Multirobot Coverage Search in Three Dimensions

Christian Dornhege
Department of Computer Science, University of Freiburg, 79110 Freiburg, Germany
Alexander Kleiner
iRobot Corp., 1055 E. Colorado Blvd., Suite 340, Pasadena, CA 91106, USA
e-mail: akleiner@irobot.com
Andreas Hertle
Department of Computer Science, University of Freiburg, 79110 Freiburg, Germany
e-mail: hertle@informatik.uni-freiburg.de
Andreas Kolling
Department of Automatic Control and Systems Engineering, University of Sheffield, Sheffield S1 3JD, United Kingdom
e-mail: a.kolling@sheffield.ac.uk

Received 18 June 2014; accepted 16 November 2014

Searching for objects and observing parts of a known environment efficiently is a fundamental problem in many real-world robotic applications, e.g., household robots searching for objects, inspection robots searching for leaking pipelines, and rescue robots searching for survivors after a disaster. We consider the problem of identifying and planning sequences of sensor locations from which robot sensors can observe and cover complex three-dimensional (3D) environments while traveling only short distances. Our approach is based on sampling and ranking a large number of sensor locations for a 3D environment represented by an OctoMap. The visible area from these sensor locations induces a minimal partition of the 3D environment that we exploit for planning sequences of sensor locations with short travel times efficiently. We present multiple planning algorithms designed for single robots and for multirobot teams. These algorithms include variants that are greedy, optimal, or based on decomposing the planning problem into a set cover and traveling salesman problem. We evaluated and compared these algorithms empirically in simulation and real-world robot experiments with up to four robots. Our results demonstrate that, despite the intractability of the overall problem, computing and executing effective solutions for multirobot coverage search in real 3D environments is feasible and ready for real-world applications. © 2015 Wiley Periodicals, Inc.

1. INTRODUCTION

Coverage search is a fundamental robotics problem and is relevant for many real-world scenarios and applications. These range from household robots searching for objects, area inspection (e.g., searching for leaking pipelines or cracks in walls), up to searching for survivors in debris after a disaster in an urban search and rescue (USAR) capacity. Particularly in USAR, survivors can be enclosed within complex and heavily confined three-dimensional (3D) structures. State-of-the-art benchmarks for autonomous rescue robots, such as those proposed by the National Institute of Standards and Technology (NIST) (Jacoff, Weiss, and Messina, 2003), are simulating such situations using artificially generated rough terrain and victims hidden in crates only accessible through confined openings. Figure 1(a) depicts such a rescue arena, and Figures 1(b)–1(f) show a sequence of sensor locations from which a robot could observe the elevated area in the center of the scene.

The primary goal of the *3D coverage search* problem is to compute a sequence of sensor locations that can be visited by mobile robots on the shortest paths and from which their sensors will have seen all areas of interest in a known 3D environment. We assume that the sensor has a specific field of view with an opening angle and a detection distance that resembles that of most infrared (ir) and regular cameras. We do not make any assumptions about the 3D environment map to cover. In particular, it is not necessary that this map was acquired with the same robot or sensor that is used for the coverage search. Coverage search is similar to *coverage planning* (LaValle, 2006), where a robot is required to pass over all points in a given environment. In these applications, the footprint of the robot does not change, while in 3D coverage search the sensor footprint, e.g., the area seen by a camera, can vary dramatically with a small change in its location. The goal of coverage planning is to compute an optimal shortest motion strategy in order to cover a 2D environment with mobile robots. Solutions to this problem

Author to whom all correspondence should be addressed: Christian Dornhege, dornhege@informatik.uni-freiburg.de

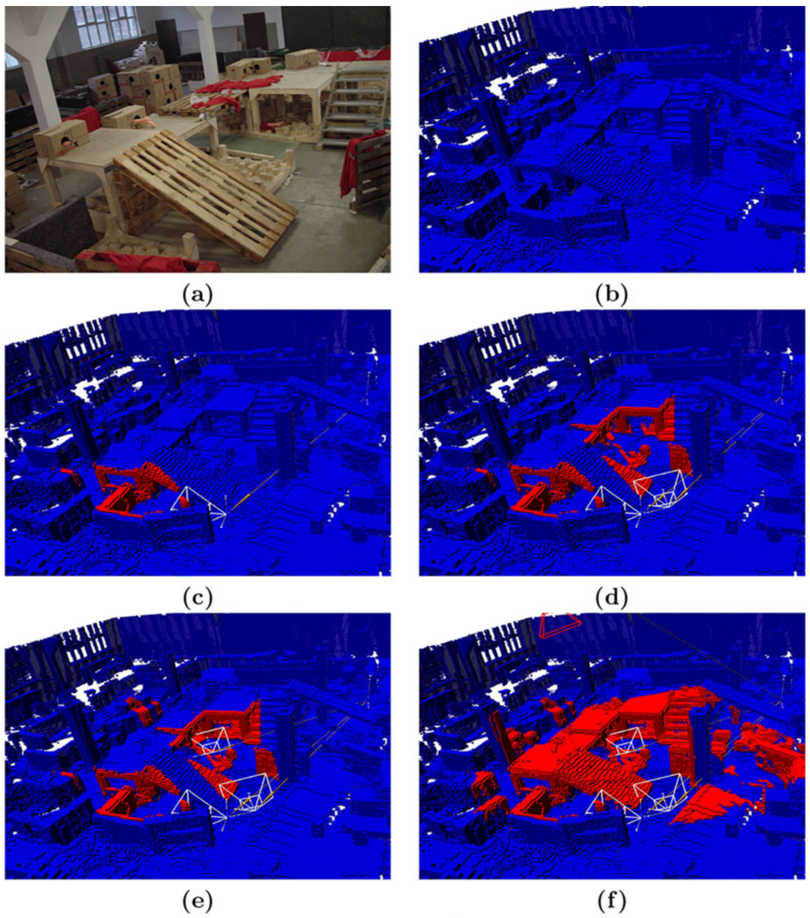


Figure 1. Motivating example: A coverage search of the NIST rescue arena, shown in (a). The map, known *a priori*, is shown in (b) with all voxels marked in blue. A mobile robot visits a set of sensor locations in sequence. The first sensor location is marked in (c) and all voxels visible to the robot's sensor are shown in red. These voxels are now considered covered. In (d) the robot visits the second location and thereby covers further voxels that are then also marked in red. Parts (e) and (f) show the result of visiting the third and fourth sensor location. The arena photo shown in (a) is courtesy of the Jacobs Robotics Group, Jacobs University, Bremen.

often employ decompositions of the free space in two dimensions (Choset, 2001). In contrast, our 3D problem relies on sampling since decompositions of 3D spaces are very costly to compute. Note that our 3D coverage problem is different from coverage problems that deploy multiple sensors to continuously monitor the same area for extended periods of time, such as in Golovin and Krause (2011). In our case, every part of the environment has to be seen at most once and does not have to be observed continuously. Other coverage-related problems often deal with unknown or partially known environments, such as in Kollar and Roy (2008) and Bourgault, Makarenko, Williams, Grocholsky, and Durrant-Whyte (2002) in two dimensions, and they attempt to improve the map or explore unknown parts of the map. In our case, the 3D environment is known and we are interested primarily in computing and visiting a set of useful sensor locations in the shortest time possible with paths that are feasible to execute for real systems. We shall review the literature relating to coverage search in slightly more detail in Section 2.

The purpose of this paper is to provide a practical solution to 3D coverage search that can be feasibly deployed. The presented solution copes with realistic sensor models and the complexity of visibility in three dimensions, as well as time constraints of robots navigating on rough terrain. Given that the configuration space for most cameras has at least six dimensions, the problem is not suited for the kinds of decomposition approaches that work well in two dimensions. Hence, we utilize a sampling approach that generates a large number of sensor locations so that the views from these locations cover the 3D environment of interest, with a bias toward locations from which the sensor can view large areas. This sampled set of sensor locations is large enough to allow the generation of many different set covers of the environment by using the sensor views from these locations. To select a good set cover and plan sequences of locations that can be visited in a short time, we use seven different variants of planning algorithms. These include algorithms that are greedy, optimal, or based on decomposing the planning problem into a set cover and traveling salesman problem. One of the key advantages of our approach is that the input size to the planning problem is significantly reduced by the sampling of sensor locations. In addition, our approach uses a *minimal partition*, introduced in detail in Section 4.2, that further facilitates the planning.

In our prior work, we presented preliminary results adopting our approach for the case of a single robot (Dornhege, Kleiner, and Kolling, 2013). As noted in Dornhege et al. (2013), already with a single robot the overall problem is intractable, which is not surprising since it contains subproblems that require solutions to the *set cover problem* (Karp, 1972) and the *traveling salesman problem* (Applegate, Bixby, Chvatal, and Cook, 2007), two well-known NP-hard problems. Our preliminary results from Dornhege et al. (2013), however, indicated that some of the algorithmic variants of our approach already perform reasonably well, both in terms of the time to compute solutions and their cost, i.e., the estimated travel time of real robots executing the solution. In this paper, we generalize and extend our approach to multiple robots with the goal to reduce the total concurrent execution time. This includes a generically applicable procedure that we use to convert single-robot solutions into multirobot solutions, as well as extensions to our greedy algorithms that consider the multirobot case directly. In addition to extensive experiments on 3D maps collected from real environments, we also present real robot experiments with four robots, and we demonstrate the applicability and feasibility of our approach under realistic conditions.

Our primary contributions are summarized as follows. In Section 3 we introduce and define our 3D multirobot coverage search problem. We then introduce the key components that make it possible to efficiently solve the multirobot 3D coverage problem for realistic scenarios. In Section 4 we present an efficient approach that utilizes hierarchical 3D maps and Monte Carlo sampling to generate a set of high utility sensor locations in order to significantly reduce the search space. The resulting views from these sensor locations are then used to generate a minimal partition, as described in Section 4.2. The minimal partition provides information about the structure of the reduced search space. This information is exploited in Section 5 for efficiently planning sequences of sensor locations with short travel times, and we present multiple planning algorithms that range from optimal to greedy and consider single and multiple robots. Greedy algorithms for single robots are presented in Section 5.1, and an optimal single robot planning formulation is presented in Section 5.2 as well as a more efficient decomposition into a set cover and traveling salesman problem. The latter enables the use of state-of-the-art solvers resulting in high-quality solutions. Section 5.4 extends the greedy approach to multiple robots, while in Section 5.5 we adapt the single-robot algorithms to the multirobot case by splitting single-robot solutions into multirobot solutions. We demonstrate feasibility and evaluate our approach with an extensive series of experiments in Section 6. These include experiments on real 3D maps (Section 6.2) that also compare our various planning algorithms. Section 6.4 presents the results of real-world experiments with up to four robots in a two-story scenario that demonstrates the applicability of the approach, verifies the results of the simulation experiments, and provides further insight about problems that are most relevant to address for efficient execution in realistic scenarios. A discussion is presented in Section 7, and our conclusions are found in Section 8.

2. RELATED WORK

There are a large number of variations of coverage problems and a vast literature on the topic. We shall only review the work that is most closely related to our 3D coverage search problem. Much of the literature on coverage in robotics is concerned with approaches for distributing a team of robots to cover an environment continuously, as one would do for environmental monitoring and surveillance applications. This is known as the *area coverage problem* (Howard, Matarić, and Sukhatme, 2002), but it is also often referred to simply as the coverage problem (Cortes, Martinez, Karatas, and Bullo, 2002). Our coverage search problem is more closely related to *coverage planning*, which is motivated by applications such as lawn mowing, automated farming, painting, vacuum cleaning, and mine sweeping. In coverage planning, the goal is to compute an optimal motion strategy in order to visit every location in the environment at least once with mobile robots (Choset, 2001; LaValle, 2006). Coverage planning for finding the optimally shortest paths is NP-hard, naturally due to the similarity to the traveling salesman problem, which also appears in our 3D coverage search problem. Solutions to coverage planning generally rely on exact or approximate cellular decompositions of the environment (Choset, 2001), which then allow planning in the resulting graph structure. The kinds of decompositions used for this approach vary from spanning trees to boustrophedon decompositions with different properties regarding practicality and optimality. Most of the work on coverage planning was concerned with 2D environments. The work in Renzaglia, Doitsidis, Martinelli, and Kosmatopoulos (2011) considers 2.5D environments and optimization techniques are applied to compute 3D paths for unmanned aerial vehicles. The environment, however, is unknown and the emphasis lies on the application of an optimization technique in order to maximize area coverage. To the best of our knowledge, most of the prior work on coverage planning, i.e., to plan global and minimal distance paths, was restricted to 2D environments. It is a considerable challenge to adapt solutions that are based on decompositions of an environment to 3D environments. Perhaps the best illustration of this appears in Lazebnik (2001). Therein, the 2D visibility-based pursuit-evasion problem (Sachs, Rajko, and LaValle, 2004) is generalized to three dimensions and the resulting complications are staggering. Needless to say, not much progress has been made on this problem since then. Other than the extension to three dimensions, there is another key distinction between our problem and coverage planning. The footprint of the robot in coverage planning is static, i.e., the lawnmower does not change its shape, while due to the camera model we use, what is visible in the field of view varies wildly. Hence the distinction between *coverage planning* and *coverage search*. As a consequence of this sensor model, we have to consider 3D visibility. Note that 2.5D visibility, as in Renzaglia et al. (2011), is still far simpler than 3D visibility, and no prior work that considers 3D visibility for coverage planning is known to the authors. Here we partly rely on our previous work, in which we extended the well-known problem of frontier-based exploration on 2D grid maps (Yamauchi, 1997) to 3D environments (Dornhege and Kleiner, 2011) by computing so called frontier voids. We will describe this in more detail in later sections. Englot and Hover (2013) address coverage planning of a ship hull, while also taking into account visibility by ray casting. Candidate views are generated by sampling locally around the geometric primitives to observe given as an input and collected into a roadmap. The only algorithm they consider to compute a coverage plan is the subsequent application of set cover and traveling salesman. They apply an iterative traveling salesman variant that enables the lazy evaluation of paths between view poses, as the time to compute feasible paths is comparably large in this setting. This results in a less generic formulation than our work, but it allows them to derive theoretical completeness results and optimized trajectories for the specific scenario.

Coverage search also relates to research that is concerned with the computation of views and visibility, such as next best view approaches, the art gallery problem, and pursuit-evasion problems. Traditional next best view (NBV) algorithms compute a sequence of viewpoints until an entire scene or the surface of an object has been observed by a sensor (Banta et al., 1995; Gonzalez-Banos, Mao, Latombe, Murali, and Efrat, 2000). These methods are, however, not suitable for coverage search on mobile robots since they ignore the costs for changing between different sensor poses (Gonzalez-Banos et al., 2000). Our computation of views is based on improvements of our previous work on *frontier-void* based exploration in three dimensions (Dornhege and Kleiner, 2011), which dealt specifically with searching for victims. In Dornhege and Kleiner (2011) we focused on finding cells at the exploration frontier having good views into unknown parts of the environment, so called voids. In this paper, we not only efficiently compute next best views in three dimensions but a data structure that contains many views and has many subsets of views that can all cover the entire environment. Computing visibility or views is also addressed by the art gallery problem (Shermer, 1992). There the problem is to find an optimal placement of guards on a polygonal representation of 2D environments so that the entire environment is continuously observed by the guards. The emphasis is on the placement of guards with relation to the complex geometric features of the environment, and the problem is known to be NP-hard. Again, the computation of views focuses on 2D environments. Pursuit-evasion problems (Chung, Hollinger, and Isler, 2011) relax this requirement since the goal is not to keep a static coverage of the environment but to search for moving targets. Toward that end, the environment can be observed in a dynamic fashion by placing and removing guards over time. This also requires the computation of the robot's field of view, which often induces a decomposition of the environment, as in Sachs et al. (2004). Again, much of this work focuses on 2D environments. Approaches for 2.5D environments appear in Kleiner, Kolling, Lewis, and Sycara (2013) and Kolling, Kleiner, Lewis, and Sycara (2010), and they have been tested within real environments.

While area coverage, pursuit-evasion, and the art gallery problem are naturally multirobot problems, our coverage search problem can be applied to a single robot or multiple robots, just as coverage planning. Multirobot coverage in two dimensions has been considered in Agmon, Hazon, and Kaminka (2008) and Kong, Peng, and Rekleitis (2006). In Rekleitis, Lee-Shue, New, and Choset (2004), a single robot approach for coverage planning is extended to multiple robots. It uses a Boustrophedon decomposition developed for the single-robot case. The environment is unknown, and robots have line of sight communication that precludes a multiple traveling salesman approach (Bektas, 2006). Coordination is achieved by having two explorers detecting critical points in the environment based on visibility. The remaining teams split and cover the cell, whose existence is implied by the behavior of the explorers. Other multirobot approaches for coverage planning in two dimensions are also discussed in Choset (2001).

Coverage search and exploration are also closely related, with the obvious difference that the environment to be explored is unknown. The literature for robot exploration is also vast, particularly in two dimensions, while 3D exploration methods are starting to appear. For example, Surmann, Nüchter, and Hertzberg (2003) propose a method for planning the next scan pose of a robot for digitalizing 3D environments. They compute a polygon representation from 3D range scans with detected lines (obstacles) and unseen lines (free space connecting detected lines). From this polygon, potential next-best-view locations are sampled and weighted according to the information gain computed from the number of polygon intersections with a virtual laser scan simulated by ray tracing. The next position approached by the robot is selected according to the location with maximal distance. Their approach has been extended from a 2D representation toward 2.5D elevation maps (Joho, Stachniss, Pfaff, and Burgard, 2007). Lozano Albalate, Devy, Miguel, and Marti (2002) also compute next best views for 3D exploration. Their approach not only weighs view utility by unseen volumes, but it also considers sufficient overlap with known volumes to support matching new views into a consistent 3D representation.

3. PROBLEM DEFINITION

In this section, a multirobot coverage search is formally described. We first introduce the model of the searchers, their sensors, and the environment followed by the definition of the coverage search problem.

We consider homogeneous mobile robot platforms, the searchers, each equipped with a 3D sensor in a bounded 3D environment $\mathcal{E} \subset \mathbb{R}^3$. The 3D sensor generates a view at each sensing cycle. A view is a set of n 3D points $\{\mathbf{p}_1, \mathbf{p}_2, \ldots, \mathbf{p}_n\}$ with $\mathbf{p}_i = (x_i, y_i, z_i)^T$ representing detected obstacles within the sensor's field of view. We associate a view with the sensor state that generates it, i.e., a sensor state $\mathbf{x} \in X \cong \mathbb{R}^3 \times \mathbb{RP}^3$ [see LaValle (2006) for details], which can also be written as a 6D pose $(x, y, z, \phi, \theta, \psi)^T$. Here $(x, y, z)^T$ denotes the translational part (also referred to as position) and $(\phi, \theta, \psi)^T$ is the rotational part in Euler angles (also referred to as orientation). The possible sensor states, and hence the resulting views that can be obtained, depend on the attainable collision-free configurations of the searcher, $q \in \mathcal{C}_{\text{free}} \subset \mathcal{C}$. We make no assumptions regarding $\mathcal{C}$ and $\mathcal{C}_{\text{free}}$ other than that we have a function $IK : X \to \{0, 1\}$ with $IK(\mathbf{x}) = 1$ if there is a valid path in $\mathcal{C}_{\text{free}}$ for the searcher that puts the sensor into state $\mathbf{x}$ and 0 otherwise.[1] This allows us to define the set of all reachable sensor states $X_{\text{reach}} := \{\mathbf{x} \in X | IK(\mathbf{x}) = 1\}$.[2] Note that we do not consider collisions between robots when determining IK and X_{reach}, so that these are static. In our experiments, we avoid collisions with a simple scheme that requires robots to wait when a collision with another robot is predicted. Depending on the robot type and their knowledge about the position of other robots, one may choose to use another deconfliction scheme or a multirobot path-planning approach.

We furthermore assume the existence of a function $\text{cost} : \mathcal{C}_{\text{free}} \times X_{\text{reach}} \to \mathbb{R}^+$, which returns the time to move a robot from one configuration to another, place the sensor in a desired state, and record a view. We will refer to this also as travel time. Strictly speaking, a cost function could also incorporate additional criteria, such as risk of failure, detection by hostiles or other undesired consequences that would lead to increased costs. For our purposes, however, we equate cost with travel time. Note that for most applications, we can conflate $\mathcal{C}_{\text{free}}$ and X_{reach} by mapping exactly one searcher configuration onto each sensor state in X_{reach}. This effectively ignores additional degrees of freedom of the searcher, and with a slight abuse of notation we can then write $\text{cost}(\mathbf{x}_i, \mathbf{x}_j)$. This is now simply the time of moving from one sensor state to another. We shall apply this simplification to the remaining sections as it simplifies the presentation without directly impacting the applicability of our solutions. For applications in which the additional degrees of freedom can be exploited, e.g., humanoid searchers with cameras, one would have to consider these separately.

Finally, the goal of coverage search is to cover every point in a given search set $\mathcal{S} \subseteq \mathcal{E}$. The search set might contain only a small part or all of $\mathcal{E}$, depending on what we are interested in covering. For every sensor state, let the detection set $D(\mathbf{x}) \subset \mathcal{S}$ be the set of points in $\mathcal{S}$ visible from $\mathbf{x} \in X_{\text{reach}}$. Note that there is a subtle formal difference between $D(\mathbf{x})$ and a view at $\mathbf{x}$, i.e., $D(\mathbf{x})$ is a 3D volume and a view is a discrete set of points in a 3D volume. In addition, $D(\mathbf{x})$ is restricted to the search set $\mathcal{S}$ while a view contains points from all of $\mathcal{E}$. Yet, in colloquial terms, we can think of a detection set as a view. We use the term "detection set" primarily in a formal context.

The multirobot coverage problem for N robots is to find and visit a sequence of sensor states for each robot $n \in \{1, \ldots, N\}$ with length $m(n)$, written as $\mathbf{x}^n_1, \mathbf{x}^n_2, \mathbf{x}^n_3, \ldots, \mathbf{x}^n_{m(n)}$, so that the entire search space $\mathcal{S}$ has been seen and covered, i.e., $\bigcup_{n=1}^{N} \bigcup_{i=1}^{m(n)} D(\mathbf{x}^n_i) = \mathcal{S}$. In addition, the overall execution time needed to visit all sensor states given by

$$\text{cost}_{\max} = \max_{n \in \{1,\ldots,N\}} \sum_{i=1}^{m(n)-1} \text{cost}(\mathbf{x}^n_i, \mathbf{x}^n_{i+1})$$

has to be minimized. We refer to *travel time* when we consider the time a single robot takes to travel between locations, and *execution time* when referring to the maximum combined travel time across all robots, i.e., the time to execute the entire sequence.

4. SAMPLING AND PARTITIONING VIEWS

4.1. Sampling High Utility Views

We now describe how to find sensor states from X_{reach} that have large views by computing a utility function $util : \mathcal{E} \to \mathbb{R}^+$ that identifies good 3D poses in $\mathcal{E}$, ignoring the orientation for now. As we have already shown experimentally, in the context of pursuit-evasion problems (Kleiner et al., 2013), an efficient sampling-based heuristic can significantly

[1]To make this definition complete, we further make the usual assumption that either a starting configuration q_0 for the searcher is given or that C_{free} is connected.
[2]Reachable states can be precomputed for efficient access during the search using capability maps (Zacharias, Borst, and Hirzinger, 2007).

decrease the number of states that have to be considered. In this spirit, we will compute *util* via sampling and then later use it to identify 3D poses from which a large part of $\mathcal{S}$ can be seen. These high-utility poses in $\mathcal{E}$ are then turned into sensor states, which will serve as a basis for the coverage search methods described in Section 5.

The representation of $\mathcal{E}$ is given in the form of an efficient hierarchical 3D grid structure, known as *OctoMap* (Hornung, Wurm, Bennewitz, Stachniss, and Burgard, 2013). Therein our 3D search region $\mathcal{S}$ is tessellated into equally sized cubes. This effectively discretizes $\mathcal{E}$ and $\mathcal{S}$, and we shall treat them as discrete sets from hereon. The minimum size of the cubes is typically chosen relative to the size of the target that one searches for, i.e., the size of the cubes should generally be smaller than the target. The implementation for OctoMap is based on an octree that represents occupied areas in a hierarchical manner. Free space as well as unknown areas are implicitly encoded in the map.

We construct *util* in two steps, shown in detail in Algorithm 1 and briefly described below. First, for every $s \in \mathcal{S}$ we sample $k_{\max}$ vectors that start at s and go toward a random position in $pos(X_{\text{reach}})$, sampled using *getRandom*(.). Here *pos*(.) returns the position of a state, simply ignoring its orientation, or the set of positions for a set of states, respectively. These vectors are collected in $\mathcal{V}$. Second, for each vector $\langle s, dir\rangle \in \mathcal{V}$ we compute by using the ray tracing function *getGridCells*(s, dir, s_r) the set of grid cells $\mathcal{GC}$ that are visible from s in direction dir up to the sensor range limit s_r. Ray tracing is performed efficiently on the OctoMap. Then, for each cell in $\mathcal{GC}$ that corresponds to a reachable sensor state, the utility value is incremented by 1.

Algorithm 1 Construct *util*

```
procedure FINDGOODVIEWS($\mathcal{S}$)
    $\mathcal{V} \leftarrow \emptyset$
    // Sample random vectors from $\mathcal{S}$ into $pos(X_{reach})$
    for all $s \in \mathcal{S}$ do
        $k \leftarrow k_{max}$
        while $k \neq 0$ do
            $\mathbf{x} \leftarrow getRandom(X_{reach})$
            $dir = normalize(pos(\mathbf{x}) - s)$
            $\mathcal{V} \leftarrow \mathcal{V} \cup \langle s, dir\rangle$
            $k \leftarrow k - 1$
        end while
    end for
    // Accumulate utilities in $\mathcal{E}$
    for all $v = \langle s, dir\rangle \in \mathcal{V}$ do
        $\mathcal{GC} \leftarrow getGridCells(s, dir, s_r)$
        for all $gc \in \mathcal{GC} \cap pos(X_{reach})$ do
            $util(gc) \leftarrow util(gc) + 1$
        end for
    end for
end procedure
```

We now obtain our set of sampled sensor states $\tilde{X}$, from which large parts of $\mathcal{S}$ are visible, as follows. First, we sample grid cells that correspond to points $(x, y, z)^T \in \mathcal{E}$ with a positive and large *util* value. Note that by construction these points are such that $(x, y, z)^T \in \text{pos}(X_{\text{reach}})$, i.e., they correspond to the poses of reachable sensor states. For each of these points, we sample one orientation $(\phi, \theta, \psi)^T$ so that we obtain a full sensor state $\mathbf{x} = (x, y, z, \phi, \theta, \psi) \in \mathbf{X}$. Note that for some robots, such as ground robots, the set of reachable sensor poses $\text{pos}(X_{\text{reach}})$ can be much smaller than $\mathcal{E}$, and our method samples only from this much smaller space. Now that we have sampled a sensor state $\mathbf{x}$, we compute its actual utility $U(\mathbf{x}) := |D(\mathbf{x})|$ by ray-tracing the sensor's field of view and counting all visible octree voxels. If $U(\mathbf{x}) \geq \epsilon$, for some given ϵ, then we add $\mathbf{x}$ to $\tilde{X}$. Once the number of poses in $\tilde{X}$ reaches a predefined limit N_{sensor} that is given as an input, we stop adding to $\tilde{X}$ and terminate the sampling.

The method described above for sampling sensor states with high utility is rather generic and can easily be modified in order to achieve additional objectives or bias the sampling. For example, to formally guarantee complete coverage of $\mathcal{S}$ with the sensor states from $\tilde{X}$, one could continue to sample poses with nonzero *util* values and incrementally add more views until $\mathcal{S}$ is covered, as done in the work by Kleiner et al. (2013). However, this requires that every part of $\mathcal{S}$ can be seen by some $\mathbf{x} \in X_{\text{reach}}$—a property required for the problem to be solvable that unfortunately can be violated in many practical applications. There are no assumptions that the environment data and maps were collected with the robot that is used for the search and thus the environment $\mathcal{E}$ can cover arbitrary nonreachable space. In short, from our practical perspective it is easier to ignore completeness and implement a best effort that is more robust and allows the user to increase N_{sensor} to increase coverage and determine whether it is sufficient for the application. Other straightforward extensions may include the use of certain sampling functions that consider movement cost or risk of failure or destruction at a certain pose. One may also sample multiple orientations for a given 3D pose $(x, y, z)^T$ or compute the orientation with the highest actual utility.

The primary feature of our sampling approach is that we provide an efficient initial estimate of the utility of sensor states with the *util* function that can be used in a number of ways leading to small sets of useful sensor states $\tilde{X}$. In this paper, we only present the most straightforward sampling that considers sampling the highest *util* poses with only one orientation and an actual utility of at least ϵ until we reach N_{sensor} sensor states for $\tilde{X}$. Investigating the wide range of possible variations to this sampling method can be a fruitful area for further work. We shall now proceed to the next section, which shows how to exploit the set $\tilde{X}$ for planning purposes by first constructing a partition of the detection sets corresponding to the sensor states.

4.2. Partition Induced by Views

Based on the sampled $\tilde{X} \subset X_{\text{reach}}$, which represents a significantly smaller number of high utility sensor states, we now seek to determine an even further smaller set that gives us sequences of sensor states $\{\mathbf{x}_1^n, \ldots, \mathbf{x}_{m(n)}^n\} \subset \tilde{X}$ whose detection sets cover all of $\mathcal{S}$, i.e., $\bigcup_{n=1}^{N} \bigcup_{i=1}^{m(n)} D(\mathbf{x}_i^n) = \mathcal{S}$. Rather than computing $\bigcup_{i=1}^{m(n)} D(\mathbf{x}_i^n)$ for different sequences, which is computationally expensive, we compute a *minimal partition* of the search set $\mathcal{S}$ that is induced by the detection sets for a given set of sensor states Q (in our case $\tilde{X} = Q$). Figure 2 illustrates the reduction achieved by such a partition. Without partitioning, one has to merge detection sets using individual voxels. A *minimal partition* collects all voxels that are contained in exactly the same detection sets into one partition part. We define the *reduction factor* achieved by a partition as the ratio of the number of individual voxels to the number of partition parts, i.e., for the example in Figure 2 it is $\frac{164}{6} \frac{\text{voxels}}{\text{partition parts}} \approx 27.3$. A minimal partition of $\mathcal{S}$ given Q is defined as follows:

Definition 1. *Minimal partition given a set of sensor states*

Given any $Q \subseteq X_{\text{reach}}$, then $P(\mathcal{S}|Q)$ is a partition of $\mathcal{S}$ minimal for Q if the following conditions are satisfied:

1. $\emptyset \notin P(\mathcal{S}|Q)$
2. $\bigcup_{A \in P(\mathcal{S}|Q)} A = \mathcal{S}$
3. $\forall A, B \in P(\mathcal{S}|Q) : A \neq B \Rightarrow A \cap B = \emptyset$
4. $\forall \mathbf{x} \in Q, \forall A \in P(\mathcal{S}|Q) : A \cap D(\mathbf{x}) = A \vee A \cap D(\mathbf{x}) = \emptyset$
5. $\forall A, B \in P(\mathcal{S}|Q) : A \neq B \Rightarrow \exists \mathbf{x} \in Q : A \cap D(\mathbf{x}) \neq \emptyset \wedge B \cap D(\mathbf{x}) = \emptyset$

Conditions 1. to 3. state that $P(\mathcal{S}|Q)$ is a partition. Condition 4. states that every part of $P(\mathcal{S}|Q)$ is either entirely in a detection set or it does not intersect the detection set. Condition 5. states that $P(\mathcal{S}|Q)$ is minimal. With slight abuse of notation, we will write $P(\mathbf{x}) \subset P(\mathcal{S}|Q)$ for all parts of $P(\mathcal{S}|Q)$ with $P(\mathbf{x}) \subset D(\mathbf{x})$.[3] Minimal partitions can be computed iteratively, as shown in Alg. 2, when Q is finite. In colloquial terms, one can think of $P(\mathcal{S}|Q)$ as the Venn diagram of the search set and all detection sets for states from Q, i.e., of $\mathcal{S}, D(\mathbf{x}_1), \ldots, D(\mathbf{x}_{|Q|})$. As a shorthand, we shall also write $P(Q) = P(\mathcal{S}|Q)$ since $\mathcal{S}$ is given and fixed.

Algorithm 2 initializes $P(Q)$ to the trivial partition $\{S\}$. For each $\mathbf{x} \in Q$, we update $P(Q)$ by splitting all parts in $P(Q)$ that violate condition 4. Note that we only test against

[3]Notice that $\left(\mathcal{S} \setminus \bigcup_{\mathbf{x} \in Q} D(\mathbf{x})\right) \in P(\mathcal{S}|Q)$ is a part of the partition for all Q that does not contain enough configurations to cover $\mathcal{S}$. From the sensors' perspective, this part is undetectable from the states in Q.

Algorithm 2 Minimal Partition for Q

```
1:  $P(Q) \leftarrow \{\mathcal{S}\}$
2:  $Views(\mathcal{S}) \leftarrow Q$
3:  for all $\mathbf{x} \in Q$ do
4:      $P(\mathbf{x}) \leftarrow P(Q)$
5:  end for
6:  for all $\mathbf{x} \in Q$ do
7:      for all $A \in P(\mathbf{x})$ do
8:          $A_{in} \leftarrow A \cap D(\mathbf{x})$
9:          $A_{out} \leftarrow A \cap (\mathcal{S} \setminus D(\mathbf{x}))$
10:         if $A_{in} = \emptyset \vee A_{out} = \emptyset$ then
11:             continue // Condition 4. holds.
12:         end if
13:         $P(Q) \leftarrow P(Q) \setminus \{A\} \cup \{A_{in}, A_{out}\}$
14:         $P(\mathbf{x}) \leftarrow P(\mathbf{x}) \setminus \{A\} \cup \{A_{in}\}$
15:         for all $\mathbf{x}' \in Views(A) \setminus \{\mathbf{x}\}$ do
16:             $P(\mathbf{x}') \leftarrow P(\mathbf{x}') \setminus \{A\} \cup \{A_{in}, A_{out}\}$
17:         end for
18:         $Views(A_{in}) \leftarrow Views(A)$
19:         $Views(A_{out}) \leftarrow Views(A) \setminus \{\mathbf{x}\}$
20:     end for
21: end for
22: return $P(Q)$
```

$P(\mathbf{x})$ instead of all parts in $P(Q)$ as required by condition 4. $P(\mathbf{x}) \subseteq P(Q)$ is maintained in addition to $P(Q)$ and only contains those parts that intersect with the detection set $D(\mathbf{x})$. Thus often $P(\mathbf{x}) \subset P(Q)$. We also maintain and update the inverse mapping $Views(A)$ for each part A in $P(Q)$ to efficiently update $P(\mathbf{x})$.

5. MULTIROBOT COVERAGE SEARCH

In this section, we are concerned with the problem of selecting a set of states from $\tilde{X}$, assigning these to individual robots, and computing the shortest paths for each robot, with the goal of covering the entire environment in the least amount of time. Note that all the above steps have dependencies that complicate the problem, e.g., the quality of the shortest paths obviously depends on the assignment, shortest paths may lead to collisions in additions to obstructions of views by other robots, and so on. To be able to find feasible solutions in a reasonable amount of time, we treat individual robot paths as if they were independent from each other. This means that we will not be dealing with multirobot collision avoidance, view obstructions, or synchronizing parallel actions for multiple robots. These problems can be dealt with for each specific application, and they have varying degrees of impact, depending on the scenario. Hence, we believe these issues should be addressed in a particular implementation and its execution. Unless a particularly hard scenario is constructed, the interactions between the coverage search and the above issues should be minimal, especially considering that the searchers should naturally spread to different parts of the environment.

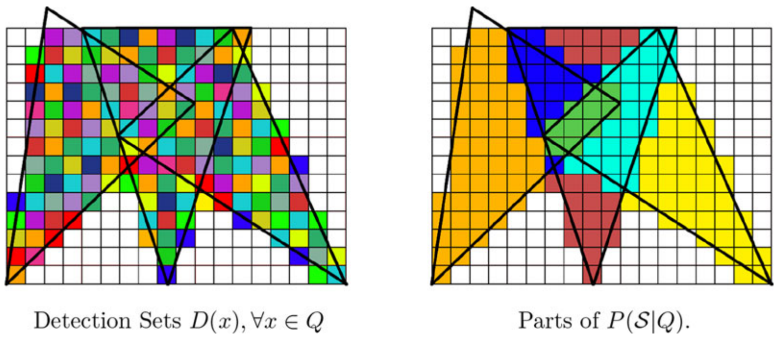

Detection Sets $D(x), \forall x \in Q$
Parts of $P(\mathcal{S}|Q)$.

Figure 2. This figure illustrates the effect of a minimal partition. The left shows the detection sets for three states, i.e., $|Q| = 3$, as black triangles. The search set, $\mathcal{S}$, is the entire grid. The individual voxels that are part of the detection sets were determined by ray tracing and are shown in random colors on the left side, while white voxels are not part of any detection set. The right shows the minimal partition $P(\mathcal{S}|Q)$ with several parts, each represented with a color. Let $\mathbf{x}$ be the state for the leftmost view, then $P(\mathbf{x})$ contains three parts, the set of blue, of green, and of orange voxels.

We now introduce the two principal ideas we use for solving the multirobot planning problem. The first is to create a high-quality single-robot coverage plan and then divide this plan into smaller segments that are assigned to multiple robots. The second is to adapt planning procedures directly to the multirobot problem, particularly for the greedy planning approaches. In the following, we will thus first describe our algorithms for single-robot coverage plans. These are based on the best performing algorithms, which have also been presented in Dornhege et al. (2013). We will then adapt these algorithms to the multirobot case.

5.1. Single-robot Greedy Solutions

We utilize $P(\tilde{X})$, as well as the corresponding mappings $P(\mathbf{x})$, to construct a sequence $\{\mathbf{x}_1, \ldots, \mathbf{x}_m\} \subseteq \tilde{X}$ that covers $\mathcal{S}$ and has a short execution time. Since we now consider the sequence of sensor states, each with an associated detection set and view, we define a new sequential utility function U_i that reduces the original utility U by the volume in $\mathcal{S}$ that has been seen previously in the sequence

$$U_i(\mathbf{x}) := \left| D(\mathbf{x}) \setminus \bigcup_{j<i} D(\mathbf{x}_j) \right|.$$

All greedy strategies compute the coverage sequence by repeatedly selecting a state $\mathbf{x} \in \tilde{X}$ until all parts in $P(\tilde{X})$ are covered. The greedy algorithms differ by how the next $\mathbf{x}$ is selected. For all algorithms, let UC be the uncovered parts initialized as $UC \leftarrow P(Q)$. For every $i = 1, \ldots, m$ we select a new $\mathbf{x}_i$ and update $UC \leftarrow UC \setminus P(\mathbf{x}_i)$. The algorithms terminate when $UC = \emptyset$. We shall now describe two greedy algorithms.

Simple Greedy The simplest greedy strategy selects a $\mathbf{x}_i$ that minimizes the travel time from the last view, i.e., $\text{cost}(\mathbf{x}_{i-1}, \mathbf{x}_i)$. The first view is chosen to be the one with maximum utility U. We denote this variant as *Simple-Greedy*.

Greedy Next Best View Our second greedy variant also considers the utility of views in addition to the travel time, i.e., it chooses a $\mathbf{x}_i$ that maximizes $U_i(\mathbf{x}_i)/\text{cost}(\mathbf{x}_{i-1}, \mathbf{x}_i)$. The idea is to prefer high utility views that are also easily reachable. For this we choose the ratio of utility and travel time to quantify the tradeoff between the two. Again, the first view is chosen to be the one with maximum utility U. We denote this variant as *Greedy Next Best View (Greedy-NBV)*.

5.2. Optimal Planning Formulation

Greedy algorithms provide simple and reasonably fast solutions that are also easily modified to consider additional criteria for specific application scenarios. These solutions, however, are usually not optimal. Hence, we also formulate the problem of finding a coverage sequence for a set of states from $\tilde{X}$ as a classical planning problem that can be solved optimally. As finding optimal solutions to this problem is often infeasible, as discussed in further detail in Section 6.3, we also present a suboptimal decomposition of the problem by solving a set cover and subsequent traveling salesman problem. We will now present the problem formulation as a planning problem, and afterward we will show how this formulation can be easily adapted to solve the decomposition.

We model our planning problem in the commonly used Planning Domain Definition Language (PDDL) (Fox and Long, 2003). The definition consists of the objects involved in the problem, logical predicates over the objects describing the state, and actions with preconditions and effects that determine how the action changes the state. In addition, the initial state as well as a goal formula must be given. Such a task definition models a search problem in a state space implicitly defined by the predicates. Action preconditions are logical formulas over those predicates that must be true for an action to be applicable in a state. Action effects assign new values to predicates. A problem is solved by finding a sequence of actions from the initial state to any state fulfilling the goal formula.

In our domain, there are two types of objects:
`(:types view view_part)`
An object of type `view` is added for each sensor state in $\tilde{X}$, and a `view_part` is defined for each part in the partition. Next, the planning state is given by the following logical predicates:

(searched ?$\mathbf{x}_i$ - view)

describes that a view $\mathbf{x}_i$ has already been visited and

(covered ?p_j - view_part)

states that part p_j has been covered in a state. `searched` and `covered` are initially set to `false` for all views and parts, respectively. The current view location of the robot is given by

`(at-view)- view`

For each view $\mathbf{x}_i$ and each part p_j, the predicate

(view-covers ?$\mathbf{x}_i$ - view ?p_j - view_part)

is set to `true` in the initial state, iff p_j is in $P(\mathbf{x}_i)$.

Only a single action is needed:

```
(:action search
  :parameters (?v - view)
  :duration (= ?duration [costSearch ?v])
  :precondition
    (not (searched ?v))
  :effect
  (and
    (searched ?v)
    (assign (at-view) ?v)
    (forall (?_vp - view_part)
      (when (view-covers ?v ?_vp)
        (covered ?_vp)))))
```

The precondition prohibits the planner from choosing the same view twice. Accordingly, `searched` is set in the effect. We also assign `at-view` to the view reached by the search action. `:duration` defines the action's cost. The term `[costSearch ?v]` states that the cost of the action is determined by an external function that is integrated via a modular interface (Wurm et al., 2010). Thus, whenever the planner requests the cost of this action, the module calls the *cost* function defined in Section 3. A detailed description of this interface is available in our previous work (Dornhege et al., 2009). The `forall` statement defines a conditional effect that sets `covered` to true for all view parts `when` the view part is in $P(\mathbf{x})$.

Finally, we specify the following as the goal formula:

`(forall (?vp - view_part) (covered ?vp))`

This requires each part to be covered and thus guarantees that any plan found by a planner actually provides a coverage plan. As we use the actual cost function to define action costs, a shortest plan found by the planner also constitutes a minimum execution time coverage sequence, i.e., an optimal solution to our coverage problem with the available views restricted to views from sensor states in $\tilde{X}$. We use a variant of the planner Temporal Fast Downward (TFD) (Dornhege et al., 2009) to solve all planning tasks in this paper. We denote this variant as *Complete Planning*.

5.3. Decomposition into Set Cover and Traveling Salesman Problem

While the previous planning formulation can yield optimal solutions, the search space is still rather large and the optimal solutions may not be found in a reasonable amount of time for large problem instances. Hence, in this section we simplify the problem by decomposing it into a set cover and a traveling salesman problem. More precisely, we first find a minimal set of views that covers all parts of the minimal partition, i.e., the classical *set cover* problem. This gives us a minimum cardinality subset $Q_C \subseteq \tilde{X}$, so that all parts of $P(\tilde{X})$ are covered, i.e., $\bigcup_{\mathbf{x}_j \in Q_C} P(\mathbf{x}_j) = \bigcup_{\mathbf{x}_i \in \tilde{X}} P(\mathbf{x}_i)$.

We can use the minimal partition to reduce the input size to the set cover problem by ignoring views covering unique parts of the search set that are not covered by any other view. We call these views *necessary*, since they have to be part of any cover, and we only determine the minimum set cover for the remaining views. The set cover problem is solved by a simple reformulation of the complete planning problem using the same planner. More precisely, action costs from the above definition are set to 1, so that the cost of a plan is identical with the number of views. These problems are solved quite fast (within seconds in all our experiments) as they contain a considerably smaller set of views, and permutations do not need to be considered.

Given the minimum cardinality subset of views that covers the search space, it only remains to find an optimal execution time sequence visiting all views. This is a *Traveling Salesman Problem* (TSP) without the requirement to return to the first location. We already have a PDDL formulation for the complete problem, and we can easily apply this formulation to the Traveling Salesman Problem by changing the goal formula to

`(forall (?v - view) (searched ?v))`

This requires all input views, which are now only the views that are part of the minimal cardinality cover, to be visited, and thus an optimal cost plan to this problem results in a minimum execution time path through all views. The coverage information can be safely ignored, as that is guaranteed by the set cover.

There exist efficient solvers specifically designed for the Traveling Salesman Problem, and thus we investigate the application of the LKH solver (Helsgaun, 2000), an efficient implementation of the Lin-Kernighan heuristic, to solve the TSP. When we use the TFD planner for solving the TSP in the decomposed formulation, we denote this variant as *Set Cover/TSP (TFD)*. When using the LKH solver we shall call the variant *Set Cover/TSP (LKH)*. TFD is always used to solve the set cover problem.

To summarize, the decomposition of the complete planning formulation into a set cover and a TSP problem is clearly suboptimal. Yet, it still requires solutions for two NP-hard problems. But from a practical perspective, this decomposition allows the application of advanced commercial solvers that have been developed specifically for these problems. In the experimental sections, we will briefly investigate the tradeoff between the time to compute solutions and the quality of these solutions between the optimal complete approach and the decomposition approach.

5.4. Multirobot Greedy Solutions

In this section, we adapt the single-robot greedy algorithms directly to the multirobot case. As before, we utilize $P(\tilde{X})$, as well as the corresponding mappings $P(\mathbf{x})$, to construct a sequence $\{\mathbf{x}_1^n, \dots, \mathbf{x}_{m(n)}^n\} \subseteq \tilde{X}$ for each robot $n \in \{1, \dots, N\}$ with length $m(n)$. The multirobot greedy algorithms compute the coverage sequences by repeatedly selecting a specific robot n and a view $\mathbf{x}^n \in \tilde{X}$ until all parts in $P(\tilde{X})$ are covered. Similar to the single-robot case, we define a sequential utility function U_k that reduces the original utility U by the volume in $\mathcal{S}$ that has been seen previously in any sequence, i.e., by any robot. Here k is the kth step in the greedy procedure, when the kth view is assigned. Let $m_k(n)$ be the sequence length for robot n in the kth step. Then the utility to choose the view from state $\mathbf{x}$ in the kth step is

$$U_k(\mathbf{x}) := \left| D(\mathbf{x}) \setminus \left(\bigcup_{n=1}^{N} \bigcup_{j \leq m_{k-1}(n)} D(\mathbf{x}_j^n) \right) \right|.$$

Note that this measure is independent of the robot. Now, using the expected utility and the time to reach the state for a view, every robot is choosing its next preferred view at step k exactly as in the single-robot case, i.e., the preferred view is selected either using the *Greedy-NBV* or the *Simple-Greedy* equations from Section 5.1, leading to the *Multi-Simple-Greedy* and *Multi-Greedy-NBV* variants.

Let $\hat{\mathbf{x}}_k^n$ be the state chosen by robot n for step k. We now select the state from the robot that leads to the minimum increase in overall execution time, i.e., we greedily select the robot with the shortest overall path to its preferred state. More precisely, let this robot be

$$n^* := \operatorname*{arg\,min}_{n \in \{1,\dots,N\}} \operatorname{cost}\left(\mathbf{x}_{m_{k-1}(n)}^n, \hat{\mathbf{x}}_k^n\right) + \sum_{i=1}^{m_{k-1}(n)-1} \operatorname{cost}\left(\mathbf{x}_i^n, \mathbf{x}_{i+1}^n\right).$$

The view $\hat{\mathbf{x}}_k^{n^*}$ is then appended to the sequence of views for robot n^*, and we continue with the next step by incrementing k. The procedure continues until all parts in $P(\tilde{X})$ are covered. Let UC again be the uncovered parts initialized as $UC \leftarrow P(Q)$. For every k, we update $UC \leftarrow UC \setminus P(\hat{\mathbf{x}}_k^{n^*})$. The algorithm terminates when $UC = \emptyset$.

5.5. Multirobot Solutions From Single-robot Solutions

The greedy spirit of the single-robot greedy algorithms was readily extendable to the multirobot case by simply greedily selecting the best robot. The single-robot planning algorithms, however, are not extendable to the multirobot case in such a straightforward manner, since the naive approach immediately leads to an exponentially growing state space that makes the application of our planners infeasible. Our experimental comparisons from Section 6 additionally support this. Therefore, our multirobot planning approach starts with a single robot plan and splits this into N parts to produce N paths that will be executed by the robots in parallel. Although this is clearly not optimal, we gain the practical advantage of being able to utilize improved planners that work well for the single robot case without any additional effort, i.e., we can simply substitute a single robot planner with an improved version and simultaneously improve our multirobot plans. This advantage should not be underestimated, especially for the development of practical and fielded systems.

Our multirobot planner takes as the input a single robot coverage sequence $\{\mathbf{x}_1, \dots, \mathbf{x}_m\} \subseteq \tilde{X}$ that covers $\mathcal{S}$. Our goal is to split this into N sequences $\{\mathbf{x}_1^n, \dots, \mathbf{x}_{m(n)}^n\} \subseteq \tilde{X}$ for each robot $n \in \{1, \dots, N\}$ with length $m(n)$. There are $N-1$ splitting points s_n for N robots. A splitting point $s_n \in \{1, \dots, m\}; i < j \Rightarrow s_i < s_j$ defines the end index of the nth robot's subsequence, so that $\mathbf{x}_{m(n)}^n = \mathbf{x}_{s_n}$. Starting points are one past the end of the previous robot's sequence, so that the subsequences connect, i.e., $\mathbf{x}_1^n = \mathbf{x}_{s_{n-1}+1}$. The first robot's subsequence must start with the first sensor view of the single-robot plan, i.e., $\mathbf{x}_1^1 = \mathbf{x}_1$, and likewise the last robot's subsequence must complete the single-robot sequence, so that all views are covered, i.e., $\mathbf{x}_{m(N)}^N = \mathbf{x}_m$. There are in the order of m^{N-1} ways to perform such splits. As long as the number of robots N is not too large, it is still feasible to enumerate all solutions, despite the exponential complexity in the number of robots. We do so and select the split that minimizes the overall execution time as our solution.

The method described above can clearly be improved, e.g., by k-means clustering of waypoints to prevent different robots operating in the same local area or using approximation algorithms for the multiple traveling salesman problem. In the experimental section, however, we shall see that the complexity of this step plays a minor role in the overall computation time. A multiple traveling salesman approach might also improve the quality of solutions, but it would require access to a robust implementation of a solver for this problem. Again, the experimental section

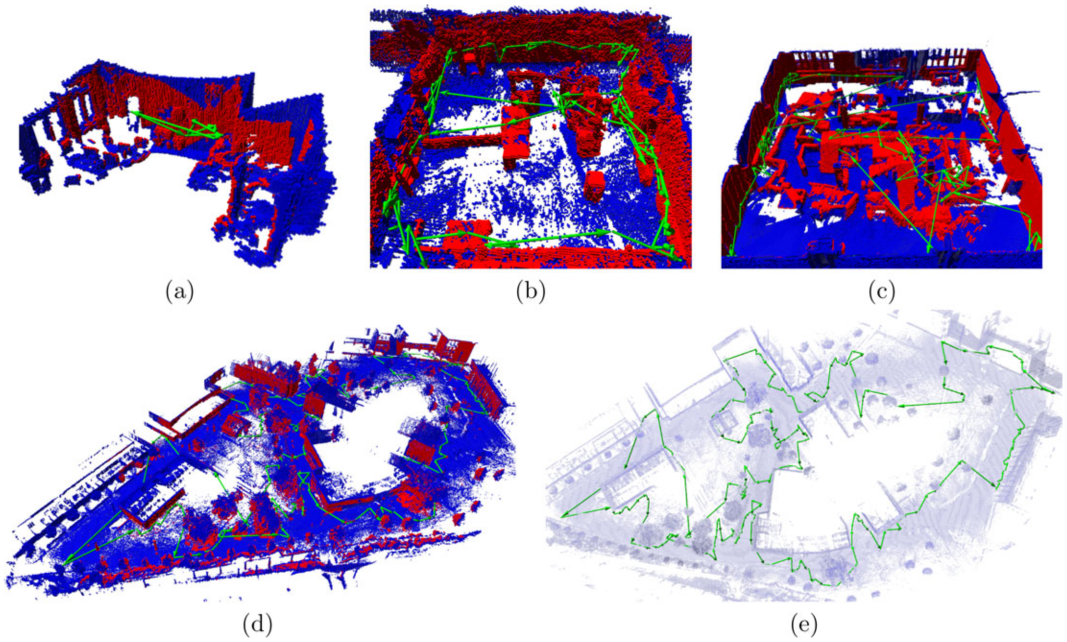


Figure 3. This figure shows the data sets used for evaluation: The 3D scan in our lab (a), Building 78 (b), the rescue arena (c), and the Computer Science Campus at the University of Freiburg (d). For a better visualization of the plan in the campus, a top-down view is given (e). The plans (green) are generated with the Set Cover/TSP (LKH) method for one robot. Occupied cells are displayed in blue, covered cells are red.

will show that even this rather naive splitting method already produces reasonable multirobot solutions. Section 7 also discusses issues and possible improvements of this approach in practice.

Applying the above splitting method to the single-robot algorithms described in Sections 5.1 and 5.3 leads to four new variants. To describe these, we append an '-S' to the single-robot variant name leading to the new variants *Simple-Greedy-S*, *Greedy-NBV-S*, *Set-Cover/TSP (TFD)-S*, and *Set-Cover/TSP (LKH)-S*.

6. EXPERIMENTS AND EVALUATION

We evaluated our approach and algorithms on four real-world data sets, i.e., OctoMaps obtained from sensor data collected by robots in real environments. In addition, we carried out real-world experiments in which robots execute the computed solutions for 3D coverage search. The results of the experiments on the data sets are presented in Section 6.2, and the results of the real-world experiments are found in Section 6.4. Section 6.1 discusses how to compute travel times efficiently, while Section 6.3 briefly discusses optimality for the planning problem.

The four data sets represent one small indoor, two large indoor, and one large outdoor environment. The small indoor data set consists of a 3D scan taken in the robotics lab at the University of Freiburg. The two large indoor data sets were recorded in Building 78 at the University of Freiburg, which consists of two rooms separated by a door and at the NIST rescue arena at Jacobs University in Bremen. The large outdoor data set was recorded on the Computer Science Campus at the University of Freiburg. OctoMaps for the indoor data sets are generated with 5 cm resolution, while the outdoor data set uses a 20 cm resolution. Visualizations of the maps obtained from the data sets are shown in Figure 3.

In all simulation experiments, the search set $\mathcal{S}$ to be examined consists of all vertical structures of the map, thus aiming for a complete coverage of everything that is not a floor or a ceiling. This choice models an inspection task for inspecting walls, but for our purposes it serves to have a large search set $\mathcal{S}$ in a larger environment $\mathcal{E}$, but within a similar order of magnitude, i.e., a similar number of cells that are vertical vs horizontal.

The robot model used is a mobile ground robot with the sensor mounted on a 6-DOF manipulator with a reach of 1 m. The sensor model is a camera with a 60° horizontal and a 40° vertical field of view. For the indoor data sets, a maximum range of 5 m was used; the outdoor data set was searched with a 35 m sensor. The small indoor map, denoted as *Lab*, was used for two scenarios. The first scenario, denoted by *Lab 1*, only allowed manipulator movements and no motion of the ground platform. The map is sufficiently small so that the manipulator has a reasonably large X_{reach}.

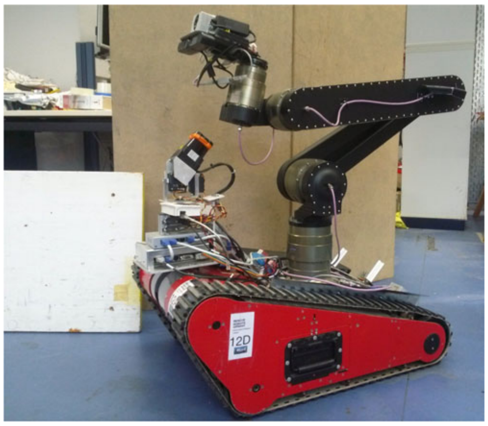


Figure 4. One of the robot platforms used for coverage search. Sensors are mounted on a versatile manipulator arm.

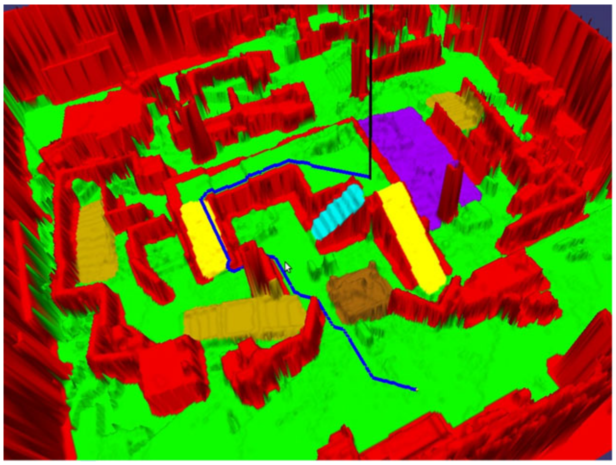


Figure 5. Computation of travel time on segmented elevation maps. Shown is the generated plan between two locations (blue line): traversable terrain (green) and nontraversable terrain (red). The segmentation represents structural elements such as stairs (magenta), ramps (yellow), and random step fields (violet).

This searcher model is motivated by a common setup for USAR robots depicted in Figure 4: A tracked robot with a manipulator arm as a sensor platform. The second scenario, denoted by *Lab 2*, allowed manipulator movements and navigation for the ground platform. In all other maps we considered manipulator movements and navigation for the ground platform.

The variants of our algorithm, described in detail in Section 5, that we used for the experiments are Multi-Greedy-NBV, Greedy-NBV-S, Multi-Simple-Greedy, Simple-Greedy-S, Set-Cover/TSP (TFD)-S, and Set-Cover/TSP (LKH)-S. Note that all variants with appended '-S' default to the single-robot variant when $N = 1$, since the single-robot solution is split into only one segment and therefore remains intact. The TFD planner that was used during the experiments supports both numerical values and temporal actions. But since we are only interested in numerical computations, temporal actions have been disabled for our experiments. The LKH solver was executed with the default parameters supplied by the software. All greedy variants of the algorithm were run until a solution was found.

6.1. Efficient Travel Time Computation

As noted in the problem definition in Section 3, we require the computation of time estimates for moving between different sensor states on the map, i.e., the function cost($\mathbf{x}_i, \mathbf{x}_j$). A major part of the computation time is spent when computing the travel time between poses. We base our travel time planner on value iteration, a popular dynamic programming algorithm frequently used for robot planning (Burgard et al., 1998). As shown by Figure 5, the planner takes as input a segmented elevation map in which important structural elements such as stairs and ramps are discriminated and indicated by a different color. Value Iteration computes then efficiently for each grid cell (e_x, e_y) on the elevation map a time estimate for reaching a goal cell (g_x, g_y). These time estimates are composed of travel distance as well as costs for overcoming different types of terrain. Costs differ according to the terrain type (e.g. stair, ramp, etc.) and the robot type (e.g. tracked or wheeled ground vehicle, UAV, etc.).

The resulting value function is then used by an $A*$ planner as heuristic for finding the shortest paths (and computing their times) on the map.

Besides these travel times between robot base poses, we are also considering the time for moving the manipulator from one view configuration to the next based on the maximum angular displacement of any joint. The expected combined time defines the cost of moving between two sensor states, and hence we obtain cost($\mathbf{x}_i, \mathbf{x}_j$).

6.2. Evaluation of Coverage Search Algorithms

The first series of experiments applies all variants of our multirobot coverage search algorithm to the scenarios generated from our real-word data sets, using one to four robots. For these experiments, we are reporting computation times and the planned execution times of the best plans found by the algorithms on each of the maps. Since the generation of views for $\tilde{X}$ involves randomization, we ran the algorithm ten times for each scenario, and we report mean values with standard deviation. The minimum utility ϵ to accept a view as well as the number of views, N_{sensor}, that need to be generated for $\tilde{X}$ was chosen with respect to the average expected utility, which depends on the sensor model and the environment. Therefore, for the outdoor environment with

Table I. This table shows computation times for the generation of $\tilde{X}$ and its minimal partition. In addition, the reduction factor achieved by the partition is given, as well as the parameters ϵ and N_{sensor} that were used.

Scenario	Lab 1	Lab 2	Bldg. 78	Arena	Campus
Computing $\tilde{X}$ (s)	2.12 ± 0.20	3.01 ± 0.45	30.80 ± 2.47	78.96 ± 4.85	201.90 ± 10.09
Minimal Partition (s)	0.06 ± 0.01	0.04 ± 0.00	2.19 ± 0.27	5.48 ± 0.67	31.54 ± 2.83
Reduction Factor	127.23 ± 17.14	49.27 ± 12.48	16.48 ± 1.88	37.02 ± 4.58	71.57 ± 4.18
ϵ (dm^3)	150	150	125	250	8000
N_{sensor}	15	15	100	142	280

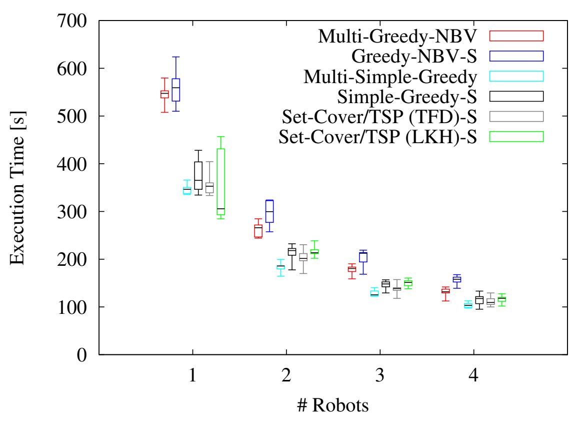
Table II. This table shows execution time in seconds for the Lab 1 and Lab 2 scenarios.

Scenario	Lab 1			
Num Robots	1	2	3	4
Multi-Greedy-NBV	25.7 ± 3.2	12.6 ± 2.3	9.1 ± 1.5	7.5 ± 0.8
Greedy-NBV-S	28.8 ± 3.5	17.3 ± 3.9	10.1 ± 1.6	9.8 ± 2.1
Multi-Simple-Greedy	25.9 ± 3.3	13.9 ± 2.2	10.0 ± 1.3	8.0 ± 1.2
Simple-Greedy-S	30.3 ± 7.3	15.8 ± 4.2	11.9 ± 3.1	9.8 ± 2.1
Set-Cover/TSP (TFD)-S	23.4 ± 7.6	11.9 ± 3.4	9.9 ± 2.6	8.0 ± 2.1
Set-Cover/TSP (LKH)-S	21.6 ± 5.4	14.9 ± 3.5	9.3 ± 2.5	8.1 ± 1.9
Scenario	Lab 2			
Num Robots	1	2	3	4
Multi-Greedy-NBV	60.4 ± 6.9	31.4 ± 3.8	22.0 ± 2.1	17.3 ± 1.8
Greedy-NBV-S	66.6 ± 8.4	38.0 ± 2.6	27.8 ± 3.2	21.1 ± 2.1
Multi-Simple-Greedy	50.5 ± 6.9	29.6 ± 2.9	22.8 ± 2.3	19.3 ± 1.9
Simple-Greedy-S	57.1 ± 5.9	33.9 ± 5.1	26.4 ± 4.1	20.9 ± 2.5
Set-Cover/TSP (TFD)-S	54.3 ± 11.6	35.7 ± 5.5	25.1 ± 4.0	21.4 ± 3.9
Set-Cover/TSP (LKH)-S	60.7 ± 10.0	35.0 ± 5.1	26.5 ± 4.7	22.3 ± 3.2

a 35 m sensor, this minimum needs to be significantly higher than the one chosen for an indoor environment with a limited field of view. The choices of parameters are shown in Table I. Table I also shows computation times for the first part of our algorithm, i.e., the generation of views for $\tilde{X}$ and the minimal partition. In addition, we provide a *reduction factor* achieved by the partition. This reduction factor is defined as the ratio of cells in $\mathcal{S}$ to parts in the partition. It provides a measure for the reduction of the search space achieved by the sampling of $\tilde{X}$ and its minimal partition. With a large reduction factor, the number of parts in the partition is significantly smaller than the number of cells that are to be covered, and the input size to the planning algorithms can be thought of as being reduced by this factor. In addition, the reduction factor also relates to the complexity of the environment, and a low reduction factor suggests that the environment is complex and cluttered.

Each of the ten $\tilde{X}$ and minimal partitions obtained for every scenario were used as an input to the variants for the coverage search presented in Section 5. This was done with one, two, three, and four robots, i.e., $N = 1, \ldots, 4$. Table II shows the average time to execute a solution for the smaller Lab scenarios, where all algorithms performed similarly well. Figures 6 and 7 plot the resulting execution time for each variant and number of robots for the Bldg. 78, Arena, and Campus scenarios. Table III shows the average measured computation time required to compute the solutions for each variant, number of robots, and scenario.

For the greedy variants, we observe that the Simple-Greedy variants usually perform better than Greedy-NBV. The balancing of travel time and utility of views in Greedy-NBV does not pay off in these experimental scenarios. One possible explanation for this effect is that the selection of views for $\tilde{X}$ is already biased toward high utility views, and the additional consideration of utility penalizes travel times too much. This suggests that when considering to use utilities in a greedy approach, a different tradeoff equation than the simple ratio could be more beneficial. It is unclear, however, which tradeoff can lead to a good greedy heuristic, and the simple greedy approach seems already to perform rather well.



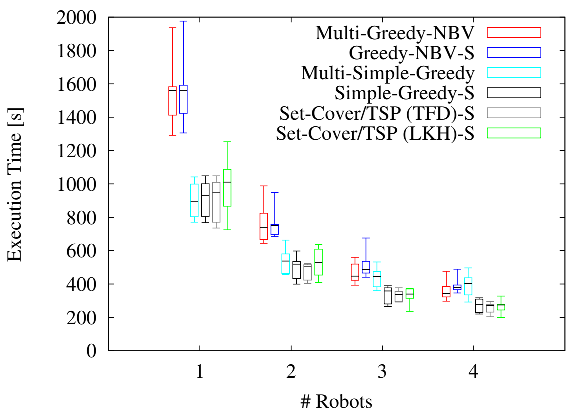


Figure 6. Box plots of the execution time for $N = 1, \dots, 4$ of multiple algorithm variants for the Bldg. 78 (left) and Arena (right) scenarios.

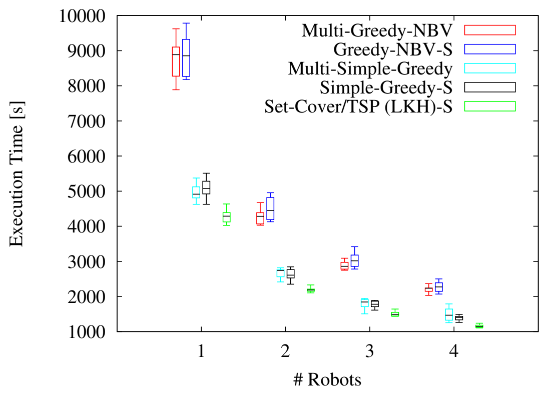


Figure 7. A box plot of the execution time for $N = 1, \dots, 4$ of multiple variants of our algorithm for the Campus scenario. The variant Set-Cover/TSP (TFD) was not applicable due to insufficient memory.

Another interesting observation is found in the comparison of the '-S' variants of the greedy algorithms (Greedy-NBV-S and Simple-Greedy-S) against the multirobot greedy variants (Multi-Simple-Greedy and Multi-Greedy-NBV). For the medium sized scenario, i.e., Bldg. 78, the multirobot variants perform slightly better (see Figure 6). This advantage, however, vanishes for the larger maps (Arena and Campus). Now, the computation times for the multirobot variants scale linearly with the number of robots (see Table III). This is due to the fact that for each step, the single robot variants compute the travel times to all other views, while the multirobot implementation computes travels times to all other views for every robot. This linear growth in the number of robots has a much larger impact on computation time than the exponential growth in the number of robots, which is due to the splitting for the '-S' variants. As it happens in real applications, the constants hidden in the complexity classes can matter more than the complexity classes themselves, especially when considering a limited range of input sizes (with $N = 1, \dots, 4$). The same effect also explains why the Set-Cover/TSP algorithms are faster to compute than the multirobot greedy variants as the number of robots increases. Again, the computation of the single-robot solution dominates the computation time, and the splitting into segments, despite the exponential complexity, is comparatively fast and its overhead does not become relevant until four robots are used. The increase in computation time for four robots is particularly noticeable on the larger maps. This indicates that computing optimal scheduling by enumeration is unlikely to scale for larger number of robots, where more sophisticated scheduling algorithms must be applied.

For the Set-Cover/TSP algorithms, we generally see better results at the cost of increased computation time. The results for the smaller scenarios are similar. However, the Simple-Greedy-S algorithm is quite competitive. Although execution times computed for the NIST arena do not significantly differ, the computation times are shorter. With increasing problem size, the decomposed variant becomes superior on the largest map when using a specialized TSP solver. The planner initially transforms a planning task in a process called "grounding" to facilitate an efficient search. The very large problems become infeasible for the planning-based variants as in those cases the system runs out of memory during the grounding phase. A brief discussion of optimal planning is found in Section 6.3.

When we compare the execution time with an increasing number of robots, we see that all algorithms were able to utilize more robots efficiently. Overall planned execution times scale down almost linearly with an increasing number of robots, which is an important aspect, especially for the larger scenarios. The question of whether this scaling behavior extends to real-world settings, especially in smaller settings where robots can obstruct each other, will be addressed in Section 6.4.

Table III. This table shows computation times in seconds for the different scenarios. Set-Cover/TSP (TFD) was not applicable in the Campus scenario due to insufficient memory.

Num Robots	1	2	3	4
Scenario		Lab 1		
Multi-Greedy-NBV	0.0 ± 0.0	0.1 ± 0.0	0.1 ± 0.0	0.1 ± 0.0
Greedy-NBV-S	0.0 ± 0.0	0.1 ± 0.0	0.1 ± 0.0	0.1 ± 0.0
Multi-Simple-Greedy	0.0 ± 0.0	0.1 ± 0.0	0.1 ± 0.0	0.1 ± 0.0
Simple-Greedy-S	0.0 ± 0.0	0.1 ± 0.0	0.1 ± 0.0	0.1 ± 0.0
Set-Cover/TSP (TFD)-S	2.6 ± 0.5	2.5 ± 0.5	2.6 ± 0.4	2.6 ± 0.5
Set-Cover/TSP (LKH)-S	0.7 ± 0.2	0.6 ± 0.2	0.7 ± 0.2	0.6 ± 0.2
Scenario		Lab 2		
Multi-Greedy-NBV	0.2 ± 0.0	0.3 ± 0.0	0.4 ± 0.0	0.6 ± 0.0
Greedy-NBV-S	0.2 ± 0.0	0.2 ± 0.0	0.2 ± 0.0	0.3 ± 0.0
Multi-Simple-Greedy	0.1 ± 0.0	0.3 ± 0.0	0.4 ± 0.0	0.6 ± 0.1
Simple-Greedy-S	0.2 ± 0.0	0.2 ± 0.0	0.2 ± 0.0	0.2 ± 0.0
Set-Cover/TSP (TFD)-S	2.8 ± 0.8	2.8 ± 0.8	2.9 ± 0.5	2.6 ± 0.6
Set-Cover/TSP (LKH)-S	0.6 ± 0.0	0.6 ± 0.0	0.6 ± 0.0	0.6 ± 0.0
Scenario		Bldg. 78		
Multi-Greedy-NBV	19.0 ± 1.2	40.7 ± 1.3	59.7 ± 1.3	79.8 ± 4.7
Greedy-NBV-S	21.9 ± 1.2	22.0 ± 1.0	22.5 ± 1.0	34.4 ± 1.2
Multi-Simple-Greedy	17.9 ± 1.1	40.5 ± 1.5	58.9 ± 3.0	79.6 ± 4.3
Simple-Greedy-S	20.1 ± 1.3	21.2 ± 1.0	20.9 ± 0.6	33.3 ± 1.6
Set-Cover/TSP (TFD)-S	158.4 ± 3.5	159.3 ± 4.6	158.8 ± 4.4	174.1 ± 4.6
Set-Cover/TSP (LKH)-S	43.3 ± 2.0	43.9 ± 2.8	43.8 ± 2.2	56.6 ± 2.5
Scenario		Arena		
Multi-Greedy-NBV	64.4 ± 3.7	133.0 ± 9.0	201.0 ± 10.6	272.5 ± 12.8
Greedy-NBV-S	69.9 ± 4.3	71.6 ± 4.0	73.9 ± 3.3	103.9 ± 3.9
Multi-Simple-Greedy	61.5 ± 3.9	132.9 ± 9.3	210.3 ± 5.9	279.6 ± 15.3
Simple-Greedy-S	65.7 ± 3.7	68.3 ± 3.9	70.9 ± 3.4	101.8 ± 3.1
Set-Cover/TSP (TFD)-S	487.4 ± 33.0	488.7 ± 24.4	504.0 ± 27.1	523.7 ± 25.6
Set-Cover/TSP (LKH)-S	131.7 ± 11.7	131.9 ± 7.9	134.7 ± 8.9	165.1 ± 12.2
Scenario		Campus		
Multi-Greedy-NBV	690.0 ± 54.6	1471.7 ± 83.3	2164.0 ± 94.4	2847.0 ± 140.3
Greedy-NBV-S	781.8 ± 60.0	803.5 ± 51.8	800.6 ± 33.9	976.0 ± 52.8
Multi-Simple-Greedy	717.9 ± 50.4	1488.8 ± 106.0	2229.5 ± 129.4	2947.6 ± 178.6
Simple-Greedy-S	815.4 ± 44.8	844.6 ± 49.8	845.6 ± 30.6	1071.4 ± 50.4
Set-Cover/TSP (LKH)-S	1249.2 ± 97.8	1285.0 ± 105.2	1275.2 ± 98.0	1432.7 ± 110.9

6.3. Optimal Solutions and Anytime Planning

The first set of experiments investigated the performances of all our variants except the Complete Planning approach. The Set Cover/TSP approaches are closely related to the formulation for the Complete Planning approach, but they decompose the problem. As briefly discussed in Section 5.3, this decomposition can lead to suboptimal solutions. Clearly, it is of interest to experimentally determine the loss of quality of the solutions that is due to the decompositions into set cover and TSP problems. One problem for such an experiment is that the Complete Planning variant only runs within a reasonable amount of time on small maps, and computing optimal solutions for the larger maps was not feasible. Thus we used the first $\tilde{X}$ and minimal partition obtained for the smallest map (Lab), and we ran the *Complete Planning* variant against the *Set Cover/TSP (TFD)* until the state space was completely explored. We used the TFD planner in both instances, which allowed us to find optimal solutions and prove optimality.

Table IV shows the quality of the first plan found by the respective variant, also including Simple-Greedy as a

Table IV. Comparison of the best results of which the algorithms are capable. Computation time and planned execution time until the first and best plan are found are listed together with the time of the full run needed by the planners to prove the best plan to be optimal.

Algorithm		First solution	Best solution	Optimality
Simple-Greedy	execution time (s)	22.22	22.22	
	computation time (s)	0.03	0.03	
Set Cover/TSP (TFD)	execution time (s)	19.34	16.83	
	computation time (s)	0.15	7.76	100.09
Complete Planning	execution time (s)	20.79	15.78	
	computation time (s)	0.16	12,722.88	33,281.29

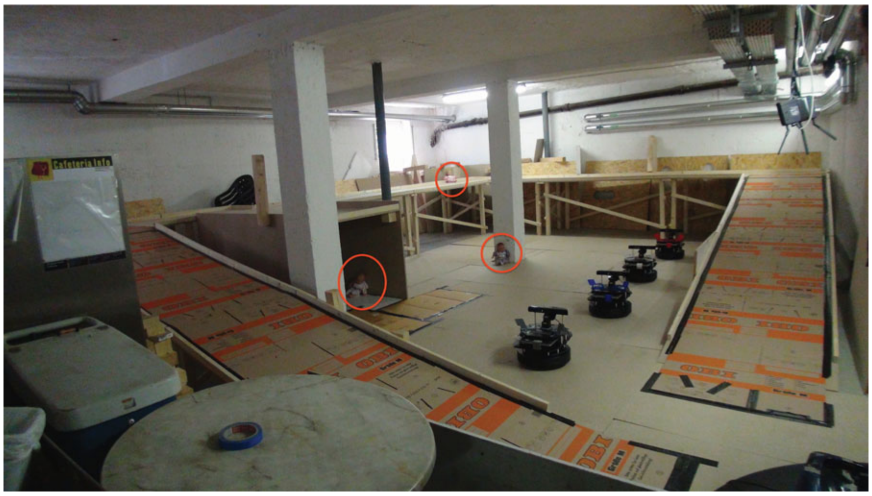


Figure 8. Two-story test environment for multirobot experiments with marked search targets.

reference, the best plan, and the time it took to determine the plan and prove optimality.

All variants quickly found a reasonable first solution with the Set Cover/TSP (TFD) variant finding the best first solution, although taking an order of magnitude longer than the greedy variant. As one would expect, the final optimal solution returned by the Complete Planning variant is better (15.78 s) than the best solution for the decomposition approach Set Cover/TSP (TFD) (16.83 s), which is only optimal for the set cover and TSP problem independently. The Set Cover/TSP (TFD) solution is 6.6% longer than optimal, yet the reduction in computation time from 12,722.88 s for the optimal solution to 7.76 s is significant. In addition, the Complete Planning approach only found a slightly better plan than 16.83 s after 4,996 s. Hence, with an anytime approach, the Complete Planning formulation is not likely to yield any improvements over the decomposition approach. A complete theoretical or experimental analysis of related questions, e.g., the derivation of approximation factors, is beyond the scope of this paper. This brief experimental investigation, however, indicates that our choice to decompose the problem has a reasonable tradeoff between quality and computation time. This observation encourages the use of our algorithms for real-world scenarios with the expectation of finding solutions with reasonable quality. The next section demonstrates the application of our approach with a team of real robots.

6.4. Real-world Experiments

We performed real-robot experiments with up to four robots in a two-story test environment,[4] as shown in Figure 8. The goals for these experiments are manifold. First, we show that the algorithms presented can be applied in practice. From the actual execution, we learn how far the simulation results can predict the real-world performance relative between algorithms, i.e., which will lead to shorter execution times, and absolute, i.e., do the observed execution times lie within reasonable margins of the planned execution time.

[4]A video of the experiments is available at http://www.youtube.com/watch?v=jEFZMoxNGMI

Table V. This table shows computation times for the generation of $\tilde{X}$ and its minimal partition for the real-world scenario shown in Figure 8.

Computing $\tilde{X}$ (s)	Minimal partition (s)	Reduction factor	ϵ (dm^3)	N_{sensor}
4.8	0.02	1.92	1500	45

In addition, we also determine how well the scaling properties of the multirobot solutions transfer to the real world and where the limitations of an offline approach lie. As noted previously, our approach does not directly consider multirobot collisions, view occlusions, and other issues arising when using multiple robots. These are dealt with on the implementation level for our specific system, and they may also have an impact on the real execution of the coverage search solutions.

The experiments are performed in a two-story test environment that allows the robots to observe the lower level from the upper level, and it has a cavelike section to create a three-dimensional problem. Figure 8 shows the test environment. The test environment was built according to a manually created three-dimensional blueprint. In contrast to the experiments presented in the previous sections, we did not build an OctoMap from sensor data, rather we used the 3D blueprint to generate an OctoMap with a resolution of 0.05 m. This OctoMap was used as an input for our algorithms. We ran the sampling for $\tilde{X}$ to search for volumes of 0.064 m^3—a volume too small for a human not to be found in. Computation times for this are shown in Table V. To determine a realistic *cost*-function, we ran preliminary experiments and matched the *cost*-function to the observed execution times. We ran the multirobot coverage search algorithms from Section 5 for one to four robots, and we executed the resulting plans on the robots. As a robotic platform, we use four modified Turtlebot 2's, shown in Figure 9, that are equipped with a laser range finder for localization and a Kinect RGBD camera that is used as the observation sensor. Note that the robot is different from the robot used for the experiments presented in the previous section; most importantly, it does not have its camera mounted on a manipulator. Each robot gets its specific path preloaded for an experiment and all robots visit their sequence of sensor locations in parallel and record a view at each of these in the form of a 3D point cloud. For safety reasons, ramp transitions between levels have been teleoperated. All other actions, especially navigation to sensor locations, were autonomous. Thus in total there were 24 multirobot coverage search runs containing 60 individual robot runs. Figure 11 gives the measured overall execution time for each algorithm and number of robots in comparison to the overall execution times computed by the algorithms. Table VI gives the computation times for running the multirobot coverage search algorithms, and it shows the total path length of all robots for each algorithm and number of robots.

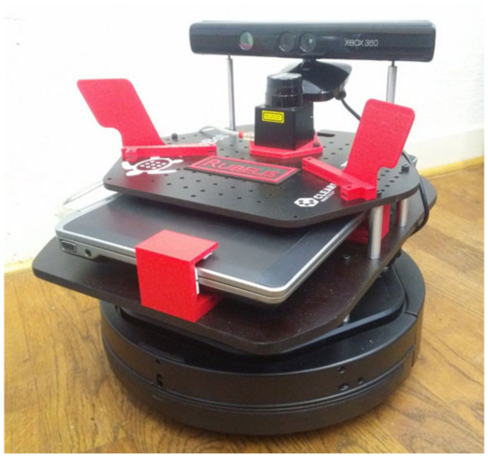


Figure 9. One of the Turtlebot 2 robots used in the experiments with a Kinect sensor and a Hokuyo laser.

7. DISCUSSION

We will now discuss in detail the results from the previous experiments, especially the applicability of our algorithms to real-world systems and with respect to the observations from simulation experiments. First, we report that the observed coverage in the real sensor data was never lower than 98.4% from what the algorithms predicted for any run. This indicates that the presented algorithms provide a viable solution to the coverage search problem for realistic environments. As an example for an execution, see Figure 10. Comparing the algorithms, we see that the Greedy-NBV variants perform well in comparison to the other variants, especially against Simple-Greedy. This is due to the fact that Greedy-NBV considering higher utility views usually uses fewer views in total, which comes into play in this particular environment. In relation to the distance to be driven between views, the time to approach and record a view matters for this smaller environment in comparison to the larger simulation maps where the driving distance dominates the execution time given by the *cost* function. We can also see that in the driven path length. Greedy-NBV usually drives longer distances, but it is still faster than Simple-Greedy with real robots. In such a situation, Greedy-NBV should be preferred to Simple-Greedy. Nevertheless, the Set-Cover/TSP variants still have better performance than all greedy algorithms in almost all cases in their planned execution times and, more importantly, also

Table VI. This table shows the computation times for the different algorithms (left) and the total path length driven by all robots in a run.

	Computation time (s)				Path length (m)			
Num Robots	1	2	3	4	1	2	3	4
Multi-Greedy-NBV	6.7	14.4	20.9	27.8	87.0	59.5	50.6	67.6
Greedy-NBV-S	6.7	7.2	7.5	7.5	70.1	80.0	70.6	76.0
Multi-Simple-Greedy	12.4	28.0	40.3	53.0	39.4	44.5	58.7	93.2
Simple-Greedy-S	13.0	13.8	14.1	14.4	44.6	48.6	52.7	53.4
Set-Cover/TSP (TFD)-S	49.7	47.0	47.6	49.2	30.5	33.3	34.7	52.1
Set-Cover/TSP (LKH)-S	37.5	31.5	29.2	29.5	48.7	32.7	48.1	40.0

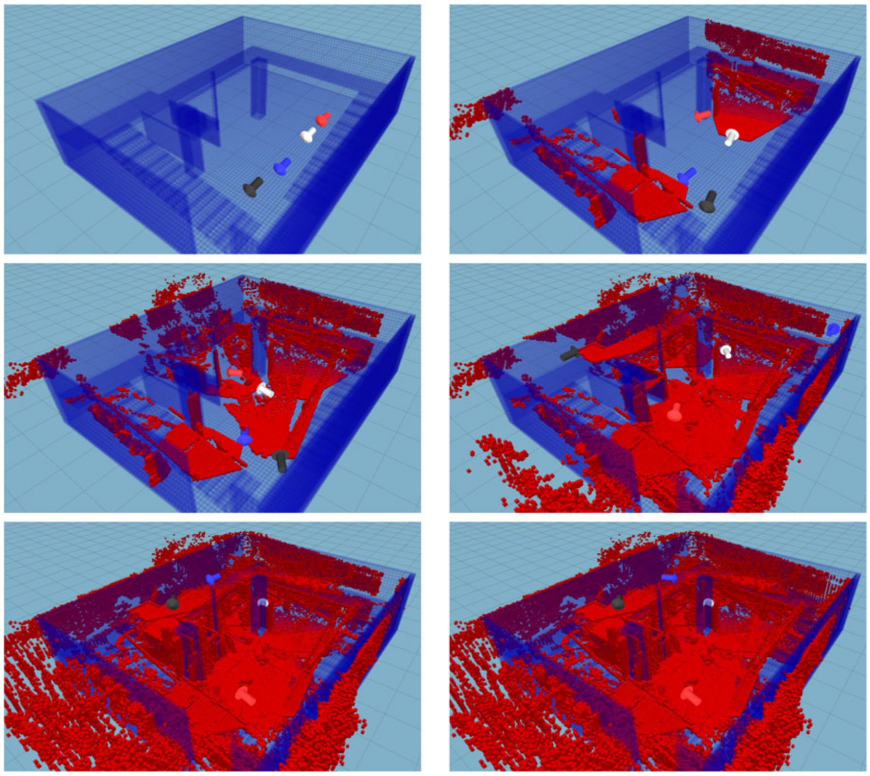


Figure 10. Illustration of multirobot coverage search in the environment shown in Figure 8. The four-robot solution for Set-Cover/TSP (LKH) is displayed. End points of view rays are shown in red.

in the observed execution times. This means that the longer computation times for these algorithms do pay off in faster execution times. This fact is relevant for real-world applications as offline computation time is usually cheaper than robot operation time. Only for online approaches or when running the algorithms on the robot itself upon deployment might one prefer a lower computation time, such as with *Greedy-NBV-S*.

If we compare the planned execution time with the real execution time, we observe that computed times are not a very accurate prediction of real execution times, although the overall ranking between algorithms is still fairly similar whether one considers computed or real execution times. This is not really surprising as autonomous robot navigation is influenced by many factors, including but not limited to sensor noise and inaccurate motion execution. These affect our autonomous navigation and are contained in measured travel times and thus execution times. But more importantly, the real execution time is determined by the slowest robot, and when running multiple robots it is more likely that

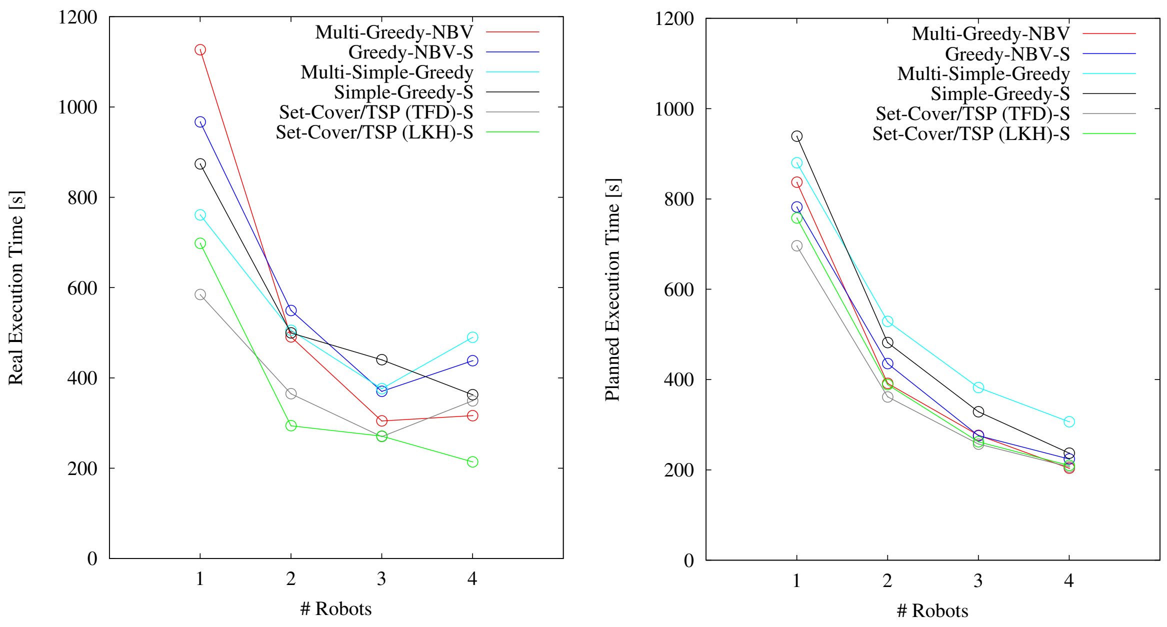


Figure 11. This figure shows the execution times of the real robots (left) in comparison to the planned execution time by the coverage search algorithms (right).

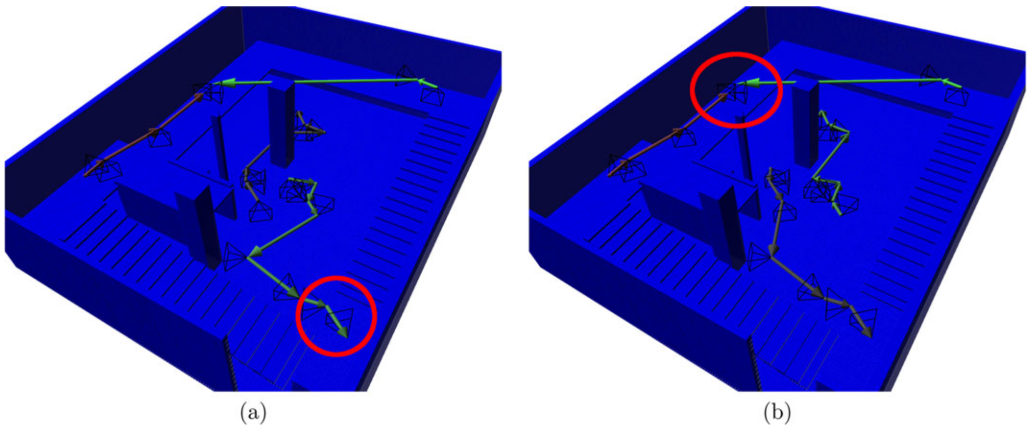


Figure 12. Illustration of suboptimal real-world behavior. Four-robot solutions from *Set-Cover/TSP (TFD)-S* (a) and *Set-Cover/TSP (LKH)-S* (b) are shown. The path to both ramps is in a confined space marked on the left image. If a robot assigned to the lower level temporarily blocks access to the ramps, robots aiming for the upper level must wait. The final poses of two robots' paths might lie next to each other (marked on the right image), predicting both robots arriving at the same time. It is likely that one robot arrives earlier and thus should target both views.

at least one will have some delays in its path, increasing the likelihood of a delay with each new robot. Here our assumption that robots operate independently also comes into play. A robot blocking the path of another robot temporarily might have bad effects on the blocked robot's travel time. This is what happened in the four-robot case for Set-Cover/TSP (TFD)-S. See Figure 12(a) for an illustration. Another example can be seen in Figure 12(b). Final poses of two robot paths lie next to each other. Although it is the best solution, in terms of computed execution time, it is very likely that one robot will arrive at that location earlier than the other. In such a case, it would be beneficial to transfer the assignment of sensor locations to the robot that is already there. Conflicting situations between robots are not always that extreme, but they do explain most of the cases in which solutions for three or four robots have computed execution times that underestimate the real execution times. These situations are most effectively solved online during execution, but it is still beneficial to have plans that distribute robots preventing possible conflicts in the first place. An example is the four-robot execution for the Set-Cover/TSP (LKH)-S solution. Nevertheless, we are approaching the limits of gaining performance by scaling up the number of robots for this environment. While keeping this in mind, using multiple robots to solve the coverage search problem clearly reduces the time needed to solve the coverage search, and the algorithms utilize additional robots fairly effectively.

8. CONCLUSIONS

We considered the problem of identifying and planning efficient sequences of sensor locations for covering complex environments represented in three dimensions. For that purpose, we introduced a sampling based method that reduces the size of the search space by selecting a large number of high utility sensor locations, which can then be used to efficiently plan sequences of sensor locations. We introduced several variants for the planning problem for single robots and multirobot teams. We evaluated these empirically in order to determine the tradeoff between computation time and execution time of the solutions. Our results in simulation and real-world experiments indicate that despite the intractability of the problem, efficient multirobot coverage in three dimensions is feasible.

Small size problems, such as incremental vicinity exploration by a single robot, were solved close to real time and thus can directly be deployed in real-world applications. For larger problems, our *Simple-Greedy* variant provided solutions that were competitive with the ones based on more elaborate planning approaches that solve the set cover and traveling salesman subproblems. If the time for visiting a single sensor location is noticeable, the *Greedy-NBV* algorithm is a viable alternative. Although the computation times for the *Simple-Greedy* variants were smaller than for the advanced solutions, the latter are still the overall better choice since they result in lower execution times. This is especially the case when the map is known prior to deployment and not transmitted to robots after deployment, giving more time for offline computations.

As one would expect, performance increases when adding more robots. All algorithms were able to reduce the execution time for the coverage search problem when given more robots. Yet, the dedicated Multi-Robot Greedy solutions only showed advantages for small- to medium-sized problems. The decomposition approach that solves the set cover and traveling salesman subproblems turned out to be superior, i.e., producing high-quality solutions in a short amount of time, especially in combination with the TSP solver. Even the splitting of the resulting single robot solution into a solution for multiple robots did not lead to inferior performance compared to the Multi-Robot Greedy solutions, which do not require the splitting. Overall, producing multirobot plans from single-robot plans was shown to be a good approach for extending single-robot algorithms.

Our real-world experiments have shown that the simulation results transfer well to robots acting in real environments. The same algorithms that have been shown to be superior in simulation also performed better in reality, especially if we value execution time more than offline computation time. An increasing number of robots also led to shorter execution times when the robots were well-distributed. Although one would expect that advanced dedicated multirobot algorithms might produce better solutions with lower execution times, our experiences from the real-world experiments suggest that the strongest potential for improvement is to consider online adaptation of solutions. This is mainly due to the fact that robot execution times are hard to predict precisely in reality, and the location of an error or unanticipated delay is crucial for finding a high-quality solution that mitigates the delay. Overall, despite the large amount of future work one may carry out to improve a multirobot 3D coverage search, we were able to show that our current approach is already well-suited for real-world applications.

ACKNOWLEDGMENTS

The Freiburg campus data set was originally recorded by Bastian Steder. Building 78 is courtesy of Jürgen Hess and Felix Endres. Both are available at http://ais.informatik.uni-freiburg.de/projects/datasets/octomap/. The rescue arena data set was recorded by Dorit Borrmann, Jan Elseberg, and Andreas Nüchter from Jacobs University Bremen gGmbH, Germany. The source data are available from http://kos.informatik.uni-osnabrueck.de/3Dscans/. This work was partially supported by the German Research Foundation (DFG) grants SFB/TR-8 project R7 and within the HYBRIS research group (NE 623/13-1) project PAC-MAN as well as by the European Commission FP7 grant PCIG14-GA-2013-631553.

APPENDIX A

INDEX TO MULTIMEDIA EXTENSIONS

Extension	Media Type	Description
1	Video	Shows execution of Multirobot Coverage Search with three different algorithms

REFERENCES

Agmon, N., Hazon, N., & Kaminka, G. A. (2008). The giving tree: Constructing trees for efficient offline and online multi-robot coverage. Annals of Mathematics and Artificial Intelligence, 52(2-4), 143–168.

Applegate, D. L., Bixby, R. E., Chvatal, V., & Cook, W. J. (2007). The traveling salesman problem: A computational study. Princeton University Press.

Banta, J., Zhieng, Y., Wang, X., Zhang, G., Smith, M., & Abidi, M. (1995). A "best-next-view" algorithm for three-dimensional scene reconstruction using range images. Proceedings of SPIE (Intelligent Robots and Computer Vision XIV: Algorithms), 2588.

Bektas, T. (2006). The multiple traveling salesman problem: An overview of formulations and solution procedures. Omega, 34(3), 209–219.

Bourgault, F., Makarenko, A., Williams, S., Grocholsky, B., & Durrant-Whyte, H. (2002). Information based adaptive robotic exploration. In Proceedings of the 2002 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (pp. 540–545). IEEE.

Burgard, W., Cremers, A. B., Fox, D., Hähnel, D., Lakemeyer, G., Schulz, D., Steiner, W., & Thrun, S. (1998). The interactive museum tour-guide robot. In Proceedings of the National Conference on Artificial Intelligence (pp. 11–18). Wiley.

Choset, H. (2001). Coverage for robotics—A survey of recent results. Annals of Mathematics and Artificial Intelligence, 31(1), 113–126.

Chung, T., Hollinger, G., & Isler, V. (2011). Search and pursuit-evasion in mobile robotics. Autonomous Robots, 31(4), 299–316.

Cortes, J., Martinez, S., Karatas, T., & Bullo, F. (2002). Coverage control for mobile sensing networks. In Proceedings of the IEEE International Conference on Robotics & Automation (ICRA) (vol. 2, pp. 1327–1332). IEEE.

Dornhege, C., Eyerich, P., Keller, T., Trüg, S., Brenner, M., & Nebel, B. (2009). Semantic attachments for domain-independent planning systems. In Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS) (pp. 114–121).

Dornhege, C., & Kleiner, A. (2011). A frontier-void-based approach for autonomous exploration in 3d. In IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR) (pp. 351–356). IEEE.

Dornhege, C., Kleiner, A., & Kolling, A. (2013). Coverage search in 3d. In Proceedings of the Symposium on Safety, Security and Rescue Robotics (SSRR).

Englot, B., & Hover, F. S. (2013). Three-dimensional coverage planning for an underwater inspection robot. The International Journal of Robotics Research, 32(9-10), 1048–1073.

Fox, M., & Long, D. (2003). PDDL2.1: An extension to PDDL for expressing temporal planning domains. Journal of Artificial Intelligence Research, 20(1), 61–124.

Golovin, D., & Krause, A. (2011). Adaptive submodularity: Theory and applications in active learning and stochastic optimization. Journal of Artificial Intelligence Research, 42, 427–486.

Gonzalez-Banos, H., Mao, E., Latombe, J., Murali, T., & Efrat, A. (2000). Planning robot motion strategies for efficient model construction. In Proceedings of the 9th International Symposium on Robotics Research (ISRR) (pp. 345–352). Berlin: Springer.

Helsgaun, K. (2000). An effective implementation of the lin-kernighan traveling salesman heuristic. European Journal of Operational Research, 126(1), 106–130.

Hornung, A., Wurm, K. M., Bennewitz, M., Stachniss, C., & Burgard, W. (2013). OctoMap: An efficient probabilistic 3D mapping framework based on octrees. Autonomous Robots, 34(3), 189–206.

Howard, A., Matarić, M. J., & Sukhatme, G. S. (2002). Mobile sensor network deployment using potential fields: A distributed, scalable solution to the area coverage problem. In Distributed Autonomous Robotic Systems 5 (pp. 299–308). Springer.

Jacoff, A., Weiss, B., & Messina, E. (2003). Evolution of a performance metric for urban search and rescue robots. In Performance Metrics for Intelligent Systems. NIST, Gaithersburg, MD.

Joho, D., Stachniss, C., Pfaff, P., & Burgard, W. (2007). Autonomous exploration for 3D map learning. Autonome Mobile Systeme (pp. 22–28). Springer-Verlag, Berlin Heidelberg.

Karp, R. (1972). Reducibility among combinatorial problems. Complexity of Computer Computations. Springer US, New York City.

Kleiner, A., Kolling, A., Lewis, M., & Sycara, K. (2013). Hierarchical visibility for guaranteed search in large-scale outdoor terrain. Journal of Autonomous Agents and Multi-Agent Systems, 26, 1–36.

Kollar, T., & Roy, N. (2008). Efficient optimization of information-theoretic exploration in slam. In Proceedings of the National Conference of Artificial Intelligence (AAAI 08). The AAAI Press, Menlo Park, California.

Kolling, A., Kleiner, A., Lewis, M., & Sycara, K. (2010). Pursuit-evasion in 2.5d based on team-visibility. In Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems.

Kong, C. S., Peng, N. A., & Rekleitis, I. (2006). Distributed coverage with multi-robot system. In Proceedings of the IEEE International Conference on Robotics & Automation (ICRA) (pp. 2423–2429). IEEE.

LaValle, S. M. (2006). Planning algorithms. Cambridge, UK: Cambridge University Press. Available at http://planning.cs.uiuc.edu/.

Lazebnik, S. (2001). Visibility-based pursuit-evasion in three-dimensional environments. Technical report, University of Illinois at Urbana-Champaign.

Lozano Albalate, M., Devy, M., Miguel, J., & Marti, S. (2002). Perception planning for an exploration task of a 3d environment. In International Conference on Pattern Recognition (pp. 704–707). IEEE Computer Society, Washington, DC.

Rekleitis, I., Lee-Shue, V., New, A. P., & Choset, H. (2004). Limited communication, multi-robot team based coverage. In Proceedings of the IEEE International Conference on Robotics & Automation (ICRA) (vol. 4, pp. 3462–3468). IEEE.

Renzaglia, A., Doitsidis, L., Martinelli, A., & Kosmatopoulos, E. B. (2011). Multi-robot 3d coverage of unknown terrains.

In Decision and Control and European Control Conference (CDC-ECC), 50th IEEE Conference (pp. 2046–2051). IEEE.

Sachs, S., Rajko, S., & LaValle, S. M. (2004). Visibility-based pursuit-evasion in an unknown planar environment. The International Journal of Robotics Research, 23(1), 3–26.

Shermer, T. (1992). Recent results in art galleries. Proceedings of the IEEE, 80(9), 1384–1399.

Surmann, H., Nüchter, A., & Hertzberg, J. (2003). An autonomous mobile robot with a 3d laser range finder for 3d exploration and digitalization of indoor environments. Robotics and Autonomous Systems, 45(3-4), 181–198.

Wurm, K. M., Dornhege, C., Eyerich, P., Stachniss, C., Nebel, B., & Burgard, W. (2010). Coordinated exploration with marsupial teams of robots using temporal symbolic planning. In Proceedings of the 2010 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2010).

Yamauchi, B. (1997). A frontier-based approach for autonomous exploration. In Computational Intelligence in Robotics and Automation, CIRA'97, Proceedings of the IEEE International Symposium (pp. 146–151). IEEE.

Zacharias, F., Borst, C., & Hirzinger, G. (2007). Capturing robot workspace structure: Representing robot capabilities. In Proceedings of the IEEE International Conference on Intelligent Robots and Systems (IROS2007) (pp. 4490–4495).