguished between three cameras, since three faces can only be taken from three separate cameras. The total number of 3-tuples is therefore $(4 \cdot N)^3$ but most of the 3-tuples can be neglected for one of the following reasons:

Rule 1 The faces \mathcal{F}_P , \mathcal{F}_O , and \mathcal{F}_S of the same camera have a common point, the position of the camera p , cf. Lemma 2.3.7. Thus, an intersection of only these faces of the same camera does not result in another vertex of the field of view except p . In the below figure, the positions of the cameras have not been illustrated as vertices.

Rule 2 Two 3-tuples are the same if they include permuted nodes.

Rule 3 Two 3-tuples are the same if all faces of one camera in the first tuple can be mapped upon faces of the same type but different camera in the second tuple, as long as the diversity of cameras remains the same. This is due to the fact that all cameras have the same parameters.

The resulting 3-tuples can be viewed in the tree of Figure 2.16. The dependency of \mathcal{F}_S , \mathcal{F}_O and \mathcal{F}_P on the first camera is illustrated as green circle, on the second camera as red, and on the third camera as blue. The tree has already been pruned by the above rules in depth (from bottom to top): Branches made of one color except gray have already been left out (Rule 1). Also if we assume the first subtree ($\mathcal{F}_E, ?, ?$) built and have a look at the choices for the subtree ($\mathcal{F}_P, \mathcal{F}_E, ?$), you will notice that all the tuples have already been covered by ($\mathcal{F}_E, \mathcal{F}_P, ?$) in the first subtree (Rule 2). Notice further, if branch ($\mathcal{F}_E, \mathcal{F}_E, \mathcal{F}_P$) has already been built with \mathcal{F}_P in green then the same branch does not need to be included with \mathcal{F}_P in red or blue (Rule 3).

2.3.4 Incidences due to Variable Camera Parameters

In Section 2.3.1 we have seen that the k -reliable coverage is a polyhedral area, the volume of which can be calculated by the coordinates of its vertices as seen in Section 2.3.2. In the context of camera placement, the volume depends on variable camera parameters. Therefore, the faces and vertices of the k -reliable coverage are discussed as functions of the camera parameters, in this section. The incidence when a face of the k -reliable coverage meets a vertex or voxel is particularly interesting for the differentiability or staircasing of the volume of the coverage. Again, we first classify these incidences for the parameters of a single camera and then turn to the parameters of a camera network. Let us start with a particular subset of the vertices and faces that have been classified in Section 2.3.3:

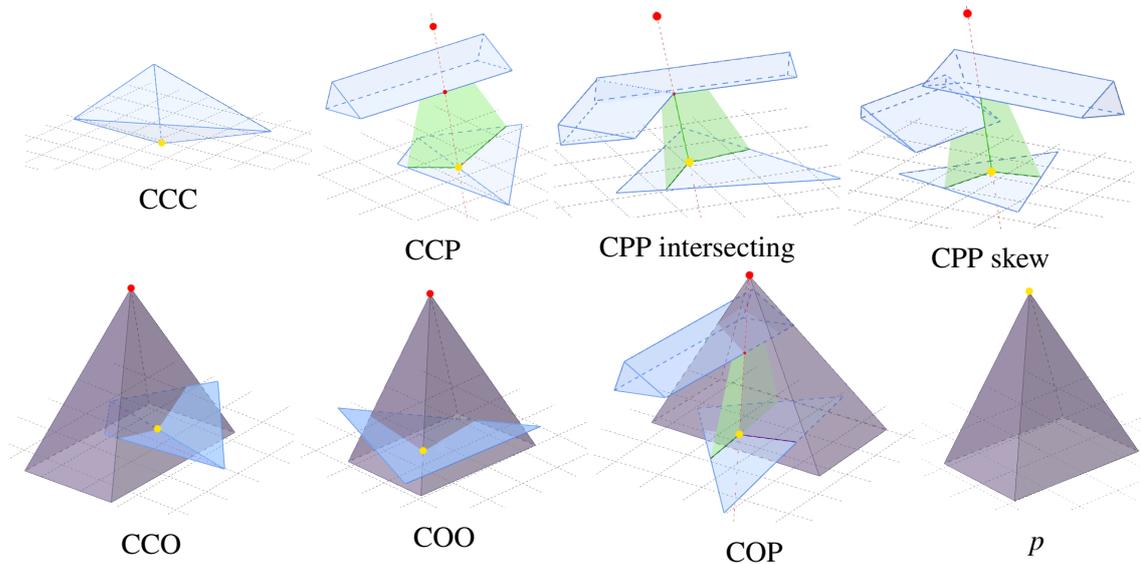


Figure 2.17: Vertex types (yellow dot) when moving the camera position (red dot) in three dimensions: Silhouette faces and projection faces (P), that adopt a similar behavior, are illustrated in green, including their anchor (blue wedge). Opening faces (O) are purple, and constant faces (C) blue. Only the combinations CCC, CCP, CPP, CCO, COO, and COP are possible as vertex types next to the position of the camera p . The two anchors of a CPP can be skew or intersecting.

Let us assume that all cameras are fixed except one, w.l.o.g. the first one with parameter vector a_1 in $(a_1, \dots, a_N) = x \in \mathbb{P}_1 \times \dots \times \mathbb{P}_N$. Thus, some faces exist whose planes (or affine subspaces in which they are situated) are constant (C): all environmental faces and the faces of the coverage of a camera with a parameter vector a_n with $n \neq 1, n \leq N$. Since only projection, silhouette, and opening faces of the coverage of camera 1 are changed, the changed faces' subspace must include the position p of the parameter vector $a_1 = (p, o)$, cf. Lemma 2.3.7. All moving vertices except p must therefore be an intersection of at least one constant face (C) and two (??) other faces, denoted by (C??). The behavior of silhouette faces corresponds to the behavior of projection faces (P) when changing the camera's parameters. Thus, only the combinations CCC, CCP, CPP, CCO, COO, and COP are possible with (O) denoting an opening face of the moving camera a_1 . In case of the vertex type CPP, two projection/silhouette faces and one constant face are intersected. Their anchors can be skew or intersecting.

When considering the gray nodes of Figure 2.16 as constant faces instead of just environmental faces, then the vertices of one moving camera correspond to the green and gray leaves in the first (lower) subtree. For a better understanding, Figure 2.17 illustrates the types of vertices in three dimensions. Therefore, only the combinations with constant faces (blue), projection/silhouette faces (green), and opening faces (purple) are provided.

In Section 2.4.1 and 2.4.4, the incidence of a vertex or voxel with a face of the coverage will prove to be a key event when moving the camera in 3D. Therefore, let us consider the parameters of a camera by which the vertex \mathcal{V} meets the face \mathcal{F} .

Definition 2.3.8

Let E be an environment, $k \in \mathbb{N}$, $S \subset \mathbb{S}$ and let $\mathcal{F} \subset {}^{\partial}C_x(k, S)$ be a face of the coverage.

- The set $I(\mathcal{V}, \mathcal{F}) := \{x \in \mathbb{P}_1 \times \dots \times \mathbb{P}_N \mid \mathcal{V} \in \mathcal{F}\}$ is called *incidence surface* of a point $\mathcal{V} \in E$.
- It is called *vertex incidence surface* if $\mathcal{V} \in {}^{\partial}C_x(k, S)$ is a vertex with $\mathcal{V} \notin \mathcal{F}$ for some $x \in \mathbb{P}_1 \times \dots \times \mathbb{P}_N$.
- The set $\mathbb{I} = \{I(\mathcal{V}, \mathcal{F}) \mid \mathcal{V} \text{ vertex and } \mathcal{F} \text{ face of } C_x(k, S) \text{ and } \exists x \in \mathbb{P}_1 \times \dots \times \mathbb{P}_N : \mathcal{V} \notin \mathcal{F}\}$ is called the *set of all vertex incidences*.

The authors of [58] call a vertex incidence surface in two-dimensions an *inflection segment*, but only one type of segment exists in 2D. In three-dimensional space the incidence surface is rarely only a segment. In order to illustrate the variety of these surfaces, have a look at the positions p of the camera where a vertex \mathcal{V} of Figure 2.17 meets an additional face \mathcal{F} of the k -reliable coverage with constant orientation and constant opening angles. A vertex \mathcal{V} meets a ...

C Constant face, if it meets the edge \mathcal{E} between \mathcal{F} and the original constant face.

P Projection or silhouette face, if the line of sight $[p, \mathcal{V}]$ meets the anchor \mathcal{E} of the (additional) projection/silhouette face, as defined in Definition 2.3.7.

O Opening face, if the line of sight is subset of the (additional) opening face \mathcal{F} .

The following Tables 2.3, 2.4, and 2.5 show the positions p of the camera where a vertex \mathcal{V} meets such a \mathcal{F} with constant orientation and opening angles. In all tables, the vertex types are depicted in the first column. In the second column, the set $I(\mathcal{V}, \mathcal{F})$ is characterized which includes all positions p where an incidence between \mathcal{V} and \mathcal{F} happens. The third column illustrates the vertex (red) together with the movement of p to a point where \mathcal{V} will definitely meet the additional face (red arrow), and the characterized vertex incidence surface (orange). The position p of the dynamic camera is listed in neither of the above mentioned tables as a vertex type. The reason to leave out the position as a vertex, is that the incidence surface where this vertex meets an additional face \mathcal{F} simply equals the face \mathcal{F} .

| Type | With the movement of p , \mathcal{V} meets an additional constant face \mathcal{F} if | Illustration of the surface |
|------------------|---|-----------------------------|
| CCP | ... it meets one of the <i>two</i> vertices $\overline{\mathcal{V}}$ of the edge at which it is situated. All the camera positions p where \mathcal{V} has encountered $\overline{\mathcal{V}}$ form a plane segment including p and the anchor. | |
| CPP in-ter-sect. | ... it meets the edge \mathcal{E} between the original and the new constant face. This is true if p trespasses a plane formed by the intersection point of both the anchors and \mathcal{E} . | |
| CPP | ... the line of sight $[p, \mathcal{V}]$ intersects both anchors and the edge \mathcal{E} between the original and new constant face. If \mathcal{E} is parallel to one of the anchors, then the surface including p is a plane including these edges. If the direction of \mathcal{E} is linearly dependent on the directions of the anchors, then the three lines lie on parallel planes. Thus, the surface is a hyperbolic paraboloid , cf. [30]. If this is not the case, then the surface is a hyperboloid of one sheet since the lines are skew [30]. | |
| CCO | ... it meets the starting or endpoint $\overline{\mathcal{V}}$ of the edge it is situated at. Additionally the line of sight $[p, \mathcal{V}]$ is at the opening face \mathcal{F}_O . Thus, all the viewpoints p where $\mathcal{V} = p$ are at a plane parallel to \mathcal{F}_O including p . | |
| COO | ... the line of sight $[p, \mathcal{V}]$ intersects the edge \mathcal{E} between the constant faces (old and new). The line of sight is subset of both the opening faces \mathcal{F}_{O1} and \mathcal{F}_{O2} . Thus, the surface of positions where \mathcal{V} meets \mathcal{E} is a plane parallel to the line of sight $[p, \mathcal{V}]$ including \mathcal{E} . | |
| CPO | ... the line of sight $[p, \mathcal{V}]$ intersects the anchor and the edge \mathcal{E} between old and new constant face. The line of sight $[p, \mathcal{V}]$ is included in the opening face \mathcal{F}_O . Varying the p such that the line of sight meets the above criteria induces three varying line of sights $[p_1, \mathcal{V}_1], [p_2, \mathcal{V}_2], [p_3, \mathcal{V}_3]$. These can either all be skew, or all be parallel. In case of all parallel, then the surface that all p_r , $r = 1, 2, 3$ are on is a plane . In case skew line of sights, consider that all these lines are situated at parallel planes, thus the surface is a hyperbolic paraboloid , cf. [30]. | |

Table 2.3: Characterization of vertex incidence surfaces $I(\mathcal{V}, \mathcal{F})$ between a \mathcal{V} and an additional **constant** face \mathcal{F} . Left: Vertex type; Middle: Characterization (bold) and proof; Right: Illustration of vertex (yellow dot), line of sight (red dashed), and vertex incidence surface (orange).

| Type | With the movement of p , \mathcal{V} meets an additional projection face \mathcal{F} if | Illustration of surface |
|------|---|-------------------------|
| CCC | Compare the vertex type CCP in Table 2.3. | |
| CCP | Compare the vertex type CPP (skew or intersecting) in Table 2.3. | |
| CPP | <p>... the line of sight $[p, \mathcal{V}]$ intersects an anchor \mathcal{E} in addition to the anchors $\bar{\mathcal{E}}$ and $\underline{\mathcal{E}}$ of the previous projection faces. If the anchors $\bar{\mathcal{E}}$ and $\underline{\mathcal{E}}$ are <i>intersecting</i> then the intersection point $\bar{\mathcal{E}} \cap \underline{\mathcal{E}}$ and the additional anchor \mathcal{E} define the vertex incidence surface, it is a plane.</p> <p>If the previous anchors are <i>skew</i> then the vertex incidence surface can be constructed as follows: If \mathcal{E} is parallel to $\bar{\mathcal{E}}$ or $\underline{\mathcal{E}}$, then the surface is a plane including \mathcal{E} and the parallel edge. If the direction of \mathcal{E} is linearly dependent on the directions of $\bar{\mathcal{E}}$ and $\underline{\mathcal{E}}$, then the edges lie on parallel planes, so the surface is a hyperbolic paraboloid [30]. If this is not the case, then the surface is a hyperboloid of one sheet since the edges are skew [30].</p> | |
| CCO | Compare the vertex type CPO in Table 2.3. | |
| COO | <p>... the line of sight $[p, \mathcal{V}]$ intersects an anchor \mathcal{E} and is subset of two opening faces. The normals of the opening faces will not change, thus $[p_1, \mathcal{V}]$ is parallel to $[p_2, \mathcal{V}]$ for all viewpoints $p_1 \neq p_2 \in E$. The swept vertex incidence surface is a plane parallel to the line of sight, including the anchor \mathcal{E}.</p> | |
| CPO | <p>... the line of sight $[p, \mathcal{V}]$ intersects a previous anchor $\bar{\mathcal{E}}$, an additional anchor \mathcal{E} and is subset of an opening face \mathcal{F}_O. Varying the intersection $\bar{z} := \bar{\mathcal{E}} \cap [p, \mathcal{V}]$ induces three points $\bar{z}_1, \bar{z}_2, \bar{z}_3$. At each point the orientation and opening angles of the camera are fixed, so the three points induce three positions of the opening face: $\bar{z}_r \in \mathcal{F}_{O_r}, r = 1, 2, 3$. $\bar{\mathcal{E}}$ cannot be parallel to or subset of \mathcal{F}_O since $\bar{\mathcal{E}}$ is the anchor of the previous projection face. The additional anchor \mathcal{E} is therefore intersected in three points by the three opening faces: $z_r \in \mathcal{F}_{O_r} \cap \mathcal{E}$. The line of sight $[p, \mathcal{V}]$ always includes both points of the tuple (z_r, \bar{z}_r). In case of parallel line of sights the surface is a plane including the line of sights. In case of skew line of sights the surface is a hyperbolic paraboloid.</p> | |

Table 2.4: Characterization of vertex incidence surfaces $I(\mathcal{V}, \mathcal{F})$ between a vertex \mathcal{V} and an additional **projection** face \mathcal{F} . Left: Vertex type; Middle: Characterization (bold) and proof; Right: Illustration of vertex (yellow dot), line of sight (red dashed), and vertex incidence surface (orange).

| Type | With the movement of p , \mathcal{V} meets an additional opening face \mathcal{F} if | Illustration of surface |
|------|---|-------------------------|
| CCC | Compare the vertex type CCO (skew or intersecting) in Table 2.3. | |
| CCP | <p>... the line of sight $[p, \mathcal{V}]$ is subset of \mathcal{F} and intersects the anchor \mathcal{E} of the original projection face. Compare the vertex type CPO (skew or intersecting) in Table 2.3:</p> <p>Three positions p_r, $r = 1, 2, 3$, with varying $\mathcal{V}(p_r) \in \bar{\mathcal{E}}$ where $\bar{\mathcal{E}}$ is the edge between the constant faces, induce three varying opening faces $\mathcal{F}(p_r)$ (unless $\bar{\mathcal{E}}$ parallel to \mathcal{F}) and thus three varying line of sights $[p_r, \mathcal{V}(p_r)]$. In case of the lines of sight being parallel, the surface is a plane including the line of sights. Otherwise, consider that all line of sights are situated at parallel planes $\mathcal{F}(p_r)$, thus the surface is a hyperbolic paraboloid, [30].</p> | |
| CPP | <p>... the line of sight $[p, \mathcal{V}]$ is subset of \mathcal{F} and intersects both anchors $\bar{\mathcal{E}}$ and $\underline{\mathcal{E}}$. In the case of <i>intersecting anchors</i>, they intersect in a common point. Upon contact of face and vertex, the position p is on a plane parallel to \mathcal{F} including the point $\bar{\mathcal{E}} \cap \underline{\mathcal{E}}$.</p> <p>In case of <i>skew anchors</i> the line of sight intersects $\bar{\mathcal{E}}$ and $\underline{\mathcal{E}}$ in two nonidentical points. The surface is either a plane or a hyperbolic paraboloid, compare the vertex type CPO in Table 2.4.</p> | |
| CCO | <p>... the line of sight $[p, \mathcal{V}]$ is subset of both the opening faces, the previous one $\bar{\mathcal{F}}_O$ and the additional one \mathcal{F}_O, and intersects the edge \mathcal{E} between the constant faces. Thus p is included in the plane parallel to $\bar{\mathcal{F}}_O \cap \mathcal{F}_O$ including the edge \mathcal{E} upon contact.</p> | |
| CPO | <p>... the line of sight $[p, \mathcal{V}]$ is subset of both the opening faces, the previous one $\bar{\mathcal{F}}_O$ and the additional one \mathcal{F}_O, and intersects the anchor \mathcal{E} of the projection face. As for the vertex type COO in Table 2.4, the position p of the camera is in a plane parallel to the intersection $\bar{\mathcal{F}}_O \cap \mathcal{F}_O$ and including \mathcal{E} upon contact.</p> | |

Table 2.5: Characterization of vertex incidence surfaces $I(\mathcal{V}, \mathcal{F})$ between a vertex \mathcal{V} and an additional opening face \mathcal{F} . Left: Vertex type; Middle: Characterization (bold) and proof; Right: Illustration of vertex (yellow dot), line of sight (red dashed), and vertex incidence surface (orange).

Table 2.3 shows the positions p of the camera where a vertex \mathcal{V} encounters a constant face $\mathcal{F} \equiv C$. Notice that the vertex type CCC and an additional constant face do not collide; this case is therefore neglected. Table 2.4 and 2.5 show positions p where a vertex \mathcal{V} meets an (additional) projection face or opening face, respectively. In Table 2.5, the vertex type COO is discarded since the vertex is already an intersection of two opening faces and will not meet another opening face of the same camera.

The above tables show the vertex incidence surfaces considering only the position of one camera as a variable to the k -reliable coverage. Next, consider the orientation of one single camera as a variable. When leaving p constant, the silhouette and projection faces of all cameras are constant. The only dynamic vertex types are CCO and COO. It can be shown that the incidence surfaces of these types are hypersurfaces in \mathbb{P} , as well. When using the volume of the k -reliable coverage as an objective function, however, not only the position and orientation of one single camera but the positions and orientations of several cameras are changed. Let us consider the general case of a vertex $\mathcal{V} = \mathcal{F}_1 \cap \mathcal{F}_2 \cap \mathcal{F}_3$ meeting a fourth face \mathcal{F} independently on how many camera positions and orientations are varied:

Theorem 2.3.9

Let E be an environment, let $N, k \in \mathbb{N}$ with $k \leq N$, $x \neq \bar{x} \in \mathbb{P}_1 \times \dots \times \mathbb{P}_N$, $S \subset \mathbb{S}$, and let $\mathcal{F}_x \subset {}^{\partial}C_x(k, S)$ be a face of the k -reliable coverage. Let $\mathcal{V} \in {}^{\partial}C_x(k, S)$ be a vertex with $\mathcal{V} \in \mathcal{F}_x$ but $\mathcal{V} \notin \mathcal{F}_{\bar{x}}$. Let $I(\mathcal{V}, \mathcal{F}_x)$ be a vertex incidence surface.

The set $I(\mathcal{V}, \mathcal{F}_x)$ is a (Lebesgue)-null set in $\mathbb{P}_1 \times \dots \times \mathbb{P}_N$.

Proof. In general, the faces $\mathcal{F}_1, \mathcal{F}_2, \mathcal{F}_3$ defining a vertex have linearly independent normals. Let $I(\mathcal{V}, \mathcal{F}_x)$ be the set of sensor network parameters with $\mathcal{V} = \mathcal{F}_1 \cap \mathcal{F}_2 \cap \mathcal{F}_3 \cap \mathcal{F}_x$. This set is a parameterized surface in $\mathbb{P}_1 \times \dots \times \mathbb{P}_N$ with a dimension lower than $\dim(\mathbb{P}_1 \times \dots \times \mathbb{P}_N)$.

In case of the position of one camera $\mathcal{V} = p$, the vertex is dependent on the camera parameters, which leads to a parameterized set, so let $\mathcal{V} \neq p$. When adjusting the network parameters two cases exist for \mathcal{F}_x 's affine subspace: The subspace...

- ... is constant (e.g., in case of an environmental face). In this case at least one of the other faces' subspaces $\mathcal{F}_1, \dots, \mathcal{F}_3$ needs to be non-constant, since otherwise $\mathcal{V} \in \mathcal{F}_{\bar{x}}$. With \mathcal{V} in \mathcal{F}_x 's subspace, one component of \mathcal{V} can be linearly combined by the other components. But the components of \mathcal{V} are completely determined by $\mathcal{F}_1, \dots, \mathcal{F}_3$ and at least one of the faces $\mathcal{F}_1, \dots, \mathcal{F}_3$ is determined by camera parameters (not environmental). Thus, some components of the parameter vectors of the cameras are determined by the other components of the parameter vectors.
- ... includes the position of one of the cameras $p \in E$. \mathcal{F}_x can be one of the two face types:

Opening face With \mathcal{V} being in \mathcal{F}_x 's subspace, the line of sight $[\mathcal{V}, p]$ is included in this subspace, as well. When fixing \mathcal{V} then \mathcal{F} is determined by o and u of the same camera that p is taken from. So p depends on o and u .

Projection/Silhouette face When fixing \mathcal{V} , then \mathcal{F}_x is determined by \mathcal{V} and an anchor \mathcal{E} (Lemma 2.3.7). Then a point $y \in \mathcal{E}$ exists with $y \in [\mathcal{V}, p]$. Thus the position p is determined completely.

So, $I(\mathcal{V}, \mathcal{F}_x)$ is a parameterized set. The dependence of the variables makes $I(\mathcal{V}, \mathcal{F}_x)$ a null set in $\mathbb{P}_1 \times \dots \times \mathbb{P}_N$. □

The proofs in the Tables 2.5, 2.4, and 2.3, and the subsequent consideration show that the set $I(\mathcal{V}, \mathcal{F}_x)$ is a surface in E if $\mathbb{P} = E$. A surface is a null set in E . In order to incorporate the orientation and the parameters of several cameras as variables, the vertex incidence surfaces need to be considered for vertices listed in Figure 2.16 and the position of each camera. The incidence surface where the position of a camera as a vertex meets an additional face \mathcal{F}_x equals the surface $\mathcal{F}_x \times \partial\mathbb{B}_1^3(0)$ which is also a null set in $\mathbb{P} = E \times \partial\mathbb{B}_1^3(0)$. The sets $\mathbb{P} = E$ and $\mathcal{F}_x \times \partial\mathbb{B}_1^3(0)$ are two examples for the general case in Theorem 2.3.9.

With Theorem 2.3.9, we have shown that the vertex incidence surfaces are null sets in the space of network parameters. The theorem is the climax of a series of considerations about the shape of the coverage and its change when adjusting the camera network's parameters. This is particularly interesting for the next section, in which we depict the mathematical properties of the volume of the k -reliable coverage.

2.4 Properties of the Volume of the Coverage of Multiple Cameras

For most optimization methods, the properties of the objective function are crucial for convergence. As an objective function we want to utilize the volume of the identical or detectable coverage as discussed in Section 2.1. We have seen in Section 2.3 that the coverage we are interested in is a polyhedral area whose volume can be computed by its vertices and faces (Section 2.3.2). We have classified all the vertices and faces of the coverage of identical sensor labels and detectable sensor labels (Section 2.3.3) and showed where (at which sensor network parameters) the faces meet other vertices of the coverage and points of the environment (Section 2.3.4). The network parameters of these incidences are very rare in the network's parameter space: They form null sets in the network's parameter space which are called *incidence surfaces*, here.

The properties of the k -reliable coverage we have derived so far are of geometrical nature and will be helpful to deduce the properties of the volume of the coverage. In Section 2.4.1 the incidence surfaces of voxels are used to prove differentiability μ -almost everywhere. The objective function is non-convex which is shown in an example in Section 2.4.2. In Section 2.4.3 the fact is stated that the k -reliable coverage of a sensor network as well as the volume of the coverage is invariant with permuting the cameras. This property is called *symmetry*. And finally, the incidence surfaces of voxels are used to prove the stair-casing effect of the objective function in Section 2.4.4.

2.4.1 Continuity and Differentiability

In two dimensions, the authors of [58] prove that the volume of the field of view of one camera with unlimited opening angle in a (two-dimensional) polygonal environment is locally Lipschitz. Furthermore, the authors prove that the non-differentiable points of the volume lie on a set of lines in the polygon. When changing the camera position, a point within these lines is a camera position where two vertices of the field of view coincide. Neither a limited opening angle, nor several cameras, nor the three-dimensional case, nor a 3DBGS method have been considered in their publication, which will be covered in the following paragraphs.

With Equation (2.7), the volume is a polynomial of the components of the vertices. So, as long as these components are continuous or differentiable, the property is inherited by the volume. But the components of the vertices are neither always differentiable nor continuous. In this section, the continuity of vertices and faces is investigated. A *visual event* is a point of network parameters at which a vertex or face is either not differentiable or not continuous.

Let $\mathcal{V} \in C_x(k, S)$ be such a vertex of the coverage with a fixed $S \subset \mathbb{S}$, threshold $k \in \mathbb{N}$, and sensor network parameters $x = (a_1, \dots, a_N) \in \mathbb{P}_1 \times \dots \times \mathbb{P}_N$. It is an intersection of at least three faces classified in Section 2.3.3. A face is defined by a plane (two-dimensional affine subspace) and by its boundary (half-spaces), cf. Definition 2.1.1. When adjusting the parameters of a network the vertex moves and the planes and boundaries could change. Thus, visual events can be caused by the following issues:

Visual event 1 The dimension of the m -dimensional affine subspace assigned to the face (or the boundary) is reduced or increased or the subspace undefined.

Visual event 2 The intersection of the faces is outside the boundaries of one/several face(s).

Furthermore, the two-dimensional affine subspace of a face is determined by a normal $n_i \in \partial\mathbb{B}_1^3(0)$ as well as the distance to the origin $d_i \in \mathbb{R}$ of the face $i = 1, 2, 3$. Since \mathcal{V} is an intersection of these planes the matrix

$$\mathcal{N} = \begin{pmatrix} n_1^T \\ n_2^T \\ n_3^T \end{pmatrix} \quad (2.9)$$

of the normals of the three planes is regular and with $\bar{d} = (d_1 \ d_2 \ d_3)^T$ the equation system $\mathcal{N} \cdot \mathcal{V} = \bar{d}$ holds. With the rule of Cramer the components of the vertex \mathcal{V} are rational functions of n_i , $i = 1, 2, 3$ and the distances d_i . The numerator is a polynomial of the parameters and the denominator is $\det \mathcal{N}$. Thus, the components of the vertex are in C^∞ as long as the above matrix \mathcal{N} remains regular and the components of \mathcal{N} and d are in C^∞ . The components of the normal of the faces are not in C^∞ if and only if Visual event 1 occurs. The additional visual event is the following:

Visual event 3 The normals of the faces are linearly dependent.

The repositioning of a camera is illustrated in Figure 2.18. The images depict examples for the Visual events 1–3: The top, left pictogram shows a projection face which vanishes after the movement of the camera. The image to the right in the same row displays the rotation of the camera to a point where the normal of an opening face and the normal of an environmental face are linearly dependent. Both pictures below depict a vertex which meets the boundary of an environmental face \mathcal{F}_E . In the left image the boundary is a vertex defined by a projection face \mathcal{F}_P and \mathcal{F}_E . In the right image the boundary is defined by an opening face \mathcal{F}_O and \mathcal{F}_E .

In the following paragraphs, it is shown that Visual event 1 only occurs if one of the vertices meets another face of the coverage or if one of the positions of the cameras are in a subspace assigned to an

anchor defined in Definition 2.1.1. In order to show this, assume (2.1) for all cameras and define the following set:

$$\mathbb{K} := \left\{ \left(A \times {}^{\partial} \mathbb{B}_1^3(0) \right) \times \mathbb{P}_{\pi(2)} \times \cdots \times \mathbb{P}_{\pi(N)} \mid A \subset \mathbb{R}^n \text{ subspace assigned to an anchor } \mathcal{E} \subset {}^{\partial} E, \pi \in S_N \right\} \quad (2.10)$$

Its elements are the subsets of network parameters where at least one camera position is in a 1-dimensional affine subspace of an anchor of the environment.

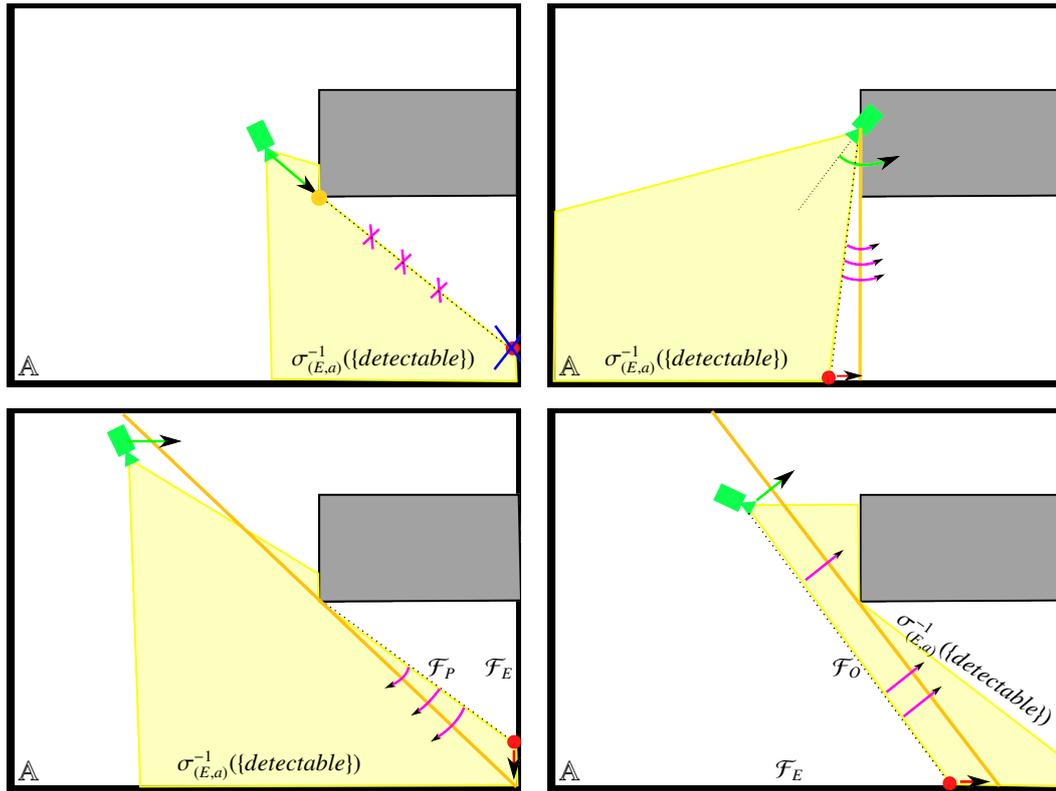


Figure 2.18: Illustration of Visual events 1–3 with the field of view (yellow) of one single camera (green) in 2D; The movement (green arrow) of the camera changes the faces (magenta arrows or crosses) of the camera coverage $\sigma_{(E,a)}^{-1}(\text{detectable})$ (yellow) and thereby changes a vertex (red dot/arrow, blue cross). The orange line and dot illustrate the critical position or orientation of the camera where the visual events arise. Top left: Event 1; Top, right: Event 3; Bottom: Event 2 with an environmental face \mathcal{F}_E and differing boundaries to \mathcal{F}_E . The boundary is a vertex defined by a projection face \mathcal{F}_P and \mathcal{F}_E (bottom, left), and defined by an opening face \mathcal{F}_O and \mathcal{F}_E (bottom, right).

The sets \mathbb{K} of Equation (2.10) and \mathbb{I} of Definition 2.3.8 are used in both the next lemmas. The latter addresses Visual event 3.

Lemma 2.4.1

Let E be an environment. Let $N, k \in \mathbb{N}$ with $k \leq N$ and $S \subset \mathbb{S}$. Let \mathbb{I} be as in Definition 2.3.8 and \mathbb{K} as in

Equation (2.10). Let the following set be connected:

$$D \subset \mathbb{P}_1 \times \dots \times \mathbb{P}_N \setminus \left(\bigcup_{I \in \mathbb{I}} I \cup \bigcup_{V \in \mathbb{K}} V \right)$$

Let $x \in D$. Let $C_x(k, S)$ be the k -reliable coverage (Section 2.1). Let \mathcal{V} be a vertex of $C_x(k, S)$. Let \mathcal{F} be a face of $C_x(k, S)$ with $\mathcal{V} \in \mathcal{F}$.

Then, \mathcal{F} exists for all $x \in D$ and its normal and distance to the origin are in C^∞ .

Proof. \mathcal{F} can either be subset a projection face (\mathcal{F}_P), silhouette face (\mathcal{F}_S), or opening face (\mathcal{F}_O) of the coverage of one camera (Equation (2.8)), w.l.o.g. camera with parameters $(p, o) \in E \times \partial \mathbb{B}_1^3(0)$, or it could be a constant face (\mathcal{F}_a).

The affine subspace of constant faces and opening faces always exists, thus, only the existence of the following faces of the coverage is critical:

(\mathcal{F}_P): The two-dimensional affine subspace assigned to \mathcal{F} is completely defined by p and an anchor $\mathcal{E} \subset \partial E$ of the environment, cf. Lemma 2.3.7, unless the position is in the subspace A of the edge. This case is excluded, however, since $A \times \partial \mathbb{B}_1^3(0) \in \mathbb{K}$.

(\mathcal{F}_S): The affine subspace is defined by p and an anchor $\mathcal{E} \subset \partial E$ of the dynamic objects of the environment, cf. Lemma 2.3.7. This is the same case as (\mathcal{F}_P), however: The face is replaced by another silhouette face if the position is in the subspace of either one of the polygons adjacent to the \mathcal{E} , which are the vertex incidence surfaces of Table 2.3 type CPP.

The normal and distance to the origin must be in C^∞ :

(\mathcal{F}_P) and (\mathcal{F}_S): The normal of a plane can be calculated by three points in the affine subspace: Let the linearly independent points $z_1, z_2, z_3 \in \mathcal{F}$ be ordered counterclockwise when viewed from a point inside the environment from which z_1, z_2, z_3 are visible. The normal n of a hyperplane satisfies

$$Z^T \cdot n = d \cdot \mathbf{1} \quad \text{with } Z = \begin{pmatrix} z_1 & z_2 & z_3 \end{pmatrix} \text{ and } \mathbf{1}^T = \begin{pmatrix} 1 & 1 & 1 \end{pmatrix}$$

We choose the position of the camera p and two points of the anchor as the points z_r , $r = 1, 2, 3$. With the rule of Cramer, the components of the normals are in C^∞ unless $\det Z^T = 0$, which are the above two cases.

(\mathcal{F}_O): The frustum of a camera is defined as an intersection of two theta-spaces, again defined by the position p , direction d , opening v , and opening angle θ . Its boundary is $\{x \in \mathbb{R}^3 \mid \angle(\rho_{(d,v)}(x), d) = \pm \frac{\theta}{2}\}$. The normal $n \in \partial \mathbb{B}_1^3(0)$ of \mathcal{F} lies in the subspace of d and v with $n = \lambda_d d + \lambda_v v$ and its angle to d is known. Unless $\theta = \pi$ in which case the normal holds $n := d$, it can be constructed with the ratio $\tan(\frac{\pi-\theta}{2}) = \pm \frac{\lambda_v}{\lambda_d}$.

In the first case, the components of the normals are in C^∞ unless a vertex meets an affine subspace of an anchor or a vertex incidence surface. In the second case the normals are in C^∞ . Additionally, the distance to the origin d holds $d = z^T \cdot n$ for any point $z \in \mathcal{F}$, w.l.o.g. this can be the position of the camera p , cf. Lemma 2.3.7. This settles the proof. \square

We have seen that the normals of the faces are continuously differentiable and the affine subspaces of the faces will keep their dimensionality unless a vertex incidence surface or an affine subspace of an anchor is met. In the following lemma, the normals of the faces are proven to be linearly independent under similar conditions:

Lemma 2.4.2

Let E be a bounded environment. Let the variables be declared as in Lemma 2.4.1. Let $x \in D$. Let $C_x(k, S)$ be the k -reliable coverage of Section 2.1. Let \mathcal{V} be a vertex of $C_x(k, S)$. Let \mathcal{N} be as in of Equation (2.9) for the normals of the vertex \mathcal{V} .

Then, \mathcal{N} is regular for all $x \in D$.

Proof. Network parameters $x \in D$ exists where \mathcal{V} is a vertex of $C_x(k, S)$. At this point, $\mathcal{N} = \mathcal{N}(x)$ is regular and constitutes the case c.ii) in Figure 2.19. The matrix of any three normals n_1, n_2, n_3 is singular if the normals can be linear combined, which is true if the planes are arranged like the illustrations a), b), c.i), or c.iii). We will give an intuitive reason why the latter cases cannot emerge from case c.ii) with the continuity of the normals (Lemma 2.4.1) in D :

First, we motivate that \mathcal{V} meets an additional face before the cases c.i) and c.iii) emerge, since the polyhedral area is bounded by E : In general, all edges of a bounded polyhedral area have two vertices as boundary points, \mathcal{V} and $\overline{\mathcal{V}}$. Since the normals are continuous with the network parameters, c.i) emerges from c.ii) if the edge $[\mathcal{V}, \overline{\mathcal{V}}]$ between two planes (intersection of two planes) levels up with the third plane. When leveling up, the $\overline{\mathcal{V}}$ meets another face of the coverage which can only happen if x is in a vertex incidence surface. These have been excluded from D .

The three edges $\mathcal{E}_1, \dots, \mathcal{E}_3$ in c.iii) as well as the two edges in b) are parallel with a distance bigger than 0. The three planes in a) are also distanced by a scalar bigger than 0. Before a), b), and c.iii) can emerge from c.ii) the edges/planes need to open up. Thereby, the vertex $\mathcal{V} = \mathcal{E}_1 \cap \dots \cap \mathcal{E}_3$ slides to the boundary of the edges since these are bounded (since the environment is bounded). The vertex therefore meets another vertex defined by some of the original faces and at least one additional face (or else it would be the same vertex). But again the parameter vector of the network is in a vertex incidence surface and holds $x \notin D$.

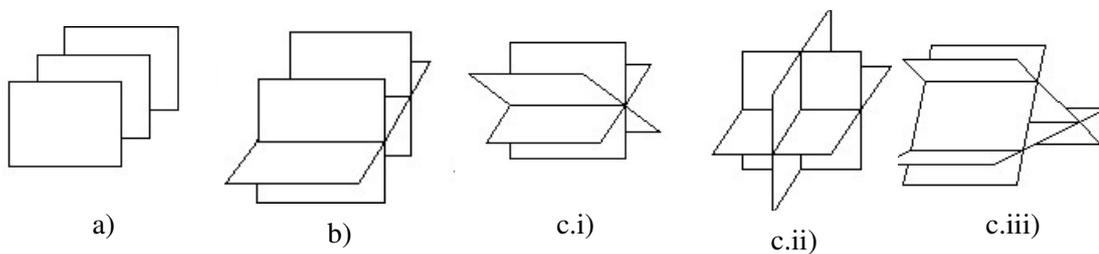


Figure 2.19: Illustration of the arrangement of the non-coinciding affine subspaces of three faces that define a vertex. They can be distinguished by how many planes are parallel, first: a) All planes are parallel (first picture); b) Two planes are parallel (second picture), the third intersects both (if it intersects one it intersects a parallel one too); and c) No two planes are parallel, instead all planes intersect. We know that two planes intersect in a line. The intersection lines of three planes can then c.i) be identical, c.ii) meet in one point or c.iii) be parallel (the intersecting lines cannot be skew since they are situated at the same three hyperplanes). Picture from [146]. \square

With these lemmas, the way is cleared for the final result of the observations of polyhedral areas and their vertices. The volume of the fused coverage is continuously differentiable except for a small set of network parameters, since the vertices are continuously differentiable μ -almost everywhere:

Theorem 2.4.3

Let the variables be declared as in Lemma 2.4.2. The components of the vertex are in C^∞ μ -almost everywhere. In particular, they are in C^∞ for all $x \in D$.

Proof. The components of the vertex are in C^∞ for $x \in D$ since: Lemma 2.4.2 and 2.4.1 show the non-existence of the Visual events 1 and 3 if $x \in D$. As a last condition, the vertex must not pass through the boundaries of the face, an edge of the fused coverage. This edge is an intersection of two faces of the coverage. When a vertex meets such an edge, it also meets an additional face, which can only happen if $x \notin D$. A set in \mathbb{K} can be denoted as $(A \times {}^0\mathbb{B}_1^3(0)) \times \mathbb{P}_{\pi(2)} \times \cdots \times \mathbb{P}_{\pi(N)}$ with a suitable affine subspace of an anchor $A \subset \mathbb{R}^n$ and permutation $\pi \in S_N$, which is a null set in $\mathbb{P}_{\pi(1)} \times \mathbb{P}_{\pi(2)} \times \cdots \times \mathbb{P}_{\pi(N)}$. With Theorem 2.3.9 it is known that the vertex incidence surfaces are null sets. \square

Thus, the volume of the coverage is continuously differentiable μ -almost everywhere. The continuity is relevant for the solver that is proposed in the next chapter. The solver also has to be chosen by the number of local optima of the objective function. More than one local optimum of the volume of the fused coverage is shown in the next section.

2.4.2 Non-Convexity

In optimization, convexity or concavity of a function is often used to prove that only one local maximum or minimum exists. In this paragraph we show with a simple example that the volume of the coverage of a single camera is not necessarily convex nor concave. Thus, the volume of the k -reliable coverage is not necessarily convex or concave.

A room illustrated in Figure 2.20 is used as an environment. Only one single camera is setup in a downward angle. The environment includes a second floor whose edge is a regular, rectangular zigzag-line along the diagonal of the room. The camera is moved along the diagonal of the room so that the corners of the additional floor are hit in a regular distance. The volume of the field of view of the camera is to be measured.

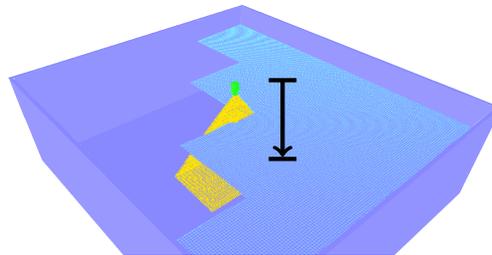


Figure 2.20: Illustration of a camera (green) with its detectable coverage (yellow) placed in an environment (gray) with an additional floor (turquoise checked). The room has the volume $(8 \times 9.2 \times 2.7) \text{ m}^3$. The second floor is at a height of 2 m. The linear trajectory of the camera's position starts at $(-0.7, -1, 2.65)$ and ends at $(1.3, 1, 2.65)$ with the origin at $(0, 0, 0)$. The camera is oriented towards a point 1.4m below the position, 1 cm to the right and 10 cm to the front.

The volume of the field of view is measured by counting the detectable voxels of the surveillance area. The camera is moved along a linear trajectory starting and ending at a point where the field of view only reaches the second floor and where its volume adopts a minimal value. The trajectory of the position is chosen such that only two ledges and a single notch of the edge is touched. The diagram in Figure 2.21 depicts the number of voxels versus the position of the camera. The function is neither convex nor concave. Furthermore, when enlarging the trajectory such that more notches are reached, the function is extended by the same graph and has several maxima and minima.

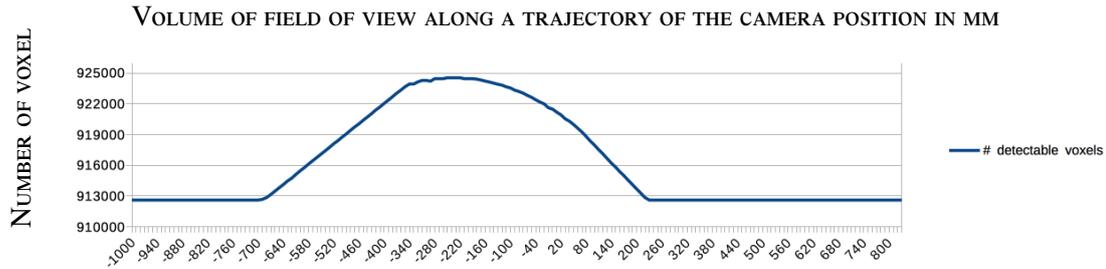


Figure 2.21: Diagram of the number of voxels which are marked as detectable when moving the camera position along the trajectory in Figure 2.20 discretized into steps of $\sqrt{2}$ cm. The voxelspace is discretized into voxels of the size $(5 \times 5 \times 5)$ cm³.

Thus, even when just considering the detectable coverage of a single camera as an objective function, it can yield more than one optimum. The same applies to the identical coverage. Some of these optima can be summarized with the following section about symmetry of a function.

2.4.3 Symmetry

When evaluating an expensive function f , meaning the function evaluations take an unfeasible amount of time or computing resources, it is useful to identify regions of the domain of the function where the function value has already been evaluated. This could be due to boundary conditions or other properties of the function, such as the knowledge of regions of equal function values. All the more efficient it is if each additional evaluated point increases the region of known function values many times over. One such property is the symmetry of a function:

Definition 2.4.4 1. Let $\mathcal{D} \subset \mathbb{R}^n, n \in \mathbb{N}$. Let V_m be orthogonal subspaces of \mathcal{D} for all $m = 1, \dots, M$, $M \in \mathbb{N}$, in which V_m has dimension $n_m \in \mathbb{N}$. Then, the tuple (V_1, \dots, V_M) is called a *decomposition* of \mathcal{D} if the domain has a product structure

$$\mathcal{D} = V_1 \times \dots \times V_M \text{ with } n = n_1 + \dots + n_M.$$

2. Let the corresponding partition of the identity matrix $\mathbb{1}_n$ be as in [103]:

$$\mathbb{1}_n =: (U_1, \dots, U_M) \in \mathbb{R}^{n \times n}, U_m \in \mathbb{R}^{n \times n_m}, m = 1, \dots, M. \quad (2.11)$$

And let $x \in \mathcal{D}$ and $x_m \in V_m$ be such that $x = \sum_{m=1}^M U_m x_m$. Then the tuple $[x_1, \dots, x_M]^T$ is called *subspace coordinates* of x .

3. Let \mathcal{D} be such that an orthogonal decomposition $\mathcal{D} = V_1 \times \dots \times V_M$ with $V_{m_1} \sim V_{m_2}$ being two isomorph subspaces $m_1, m_2 \in \{1, \dots, M\}$. Additionally, let $f : \mathcal{D} \rightarrow \mathbb{R}$ be a real function. Then, f is called *symmetric in the pair* (V_{m_1}, V_{m_2}) , if

$$f(x) = f\left(U_{m_1}x_{m_2} + U_{m_2}x_{m_1} + \sum_{\substack{m=1 \\ m \neq m_1, m_2}}^M U_m x_m\right) \text{ for all subspace coordinates } [x_1, \dots, x_M] \text{ of } x \in \mathcal{D}$$

4. f is *symmetric in the subspaces* $V_{m_1}, \dots, V_{m_B}, B \in \{1, \dots, M\}$ if f is symmetric in all the pairs.

In sensor network optimization the space of parameters is made of the parameter spaces of each sensor $\mathbb{P}_1 \times \dots \times \mathbb{P}_N$. Additionally, the parameter space of each sensor can be decomposed into sensor specific, smaller parts. A camera, e.g., can be adjusted by the position $p \in E$ and orientation $o \in \mathbb{B}_1^3(0)$, as in Equation (2.1): The parameter space of the network of several cameras with these parameters is:

$$\mathbb{P}_1 \times \dots \times \mathbb{P}_N = (E \times \mathbb{B}_1^3(0))^N$$

Both the decompositions $(\mathbb{P}_1, \dots, \mathbb{P}_N)$ and $(E^N, \mathbb{B}_1^3(0)^N)$ out of a variety of subspace combinations are important for sensor network optimization: When optimizing the volume of the k -reliable coverage on subspaces alternately, the optimization on the first tuple will place each camera separately. The optimization on the second tuple will place all cameras together, then orient them together. The first tuple is even more relevant since the k -reliable coverage of a sensor network is symmetric in two or more equally built sensors. This fact is discussed in the following lemma:

Lemma 2.4.5

Let E be an environment. Let $N, k \in \mathbb{N}$ and $k \leq N$. Let us utilize a sensor network with N sensors and sensor labels $S \subset \mathbb{S}$ as in Section 2.1.3 and 2.1.4. Let the sensor $n_1 \in \{1, \dots, N\}$ and sensor $n_2 \in \{1, \dots, N\}$ be equally built: $\mathbb{P}_{n_1} \sim \mathbb{P}_{n_2}$. Then the function of the coverage

$$\begin{aligned} C : \mathbb{P}_1 \times \dots \times \mathbb{P}_N &\rightarrow \mathcal{P}(\mathbb{R}^n) \\ x &\mapsto C_x(k, S) \end{aligned}$$

is symmetric in the pair $(\mathbb{P}_{n_1}, \mathbb{P}_{n_2})$.

Proof. Let $x := [a_1, \dots, a_N] \in \mathbb{P}_1 \times \dots \times \mathbb{P}_N$ be the parameter vector of the network consisting of the parameter vectors $a_n \in \mathbb{P}_n, n = 1, \dots, N$. Let $C_N(k) \subset S_N$ be the set of k -combinations of the N -tuple $(1, \dots, N)$ with order $\frac{N!}{k!(N-k)!}$. The k -reliable coverage of the sensor network defined in Definition 2.1.11 holds

$$C_{(a_1, \dots, a_{n_1}, \dots, a_{n_2}, \dots, a_N)}(k, S) = \bigcup_{\pi \in C_N(k)} \bigcap_{n=1}^k \sigma_{(E, a_{\pi(n)})}^{-1}(S)$$

The combinations π are determined by choosing k sensors out of N total without replacement disregarding the sequence of the tuple. When considering $C_{(a_1, \dots, a_{n_2}, \dots, a_{n_1}, \dots, a_N)}(k, S)$ the above union is just rearranged. \square

The intuitive meaning of the lemma is as follows: With the use of several equally built cameras in a network, the k -reliable coverage is invariant under a permutation of the cameras. This lemma is especially helpful in a later chapter, Section 3.1.3, where we will incorporate such symmetry into one of our solvers in order to accelerate the optimization of the coverage.

2.4.4 Stair-casing

Section 2.4.1 contains the continuity of the exact volume of the k -reliable coverage of a network introduced in Section 2.3.2. For exactly calculating the volume, however, the vertices and faces of the polyhedral coverage must be deduced by intersection. A more robust but approximate calculation of the volume uses an occupancy grid as introduced in Section 2.2.1. In due course, the surveillance area is discretized into an orthogonal grid, composed of small cubes of the room, called *voxels*. The volume is measured by summing the volume of the covered voxels. When moving a camera marginally, the covered voxels may not change, thus the volume remains constant.

A piecewise constant function on a finite number of intervals is called a *stair-cased* function in the one-dimensional case, [19]. Similarly to the 1D case, we call a function on an n D domain a stair-cased function if it is constant on a finite number of sets that form a partition of the domain. In literature, such a function is also called step function, simple function, or discrete function. It occurs in imaging, for example.

Definition 2.4.6

Let $\mathcal{D} \in \mathbb{R}^n$, $n \in \mathbb{N}$ be a domain. A function $f : \mathcal{D} \rightarrow \mathbb{R}$ is called a *stair-cased function* if for a number $B \in \mathbb{N}$

- disjunct, connected sets $\mathcal{B}_b \subset \mathcal{D}$, $b = 1, \dots, B$, exist with $\mathcal{D} = \bigcup_{b=1}^B \mathcal{B}_b$ and
- constant scalars $\beta_b \in \mathbb{R}$, $b = 1, \dots, B$ exist

with the function value being expressed by indicator functions as

$$f(x) = \sum_b^B \beta_b \cdot \mathbf{1}_{\mathcal{B}_b}(x) \quad \text{with } \mathbf{1}_{\mathcal{B}_b}(x) = \begin{cases} 1 & \text{if } x \in \mathcal{B}_b \\ 0 & \text{otherwise} \end{cases} \quad \text{for all } b = 1, \dots, B.$$

When implemented as suggested in Section 2.2.1, the function $x \mapsto \lambda(C_x(k, S))$ is a stair-cased function since only a finite number of voxels exist in the occupancy grid. The scalars β_b are known, they represent all sums of the volumes of the individual voxels. The missing facts are the size and shape of the connected sets \mathcal{B}_b . We will give an intuitive overview on how the visibility analysis of Section 2.2 affects these sets.

The coverage or visibility of a voxel is deduced by two methods in Section 2.2.1. One variant is an inverse *ray tracing* where the ray between camera position and each voxel is checked whether it is interrupted by any face. The image plane $\mathcal{I} \subset E$ is not discretized into pixels. As before, let a *voxel check* be defined as an operation on the voxel cube's center $y \in \mathbb{A}$ in the surveillance area. The sensor label $\sigma_{(E,a)}(y)$ of a voxel changes for the camera with parameters $a = (p, o)$ if the ray $[p, y)$ encounters a face of the k -reliable coverage. Thereby the decision whether the volume of this voxel is added to the overall volume

may change, as well. Recall the definition of an incidence surface (Definition 2.3.8) of a face \mathcal{F} and the voxel center y : $I(y, \mathcal{F}) := \{x \in \mathbb{P}_1 \times \dots \times \mathbb{P}_N \mid y \in \mathcal{F}\}$. Let

$$\mathbb{J} := \{I(y, \mathcal{F}) \mid y \in \mathbb{A} \text{ is voxel center, } \mathcal{F} \text{ is face of } C(k, S)\} \quad (2.12)$$

be the *set of all voxel incidence surfaces*. Trespassing exactly one incidence surface decreases or increases the volume of the k -reliable coverage by exactly one voxel volume. In Figure 2.22, the voxel incidence surfaces are illustrated in two dimensions for one single camera's coverage and its projection face (left) or opening face (right) or opening face (right).

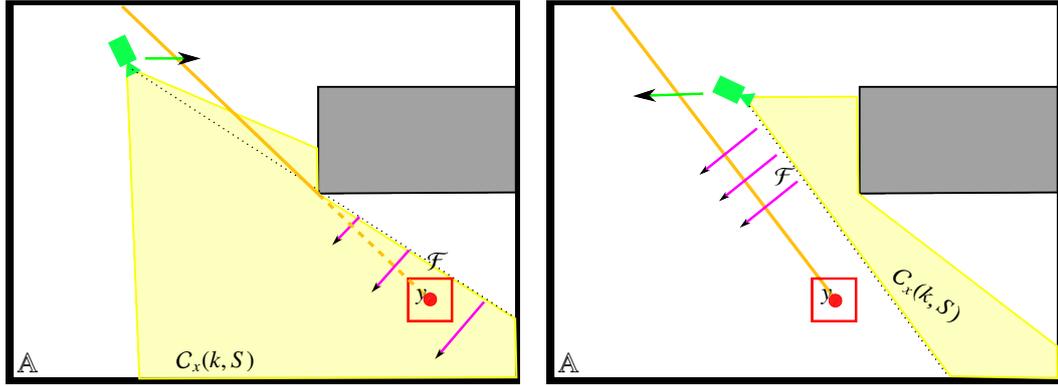


Figure 2.22: Illustration of incidence surfaces (orange) of a voxel $y \in \mathbb{A}$ and a projection or silhouette face (dotted black, left image) or opening face (dotted black, right image) \mathcal{F} of the k -reliable coverage $C_x(k, S)$ (yellow area). The camera movement (green arrow) induces the change of the face \mathcal{F} (magenta arrows). The volume of the k -reliable coverage is the sum of voxel volumes and changes if \mathcal{F} meets y .

The set of all voxel incidence surfaces defines the partition of $\mathcal{D} = \mathbb{P}_1 \times \dots \times \mathbb{P}_N$:

Lemma 2.4.7

Let $k \in \mathbb{N}$ and $S = \{\text{detectable}\}$ or $S = \{\text{identical}\}$. Let the variables be declared as in Theorem 2.4.3. Let us utilize the inverse ray tracing method in Section 2.2.1 as a visibility analysis.

- Let the following set be connected

$$\mathcal{B} \subset \mathbb{P}_1 \times \dots \times \mathbb{P}_N \setminus \left(\bigcup_{I \in \mathbb{I}} I \cup \bigcup_{I \in \mathbb{J}} I \cup \bigcup_{I \in \mathbb{K}} I \right) =: \mathbb{P}_{(\mathbb{I}, \mathbb{J}, \mathbb{K})}.$$

The function $x \in \mathbb{P}_1 \times \dots \times \mathbb{P}_N \mapsto \lambda(C_x(k, S))$ is constant on \mathcal{B} .

- Let $\mathcal{B}_b \subset \mathbb{P}_{(\mathbb{I}, \mathbb{J}, \mathbb{K})}$, $b = 1, \dots, B$, be all the maximal and connected sets in the domain $\mathcal{D} = \mathbb{P}_1 \times \dots \times \mathbb{P}_N$. Then, $\mathcal{D} = \bigcup_b^B \mathcal{B}_b$ holds.

Proof. The volume $\lambda(C_x(k, S))$ only changes if the volume of a voxel is added or left out. W.l.o.g. let $y \in \mathbb{A}$ be the center of this voxel. The decision of whether the volume of y is added to the overall volume only changes if the map $\sigma : (E, a, y) \mapsto s \in S$ of one of the cameras with parameter vector $a \in \mathbb{P}$ is

changed in the environment E . As long as the vertex components of the k -reliable coverage $C_x(k, S)$ are continuous, the label $\sigma_{(E,a)}(y)$ of a voxel only changes if y encounters a face \mathcal{F} of C . Per definition this incident is only possible if $a \in I(y, \mathcal{F})$. Since \mathcal{B} is connected the coverage of one single camera is therefore constant on \mathcal{B} . When using more than one camera the function $\lambda(C_x(k, S))$ is constant on \mathcal{B} since the volume of all camera coverages are constant on \mathcal{B} .

We still need to show that each point $z \in \mathcal{D} \setminus \bigcup_b^B \mathcal{B}_b$ is in the boundary $\partial \mathcal{B}_b$ of any $b = 1, \dots, B$. The center of such a voxel is constant with a , therefore the voxel incidence surface corresponds to the vertex incidence surface of type CCC when meeting a projection/silhouette face in Table 2.4 or when meeting an opening face in Table 2.5, a plane. Theorem 2.3.9 proves that the vertex incidence surfaces are null sets in $\mathbb{P}_1 \times \dots \times \mathbb{P}_N$. A similar statement can be made for voxel incidence surfaces. Thus, $\mathcal{D} \setminus \bigcup_b^B \mathcal{B}_b$ is a null set in \mathcal{D} . An excluded null set of \mathcal{D} is in \mathcal{D} 's boundary, which leads to the fact that $z \in (\mathcal{D} \setminus \bigcup_b^B \mathcal{B}_b) \subset \partial (\bigcup_b^B \mathcal{B}_b) \subset (\bigcup_b^B \partial \mathcal{B}_b)$. \square

With this lemma, we know that the boundary of the partition of \mathcal{D} is defined by voxel incidence surfaces when utilizing the inverse ray tracing method.

Conversely, in the second method of Section 2.2.1, the image plane \mathcal{I} is discretized into pixels to synthesize images. As in Definition 2.1.7, a point of the environment is mapped onto a specific pixel by projecting it onto the image plane, by a projection $\pi : \mathbb{V}_a(E) \rightarrow \mathcal{I}$, and afterwards by rasterizing the image plane and thereby assigning a specific pixel with $\rho : \mathcal{I} \rightarrow \mathbb{N}^2$, e.g., by rounding. The value of this pixel is the depth to the next face of the environment. The next face is determined, e.g., by intersecting the ray $[p, x)$ with the faces of the environment (static or dynamic depending on whether or not synthesizing the reference image) where $p \in E$ denotes the position of the camera and $x \in \mathcal{I}$ denotes the pixel center. Another method to determine the value of the pixel is the z-Buffer method in which all faces are projected and rasterized by π and ρ . The steps of the *method of image synthesis* are the following:

1. The center of the voxel y is assigned a pixel with the pixel center $x := \rho(\pi(y))$.
2. The pixel center x is assigned a point $z \equiv z(p, x)$ on the nearest face, e.g. by $z \in \partial E \cap [p, x)$.
3. Finally, the sensor label of the voxel center y is determined by the decisions whether or not the voxel is inside the frustum and whether or not the difference $d(p, z) - d(p, y)$ with $z \equiv z(p, \rho(\pi(y)))$ is larger than 0.

The sensor label of y changes if one of the decisions in Step 3 are changed. We have already covered the frustum and its opening faces in the incidence surfaces in Lemma 2.4.7. The sensor label of y also changes if the second decision in Step 3 is changed: The decision concerns the projection faces and silhouette faces and is more tricky: The voxel y and faces of the environment are invariant under the change of camera parameters. The decision changes either if the voxel is assigned a different pixel (in $\rho(\pi(y))$) or if the pixel value $d(p, z)$ changes. These two changes to incidence surfaces of projection and silhouette faces of the set \mathbb{J} are described in the following paragraphs.

Firstly, the sensor label of a voxel y may vary if the voxel is assigned a different pixel: Imagine each pixel as a separate camera with four opening faces as illustrated in Figure 2.23 (left). If such a *pixel opening face* trespasses a voxel center y when changing the camera parameters, then the voxel center is assigned

to a neighboring pixel with $\rho(\pi(y))$ possibly containing a different depth value. The change in the depth values may result in a change of the sensor label of the voxel.

The projection of y to a different pixel does not necessarily result in a variation of the sign of $d(p, z) - d(p, y)$. While inverse ray-tracing with a non-discretized image plane, the difference changes its sign if one of the faces of the coverage (projection or silhouette faces since opening faces have already been discussed) hit the voxel. Before image space discretization, the anchor \mathcal{E} of a projection or silhouette face projected into the image space $\pi(\mathcal{E})$ was a segment \mathcal{S} (or point). The neighboring polygons, $\mathcal{F}_1, \mathcal{F}_2$ of this edge are now rasterized in the image plane, as illustrated in Figure 2.23 (right). Thereby, the segment becomes a polygonal chain \mathcal{S} in the image space, defined by the boundaries of the rasterized polygons

$$\mathcal{S} := \rho(\pi(\mathcal{F}_1)) \cap \rho(\pi(\mathcal{F}_2)). \quad (2.13)$$

One projection or silhouette face therefore becomes a set of projection and silhouette faces, each defined by a segment of the polygonal chain \mathcal{S} and the position of the camera. We will call the set *pixel projection face* or *pixel silhouette face* in the next paragraphs. The pixel projection or silhouette faces are a subset of the pixel opening faces above.

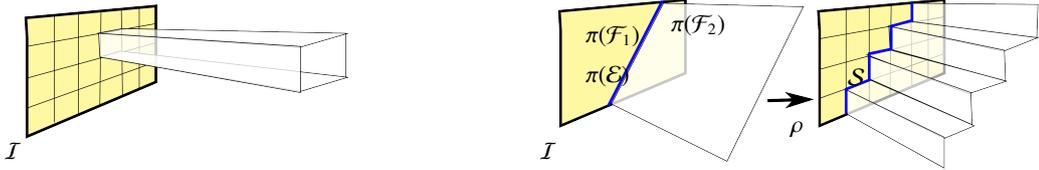


Figure 2.23: Illustration of pixel opening faces (left) and pixel projection faces or pixel silhouette faces (right) in the context of a discretization of the image plane $\mathcal{I} \subset E$ (yellow). The projection of the anchor $\pi(\mathcal{E})$ (blue) of a pixel projection/silhouette face in the image plane becomes a polygonal chain $\mathcal{S} \subset \mathcal{I}$ by the discretization ρ (right image).

The voxel is assigned a different pixel if the sensor network parameters are part of an incidence surfaces of these pixel opening faces and the voxel center y as illustrated in Figure 2.24 (left). In the set of incidence surfaces \mathbb{J} the projection and silhouette incidence surfaces need to be adapted.

The second adaption to pixel projection and pixel silhouette faces is not caused by a voxel y being assigned to a different pixel x by $x = \rho(\pi(y))$. The sign of $d(p, z) - d(p, y)$ may also vary if the point of the nearest face $z \equiv z(p, x)$ changes whose depth value is stored in the pixel. Again, while inverse ray-tracing with a non-discretized image plane, the sign will only change if the voxel status switched from *visible* to *occluded* by one of the polygons of the environment. Then, one of the pixel projection faces trespasses the z . The pixel projection faces are defined by the position of the camera and a polygonal chain \mathcal{S} . The polygonal chain is defined by the rasterized faces \mathcal{F}_1 and \mathcal{F}_2 . A voxel can be suddenly occluded if the point z changes from \mathcal{F}_1 to \mathcal{F}_2 . The pixel silhouette faces adopt similar behavior like the pixel projection faces.

The variation of the pixel value is illustrated in Figure 2.24 (right) including incidence surface in which such a change occurs. Without image discretization, the polygonal chain \mathcal{S} and thereby the parameters of

a projection or silhouette face are continuous with the parameters of the camera. With the discretization of the image space, when trespassing such an incidence surface, a rasterized face shows up in the pixel which the considered voxel is assigned to and occludes the voxel. The segment S has changed non-continuously.

In the course of this section, we have developed the boundaries of the connected sets on which the volume of the coverage of a camera is constant. The function $\lambda(C_x(k, S))$ as discussed in the Sections 2.1 and 2.2 is therefore stair-cased on the same connected sets, separated by the surfaces and curves in \mathbb{I} , \mathbb{J} , and \mathbb{K} . The set \mathbb{J} as defined in (2.12) includes the incidence surfaces when using the inverse ray-tracing method. The additional incidence surfaces for the image synthesis method encompass the camera positions and orientations where voxels are assigned different pixels or the pixel value varies. The incidence surfaces can be constructed by pixel opening faces including pixel projection faces and pixel silhouette faces. The stair-casing behavior of the objective function will be an important fact to consider when developing a solver.

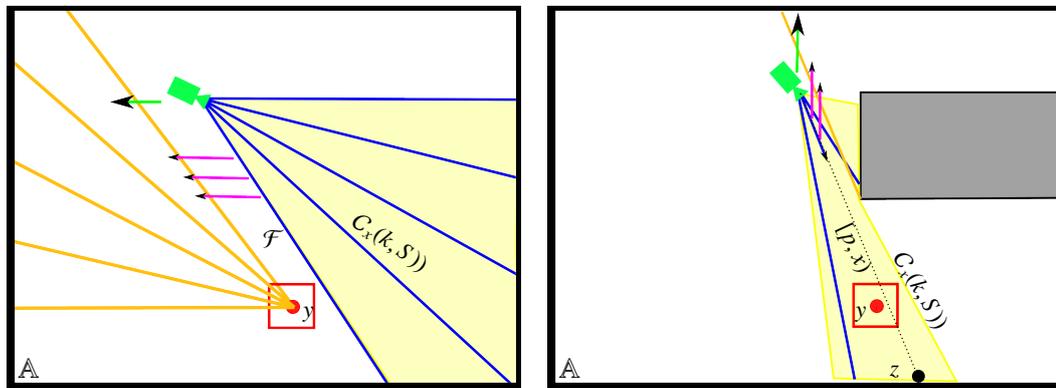


Figure 2.24: Illustration of incidence surfaces (orange) of a voxel $y \in \mathbb{A}$ and a pixel projection/silhouette face (blue line) of the k -reliable coverage $C_x(k, S)$ (yellow area). The camera movement (green arrow) induces the change of the face \mathcal{F} and pixel ray $[p, x]$ (magenta arrows). The sensor label $s \in S$ will change if the parameters of the network lie on these incidence surfaces. The sensor label changes if y is assigned to a different pixel center $x \in \mathcal{I}$ of the image space (left) and if the value of the pixel with pixel center x changes (right). The value changes if a rasterized polygon of the environment shows up in the pixel and changes $z \in {}^\partial E$ abruptly.

The investigation about the stair-cases of this function is the last property of the objective function $\lambda(C_x(k, S))$ that is discussed in this section. We have also considered the continuity, convexity, and symmetry of the function, which further on will be important facts to consider when developing a solver for camera network optimization.

2.5 Summary

We have introduced a method to detect changes in an environment in order to reconstruct a human silhouette. This is done by finding the difference between a reference image and a subsequent image of

a single camera in Section 2.1.2. The method is called *background subtraction* method. This purely 2D method considering only one single camera can be extended into the 3D environment to approximate a target conservatively by multiple cameras, in Section 2.1.3. The approximation is only as good as the change detection system. We come to the conclusion that the identical or detectable coverage needs to be maximized in order to approximate the target more accurately in Section 2.1.4. In the final result, the permitted number of failing cameras is considered for constructing a failure-resistant system.

The implementation of the visibility analysis is addressed in the second section of this chapter. In order to accelerate the optimization of camera network parameters we have accelerated the construction of the approximation of one sensor (Sections 2.2.1 and 2.2.2), and decreased the runtime of subsequent constructions by the variation of only a selected number of sensors in two methods (Section 2.2.3). In Section 2.4.3 the fact is stated that switching two sensors does neither change the coverage nor the volume of the coverage of the sensor network. This allows us to exploit symmetry. All four methods can be used in an optimization of camera network by the solvers developed in Chapter 3.

The key to objective function is the k -reliable coverage, it can be used to express both an approximation of the target or the detectable regions of a room. This is why we address the shape of the coverage disregarding the discretization. In Section 2.3.1, we have proven that the shape is a polyhedral area, a polyhedron that can be disconnected and/or “flat”. Its faces and vertices are necessary to calculate the volume of the polyhedral area (Section 2.3.2). We have classified all the vertices and faces of the coverage (Section 2.3.3) in order to show where the faces meet other points of the environment (Section 2.3.4). These points are in non-planar surfaces in the network’s parameter space and are called *incidence surfaces*.

There are two good reasons to consider these surfaces: Firstly, an incidence surface where a face of the coverage meets a vertex of the coverage causes non-differentiable or non-continuous points in the parameter space of the camera network (Section 2.4.1). The second reason to consider incidence surfaces is the stair-casing effect of the volume of the coverage. A function is a stair-cased function if is piecewise constant on a partition of the domain. The subsets defining the partition are separated by the incidence surfaces of voxels (Section 2.4.4). The number of these sets is influenced by the discretization of the image plane (pixel) and of the surveillance area (voxel).

The new achievements in this chapter can be sorted into two groups: The purely analytic part has addressed the shape of the k -reliable coverage with the classification of incidence surfaces and vertices and has derived the properties of its volume, i.e. staircasing, continuity, differentiability, symmetry, and convexity. These properties help to choose the solver of the optimization. In the computational part, an acceleration of the construction of the coverage has been addressed, which also accelerates the optimization.

Chapter 3

Global Optimization of Costly, Non-differentiable, or Stair-cased Black-box Functions

The objective function (1.2) is usually not given analytically, but rather by a black box from simulation. Evaluating the objective is costly due to the costs of the simulation. Simulating the objective with voxels, as done in Section 2.2, causes the function to be stair-cased, meaning it is piecewise constant on a finite number of sets that form a non-uniform distribution of the domain. Furthermore, black-box, stair-cased functions have the disadvantage that a gradient can only be evaluated by numerical approximation, in case a useful gradient exists in the mathematical sense. Even in absence of numerical complications, the volume of the fused coverage of several cameras is still non-continuous and non-differentiable on a Lebesgue-null set and has an undefined number of local optima, as we have seen in Section 2.4.

These are the difficulties that we need to overcome in this chapter when designing suitable solvers for the problem at hand. Local solvers typically utilize a gradient of the objective function to establish fast convergence. An alternative class of solvers are randomized global solvers, but a general continuous function needs to be sampled dense in order to establish convergence, c.f. [153]. Thus, randomized solvers usually require more objective function evaluations than solvers that utilize a gradient. But the goal is to design a solver which calls the costly objective function less.

In this chapter we develop strategies how to globally solve a maximization problem such as the following, in an efficient manner.

Problem 3.0.1

$$\text{Find a global: } \operatorname{argmax}_{x \in \mathcal{D}} f(x) \tag{3.1}$$

in which $f : \mathcal{D} \rightarrow \mathbb{R}$ is a function on the domain $\mathcal{D} \subset \mathbb{R}^n$ whose evaluations are costly. Additionally, it may have one or more of, but is not limited to the following properties:

- f is stair-cased.

- f is given only as a black box, i.e. any operations other than function evaluations are hardly possible.
- A gradient ∇f is not known.
- f is non-differentiable.
- f is non-convex.

How to approach such a complicated problem? The thesis has started by collecting some specific properties of f that our solvers take advantage of in this chapter: On the one hand, when utilizing equally designed sensors, the objective function is symmetric, cf. Definition 2.4.4. The incorporation of this prior information about the objective will prove useful in this chapter. Also, the costs of recomputing the fused coverage of multiple cameras will decrease if only one camera is readjusted. Therefore, an optimization on a subspace of the domain seems promising, as this means restricting the parameter space to the parameters of one single camera rather than optimizing on the space of the whole camera network at once. Summarizing, the objective function may also have, but (again) is not limited to the following properties:

- Particular prior information about f exists, such as symmetry.
- The calls of f are substantially cheaper on subspaces of the parameter space.

The rest of this chapter is organized as follows: In Section 3.1, we will discuss a strategy how to solve problems with costly function calls. The lack of a gradient and the stair-casing can be compensated in this section, by approximating the costly objective by a simpler function model. My contribution to this field of research follows thereafter and can be read in the context of a summary of the whole thesis in Section 5.2.

In Section 3.1 we incorporate prior information into the function model. In Section 3.2, the cheap function calls on subspaces of the domain will be used as motivation for designing two subspace maximization methods on such a function model. One of the methods can be computed in parallel, plus the convergence of the procedure on a subspace is faster, which is shown in the experimental results of Section 3.4. The newly designed methods are proved to converge. The symmetry of the problem increases the convergence speed which is shown in the experiments. Additionally, both methods are *anytime algorithms*, meaning after an initializing phase they return a valid solution even if they are interrupted at any point in time before they end (the algorithms compute a solution that is inside the domain, although it might not be optimal at the interruption) and the solutions are iteratively improved with time.

3.1 Optimization Procedure utilizing a Radial Basis Function as a Response Surface Model

We start this section with the introduction of some terminology needed to phrase the optimization procedure we discuss. By the term *candidate* we denote a feasible point of the domain for which the costly objective function has already been or is going to be evaluated. Let the term *sample pair* denote a tuple of

a candidate together with its objective value. The issue of costly function calls can be overcome by storing the previous candidates and their objective function values. The only difficulty is that the values from candidates that lie in between previous candidates are not known precisely and have to be approximated.

In Optimization, a *response surface model* or *surrogate* of an objective function $f : \mathcal{D} \rightarrow \mathbb{R}$ is an easily evaluated function $\tilde{f} : \mathcal{D} \rightarrow \mathbb{R}$ that interpolates previous sample pairs. This response surface model is called instead of the costly objective in order to reduce the number of objective function calls. When a good solution on the response surface is found, the actual (costly) objective function is evaluated and the response surface is updated by the new sample pair. In our case we are looking for a response surface model with an easily evaluated gradient so that some of the arising problems, like stair-casing and the lack of a gradient, can be overcome by such a strategy, as well.

In Figure 3.1 (left), a stair-cased function with several local optima is depicted. The global optimum is illustrated by a blue star. The right plot illustrates a response surface model (colored surface) that interpolates 21 sample pairs of the stair-cased function to the left, shown by red crosses. The global optimum of the objective function (blue star in the left picture) and the optimum of the response surface model (within the yellow area) are already very similar. The next best candidate for a function evaluation will be chosen out of the yellow region of the response surface model, since an optimum is likely to lie there.

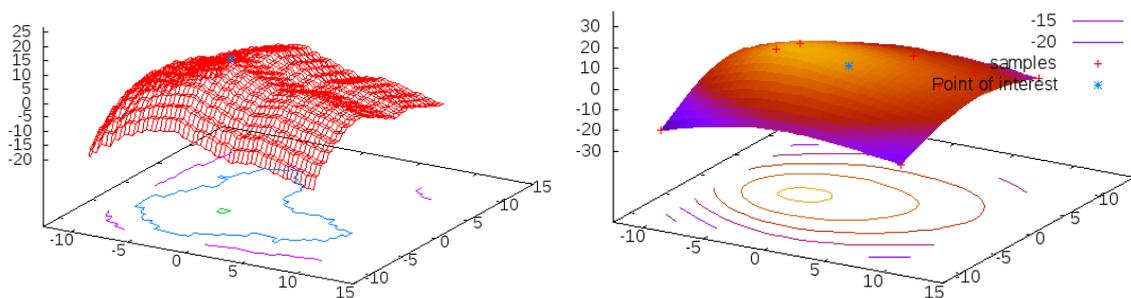


Figure 3.1: Illustration of actual costly and stair-cased objective function (red, left) and response surface model (colored, right) after an interpolation using 8 sample pairs (red crosses); The next candidate for a function evaluation will be chosen out of the yellow region of the response surface model.

In the following sections, a method to search for the next candidate is investigated, for which the objective function will be evaluated (Section 3.1.1). Then, a suitable response surface is introduced (Section 3.1.2). The fact that our objective function might be symmetric is incorporated in the investigated strategy (Section 3.1.3) and in the end the convergence of the solver incorporating symmetry and the error of the response surface model is discussed (Section 3.1.4).

3.1.1 Exclusion Area Method with a Response Surface Model

In this section, we investigate a strategy to search for the next candidate, which the costly objective function will be evaluated for. We are utilizing a randomized solver on a response surface model, here. The convergence of a randomized solver depends on the density of the sampled candidates. Additionally,

the speed of the solver utilizing a response surface model depends on the convergence of the model $\bar{f} : \mathcal{D} \rightarrow \mathbb{R}$ to the actual objective function $f : \mathcal{D} \rightarrow \mathbb{R}$.

In this section, the convergence of solver is established by choosing the candidate for the next costly function evaluation in each iteration step outside of an *exclusion area* around the candidates which were evaluated in the previous iteration steps. This ensures that regions of the room are sampled which have not been evaluated before and is adopted from [122]. The same strategy will prove the convergence of the model of Section 3.1.2 to the actual objective function. This procedure is illustrated in Figure 3.2.

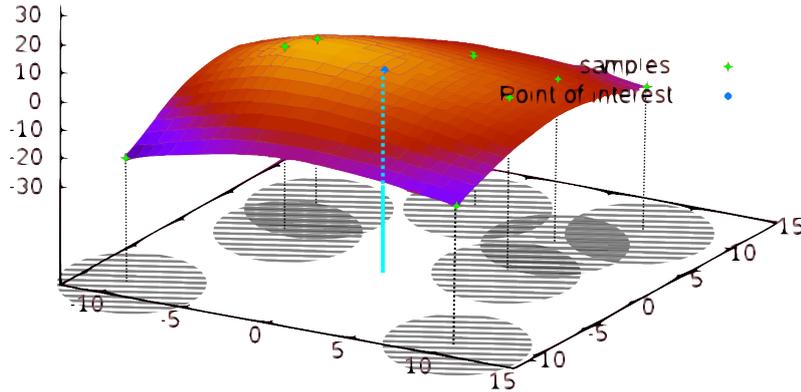


Figure 3.2: Illustration of the procedure to choose the candidate for the costly function evaluation in each iteration step: The previous sample pairs are represented by green dots. The exclusion areas around the sample pairs' candidates are depicted by the hatched areas. The next sample pair is illustrated by the blue dot and blue line. Its candidate is chosen outside of the exclusion areas.

The size of the exclusion area depends on the ratio $\beta \in [0, 1)$, the bigger β the larger the area where no candidate should be chosen. After evaluating the objective function for this candidate the response surface model is updated with the newly found sample pair. In order to cover the areas close to and far away from previous candidates the iteration is held in cycles of fixed ratios $1 \geq \beta_1 > \beta_2 > \dots > \beta_L \geq 0$. The tuple $(\beta_1, \beta_2, \dots, \beta_L)$ is called *search pattern*. The resulting iteration can be sketched as follows:

Iteration 3.1.1 The K th iteration step is given by $L \in \mathbb{N}$ updates to the response surface model:

$$\begin{aligned} K \rightarrow K + L : \quad & \bar{f}_{K+1} := \text{search_for_candidate}(f, \bar{f}_K, \beta_1) \\ & \bar{f}_{K+2} := \text{search_for_candidate}(f, \bar{f}_{K+1}, \beta_2) \\ & \vdots \\ & \bar{f}_{K+L} := \text{search_for_candidate}(f, \bar{f}_{K+L-1}, \beta_L) \end{aligned}$$

Observe that the response surface model subsequently changes with each line of the above iteration, since each time a candidate is added to the samples that are used for interpolation of the model.

In order to better understand the search for the candidate and the ratio β , consider the following: Let $\Delta \in \mathbb{R}$ be the solution of Equation (3.2) that is the largest distance between any previously chosen candidate $s_k, k = 1, \dots, K$ and the furthest point in the domain \mathcal{D} . We are looking for a candidate s_{K+1}

with a (slightly smaller) maximum distance $\beta \cdot \Delta$ to the previous candidates, in order to cover areas close to and far away from previous candidates. Under these constraints, the new candidate should maximize the response surface model \bar{f} , see (3.3). So, Δ is a measure of how large the previously mentioned exclusion area can be. After the maximization, the response surface model is updated with the sample pair $(s_{K+1}, f(s_{K+1}))$, depicted in (3.4).

$$\text{search_for_candidate}(f, \bar{f}, \beta) : \Delta := \max_{y \in \mathcal{D}} \min_{1 \leq k \leq K} \|y - s_k\| \quad (3.2)$$

$$s_{K+1} := \operatorname{argmax}_{x \in \mathcal{D}} \bar{f}(x) \text{ subj. to} \quad (3.3)$$

$$\|x - s_k\| \geq \beta \Delta, \quad k = 1, \dots, K$$

$$\bar{f} := \text{Update } \bar{f} \text{ with sample pair } (s_{K+1}, f(s_{K+1})) \quad (3.4)$$

The complete iteration is summarized in Algorithm 1.

Algorithm 1 Response Surface Model-Based Solver

Require: Finite set of initial points $C := \{s_1, \dots, s_K\} \subset \mathcal{D}$

Require: Costly function $f(\cdot)$

```

1: Evaluate costly function  $f_k \leftarrow f(s_k), \forall s_k \in S$ 
2: Update response surface model  $\bar{f}$  with sample pairs  $S := \{(s, f(s)) \mid s \in S\}$ 
3: while termination condition is not satisfied do
4:   for all  $\beta$  in the search pattern  $\langle \beta_1, \dots, \beta_L \rangle$  do
5:      $\Delta \leftarrow \max_{y \in \mathcal{D}} \min_{1 \leq k \leq K} \|y - s_k\|$  ▷ Furthest distance between samples and rest of  $\mathcal{D}$ 
6:     Maximize  $\bar{f}(x)$  ▷ Selection of candidate  $s_{K+1} \in \mathcal{D}$  for the next costly evaluation
7:     Subject to
8:        $\|x - s_k\| \geq \beta \Delta, \quad k = 1, \dots, K$ 
9:        $x \in \mathcal{D}$ 
10:
11:     Evaluate costly function  $f_{K+1} \leftarrow f(s_{K+1})$ 
12:     Update  $C \leftarrow C \cup \{s_{K+1}\}, S \leftarrow S \cup \{(s_{K+1}, f(s_{K+1}))\}$ 
13:     Update response surface model  $\bar{f}$  with sample pairs  $S$ 
14:     if  $f_{K+1} > f_k$  for all  $k = 1, \dots, K$  then
15:        $f_o \leftarrow f_{K+1}$ 
16:        $x_o \leftarrow s_{K+1}$ 
17:     end if
18:     Reset  $K \leftarrow K + 1$ .
19:   end for
20: end while
21: return  $(x_o, f_o)$ 

```

Two modifications to its original form in [122] have been applied: First, the problem (3.3) is called auxiliary problem in [122]. As can be seen above we have applied a maximization to the auxiliary problem instead of the minimization. The second alteration is merely a formalization: The authors

of [122] describe that the parameters β are chosen such that the iteration performs cycles of length L of usually decreasing β , which is why we inserted the for-loop in Line 4.

Note that in Algorithm 1 the term \bar{f} denotes the surrogate whose function value and gradient can be evaluated cheaply, and f denotes the costly objective function. The evaluation of the surrogate will be discussed in the next section. This is the basic strategy that we will refine in the rest of the chapter.

3.1.2 Radial Basis Functions as a Response Surface Model

For the response surface model \bar{f} of the actual objective function f we use a radial basis function interpolant (RBF) [116]. Colloquially speaking an RBF is a polynomial which is modified to interpolate scattered sample pairs by allowing “hills” and “valleys”. The latter are radially symmetric, hence the name. The adjective *scattered* means that the candidates of the sample pairs do not need to lie on a regular grid. An RBF has the advantage that every sample pair is interpolated, that the model is unique for a set of sample pairs, and that it is particularly smooth, as long as the candidates are unique.

The RBF defined in the next definition has been studied by [39, 116]. This model was used as a surrogate for optimization purposes [62, 100, 121] a few years later.

Definition 3.1.2

Let $\phi : \mathbb{R}_o^+ \rightarrow \mathbb{R}$ be a continuously differentiable function with $\phi(0) = 0$. Let $f : \mathcal{D} \rightarrow \mathbb{R}$ be a real function with domain $\mathcal{D} \subset \mathbb{R}^n$, $n \in \mathbb{N}$. Let Π_m^n be the linear space of polynomials of degree less than or equal to m with n variables, and let $(s_1, f(s_1)), \dots, (s_K, f(s_K)) \in \mathcal{D} \times \mathbb{R}$ be a set of sample pairs.

1. A real function $\bar{f} : \mathcal{D} \rightarrow \mathbb{R}$ is called a *radial basis function interpolant* (RBF) of s_1, \dots, s_K if weights $\omega_1, \dots, \omega_K \in \mathbb{R}$ and a polynomial $p \in \Pi_m^n$ exist with

$$\bar{f}(x) := \sum_{k=1}^K \omega_k \phi(\|x - s_k\|) + p(x), \quad x \in \mathcal{D} \quad (3.5)$$

$$\mathbf{0} = \sum_{k=1}^K \omega_k q(s_k) \quad \forall q \in \Pi_m^n \quad (3.6)$$

and if the interpolation condition $\bar{f}(s_k) = f(s_k)$ for all candidates $k = 1, \dots, K$ is met. Here, $\|\cdot\|$ denotes the Euclidean norm.

2. The function $\phi : \mathbb{R}_o^+ \rightarrow \mathbb{R}$ with $\phi(0) = 0$ is called the *kernel* of the RBF.
3. $C_K := \{s_1, \dots, s_K\}$ is called the *set of candidates*.
4. $S_K := \{(s, f(s)) \mid s \in C_K\}$ is called the *set of sample pairs*.

The first Equation (3.5) is an interpolant of the sample pairs, cf. Corollary 3.1.4. The conditions (3.6) enforce uniqueness of the interpolant (3.5). The kernel of the interpolant is usually bijective in \mathbb{R}_o^+ . Notice that the argument of the kernel of the RBF is the distance of x to a candidate s_k . This argument is radially symmetric. In this case, the kernel can be considered as a function that maps each point around s_k onto a “hill” at s_k (its value is exactly 0). So, colloquially speaking the RBF is a collection of “hills” and “valleys” on a polynomial.

In Table 3.1, several possible shapes of the kernel $\phi(r)$ are depicted in terms of the distance $r \in \mathbb{R}_o^+$ and, in some cases, a kernel parameter $\gamma \in \mathbb{R}^+$. The kernel types and shapes are shown in the first and second columns, respectively. It is known from [62] that the vector space of polynomials, where p is taken from, must be chosen according to the shape of the kernel ϕ , otherwise an RBF of Equation (3.5) with the constraints (3.6) may not exist. In particular, the polynomial's degree must exceed a minimal bound. An example which shows that the thin plate spline at least must be combined with a linear polynomial and the reason for the existence of the interpolant can be found in the next section (Corollary 3.1.8). The third column of Table 3.1 shows the minimal degree of the polynomial. In the last column, the graph of the kernel is illustrated in terms of the distance r .

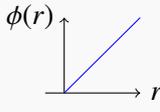
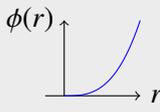
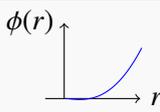
| Kernel type | $\phi(r)$ | Minimal degree m | Graph |
|-------------------|--|--------------------|--|
| Linear | r | 0 |  |
| Cubic | r^3 | 1 |  |
| Thin plate spline | $r^2 \log(\gamma \cdot r)$ (not injective) | 1 |  |

Table 3.1: Possible shapes of the kernel $\phi(r)$ depending on the distance $r \in \mathbb{R}_o^+$ and a kernel parameter $\gamma \in \mathbb{R}^+$. The degree m of the polynomial p must be chosen according to the shape of the kernel ϕ , c.f. [116].

We will use a thin plate spline as a kernel, later. In the last table, it can be observed that the polynomial needs to be of a minimal degree $m = 1$:

Example 3.1.3

In case of a thin plate spline as a kernel, a basis of the vector space of polynomials $p \in \Pi_m^n$ must be at least incorporating the linear polynomials $m = 1$, cf. [62]. Thus, with $x^{(i)}$ denoting the variable associated with the i th dimension, $i = 1, \dots, n$, one particular basis of the polynomial could be $\{1, x^{(1)}, \dots, x^{(n)}\}$, which is a hyperplane in \mathbb{R}^n .

In the following paragraphs, the form and properties of the interpolant of Definition 3.1.2 are discussed.

Interpolation of Sample Pairs, Uniqueness, and Differentiability for the Thin Plate Spline Kernel

It has not been explained, yet, how to construct a radial basis function, that interpolates given sample pairs S_K . Briefly, its construction entails defining a kernel ϕ , determining the polynomial p , and determining the weights of the kernel ω . First, we determine the weights and the polynomial abstractly without specifying on a kernel type. Then, we answer the following questions by specifying on a thin plate spline (TPS) as a kernel: Are the sample pairs interpolated by \tilde{f} ? Why are the polynomial term and the constraints (3.6) required, at all? Is the radial basis function interpolant differentiable?

For the determination of the weights, the kernel matrix of all distances between candidate pairs is defined as:

$$(\Phi)_{jk} := \phi(\|s_j - s_k\|), \text{ with } (s_j, s_k) \in (C_K)^2.$$

With $p_1, \dots, p_{\bar{m}}$ being a basis of the linear space Π_m^n ($\bar{m} \in \mathbb{N}$ such that the former is a basis of Π_m^n), we define a matrix P by

$$P = \begin{pmatrix} p_1(s_1) & \dots & p_{\bar{m}}(s_1) \\ \dots & & \dots \\ p_1(s_K) & \dots & p_{\bar{m}}(s_K) \end{pmatrix}.$$

With the function values $F = (f_1, \dots, f_K)^T$ from the sample pairs $(s_1, f_1), \dots, (s_K, f_K)$, with the weights of the kernel $\omega = (\omega_1, \dots, \omega_K)^T$, and the coefficients of the polynomial $\nu = (\nu_1, \dots, \nu_{\bar{m}})$, the solution to the equation system

$$\begin{pmatrix} \Phi & P \\ P^T & \mathbf{0} \end{pmatrix} \begin{pmatrix} \omega \\ \nu \end{pmatrix} = \begin{pmatrix} F \\ \mathbf{0} \end{pmatrix} \quad (3.7)$$

gives us the parameters ω and ν of a RBF \bar{f} that interpolates the sample pairs:

Proposition 3.1.4

Let ν and ω solve the equation system (3.7). Then, the RBF (3.5) with the weights ω and the polynomial coefficients ν interpolates the sample pairs of Definition 3.1.2.

Proof. We consider the j -th row of equation system (3.7)

$$f_j = \sum_{k=1}^K \omega_k \phi(\|s_j - s_k\|) + \underbrace{\sum_{k=1}^{\bar{m}} \nu_k p_k(s_j)}_{p(s_j)}$$

The left hand side is the function value of the objective function, while the right hand side, by definition, is the function value of the radial basis function $\bar{f}(s_j)$. \square

This proposition is true for all the mentioned kernel types. Now, we will consider a thin plate spline with kernel parameter $\gamma \in \mathbb{R}^+$ and a basis of linear polynomials, and prove some of the RBF's properties. So, let $\bar{m} := n + 1$,

$$\phi(r) := r^2 \log(\gamma \cdot r) \quad \text{and} \quad p_1(x) := 1, p_2(x) := x^{(1)}, \dots, p_{\bar{m}}(x) := x^{(n)}. \quad (3.8)$$

With these assumptions, the following lemma clarifies that the RBF is continuously differentiable:

Lemma 3.1.5

The radial basis function interpolant (3.5) with assumption (3.8) is continuously differentiable: $\bar{f} \in C^1$.

Proof. The radial basis function of (3.5) is continuously differentiable if the basis function $\phi(\|x - s_k\|)$ is continuously differentiable for all candidates s_k , since p is ∞ -differentiable. For a particular candidate s_k , the function $u = r(x) = \|x - s_k\|^2 = (x^{(1)} - s_k^{(1)})^2 + \dots + (x^{(n)} - s_k^{(n)})^2$ is the argument of our thin plate

spline kernel $\phi(\sqrt{\cdot})$ from Table 3.1 of the radial basis function (3.5). Then the first derivative of the two functions are:

$$\frac{d}{du}\phi(\sqrt{u}) = \frac{d}{du}u\left(\log(\gamma) + \frac{1}{2}\log(u)\right) = \log(\gamma) + \frac{1}{2}\log(u) + \frac{1}{2}, \quad \frac{\partial r(x)}{\partial x^{(i)}} = \frac{\partial(x^{(i)} - s_k^{(i)})^2}{\partial x^{(i)}} = 2(x^{(i)} - s_k^{(i)}),$$

The composition of these functions is therefore:

$$\frac{\partial(\phi \circ r)}{\partial x^{(i)}}(c) = \left(\frac{\partial\phi}{\partial u} \circ r\right)(c) \cdot \frac{\partial r}{\partial x^{(i)}}(c) = \left(\log(\gamma) + \frac{1}{2}\log(\|x - s_k\|^2) + \frac{1}{2}\right) \cdot 2(x^{(i)} - s_k^{(i)})$$

It is obvious for every point except $x = s_k$ that the first derivative exists and is continuous. By l'Hospital it is gained

$$(x^{(i)} - s_k^{(i)}) \cdot \log\|x - s_k\|^2 = \frac{\log\|x - s_k\|^2}{\frac{1}{(x^{(i)} - s_k^{(i)})}} \rightarrow \frac{\frac{2(x^{(i)} - s_k^{(i)})}{\|x - s_k\|^2}}{-\frac{1}{(x^{(i)} - s_k^{(i)})^2}} = -\frac{2(x^{(i)} - s_k^{(i)})^3}{\|x - s_k\|^2} \rightarrow 0 \quad (x^{(i)} \rightarrow s_k^{(i)})$$

which ensures that the derivative is 0 at $x = s_k$ and is continuous. □

Two questions arise: Why do we need the polynomial term at all? And why does it need to be added as a term $P^T \omega = \mathbf{0}$ to the equation system? The answer to the first question is, that the matrix Φ might be singular, as can be seen in the following example by [62]:

Example 3.1.6

Set $\phi(r) := r^2 \log(\gamma \cdot r)$, $K := n + 1$, $\gamma := 1$, and let the candidates in C_K form a simplex where all the edges have length 1. It holds:

$$\|s_j - s_k\| = \begin{cases} 0 & \text{if } j = k \\ 1 & \text{otherwise} \end{cases}, \quad \text{with } (s_j, s_k) \in (C_K)^2$$

which means that the matrix $(\Phi)_{jk} := \phi(\|s_j - s_k\|) := \|s_j - s_k\|^2 \log(\gamma \cdot \|s_j - s_k\|)$ has only the entries 0. The singularity indicates that the interpolant is not unique. But in the space \mathbb{R}^n an interpolant of $(n + 1)$ sample pairs definitely exists if $\omega = \mathbf{0}$: It is a hyperplane in \mathbb{R}^n .

The example shows that even with distinct candidates the radial basis function interpolant with a thin plate spline as kernel is not unique without linear polynomial. Tending to the second question, why does the condition $P^T \omega = \mathbf{0}$ have to be added to the equation system when the kernel is a thin plate spline? It makes sense to add some constraints to the equation system (3.7) since otherwise it would be underdetermined. An intuitive idea is using a polynomial whose function value is more determined by those candidates s_k whose ‘‘hills’’ $\phi(\|s_j - s_k\|)$ are less weighted. It is known from linear Algebra that the kernel of a matrix P^T is orthogonal to the image of P . The image can be any polynomial with the function values Pv and the kernel is defined by $P^T \omega = \mathbf{0}$.

It is possible to establish the uniqueness with these constraints [21, Proposition 5.2], given by the following definition and corollary. If a suitable set of sample pairs is provided, then the polynomial term makes sure that the RBF is unique, and moreover, it is a minimum norm interpolant.

Definition 3.1.7

Let $f : \mathcal{D} \rightarrow \mathbb{R}$ be a real valued function on the domain with dimension $n \in \mathbb{N}$, i.e. $\mathcal{D} \in \mathbb{R}^n$. Let $C \subset \mathcal{D}$ be a set of $K > n$ distinct candidates.

1. The set of sample pairs $S = \{(s, f(s)) \mid s \in C\}$ is called *suitable* if $(n+1)$ candidates $s_1, \dots, s_{(n+1)} \in C$ exist with the following matrix being regular, i.e. not singular:

$$\mathcal{M} := \left(\begin{pmatrix} 1 \\ s_1 \end{pmatrix} \cdots \begin{pmatrix} 1 \\ s_{(n+1)} \end{pmatrix} \right)$$

2. Let $\bar{f} : \mathcal{D} \rightarrow \mathbb{R}$ be a function of the form (3.5) interpolating the given sample pairs in S . \bar{f} is called a *minimum norm interpolant* of f if for all functions $g : \mathcal{D} \rightarrow \mathbb{R}$ of the form (3.5) that interpolate all sample pairs in S , we have

$$(\bar{f}, \bar{f})_* \leq (g, g)_*.$$

Thereby, $(\cdot, \cdot)_*$ denotes a polynomial extension of the semi-norm

$$(f_1, f_2)_* = \sum_{s_1 \in C_1} \sum_{s_2 \in C_2} \lambda_{s_1} \lambda_{s_2} \phi(\|s_1 - s_2\|)$$

on the Hilbert space completion of the space of all functions of the form $f_{1/2} = \sum_{s \in C_{1/2}} \lambda_s \phi(\|\cdot - s\|)$. Compare [21, Equation (5.8)] and thereafter.

The following corollary is a special case of [21, Proposition 5.2]. If a suitable sample pair set is provided, then the polynomial condition (3.6) makes sure that the RBF is unique and minimizing the semi-norm above.

Corollary 3.1.8

Let the kernel ϕ of the RBF in Definition 3.1.2 be a thin plate spline. Also, let the domain $\mathcal{D} \subset \mathbb{R}^n$ of the function f of the same definition be compact, have a Lipschitz-continuous boundary $\partial \mathcal{D}$ and a non-empty interior. Let the polynomial be of degree 1: $p \in \Pi_1^n$.

If S_K as defined in Definition 3.1.2 is suitable, then the RBF that satisfies (3.5) and (3.6) exists, is unique and is a minimum norm interpolant.

Proof. If the matrix Φ in (3.7) was positive definite, all eigenvalues would be positive, thus the matrix would be regular. [21] has shown before Theorem 2.2. that for a completely monotonic, non-constant $\phi(\sqrt{\cdot})$ and distinct candidates the matrix Φ is indeed positive definite. But in the Example 3.1.6 this is not the case. The thin plate spline as a kernel is not strictly monotonical. Thus, the matrix Φ is not necessarily positive definite. The following two definitions help to show the corollary:

1. The conditional positive definiteness [21, Definition 5.1] of the kernel $\phi(\sqrt{\cdot})$ only demands that Φ is positive definite on our constraints (the subspace defined by the constraints): A function $\phi : \mathbb{R}^n \rightarrow \mathbb{R}$ is *strictly conditionally positive definite* of order 2 on \mathbb{R}^n if for all finite subsets $S \subset \mathbb{R}^n$ with $K \in \mathbb{N}$ candidates the quadratic form

$$\sum_{s_1 \in C} \sum_{s_2 \in C} \omega_{s_1} \omega_{s_2} \phi(\|s_1 - s_2\|)$$

is positive for all tuples $\omega \in \mathbb{R}^K$ which satisfy $\omega^T P^T = \mathbf{0}$ for the matrix P of the polynomial basis' in Π_1^n . Instead of the monotony of the function $\phi(\sqrt{u})$ only the monotony of a derivative of this function is demanded: From the first derivative in the proof of Lemma 3.1.5 you can derive $\frac{d^2}{(du)^2} \phi(\sqrt{u}) = \frac{1}{2u}$ with (3.8) as a kernel ϕ . It is completely monotonic for $u > 0$. Thus, from [21, Theorem 5.1] follows that ϕ is strictly conditionally positive definite of order 2.

2. The unisolvency is a property that makes sure that the only polynomial that vanishes on C is $\mathbf{0}$: *Unisolvent* for Π_1^n means that for all $q \in \Pi_1^n$ the condition $q|_C = 0$ implies $q \equiv 0$. Why does S_K contain a unisolvent subset of candidates C ? Now, consider an arbitrary polynomial $q \neq 0$ in Π_1^n : One basis of this space is $\{p_0(x) = 1, p_i(x) = x_i, i = 1, \dots, n\}$. Therefore, q can be expressed by $q(x) = \sum v_i \cdot p_i(x)$ with the weights $v_i, i = 0, \dots, n$, uniquely. The matrix \mathcal{M} is regular, which means the linear equation system $b = \mathcal{M}^T \cdot v$ with weights $v = (v_0 \ \dots \ v_n)$ and an arbitrary $(n+1)$ -vector b is uniquely solvable, and in particular for $q|_C = b = 0$. The fact that the trivial solution $v = 0$ always exists leads to the conclusion that the trivial solution is the only solution. The polynomial is $q \equiv 0$. Which means that C is unisolvent for $\Pi_{m=1}^n$.

For the uniqueness of the solution to the Equation System (3.7), we will show $(\omega, v) \neq \mathbf{0} \Rightarrow F \neq \mathbf{0}$: On the one hand, ϕ is strictly conditionally positive definite (1) on the constraints $P^T \omega = \mathbf{0}$, so

$$\langle \underbrace{\Phi\omega + Pv}_{=F}, \omega \rangle = \langle \Phi\omega, \omega \rangle + \langle v, \underbrace{P^T\omega}_{=0} \rangle = \omega^T \Phi\omega > 0$$

holds. Thus, if $\omega \neq \mathbf{0} \Rightarrow F \neq \mathbf{0}$. Thus, the only possibility to have a non-zero argument is $v \neq \mathbf{0}$. On the other hand, the unisolvency (2) means that the only polynomial vanishing on C is the $\mathbf{0}$ polynomial. However, from $\mathbf{0} = F = \underbrace{\Phi\omega}_{=0} + Pv$ and the unisolvency follows $v = 0$, which is a contradiction.

Proposition 5.2 of [21] states that if the set of candidates C_K of S_K contains an unisolvent subset $C \subset C_K$ for Π_1^n and if the kernel $\phi(\sqrt{\cdot})$ is strictly conditionally positive definite of order 2 then the RBF that satisfies (3.5) and (3.6) is the minimum norm interpolant.

□

The response surface model we introduced in this section has particularly interesting properties: It interpolates scattered sample pairs in a smooth (continuously differentiable) way and minimizes a semi norm thereby. Additionally, it is the only RBF that minimizes this norm.

3.1.3 Update Rules and the Incorporation of Prior Information

Without an optimization procedure, it is reasonable to place one camera on a wall of the environment facing the interior of the room (not the wall) in order to maximize its coverage. Firstly, this holds since the camera can only see in one direction, and secondly, orienting the camera to the wall would lead to no coverage at all. In fact, most of the sensors have properties that can be exploited when maximizing coverage. Cameras and depth sensors cannot penetrate walls, contact sensors need to establish contact with an object, some sensors might have a limited range or depth of field, some a limited field of view, etc.

Instead of utilizing an imprecise heuristic for optimizing the network, this knowledge can be incorporated in the response surface model. For Algorithm 1, two categories of incorporating knowledge have been investigated in this thesis: One concerning the provision of prior information when starting the algorithm (Line 2, Algorithm 1), and one considering advanced information (Line 13 and Equation (3.4)). These two categories are addressed in the following definition.

Definition 3.1.9

Let $\mathcal{D} := \mathbb{R}^n$ be a set and let $\mathcal{P}(\cdot)$ denote the power set. Let $f : \mathcal{D} \rightarrow \mathbb{R}$ be a real-valued function and $G_f := \{(s, f(s)) \mid s \in \mathcal{D}\}$ the graph of f .

1. A function $\chi : \mathcal{P}(G_f) \rightarrow \mathcal{P}(G_f)$ is called *update rule for f* if $\chi(S)$ is a suitable set of sample pairs for all $S \subset G_f$.
2. Then $\chi(S)$ is called *advanced* if S is suitable and *basic* if S is not suitable.
3. The update is called *additive* if for every $S \subset G_f$ a set $\bar{S} \subset G_f$ exists with $\chi(S) = S \cup \bar{S}$.
4. The *effect* of an update rule χ is defined as the function $e_\chi(S) = |\chi(S) \setminus S|$
5. The function $\alpha_f : \mathcal{P}(G_f) \rightarrow \mathcal{P}(G_f)$ is called *prior information about f*

$$\alpha_f(S) = \{(x, f(x)) \in G_f \setminus S \mid f(x) \text{ is known from the tuple } (S, f) \text{ without evaluation of } f\}$$

6. Let α be the prior information. Let $\bar{\chi}$ be an update rule. The function $\chi : \mathcal{P}(G_f) \rightarrow \mathcal{P}(G_f)$ with the following property is called the *combined update χ with $\bar{\chi}$ and α*

$$\chi(S) = \bar{\chi}(S) \cup \alpha(\bar{\chi}(S)), \quad \forall S \subset G_f.$$

An RBF can be constructed with the given sample pairs if the initial sample pairs are suitable, as shown in Corollary 3.1.8. An update rule ensures that the resulting sample pairs are suitable. The following trick is useful to generate n suitable candidates of dimension n from a single initial candidate of the interior of the domain $\mathcal{D} \subset \mathbb{R}^n$ if the domain is bounded by box constraints only.

Example 3.1.10

Let $\mathcal{D} \subset \mathbb{R}^n$ be bounded by box constraints: Let $s_0 \in \mathcal{D}$. For the i th candidate s_i duplicate the initial candidate and add a scalar $\gamma(s_0^{(i)})$ to the i th coordinate $s_0^{(i)}$. The addend $\gamma(s_0^{(i)})$ is positive if the initial candidate is nearer to the lower domain boundary than the upper boundary and otherwise negative. $\gamma(s_0^{(i)})$ is set to a quarter of the distance between upper and lower domain boundary in the same coordinate. Let

$$s_0 := \begin{pmatrix} s_0^{(1)} \\ \vdots \\ s_0^{(n)} \end{pmatrix}, s_1 := \begin{pmatrix} s_0^{(1)} + \gamma(s_0^{(1)}) \\ \vdots \\ s_0^{(n)} \end{pmatrix}, \dots, s_n := \begin{pmatrix} s_0^{(1)} \\ \vdots \\ s_0^{(n)} + \gamma(s_0^{(n)}) \end{pmatrix}.$$

The resulting set of sample pairs is $S(s_0) := \{(s_k, f(s_k)) \mid k \in \{0, \dots, n\}\}$. With this rule, the matrix of Definition 3.1.7 (4) is regular and by construction every candidate is in the domain. Thus, the sample pairs are suitable. These are the conditions of Corollary 3.1.8 for the RBF to be unique and to exist. The function $\chi : \mathcal{P}(G_f) \rightarrow \mathcal{P}(G_f)$ with

$$\chi(\bar{S}) = \bigcup_{(s, f(s)) \in \bar{S}} S(s),$$

for all $\bar{S} \subset G_f$ is an update rule. You can use χ with suitable sets S , although you would usually use it with sets of $|\bar{S}| = 1$. In the latter case, the update rule is basic, since S is not suitable. The latter update rule is additive since the original sample pair $(s_0, f(s_0))$ is included in $S(\cdot)$. Its effect is $e_\chi(\{(s_0, f(s_0))\}) = n$.

This is a basic update rule. However, there is no evidence that it leads to a converging algorithm if used as an advanced update. Revisiting Algorithm 1 the basic additive update rule can be found in Line 2, while the advanced additive updates are in Line 12. The formulation for the latter update rule is

$$\bar{\chi} : S \mapsto S \cup \left\{ (s, f(s)) \mid s \in \operatorname{argmax}_{x \in \mathcal{D}} \left\{ \bar{f}(x), \|x - \bar{s}\| \geq \beta \Delta, \forall \bar{s} \in C \right\}^{(*)}, \text{ with } C \text{ such that } S = C \times f(C) \right\}. \quad (3.9)$$

In case several maxima of the update rule (3.9) exist, only one is chosen (*). Thus, the effect is 1.

In the context of a costly objective function f , the evaluation of the f should be avoided in points of the domain where the function value is already known. This is why we have introduced the notion of prior information. Prior information are sample pairs that are gained without evaluating the objective function f , again. We have established an update rule that specifically incorporates prior information into an update rule.

Lemma 3.1.11

Let α be the prior information. Let $\bar{\chi}$ be an additive update rule. The combined update χ of $\bar{\chi}$ and α is an additive update rule. Moreover, if $\alpha(\bar{\chi}(S)) \not\subset \bar{\chi}(S)$ for a particular S , χ has a higher effect than $\bar{\chi}$.

Proof. χ is an update rule since all sample pairs are distinct, and a suitable subset of sample pairs can already be found in $\bar{\chi}(S)$. The additivity is inherited from $\bar{\chi}$. Why does it have a higher effect? It holds

$$e_\chi(S) = |(\bar{\chi}(S) \cup \alpha(\bar{\chi}(S))) \setminus S| = \underbrace{|\bar{\chi}(S) \setminus S| + |\alpha(\bar{\chi}(S)) \setminus S| - |(\bar{\chi}(S) \cap \alpha(\bar{\chi}(S))) \setminus S|}_{>0} > |\bar{\chi}(S) \setminus S|$$

□

The effect of a combined update rule $\bar{\chi}$ is higher than the effect of its base update rule χ . The effect measures the number of gained sample pairs independent on the number of times the objective function has to be evaluated. It remains to be said that the sample pairs of prior information do not increase the number of objective function evaluations, compare Definition 3.1.9 (5.-6.). The type of prior information can be distinguished by the number of iterations in which the effect is larger than 0. If the effect is larger than 0 once, the pieces of prior information are constant with the set of sample pairs S , such as in the following example in camera placement.

Example 3.1.12

Let us consider the problem (1.2) of $N = 1$ camera with limited field of view and a viewing angle smaller than π , in a closed box constrained surveillance zone, w.l.o.g. the unit cube with walls at $x_1 \in \{-1, 1\}$, ..., $x_3 \in \{-1, 1\}$. The objective function is denoted by the volume of the coverage λ . Assume that the set of network parameters are such that orientation is given as a vector in direction of the camera plane normal,

and the position is in $[-1, 1] \times [-1, 1] \times [-1, 1]$. Let the sensor label that is used to define the coverage simply be “detectable”, which means we want to maximize the area λ that is detectable by this camera. Then the following can be considered prior information to any additive update rule:

$$\alpha_\lambda(S) := \left\{ (s, 0) \mid s = (1, 0, 0, 1, 0, 0)^T, (0, 1, 0, 0, 1, 0)^T, (0, 0, 1, 0, 0, 1)^T \right\}, \quad \forall S \subset G_\lambda$$

The assumption $\alpha_\lambda \subset G_\lambda$ holds since the camera at position $(1, 0, 0)^T$ is situated at a wall and is oriented with $(0, 0, 1)^T$ towards the wall.

This can also be transferred to $N \in \mathbb{N}$ cameras when assuming that a region is meant to be covered by all cameras as in Definition 1.2.1: The coverage $\sigma_s^{-1}(\{\text{detectable}\})$ of a candidate of any of the above sample pairs $(s, 0) \in \alpha_\lambda$ is the empty set, which means the intersection C of any of these coverages is empty again, the objective is therefore known to be zero, as well.

These pieces of information can be helpful only once. The effect of this Example is 3 in the beginning of the Algorithm, but as soon as the sample pairs have been incorporated to the response surface model, the effect is 0 in the following iterations. In order to gain prior information that increases with the set of samples S , some properties of the function must be exploited. This can be symmetry of the function, boundary conditions, or other properties that depend on the already evaluated set of candidates.

The question of how to exploit symmetry in objective functions has been an active research topic in the past decade. One strategy is to include *symmetry breaking* constraints, cf. [118], which limit the domain to a non-symmetric part of the search space. However, recently a “negative effect of symmetry breaking constraints on the local search performance” has been reported, cf. [117]. Using the prior information by exploiting the symmetry of a function as in Algorithm 1 avoids these drawbacks, since no symmetry breaking constraints need to be implemented. In the following, an update rule exploiting symmetry of the objective function is defined and its effect is calculated.

Definition 3.1.13

Let \mathcal{D} be such that there exist a decomposition $\mathcal{D} = V_1 \times \dots \times V_M$ into $M \in \mathbb{N}$ subspaces with two subspaces $m, \bar{m} \in \{1, \dots, M\}$ being isomorph: $V_m \sim V_{\bar{m}}$. Additionally, let $f : \mathcal{D} \rightarrow \mathbb{R}$ be a real function, symmetric in these two subspaces as in Definition 2.4.4 of Section 2.4.3. Let $[\cdot]$ denote the subspace coordinates as in the same definition and let $S \subset G_f$ be a set of sample pairs from the graph G_f be its graph. Then the following is called *symmetric prior information*:

$$\alpha_f(S) := \left\{ \left([x_1, \dots, x_{m-1}, x_{\bar{m}}, x_{m+1}, \dots, x_{\bar{m}-1}, x_m, x_{\bar{m}+1}, \dots, x_M], f(x) \right) \subset G_f \mid [x_1, \dots, x_M] = x \in S \right\}$$

Figure 3.3 shows that the incorporation of prior information, such as symmetry, can be rewarding.

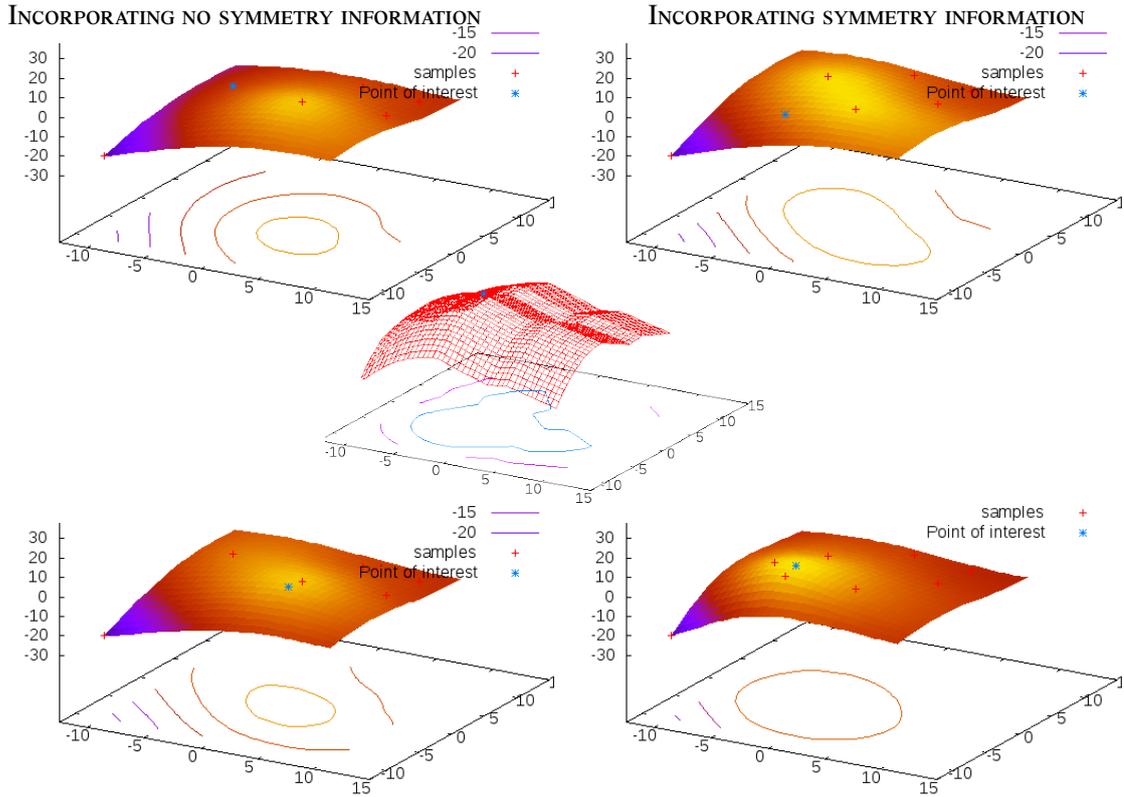


Figure 3.3: Illustration of an update rule incorporating symmetry as prior information (right) and not incorporating symmetry (left) when optimizing the original objective function (middle) of Figure 3.12; The rows show the response surface models of both strategies after five and six costly function evaluations. With the incorporation of symmetric information the response surface model is closer to the original.

Lemma 3.1.11 shows that the effect of a symmetric update rule is usually higher than without incorporation of symmetry.

Lemma 3.1.14

Let \mathcal{D} be such that there exist a decomposition $\mathcal{D} = V_1 \times \dots \times V_M$ into $M \in \mathbb{N}$ subspaces with $V_1 \sim V_2 \sim \dots \sim V_M$. And let $\bar{\chi}$ be an additive update rule.

If no sample pair is updated twice, then a combined update rule $\bar{\chi}$ in Lemma 3.1.11 with symmetric prior information is $M!$ times more effective.

Proof. = The effect $|\bar{\chi}(S) \cup \alpha_f(\bar{\chi}(S)) \setminus S|$ of a symmetric pair of subspaces is 2 if none of the two samples in $\bar{\chi}(S)$ and in $\alpha_f(\bar{\chi}(S))$ have already been updated. Since f is symmetric on M subspaces, the a priori information includes all $M! - 1$ permutations from the solution x excluding the identity. Then, for all sets $S \subset G$ the effect of the update rule in Lemma 3.1.11 with the above piece of prior information

is

$$\begin{aligned}
e_\chi(S) &= |\chi(S)| = |(\bar{\chi}(S) \setminus S) \cup \underbrace{(\alpha_f(\bar{\chi}(S)) \setminus S)}_{=\alpha_f(\bar{\chi}(S))(**)}| \\
&= \underbrace{|\bar{\chi}(S) \setminus S|}_{e_{\bar{\chi}}(S)} + \underbrace{|\alpha_f(\bar{\chi}(S))|}_{(M!-1) \cdot e_{\bar{\chi}}(S)(*)} - \underbrace{|(\bar{\chi}(S) \cap \alpha_f(\bar{\chi}(S)) \setminus S)|}_{=\emptyset} = M! \cdot e_{\bar{\chi}}(S)
\end{aligned}$$

Relation (*) holds because the update rule χ is additive. Thus, any sample pairs symmetric to the sample pairs in S are included in S already. Relation (**) holds since $\bar{\chi}$ additive and is not included in α_f . \square

The symmetry is also important for camera placement. When optimizing the positions of $N \in \mathbb{N}$ identical cameras, each with a set of parameters in the same space \mathbb{P} the objective function of problem (1.2) is symmetric in all the subspaces \mathbb{P} of the whole domain of the camera network $\mathcal{D} := \mathbb{P}^N$. This is shown in Section 2.4.3. The result of Lemma 3.1.14 shows that updating with symmetric prior information is $N!$ times more effective than the standard update (3.9).

3.1.4 Convergence

In this section, the convergence of the Algorithm 1 to the global maximum is proved. The proof is inspired by [122] but is adapted to the more effective updates as in Lemma 3.1.11 and using the prior information of Lemma 3.1.14. Also, a brief summary of an error estimate between an RBF as a response surface model and the actual costly objective is provided. Details can be found in [21]. Furthermore, we will discuss whether the convergence of the response surface model to the objective function plays an important role for the convergence to the global optimum. Both convergences depend on the following fact: Over time, the regions without candidates are *all* decreasing due to the constraints of Line 8 in Algorithm 1. This useful fact is proved by the following lemma:

Lemma 3.1.15

Let \mathcal{D} be bounded. The sequence Δ_K of Algorithm 1 is monotonically decreasing and converges to 0 if the update rule of Lemma 3.1.11 is utilized with the standard update rule (3.9) with or without prior information.

Proof. First, the monotony is proved for the standard update rule. Let \tilde{f}_K be the RBF that interpolates K candidates in C_K . A new candidate for the next costly function evaluation is searched in the set $\mathcal{D}_{K,\beta} := \{x \in \mathcal{D} \mid \|x - s\| \geq \beta \Delta_K, \forall s \in C_K\}$, whereas the ratio β is a ratio from the search pattern $\langle \beta_1, \dots, \beta_L \rangle$ with $\beta_1 > 0$. The next candidate s_{K+1} is taken from $\operatorname{argmax}_{x \in \mathcal{D}_{K,\beta}} \tilde{f}(x)$ and therefore satisfies

$$\|s_{K+1} - s\| \geq \beta \Delta_K = \beta \cdot \underbrace{\max_{x \in \mathcal{D}} \min_{s \in C_K} \|x - s\|}_{(*)} \quad \text{for all } s \in C_K.$$

The next candidate set is $C_{K+1} = C_K \cup \{s_{K+1}\}$. What is the difference between Δ_{K+1} and Δ_K ?

$$\begin{aligned}
\Delta_{K+1} &= \max_{x \in \mathcal{D}} \min_{s \in C_{K+1}} \|x - s\| = \max_{x \in \mathcal{D}} \min \left\{ \min_{s \in C_K} \|x - s\|, \|x - s_{K+1}\| \right\} \\
&= \begin{cases} \Delta_K, & \min_{s \in C_K} \|x - s\| < \|x - s_{K+1}\|, \\ \max_{x \in \mathcal{D}} \|x - s_{K+1}\|, & \text{otherwise.} \end{cases} \tag{3.10}
\end{aligned}$$

First of all, this means that the inequality $\Delta_{K+1} \leq \Delta_K$ holds for the standard update rule. The update rule of Lemma 3.1.11 is additive, thus additional candidates can only be added and none is removed. The exclusion area Δ_{K+1} is even smaller, if not equal.

In the context of Theorem 2 of [122], a proof is given for a sequence $(s_K)_{K \in \mathbb{N}}$ with the following property to be dense: For some fixed $0 < \beta$, a strictly increasing sequence $\{K_t\}_{t \in \mathbb{N}}$ of positive integers exists such that the sub-sequence $(s_{K_t})_t$ satisfies

$$\min_{k=1, \dots, K_t-1} \|s_{K_t} - s_k\| \geq \beta \max_{x \in \mathcal{D}} \min_{k=1, \dots, K_t-1} \|x - s_k\|.$$

The idea of the proof is to assume that (s_t) is not dense, i.e., there exists a point with an open $\delta\beta$ -neighborhood that does not contain any elements of the sequence. The bounding box of the domain with the side length $r\delta\beta/(2\sqrt{n})$ of some positive integer $r \in \mathbb{N}^+$ is then divided into sub-hypercubes of side length strictly smaller than $\delta\beta/\sqrt{n}$, e.g. $\delta\beta/(2\sqrt{n})$. The sub-sequence (s_K) is infinitely long. Together with the above property of the sub-sequence it is implied that each sub-hypercube (only a finite number exists) is occupied by at least one member of the sequence. One sub-hypercube has a diameter of less than $\sqrt{n} \cdot (\delta\beta/\sqrt{n})^2 = \delta\beta$ by construction. Thus, no such δ -neighborhood can be found. With the sequence of candidates being dense, the sequence Δ_K must then converge to 0.

Adding any sample pairs can only densify the samples and thereby improve the convergence. Thus, the additive update rule of Lemma 3.1.11 with the standard update (3.9) and other prior information will equally force the sequence Δ_K to converge, either. Additionally, if this is an update based on symmetry, the sequence will generally converge faster, since the non-symmetric domain is smaller. \square

Knowing that the candidate-free space is decreasing monotonically in Algorithm 1, two conclusions can be drawn: On a compact domain, the convergence of the algorithm to the global optimum of any continuous function is established. Additionally, the RBF as a response surface model converges to the costly objective, given a few assumptions on the objective, as can be seen after the following theorem.

Theorem 3.1.16

Let \mathcal{D} be bounded. Let f be a bounded, continuous function on \mathcal{D} . Then the sequence $(f(s_K))_{K \in \mathbb{N}}$ of Algorithm 1 converges to the global maximum value. Additionally, Algorithm 1 is an anytime algorithm.

Proof. As the proof of Lemma 3.1.15 shows, the sequence of candidates $(s_K)_{K \in \mathbb{N}}$ is constructed dense in \mathcal{D} . Thus, for every point $x \in \mathcal{D}$ a sub-sequence $(s_{K_t})_{t \in \mathbb{N}}$ exists with $s_{K_t} \rightarrow x$, in particular also for the point of the maximum $x_0 := \operatorname{argmax}_{x \in \mathcal{D}} f(x)$. With the objective being continuous on \mathcal{D} for all convergent sequences t the values of the objective function hold $f(s_{K_t}) \rightarrow f(x_0)$, which is the maximum value of the function.

An algorithm is an *anytime* system if 1st it returns a valid solution even if interrupted at any point in time before it ends and if 2nd the solutions are iteratively improved. The first requirement is true since the initializing candidates are in the domain. So, a valid candidate can be stated any time. The second requirement holds because of Lines 14 to 16 of Algorithm 1: If a candidate is better than the last, the best candidate is updated. Now this update could take infinitely long if no better candidate is found, but in this Algorithm it does not since the iterates are generated dense, see Lemma 3.1.15. \square

In fact, on a compact domain \mathcal{D} the inverse is true, as well. The sequence is dense in \mathcal{D} if the Algorithm converges. This was provided by [153, Theorem 1.3]. Here, only the above-mentioned direction is needed.

As can be seen from the above theorem, the convergence of the response surface model to the costly objective is actually not required for the algorithm to converge to the global maximum function value. However, the response surface indicates the importance of a region for a candidate search. In contrast to a converging response surface, a function could be used that is constant on the whole domain and not changing during Algorithm 1. Then, the algorithm just picks a random candidate outside the exclusion area since they all seem equally important. Let us mention that the importance of a region could be revealed by other means than a converging response surface model. If prior information about the location of the maxima is known one could establish a function with higher values at these locations. This procedure would save the computation of the RBF. The adaption of the response surface to the objective function, on the other hand, yields the advantage that unconsidered regions are added if they prove more important than the already considered ones. In this way, the convergence of the response surface model to the costly objective function accelerates the algorithm.

The convergence of RBFs on scattered candidates, i.e., candidates not necessarily arranged in a grid, is extensively investigated in the book [21, Chapter 5.2]. First, some definitions need to be made to be able to understand the theory: As a requirement to the convergence the domain of the function that is approximated (here the costly objective) needs to have a particular shape: it must satisfy an interior cone condition, which means in colloquial terms that the corners of \mathcal{D} are always wide enough. The second requirement is that the costly objective needs to be square-integrable. Furthermore, the difference between the objective and the model depends on the shape of the objective function and its derivatives, which is given as a semi-norm. More precisely:

Definition 3.1.17

Let $(\mathcal{D}, \mathcal{A}, \mu)$ be the measure-space of $\mathcal{D} \subset \mathbb{R}^n$, $n \in \mathbb{N}$, with the Borel- σ -algebra \mathcal{A} , and the Lebesgue-measure μ . Also let $k, p \in \mathbb{N}$ with $0 < p < \infty$.

1. \mathcal{D} is said to satisfy an *interior cone condition* if for each $x \in \mathcal{D}$ a fixed vector $\xi(x) \in \mathbb{R}^n$ of unit length exists such that for a positive \bar{r} and θ the following inclusion holds:

$$\{x + \lambda \cdot \eta \mid \eta \in \mathbb{R}^n, \|\eta\| = \bar{r}, \eta \cdot \xi(x) \geq \cos \theta, 0 \leq \lambda \leq 1\} \subset \mathcal{D}.$$

2. $L^p(\mathcal{D}) := \left\{ f : \mathcal{D} \rightarrow \mathbb{R} \mid f \text{ measurable, } \int_{\mathcal{D}} |f(x)|^p d\mu(x) < \infty \right\} / \left\{ \left(\int_{\mathcal{D}} |f(x)|^p d\mu(x) \right)^{\frac{1}{p}} = 0 \right\}$ is the space of *p-integrable functions*. The quotient space denoted by / resembles that two functions are identified if they have the same function values μ -almost everywhere.
3. A *n-multi-index* α is a tuple $\alpha = (\alpha_1, \dots, \alpha_n) \in \mathbb{N}^n$ with $\alpha_1, \dots, \alpha_n \in \mathbb{N}$. Its total degree is defined as $|\alpha| := \alpha_1 + \dots + \alpha_n$, and its factorial by $\alpha! := \alpha_1! \cdot \dots \cdot \alpha_n!$.
4. D^α defines the operator of the partial derivative $D^\alpha := \left(\frac{\partial^{\alpha_1}}{(\partial x)^{\alpha_1}} \cdots \frac{\partial^{\alpha_n}}{(\partial x)^{\alpha_n}} \right)$ with the multi-index α . The derivative has a total degree of $|\alpha|$ distributed over all dimensions.

5. Let the real valued function $f : \mathcal{D} \rightarrow \mathbb{R}$ be such that D^α exists on \mathcal{D} . The semi-norm of the derivatives of a function is defined by

$$|f|_{k,\mathcal{D}}^2 := \int_{\mathcal{D}} \sum_{|\alpha|=k, \alpha \in \mathbb{N}^n} \frac{k!}{\alpha!} |D^\alpha f(x)|^2 dx$$

6. The space $D^{-k}L^2(\mathcal{D}) := \{f : \mathcal{D} \rightarrow \mathbb{R} \mid D^\alpha f \in L^2(\mathcal{D}), \forall \text{ multi-index } \alpha \text{ with } |\alpha| = k\}$ denotes the linear space of all functions whose k -th total degree distributional partial derivatives are in $L^2(\mathcal{D})$. (this also means $D^{-k}L^2(\mathbb{R}^n) \subset C(\mathbb{R}^n)$ for $k \geq 1$, cf. [21]).

With these definitions the following theorem estimates the approximation error between the costly objective f and an RBF as response surface model. The error decreases as the set of candidates becomes denser, expressed by the distance h , and as the norm of derivatives $|f|_{\frac{2+n}{2},\mathcal{D}}$ of f decreases. In our application, the objective f and its derivatives are fixed by the geometry of the environment and the intrinsic parameters of the cameras. Thus, we cannot influence the magnitude $|f|_{\frac{2+n}{2},\mathcal{D}}$ of f when applying Corollary 3.1.18. Conveniently, we have already proved that the candidates are densifying, a fact that is used in the proof of the following theorem, a specified version of [21, Chapter 5.2].

Corollary 3.1.18

Let \mathcal{D} be bounded and open with a Lipschitz-continuous boundary, and let it satisfy the interior cone condition. Let $s_1, \dots, s_K \in C_K, K \in \mathbb{N}$ be candidates, and let Δ_K be constructed as Δ in Eq. (3.2) for the candidates in C_K . Consider an RBF with a thin plate spline and a linear polynomial as the response surface model \bar{f} .

Then, the response surface model holds $\bar{f} \in D^{-\frac{2+n}{2}}L^2(\mathbb{R}^n)$ and for all integrable functions with square integrable derivatives $f \in D^{-\frac{2+n}{2}}L^2(\mathcal{D}) \cap L^p(\mathcal{D})$ the error estimate

$$\|f - \bar{f}\|_{p,\mathcal{D}} \leq c \cdot \Delta_K^{1+\frac{n}{p}} |f|_{\frac{2+n}{2},\mathcal{D}}$$

holds for any $p \geq 2$, where $c \in \mathbb{R}$ depends on n , and p , and neither on Δ_K nor f .

Proof. The book chapter 5.2 in [21] prepares for the proof of [21, Theorem 5.5], which is a more general form of this Corollary. The proof is rather technical and uses non-trivial results from functional analysis and the theory of reproducing kernel-hilbert-spaces. The changed notation is stated here:

Theorem [21, Theorem 5.5] has been specialized to the kernel of (4.4) in [21]. Be aware that the parameter k , which is used in the theorem, also corresponds to the parameter in (4.4) of [21, page 61] which satisfies $2 = 2k - n$ in this corollary. This parameter does not correspond to the k which gives the order of conditional positive definiteness in the definition of the radial basis function interpolant in the same book.

The candidate set C_K has a candidate free region measured by Δ_K , where Δ_K is converging to 0 and is monotonically decreasing (Lemma 3.1.15). Thus, there exists a K_0 with $\Delta_K < 1$ for all $K \geq K_0$. Define $h_1 := \Delta_{K_0}$ and let $h := \Delta_K$ for all additional candidates $K > K_0$, then

$$\sup_{x \in \mathcal{D}} \inf_{s \in C_K} \|x - s\| \leq h. \quad (3.11)$$

The parameter h corresponds to the same parameter in Theorem 5.5. of [21]. \square

In the last section, we have introduced a provably convergent method to optimize an objective function which is given as a black-box without gradient. The method can cope with stair-cased, non-differentiable, non-convex functions. Additionally, prior information like symmetry can be incorporated. The method stores the already evaluated sample pairs in a response surface model as not to evaluate these again and provides a gradient for the objective function. Although it is not necessary for the convergence of the solver, the response surface converges to the objective function if the latter and its first derivatives are squared integrable. The objective function for camera network optimization is substantially cheaper if only one camera is changed. In order to use the acceleration of the objective function, we alter the method above in the next section.

3.2 Optimization Procedure utilizing a Block Coordinate Ascent

We take a step back to the application of the optimization method, the placement of several cameras depicted in Problem (1.2). Manually, it is reasonable to place the cameras in turns. For one, because a single person can only attach a single camera to the wall at a time, but also for efficiency reasons: Firstly, our imagination can rather cope with estimating and optimizing the coverage of one camera than several ones, and secondly, there are fewer placement possibilities for a single camera than for several. After having placed each camera we would have to check whether the placement of the cameras is acceptable regarding each other. If not, or if an improvement is obvious, the cameras should be readjusted. Of course, this procedure might not be very efficient in real life, since a readjustment of a camera means physically detaching it from the wall and placing it somewhere else. But in the simulation of an optimization algorithm, such a strategy is promising.

Transferring the inspiration from real life to optimization, the placement options of a single camera represent a subspace of the domain. Thus, the hope emerges to be able to accelerate Algorithm 1 in the following way:

1. Decompose the domain of the objective function into orthogonal subspaces,
2. Optimize the objective function on a subspace of the domain and, following this,
3. Communicate the result to the optimization process of the next subspace.

Here, only subspaces parallel to the coordinate axes are considered, which means a group of variables of the domain is optimized en bloc, the other variables are constant. This procedure is called *block coordinate ascent/descent* (BCA/BCD), or *domain decomposition*, and here it is utilized for nonlinear optimization on the domain $\mathcal{D} \in \mathbb{R}^n$, as in [54]. Just as in real life, the acceleration is expected from a faster evaluation of the objective – the coverage of only one single camera can be simulated faster than the coverage of two or more – and, secondly, from a faster optimization due to the smaller subspaces. In the experimental Section 3.4 it will be shown whether this strategy is successful without parallelization.

This section is organized as follows: Section 3.2.1 discusses two general update rules for a BCA. Section 3.2.2 concretizes the BCA for camera placement. The challenges of a BCA in general and specified for camera placement are the main reason for Section 3.2.3. In the last Section, these challenges are overcome by marrying the BCA and the RBF solver of Section 3.1.

3.2.1 Block Coordinate Ascent

We consider the Maximization Problem (3.1). This problem is called the *global problem* further on. As in Definition 2.4.4, we apply the decomposition $\mathcal{D} = V_1 \times \dots \times V_M$, $M \in \mathbb{N}$ to the solution space \mathcal{D} with the dimension n_1, \dots, n_M , respectively, and $n = n_1 + \dots + n_M$. The corresponding partition of the identity matrix is denoted by

$$\mathbb{1}_n := (U_1, \dots, U_M) \in \mathbb{R}^{n \times n}, U_m \in \mathbb{R}^{n \times n_m}, m = 1, \dots, M. \quad (3.12)$$

Instead of solving the global problem (3.1) a sequence of smaller problems is considered iteratively. Choose an initial solution $x^{(0)} := U_1 x_1^{(0)} + \dots + U_M x_M^{(0)} \in V_1 \times \dots \times V_M$, e.g. $x^{(0)} = \mathbf{0}$. Then iterate:

Sequential iteration 3.2.1 $(i \rightarrow i + 1)$

$$\begin{aligned} u_1 &:= \operatorname{argmax}_{v_1 \in V_1} f(U_1 v_1 + x^{(i)}) \\ u_2 &:= \operatorname{argmax}_{v_2 \in V_2} f(U_2 v_2 + U_1 u_1 + x^{(i)}) \\ &\vdots \\ u_M &:= \operatorname{argmax}_{v_M \in V_M} f(U_M v_M + \sum_{m=1}^{M-1} U_m u_m + x^{(i)}) \\ x^{(i+1)} &:= U_1 u_1 + \dots + U_M u_M + x^{(i)} \end{aligned}$$

Each line of the above iteration corresponds to an optimization on a subspace. Starting from the previous solution vector $x^{(i)}$ in the first row the subspace V_1 is searched for the best solution according to the objective function f . Thereby, the vector $u_1 \in V_1$ in subspace coordinates (defined in Definition 2.4.4) resembles an *offset* between the previous solution vector $x^{(i)}$ and the best in this subspace. In the next row, the subspace V_2 is searched for the best solution starting at $U_2 u_2 + x^{(i)}$. This procedure is continued to the subspace V_M , in the last row, where the search begins at $\sum_{m=1}^M U_m u_m + x^{(i)}$. In the last step, the communication between the subspaces is made: A new solution is produced by adding the offsets of the last iteration step.

Please note that for the subscript $v_1 \in V_1$ in the first line the point $U_1 v_1 + x^{(i)}$ might not be in \mathcal{D} even if $x^{(i)} \in \mathcal{D}$. Thus, the correct notation of the subscript should actually be $U_1 v_1 \in U_1 V_1 - x^{(i)}$ but we prefer the shorter version for easier readability. This applies to the subscripts of all the subspace optimizations in the whole thesis.

The iteration 3.2.1 can also be turned into a parallel algorithm: The first subspace maximization stays the same. However, the following maximizations are equally started at the previous solution vector $x^{(i)}$ instead of the updated point $\sum_{m=1}^M U_m u_m + x^{(i)}$. The omission of the term $\sum_{m=1}^M U_m u_m$ ensures that each subspace iteration has the same starting point, namely the previous solution vector $x^{(i)}$, instead of depending on other subspace iterations. This is the basis for a parallel computation.

Parallel iteration 3.2.2 ($i \rightarrow i + 1$)

$$\begin{aligned}
u_1 &:= \operatorname{argmax}_{v_1 \in V_1} f(U_1 v_1 + x^{(i)}) \\
u_2 &:= \operatorname{argmax}_{v_2 \in V_2} f(U_2 v_2 + x^{(i)}) \\
&\vdots \\
u_M &:= \operatorname{argmax}_{v_M \in V_M} f(U_M v_M + x^{(i)}) \\
x^{(i+1)} &:= U_1 u_1 + \dots + U_M u_M + x^{(i)}
\end{aligned}$$

The Figure 3.4 shows evidence that a function type exists on which both the iterations converge to the same $x^{(i+1)}$ when starting at the same $x^{(i)}$. One iteration step $i \rightarrow i + 1$ of the Iteration 3.2.1 is illustrated to the left. The image to the right illustrates the Iteration 3.2.2. In the second method, each maximization on one subspace can be launched in a separate thread, all starting at the same solution vector, namely $x^{(i)}$.

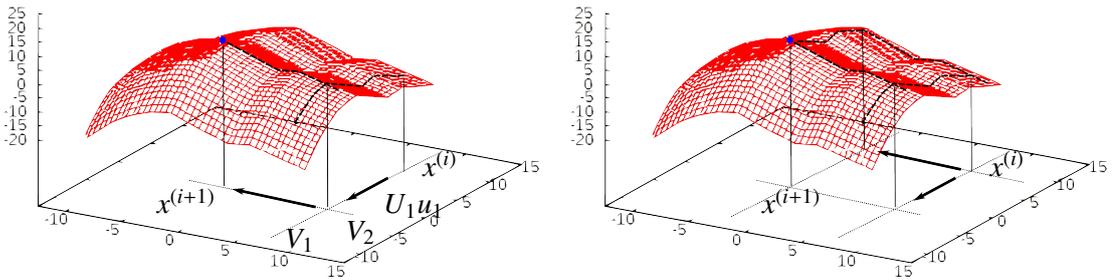


Figure 3.4: Illustration of Iterations 3.2.1 (left) and 3.2.2; The red surface is to be maximized, starting from the previous solution $x^{(i)}$. To the left, the maximization of the first subspace V_1 (dotted line) starting at the previous solution $x^{(i)}$ is illustrated by the dashed line. The maximum is the solution vector $x^{(i)}$ translated by the offset $U_1 u_1$. Starting at the new point, the subspace is switched and a new maximum is found. In the parallel method (right), each subspace maximization starts at $x^{(i)}$. After all subspaces have been processed, in both methods, the solution vector needs to be updated.

This is the basic strategy of a BCA. As we will see in the next section, the call of the objective function in Equation (1.2) is cheaper when utilizing a BCA. However, the section thereafter shows that the objective function is not necessarily a function on which the Iterations 3.2.1 and 3.2.2 converge to the same solution.

3.2.2 Cost Reduction in Camera Placement

In camera placement, the block coordinate ascent on a subspace decomposition can be applied in various ways, two of which are illustrated in the following examples.

Example 3.2.3

For the first example consider the construction of the space of camera parameters of Equation (2.1) $\mathcal{D} := \mathbb{P}^N = (E \times {}^\partial\mathbb{B}_1^3(0))^N$. One possible choice is $M := 2$ subspaces with $V_1 := E^N$ and $V_2 := ({}^\partial\mathbb{B}_1^3(0))^N$. In this case, all of the cameras are first positioned and then, in a second step, oriented. The new positions and orientations are applied to all cameras and the next iteration step starts.

However, the blocks of subspaces can also be reformed, which results in the following BCA:

Example 3.2.4

Consider the general problem of camera placement with different cameras $\mathcal{D} := \mathbb{P}_1 \times \dots \times \mathbb{P}_N$ of Equation (2.1). If we choose $M := N$ and $V_m := \mathbb{P}_m$ for $m \in \{1, \dots, M\}$ then the Algorithm 3.2.1 first places the first camera based on where all cameras have been placed in the previous iteration step. Then, the second camera is placed based on the first camera's update, the third on the second camera's update, etc. After the last camera, the method continues with the first camera, again. Algorithm 3.2.2 places each camera separately, not including the information of any updates in this iteration step. In the communication step, all the places of the cameras are updated.

The last example yields some advantages for both the sequential and the parallel version: Consider $f := \lambda$ as in Problem 1.2, and $x \in \mathbb{P}_1 \times \dots \times \mathbb{P}_N$. In each iteration step $(i + 1)$ of Iteration 3.2.2 the problem for the subspace V_m looks like

$$\operatorname{argmax}_{v_m \in V_m} f(U_m v_m + x^{(i)}) = \operatorname{argmax}_{v_m \in V_m} \underbrace{\lambda(C_{U_m v_m + x^{(i)}}(S))}_{\substack{\text{fused coverage} \\ \text{of Def. 1.2.1}}} = \operatorname{argmax}_{v_m \in V_m} \lambda \left(\underbrace{\left(\bigcup_{\substack{n=1 \\ n \neq m}}^N \sigma_{U_n^T x^{(i)}}^{-1}(S) \cup \sigma_{v_m + U_m^T x^{(i)}}^{-1}(S) \right)}_{\text{const}} \right)$$

with $S \subset \mathbb{S}$ being the chosen set of camera labels. As only the variable v_m is optimized, the camera coverages $\sigma_{U_n^T x^{(i)}}^{-1}(S)$ are constant for all cameras $n \neq m$. Consequently, if the other coverages $\sigma_{U_m^T x^{(i)}}^{-1}(S)$, $n \neq m$ are adequately stored, the cost of the evaluation is reduced. In Figure 2.8 just one of the parallel rows and the subsequent union and volume needs to be calculated.

As in the cost notation of Section 2.2.3, let us assume that calculating the coverage of one camera costs c_c , calculating the union or intersection of the camera's coverages costs c_u , and measuring a volume costs c_v . One evaluation of the fused coverage of the cameras previously had a complexity of $N \cdot c_c + c_u + c_v$. Now, within one iteration step of the Block Coordinate Ascent (BCA) the complexity becomes $c_c + c_u + c_v$. This is a motivation for continuing with designing the solver. The next section deals with the challenges that we have to overcome in camera placement.

3.2.3 Challenges Concerning the Convergence

Both the Iterations 3.2.2 and 3.2.1 demand very restrictive requirements in order to guarantee convergence. A solution vector x^* which is an optimum in one subspace is not necessarily an optimum in an affine translated subspace, let alone on the global problem. In the context of camera placement, the optimum of the volume of the first camera's coverage is not necessarily an optimum anymore if a second camera has moved, nor does it have to be an optimum of the volume of the fused coverage. These requirements and their challenges in camera network optimization are discussed in this section.

In order to see the difference of these optima, consider that for a maximizer x^* of an objective function f of the global problem (3.1) the following holds:

$$\mathbf{0} \in \operatorname{argmax}_{x \in \mathcal{D}} f(x + x^*)$$

In local optimization this corresponds to the gradient being zero at $x = \mathbf{0}$. An optimization procedure should terminate if we can assure this condition. Again, $x + x^*$ might not be in \mathcal{D} even if $x^* \in \mathcal{D}$, however again, we prefer to write the simplified subscript $x \in \mathcal{D}$ instead of $x \in \mathcal{D} - x^*$. Similarly, one is able to define the termination of the subspace maximizations with the following definition.

Definition 3.2.5

Let $f : \mathcal{D} \rightarrow \mathbb{R}$ be a real-valued function and let $V_1 \times \dots \times V_M$ be a decomposition of \mathcal{D} . Additionally let the solution $x^* \in \mathcal{D}$ be given.

1. x^* is called V_m -subspace maximum of the subspace $V_m, m \in \{1, \dots, M\}$, if

$$\mathbf{0} \in \operatorname{argmax}_{v_m \in V_m} f(U_m v_m + x^*) \tag{3.13}$$

2. x^* is called *stationary point* if it is a V_m -subspace maximum for all the subspaces $m = 1, \dots, M$.

3. In contrast to the stationary point, we call x^* a *global maximum* if $\mathbf{0} \in \operatorname{argmax}_{x \in \mathcal{D}} f(x + x^*)$.

In the upcoming paragraph, only one iteration step of 3.2.1 or 3.2.2 is discussed: We will see that there are functions for which a V_m -subspace maximum x^* stays a V_m -subspace maximum even if a coordinate of x^* is changed which does not belong to the same subspace. This is necessary for the convergence of the BCA to a stationary point in one iteration step. However, stationary points are not necessarily global maxima. This is shown in the paragraphs thereafter. Since the objective function discussed in this thesis is particularly difficult for a BCA, these challenges have to be dealt with in the rest of this chapter.

Additive Separability

Figure 3.4 suggests that the achievement of the new solution $x^{(i+1)}$ is the same for the parallel as well as the sequential version. But this is only true for objective functions that are *separable*, which means f can be decomposed into M functions ϕ_m that each depend on a single subspace coordinate:

Definition 3.2.6

A function $f : \mathcal{D} \rightarrow \mathbb{R}$ is called *additively separable* on the decomposition (as in Definition 2.4.4) $V_1 \times \dots \times V_M$ if M functions $\phi_m : V_m \rightarrow \mathbb{R}, m = 1, \dots, M$, exist with

$$f(x) = \sum_{m=1}^M \phi_m(x_m), \quad \text{where the subspace coordinates are given by } x = [x_1, \dots, x_M] \in \mathcal{D}.$$

The following corollary connects the notation of a separable function and a subspace maximum:

Corollary 3.2.7

For an additively separable function f the following holds: If x^* is a $V_{\bar{m}}$ -subspace maximum, the solution vector $y = [x_1, \dots, x_{\bar{m}-1}, x_{\bar{m}}^*, x_{\bar{m}+1}, \dots, x_M]^T$ is a V_m -subspace maximum for all subspace coordinates $x_m \in V_m$ with $m = 1, \dots, \bar{m} - 1, \bar{m} + 1, \dots, M$, since

$$\operatorname{argmax}_{v_{\bar{m}} \in V_{\bar{m}}} f(U_{\bar{m}}v_{\bar{m}} + y) = \operatorname{argmax}_{v_{\bar{m}} \in V_{\bar{m}}} \left(\sum_{\substack{n=1 \\ \bar{m} \neq n}}^M \phi_n(x_n) + \phi_{\bar{m}}(v_{\bar{m}} + x_{\bar{m}}^*) \right) = \operatorname{argmax}_{v_{\bar{m}} \in V_{\bar{m}}} (\phi_{\bar{m}}(v_{\bar{m}} + x_{\bar{m}}^*)) = \operatorname{argmax}_{v_{\bar{m}} \in V_{\bar{m}}} f(U_{\bar{m}}v_{\bar{m}} + x^*) = \mathbf{0}$$

With the property of additively separability the maximum of a function on a subspace $V_{\bar{m}}$ stays a maximum when adjusting one of the other subspace coordinates of the other subspaces $m = 1, \dots, \bar{m} - 1, \bar{m} + 1, \dots, M$. With this information, the Algorithm 3.2.1 converges within one iteration step to a stationary point if the subspace maximizations converge. Also, the procedures of Algorithm 3.2.1 and 3.2.2 come to the same result if the objective function is additively separable.

Lemma 3.2.8

Let f be an additively separable function with a single global maximum. Let $x^{(i)} \in \mathcal{D}$.

1. The result $x^{(i+1)}$ of an iteration step in both iterations 3.2.1 and 3.2.2 starting at the initial solution $x^{(i)}$ is the same. 2. $x^{(i+1)}$ is a stationary point. 3. $x^{(i+1)}$ is a global maximum.

Proof. 1. Let the solution vector $x^{(i+1)}$ be the result of an iteration step of Iteration 3.2.2:

$$x^{(i+1)} = \sum_{m=1}^M U_m \operatorname{argmax}_{v_m \in V_m} f(U_m v_m + x^{(i)}) + x^{(i)}$$

Corollary 3.2.7 states that an addition of a term $\sum_{\bar{m}=1}^m U_{\bar{m}} u_{\bar{m}}$ to the argument of the objective function f in $\operatorname{argmax}_{v_m \in V_m} f(U_m v_m + x^{(i)})$ does not change the subspace optimum, since $\bar{m} < m$. This is the solution of one iteration step of Iteration 3.2.1.

2. Furthermore, the solution from Iteration 3.2.2 of a separable function of the form $f(x) = \sum_{\bar{m}=1}^M \phi_{\bar{m}}(x_{\bar{m}})$ is the following since $(U_m v_m)_{\bar{m}} = \mathbf{0}$ for $\bar{m} \neq m$:

$$x^{(i+1)} = \sum_{m=1}^M U_m \operatorname{argmax}_{v_m \in V_m} \sum_{\bar{m}=1}^M \phi_{\bar{m}}((U_m v_m)_{\bar{m}} + x_{\bar{m}}^{(i)}) + x^{(i)} = \begin{pmatrix} \operatorname{argmax}_{v_1 \in V_1} [\phi_1(v_1 + x_1^{(i)}) + \overbrace{\sum_{\substack{\bar{m}=1 \\ \bar{m} \neq 1}}^M \phi_{\bar{m}}(x_{\bar{m}}^{(i)})}] \\ \vdots \\ \operatorname{argmax}_{v_M \in V_M} [\phi_M(v_M + x_M^{(i)}) + \underbrace{\sum_{\substack{\bar{m}=1 \\ \bar{m} \neq M}}^M \phi_{\bar{m}}(x_{\bar{m}}^{(i)})}_{\text{irrelevant}}] \end{pmatrix} + x^{(i)}$$

The following is true:

$$\operatorname{argmax}_{y \in V_m} f(U_m y + x^{(i+1)}) = \operatorname{argmax}_{y \in V_m} \phi_m(y + x_m^{(i+1)}) = \operatorname{argmax}_{y \in V_m} \phi_m(y + \operatorname{argmax}_{v_m \in V_m} \phi_m(v_m + x_m^{(i)}) + x_m^{(i)}) \ni \mathbf{0}$$

So, $y = \mathbf{0}$ maximizes ϕ_m for all $m = 1, \dots, M$, which makes $x^{(i+1)}$ a stationary point.

3. Now, let $y \in \mathcal{D}$ in $f(y + x^{(i+1)}) = \sum_{m=1}^M \phi_m(y_m + x_m^{(i+1)})$. Since $x_m^{(i+1)}$ maximizes ϕ_m it follows that $y = \mathbf{0}$ maximizes $f(y + x^{(i+1)})$. Thus, $x^{(i+1)}$ not only is a stationary point but also a global maximum. \square

The previous lemma shows that both Iterations come to the same result for additively separable functions. This result is a global maximum. Unfortunately, in camera placement the objective is not necessarily additively separable:

Example 3.2.4 (again):

After having found the best place for camera one, an equally designed second camera would independently be placed at the same spot. But this is not the best place for the second camera taking the placement of the first camera into account, as their fused coverage is larger if they are further apart. Here, the additive separability depends on the possible positions of the cameras \mathbb{P} and the geometry of the surveillance zone \mathbb{A} : The fused coverage of two cameras placed in two separate rooms is indeed separable in the above sense.

With this example we have seen that the volume of the k -reliable coverage may be separable, but in important cases is not. Thus, the BCA will possibly not converge to a maximum in one iteration step. Nevertheless, we have seen in Section 3.2.2 that a BCA yields some advantages for camera placement, so, the BCA needs to be further investigated as an iteration with more than one iteration step.

Stationary Points are not necessarily Global Maxima

Using additively separable functions, an obtained stationary point is always a global maximum. The objective function of camera placement is not necessarily additively separable. In this section, we reveal the issue of obtaining a stationary point in the interior of the domain \mathcal{D} that is not a global maximum. Let us motivate the problem at hand, first.

Example 3.2.9

In order to illustrate the problem of Iteration 3.2.1 consider two functions that are slowly linearly ascending on the diagonal from $(-10, -10)$ to $(10, 10)$. One has a parabolic shape in the orthogonal direction and one is piecewise linear, as can be seen in the first plots of Figure 3.5 and 3.6.

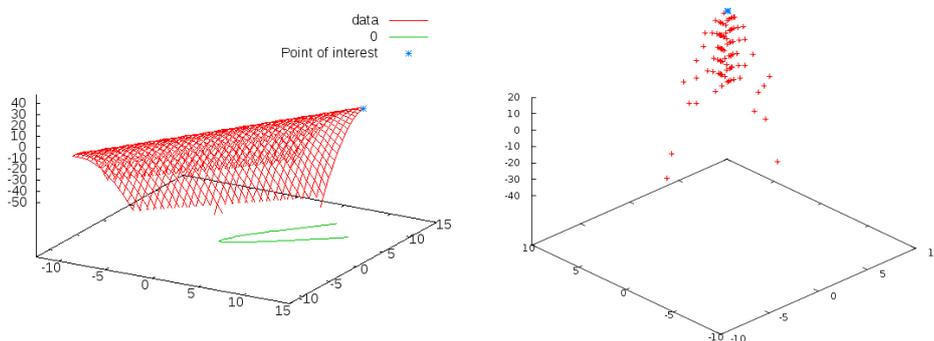


Figure 3.5: Left: A differentiable function that is slowly, linearly ascending from $(-10, -10)$ to $(10, 10)$ (left, red). The maximum is illustrated by a blue star. Right: Optimizing this function with Algorithm 3.2.1. Note that the graph is rotated by 90 degrees. The intermediate solution vectors (red crosses) are slowly oscillating all the way up to the maximum (blue star)

In the second pictures of these figures the intermediate and result solution vectors of the BCA are illustrated. While the BCA is slowly oscillating its way up to the top of the objective function in the differentiable case, the BCA in the second case is stuck in the same orthogonal subspaces over and over again. The reason: In the second example, the procedure has found a stationary point in the intersection of these subspaces, but this one obviously differs from the global maximum.

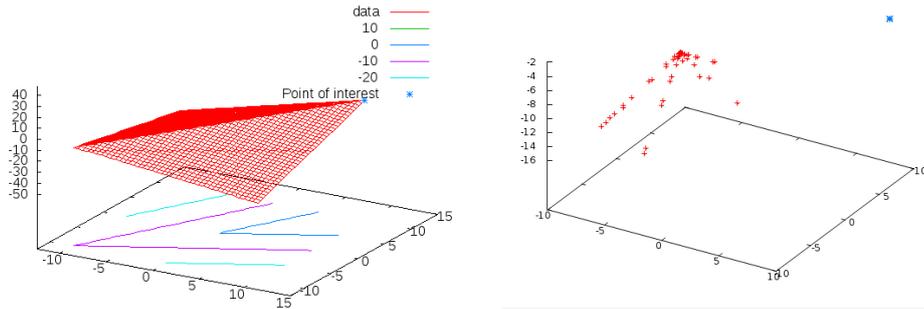


Figure 3.6: Left: A non-differentiable function that is slowly, linearly ascending from $(-10, -10)$ to $(10, 10)$ (left, red). The maximum is illustrated by a blue star. Right: Optimizing this function with Algorithm 3.2.1. The intermediate solution vectors (red crosses) stay in the same subspaces after the BCA has found a stationary point which is not a global maximum (blue star).

The question is now whether there are any differentiable, stationary points which are not global maxima. The optimality of a solution at a differentiable point of the objective function f is indicated by a vanishing gradient of the function. Generally, the gradient of a differentiable function at an interior point of the domain is zero if and only if the point is a local optimum of the function or a saddle point. We would like to be able to have a corresponding statement for the subspace V_m with a suitable gradient:

Definition 3.2.10

Let (V_1, \dots, V_M) be a decomposition of \mathcal{D} for a function f which is differentiable at a point $x^* \in \mathcal{D}$.

1. The V_m -subspace gradient of f is defined as $\nabla_m f(x^*) := U_m^T \nabla f(x^*)$.
2. In contrast to the subspace gradient we call ∇f the *global gradient* of f .

The problem illustrated in the Figures 3.6 and 3.5 is a result of the non-differentiabilities: If f was differentiable at the point of interest then either all the subspace gradients are 0 or no local optimum of the global problem has been found. This result is proven with the following lemma.

Lemma 3.2.11

Let f be a function which is differentiable at point $x^* \in {}^i\mathcal{D}$, i denoting the interior of \mathcal{D} . Then

$$\nabla_m f(x^*) = \mathbf{0} \text{ for all subspaces } m = 1, \dots, M \quad \Leftrightarrow \quad \nabla f(x^*) = \mathbf{0}$$

Proof. With the partition of the identity-matrix $\mathbb{1}_n := (U_1, \dots, U_M) \in \mathbb{R}^{n \times n}$ from Equation (3.12), the gradient holds

$$\mathbf{0} = \nabla f(x^*) = \mathbb{1}_n^T \nabla f(x^*) = \begin{pmatrix} U_1^T \\ \vdots \\ U_M^T \end{pmatrix} \nabla f(x^*) = \begin{pmatrix} \nabla_1 f(x^*) \\ \vdots \\ \nabla_M f(x^*) \end{pmatrix}$$

which shows the claimed equivalence at differentiable points of the domain. \square

Therefore, if a function is differentiable at a solution vector x the stationary point corresponds to a local optimum or a saddle point. If a direction exists in which the solution can be improved, the stationary point is not a saddle point. Thus, the issue of Figure 3.6 in Example 3.2.9 results from the non-differentiabilities, since the direction $(1, 1)$ improves the solution.

Now, one could argue that the non-differentiabilities of the objective f are so rare that this problem will not cause harm. However, in the concrete case of the placement of several cameras this issue does occur, as the following example shows. Thus, we need to figure out a way to smoothen the objective function when dealing with non-differentiable functions.

Example 3.2.4 (again):

Let us consider the space $\mathcal{D} = \mathbb{L}^2$ where \mathbb{L} is the set of possible locations for one camera, in Figure 3.7 depicted by the grey line. Also, let the orientation of a camera always be such that the camera always faces the middle of the environment directly, as illustrated by the black cross. Thus, the domain is two dimensional.

Objective function Maximize the volume $\lambda(\cdot)$ of the intersection of the field of view of both the cameras. A global maximum of λ is roughly such that both the cameras are situated where the left camera is right now:

$$\operatorname{argmax}_{a_1, a_2 \in \mathbb{L}} \lambda(a_1, a_2) = (x_1, x_1).$$

Let us place the cameras by the Iteration 3.2.1 starting with the placement of the left camera in the subspace \mathbb{L} . The camera moves over to the right camera, since the intersection is at a maximum there, thereby reaching the point (x_2, x_2) . In a second step the right camera should be readjusted but the intersection of both cameras is already at a subspace maximum.

The algorithm has reached a stationary point but it has not reached the global maximum. In fact, from point (x_2, x_2) , there exists a direction in the domain on which the solution can be improved, namely $(1, 1)$. Thus, the stationary point cannot be anywhere near a *local* maximum. Placing the cameras in the subspace of $(1, 1)$ means moving them simultaneously.

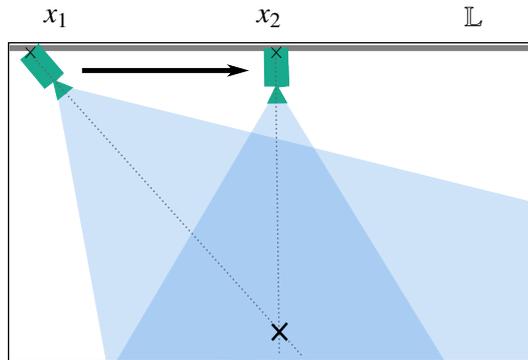


Figure 3.7: Illustrating a stationary point, not a global maximum: The domain is built by two spaces of camera locations $\mathcal{D} = \mathbb{L}^2$ (grey line). Each camera always faces the cross in the middle of the environment, thus the domain is two dimensional. Placing the cameras such that the intersection of the field of view of both the cameras is maximized by Iteration 3.2.1 starting with the left camera reaches a stationary point (x_1, x_1) which is not a global maximum.

If a regular BCA converges, it reaches a particular type of maximum. This type of maximum is called stationary point in this thesis. On a particular type of function, the additively separable functions, a BCA reaches the stationary point in one iteration step. On this function type, a stationary point corresponds to a global maximum. If a function is differentiable, a stationary point on an inner point of the domain corresponds to a local maximum or a saddle point of the domain. If the objective function is not differentiable, stationary points can additionally be caused by non-differentiable points. Unfortunately, the objective function in Equation (1.2) is neither separable nor differentiable everywhere. In the following sections, a BCA is introduced that converges nevertheless.

3.2.4 Block Coordinate Ascent for Non-differentiable, Non-separable Functions

Due to the non-differentiability and non-separability of the objective (1.2), a sequential or parallel BCA as in Iteration 3.2.1 or 3.2.2 is not reasonable. The non-separability hinders the algorithm to find good positions in one step, and due to the non-differentiability the algorithm may not even converge to a local optimum or saddle point. Thus, we first need to smoothen the objective function to ensure convergence.

One possibility is to use a Radial Basis Function solver as depicted in Section 3.1. Two ideas seem promising: Utilizing an RBF solver in order to optimize the subspace maximizations of the BCA Algorithm 3.2.2, or utilizing a BCA in order to optimize the candidate search of the RBF solver. We would like to be able to distribute the computation of the costly objective function to several cores. Therefore, it is tempting to use an RBF solver in order to solve the subspace maximizations of the iteration, as shown in Algorithm 2 from Line 7 to 15.

Algorithm 2 A block coordinate ascent method using an RBF solver in order to solve the subspace maximizations

Require: As in Algorithm 1

```

1: Update RBF as in Algorithm 1, Line 1 and 2
2:
3: while BCA termination condition do
4:   for all subspaces  $V_m, m = 1, \dots, M$  do
5:
6:     while RBF-solver termination condition do
7:       for all  $\beta$  in the search pattern  $\langle \beta_1, \dots, \beta_L \rangle$  do
8:          $\Delta \leftarrow \max_{x \in V_m} \min_{1 \leq k \leq K} \|x - s_k\|$ 
9:         Maximize the surrogate  $\tilde{f}(x)$ 
10:        Subject to
11:         $\|x - s_k\| \geq \beta \Delta, k = 1, \dots, K$ 
12:         $x \in V_m$ 
13:
14:        Costly function evaluation and updates as in Algorithm 1, from Line 11 to 18
15:      end for
16:    end while
17:
18:  end for
19: end while

```

For this method, two types of response surfaces were envisaged: A separate RBF for each subspace, or a single global RBF. Unfortunately, the constraints of the candidate search of the RBF solver (Lines 8, 9 of Algorithm 1) neither ensure that the procedure nor that the radial basis function as a response surface model converges. The problem for the Algorithm 2 is that a new candidate is chosen only out of the current subspace not from the global domain. Therefore, the convergence of neither the method, nor the RBF is guaranteed. This is true no matter whether a global RBF or separate subspace RBFs are used.

What happens if we use Algorithm 2 with a global RBF is illustrated in Figure 3.8. As an objective function (left upper corner) the roof top function known from Figure 3.6 is chosen. The RBF at the moment of the termination of the procedure is illustrated as a colored hypersurface from blue to yellow, where yellow indicates the largest function value. The intermediate solutions (red crosses) correspond to candidates of the RBF. At the moment of termination the RBF features a maximum (5, 0) (at the blue star) in the interior of the domain, but the global maximum of the objective function is at the boundary (10, 10).

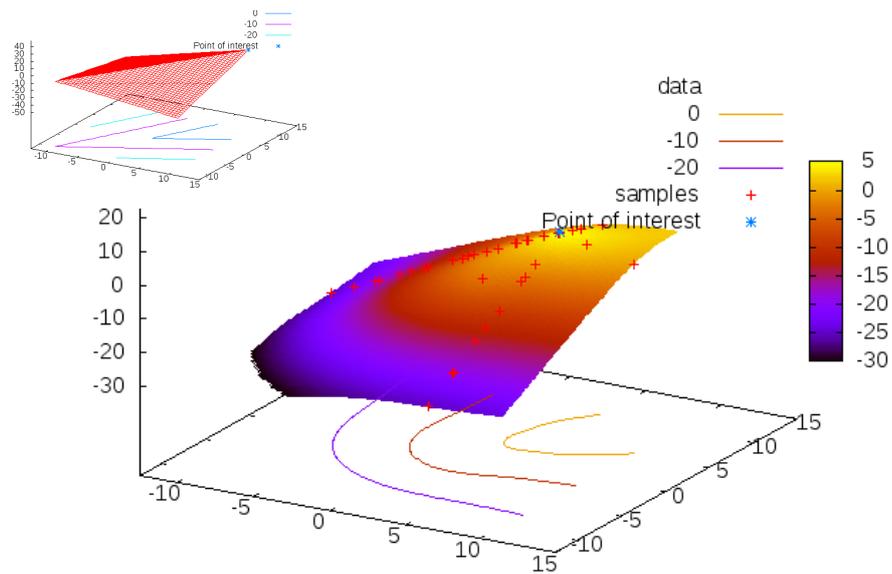


Figure 3.8: Illustration of Algorithm 2; The objective function (left upper corner, Figure 3.6) is non-differentiable and linearly ascending from (-10,-10) to (10,10). The RBF at the moment of the termination is shown as colored surface, where yellow corresponds to the largest function value. The intermediate solutions (red crosses) correspond to candidates of the RBF and were chosen along the subspaces. The maximum of the RBF does not converge to the global maximum of the objective function.

The issues can be resolved by exchanging the order of the BCA iteration and RBF solver iteration. Instead of utilizing the RBF solver to solve the subspace maximization in the BCA, the **BCA** will then be used for the candidate search in the **RBF-solver** (BCA-R).

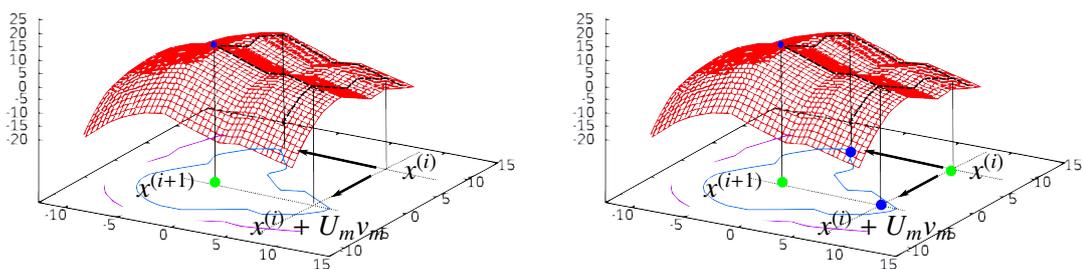


Figure 3.9: Illustration of three update methods in Algorithm 3: BCAIR (left) iterates the BCA on an **Invariant** RBF until a subspace maximum (green dot) is found. This point and the initial candidate are updated to the RBF. BCAUR (right) Updates the RBF at the beginning and after each subspace maximization (green/blue dots); The blue updates can be **Distributed** (BCADR).

Three instances of the method BCA-R are discussed in this section, as illustrated in Figure 3.9: A version that iterates the BCA on an **Invariant** response surface model until a stationary point is found which is

updated to the RBF (BCAIR); a version that Updates the response surface model after each subspace maximization (additional updates grayed, BCAUR); and a Distributed version of the last (BCADR).

The methods BCAIR and BCAUR are depicted in Algorithm 3. The lines shaded in gray represent modifications between BCAIR and BCAUR due to the following circumstance: The BCAIR is exclusively calling the response surface model (not the costly objective) when changing the subspace. Two function evaluations are appended, before and after the BCA iteration, which yields the following disadvantages. Firstly, it is very rare that two successive costly function evaluations are evaluated in a common subspace. These subspace optimizations correspond to the placement of one camera in the Example 3.2.4. Therefore, the method for decreasing the complexity of the costly objective function later on in Section 3.2.2 can not be utilized. On a second matter, computing the BCA iteration in parallel is not necessary, anymore, since the function evaluations inside the iteration are very cheap.

Algorithm 3 BCAIR, BCAUR, and BCADR when distributing the costly function evaluations in Line 18

```

1: Update RBF as in Algorithm 1, Line 1 and 2
2:
3: while RBF-solver termination condition do
4:   for all  $\beta$  in the search pattern  $\langle \beta_1, \dots, \beta_L \rangle$  do

5:      $\Delta \leftarrow \max_{x \in \mathcal{D}} \min_{1 \leq k \leq K} \|x - s_k\|$ 
6:      $\bar{x} \leftarrow \operatorname{argmax}_{x \in \mathcal{D}} \min_{1 \leq k \leq K} \|x - s_k\|$ 
7:      $s_{K+1} \leftarrow \bar{x}$ 
8:     Costly function evaluation and updates as in Algorithm 1, from Line 11 to 18
9:
10:    while BCA termination condition do
11:      for all subspaces  $V_m, m = 1, \dots, M$  do
12:         $\Delta_m \leftarrow \begin{cases} \|x^{(m)} - \bar{x}^{(m)}\| \\ \max_{x \in V_m} \min_{1 \leq k \leq K} \|x - s_k\| \end{cases}$ 
13:        Maximize  $f(x)$ 
14:        Subject to
15:           $\|x - s_k\| \geq \beta \Delta_m, k = 1, \dots, K$ 
16:           $x \in V_m$ 
17:
18:          Costly function evaluation and updates as in Algorithm 1, from Line 11 to 18
19:        end for
20:      end while
21:      Costly function evaluation and updates as in Algorithm 1, from Line 11 to 18
22:    end for
23:  end while

```

For a distribution of the costly function evaluations, let us make the following alterations to Algorithm 3, shaded in gray: The costly function evaluations of Line 18 can be parallelized. The sequentially

executed function evaluations of Line 8 and 18 can be calculated by the cheap function evaluations of Section 3.2.2. Changing the calculations of Line 12 transforms the inner iteration into a second RBF-solver on only one subspace. These two alterations lead to the fact that we can utilize the parallel method with the cheap function evaluations of Section 3.2.2.

It can be seen in the following theorem that Algorithm 3 converges with all three types of updates.

Theorem 3.2.12

Let \mathcal{D} be bounded. The BCA-R in Algorithm 3 converges to the global maximum objective value of any bounded, continuous objective function f on \mathcal{D} . Additionally, it is an anytime algorithm.

Proof. For the BCAUR/BCADR the candidates constructed until Line 8 constitute a sequence of iterates that are densifying in \mathcal{D} as in Lemma 3.1.15. With the same reason as in Theorem 3.1.16 the proof can be closed. □

In this subsection, two globally convergent optimization methods have been developed, the BCAIR and BCAUR of Algorithm 3. The last procedure can be computed in parallel, this version will be called BCADR. The computation in parallel threads can be achieved by using a global RBF as a response surface model for the first update in Line 8. For the updates 18 of each parallel thread (or each subspace for that matter) an exact copy of this model needs to be used. The same model is also used for the maximization of Line 13 till 16. After distributing the iteration into subspaces optimizations, the subspace models need to be merged by updating the newly found sample pairs into the global one. The update of Line 21 again affects the global model.

The distributed version has the following advantage:

Lemma 3.2.13

Consider the BCAUR with $M \in \mathbb{N}$ being the number of subspaces and I_0 being the number of steps of the inner iteration (Lines 10–20). Let $T \in \mathbb{N}$ be the number of parallel threads of the BCADR with the same number of subspaces and iteration steps.

Then the total number of subsequent function calls of one outer iteration step (Lines 3–23) is

- $I_0 \cdot M + 1$ in case of the BCAUR and
- $I_0 \cdot \left\lceil \frac{M}{T} \right\rceil + 2$ in case of the BCADR.

Proof. Let us suppose the number of parallel threads T is large enough, I will comment on this later in the proof. The costly objective function calls (Line 8, 18) in the non-distributed version of Algorithm 3 subdivide into two groups: The first call in Line 8 needs to be done before the other calls in the same RBF-solver iteration step are done. The following calls in Line 18 are calls in M subspaces. The calls within each subspace need to be done subsequently, all the subspaces can be computed in parallel. When limiting the BCA iteration to a number of iteration steps I_0 for a problem with M subspaces then the number of all calls in one RBF-solver iteration step is $I_0 \cdot M + 1$.

When distributing the Algorithm, the call in Line 22 has to be done after the parallelization, additionally. A number of M calls can be computed parallel. $I_0 + 1 + 1$ need to be done sequentially. Now, consider

that the number of parallel threads T is limited. If $M \leq T$ the maximum number of parallel computable function calls stays the same, this is considered “large enough”. If this is not the case, then, out of the former M parallel function calls, a number of $\lceil \frac{M}{T} \rceil$ calls need to be done subsequently. The minimum total number of subsequent costly function calls is $I_0 \cdot \lceil \frac{M}{T} \rceil + 2$. \square

The advantage of BCADR compared to BCAUR is the decreased number of sequentially computed costly function calls, which is shown in the last lemma. Furthermore, both as well as the BCAIR can be equipped with symmetric prior information as in 3.1.14. In the following sections, we will explain the experimental setup that is needed to investigate whether this advantage helps the BCA-R versions to play in a better league than the plain RBF-solver.

3.3 Experimental Setup

The aim of the experiments is to test the proposed optimization methods, namely the *BCAIR*, *BCAUR*, and *BCADR*, for flaws and errors. Of particular interest are the efficiency and accuracy of these methods, and the question whether these criteria depend on the objective function. In this section, the following questions are answered:

How do you quantify the notions of efficiency and accuracy of a solver in our context? And what types of tests have been done? (Section 3.3.3) What types of functions need to be tested in order to get significant test results? (Section 3.3.1) We also briefly comment on the hardware setup and argue on the details of the implementation. (Section 3.3.4 and 3.3.2)

3.3.1 Test Problems

The problem (1.2) is non-convex, stair-cased or differentiable μ -almost everywhere, and sometimes symmetric. From these criteria several objective functions $f : \mathbb{R}^n \rightarrow \mathbb{R}$ have been chosen to test the solvers in the domain of $\mathcal{D} = [-10, 10]^n$, where n is the dimension of the function, or in other words the size of the problem.

First, illustrated in Figure 3.10, there are two objective functions of dimension two that are both ascending from $(-10, -10)$ to $(10, 10)$ linearly with equal slope and attain their maximum of 10 at the boundary point $(10, 10)$. The function to the left has a shape like a sloping hill and is differentiable at every point. The function to the right attains a sloping roof shape and is only differentiable outside of the set $\{(x_1, x_2) \in \mathbb{R}^2 \mid x_1 = x_2\}$. We have tested both functions as depicted, and additionally, we have varied the slopes on either side of the diagonal by multiplying the function by a constant factor. The maximum function value of both functions is the same when using the same factor.

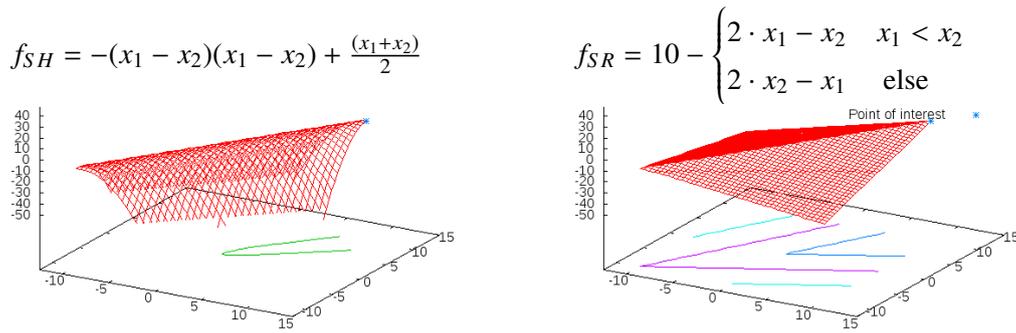


Figure 3.10: Differentiable (sloping-hill-shaped, left) and non-differentiable (sloping-roof-shaped, right) objective function, ascending from $(-10, -10)$ to $(10, 10)$ linearly and obtaining their maximum at the boundary point $(10, 10)$;

In Figure 3.11, two objective functions of dimension two can be seen which obtain their unique local and global maximum at an interior point $(-2, -2)$. The function to the left has a parabolic shape and is differentiable at every point. The function to the right obtains a roof top shape and is not differentiable at the maximum. Again, we have also tested both functions with various slopes on either side of the maximum. The maximum function value of both functions is the same when using the same factor.

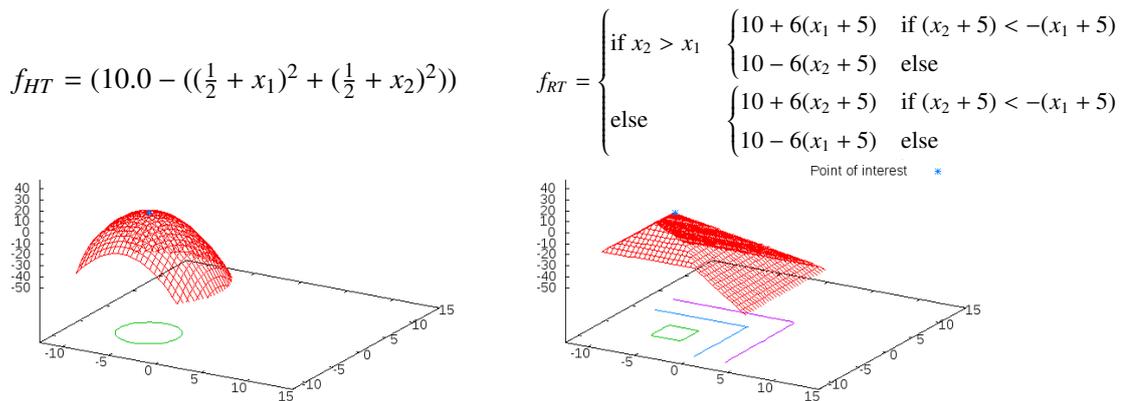


Figure 3.11: Differentiable (hill-top-shaped, left) and non-differentiable (roof-top-shaped, right) objective function, both attaining their single local maximum at the interior point $(-2, -2)$;

Another objective function that is used to test the solver is the additively separable function $f_n : \mathbb{R}^n \rightarrow \mathbb{R}$ of adjustable size $n \in \mathbb{N}$:

$$f_n(x_1, \dots, x_n) = 10 + \sum_i^n \begin{cases} -\frac{(x_i+2)^2}{4} & x_i < -2 \\ -1.4x_i - 2.8 & x_i < 3, \quad x_i \geq -2 \\ 0.5x_i - 8.5 & x_i < 5, \quad x_i \geq 3 \\ -x_i - 1 & \text{otherwise.} \end{cases}$$

The left image of Figure 3.12 illustrates f_n with $n = 1$, the right image shows $n = 2$. Note that in this work the function is experimentally tested up to $n = 5$. The function has several local maxima, is neither

convex nor concave, and is only piecewise differentiable. The local maxima of this function are interior points depicted in the set $\{(x_1, \dots, x_n) \mid x_i \in \{-2, 5\}, \forall i \in 1, \dots, n\}$, but the only global maximum of this function is at $(-2, \dots, -2)$. The non-differentiable points lie on hyperplanes with $x_i \in \{-2, 3, 5\}$. The function is not differentiable at the maximum.

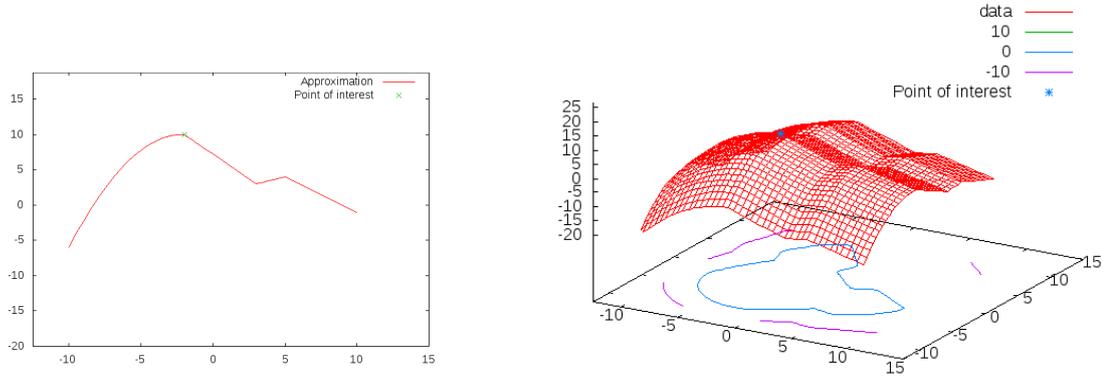


Figure 3.12: Additively separable objective function f_n with adjustable size, $n = 1$ (left) and $n = 2$ (right); this function is tested up to $n = 5$. The function has several local maxima, is neither convex nor concave, and is only piecewise differentiable. The only global maximum of this function is $(-2, \dots, -2)$.

Additively separable means f can be written as a sum of functions ϕ_i whereas ϕ_i only depends on the variable x_i , $i = 1, \dots, n$. In our case these subspace functions are piecewise defined by the conditions also depending on the same single variable:

$$f(x_1, \dots, x_n) = \sum_i^n \sum_c f_{c,i}(x_i) \cdot \delta_{\text{Condition}_c}(x_i)$$

Last but not least, a stair-cased functions need to be tested, as well. Here, we test the function $f_n^{sc}(x) := f_n(\lceil x \rceil)$, again of adjustable dimension $n \in \mathbb{N}$. The left image of Figure 3.13 illustrates f_n^{sc} with $n = 1$, the right image shows $n = 2$, but again we tested up to $n = 5$. The function is stair-cased on the intervals between $[-10.5, -9.5]$, ..., $[9.5, 10.5]$ of one dimension. It has several local maxima, is neither convex nor concave. The local maxima of this function are interior points that are contained in the set $\{(x_1, \dots, x_n) \mid x_i \in \{-2, 5\}, \forall i \in 1, \dots, n\}$ (as well as the interval around it), but the only set of global maximizer is at the plateau of $(-2, \dots, -2)$.

The functions that are introduced as test objective functions in this section are piecewise differentiable, stair-cased, and additively separable. They are used in various dimensions and with various slopes in Section 3.4 to test the solvers developed in the Sections 3.1 and 3.2.

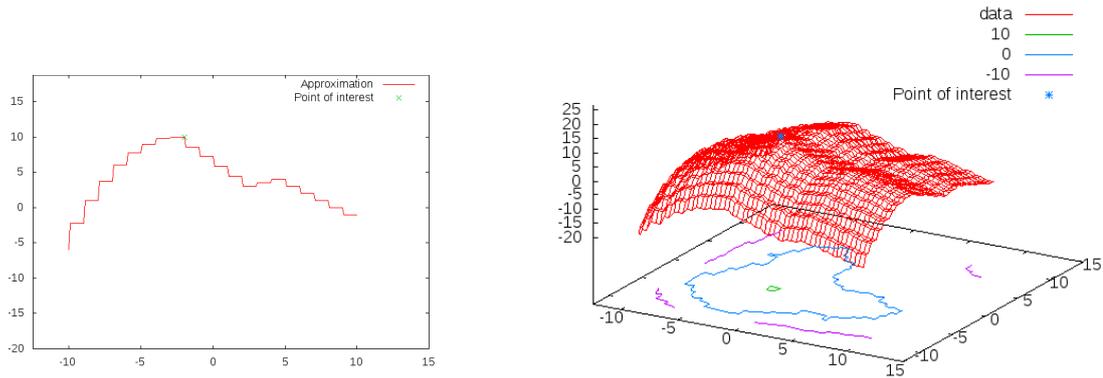


Figure 3.13: Stair-cased objective function f_n^{sc} with adjustable size, $n = 1$ (left) and $n = 2$ (right); this function is tested up to $n = 5$. The function has several local maxima, is neither convex nor concave, and is only piecewise differentiable.

3.3.2 Implementation Details

For a fair comparison, all the algorithms share some common components: The candidates for an initialization of the algorithms, the optimization method searching for the size of the exclusion area Δ (solving the max min-term), an optimization method searching for the next candidate outside the exclusion area, and the update of the RBF using newly generated candidates. The requirements and implementation details of these components are summarized, here:

Type of RBF The requirements to the initial candidates and their update to the RBF depend on the kernel type and the polynomial that is used in the RBF. The implementation uses a thin plate spline as kernel $\phi(r) = r^2 \log(\gamma \cdot r)$ and a linear polynomial $p \in \Pi_m^n$ with the basis $\{1, x^{(1)}, \dots, x^{(n)}\}$. The polynomial describes a hyperplane in \mathbb{R}^n .

Optimization methods Searching for the size of the exclusion area Δ and afterwards searching for a candidate requires nonlinear solvers coping with constraints.

In order to solve the maximization of the distance between all the candidates and the furthest point of the domain (in Line 5 of Algorithm 3) the method of moving asymptotes implemented by [72] is used. It utilizes a *local* approximation to search for a good solution and was originally proposed by [148]. The convergence of this solver depends on the ability to establish a gradient, but the function $\|x - s_k\|$ is not differentiable at s_k . To pick a differentiable initial point a point is randomly chosen in the domain excluding the candidates $s_k, k = 1, \dots, K$.

The constraints of the candidate search (in Line 13) need to be modified before each optimization, this can be done by an interior point method implemented in the library [156], originally proposed by [155]. Both methods are designed for large scale, inequality-constrained nonlinear programming problems.

Initial candidates The solvers proceed stochastically due to the randomly chosen starting point of the method of moving asymptotes (in Line 5). The outcome of a solver call could therefore differ from the outcome of another solver call of the same initial solver configurations. Thus, the optimization

methods are called with the same solver parameters but different initial candidates, to get a better understanding of the average performance. For the synthetic functions, the optimum lies either at $(-2, \dots, -2)$ or $(10, 10)$. Thereby, the first initial candidate is taken randomly from the interval $[4, 9)$, which by construction is free of optima. The rest of the candidates are generated as proposed in Example 3.1.10.

RBF update When adding a candidate to the RBF the equation system (3.7) needs to be solved. The distances between candidates inside this equation system do not change when adding a candidate, so these can be reused from the previous iteration step. To solve the system a LU-decomposition by [56] is used. This has the advantage that the determinant of the matrix can be calculated, easily. The determinant is zero if the equation system is singular, which is the case if a redundant candidate is added, a candidate that has already been updated. Since two similar candidates are not needed in the response surface, the determinant test is useful to avoid this situation.

A good reason to use an LU-decomposition is also that the update of a candidate to the RBF results in only a single additional row and column in the equation system (3.7). Adding single rows and columns to an LU-decomposed matrix has already been researched several decades ago, and using a method such as [59] can be achieved in $o(n^2)$ instead of $o(n^3)$. As an alternative one could also utilize the update method of [116] who adds candidates to the above equation system just as the Newton's subsequent interpolation method.

The methods BCAIR, BCAUR, and BCADR, as well as the original RBF-solver are initialized as above.

3.3.3 Focus of the Experiments and Test Parameter

The solvers above proceed stochastically to find one of these global optima. Even if the same initial candidates and the same test parameters are used to start the solver, the solutions could differ due to the stochastic component. In order to show that the solver's performance does not depend on the choice of the random component, the tests are grouped into a given number $H \in \mathbb{N}$ of test runs using the same solver parameter but different initial points. In the following chapter, the term *test run* is used to express one solver call, the term *test group* indicates the given number of solver calls with the same parameters, and the term *investigation* emphasizes the number of test groups in which only one parameter is varied.

The focus of this chapter is on examining the efficiency and accuracy of an optimization procedure for various termination conditions of the algorithms. The methods we want to examine are BCAIR, BCAUR, BCADR, and RBF-solver. They are compared to the RBF-solver, a globally convergent algorithm and several locally convergent algorithms, both utilizing derivatives and derivative-free.

Efficiency Several efficiency measures are thinkable, e.g. time or memory consumption, for realtime systems or mobile devices, but we assume a powerful CPU with several cores. The time will mostly be consumed by the calls of the costly objective function. When optimizing such a function, the number of function calls is the critical criterium for measuring the efficiency of the solver.

Accuracy The accuracy of a solver is measured by the deviation from the optimal value, i.e. the difference between the optimal value of the objective function and the value of the solution that

was produced by the method. The difference can be measured as an absolute value or a relative percentage.

The efficiency and accuracy need to be condensed from all test runs of one test group. The *mode* of the investigation defines whether the maximum, average, or minimum of the measure of efficiency and accuracy is used.

The crucial parameters in the algorithms are the termination conditions of their iterations. A straight forward termination condition is to limit the number of objective function calls by a number $F_0 \in \mathbb{N}$. We have used a maximum number of 1000 function calls if not declared otherwise. But in order to compare which method is converging faster, the costly objective function calls need to be counted. In Algorithm 3 there are two iterations an outer iteration and an inner iteration. The outer iteration corresponds to the RBF-solver iteration from Algorithm 1, the inner iteration to the BCA. The termination of both should be considered adequately, here, beginning with the BCA.

BCA iteration If the subsequent subspace maximizations results in the same solution that the iteration has started with, then the iteration has arrived at a stationary point. The solution will not change when going through the subspaces again. So, the BCA termination condition that was used here is an minimum bound δ_0 for the absolute distance of two subsequent candidates s_K and s_{K+1} : If the inequality

$$\delta_0 \geq \|s_K - s_{K+1}\|.$$

holds, then the iteration is terminated. This yields the problem that a very slow progress of the solver is prohibited. In order to allow some small progress the BCA iteration is stopped if this bound is undercut *twice in a row*. Additionally, the number of BCA iteration steps I_0 can be limited.

RBF-solver iteration The RBF-solver is a globally convergent solver that generates the candidates dense in \mathcal{D} . One would want to terminate the RBF-Solver iteration if a minimum bound Δ_0 for this density is obtained. The degree of density is measured by the measure Δ of the exclusion area.

The investigations for the synthetic examples are depicted in the rows of Table 3.2. An investigation is only valuable if just one single parameter is varied in an investigation. If in a row two parameters 1,2 are varied then the investigation consists of all combinations of parameter 1 and 2, the same goes for more parameters. The symbol \rightarrow denotes the variable parameter, this parameter is further depicted in column “Investigation”.

| Focus | H | Mode | F_0 | I_0 | δ_0 | Δ_0 | f | Investigation | Section |
|-------------------------|-----|--------------------|-------|---------------|---------------|---------------|---------------|---|---------|
| Accuracy | 20 | Av., Max Min | 2000 | (1) | 0.15 | 5 | \rightarrow | Comparison to other solvers of [72]: $f = f_n, n = 2, 3, 4, 5$ | 3.4.1 |
| Efficiency | 20 | Av. | 2000 | (1) | 0.15 | 5 | \rightarrow | Comparison to other solvers of [72]: $f = f_n, n = 2, 3, 4, 5$ | 3.4.1 |
| Accuracy | 20 | Av. | 2000 | (1) | 0.15 | 5 | \rightarrow | Comparison to other solvers [72]: Stair-casing of $f = f_n^{sc}, n = 2, 3, 4, 5$ | 3.4.1 |
| Efficiency | 20 | Max | 1000 | (1) | 0.15 | \rightarrow | \rightarrow | Comparison of new methods: $\Delta_0 = 2, 3, 4, 5, 6;$ $f = f_n, n = 2, 3, 4$ | 3.4.2 |
| Efficiency, Accuracy | 20 | Av. | 2000 | (1) | 0.15 | 5 | \rightarrow | Comparison of new methods: $f = f_n, n = 2, 3, 4, 5$ | 3.4.2 |
| Efficiency, Accuracy | 20 | Av. | 500 | (1) | 0.15 | 2 | \rightarrow | Slope of function: $f = c \cdot f_{SH}, c \cdot f_{SR}, c \cdot f_{RT}, c \cdot f_{HT}, c = 1, 2, 3, 4$ | 3.4.3 |
| Efficiency | 20 | Av. | 500 | (1) | 0.15 | 2 | \rightarrow | Type of function: $f = f_2, f_{SH}, f_{SR}, f_{RT}, f_{HT}$ | 3.4.3 |
| Efficiency | 20 | Av. | 2000 | (1) | 0.15 | 5 | \rightarrow | Symmetry: $f_n, n = 2, 3, 4$ | 3.4.4 |
| Efficiency Accuracy | 10 | Av. Max Min | 500 | \rightarrow | \rightarrow | \rightarrow | \rightarrow | Termination criteria: $f = c \cdot f_{SH}, c \cdot f_{SR}, c \cdot f_{RT}, c \cdot f_{HT}, c = 1, 2$ $I_0 = 1, 2, \dots, 9;$ $\delta_0 = 0.05, 0.15, \dots, 1.45;$ $\Delta_0 = 1.0, 1.4, \dots, 3.8$ | 3.4.5 |
| Efficiency Accuracy | 10 | Av. Max Min | 500 | \rightarrow | 0.15 | 2.6 | f_{SR} | Increasing the number of inner iteration steps: $I_0 = 1, 2, \dots, 6;$ | 3.4.5 |
| Efficiency Accuracy | 10 | Av. Max Min | 1000 | 3 | \rightarrow | 2.6 | f_{SR} | Increasing the termination criterium of the inner iteration: $\delta_0 = 0.05, 0.15, \dots, 1.45$ | 3.4.5 |
| Efficiency Accuracy | 10 | Av. Max Min | 500 | 1 | 0.15 | \rightarrow | f_{SR} | Increasing the termination criterium of the outer iteration: $\Delta_0 = 1.0, 1.4, \dots, 5.8$ | 3.4.5 |

(1) 10 in case of BCAIR and one in case of both BCADR and BCAUR

Table 3.2: Table of investigations: Groups of H test runs are started with the same solver configurations but different initial candidates. The focus is then averaged or maximized (according to the mode) over the test runs in one group. The bounds for the maximum function calls F_0 , minimum distance between candidates δ_0 , minimum density Δ_0 , and number of BCA iteration steps, as well as the objective function f are varied as test parameters, denoted by the symbol \rightarrow .

3.3.4 Hardware Configuration

The experiments have been performed in a virtual machine “VMware Fusion 6” with “Hardware version 10” on an Apple “MacBook Pro Retina 15 inch, late 2013” on a four-core 2.6 GHz Intel Core i7 computing unit. The virtual machine has the 64-bit operating system “openSUSE 13.1 (x86_64)” installed. Two cores and 5948 MB RAM are forwarded to the virtual machine. The used compiler is “gcc (SUSE Linux) 4.8.1 20130909 [gcc-4_8-branch revision 202388]”. The following Table 3.3 shows an excerpt from the terminal command `$: cat /proc/cpuinfo`.

| | |
|-------------------|---|
| model name : | Intel(R) Core(TM) i7-4960HQ CPU @ 2.60GHz |
| cpu MHz : | 2592.697 |
| cache size : | 6144 KB |
| bogomips : | 5185.39 |
| clflush size : | 64 |
| cache_alignment : | 64 |
| address sizes : | 40 bits physical, 48 bits virtual |

Table 3.3: System configuration of the test machine.

In the last section, the modalities for the investigations in the next section have been discussed. The test problems are chosen in a variety of different slopes and function types, such as non-differentiable functions and separable functions. They have local optima at differentiable and non-differentiable points. The initialization of the algorithms has been discussed in the context of a fair comparison. In the end, the tests were introduced and the hardware setting was specified. With this hardware we have tested Algorithm 1 and Algorithm 3 in its three versions.

3.4 Experimental Results

The aim of the investigations is to answer the following questions:

- Which here proposed optimization method is more efficient and more accurate, the BCAIR, BCAUR, or BCADR?
- To which extend do the efficiency and accuracy depend on the objective function?
- How do the parameters of the algorithms need to be adjusted?
- To which extend may an exploitation of symmetry as prior information increase the efficiency?
- How are the solvers performing on a stair-cased function?

The parameters are tested on various test problems that differ in size, slope, differentiability, and whether their optimum is a boundary point or an interior. Additionally, the results are compared to other solvers like the original RBF solver, several local solvers that use a subspace decomposition and/or an approximation of the objective function, and a global solver.

3.4.1 Comparison to State of the Art

The derivatives of an objective function are utilized by local solvers, such as the sequential quadratic programming methods, to establish local optimality. In contrast to the local solvers, the stochastic solvers take advantage of the following fact: It has been proven that an optimization algorithm needs dense iterates to converge to the global optimum of an arbitrary continuous function, c.f. [153]. The word “dense” already hints that the convergence speed of local solvers is higher, but a global optimum might not be found. We will see now that for costly objective functions who cannot establish a gradient on their own, a third way exists, that is efficient and proven globally convergent. In this investigation the globally convergent optimization procedures from Algorithm 3, and the RBF solver, as discussed in Algorithm 1 are compared to already existing solvers.

The parameters that changed between test groups in this investigation were the size of the problem, it was varied between $n \in \{2, 3, 4, 5\}$ by using the separable function f_n . The function is only piecewise differentiable, non-convex and has several local optima, just as the function of problem (1.2). The minimum bound for the termination condition of the RBF solver iteration was set to $\Delta_0 = 5$, the bound for the BCA iteration to $\delta_0 = 0.15$. Again we averaged the deviation over a group of $H = 20$ test runs with the same solver parameters and the same objective function.

Figure 3.14 depicts the deviation from the optimal value for each solver, problem, and problem size. One column of the chart shows a line from the minimum to the maximum of the group of 20 test runs, the mark in the middle emphasizes the average of the group. Each color corresponds to a different number of dimensions. The methods proposed in this thesis are to the left. The original RBF solver is to the right. These solvers were tested versus several local convergent methods implemented in [72], the method of moving asymptotes (LD_MMA), a Nelder-Mead-Simplex Algorithm (LN_NELDERMEAD), a version of Subplex (LD_SBPLX), and a sequential quadratic program (LD_SLQP), as well as a globally convergent algorithm an Improved Stochastic Ranking Evolution Strategy (GN_ISRES) taken from [72]. The first letter of the short names indicates the global (G) or local (L) character. The second letter shows whether the method needs the gradient of the function (D) or works without (N).

Regarding the accuracy, the figure shows that the developed methods play in the same league as this global convergent solver. The evolution strategy includes a stochastic mutation rule described in [130, 131] which helps the solver to converge to the global optimum. The local solvers lack a globally convergent rule as the mutation of the evolution strategy. The programs developed in this work feature such a global convergent rule.

But a globally convergent rule such as the mutation rule also increases the number of function calls. Thus, the question arises whether the newly developed methods offer any advantage in terms of the number of function calls in practice. Figure 3.15 shows that this is in fact the case. As in the last figure, the same test parameters were used. The figure illustrates the average number of function calls of a group of test runs as one bar of the diagram.

It can be observed that the new methods are almost as efficient as the local solvers, and even more: The method of moving asymptotes and the sequential quadratic program, based on [148] and [77], respectively, utilize a local approximation of the objective function (in contrast to a global response surface used here) to search for the next candidate. The approximation is convex and separable in the first case, and quadratic in the second, which makes the search for a new candidate very easy, but in order

to get such an approximation a gradient is needed. Only the versions of Nelder-Mead-Simplex [18, 102] as well as the further developed Subplex [129] do not need the objective function to provide a derivative, just as the methods developed here. The Subplex-version even applies the Nelder-Mead-Simplex method on a sequence of subspaces of the domain, just as our algorithms. All four of these methods show a very high inaccuracy in the figure above.

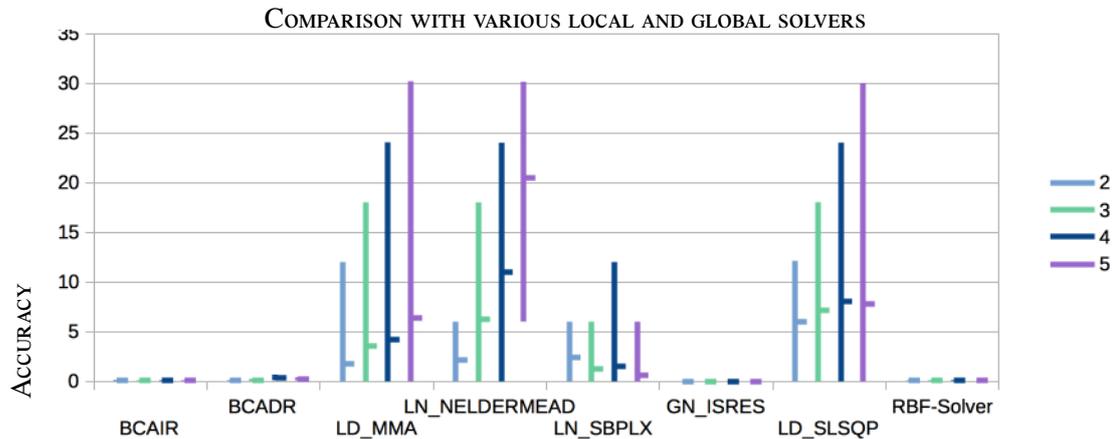


Figure 3.14: Line diagram: The deviation to the optimal value for problems in 2, 3, 4, and 5 dimensions; The line stretches from the minimum to the maximum observed deviation, and the average taken over 20 test runs is resembled by the mark in the middle. Several solvers are tested against the developed methods BCAIR, BCAUR/BCADR. The global GN_ISRES and additionally our methods have a significantly higher accuracy in all dimensions (2,3,4,5).

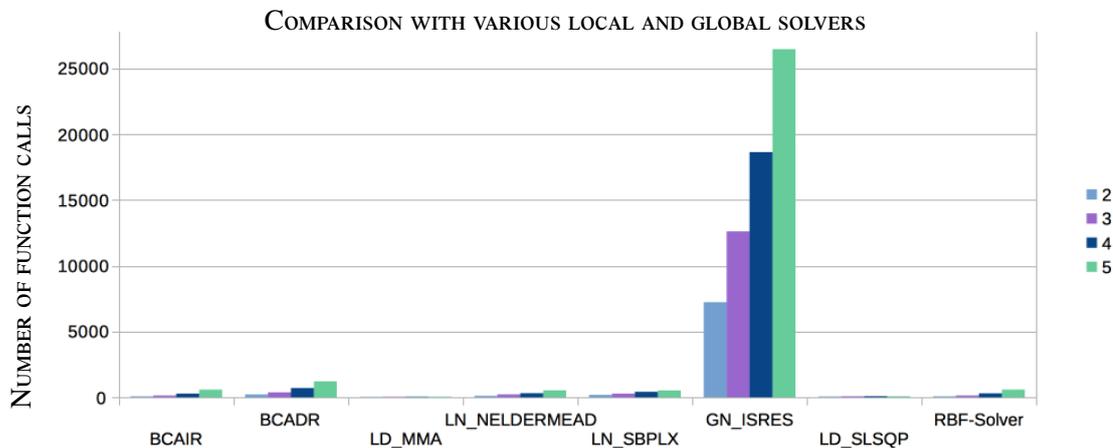


Figure 3.15: Bar chart: Average number of objective function calls for the tested solvers. All local solvers, namely LD_MMA, LN_NELDERMEAD, LN_SBPLX, and LD_SLSQP, and additionally our methods have a significantly higher efficiency in all tested dimensions (2,3,4,5) than the global one.

On this data set the new solvers outperform any of the other solvers, either in accuracy or efficiency. But is this also the case for objective functions which are stair-cased? Therefore, all solvers that can

be utilized without a gradient of the objective function, namely LN_NELDERMEAD, LN_SBPLX, and GN_ISRES, are used as comparison to our methods. The parameters that changed between test groups in this investigation were the size of the problem, it was varied between $n \in \{2, 3, 4, 5\}$ using the function f_n^{sc} . The function is stair-cased, non-convex and has several local optima, just as the function of problem (1.2). The minimum bound for the termination condition of the RBF solver iteration was set to $\Delta_0 = 5$, the bound for the BCA iteration to $\delta_0 = 0.15$. Again we averaged the deviation over a group of $H = 20$ test runs with the same solver parameters and the same size of the problem.

The first observation is that the global optimizer GN_ISRES could never establish convergence to one point. Therefore, its maximum number of function calls was limited to 50,000. This allowed the solver to always find the correct maximum, this can be seen in Figure 3.16. The other solvers that are used as comparison did converge satisfactorily. They were very inaccurate back in the tests in Figure 3.14 and it can be seen in the recent image that the inaccuracy increases while our methods stay as accurate.

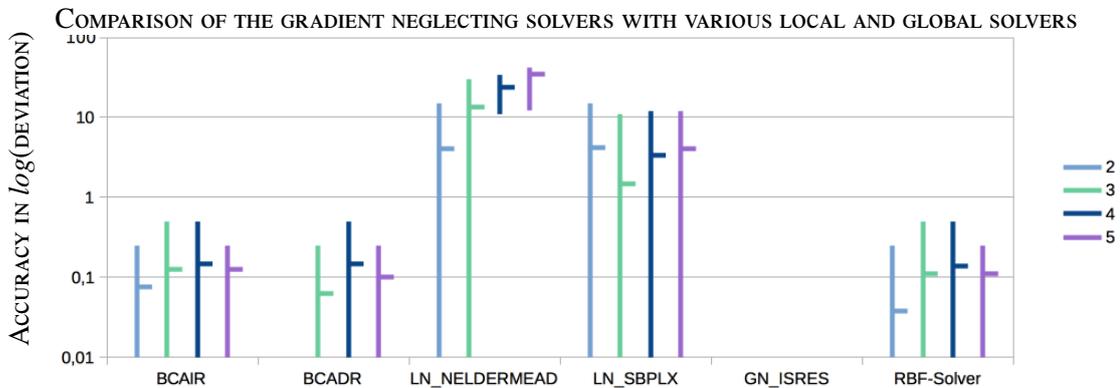


Figure 3.16: Maximum, minimum, and average inaccuracy illustrated for the stair-cased objective on a logarithmic scale. Several solvers are tested against the proposed methods BCAIR, BCADR. The local solvers, that can be utilized without a gradient of the objective function, namely LN_NELDERMEAD, LN_SBPLX are significantly more inaccurate than our methods in all tested dimensions (2,3,4,5). The deviation of GN_ISRES is 0 in all dimensions and the BCADR in two dimensions which cannot be shown on a logarithmic scale.

The examinations showed that the newly developed methods are as efficient as fast converging local solvers, and as accurate in finding the global maximum as stochastic solvers, even despite not using derivatives. While some of the other solvers had convergence issues or became more inaccurate when optimizing a stair-cased function, our solvers kept their performance.

3.4.2 Efficiency and Accuracy

In this section, we aim to characterize and differentiate the newly developed methods. In this investigation, the globally convergent optimization programs from Algorithm 3, namely the BCAIR, BCAUR, and BCADR, are compared to the original RBF-solver, as discussed in Algorithm 1.

Again, each test group consists of 20 test runs with the same parameters. The parameters that are varied between test groups in this investigation were the size of the problem, and the termination condition of the

RBF-solver iteration. The size of the problem is changed between $n \in \{2, 3, 4\}$ for the separable function f_n . The termination condition of the RBF-solver iteration is a minimum bound $\Delta_0 \in \{2, 3, 4, 5, 6\}$.

In Figure 3.17 the number of objective calls of each test group were maximized over 20 test runs and plotted as one bar of the diagram. Each color corresponds to one of the solvers. The abscissa shows the 15 test groups for each solver, on the left the ones with $n = 2$, in the middle the ones with $n = 3$, and to the right the ones with $n = 4$. As expected in all dimensions and with all termination conditions the maximum objective function calls decrease when using a higher bound for the termination condition. With increasing the dimension, the number of function calls increases as well. In can also be observed that the BCAIR is head-to-head with the RBF-solver. The maximum number of function evaluations of the BCAUR and BCADR are much higher but this is due to the fact that the objective function calls that can be computed in parallel are counted individually.

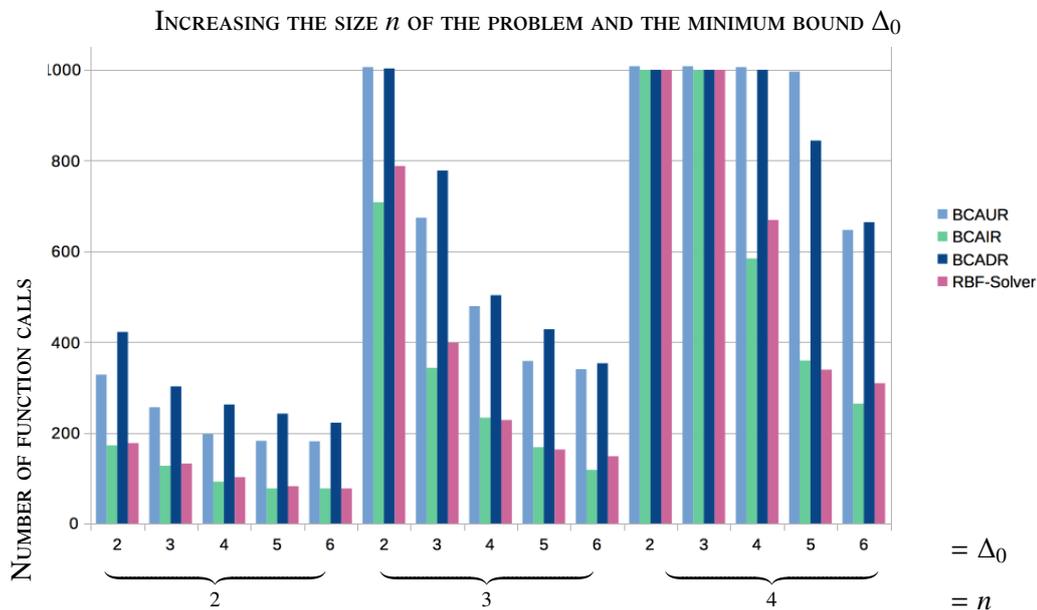


Figure 3.17: Bar chart of maximum number of objective function calls; each color corresponds to one of the solvers. The bars correspond to the maximum number of function calls in a group of 20 test runs which were started with the exact same parameters. The size of the problem was varied in $n \in \{2, 3, 4\}$, the minimum bound of the termination condition was varied between 2 and 6. In the test groups with $\Delta_0 = 5$ and 6 the RBF-solver and BCADR never exceeded the maximum of 1000 function calls in any dimensions.

The intriguing question is whether the RBF-solver will overtake the BCADR in the number of function calls when increasing the dimension. We will investigate, now, what happens to the *average* number of function calls in 2, 3, 4, and 5 dimensions. For this investigation, notice that the BCADR, BCAIR and the RBF-solver never exceeded the maximum number of 1000 function calls when using a termination condition of $\Delta_0 \in \{5, 6\}$ for all tested dimensions in Figure 3.17. For averaging the number of function calls, it is crucial to never reach the maximum number of function calls. Therefore, let us use the termination condition of $\Delta_0 = 5$ and increase the maximum number from 1000 to 2000.

Lemma 3.2.13 states the BCADR's minimum number of subsequent function calls when distributing the computation. Here, we used one BCA iteration step, so the function was maximized on each subspace exactly once before continuing with the communication (Line 22 of Algorithm 3). By the Lemma, when each dimension of the problem is computed in one subspace and the number of threads is larger than the dimension of the problem n , then just three out of $(n + 2)$ calls have to be computed subsequently.

How many subspaces are needed for the BCADR to be more efficient than the RBF-solver? The answer to the question is addressed in Figure 3.18. Each bar represents the number of function calls averaged over 20 test runs, depicted in 2, 3, 4, and 5 dimensions for the three solvers BCADR (blue) and RBF-solver (purple), and the BCAIR (turquoise). The average numbers of function calls show that the BCAIR develops a slight advantage against the plain RBF-solver. The numbers of BCADR are steadily higher than the numbers of the other solvers, but this is due to the fact that the objective function calls that can be computed in parallel are counted individually, here. The red rectangles resemble the solver calls that need to be computed sequentially according to Lemma 3.2.13. The rest of a BCADR bar resembles the function calls which can be computed parallel. The sequentially computed function calls start at a higher level than the ones of the remaining solvers in lower dimensions. But from dimension 5 on they show a clear advantage. The maximum number of function calls in five dimensions was 770, 1794, and 1020, in case of the BCAIR, the BCADR, and RBF-solver, respectively. So, the average number of solver calls is actually significant.

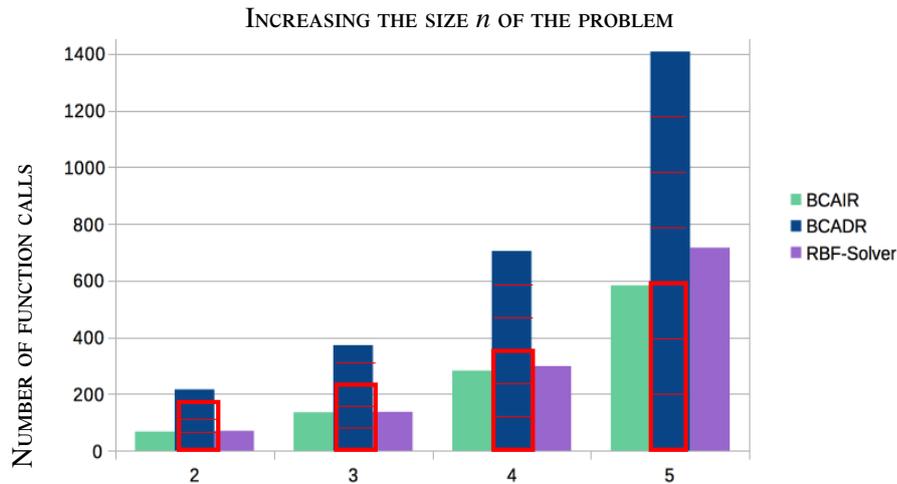


Figure 3.18: Bar chart: Each bar represents the number of function calls averaged over 20 test runs, depicted in 2, 3, 4, and 5 dimensions for the three solvers BCADR (red outline) and RBF-solver (purple), and the BCAUR (turquoise). The average numbers of function calls for the BCAUR are higher due to the summation of parallel function calls. The red rectangles show the number of objective function calls that actually need to be computed sequentially. Both methods develop an advantage against the original RBF-solver. For higher dimensions the BCADR looks most promising

The advantage of less number of function calls also comes with a disadvantage: For the same investigation the average deviation from the optimal objective function value is depicted in Figure 3.19. Starting

off with about the same deviation from the optimal objective value, the RBF-solver and the BCAIR advance slightly when coming to higher dimensions.

On the one hand, we have managed to increase the efficiency by decreasing the number of costly function calls that one core has to deal with. In comparison to the original RBF-solver, all developed methods show their advantage in larger problems. On the other hand, the accuracy is suffering marginally but much less so than for the local solvers.

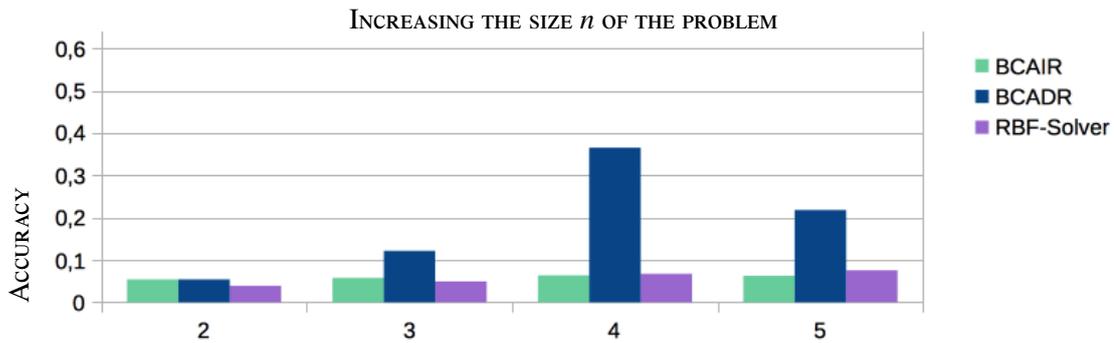


Figure 3.19: Bar chart: Deviation from the optimal objective value averaged over 20 test runs versus the dimensions 2, 3, 4, and 5 three solvers: BCADR (blue) and RBF-solver (purple), and the BCAIR (turquoise). Starting off with about the same deviation from the optimal objective value in lower dimensions, the RBF-solver and the BCA inside the RBF-solver advance slightly when coming to higher dimensions.

3.4.3 Dependency on Type and Slope of the Objective Function

In order to investigate the deviation of the optimal objective value a little bit further, several objective functions, namely f_{SH} , f_{SR} , f_{HT} , f_{RT} , were used. The slopes of these functions are easily changed by the multiplication with a factor $c \in \mathbb{R}$. We used $c = 1, 2, 3, 4$. The minimum bound for the termination condition of the RBF-solver-iteration was set to $\Delta_0 = 2$, the bound for the BCA iteration to $\delta_0 = 0.15$. A total number of 500 objective function calls were allowed. Again we averaged the deviation over a group of $H = 20$ test runs with the same solver parameters and the same objective function.

In Figure 3.20 to the left the range of deviation over all functions, regarding a particular solver and slope is depicted as one column of the chart. Each column is a vertical line from the minimum deviation to the maximum deviation over all test runs in one group and over all functions. The mark in the middle of the line represents the average. The BCAIR has one outlier for the slope 4 in one of the tests with the roof-top-function, which was excluded to show the average tendency. Other than that it can be observed that the slope plays an important role for the difference between the function value of the result and the actual optimal value for both functions. The higher the slope, the larger the range of the function values, of course, the more inaccurate the objective value of the result. It can also be observed that the deviation of the BCADR increases more strongly with higher slopes.

In a similar investigation, the function calls were measured and averaged over the group. This is illustrated in Figure 3.20 to the right. Again, each column is a vertical line from the minimum number to the maximum number of function calls of all types of functions and all test runs in one group. The mark in

the middle of the line represents the average. Surprisingly, both solvers show a slight reduction in the number of function calls when increasing the slope.

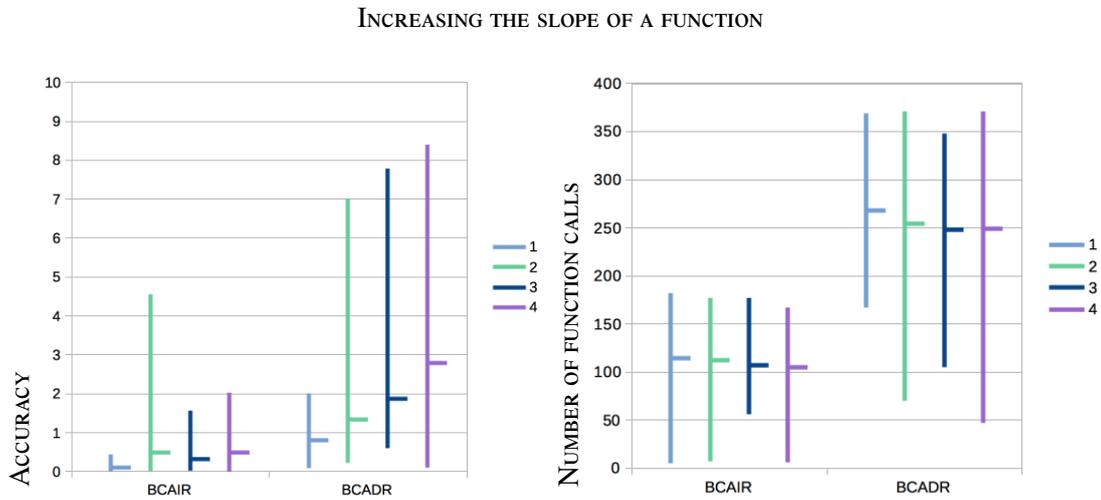


Figure 3.20: Left: Maximum, minimum, and average deviation over all functions regarding a particular solver (BCAIR, BCADR) and slope (1, 2, 3, 4) is depicted as one column of the chart. Right: Maximum, minimum, and average number of function calls over all functions and test runs in one group, for each solver (BCAIR, BCADR) and slope (1, 2, 3, 4) ;

An interesting question is also whether any of the solvers had a particular problem with either an optimum at the boundary or the interior of the domain, or with non-differentiabilities. We compared a differentiable function f_{SH} with a boundary optimum, a non-differentiable function f_{SR} with a boundary optimum, a differentiable function f_{HT} with an interior optimum, and a non-differentiable function f_{RT} with an interior optimum. The same parameters as in the last investigation were used, which are $\Delta_0 = 2$, $\delta_0 = 0.15$, $F_0 = 500$, and $H = 20$.

The first picture in Figure 3.21 shows the average number of function calls of the BCADR as an example, the relative numbers of all other solvers with the other slopes $c = 2, 3, 4$ were similar. The numbers of the non-differentiable functions (Hill top, roof top, and separable function) were slightly higher than the numbers of the differentiable ones. The function that has an optimum in the interior of the domain at a non-differentiable point (roof top) turned out to be more difficult for the BCAUR and BCADR, the average deviation of the objective value was higher. This can be observed in the second picture of the same figure.

Having a look at what produced the increased deviation yields that one of the solutions was $(-5.3, -4.9)$. This is not particularly far from the optimal solution, but for a better outcome a smaller bound than $\Delta_0 = 2$ might be necessary. Both solvers, on the other hand, have a marginal advantage when optimizing the differentiable function with a boundary optimum (sloping hill).

Many global solvers proceed stochastically, they produce dense iterates instead of concerning a gradient. The slope and type of a function does not matter so much. What we have found in the last section is that the slope of a function does matter to the accuracy and efficiency of the here developed solvers, despite their global property.

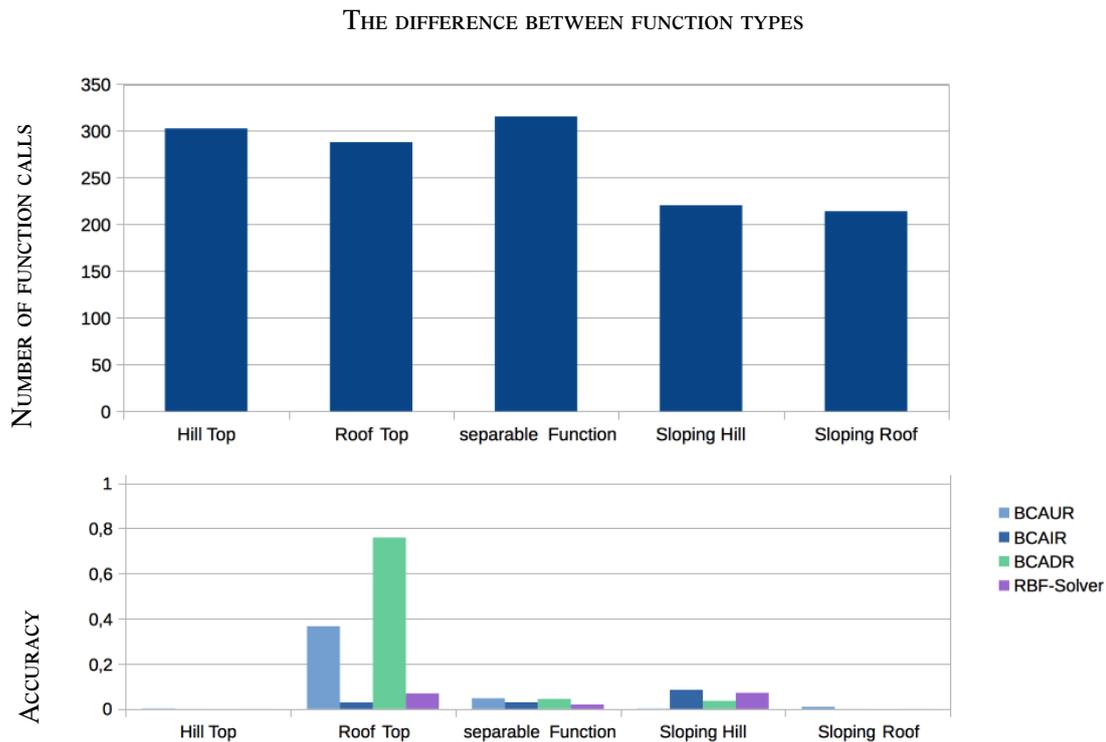


Figure 3.21: Bar chart: Average number of function calls (top) over all test runs of a particular function with slope $c = 1$ when optimizing with BCADR; Average deviation (bottom) from the optimal value regarding all four, the BCAIR, BCADR, BCAUR, and RBF-solver.

3.4.4 Incorporation of Symmetry as Prior Information

In the Sections 3.1.3 and 3.1.4, we have proved that the proposed solvers converge when incorporating prior information. We have also seen how much effect the incorporation of symmetry has, in Lemma 3.1.14. We examine now, how the efficiency of the developed solvers is correlated with the effect of the prior information.

In particular, we investigate how the number of function evaluation changes when incorporating the update for the symmetric solutions in a 2, 3, and 4-dimensional problem. The minimum bound for the termination condition of the RBF-solver-iteration is set to $H = 5$. Again we computed the average, maximum, and minimum the number of function calls for each group of $H = 20$ test runs with the same solver parameters and the same objective function. The allowed maximum number function calls is set to $F_0 = 2000$, the bound for the BCA iteration to $\delta_0 = 0.15$. The symmetric separable function f_n , with $n = 2$ is utilized as an objective function.

Figure 3.22 shows the average number of function calls when varying the dimension for both the solvers BCAUR (to the left) and BCAIR. Each solver is tested with and without symmetry as prior information, in blue and turquoise, respectively. The abscissa depicts the two solvers. You can see that the incorporation of symmetry as prior information increases the average efficiency for each solver significantly. The symmetry incorporating update rule for the BCAUR in four dimensions is not illustrated because we had

to stop the evaluation: In four dimensions, four subspaces were used, thus the RBF is updated with 6 sample pairs of distinct objective value each step. Each sample pair has $4!$ symmetric sample pairs with the same objective value, resulting in $6 \cdot 4! = 144$ updates of the RBF in each iteration step. Having a look at the diagram again, the expected number of function calls for a problem of the size 4 is about 200 or more. Thus, in the end, the equation system (3.7) has over $K + n + 1 = 200 \cdot 144 + 4 + 1 = 28805$ rows and columns.

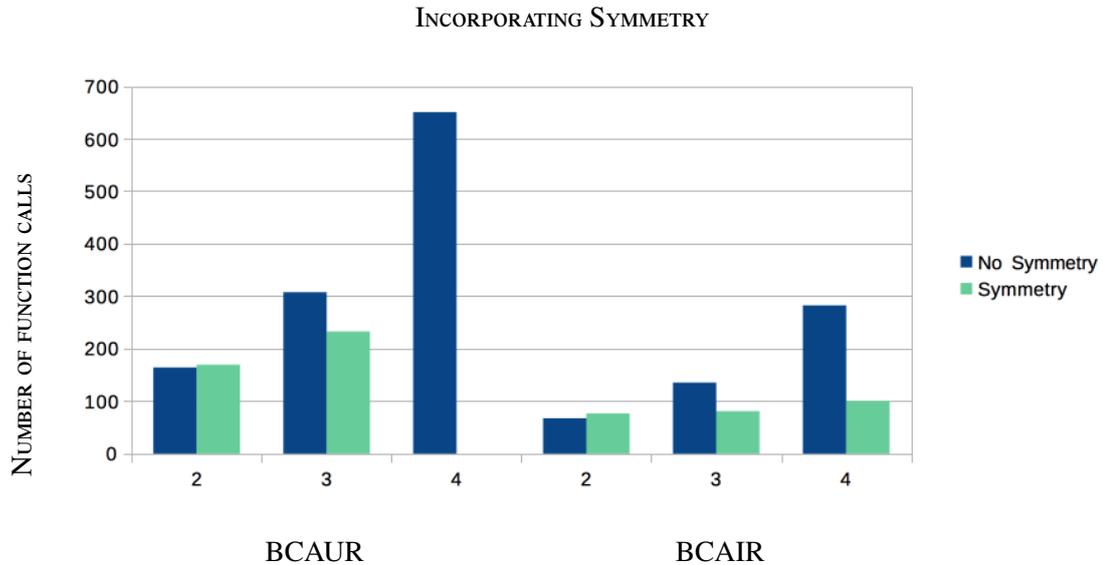


Figure 3.22: Bar chart: Average number of function calls of the recently developed solvers, BCAUR (left) and BCAIR (right) with (blue) and without symmetry (turquoise). Incorporating symmetry as prior information increases the efficiency for each solver significantly.

In general, the incorporation of prior information is advantageous for all the proposed methods. When considering higher dimensionally symmetric problems the updates of the RBF can exceed the capability of the solver for the equation system (3.7), though. Here, it is necessary to find a suitable rule for discarding irrelevant sample pairs.

3.4.5 Inner and Outer Termination Criteria

The newly developed methods are two nested iterations, the iteration of the BCA and the iteration of the RBF-solver. A naturally imposed question is when to terminate these iterations so that the number of function calls is minimized.

In order to investigate this for the BCAIR, the objective functions f_{SH} , f_{SR} , f_{HT} , f_{RT} with slopes $c \in \{1, 2\}$. The minimum bound for the termination condition of the RBF-solver-iteration is varied between $\Delta_0 = 1$ and 3.8 with a step size of 0.4. The bound of the BCA iteration is tested from $\delta_0 = 0.05$ to 1.45 with a step size of 0.1. The maximally allowed inner iterations (BCA iterations) were chosen between $I_0 = 1$ and 9 with a stepsize of 1. A total number of 500 objective function calls were allowed. Each parameter combination $(\Delta_0, \delta_0, I_0)$ was tested $H = 10$ times for each of the above objective functions.

The results are averaged over the whole lot of test runs corresponding to one function type. The observations are now illustrated on the example of f_{SH} with slope 1, the results of the other functions are transferable to this one. The first observation is that the efficiency and accuracy of the solvers does not depend much on the bound of the termination condition for the BCA iteration δ_0 . Thus, it is not illustrated, here.

Let us have a look at the efficiency and the average objective value when increasing Δ_0 in Figure 3.23. In the first image, each column is a vertical line from the minimum to the maximum number of function calls over all test runs for one function type, here f_{SH} . Again, the mark in the middle of the line represents the average. Below the average number of objective values is depicted in the same interval. Both the number of function calls and their variance decrease nonlinearly when increasing the minimum bound. Importantly, the variance of the optimal value (which is again 10) increases as well. One can make out a slight linear dependence between the average optimal value and the minimum bound.

INCREASING THE MINIMUM BOUND FOR THE TERMINATION CONDITION OF THE RBF-SOLVER-ITERATION Δ_0

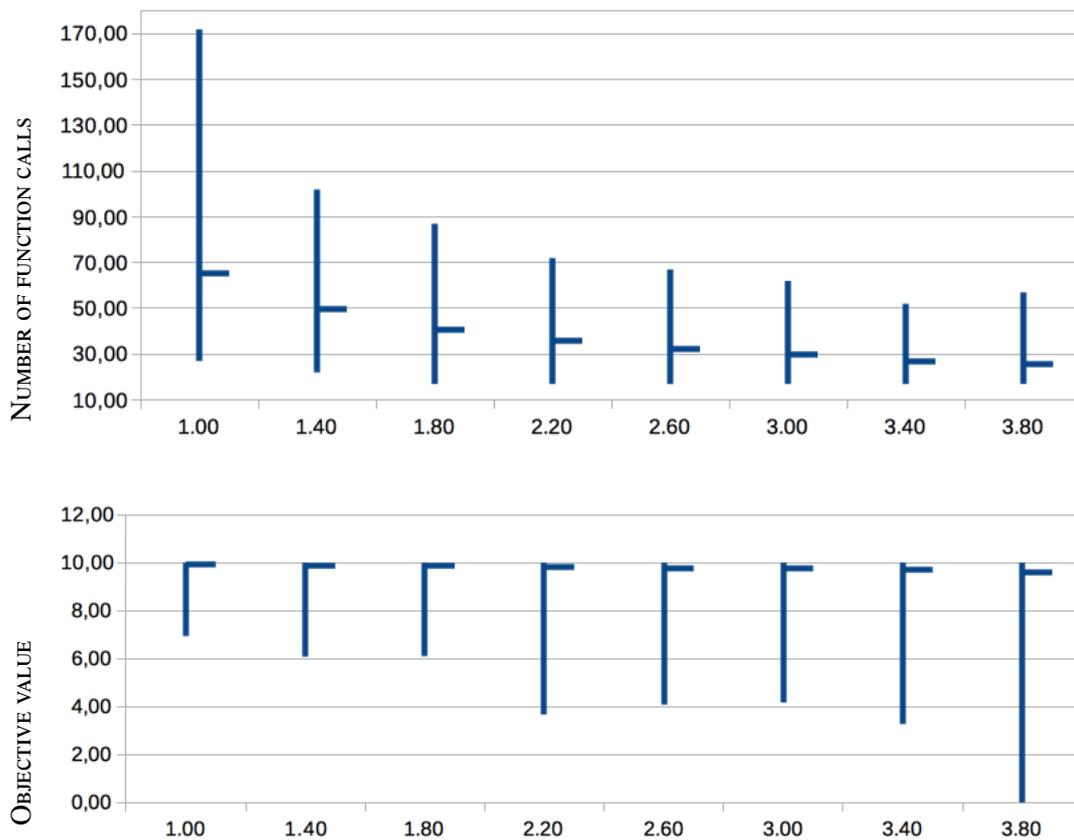


Figure 3.23: Function calls and average function values vs. the termination condition for the RBF-solver-iteration: The first image shows the number of function calls decreasing with higher bound, but the second image shows that the average function value at termination is also decreasing, making it more inaccurate.

While the efficiency, measured by the number of function calls, stayed almost constant for increasing the

number of allowed inner iterations I_0 (not illustrated, here), the objective value approaches the optimal value the higher the number of iterations is. Figure 3.24 depicts the dependency of the objective value on the maximum number of inner iteration steps, averaged over all tuples (Δ_0, δ_0) .

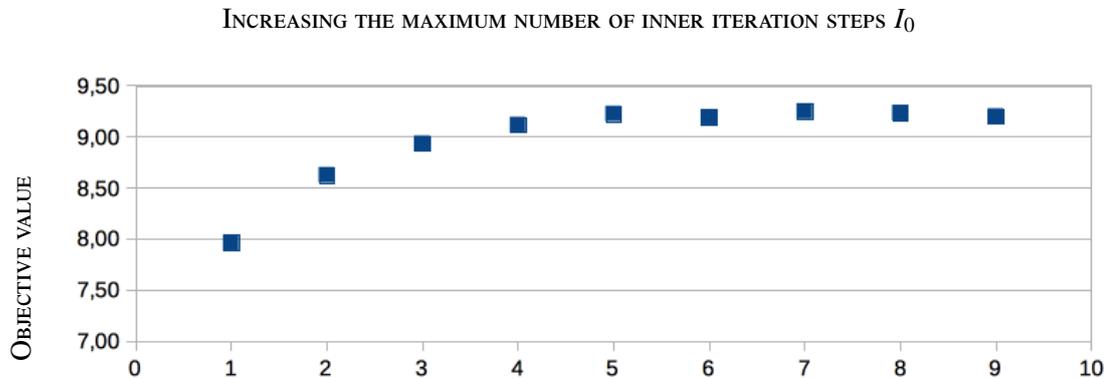


Figure 3.24: Plot about the dependency of the average objective value on the maximum number of inner iteration steps. The average objective value increases with the number of inner iteration steps.

The increase of the accuracy when increasing the number of inner iterations in case of the BCAIR is actually quite curious, because we can observe a different phenomena in case of the BCAUR/BCADR: In order to show this the function f_{RT} without factor is used, again. The maximally allowed inner iterations (BCA iterations) are tested between $I_0 = 1$ and 6 with a stepsize of 1. The minimum bound for the termination condition of the RBF-solver-iteration is $\Delta_0 = 2.6$, the bound of the BCA iteration $\delta_0 = 0.15$. A total number of 500 objective function calls is allowed. The number of function calls of a group of $H = 20$ test runs is averaged, maximized, and minimized.

The results in Figure 3.25 show that the number of function calls is linearly increasing with the number of allowed inner iteration steps from $I_0 = 1$ till 4. After that the maximum number of function calls is exceeded regularly, thus no linear dependency. However the minimum, maximum, and average accuracy have not changed significantly. This is quite an opposite observation than the observation when increasing the maximum number of inner iteration steps for the BCAIR. There the accuracy increased, but the number of function evaluations stayed constant.

A natural question is whether these function calls can be parallelized or need to be computed sequentially. Lemma 3.2.13 states the BCAUR's minimum number of subsequent function calls when distributing the computation. We have applied the solver with a problem of size $n = 2$ and number of subspaces $M = 2$. Let us assume we have two threads, as well. I_0+2 out of $I_0 \cdot 2 + 2$ function calls need to be done subsequently. The resulting numbers are plotted as blue bars in the same chart. The number of subsequent function calls is still increasing when increasing the number of iteration steps (except for $I_0 = 5, 6$). Practically, this means that the maximum number of inner iteration steps should be chosen as 1 unless additional information decreases the costs of a solver call in one subspace.

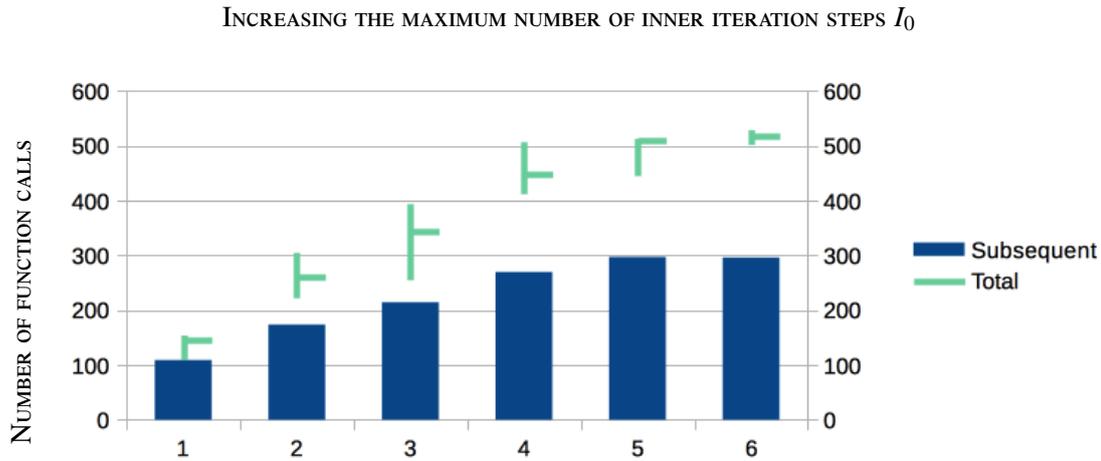


Figure 3.25: Plot about the dependency of the average objective value on the maximum number of inner iteration steps. The average objective value increases with more inner iteration steps.

We found that increasing the bound δ_0 and bound Δ_0 has a similar effect in terms of efficiency for the BCAUR/BCADR (not illustrated). This is why the investigations are not illustrated, here, either. In the first investigation $I_0 = 1$ is used and in the second $I_0 = 3$. The efficiency is decreasing when testing a higher $\Delta_0 = 1.0, 1.4, \dots, 3.8$, but staying approximately constant in the investigation range $\delta_0 = 0.05, \dots, 1.45$. The accuracy is very high (better than 0.05) in almost all groups. Only a marginal decrease is observed when increasing the bound Δ_0 , if any at all.

To conclude, choosing the minimum bound Δ_0 involves a trade off between accuracy and efficiency of both the tested solvers. The choice of the minimum bound δ_0 turned out to be non-critical in the investigated range. But increasing the maximum number of inner iteration steps the accuracy when utilizing an BCAIR, and decreases the efficiency in case of a BCAUR/BCADR.

3.5 Summary

Before listing the advantages and disadvantages of the proposed methods in Table 3.4, significant criteria are needed to evaluate the solvers. The criteria are arranged in four groups, a group related to the radial basis function (RBF) as a response surface model, related to the computation in parallel that comes with the block coordinate ascent (BCA), solver-specific properties such as convergence, and the types of objective functions which are suitable for these solvers.

Global radial basis function (RBF) The first three criteria are due to a global/subspace RBF: If the RBF is global then every candidate is updated to the same unique RBF and is cached in each subspace and the iteration further on. Furthermore, a global gradient will be provided by the RBF which means the problem of a stationary point at a non-differentiable point can be overcome. Additionally, the equation system of the RBF has the size $(K + n + 1)$ where K is the number of candidates, and n the dimension of the domain.

These three properties change if the samples of each subspace are only updated to a separate, subspace specific RBF. The size of the equation system, for example, is smaller since the number of candidates in one subspace is smaller, as well as the dimension of the subspace n_m . If the size of the equation system is tagged by (+) then it is comparably small. If it is marked by (–) it has the full size ($K + n + 1$).

Solver The following two criteria are properties of the solver which are not related to the RBF as a response surface model. E.g., the solver is guaranteed to converge if the related row is tagged by (+).

When referring to the maxmin (Line 5 and 12 in Algorithm 3, or Line 5 in Algorithm 1) or the candidate search (Line 13–16 in Algorithm 3, or Line 6–9 in Algorithm 1), these steps are related to the optimization based on a response surface model. The size of the problems corresponds to the dimension of the subspace or domain. The size of the maxmin problem is as large as the whole domain n if the corresponding row in the table is marked with (–) and is as small as the subspace if it is marked by (+). The same goes for the size of the search for a candidate. In case of the BCA in the last column the criterium “size of candidate search” refers to the size of the subspace problems, which is n_m for subspace V_m , so it is tagged with (+).

Parallelization Our goal was to speed up on the optimization by decreasing the number of costly objective function calls and by decreasing the costs of one function call. The number of function calls is decreased for one core if the solver can be computed in parallel ((+) at “parallelization is possible”). But the computation in parallel might stand in the way of the reduction of the function call costs. If the cost reduction of the method of Recalculation of the Sections 3.2.2 and 2.2.3 works with the parallel version, the line is marked with (+). The same holds for the cost reduction of the method of Multiplication in Section 2.2.3.

Objective function Since the original objective function (1.2) may be costly, symmetric, stair-cased, and likely to be a black box function, these are the properties that we want to distinguish the solvers by.

We have discussed the following combinations of the Radial Basis Function Solver and the Block Coordinate Ascent:

RBF-solver inside BCA The RBF-solver is used to optimize each subspace maximization of Iteration 3.2.1 or 3.2.2. Even when using a global RBF for all the subspaces, the convergence of neither the method nor the model is guaranteed.

BCAIR The BCA is used as a solver to search for the next candidate while staying away from already evaluated candidates in Line 6 till 9 of the RBF-solver from Algorithm 1; The BCA calls a global RBF as a response surface model. Once a maximum is found on all the subspaces the iteration is continued with exactly one costly function evaluation and the updates. So, distributing the BCA iteration does not decrease the costly function evaluations for one core. On the other hand, the candidates are updated to the global RBF and are therefore remembered.

BCAUR/BCADR These are further developed versions of BCAIR where costly function evaluations are added within the subspace maximizations. The following versions are possible sequential (BCAUR) and parallel computation (BCADR)

Here are advantages (+) and disadvantages (–) of the methods due to the mentioned criteria. The tag (×) means that the criterion does not apply to the particular solver.

| Criteria | Method | BCAUR BCADR | BCAIR | RBF-solver inside BCA | RBF- solver | BCA |
|------------------------------|--------|----------------|-------|--------------------------|----------------|-----|
| Global RBF | | | | | | |
| Candidates remembered | | + | + | – | + | – |
| Global gradient provided | | + | + | – | + | – |
| Size of RBF equation system | | – | – | + | – | × |
| Solver | | | | | | |
| Size of maxmin | | +(2) | – | + | – | × |
| Size of candidate search | | + | + | + | – | + |
| Global Convergence | | + | + | – | + | – |
| Parallelization | | | | | | |
| Is possible | | +(3) | + | + | – | + |
| Works with Section 3.2.2 | | + | +(1) | + | – | + |
| Works with Section 2.2.3 | | + | + | + | + | – |
| Property of Objective | | | | | | |
| Symmetric | | + | + | – | + | – |
| Stair-cased | | + | + | – | + | – |
| Costly | | + | + | – | + | – |
| Black box | | + | + | – | + | – |

Table 3.4: The advantages (+) and disadvantages (–) due to the criteria as explained in the text. The symbol (×) means that the criterion does not apply to the solver.

(1) We cannot reduce the costs by the method in Section 3.2.2, however this is no disadvantage since the evaluations of an RBF are cheap, anyway.

(2) There are two maxmin computations at the full dimension n of the domain \mathcal{D} . In each subspace (or parallel thread) the size of the maxmin problem computation is only as large as the dimension of the subspace.

(3) The parallel and sequential function calls are determined by Lemma 3.2.13.

The experiments show that the newly developed methods BCAIR, BCAUR, and BCADR, which are guaranteed to converge, are as efficient as local solvers, and as accurate in the choice of the global maximum as global, stochastic solvers, and this even despite the lack of a gradient. All the methods have

an increased efficiency in comparison to the original RBF-solver on the tested functions. The accuracy is marginally reduced in case of BCAUR/BCADR, if at all. This is true because the new solvers depend on the type of the function and its slope, although they are global solvers.

The incorporating of symmetry as prior information has a huge impact on the number of function calls from three subspaces upwards. Particular care should be taken when choosing functions that are symmetric in a high number of subspaces since the number of rows and columns of the RBF's equation system is very large there (>20000), in the end. Thus, further research should go into rules that discard irrelevant sample pairs.

Last but not least, when choosing a minimum bound for the termination condition in the RBF-solver-iteration a trade-off between accuracy and efficiency needs to be made. When increasing the maximum number of inner iteration steps the BCAIR and BCAUR show a different character: In case of the BCAUR the efficiency decreases when increasing the number of maximum iteration steps, but the accuracy stays the same, so it is best to utilize only one inner iteration step. This is not true for the BCAIR for which the accuracy increases until reaching a saturation value. The efficiency does not seem to suffer from an increase of the number of inner iteration steps here.

The experiments show that the BCAIR takes the lead of all tested solvers when incorporating symmetry as prior information, which is the case when identically built cameras are used, and when computing on one core. The BCAUR/BCADR develops an advantage when the number of symmetric subspaces is lower and the number of parallel threads is higher, this is the case when more than four subspaces were used in our example.

Chapter 4

Application Examples

A motivation of this thesis is to protect the human, which can only be done if an approximation of the human is available. In order to approximate the human, a camera network is installed with multiple cameras. In order to reconstruct the human target more accurately and to better cover the relevant regions of the room, the camera network is optimized.

The procedure that approximates the human is a 3DBGS. Two images of the same camera, one before and one after a human comes into play are compared. The difference between the images shows where a human can be located in the scene. In order to decrease the approximation error of the target, the coverage $C(1, \{identical\})$ is maximized, as suggested in Corollary 2.1.14 of Section 2.1.3. The 1-reliable coverage $C(1, \{identical\})$ is the region of the room which has not changed according to the view of at least one camera.

In this chapter, we will test the proposed optimization methods of Chapter 3 with the objective function of Chapter 2, in order to approximate a human more closely and to increase the coverage of the important regions of a three-dimensional room. As an input to the optimization method, the CAD model of an environment is required which distinguishes between static and dynamic objects, the latter move on a trajectory in time. Two three-dimensional test environments including obstacles are used: The motivating robot cell of Figure 1.1, in the first chapter, is used in Section 4.2 to address the issue of dynamic obstacles. A generic room is used in Section 4.3 to show a way to incorporate the movement of dynamic targets into an optimization. Our methods are compared to:

- A random placement in the domain for the comparison to a method without costs,
- A heuristic placement in the corners of the room (and between) for the comparison to a placement that a human could choose,
- The Sbpplx for the comparison to a local solver that proved to be the most suitable alternative to our methods in Section 3.4.1,
- The original RBF solver of Algorithm 1 in Section 3.1 for the comparison to the state of the art.

The chapter is organized as follows: First, the hardware and software configurations are discussed in Section 4.1. Two test environments and the qualitative results are depicted in the Sections 4.2 and 4.3. In the end, the quantitative results are compared in Section 4.4.

4.1 Hardware and Software Configuration

In this section, the parameters of the solvers, the visibility analysis, and the hardware configuration are discussed in this order. We begin with the choice of the parameters in the four solvers.

The local solver, RBF solver, BCAIR/BCADR are stopped after $F_0 = 50$ function calls. The heuristic method and the random placement do not call the objective function at all. The BCAIR and BCADR both incorporate the a priori information of a symmetrical objective function. The parameters of the BCAIR/BCADR are selected as follows: The search pattern $\langle \beta_1, \dots, \beta_L \rangle = \langle 0.98, 0.6, 0.75, 0.2, 0.01 \rangle$ is used with $I_0 = 1$ inner iteration step, a maximum bound of the outer iteration $\Delta_0 = 0.7$, and a maximum bound of the inner iteration $\delta_0 = 0.0525$. For the RBF solver the same maximum bound of the outer iteration $\Delta_0 = 0.7$ and the same search pattern is used.

The visibility analysis and coverage is simulated with a pixel based method on an occupancy grid, as described in Section 2.2.1 and parallelized by OpenGL. The sequential calls are accelerated by the method of Recalculation in Section 2.2.3 in all our proposed methods.

A particular voxel can be marked with the following sensor labels by each camera: “identical”, “changed”, “out of range”, “occluded”, whereas “out of range” and “occluded” are summarized into “undetectable”. A voxel can be marked with distinct labels by each camera. The larger the number of cameras $N \in \mathbb{N}$ the larger the number of label combinations. For example three cameras could mark a voxel by (changed, changed, changed), (identical, changed, changed), (undetectable, changed, changed), etc. For illustration purposes, we will distinguish between voxels that are marked as

- Undetectable by all cameras $C(N, \{out\ of\ range, occluded\})$ (the set is defined in in Section 2.1.4),
- Identical by at least one camera $C(1, \{identical\})$,
- And the rest.

As an objective function, a weighted sum over the volumes of each of these sets is used. In the following tests, we have used uniform weights. For other applications, non-uniform weights are thinkable, e.g., to stress that the reduction of undetectable regions are more important than the reduction of occlusions due to background subtraction. In order to enforce in an optimization that the cameras pay more attention to a particular region of the room or an important pose of the human, one could also assign weights to a voxel depending on its position. Thus, the voxels next to a dangerous object, or inside a passage way of the human can be weighted higher. In these tests, the voxels are again uniformly weighted.

| | |
|-----------------|--|
| model name : | Intel(R) Core(TM)2 Duo CPU E8400 @ 3.00GHz |
| cpu MHz : | 1999.000 |
| cache size : | 6144 KB |
| address sizes : | 36 bits physical, 48 bits virtual |

Table 4.1: System configuration of the test machine.

The experiments have been performed on a computer using the 64-bit operating system “openSUSE 13.1 (x86_64)” and 2GB RAM. The OpenGL version 2.2.0 and the graphics card driver NVIDIA 340.46 is

installed on the test machine. The used compiler is “gcc (SUSE Linux) 4.8.1 20130909 [gcc-4_8-branch revision 202388]”. In Table 4.1, an excerpt of the system information tool `$: cat /proc/cpuinfo` shows the configuration of one of the two cores. With this hardware we have tested the four solvers, the BCAIR, BCADR, RBF solver, and Sbplx. The following sections show the test environments and results.

4.2 Robot Cell

The motivation of this thesis is to prevent damage to a human coexisting and cooperating with a robot in a common working environment, such as the robot work cell of Figure 4.1 (right). The protection system approximates the human by a camera network despite the existence of visible obstacles in the environment.

The visible obstacles include static obstacles, such as tables and racks, and dynamic obstacles, such as the robot. The detectable coverage of a camera is influenced by the static obstacles as described in Section 2.1.1. When maximizing the detectable or undetectable coverage, as often done in literature, the dynamic obstacles are not payed attention to. However, the dynamic obstacles are one of the reasons why the approximation of the human may be inaccurate: In the 3DBGS method of Section 2.1.2, a target and dynamic obstacle are both dynamic objects and determine the changed coverage of a single camera. Thus, when approximating a target by the method in the Sections 2.1.3 and 2.1.4, the volume of the dynamic obstacles resembles an offset to the volume of the target.

In contrast to optimizing the detectable or undetectable coverage, in this thesis the whole procedure of the 3DBGS is simulated in every iteration step of the optimization. Thus, both the identical and undetectable coverage of the cameras are optimized, in order to improve the camera network despite any type of obstacle.

This test focuses on the incorporation of static and dynamic obstacles in the optimization of the scene regarding only one moment in time. The incorporation of a whole trajectory of a dynamic obstacle is depicted in Section 4.3. First, the setup of the test is addressed in the next section and then we turn to the qualitative results of the test. The quantitative results are compared in Section 4.4.

4.2.1 Setup

Figure 4.1 illustrates the first test environment. For this test the real robot cell (right) has been rebuilt as a CAD model (left). A few static and dynamic objects of one moment in time are included in the CAD model.

In this environment $N = 6$ cameras are placed and oriented at the ceiling of the robot cell with four parameters each: Two variables of each camera determine where the camera is located in the plane $p \in [-1.9, 1.9] \times [-2.15, 2.15] \times \{2.7\}$. The orientation o of the camera is determined by the point $e = p + o$ which the camera eyes directly. Here, the robot at $(0,0,0)$ and the human at $(0,2,0)$ are important, so the domain of e is selected accordingly: $e \in \{0\} \times [0, 2] \times [0, 2.85]$. The initial positions for the BCAIR, BCAUR, RBF solver, and Sbplx are $(\pm 0.5, \pm 0.5)$, $(0.2, 0.5)$, and $(-0.2, -0.5)$. The initial orientation of all cameras is $(0, 0.68)$. As a suitable heuristic the four corners of the room, a position above the robot and a position above the human is used. All cameras except the last are facing the human, the robot is faced by the last camera. In these tests, three symmetrical subspaces instead of six are used for

the BCAIR/BCADR, since the investigation in Section 3.4.4 showed that a rule neglecting samples is necessary for more than or equal to four subspaces for the BCADR.

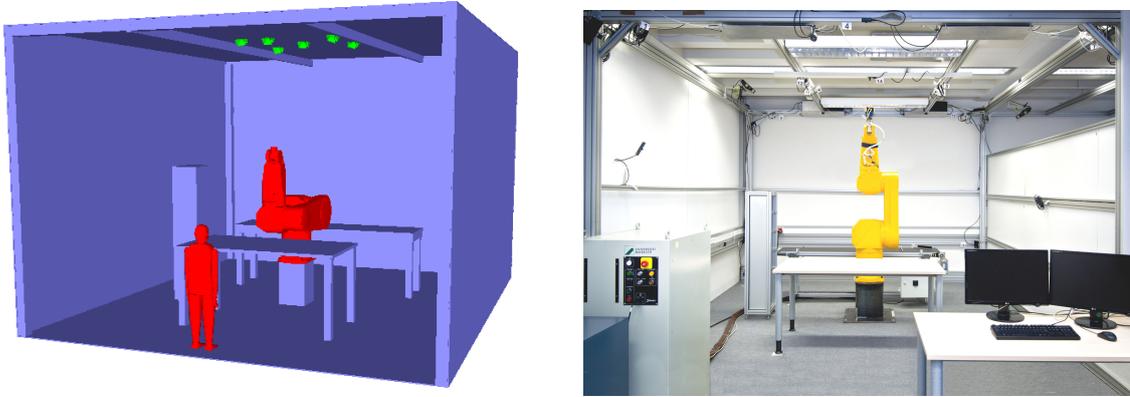


Figure 4.1: CAD model (left) of the robot cell (right): The static (left, grey) objects are two tables, a rack, and the robot cell ($4\text{m} \times 4.5\text{m} \times 2.85\text{m}$). $N = 6$ cameras (green) are placed with initial parameters for the solvers. Initially, the cameras are oriented towards the human. The static objects encompass 200 vertices and 300 faces. The human has 56k vertices and 58k faces, and the robot 94k vertices and 35k faces. The environment is rendered in images of 320×240 pixels and the occupancy grids have $80 \times 90 \times 57$ voxels.

4.2.2 Qualitative Results

We aim for a reconstruction of the human that minimizes the error caused by the static and dynamic obstacles. In Figure 4.2 the qualitative results of these tests are illustrated. The results are discussed regarding the separation of the voxel cluster around the human from the cluster around the robot, regarding the volume of the identical coverage and regarding the originality of the camera positions.

Let us consider the separation of the cluster of voxels around the human, and the cluster of voxels around the robot. Surprisingly, the results of all the methods showed the two mentioned clusters with a gap in between the human shape and the table, the result of the random placement as well; This result can be led back to the good choice of the domain: In particular, the orientation of the cameras was set to face the vertical hyperplane defined by the human's and robot's position. Furthermore, the random, heuristical placement, and the placement of the local solver improve the coverage next to the human but in the back or at the ceiling, right next to the robot, huge undetectable regions have occurred. In an application, both the changed and the undetectable voxels may include the human and it is not obvious which of the two the human belongs to. In all three placements, the robot needs to stop right away, not to risk hurting the human. In order to increase the separation of human and robot voxel clusters in an optimization, higher weights should be assigned to the voxels next to the human and the robot.

Additionally, when considering the goal of maximizing identical coverage, the methods utilizing a radial basis function interpolant as a surrogate, which includes the methods developed in this thesis, prove to be more suitable than the other methods. In general, the more identical coverage (voxel free space), the less changed and undetectable coverage (white and blue voxels), the better is the result. From the angle of view that is used to create the plots, the original RBF solver indeed seems to have the largest identical

coverage, but in the quantitative results of Section 4.4 we will see that the view is deceptive and that our methods improve the coverage even more.

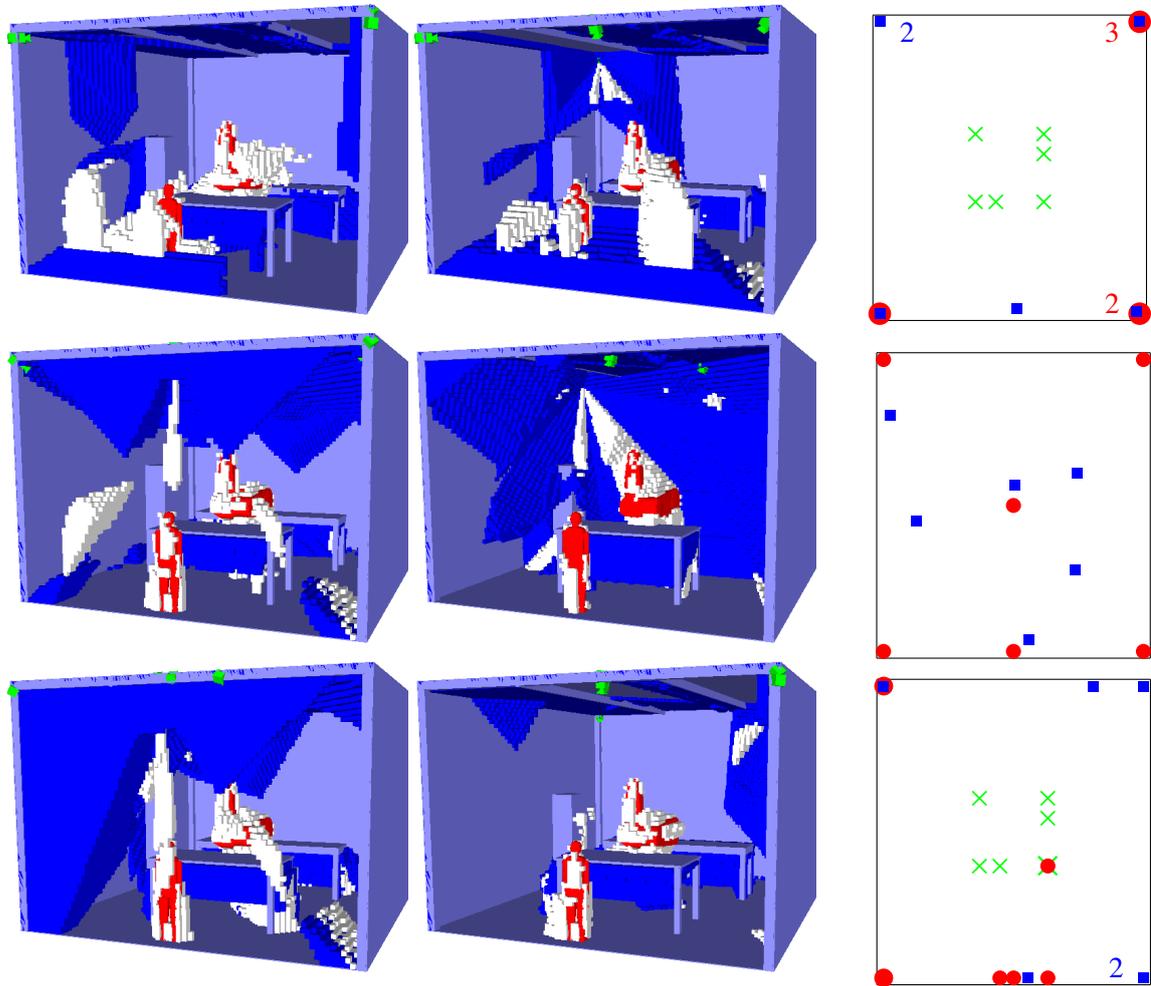


Figure 4.2: Left two columns: Screen shots of the CAD robot cell (grey) with dynamic objects (red) and final positions of the cameras (green) with voxels that are marked as undetectable by all cameras (blue) and voxels that are marked as changed by at least one camera (white). The white and blue voxels represent the fused coverage $C(N, \{changed, out\ of\ range, occluded\})$, the empty space represents $C(1, \{identical\})$. Each screen shot shows the result of one method: BCAIR (left, top), BCADR (middle, top), heuristical placement (left, middle), random placement (middle, middle), Sbp1x (left, bottom), and the original RBF solver (middle, bottom).

Right column: Plot of the positions of the cameras as top view. The open front of the robot cell corresponds to the bottom horizontal line. The method which is illustrated in the same row on the left side is represented by red circles, the method in the middle by blue squares, the initial positions by green crosses. The numbers next to the marks represent the number of cameras larger than 1 which are mounted on the same spot. The optimized orientations (not illustrated) in such a case differ completely. The result suggests that a camera with a wider opening angle is to be used instead, an uncommon placement of which a human may not have thought, right away.

The corners, edges, ceiling, floor, or walls of a room are usually used as guidelines to place the cameras heuristically. Not surprisingly, the random placement has generated camera positions which are not related to any guideline. The tested optimization solvers, including the solvers developed in this thesis, the original RBF solver, and the Sbplx, have found both guided positions and camera positions in the middle of the ceiling not aligned to any edge or corner of the ceiling. The final positions of the Sbplx, however, are either aligned to an edge or the initial positions of the cameras before the optimization. In contrast to this, our methods have also found innovative, non-intuitive positions and orientations which are neither related to the initial points nor aligned to the edges of the ceiling which may have been neglected in a human chosen heuristic or the Sbplx. Moreover, our solvers are able to suggest an enlarged opening angle of a camera, even though the opening angle does not belong to the variables of the optimization. If a camera with a wider opening angle has a higher priority than several cameras with a narrower opening angle, the solvers place more than one camera at the same spot with varying orientation. Again, this placement may have been neglected in a non-automated heuristic.

4.3 Generic Room with a Human Walk

In a robot cell as well as other public areas, humans and robots move in time. In the last section, we have discussed that objects, such as targets and static as well as dynamic obstacles, have already been incorporated within the objective function. Furthermore, we have investigated the behavior and performance of the proposed methods with such an objective function.

In contrast to the incorporation of objects of one moment in time, we have a closer look upon several differing places and poses of the target, in this section. In this test, a three-dimensional placement of cameras is addressed regarding the movement of the target. Therefore, we assume that the target moves on a trajectory in time. We assume that this trajectory is discretized and the important poses of the target are known. When optimizing for only one of the poses, such as in the last test, the coverage of a second pose might not be good enough for the later application. This is why the incorporation of these trajectory poses is relevant in an optimization. First, the setup of the test is addressed and then the qualitative results are discussed. The quantitative results are compared in Section 4.4.

4.3.1 Setup

For this test, we have generated a generic room with several obstacles: Four pillars, a wall with a door, and a box that hovers above the ground. Doors, in general, are the most frequented nodes of a house, which is why a human walking through the door is utilized as a target, here. The movement has not been incorporated in the last test since the illustration of both human and robot movement is confusing in one image. The basic setup is depicted in Figure 4.3 (left).

The CAD model includes a representation for the movement of the human. However, in Section 2.2, only the visibility analysis and coverage generation of one single pose in this movement has been addressed. This issue has not been covered much in literature, but one approach of incorporating the movement in one scene is sweeping the object along its trajectory, [2]. Unfortunately, when utilizing a background subtraction method to separate targets from obstacles, the incorporation of a swept volume is not advantageous: The swept objects might intersect although the pose of the object at each time does not. Thus,

the fused coverage of the swept volume does not resemble the swept volume of the coverages at each pose.

Instead of the swept volume, here, we use a trajectory that is discretized into time steps, in other words the movement is discretized into several poses of the human. Each time step, the three-dimensional background subtraction method needs to be used to generate the fused coverages. As an objective function of the network optimization, the volumes of these time dependable coverages are averaged. In Figure 4.3 (right) the five time steps are illustrated in one image. Nevertheless, there are five visibility analysis', each of which only incorporates one human pose at a time.

In this environment $N = 4$ cameras are placed all over the generic 3D room with three parameters each, determining where the camera position p is located: $p \in [-3.5, 3.5] \times [-4, 4] \times [0, 2.65]$. The orientation o of the camera is determined by the point $e = p + o$ which the camera eyes directly. Here, the door at $(0,0,1.0)$ is important, so each camera's orientation is set to $o = (0, 0, 1.0) - p$. The initial positions for the BCAIR, BCADR, original RBF solver and Sbplx are $(\pm 0.5, \pm 0.5, 2.0)$. As a suitable heuristic the four corners of the room are used as positions for the four cameras.

The Sbplx, BCAIR, BCADR, and RBF solver are terminated after $F_0 = 50$ function evaluations. The heuristic method and the random placement do not evaluate the objective function. For the BCAIR and RBF solver, four symmetrical subspaces are used, one for each camera. For the BCADR two symmetrical subspaces are used.

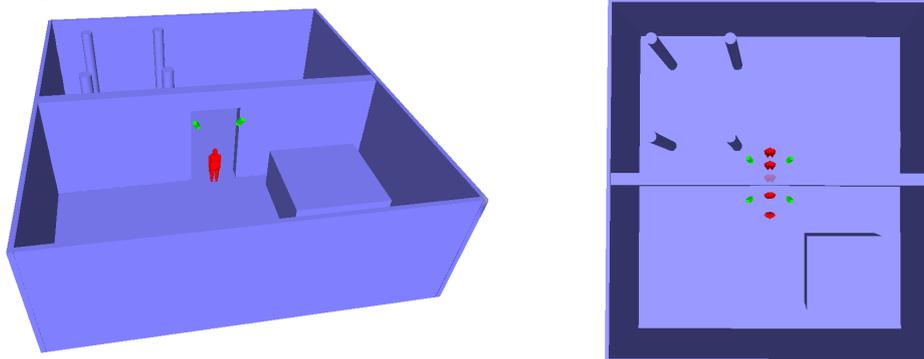


Figure 4.3: Left image: Illustration of the setup; the static (grey) obstacles of the environment include the room ($8\text{m} \times 9\text{m} \times 2.7\text{m}$), four pillars, a wall with a door, and a box that hovers 0.85m above the ground. The target (red human) walks from $(0, -1, 0)$ to $(0, 1, 0)$ through the door. Initially, the $N = 4$ cameras (green) are grouped around the door facing it. Right image: For the optimization, the movement of the dynamic target is discretized into five poses. A visibility analysis has to be done for each pose in order to average the volume of the coverages. The static obstacles encompass 328 vertices and 608 faces and the target has 56k vertices and 58k faces in each time step. The environment is rendered in images of 320×240 pixels and the occupancy grids have $120 \times 135 \times 40$ voxels.

4.3.2 Qualitative Results

In the Figures 4.4 and 4.5, the qualitative result of the solvers and of the random and heuristical placement is illustrated for three of the five poses of the human. The additional two poses were also optimized but

not illustrated, here. In the following section, the qualitative results are discussed regarding the volume of the identical coverage in total, the coverage at the important regions of the room, and the originality of the camera positions.

Additionally, we aim to stress important regions of the room, i.e. an entrance or exit through the door. The result of the three solvers utilizing an RBF as a surrogate cover the important regions around the door along the movement of the human well. The local solver and the random placement have produced an occlusion right in front of the door. The human chosen heuristic produces a large changed part (white voxels) around every human pose. The original RBF solver seems to stress the important poses with the trade off of large undetectable space occurring all around the walls of the room. The consequences that we draw from this test is that the important poses of the room can be stressed even more by an objective function, such as ours, if weights are assigned to the important voxels next to the human position.

Let us consider the maximization of the identical regions (illustrated as empty space), next. In general, the less changed and undetectable coverage (white and blue voxels) are shown in an image, the more identical coverage (voxel free space), and the better is the result. Despite the orientation of the cameras towards the door, the coverage of the random placement is unacceptable for the last two poses of the human. This result shows that a suitable placement can enlarge the visible area by a huge factor. The methods proposed in this thesis display the most voxel-free space in the figures.

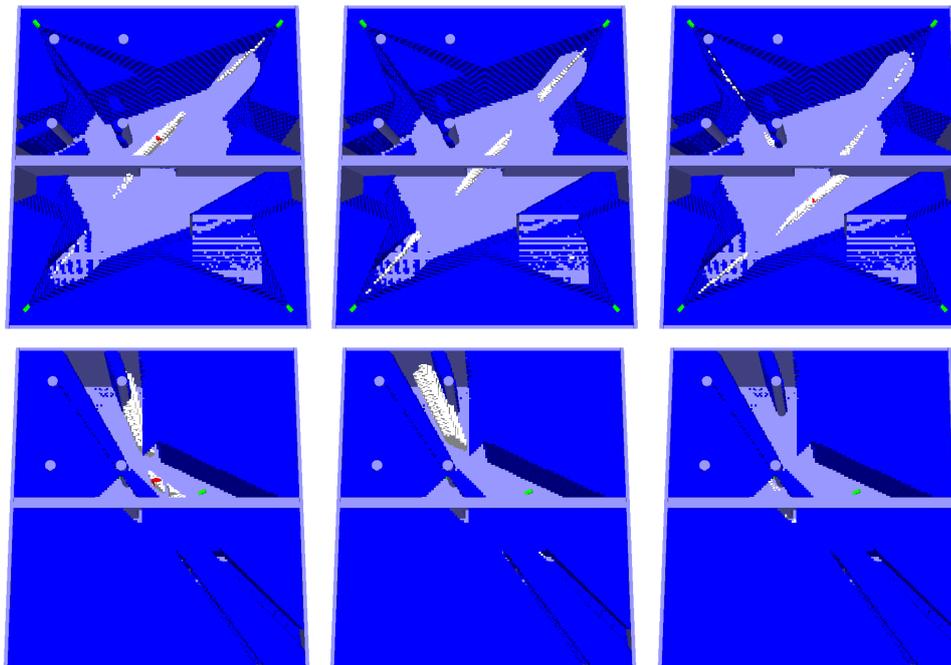


Figure 4.4: Screen shots of the generic room (grey) with a dynamic target (red) and final positions of the cameras (green) with voxels that are marked as undetectable by all cameras (blue) and voxels that are marked as changed by at least one camera (white). All the visible voxels represent the fused coverage $C(N, \{changed, out\ of\ range, occluded\})$, the non-visible voxels represent $C(1, \{identical\})$. Columns: Time step one, three, and five (left to right); Rows: Heuristic (top), and random placement (bottom).

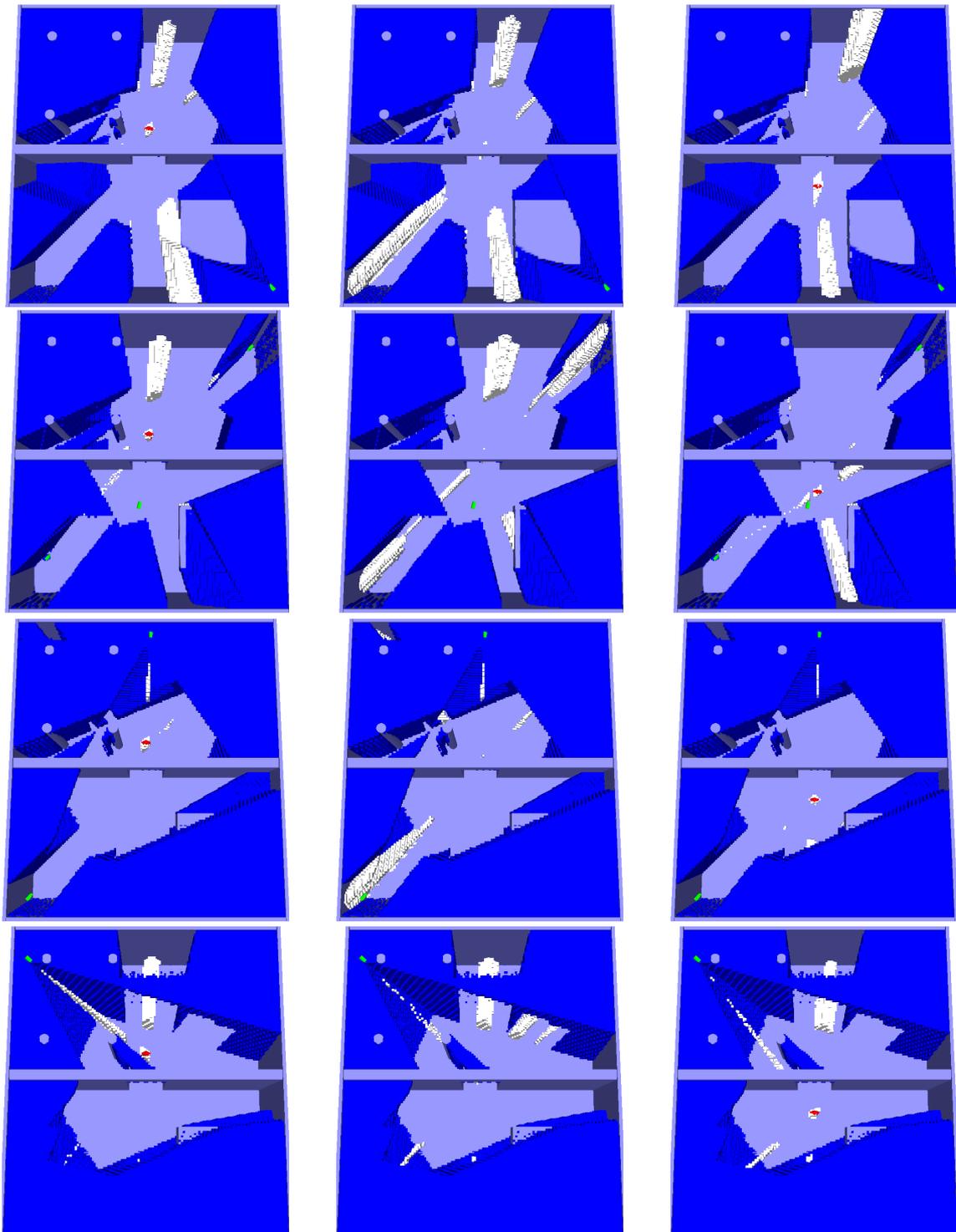


Figure 4.5: Screen shots of the generic room (grey) with a dynamic target (red) and positions of the cameras (green) with voxels that are marked as undetectable by all cameras (blue) and voxels that are marked as changed by at least one camera (white). All the visible voxels represent the fused coverage $C(N, \{changed, out\ of\ range, occluded\})$, the non-visible voxels represent $C(1, \{identical\})$.

Columns: Time step one, three, and five (left to right); Rows: The result of each solver (top to bottom), BCADR, BCAIR, RBF solver, and Sbplx.

In the four solver tests, none of the cameras were placed exactly at the same spot, in contrast to the placement inside the robot cell. One reason could be that the cameras were allowed to be placed in the whole room, the positions were not limited to the ceiling. Thus, the domain for the positions is larger. Another reason may be that a smaller number of cameras (four instead of six) is optimized in a relatively large room (the robot cell has about a quarter of the volume) with a large occluding obstacle (the wall). The four solvers have found innovative positions:

$$\begin{array}{ll} \text{BCADR: } \begin{pmatrix} -3.5 \\ 4 \\ 2.65 \end{pmatrix}, \begin{pmatrix} -3.5 \\ -4 \\ 0 \end{pmatrix}, \begin{pmatrix} 0.898 \\ 4 \\ 0 \end{pmatrix}, \begin{pmatrix} 0.584 \\ 4 \\ 0 \end{pmatrix}, & \text{BCAIR: } \begin{pmatrix} 0.302 \\ 1.4113 \\ 1.1 \end{pmatrix}, \begin{pmatrix} -3.5 \\ -4 \\ 1.054 \end{pmatrix}, \begin{pmatrix} 3.5 \\ 3.249 \\ 0 \end{pmatrix}, \begin{pmatrix} 1.520 \\ -4 \\ 0.251 \end{pmatrix}, \\ \text{RBF solver: } \begin{pmatrix} 3.5 \\ 4 \\ 2.02 \end{pmatrix}, \begin{pmatrix} -0.149 \\ -4 \\ 2.65 \end{pmatrix}, \begin{pmatrix} -3.5 \\ 4 \\ 0 \end{pmatrix}, \begin{pmatrix} -3.5 \\ -4 \\ 0 \end{pmatrix}, & \text{Sbplx: } \begin{pmatrix} 3.5 \\ 4 \\ 2 \end{pmatrix}, \begin{pmatrix} 3.5 \\ -3.5 \\ 2.65 \end{pmatrix}, \begin{pmatrix} 0.334 \\ 4 \\ 1.51 \end{pmatrix}, \begin{pmatrix} 3.5 \\ 2.28 \\ 2.41 \end{pmatrix} \end{array}$$

It is remarkable, that all the cameras in each result, except one camera from BCAIR, are placed at the boundary of the room. In particular, all cameras except one are placed at one of the walls, at least. Two of the cameras in the result of the BCADR and two in the RBF solver are even positioned in corners of the room. In total, six cameras are placed on the floor and three on the ceiling. The reason for the walls being more attractive than the ceiling or floor can be due to the model of the room. It is a relatively flat room (2.65 m) in comparison to the large sides (8m and 9m). The opening in the wall, the door, is the reason for the floor being more attractive than the ceiling: It is not open all the way to the ceiling.

The qualitative results, which leave out the exact detail of the objective value and the computation time spent on an iteration, are now discussed in comparison to the quantitative results.

4.4 Quantitative Results

We have seen in the qualitative results of the Sections 4.3.2 and 4.2.2 that the solvers utilizing an RBF as a surrogate take the lead when considering the identical coverage of the environments, the robot cell in particular. The RBF solver seems to fulfill the goal best, but we will now see that the quantitative results tell otherwise.

In general, the maximum objective value of an environment consists of the weighted number of voxels in an occupancy grid multiplied by the number of the time steps of the target's trajectories. In Table 4.2, the solvers, the heuristic, and the random placement are opposed in order to compare the objective value after 50 objective function calls (second and third column) in each test environment. The coverage of neither method has reached the maximum number of voxels, which is due to the fact that the voxels included in an object cannot be covered by any camera. The optimization procedure is not affected by excluding these voxels, just the maximum objective value is.

In the third and fourth column, of the same table the percentage of the objective value in relation to the maximum objective value is depicted. The table shows that the random placement achieves about half the objective value than the other methods regarding both environments, despite the seemingly good result regarding the robot cell in Figure 4.2. Also the original RBF solver seemed to be better than our methods at a first glance at the images in the same figure. To the contrary, the percentage of the

maximum objective value of the RBF solver regarding the robot cell is worse than the percentage of both our proposed solvers, BCAIR and BCADR. Additionally, the percentage regarding the generic room is worse than the BCAIR.

| | Objective value | | Percentage of maximum | | Computation time | |
|------------|-----------------|--------------|-----------------------|--------------|------------------|--------------|
| | Robot cell | Generic room | Robot cell | Generic room | Robot cell | Generic room |
| BCAIR | 179002 | 912114 | 87.23% | 56.30% | 2.5min | 8min |
| BCADR | 175002 | 876934 | 85.28% | 54.13% | 1.5min | 6min |
| RBF solver | 173668.5 | 896770 | 84.63% | 55.36% | 2min | 8min |
| Sbplx | 163918.5 | 892262 | 79.88% | 55.08% | 1.5min | 7min |
| Heuristic | 167699.5 | 880734 | 81.72% | 54.37% | – | – |
| Random | 96519 | 437370 | 47.04% | 27.0% | – | – |
| Maximum: | 205200 | 1620000 | 100% | 100% | – | – |

Table 4.2: Table of the objective function value $f(x)$ after 50 function calls, the percentage of the objective value in relation to the maximum number of voxels, and the computation time of the iteration versus the four solvers, the heuristic and the random placement. The objective value and computation time is depicted for each test environment, the robot cell of Section 4.2 and the generic room of Section 4.3. The spent time does not only include the iteration of the solver, but also the initial costs, such as the faces being sorted into an octree and the generation of the initial candidates for the first built of the RBF.

The BCAIR has achieved the highest objective value in both tests, the BCADR is second best in case of the robot cell. The reason why the BCADR is not at the top of the list regarding the accuracy may be a result of the termination criterion: Both solvers are stopped after 50 objective function evaluations. However, the BCADR evaluates the objective function several times in parallelizable subspaces during one iteration step, which the BCAIR does not. In the case of the generic room, e.g., there are two additional evaluations in each iteration step. The BCADR has not been parallelized in this test and the additional function evaluations have not been considered in the termination criterion. Thus, the BCADR does not outperform the other methods, as the BCAIR does. However, to be fair, the test results of all the solvers, except for the random placement, in the generic room lie very close to each other (less than two percentage points).

The fact that a solver has not achieved the highest value means that the solver can only have found a local maximum at termination. E.g., in the case of the robot cell, the coverage of all the other solvers and the heuristical placement is better than the Sbplx, which is quite surprising since this solver has performed quite well with the synthetic examples of Section 3.4. In order to prevent such a behavior in the proposed methods, the following three methods can help: Either the initial parameter vector of the network needs to be chosen more carefully for the initialization of the solver, the domain of the variables needs to be narrowed down, or the solver needs more time/function calls.

For placing the cameras in simple environments, it is relatively easy to make up a good heuristic placement. The manual placement in buildings with several rooms and complex obstacles, however, becomes less intuitive and takes consideration time. With the proposed architecture for optimal camera placement the thinking process can be forwarded to the computer which successively generates good placements.

In such a short amount of computing time (1.5min-8min), a human can only make up a rudimentary heuristic for a complex environment let alone consider a series of good placements as the solver does.

The computation time of the methods, depicted in the sixth and seventh column of the Table 4.2, consists of the initial costs and iteration costs. The initial costs account for about 26 seconds in the case of the robot cell and 1.8 minutes in the case of the generic room and consist of loading the CAD model of room, obstacles, and targets into the RAM, sorting the faces into an octree, and voxelizing the interior of objects. The time consumption of the iteration has the costs of 50 function calls.

It can be seen that the computation times of the solvers in case of the robot cell are about a quarter of the computation times in case of the generic room. This is due to the fact that five scenes with different poses of the human have to be processed in the initial costs as well as in each function call. The time consumption of the BCAIR and the RBF solver is higher than the consumption of the BCADR for the same amount of function calls. The reason most likely is the following: Both BCADR and BCAIR incorporate the method of Recalculation in subsequent function calls (Section 2.2.3). However, only the BCADR calls the objective function on orthogonal subspaces in subsequent function calls. This is why the BCADR leads the table considering the computation times, although it has not even been computed in parallel, yet.

The time consumption depicted here is the total amount of time used by all the subspaces of the BCADR. The distinct subspaces of the BCADR can be computed in parallel. Here, we have used three subspaces in the case of the robot cell and two in the case of the generic room. This means only $\frac{3}{5}$ of the number of function calls of the BCADR in case of the robot cell and $\frac{3}{4}$ of the number of function calls need to be computed sequentially, cf. Lemma 3.2.13. We have not distributed the method here since we have used an OpenGL parallelized version for generating the coverage of each camera. In a distributed version, the GPU either needs to be shared between threads or several GPUs need to be addressed. In order to parallelize the BCADR for this particular objective function, further research should go into a way how to incorporate several k -reliable coverages on the same GPU or distribute them onto several GPUs. This consideration shows that there is much potential when further researching this topic.

4.5 Summary

The aim of camera network optimization discussed in this chapter is to approximate the human in different kinds of poses despite obstacles and to cover the important areas of the room. The volume of the 1-reliable identical coverage of the room, which is the area that is labeled with “identical” by at least one camera, is used as an objective function. In this chapter, we have compared the solvers BCADR, BCAIR, Sbp1x, and the original RBF solver with a human chosen heuristic placement of the cameras and a random placement regarding these three goals. The qualitative results of two test environments are depicted in the Sections 4.2.2 and 4.3.2 and a quantitative comparison can be observed in Section 4.4.

The solvers placed the cameras at the boundary of the domain in most cases: In case of the placement at the ceiling of the robot cell in Section 4.2, all except two cameras were placed in the edges of the ceiling. In case of the three-dimensional placement in Section 4.3 all except one were placed at a wall, ceiling, or floor. Qualitatively, the approximation of the human poses could be improved by assigning higher weights to the voxels in the neighborhood of the human.

The quantitative results tell that the BCAIR outperformed all the other methods in accuracy of both test environments. The BCADR leads the table of computation times. To speed up the latter optimization method even further, the objective function calls can be computed in parallel. In general, the chosen heuristics have been good choices. The consideration of a heuristic placement, however, takes consideration time for the human and cannot be automated. Furthermore, an optimization method for optimal camera placement presents positions and orientations of the cameras that a human may not think of, e.g., two or three coinciding positions of cameras that enlarge the viewing angle in one corner, cf. Section 4.2.2.

Chapter 5

Conclusion

In contrast to existing work (Section 1.3.1), the problem of camera placement is considered in this thesis on a continuous domain in a three-dimensional environment regarding static and dynamic obstacles. In Section 1.1.3 some general goals have been specified: Global convergence, efficiency, flexibility, and the production of a solution in anytime. For accelerating the whole iteration, the optimization method should be ready to incorporate prior information of the objective function and be ready for parallel computation.

In this chapter, we provide an overview on the proposed methods and summarize the thesis (Section 5.1). Then, the contribution of this thesis is addressed in context of the predefined goals (Section 5.2). Last, in Section 5.3, open questions are presented.

5.1 Summary

Figure 5.1 shows an overview of the proposed architecture for camera placement. The number of cameras of the network is $N \in \mathbb{N}$. As the diagram shows, the thesis includes two parts, the components for analyzing and implementing the objective function (top), and the solver components (bottom).

In Chapter 2, we analyze the objective function and develop strategies for an accelerated implementation. The key to objective function is the k -reliable coverage of Section 2.1. It can be used to express both an approximation of a target or the detectable regions of a room. The chapter contains the following facts about the coverage:

- An acceleration of the construction of the k -reliable coverage (Section 2.2)
- An analysis of the shape of the k -reliable coverage (Section 2.3),
- An analysis and derivation of the properties of the volume of the k -reliable coverage (Section 2.4).

These properties greatly influence the choice of the solver. In particular, as we have investigated, the evaluations of the objective function are expensive. Additionally, the function can be stair-cased, non-differentiable or non-continuous, non-convex, and may only be given as a black-box, so no gradient can be derived analytically.

In contrast to the issues of the objective function, there are also some good news: If the cameras are identically built, the function is symmetric, i.e. invariant under permutation of the cameras. Also, the recalculation of the objective function is substantially cheaper on subspaces of the domain:

In Chapter 3, suitable solvers are developed that make use of these properties in order to cope with the issues of the objective function. The issues of lacking a gradient, of the stair-casing, and of the expensive function calls have been compensated by optimizing on an approximation of the objective function, instead of the actual objective. Such a function approximation is called *surrogate* or *response surface model* in terms of optimization. Furthermore, the solvers are developed to only call the function on subspaces of the domain. This development was achieved in the following way:

- In Section 3.1, an optimization method on a surrogate has been adapted to incorporate *prior* information, such as symmetry.
- In Section 3.2, a Block Coordinate Ascent (BCA, Iterations 3.2.1 and 3.2.2) has been developed that works with such a surrogate. The method optimizes on a series of subspaces of the domain, in particular each subspace includes the parameters of one single camera instead of all the parameters of all cameras.
- This combination of BCA and a surrogate was then adapted for parallel computing (Section 3.2).

In Figure 5.1, the general method of the three developed solvers is demonstrated (bottom grey box). Starting at an initial point (red point) the solvers search for the best solution in one subspace (shaded in blue) on the surrogate (red graph). After having found the maximum, the subspace is changed and the iteration is continued on the next subspace of the domain. After the last subspace, the optimization restarts the iteration with the first subspace, again.

The objective function is only evaluated at such a changing point. The sample pair consisting of point and function value is updated to the surrogate, meaning the future surrogate interpolates all the previous sample pairs. This results in the following three versions of Algorithm 3: An original version of a **B**lock **C**oordinate **A**scent (BCAIR) that solves all the subspace optimizations on an **I**nvariant **R**esponse surface model, a version that **U**pdates the response surface model at each changing point (BCAUR), and a version of the last **D**istributing the function calls (BCADR). The efficiency of the proposed approaches has been demonstrated on several synthetic (Sections 3.3 and 3.4) functions and realistic examples (Chapter 4).

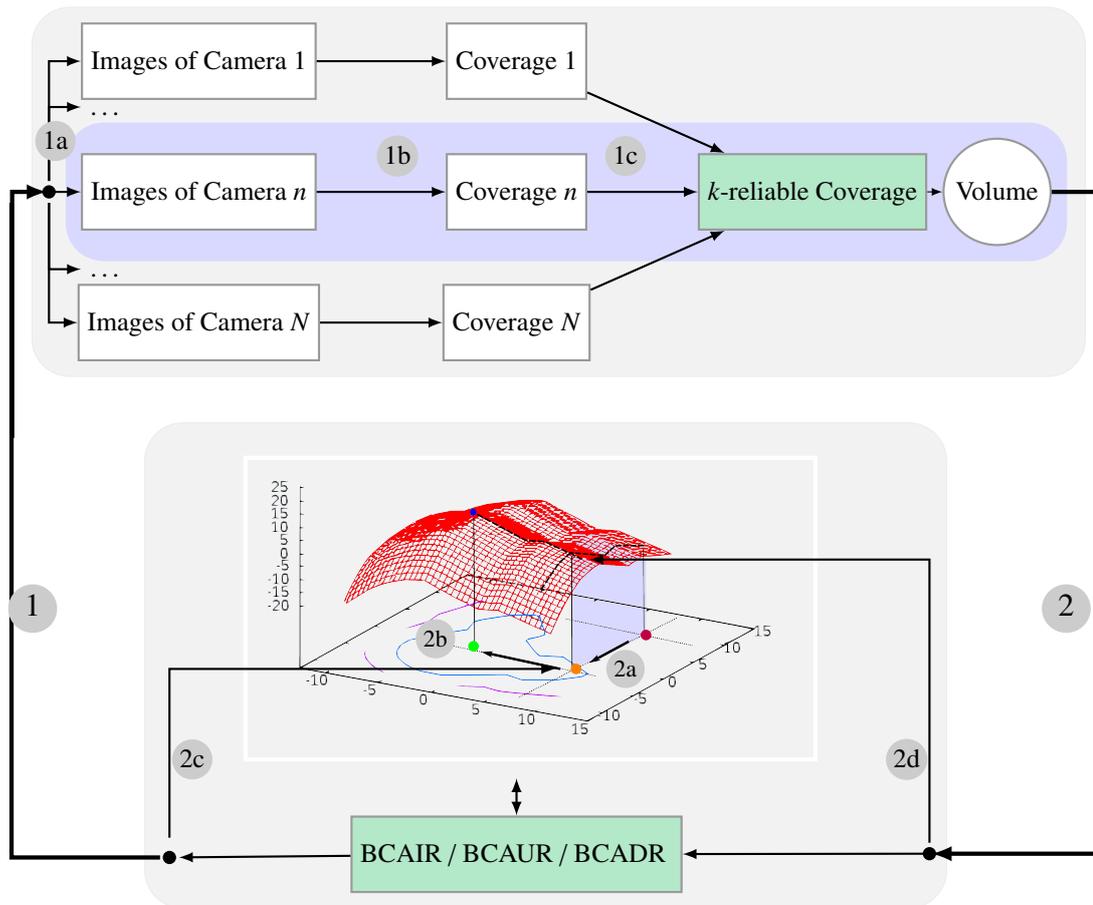


Figure 5.1: Diagram of the implemented architecture for camera placement in a network separated into two parts, the objective function (top grey box) and the solver (bottom grey box); The solver suggests the positions and orientations of the cameras **(1)** and the objective function provides the quality of the placement **(2)** from which the solver generates the next placement.

The objective value of a camera placement is calculated by synthesizing several images of the environment **(1a)** from which the coverage of the camera in the scene is established **(1b)**. Their intersection or union yields the k -reliable coverage **(1c)**, whose volume is used as the quality. The function call is substantially cheaper if only the placement of one camera $n \in \{1, \dots, N\}$ changes (top blue shaded area instead of top grey box). Additionally, the function is invariant under the permutation of the cameras (parallel rows).

The parameters of one single camera define a subspace of the whole domain. Beginning at initial parameters (red point), the solver searches the surrogate (red graph) for the best solution in one subspace of the domain **(2a)**. After having found the maximum in the subspace, the subspace is changed **(2b)**. The expensive objective function is only evaluated at such a change (orange or green point). The pair of candidate **(2c)** and function value **(2d)** is then incorporated into the surrogate.

5.2 Contributions

The proposed architecture of the program, summarized in the last section, is now discussed in the context of the goals that we wanted to achieve: The positions and orientations of the sensors of a camera network utilized in a three-dimensional room must be optimized on a continuous domain. The optimization must be global convergent and an anytime system after an initialization phase. It needs to be efficient, ready for the incorporation of prior information, parallelizable in order to increase the efficiency in further projects, and flexible to the type of objective function. In the following, my contribution to camera placement is stated. It can be grouped into two sections, the analysis of the objective function and the achievement of the computational goals.

Analysis

The key to the flexibility of the objective function is the k -reliable coverage, it can be used to express both an approximation of the target or the detectable regions of a room. Moreover, the k -reliable coverage approximates the target conservatively (Lemmas 2.1.9 and 2.1.10) and can be extended to a more failure resistant version (Lemma 2.1.12, Theorem 2.1.13, and Corollary 2.1.14). The shape of the coverage has been investigated neglecting any kind of discretization, at first. In Lemmas 2.3.2, 2.3.3, 2.3.4, and Theorem 2.3.5, we have proved that the shape of the k -reliable coverage for multiple cameras is a polyhedral area.

Its vertices (classified in Figure 2.16) and faces (Lemma 2.3.7) can be used to calculate the volume of the shape. Thereby, the vertices and faces are also relevant to the non-differentiable and stair-casing behavior of the volume of the k -reliable coverage. In particular, a critical event is the contact of a face of the k -reliable coverage and a given point of the environment, e.g., a vertex of the coverage. The parameters of the network where such an event takes place can be divided into null sets in the network's parameter space which are called *incidence surfaces* (classified in Theorem 2.3.9 and the Tables 2.3, 2.4, and 2.5). In a two-dimensional environment the classification of the incidence surfaces is less complicated since there is only one kind which is a line segment, cf. [58], while in three dimensions there are several types which have a more general, non-planar shape. There are two good reasons to study these surfaces:

- We showed in Lemmas 2.4.1, 2.4.2, and Theorem 2.4.3 that the volume of the k -reliable coverage is in C^∞ unless the parameters of the network are in an incidence surface where a face of the coverage meets another *vertex* of the coverage, and unless the camera position is in an affine subspace defined by the edges of the environment. Thus, the volume is in C^∞ μ -almost everywhere.
- In the fully discretized problem, the stair-casing of the objective function has been analyzed. The steps of the “stair-cases” are separated by the incidence surfaces where a face meets an individual *voxel* (Lemma 2.4.7).

These characterizations conclude the purely analytic part of the thesis.

Achievement of Computational Goals

In case of a convergence, a regular BCA reaches a stationary point. On a particular type of function, an *additively separable* function, a stationary point is found in one iteration step, cf. Corollary 3.2.7 and Lemma 3.2.8. But stationary points are not necessarily actual maxima, see Example 3.2.9, Lemma 3.2.11, and the paragraphs thereafter. Unfortunately the objective function is neither additively separable nor differentiable everywhere. In fact, Figure 3.7 shows a stationary point that is not a global maximum in the context of camera placement. However, we have achieved the five goals set forth in the introduction:

1. The *global convergence* of the solvers BCAIR, BCAUR, and BCADR is established by the choice of the next initial candidate for the subspace iteration. A strategy of [122], that is provably convergent to the optimal value of a continuous function, has been adopted, where the next initial point is chosen outside of an exclusion area around the previous evaluation points.
2. The convergence of the algorithms with additionally incorporated *prior information* such as symmetry is proved in Lemma 3.1.15, Theorem 3.1.16, and Theorem 3.2.12.
3. The same theorems prove that the demands of an *anytime system* are satisfied as well.
4. The BCADR is proved to converge when calling the objective function in parallel.
5. The *efficiency* of solving the problem of camera network optimization (1.1) is discussed in the following paragraphs.

Three different strategies for increasing the efficiency were proposed: Increasing the density of candidates, accelerating the objective function, and accelerating the iteration. Firstly, the objective function is quite costly. By strategies like computing the image of a single camera on the GPU as described in Section 2.2.2, the computation time has decreased, which also accelerates the optimization. A key strategy for our program to accelerate the objective function is to make use of the following fact. The evaluation of the objective function is substantially accelerated by the following methods:

- The objective function is called on subspaces of the domain, cf. the first strategy of Lemma 2.2.1. The BCAUR or BCADR do subsequently call the objective function on a subspace.
- A less expensive approximation of the objective function is called instead. All methods utilize a surrogate of the objective function that interpolates the previous function evaluations (Proposition 3.1.4).

Secondly, the convergence speed of the solvers depends on the density of the candidates: Thus, the more candidates with a known function value can be generated from a single function evaluation, the fewer function evaluations are required. The efficiency of generating candidates with a known function value is subsumed under the term *effect of prior information*, Lemma 3.1.11, and was increased in the following three ways:

- A strategy as in Example 3.1.12 for increasing the number of candidates once at the beginning of the iteration.

- As the problem is symmetric in the subspaces of the cameras, shown in Lemma 2.4.5, a permutation of the parameter vectors of different cameras does not change the objective value. Thus, after every objective function evaluation the objective values of the symmetric candidates are available. The incorporation of these in the BCAIR, BCAUR, and BCADR has been addressed in Lemma 3.1.14 and increases the convergence speed in practice (Section 3.4.4).
- Instead of generating candidates with a known objective value, we can also generate candidates with a partly known objective value. In one single objective function evaluation, the coverage of each camera needs to be calculated. Two subsequent evaluations contain the coverages of each single camera from two different positions or orientations. The second part of Lemma 2.2.1 shows the number of cheap function evaluations that can be generated from these two subsequent function calls for the developed solver.

Thirdly, the whole iteration is also accelerated by the following strategies:

- The BCADR was designed to facilitate a parallel evaluation of the objective function. The number of parallel function calls of the BCADR can be calculated by Lemma 3.2.13.
- The surrogate of the objective function is continuously differentiable (Lemma 3.1.5). Thus, the optimization on the surrogate of the objective function can be performed by a solver that uses a gradient for higher convergence rate

The experiments on the synthetic functions show that the developed methods BCAIR, BCAUR, and BCADR, are as efficient as local solvers despite the lack of a gradient, as accurate in the choice of the global maximum as stochastic solvers (Section 3.4.1), and outperform the original RBF solver in efficiency (Section 3.4.2). BCAIR evaluates the objective function the least of all tested solvers when incorporating symmetry as *prior* information when computing on one core. The BCAUR/BCADR develops an advantage when the number of parallel threads is higher and the number of symmetric subspaces is lower: This has been the case if more than four subspaces with less than four *symmetric* subspaces (cameras) were used in our example.

In terms of accuracy, BCAIR has outperformed all the other methods in both of the realistic examples (Section 4.4). The accuracy of BCAUR/BCADR was marginally lower than the accuracy of BCAIR for the same limited maximum number of function calls. This is most likely due to the fact that with the same maximum number of function calls the number of iterations within the BCAUR/BCADR is lower.

An optimized placement in contrast to a random placement can double the covered area, this can be observed in both realistic test environments. Even compared to a manual heuristic placement better results can be achieved. For placing cameras in relatively simple environments it is easy to fall back on a heuristical placement but the placement in buildings with several rooms and complex obstacles becomes much less intuitive and takes time for a human as well. With the proposed architecture for camera placement, a system has been developed that generates provably good positions and orientations of cameras in three dimensions that a human might not think of in a reasonable amount of computing time (less than 10 min, Sections 4.2.2, 4.3.2, and 4.4).

5.3 Future Work

The placement of the cameras in a network depends on the model of the environment which the cameras ought to observe. The following question arises: Is it possible to derive provably good positions and orientations of the cameras right from this model? This is a fact that should be researched in further investigations.

The search for an answer to this question can be started with the results that we have achieved: In both test environments the solver has placed more than the three quarters of cameras at the boundary of the room or domain. Another idea to achieve such a derivation is subdividing the environment into primitives and deriving the coverage for the primitives. For example, take the second realistic test environment which is composed of two rooms separated by a wall with a door. In this example, the network configurations achieved better results when incorporating cameras at the floor of the room since the door is not open all the way to the ceiling. One could now decompose the environment into two primitives, a room without a door and an environment consisting only of the wall with a door, derive the coverage for both primitives, and compare the results.

Another good reason for a subdivision into primitives is the following: When two cameras are placed in two separate rooms, the function is additively separable. This is particularly interesting for the BCA used on the surrogate, since it converges to a local maximum in the differentiable case. Without additively separability, we have used the exclusion area strategy in order to prove convergence. Such a strategy is not necessary anymore if the BCA converges on its own. Thus, a subdivision into primitives should be investigated that causes additively separable behavior of the volume of subdivided coverages.

Last but not least, the convergence speed of the solver is as high as a local solver which makes it practical for a variety of “real world problems” which are non-convex, stair-cased, expensive, or given only as a black-box. Additionally, the architecture of the program can be applied to the placement of sensors other than cameras, as well. Their coverage is not necessarily a polyhedral area and its volume might be less differentiable or less continuous, which needs to be integrated in the system in that case. As soon as this is established, the methods will work for the sensors the same way as in case of the cameras as sensors. They have found several high quality but non-intuitive positions and orientations for the cameras, such as a position on the floor of the room.

Appendix A

Bibliography

- [1] B.R. Abidi, N.R. Aragam, Y. Yao, and M.A. Abidi. Survey and analysis of multimodal sensor planning and integration for wide area surveillance. *ACM Computing Surveys (CSUR)*, 41(1):1–36, 2008.
- [2] S. Abrams, P.K. Allen, and K. Tarabanis. Computing camera viewpoints in an active robot work cell. *International Journal of Robotics Research*, 18(3):267–285, 1999.
- [3] M. Ahmed. *Decentralized Information Processing in Wireless Peer-to-Peer Networks*. PhD thesis, University of California, 2002.
- [4] I.F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci. Wireless sensor networks: a survey. *Computer networks*, 38(4):393–422, 2002.
- [5] A. Appel. The notion of quantitative invisibility and the machine rendering of solids. In *Proceedings of the 1967 22nd national conference*, pages 387–393. ACM, 1967.
- [6] A. Appel, F.J. Rohlf, and A.J. Stein. *The haloed line effect for hidden line elimination.*, volume 13. ACM, 1979.
- [7] J.E. Banta, L.R. Wong, C. Dumont, and M.A. Abidi. A next-best-view system for autonomous 3-D object reconstruction. *Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on*, 30(5):589–598, 2000.
- [8] J.P. Barreto, L. Perdigoto, R. Caseiro, and H. Araujo. Active stereo tracking of targets using line scan cameras. *Robotics, IEEE Transactions on*, 26(3):442–457, 2010.
- [9] M. Baumann, S. Léonard, E.A. Croft, and J.J. Little. Path planning for improved visibility using a probabilistic road map. *Robotics, IEEE Transactions on*, 26(1):195–200, 2010.
- [10] M.A. Baumann, D.C. Dupuis, S. Léonard, E.A. Croft, and J.J. Little. Occlusion-free path planning with a probabilistic roadmap. In *Intelligent Robots and Systems, 2008. IROS 2008. IEEE/RSJ International Conference on*, pages 2151–2156. IEEE, 2008.
- [11] A. Beck and L. Tretushvili. On the convergence of block coordinate descent type methods. *SIAM Journal on Optimization*, 23(4):2037–2060, 2013.

- [12] E. Becker, G. Guerra-Filho, and F. Makedon. Automatic sensor placement in a 3D volume. In *Proceedings of the 2nd International Conference on Pervasive Technologies Related to Assistive Environments*, page 36. ACM, 2009.
- [13] D.P. Bertsekas. *Nonlinear programming*. Athena Scientific, 1999.
- [14] D.P. Bertsekas and J.N. Tsitsiklis. *Parallel and distributed computation: Numerical methods*. Athena Scientific, 1997.
- [15] J. Bittner and P. Wonka. Visibility in computer graphics. *Environment and Planning B: Planning and Design*, 30:729–755, 2003.
- [16] M. Björkman and K. Holmström. Global optimization of costly nonconvex functions using radial basis functions. *Optimization and Engineering*, 1(4):373–397, 2000.
- [17] R. Bodor, A. Drenner, P. Schrater, and N. Papanikolopoulos. Optimal camera placement for automated surveillance tasks. *Journal of Intelligent and Robotic Systems*, 50(3):257–295, 2007.
- [18] M.J. Box. A new method of constrained optimization and a comparison with other methods. *The Computer Journal*, 8(1):42–52, 1965.
- [19] I.N. Bronstein, K.A. Semendjajew, G. Musiol, and H. Mühling. *Taschenbuch der Mathematik*. Verlag Harri Deutsch, 2001.
- [20] M.D. Buhmann. Radial basis functions. *Acta numerica*, 9(1):1–38, 2000.
- [21] M.D. Buhmann. *Radial basis functions: Theory and implementations*, volume 5. Cambridge university press Cambridge, 2003.
- [22] B. Chapman, G. Jost, and R. Van Der Pas. *Using OpenMP: Portable shared memory parallel programming*, volume 10. MIT press, 2008.
- [23] Y. Charfi, N. Wakamiya, and M. Murata. Challenging issues in visual sensor networks. *Wireless Communications, IEEE*, 16(2):44–49, 2009.
- [24] S. Chen, Y. Li, and N.M. Kwok. Active vision in robotic systems: A survey of recent developments. *The International Journal of Robotics Research*, 30(11):1343–1377, 2011.
- [25] D.M. Chu and A.W.M. Smeulders. Thirteen hard cases in visual tracking. In *Advanced Video and Signal Based Surveillance (AVSS), 2010 Seventh IEEE International Conference on*, pages 103–110. IEEE, 2010.
- [26] R. Church and C.R. Velle. The maximal covering location problem. *Papers in regional science*, 32(1):101–118, 1974.
- [27] F.C. Crow. Shadow algorithms for computer graphics. In *ACM SIGGRAPH Computer Graphics*, volume 11, pages 242–248. ACM, 1977.
- [28] R. Cutler and M. Turk. View-based interpretation of real-time optical flow for gesture recognition. In *Proceedings of the Third IEEE Conference on Face and Gesture Recognition*, 1998.

-
- [29] Y. Dai and K. Schittkowski. A sequential quadratic programming algorithm with non-monotone line search. *Pacific Journal of Optimization*, 4:335–351, 2008.
- [30] X. Dan. Geometry and the imagination. An academic exercise in partial fulfillment for the degree of bachelor of science with honours in applied mathematics, National University of Singapore, 2004.
- [31] L. De Floriani and P. Magillo. Visibility algorithms on triangulated digital terrain models. *Journal of Geographical Information Systems*, 8:13–41, 1994.
- [32] L. De Floriani and P. Magillo. Algorithms for visibility computation on terrains: A survey. *Environment and Planning B*, 30(5):709–728, 2003.
- [33] H. De Ruiter, M. Mackay, and B. Benhabib. Autonomous three-dimensional tracking for reconfigurable active-vision-based object recognition. *Proceedings of the Institution of Mechanical Engineers, Part B: Journal of Engineering Manufacture*, 224(3):343–360, 2010.
- [34] X. Desurmont, C. Carincotte, and F. Bremond. Intelligent video systems: A review of performance evaluation metrics that use mapping procedures. In *2010 7th IEEE International Conference on Advanced Video and Signal Based Surveillance*, 2010.
- [35] A. Discant, A. Rogozan, C. Rusu, and A. Benschair. Sensors for obstacle detection-a survey. In *Electronics Technology, 30th International Spring Seminar on*, pages 100–105. IEEE, 2007.
- [36] DreamQii. Plexidrone, 10. 2014. <http://plexidrone.com/>.
- [37] G. Drettakis and E. Fiume. A fast shadow algorithm for area light sources using backprojection. In *Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, pages 223–230. ACM, 1994.
- [38] G. Drettakis and F. Sillion. Accurate visibility and meshing calculations for hierarchical radiosity. In *Rendering Techniques' 96*, pages 269–278. Springer, 1996.
- [39] J. Duchon. Splines minimizing rotation-invariant semi-norms in Sobolev spaces. In *Constructive theory of functions of several variables*, pages 85–100. Springer, 1977.
- [40] F. Durand. A multidisciplinary survey of visibility. *ACM SIGGRAPH course notes: Visibility, Problems, Techniques, and Applications*. ACM, 2000.
- [41] F. Durand. *3D visibility: Analytical study and applications*. PhD thesis, Université Joseph Fourier, 2010.
- [42] F. Durand, G. Drettakis, and C. Puech. The visibility skeleton: A powerful and efficient multi-purpose global visibility tool. In *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, pages 89–100. ACM Press/Addison-Wesley Publishing Co., 1997.
- [43] H. Edelsbrunner. *Algorithms in combinatorial geometry*, volume 10. Springer, 1987.
- [44] H. Edelsbrunner. Algebraic decomposition of non-convex polyhedra. In *Foundations of Computer Science, 1995. Proceedings., 36th Annual Symposium on*, pages 248–257. IEEE, 1995.

- [45] A.O. Ercan. *Object tracking via a collaborative camera network*. PhD thesis, Stanford University, 2007.
- [46] A.O. Ercan, A.E. Gamal, and L.J. Guibas. Camera network node selection for target localization in the presence of occlusions. In *In SenSys Workshop on Distributed Cameras*, 2006.
- [47] A.O. Ercan, D.B. Yang, A.E. Gamal, and L.J. Guibas. Optimal placement and selection of camera network nodes for target localization. In *Lecture notes in computer science*, pages 389–404. Springer Verlag Berlin Heidelberg, 2006.
- [48] U.M. Erdem and S. Sclaroff. Optimal placement of cameras in floorplans to satisfy task requirements and cost constraints. In *OMNIVIS Workshop*. Citeseer, 2004.
- [49] R. Fablet and M.J. Black. Automatic detection and tracking of human motion with a view-based representation. In *European Conference on Computer Vision*, 2002.
- [50] L. Fiore, D. Fehr, R. Bodor, A. Drenner, G. Somasundaram, and N. Papanikolopoulos. Multi-camera human activity monitoring. *J Intell Robot Syst*, 52(1):5–43, 2008.
- [51] L. Fiore, G. Somasundaram, A. Drenner, and N. Papanikolopoulos. Optimal camera placement with adaptation to dynamic scenes. In *IEEE International Conference on Robotics and Automation*, pages 956–961, 2008.
- [52] J.D. Foley, A. van Dam, S.K. Feiner, and J.F. Hughes. *Computer graphics, principles and practice*. Addison-Wesley: Cornell, USA, 1997.
- [53] M. Fornasier, A. Langer, and C.-B. Schönlieb. A convergent overlapping domain decomposition method for total variation minimization. *Numerische Mathematik*, 116(4):645–685, 2010.
- [54] M. Fornasier and C.-B. Schönlieb. Subspace correction methods for total variation and ℓ_1 -minimization. *SIAM Journal on Numerical Analysis*, 47(5):3397–3428, 2009.
- [55] R. Franke. Scattered data interpolation: Tests of some methods. *Mathematics of computation*, 38(157):181–200, 1982.
- [56] Free Software Foundation. GNU Scientific Library 1.16-2.1.2. GPL-3.0+ License, 09 2013. <http://www.gnu.org/software/gsl/>.
- [57] R. Galimberti. An algorithm for hidden line elimination. *Communications of the ACM*, 12(4):206–211, 1969.
- [58] A. Ganguli, J. Cortés, and F. Bullo. Maximizing visibility in nonconvex polygons: Nonsmooth analysis and gradient algorithm design. *SIAM Journal on Control and Optimization*, 45(5):1657–1679, 2006.
- [59] J. Gondzio. Stable algorithm for updating dense LU factorization after row or column exchange and row and column addition or deletion. *Optimization*, 23(1):7–26, 1992.
- [60] M.F. Goodchild and J. Lee. Coverage problems and visibility regions on topographic surfaces. *Annals of Operations Research*, 18(1):175–186, 1989.

-
- [61] L. Grippo and M. Sciandrone. Globally convergent block-coordinate techniques for unconstrained optimization. *Optimization methods and software*, 10(4):587–637, 1999.
- [62] H.-M. Gutmann. A radial basis function method for global optimization. *Journal of Global Optimization*, 19:201–227, 2001.
- [63] M.L. Hänel. Optimierung von Kamerapositionen zur Überwachung teils unbekannter Umgebungen. Master’s thesis, University of Bayreuth, Germany, 2010.
- [64] M.L. Hänel, S. Kuhn, D. Henrich, L. Grüne, and J. Pannek. Optimal camera placement to measure distances regarding static and dynamic obstacles. *International Journal of Sensor Networks*, 12(1):25–36, 2012.
- [65] R.L. Hardy. Multiquadric equations of topography and other irregular surfaces. *Journal of geophysical research*, 76(8):1905–1915, 1971.
- [66] I. Haritaoglu, D. Harwood, and L. Davis. Real-time surveillance of people and their activities. *IEEE Trans. Pattern Anal. Mach. Intell.*, 22(8):809–831, 2000.
- [67] R.I. Hartley. Estimation of relative camera positions for uncalibrated cameras. In *Computer Vision-ECCV’92*, pages 579–587, 1992.
- [68] P.S. Heckbert. Discontinuity meshing for radiosity. In *Third Eurographics Workshop on Rendering*, pages 203–226, 1992.
- [69] P.S. Heckbert. Radiosity in flatland. In *Computer Graphics Forum*, volume 11, pages 181–192. Wiley Online Library, 1992.
- [70] R.J. Holt, M. Hong, R. Martini, I. Mukherjee, R. Netravali, and J. Wang. Summary of results on optimal camera placement for boundary monitoring. In *Proceedings of SPIE*, volume 6570, page 657005, 2007.
- [71] E. Hörster and R. Lienhart. On the optimal placement of multiple visual sensors. In *Proceedings of the 4th ACM international workshop on Video surveillance and sensor networks*, pages 111–120. ACM, 2006.
- [72] S.G. Johnson. NLOpt, 08. 2013. <http://ab-initio.mit.edu/wiki/index.php/NLOpt>.
- [73] D.R. Jones. A taxonomy of global optimization methods based on response surfaces. *Journal of global optimization*, 21(4):345–383, 2001.
- [74] G.A. Kaminka, R. Schechter-Glick, and V. Sadov. Using sensor morphology for multirobot formations. *Robotics, IEEE Transactions on*, 24(2):271–282, 2008.
- [75] W. Knight. Kollege Roboter am Fließband, 12. 2014. <http://www.heise.de/tr/artikel/Kollege-Roboter-am-Fließband-1972126.html>.
- [76] C. Kohstall, J. Jovanovic, and M. Niedermayr. Nixie - flyable and wearable camera, 10. 2014. <https://makeit.intel.com/finalists>.

- [77] D. Kraft. Algorithm 733: TOMP–Fortran modules for optimal control calculations. *ACM Transactions on Mathematical Software (TOMS)*, 20(3):262–281, 1994.
- [78] S. Kuhn and D. Henrich. Multi-view reconstruction of unknown objects in the presence of known occlusions. Technical report, Universität Bayreuth, Angewandte Informatik III (Robotik und Eingebettete Systeme), 2009.
- [79] S. Kuhn and D. Henrich. Multi-view reconstruction in-between known environments. Technical report, Lehrstuhl für Angewandte Informatik III (Robotik und Eingebettete Systeme), Universität Bayreuth, 2010.
- [80] Kuka. Mit viel Feingefühl revolutioniert KUKA die Robotik, 12. 2014. http://www.kuka-robotics.com/de/pressevents/news/NN_091203_LBRDaimler.htm.
- [81] L. Lazos, R. Poovendran, and J.A. Ritcey. Detection of mobile targets on the plane and in space using heterogeneous sensor networks. *Wireless Networks*, 15(5):667–690, 2009.
- [82] K. Lee. *Principles of CAD/CAM/CAE systems*. Addison-Wesley Longman Publishing Co., Inc., 1999.
- [83] S. Lien and J.T. Kajiya. A symbolic method for calculating the integral properties of arbitrary non convex polyhedra. *Computer Graphics and Applications, IEEE*, 4(10):35–42, 1984. Dept. of Comput. Sci., California Inst. of Technol., Pasadena, CA, USA.
- [84] D. Lischinski, F. Tampieri, and D.P. Greenberg. Discontinuity meshing for accurate radiosity. *IEEE Computer graphics and Applications*, 12(6):25–39, 1992.
- [85] P.P. Loutrel. A solution to the hidden-line problem for computer-drawn polyhedra. *Computers, IEEE Transactions on*, 100(3):205–213, 1970.
- [86] Z.-Q. Luo and P. Tseng. On the convergence of the coordinate descent method for convex differentiable minimization. *Journal of Optimization Theory and Applications*, 72(1):7–35, 1992.
- [87] R. MacRuairi, M.T. Keane, and G. Coleman. A wireless sensor network application requirements taxonomy. In *Sensor Technologies and Applications, 2008. SENSORCOMM'08. Second International Conference on*, pages 209–216. IEEE, 2008.
- [88] E. Marchand. Control camera and light source positions using image gradient information. In *Robotics and Automation, 2007 IEEE International Conference on*, pages 417–422. IEEE, 2007.
- [89] Éric Marchand, Fabien Spindler, and François Chaumette. ViSP for visual servoing: A generic software platform with a wide class of robot control skills. *Robotics & Automation Magazine, IEEE*, 12(4):40–52, 2005.
- [90] J. Marzal. *The three-dimensional art gallery problem and its solutions*. PhD thesis, Murdoch University, 2012.
- [91] S.O. Mason and A. Grün. Automatic sensor placement for accurate dimensional inspection. *Computer vision and image understanding*, 61(3):454–467, 1995.

-
- [92] B. Maurin, O. Masoud, and N. Papanikolopoulos. Monitoring crowded traffic scenes. In *Proceedings of the IEEE International Conference on Intelligent Transportation Systems*, 2002.
- [93] D.B. McDonald, W.J. Grantham, W.L. Tabor, and M.J. Murphy. Global and local optimization using radial basis function response surface models. *Applied Mathematical Modelling*, 31(10):2095–2110, 2007.
- [94] C.A. Micchelli. *Interpolation of scattered data: Distance matrices and conditionally positive definite functions*. Springer, 1986.
- [95] A. Mittal. Generalized multi-sensor planning. In *Computer Vision—ECCV 2006*, pages 522–535. Springer, 2006.
- [96] A. Mittal and L.S. Davis. Visibility analysis and sensor planning in dynamic environments. In *Computer Vision-ECCV 2004*, pages 175–189. Springer, 2004.
- [97] A. Mittal and L.S. Davis. A general method for sensor planning in multi-sensor systems: Extension to random occlusion. *International Journal of Computer Vision*, 76(1):31–52, 2008.
- [98] H. Mori, M. Charkari, and T. Matsushita. On-line vehicle and pedestrian detections based on sign pattern. *IEEE Trans. Ind. Electron.* 41(4), 41(4):384–391, 1994.
- [99] Yuichi Motai and Akio Kosaka. Hand-eye calibration applied to viewpoint selection for robotic vision. *Industrial Electronics, IEEE Transactions on*, 55(10):3731–3741, 2008.
- [100] J. Müller, T. Kritiyakierne, C. Shoemaker, et al. SO-MODS: Optimization for high dimensional computationally expensive multi-modal functions with surrogate search. In *Evolutionary Computation (CEC), 2014 IEEE Congress on*, pages 1092–1099. IEEE, 2014.
- [101] A.T. Murray, K. Kim, J.W. Davis, R. Machiraju, and R. Parent. Coverage optimization to support security monitoring. *Computers, Environment and Urban Systems*, 31(2):133–147, 2007.
- [102] J.A. Nelder and R. Mead. A simplex method for function minimization. *The computer journal*, 7(4):308–313, 1965.
- [103] Y. Nesterov. Efficiency of coordinate descent methods on huge-scale optimization problems. *SIAM Journal on Optimization*, 22(2):341–362, 2012.
- [104] W. Niem and M. Steinmetz. Camera viewpoint control for the automatic reconstruction of 3D objects. In *Image Processing, 1996. Proceedings., International Conference on*, volume 3, pages 655–658. IEEE, 1996. Min Error, Rekonstruktion, 3D-BG-Subtraction, No Occlusions.
- [105] T. Nishita and E. Nakamae. Continuous tone representation of three-dimensional objects taking account of shadows and interreflection. In *ACM SIGGRAPH Computer Graphics*, volume 19, pages 23–30. ACM, 1985.
- [106] J. Nocedal and S.J. Wright. *Numerical Optimization*. Springer, 2006.

- [107] A. Ober-Gecks, M.L. Hänel, T. Werner, and D. Henrich. Fast multi-camera reconstruction and surveillance with human tracking and optimized camera configurations. In *ISR/Robotik 2014; 41st International Symposium on Robotics; Proceedings of*, pages 1–8. VDE, 2014.
- [108] G. Olague. Design and simulation of photogrammetric networks using genetic algorithm. In *Proceedings of the 2000 Meeting of the American Society for Photogrammetry and Remote Sensing (ASPRS 2000)*, 2000.
- [109] G. Olague and E. Dunn. Development of a practical photogrammetric network design using evolutionary computing. *The Photogrammetric Record*, 22:22–38, 2007.
- [110] G. Olague and R. Mohr. Optimal 3D sensors placement to obtain accurate 3D point positions. In *Proceedings of the Fourteenth International Conference on Pattern Recognition*, volume 1, pages 16–20, 1998.
- [111] G. Olague and R. Mohr. Optimal camera placement for accurate reconstruction. *Pattern Recognition*, 35:927–944, Januar 1998. Publisher: Elsevier.
- [112] OpenMP ARB Corporation. OpenMP, 2014. <http://openmp.org/openmp-faq.html>.
- [113] J. O’Rourke. *Art gallery theorems and algorithms*, volume 1092. Oxford University Press Oxford, 1987.
- [114] Parrot. Parrot’s drone, 08. 2013. www.parrot.com.
- [115] M.J.D. Powell. *The theory of radial basis function approximation in 1990*, chapter 3, pages 105–210. Oxford University Press Oxford, 1992.
- [116] M.J.D. Powell. *Recent research at Cambridge on radial basis functions*. Springer, 1999.
- [117] S. Prestwich and A. Roli. Symmetry breaking and local search spaces. In *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, pages 273–287. Springer, 2005.
- [118] J.-F. Puget. Symmetry breaking revisited. In *Principles and Practice of Constraint Programming-CP 2002*, pages 446–461. Springer, 2002.
- [119] R.J. Radke. *A survey of distributed computer vision algorithms*, chapter 2 of Handbook of Ambient Intelligence and Smart Environments, pages 35–55. Springer, 2010.
- [120] M.K. Reed and P.K. Allen. Constraint-based sensor planning for scene modeling. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 22(12):1460–1467, 2000.
- [121] R.G. Regis. Stochastic radial basis function algorithms for large-scale optimization involving expensive black-box objective and constraint functions. *Computers & Operations Research*, 38(5):837–853, 2011.
- [122] R.G. Regis and C.A. Shoemaker. Constrained global optimization of expensive black box functions using radial basis functions. *Journal of Global Optimization*, 31:153–171, 2005.

-
- [123] Rommel G Regis and Christine A Shoemaker. Combining radial basis function surrogates and dynamic coordinate search in high-dimensional expensive black-box optimization. *Engineering Optimization*, 45(5):529–555, 2013.
- [124] M. Reischl. GPU-Optimierung für die Tiefenbilderstellung. Presentation at the Chair Applied Computer Science III, University of Bayreuth, Germany, June 2013.
- [125] P. Richtárik and M. Takáč. Iteration complexity of randomized block-coordinate descent methods for minimizing a composite function. *Mathematical Programming*, 144(1-2):1–38, 2014.
- [126] L.G. Roberts. Machine perception of three-dimensional solids. Technical report, DTIC Document, 1963.
- [127] K. Römer. *Time Synchronization and Localization in Sensor Networks*. PhD thesis, University of Frankfurt am Main, 2005.
- [128] K. Römer and F. Mattern. The design space of wireless sensor networks. In *IEEE Wireless Communications*, 2004.
- [129] T.H. Rowan. *Functional stability analysis of numerical algorithms*. PhD thesis, 1990. Subplex.
- [130] T.P. Runarsson and X. Yao. Stochastic ranking for constrained evolutionary optimization. *Evolutionary Computation, IEEE Transactions on*, 4(3):284–294, 2000.
- [131] T.P. Runarsson and X. Yao. Search biases in constrained evolutionary optimization. *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, 35(2):233–243, 2005.
- [132] M. Saadat-Seresht, F. Samadzadegan, and A. Azizi. Automatic camera placement in vision metrology based on a fuzzy inference system. *Photogrammetric Engineering and Remote Sensing*, 71(12):1375, 2005. Publisher: ASPRS.
- [133] M. Saadat-Seresht, F. Samdzadegan, A. Azizi, and M. Hahn. Camera placement for network design in vision metrology based on fuzzy inference system! In *XXth ISPRS Congress*. Citeseer, 2004.
- [134] K. Schittkowski. A robust implementation of a sequential quadratic programming algorithm with successive error restoration. *Optimization Letters*, 5(2):283–296, 2011.
- [135] M. Schlüter, J.A. Egea, L.T. Antelo, A.A. Alonso, and J.R. Banga. An extended ant colony optimization algorithm for integrated process and control system design. *Ind. Eng. Chem.*, 48(14):6723–6738, 2009.
- [136] M. Schlüter, J.A. Egea, and J.R. Banga. Extended ant colony optimization for non-convex mixed integer nonlinear programming. *Comput. Oper. Res.*, 36(7):2217–2229, 2009.
- [137] M. Schlüter and M. Gerdt. The oracle penalty method. *Springer Science+Business Media, LLC*, 30:293–325, 2010.
- [138] A. Schrijver. *A course in combinatorial optimization*. TU Delft, 2009.

- [139] H.R. Schwarz and N. Köckler. *Numerische Mathematik*. Springer DE, 2009.
- [140] Silicon Graphics International Corp. OpenGL, 1992. <http://www.sgi.com/tech/opengl/?/overview.html>.
- [141] G.S. Sivaram, M.S. Kankanhalli, and K.R. Ramakrishnan. Design of multimedia surveillance systems. *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMCCAP)*, 5(3):23, 2009.
- [142] B. Song, C. Soto, A.K. Roy-Chowdhury, and J.A. Farrell. Decentralized camera network control using game theory. In *Second ACM/IEEE International Conference on Distributed Smart Cameras*, 2008.
- [143] S. Soro and W. Heinzelman. A survey of visual sensor networks. *Advances in Multimedia*, 2009:1–21, 2009.
- [144] A.J. Stewart and S. Ghali. Fast computation of shadow boundaries using spatial coherence and backprojections. In *Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, pages 231–238. ACM, 1994.
- [145] L. Subramanian and R.H. Katz. An architecture for building self-configurable systems. In *Mobile and Ad Hoc Networking and Computing, 2000. MobiHOC. 2000 First Annual Workshop on*, pages 63–73. IEEE, 2000.
- [146] D. Sunday. Plane intersections, 12. 2014. http://geomalgorithms.com/a05-_intersect-1.html.
- [147] I.E. Sutherland, R.F. Sproull, and R.A. Schumacker. A characterization of ten hidden-surface algorithms. *ACM Computing Surveys (CSUR)*, 6(1):1–55, 1974.
- [148] K. Svanberg. A class of globally convergent optimization methods based on conservative convex separable approximations. *SIAM journal on optimization*, 12(2):555–573, 2002.
- [149] K.A. Tarabanis, P.K. Allen, and R.Y. Tsai. A survey of sensor planning in computer vision. *Robotics and Automation, IEEE Transactions on*, 11(1):86–104, 1995.
- [150] S.J. Teller. *Computing the antipenumbra of an area light source*, volume 26. ACM, 1992.
- [151] N. Tezcan and W. Wang. Self-orienting wireless multimedia sensor networks for occlusion-free viewpoints. *Computer Networks*, 52(13):2558–2567, 2008.
- [152] The MathWorks, Inc. Occupancy Grids, 10. 2015. <http://de.mathworks.com/help/robotics/ug/occupancy-grids.html>.
- [153] A. Törn and A. Zilinskas. *Global optimization, lecture notes in computer science*, volume 350. Springer-Verlag, Berlin, 1989.
- [154] P. Tseng. Convergence of a block coordinate descent method for nondifferentiable minimization. *Journal of optimization theory and applications*, 109(3):475–494, 2001.

-
- [155] A. Wächter and L.T. Biegler. On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. *Mathematical programming*, 106(1):25–57, 2006.
- [156] A. Wächter and C. Laird. IPOPT-3.11. Eclipse Public License, 7. 2014. <http://projects.coin-or.org/Ipopt>.
- [157] C.C. Wang. Approximate boolean operations on large polyhedral solids with partial mesh reconstruction. *IEEE Transactions on Visualization and Computer Graphics*, 17:836–849, 2010.
- [158] K. Weiler and P. Atherton. Hidden surface removal using polygon area sorting. In *ACM SIGGRAPH Computer Graphics*, volume 11, pages 214–222. ACM, 1977.
- [159] T. Werner and D. Henrich. Efficient and precise multi-camera reconstruction. In *Proceedings of the International Conference on Distributed Smart Cameras*, page 23. ACM, 2014.
- [160] M.A. Wesley. Construction and use of geometric models. In *Computer Aided Design Modelling, Systems Engineering, CAD-Systems*, pages 79–136. Springer, 1980.
- [161] K. Yabuta and H. Kitazawa. Optimum camera placement considering camera specification for security monitoring. In *IEEE International Symposium on Circuits and Systems, 2008. ISCAS 2008*, pages 2114–2117, 2008.
- [162] D. Yang, H. González-Baños, and L. Guibas. Counting people in crowds with a real-time network of simple image sensors. In *Proceedings of the Ninth IEEE International Conference on Computer Vision (ICCV 2003)*, 2003.
- [163] D.B. Yang, J. Shin, L.J. Guibas, and A.O. Ercan. Sensor tasking for occupancy reasoning in a network of cameras. In *Proceedings of 2nd IEEE International Conference on Broadband Communications, Networks and Systems (BaseNets' 04)*. Citeseer, 2004.
- [164] K.-H. Yoo, D.S. Kim, S.Y. Shin, and K.-Y. Chwa. Linear-time algorithms for finding the shadow volumes from a convex area light source. *Algorithmica*, 20(3):227–241, 1998.

Appendix B

Symbols and Definitions

| Spaces, sets, and groups | |
|--------------------------|---|
| \mathbb{N}^n | Space of n -tuples of natural numbers |
| \mathbb{Q}^n | Space of n -tuples of rational numbers |
| \mathbb{Z}^n | Space of n -tuples of integer numbers |
| \mathbb{R}^n | Space of n -tuples of real numbers |
| Π_m^n | Space of polynomials of degree less than or equal to m with n variables |
| C^n | Space of n -times continuous differentiable functions |
| $L^p(\mathcal{D})$ | Space of p -integrable functions defined on the domain \mathcal{D} as in Definition 3.1.17 |
| $D^{-k}L^2(\mathcal{D})$ | Space of functions whose k -th total degree distributional partial derivatives are in $L^2(\mathcal{D})$ as in Definition 3.1.17 |
| $\mathcal{P}(S)$ | Power set of a set S : The set of all subsets of S |
| S_N | Symmetric group of degree N |
| $C_N(k) \subset S_N$ | The set of k -combinations of the N -tuple $(1, \dots, N)$. The set has the order $\frac{N!}{k!(N-k)!}$ |
| Set Operations | |
| $[x, b]$ | Line segment between $x \in \mathbb{R}^3$ and $b \in \mathbb{R}^3$ defined by $[x, b] := \{c \in \mathbb{R}^n \mid c = x + \epsilon \cdot (b - x), \epsilon \in [0, 1]\}$ |
| $[x, C]$ | Pyramid between $x \in \mathbb{R}^3$ and a planar $C \subset \mathbb{R}^3$ defined by $[x, C] := \bigcup_{c \in C} [x, c]$ |
| $[x, b)$ | Half line (also ray) starting at $x \in \mathbb{R}^3$ in direction of $b \in \mathbb{R}^3$ defined by $[x, b) := \{y = x + \epsilon(b - x) \mid \epsilon \geq 0\}$ |
| ∂A | Set of boundary points of $A \subset \mathbb{R}^n$ |

| | |
|-------------------------------------|--|
| ${}^c A$ | Closure of $A \subset \mathbb{R}^n$ |
| $\complement A$ | Complement of $A \subset S \subset \mathbb{R}^n$ is $S \setminus A$ |
| $\mathbb{B}_\epsilon^n(b)$ | Open ϵ -ball about $b \in \mathbb{R}^3$ defined by $\mathbb{B}_\epsilon^n(b) := \{y \in \mathbb{R}^n \mid d(y, b) < \epsilon\}$ |
| $\partial \mathbb{B}_\epsilon^n(b)$ | ϵ -sphere about $b \in \mathbb{R}^3$ with $\partial \mathbb{B}_\epsilon^n(b) := \{y \in \mathbb{R}^n \mid d(y, b) = \epsilon\}$ |
| $d(x, b)$ | Distance between $x \in \mathbb{R}^3$ and $b \in \mathbb{R}^3$ defined by $d(x, b) := \ x - b\ _2$ |
| $\lambda(C)$ | Volume of $C \subset \mathbb{R}^3$ |

Function Operations

| | |
|----------------------------|---|
| $J_x(f)$ | Jacobian matrix of a function f with input x |
| D^α | Operator of the partial derivative $D^\alpha := \left(\frac{\partial^{\alpha_1}}{(\partial x)^{\alpha_1}} \cdots \frac{\partial^{\alpha_n}}{(\partial x)^{\alpha_n}} \right)$ with the multi-index $\alpha = (\alpha_1, \dots, \alpha_n) \in \mathbb{N}^n$ of Definition 3.1.7 |
| $\ f\ _{k, \mathcal{D}}^2$ | Semi-norm of the derivatives of a function $f : \mathcal{D} \rightarrow \mathbb{R}$ as in Definition 3.1.7 |

Linear Algebra

| | |
|------------------------------------|---|
| $\mathbb{1}_n$ | n -dimensional unit matrix |
| $(U_1, \dots, U_M) = \mathbb{1}_n$ | Partition of the n -dimensional unit matrix into M matrices |
| $\mathbf{1}$ | Matrix or vector of ones |
| $\mathbf{0}$ | Matrix or vector of zeros |

Sets of the Camera Network Application

| | |
|--------------|---|
| \mathbb{E} | Set of all environments which need to be monitored and where the sensors can be placed, first mentioned in Section 1.2 |
| \mathbb{A} | Surveillance area of the camera network application, first mentioned in Section 1.2 and further specified in Section 2.1.2; In this thesis, the surveillance area is a polyhedral area. The points in this area are usually denoted by $y \in \mathbb{A}$ |
| \mathbb{P} | Space of parameters of one single camera, first mentioned in Section 1.2 and further specified in Equation (2.1): The points in this space are tuples of parameters and are denoted by $a \in \mathbb{P}$ |
| \mathbb{S} | Set of sensor labels, first mentioned in Section 1.2 and further specified in Section 2.1.2; A subset of this set is usually denoted by $S \subset \mathbb{S}$ |

| | |
|------------------------|--|
| $E \in \mathbb{E}$ | Particular environment which needs to be monitored, which models the furniture, walls, and the places of the cameras. In this thesis, the environment is the empty space of a room, modeled by a polyhedral area as defined in Definition 2.1.1 and Notation 2.1.2. The boundary of the environment is composed of static $E_s \subset E$ and dynamic objects $E_d \subset E$, both surrounding polyhedral areas, as in Definition 2.1.6. Objects can be targets T , i.e. objects that are to be approximated as in Definition 2.1.8, and obstacles, otherwise. |
| $\mathbb{L} \subset E$ | Space of camera positions first mentioned in the Example 1.2.2 |

Camera Parameters

| | |
|--|--|
| $N \in \mathbb{N}$ | Number of sensors/cameras |
| $p \in E$ | Position of a camera defined in Definition 2.1.3 |
| $o \in \mathbb{B}_1^3(0)$ | Orientation of a camera defined in Definition 2.1.3 |
| $u \in \mathbb{B}_1^3(0)$ | Vector defining the two openings of a camera frustum and the orientation of the camera coordinate system (Definition 2.1.3) |
| $\theta_u, \theta_{o \times u} \in [0, \pi]$ | Opening angles of one camera with direction o and opening vector u and $o \times u$, respectively (Definition 2.1.3) |
| $T_d(\theta, v)$ | θ -space with direction d and opening vector v as defined in Definition 2.1.4: A θ -space is the intersection of two half-spaces with an angle of $\theta \in [0, \pi)$ whose boundary normals add up to d and can be linearly combined to d . |
| $\mathbb{V}(p, S)$ | Set of all visible points from point $p \in S$ inside the set S disregarding the opening angle of a camera (Definition 2.1.4) |
| \mathbb{F}_a | Frustum of the camera with parameters $a = (p, o, u, \theta_u, \theta_{o \times u}) \in \mathbb{P}$ (Definition 2.1.4) |
| $\mathbb{V}_a(S)$ | Field of view of a camera with parameters $a = (p, o, u, \theta_u, \theta_{o \times u}) \in \mathbb{P}$ inside a set S defined by $\mathbb{V}_a(S) := \mathbb{V}(p, S) \cap \mathbb{F}_a$ in Definition 2.1.4 |

Notation in the Context of the Coverage of a Camera Network

| | |
|--|---|
| $\sigma : \mathbb{E} \times \mathbb{P} \times \mathbb{A} \rightarrow \mathbb{S}$ | Function defined in Definition 1.2.1 which maps a surveillance part $y \in \mathbb{A}$ with the camera parameters of $a \in \mathbb{P}$ and the constraints of the environment $E \in \mathbb{E}$ onto the set of sensor labels. Thus, a camera or general sensor becomes a device to classify regions or parts of the surveillance area, e.g. by the labels “detectable” and “undetectable”, or by the labels “changed” and “identical”. |
| $\sigma_a^{-1}(S)$ | Coverage of one single camera defined in Definition 1.2.1: The set is the preimage of the sensor function σ in a chosen environment with camera parameters $a \in \mathbb{P}$ and with the sensor labels $S \subset \mathbb{S}$ |

| | |
|--|---|
| $\mathbb{P}_1 \times \dots \times \mathbb{P}_N$ | Space of parameters of the whole camera network first mentioned in Definition 1.2.1. The elements of this set are denoted by $x = (a_1, \dots, a_N)$. |
| \mathcal{A} | Conservative approximation of a given target defined in Definition 2.1.8 |
| $C(k, S)$ | Coverage of a network of N sensors with reliability $k \in \mathbb{N}$ and the sensor labels $S \subset \mathbb{S}$ first mentioned in Definition 1.2.1 and refined in Definition 2.1.11: The coverage includes all parts of the surveillance area that are marked as one of the labels in S by at least k cameras. |
| $\pi \in C_N(k)$ | k -Combination of the N -tuple (a_1, \dots, a_N) ; The set is used to denote the coverage in Definition 2.1.11. The elements are determined by choosing k sensors out of N total without replacement disregarding the sequence of the tuple. |
| $q : \mathbb{P}_1 \times \dots \times \mathbb{P}_N \rightarrow \mathbb{R}$ | Quality of the network measured as the volume of the coverage as defined in Equation (1.2) |
| \mathcal{P} | A not necessarily convex nor manifold polyhedral area as defined in Definition 2.1.1 |
| $\mathcal{V} \in \partial\mathcal{P}$ | Vertex of the polyhedron or polyhedral area \mathcal{P} defined in Definition 2.1.1 |
| $\mathcal{E} \subset \partial\mathcal{P}$ | Edge of the polyhedron or polyhedral area \mathcal{P} defined in Definition 2.1.1; In the context of projection and silhouette faces an edge of the environment is also called anchor as in Lemma 2.3.7 |
| $\mathcal{F} \subset \partial\mathcal{P}$ | Face of the polyhedron or polyhedral area \mathcal{P} defined in Definition 2.1.1 |
| $\sigma_{(E,a)}^{-1}(\{ident.\})$ | Identical coverage of one camera with parameters $a = (p, o, u, \theta_u, \theta_{o \times u}) \in \mathbb{P}$ inside the environment $E \in \mathbb{E}$. If E is a polyhedral area, the coverage is a polyhedral area, as well, and has the faces defined in Definition 2.3.6: Environmental faces (\mathcal{F}_E), projection faces (\mathcal{F}_P), opening faces (\mathcal{F}_O), and silhouette faces (\mathcal{F}_S). Projection and silhouette faces are defined by the position of the camera and an edge of the environment (anchor) as in Lemma 2.3.7. |
| $\sigma_{(E,a)}^{-1}(\{detectable\})$ | Detectable coverage of one camera (also field of view of the camera) with parameters $a = (p, o, u, \theta_u, \theta_{o \times u}) \in \mathbb{P}$ inside the environment $E \in \mathbb{E}$. If E is a polyhedral area, the coverage is a polyhedral area, as well, and has the faces defined in Definition 2.3.6: Environmental faces (\mathcal{F}_E), projection faces (\mathcal{F}_P), and opening faces (\mathcal{F}_O). Projection faces are defined by the position of the camera and an edge of the environment (anchor) as in Lemma 2.3.7. |

Notation in the Context of the Volume of the Coverage

| | |
|---|---|
| $\mathcal{N} = \begin{pmatrix} n_1 & n_2 & n_3 \end{pmatrix}^T$ | Matrix of the normals of three faces which share a common vertex of the k -reliable coverage of Equation (2.9) |
| \mathbb{K} | Set of network parameter subsets where at least one camera position is in a 1-dimensional affine subspace of an anchor defined in Equation (2.10) |

| | |
|-------------------------------------|--|
| I | Set of all vertex incidence surfaces as defined in Definition 2.3.8. The elements of this set are defined by a face of the camera coverage \mathcal{F} and a vertex \mathcal{V} and are usually denoted by $I(\mathcal{V}, \mathcal{F})$ |
| J | Set of all voxel incidence surfaces as defined in Equation (2.12) |
| D | A connected set of network parameters where the volume of the k -reliable coverage is continuously differentiable as in Theorem 2.4.3 |
| $\mathcal{B}_b \subset \mathcal{D}$ | One of $B \in \mathbb{N}$ simply connected sets of the domain $b = 1, \dots, B$ on which a stair-cased function is constant as defined in Definition 2.4.6 |
| $\beta_b \in \mathbb{R}$ | One of the quantized function values of a stair-cased function as used in Definition 2.4.6 |
| $I \subset E$ | Image plane of a camera used in Lemma 2.3.4. The points of the image plane are usually called x and the rays through such a point R . |
| $\pi : E \rightarrow I$ | Projection from the environment $E \in \mathbb{E}$ into the image plane $I \subset E$ |
| $\rho : I \rightarrow \mathbb{N}^2$ | Rasterization of the image plane $I \subset E$ into pixel |
| S | Image of an anchor of a projection or silhouette face in the rasterized image plane $S := \rho(\pi(\mathcal{F}_1)) \cap \rho(\pi(\mathcal{F}_2))$ as defined in Equation (2.13) |

Notation in the Context of Evaluation Costs

| | |
|----------------------|--|
| $c_v \in \mathbb{R}$ | Costs of measuring the volume of a collection of voxel in an occupancy grid used in the Sections 3.2.2 and <code>sec:objective:implementation:SequentialCalls</code> |
| $c_c \in \mathbb{R}$ | Costs of creating an occupancy grid of the coverage of one single camera used in the Sections 3.2.2 and <code>sec:objective:implementation:SequentialCalls</code> |
| $c_u \in \mathbb{R}$ | Costs of applying a set operation as the union or intersection of two occupancy grids used in the Sections 3.2.2 and <code>sec:objective:implementation:SequentialCalls</code> |
| $I_0 \in \mathbb{N}$ | Number of steps of the inner iteration (Lines 10–20) of the BCAUR/BCADR in Algorithm 3 used in Lemma 3.2.13 |
| $T \in \mathbb{N}$ | Number of threads used in Lemma 3.2.13 |
| $t \in \mathbb{N}$ | Number of moments in time that define the state of the dynamic objects and thus the state of their faces used in Section 2.2.1 |
| $f_s \in \mathbb{N}$ | Number of faces of static objects used in Section 2.2.1 |
| $f_d \in \mathbb{N}$ | Number of faces of dynamic objects used in Section 2.2.1 |
| $f \in \mathbb{N}$ | Number of faces in total used in Section 2.2.1 |
| $v \in \mathbb{N}$ | Number of voxels in an occupancy grid used in Section 2.2.1 |

$p \in \mathbb{N}$ Number of pixels on an image plane of a camera used in Section 2.2.1

Notation in the Context of General Optimization

$n \in \mathbb{N}$ Dimension of the domain of the objective function of Problem (3.1)

$\mathcal{D} \subset \mathbb{R}^n$ Domain of the objective function of Problem (3.1) of a size $n \in \mathbb{N}$; Points in this space are usually denoted by x and y

$f : \mathcal{D} \rightarrow \mathbb{R}$ Objective function of Problem (3.1)

G_f Graph of f : $G_f := \{(x, f(x)) \mid x \in \mathcal{D}\}$

Exclusion Area Strategy

$\bar{f} : \mathcal{D} \rightarrow \mathbb{R}$ Surrogate of the objective function as first described in section 3.1, also called response surface model. Later in the same Section, the surrogate is specified as a radial basis function interpolant in Definition 3.1.2

$L \in \mathbb{N}$ Number of elements in one search pattern described in Section 3.1.1

$K \in \mathbb{N}$ Number of candidates which have already been evaluated by the actual objective function f and are updated to the surrogate \bar{f}

$C_K \subset \mathcal{D}$ Set of K candidates (Definition 3.1.2) which have already been evaluated by the actual objective function f and are updated to the surrogate \bar{f} ; Points in this set are usually denoted by $s \in C_K$

$S_K \subset (\mathcal{D} \times \mathbb{R})$ Set of K sample pairs (Definition 3.1.2): A sample pair is a 2-tuple of a candidate $s \in C_K$ and its function value $(s, f(s))$

$\beta\Delta \in \mathbb{R}$ Measure of the exclusion area for the next candidate described in Section 3.1.1

Notation in the Context of Radial Basis Functions Interpolants (RBF)

$\bar{m} \in \mathbb{N}$ Number of polynomials in the basis of the polynomial space $\Pi_{\bar{m}}^n$

$\phi : \mathbb{R}_0^+ \rightarrow \mathbb{R}$ Kernel of the RBF of Definition 3.1.2; It defines the shape of the “peaks” and “troughs” that ensure the interpolation of the given $K \in \mathbb{N}$ sample pairs. The height of the “peaks” and “troughs” are defined by the weights $\omega \in \mathbb{R}^K$. The width of the “peaks” and “troughs” is defined by $\gamma \in \mathbb{R}^+$

$p : \mathbb{R}^n \rightarrow \mathbb{R}$ Polynomial of the RBF of Definition 3.1.2, which is a linear combination of the $\bar{m} \in \mathbb{N}$ basis polynomials of $\Pi_{\bar{m}}^n$ of degree less than or equal to $m \in \mathbb{N}$ with $n \in \mathbb{N}$ variables and their weights $\nu \in \mathbb{R}^{\bar{m}}$

$\Phi \in \mathbb{R}^K \times \mathbb{R}^K$ Matrix of the kernel evaluations of the distances between all candidates $\phi(\|s_1 - s_2\|)$, $(s_1, s_2) \in C_K^2$ for the equation system (3.7)

| | |
|--|--|
| $P \in \mathbb{R}^{K \cdot \bar{m}}$ | Matrix of the $\bar{m} \in \mathbb{N}$ basis polynomials $p \in \Pi_m^n$ of degree less than or equal to m with n variables, evaluated for all the candidates $p(s)$, $s \in C_K \subset \mathcal{D}$ for the equation system (3.7) |
| $F \in \mathbb{R}^K$ | Vector of function values of the $K \in \mathbb{N}$ candidates important for the equation system of (3.7) |
| $\chi : \mathcal{P}(G_f) \rightarrow \mathcal{P}(G_f)$ | Update of the sample pairs that are interpolated by the RBF depending on the function $f : \mathcal{D} \rightarrow \mathbb{R}$ as in Definition 3.1.9 |
| $\alpha : \mathcal{P}(G_f) \rightarrow \mathcal{P}(G_f)$ | <i>Prior</i> information of Definition 3.1.9 that can be generated by a set of sample pairs without calling the function $f : \mathcal{D} \rightarrow \mathbb{R}$ again, and can be updated to the RBF by the combined update rule. |
| $c_\chi(S)$ | Costs of an update rule χ for the previous set of sample pairs S (Definition 3.1.9): The number of objective function evaluations |
| $e_\chi(S)$ | Effect of an update rule χ for the previous set of sample pairs S (Definition 3.1.9): The number of gained sample pairs |
| $M \in \mathbb{R}^{(n+1) \cdot (n+1)}$ | Matrix of candidates $s_1, \dots, s_{(n+1)} \in \mathcal{D} \subset \mathbb{R}^n$ in the form of $M := \left(\begin{pmatrix} 1 \\ s_1 \end{pmatrix} \cdots \begin{pmatrix} 1 \\ s_{(n+1)} \end{pmatrix} \right)$ as in Definition 3.1.7 |

Notation in the Context of Block Coordinate Ascent (BCA) incorporating an RBF

| | |
|---|---|
| $M \in \mathbb{N}$ | Number of subspaces |
| $V_1 \times \dots \times V_M = \mathcal{D}$ | Orthogonal decomposition of the domain $\mathcal{D} \subset \mathbb{R}^n$ into subspaces with the dimensions $n_1, \dots, n_M \in \mathbb{N}$ and $n_1 + \dots + n_M = n$, defined in Definition 2.4.4; The elements of a subspace V_m are usually named u_m or v_m and can be transformed by the corresponding part of the unit matrix U_m into the domain. |
| $[x_1, \dots, x_M]$ | Subspace coordinates of $x \in \mathcal{D}$ defined in Definition 2.4.4 |
| $x^* \in \mathcal{D}$ | Subspace maximum: A point at which either $\operatorname{argmax}_{v \in V} f(Uv + x^*) = \mathbf{0}$ holds for one or all subspaces V and their corresponding part of the unit matrix U as defined in Definition 3.2.5 |
| $\delta_0 \in \mathbb{R}$ | Minimum bound for the absolute distance of two subsequent candidates used as a termination criterium as described in Section 3.3.3 |
| $\Delta_0 \in \mathbb{R}$ | Minimum bound for density of candidates used as a termination criterium as described in Section 3.3.3 |
| $F_0 \in \mathbb{N}$ | Maximum limit of the number of objective function calls used as a termination criterium as described in Section 3.3.3 |

Erklärung

Hiermit versichere ich an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die von mir angegebenen Quellen und Hilfsmittel verwendet habe.

Weiterhin erkläre ich, dass ich die Hilfe von gewerblichen Promotionsberatern bzw. -vermittlern oder ähnlichen Dienstleistern weder bisher in Anspruch genommen habe, noch künftig in Anspruch nehmen werde.

Zusätzlich erkläre ich hiermit, dass ich keinerlei frühere Promotionsversuche unternommen habe. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Declaration

Hereby, I affirm in lieu of oath that the presented thesis was independently authored by me and that no sources and tools other than the listed ones have been used.

Furthermore, I declare that I have never utilized and never will utilize support from any commercial PhD adviser, intermediary, or similar service provider.

Additionally, I declare that this is the first time that I present any thesis to a PhD board. This work has not been presented to any other PhD board in the same state or a similar version.

Bayreuth, January 20th, 2015

City, date

Maria L. Hänel