Coverage Search in 3D

Christian Dornhege, Alexander Kleiner and Andreas Kolling

Christian Dornhege, Alexander Kleiner and Andreas Kolling, Coverage Search in 3D, 2013, accepted: 11th IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR 2013).

Postprint available at: Linköping University Electronic Press
http://urn.kb.se/resolve?urn=urn:nbn:se:liu:diva-97300

Coverage Search in 3D

Christian Dornhege[1] and Alexander Kleiner[2] and Andreas Kolling[3]

***Abstract*—Searching with a sensor for objects and to observe parts of a known environment efficiently is a fundamental problem in many real-world robotic applications such as 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 efficient view point sequences for covering complex 3d environments. We compare empirically several variants of our algorithm that allow to trade-off schedule computation against execution time. Our results demonstrate that, despite the intractability of the overall problem, computing effective solutions for coverage search in real 3d environments is feasible.**

I. INTRODUCTION

Coverage search is a fundamental problem in many real-world scenarios and robotic applications such as household robots searching for objects in the house, area inspection in dynamically changing environments, e.g., searching for leaking pipelines or cracks in walls, and searching for survivors in debris after a disaster in Urban Search And Rescue (USAR). Particularly in USAR, survivors can be entombed within complex and heavily confined 3d structures. State-of-the-art tests for autonomous rescue robots, such as those proposed by NIST [1], 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 the execution of a planned viewpoint sequence in order to search the elevated area in the center of the scene (b-f).

We consider the problem of *3d coverage search* which is concerned with computing a sequence of 3d views for a searcher in order to cover a known 3d environment. The goal is to find a sequence of views that can be visited in the shortest time possible. It is assumed that the sensor has a specific field of view such as the opening angle and detection distance of an IR camera. This is similar to *coverage planning*, which has applications in lawn mowing, automated farming, painting, vacuum cleaning, and mine sweeping [2]. In coverage planning the goal is to compute an optimal shortest motion strategy

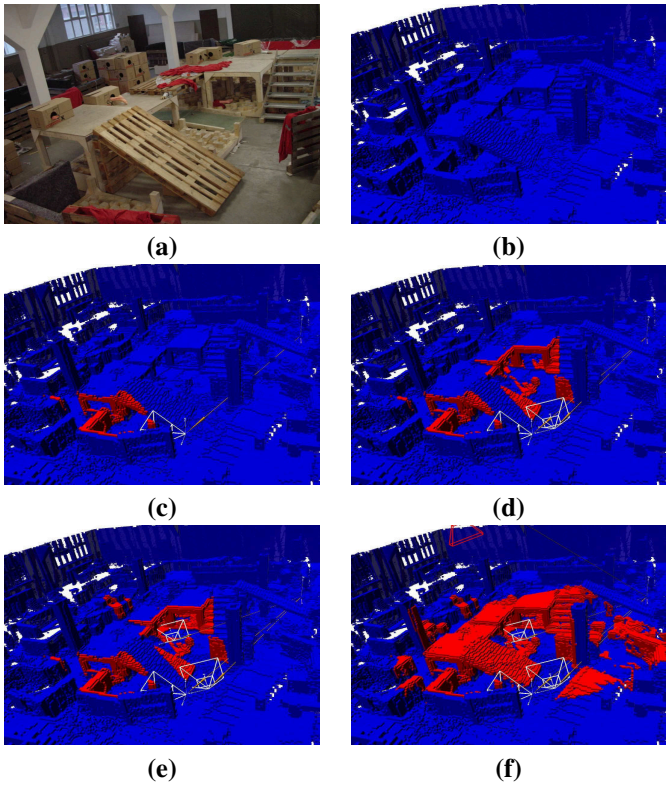

Fig. 1. Motivating example: Progressing search of the NIST rescue arena by visiting planned viewpoint locations. Unexplored area (blue) is stepwise covered (red) by sensors of the mobile platform. Arena photo shown in (a) is courtesy of the Jacobs Robotics Group, Jacobs University Bremen.

in order to cover a 2d environment with a mobile robot [3]. Solutions to coverage planning often employ decompositions of the free space in 2d. In contrast, our 3d problem relies on sampling as decompositions of 3d spaces is too costly. Note that our coverage problem is different from coverage problems that deploy multiple sensors to continuously monitor the same area for extend periods of time, such as in [4]. In our case every part of the environment has to be seen at most once. Other coverage-related problems often deal with unknown or partially known environments, such as in [5; 6] in 2D, and attempt to improve the map or explore unknown parts of the map. In our case the 3d map is already known and we are foremost interested in computing and visiting a set of useful views in the shortest time possible.

Coverage search also relates to other research fields that require the computation of views, such as next best view approaches for 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

[1]Christian Dornhege is with the University of Freiburg, Department of Computer Science, 79110 Freiburg, Germany, *email:dornhege@informatik.uni-freiburg.de*.

[2]Alexander Kleiner is with the Linköping University, Department of Computer Science, 58183 Linköping, Sweden, *email:alexander.kleiner@liu.se*.

[3]Andreas Kolling is with The University of Sheffield, Department of Automatic Control and Systems Engineering, Sheffield S1 3JD, United Kingdom, *email:andreas.kolling@gmail.com*.

This work is partially supported by the German Research Foundation (DFG) under contract number SFB/TR-8 project R7, the Swedish Foundation for Strategic Research and the Excellence Center at Linköping and Lund in Information Technology (ELLIIT), the Research Council (VR) Linnaeus Center CADICS, the Swedish Foundation for Strategic Research CUAS Project, and the EU FP7 project SHERPA, grant agreement 600958.

surface of an object has been observed by a sensor [7; 8]. These methods are, however, not suitable for coverage search on mobile robots since they ignore the costs for changing between different sensor poses [8]. Viewpoint calculation is also addressed by the art gallery problem [9]. There the task is to find an optimal placement of guards on a polygonal representation of 2d environments in that the entire space is observed by the guards. Furthermore, pursuit-evasion problems assume that the searched target is changing locations dynamically [10]. The challenge is to compute trajectories for the searchers (pursuers) in order to detect an evader moving arbitrarily through the environment. Besides 2d environments, 2.5d environments represented by elevation maps have been considered [10].

In contrast to previous work, our work on *3d coverage search* deals with realistic sensor models, the resulting complexity of visibility in 3d, and time constraints of robots navigating on rough terrain. With this we are generalizing and improving our previous work on *frontier-void* based exploration in 3d [11], which dealt specifically with searching for victims. Whereas in [11] search was focused on frontier cells combined with void structures detected in the scene, coverage search has the general goal of covering any a priori specified part of the 3d space with a specific sensor. More precisely, our goal is to find the set of locations which can be visited within the least amount of time and from which the entire search space can be observed. We adapt our previous exploration algorithm to generate not a single best view, but a set of multiple high utility views and their observed volumes. These volumes are partitioned into a discrete set of parts that has to be covered by a view sequence with minimal cost. Obviously, this is an intractable problem since it contains the *set cover problem* [12] and the *traveling salesman problem* [13] as subproblems. Therefore, we are considering several *greedy* and anytime search-based variants of our core algorithm and compare them with respect to plan computation time and plan execution time.

II. FORMAL PROBLEM DESCRIPTION

In this section coverage search is formally described. We first describe the model of the searcher and then the structure of the search space. Then we formulate the search problem based on these two definitions.

We consider a mobile robot platform, the searcher, equipped with a 3d sensor in a bounded 3d environment $\mathcal{E} \subset \mathbb{R}^3$. The 3d sensor generates at each sensing cycle a view. 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 $\mathbf{x} \in X \cong \mathbb{R}^3 \times \mathbb{RP}^3$ [2], 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 (position) and $(\phi, \theta, \psi)^T$ the rotational part (orientation) as Euler angles. In general, the possible sensor states, and hence the views that can be obtained, depend on the collision-free configurations for the searcher $q \in \mathcal{C}_{free} \subset \mathcal{C}$. We make no assumptions regarding $\mathcal{C}$ and $\mathcal{C}_{free}$ other than that we have a function $IK : X \to \{0, 1\}$ with $IK(\mathbf{x}) = 1$ if there is a valid path in $\mathcal{C}_{free}$ 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 $X_{reach} := \{\mathbf{x} | IK(\mathbf{x}) = 1, \mathbf{x} \in X\}$. [2]

We furthermore assume the existence of a cost function $cost : \mathcal{C}_{free} \times X_{reach} \to \mathbb{R}^+$ that returns the travel time when moving the robot from one configuration to another that places the sensor in a desired state. Note that for most applications we can conflate $\mathcal{C}_{free}$ 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 slight abuse of notation we can then write $cost(\mathbf{x_i}, \mathbf{x_j})$. This is now simply the cost of moving from one sensor state to another.

The goal of the search is to cover every point in a given search set $\mathcal{S} \subset \mathcal{E}$. For every sensor state let the detection set $D(\mathbf{x}) \subset \mathcal{S}$ be the set of points in $\mathcal{S}$ visible from $\mathbf{x}$. [3] The coverage problem is to visit a sequence $\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \ldots$ of sensor states until the entire search space $\mathcal{S}$ has been seen, i.e., $\bigcup_{i=1}^{m} D(\mathbf{x}_i) = \mathcal{S}$. In addition, the total time needed to visit all sensor states $\sum_{i=1}^{m-1} cost(\mathbf{x_i}, \mathbf{x_{i+1}})$ has to be minimized.

III. COMPUTING CANDIDATE VIEWS

A. Generating Candidate Views

We now describe how to find good sensor states from X_{reach} by computing a utility function $util : \mathcal{E} \to \mathbb{R}^+$ that identifies good $3d$ poses, ignoring the orientation for now. As we have already shown experimentally, in the context of pursuit evasion problems [10], an efficient sampling-based heuristic can significantly 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 $\mathcal{E}$ will serve as a basis for the coverage search methods described in Section IV.

The representation of $\mathcal{E}$ is given in form of an efficient hierarchical 3d grid structure, known as *OctoMap* [15]. Therein our 3d search region $\mathcal{S}$ is tessellated into equally sized cubes. The minimum size of the cubes is typically chosen according to the size of the target that is to be searched. 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 Algorithm 1. First, for every $s \in \mathcal{S}$ we sample k_{max} vectors starting at s and going towards a random position in $pos(X_{reach})$, sampled using $getRandom(.)$. Here $pos(.)$ returns the position

[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 simply-connected.

[2] Reachable states can be precomputed for efficient access during the search using capability maps [14].

[3] Note that there is a subtle formal difference between $D(\mathbf{x})$ and a view at $\mathbf{x}$, i.e. $D(\mathbf{x})$ is the entire 3d volume and a view a discretized subset of it and in addition $D(\mathbf{x})$ is restricted to $\mathcal{S}$. Yet, in colloquial terms, we can think of a detection set as a view.

of a state, simply ignoring its orientation. Second, for each $\langle s, dir\rangle \in \mathcal{V}$ we compute, 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, every cell in $\mathcal{GC}$ has its utility value incremented.

Algorithm 1 Construct $util$

```
1: procedure FINDGOODVIEWS(S)
2:     V ← ∅
3:     // Sample random vectors from S into pos(X_reach)
4:     for all s ∈ S do
5:         k ← k_max
6:         while k ≠ 0 do
7:             x ← getRandom(X_reach)
8:             dir = normalize(pos(x) − s)
9:             V ← V ∪ ⟨s, dir⟩
10:            k ← k − 1
11:        end while
12:    end for
13:    // Accumulate utilities in E
14:    for all v = ⟨s, dir⟩ ∈ V do
15:        GC ← getGridCells(s, dir, s_r)
16:        for all gc ∈ GC do
17:            util(gc) ← util(gc) + 1
18:        end for
19:    end for
20: end procedure
```

We now obtain our set of sampled sensor states $\tilde{X}$, from which large parts of $\mathcal{S}$ are visible, as follows. Grid cells are sorted by $util$ and in descending order turned into sensor states by sampling valid sensor orientations, i.e. for every $(x, y, z)^T \in pos(X_{reach})$ we sample one orientation $(\phi, \theta, \psi)^T$ so that $\mathbf{x} = (x, y, z, \phi, \theta, \psi) \in \mathbf{X}$. Then we compute $U(\mathbf{x}) := |D(\mathbf{x})|$, the actual utility of the sensor state, computed by raytracing the sensors field of view. If $U(\mathbf{x}) \geq \epsilon$, then we add $\mathbf{x}$ to $\tilde{X}$. Once $N = |\tilde{X}|$, for a given N, we stop adding to $\tilde{X}$.

To guarantee complete coverage of $\mathcal{S}$ one could continue to extend $util$ by sampling and incrementally adding more views until $\mathcal{S}$ is covered, similar as in, e.g., the work by Kleiner et al. [10]. However, this relies on the fact that every part of $\mathcal{S}$ can be seen by some $\mathbf{x} \in X_{reach}$ – a property required for the problem to be solvable that yet is violated in many practical applications. There are no assumptions that the environment data was recorded with the robot used for the search and thus the environment $\mathcal{E}$ can cover arbitrary non-reachable space. Determining that a part of the environment cannot be seen from X_{reach} is not easily computable and thus demanding $\mathcal{S} \subset \mathcal{E}$ to be chosen in this way is not acceptable for practical applications. Therefore we make the assumption that the number of views chosen by the user does provide the intended coverage.

B. Partition Induced by the Selected Views

Based on the sampled $\tilde{X} \subset X_{reach}$, which should contain a significantly smaller number of high-utility sensor states we now seek to determine an even further smaller set that gives us a sequence of sensor states $\{\mathbf{x}_1, \ldots, \mathbf{x}_m\} \subset \tilde{X}$ whose detection sets cover all of $\mathcal{S}$, i.e. $\bigcup_{i=1}^{m} D(\mathbf{x}_i) = \mathcal{S}$. Rather than computing $\bigcup_{i=1}^{m} D(\mathbf{x}_i)$ for different sequences, which is computationally expensive, we compute a minimal partition of the search space $\mathcal{S}$ induced by the detection sets for a given set of sensor states Q (in our case $\tilde{X}$).

Definition 1 ***Minimal partition of a set of sensor states*** Given any $Q \subseteq X_{reach}$ let $P(Q)$ be a partition of $\mathcal{S}$ minimal for Q defined as follows:

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

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

Algorithm 2 initializes $P(Q)$ to the trivial partition of $\{S\}$. 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 $P(\mathbf{x})$ 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 $P(\mathbf{x})$.

IV. COVERAGE SEARCH

We now utilize $P(\tilde{X})$, as well as the corresponding mappings $P(\mathbf{x})$ and $Views(A)$, to construct a sequence $\{\mathbf{x}_1, \ldots, \mathbf{x}_m\} \subseteq \tilde{X}$ that covers $\mathcal{S}$ and has a short travel cost. Since we now consider the sequence of sensor states, also called sensor views, we define a new sequential utility function U_i that reduces the original utility U by the number of points in $\mathcal{S}$ that have been seen previously in the sequence:

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

[4] Notice that $\left(\mathcal{S} \setminus \bigcup_{\mathbf{x} \in Q} D(\mathbf{x})\right) \in \mathcal{P}(Q)$ is a part of the partition for all Q that do 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

$P(Q) \leftarrow \{\mathcal{S}\}$
$Views(\mathcal{S}) \leftarrow Q$
for all $\mathbf{x} \in Q$ **do**
  $P(\mathbf{x}) \leftarrow P(Q)$
end for
for all $\mathbf{x} \in Q$ **do**
  for all $A \in P(\mathbf{x})$ **do**
    $A_{in} \leftarrow A \cap D(\mathbf{x})$
    $A_{out} \leftarrow A \cap (\mathcal{S} \setminus D(\mathbf{x}))$
    if $A_{in} = \emptyset \vee A_{out} = \emptyset$ **then**
      continue *// Condition 4. holds.*
    end if
    $P(Q) \leftarrow P(Q) \setminus \{A\} \cup \{A_{in}, A_{out}\}$
    $P(\mathbf{x}) \leftarrow P(\mathbf{x}) \setminus \{A\} \cup \{A_{in}\}$
    for all $\mathbf{x}' \in Views(A) \setminus \{\mathbf{x}\}$ **do**
      $P(\mathbf{x}') \leftarrow P(\mathbf{x}') \setminus \{A\} \cup \{A_{in}, A_{out}\}$
    end for
    $Views(A_{in}) \leftarrow Views(A)$
    $Views(A_{out}) \leftarrow Views(A) \setminus \{\mathbf{x}\}$
  end for
end for
return $P(Q)$

A. Greedy-Solutions

First, we present a selection of greedy algorithms. All greedy strategies compute the coverage sequence by incrementally selecting $\mathbf{x} \in \tilde{X}$ until all parts in $P(\tilde{X})$ are covered. Let UC be the uncovered parts initialized as $UC \leftarrow P(Q)$. For every $i = 1, \ldots, m$ we select a $\mathbf{x}_i$ and update $UC \leftarrow UC \setminus P(\mathbf{x}_i)$. The algorithm terminates when $UC = \emptyset$.

1) Next Best View: As a base line, we use an incremental next highest sequential utility algorithm. At every step i, this strategy chooses a $\mathbf{x}_i$ with maximum utility $U_i(\mathbf{x}_i)$.

2) Cost-based Next Best View: The second alternative is to be greedy with regard to cost in addition to utility, i.e. one that chooses a $\mathbf{x}_i$ that maximizes $U_i(\mathbf{x}_i)/cost(\mathbf{x}_{i-1}, \mathbf{x}_i)$. The idea is to prefer high utility views that are easily reachable. The first view is chosen to be the one with maximum utility U.

3) Greedy Cost: A similar strategy selects a $\mathbf{x}_i$ that only minimizes the cost from the last view $cost(\mathbf{x}_{i-1}, \mathbf{x}_i)$. Again, the first view is chosen be the one with maximum utility U.

B. Planning Solutions

Greedy algorithms provide simple and reasonably fast solutions. However, those solutions are usually sub-optimal. We formulate the problem of finding a coverage sequence for a set of views $\tilde{X}$ as a classical planning problem in order to find the optimal and hence lowest cost coverage sequence.

1) Complete Planning: We model the task in the commonly used Planning Domain Definition Language (PDDL) [16]. 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 formulae 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 view 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. The term `[costSearch ?v]` states that the cost of the action is determined by an external function that is integrated via a modular interface [17]. The module calls the *cost* function defined in Sec. II. A detailed description of this interface is available in our previous work [18]. The `forall` statement defines a conditional effect that sets `covered` to true for all view parts that are in $P(\mathbf{x})$.

Finally, as the goal formula we specify:

```
(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 the action costs, a shortest plan found by the planner also constitutes a minimum-cost coverage sequence. We use a variant of the planner Temporal Fast Downward (TFD) [18] to solve all planning tasks in this paper.

2) Set Cover/Traveling Salesman: As finding optimal solutions to this problem might prove infeasible, we present a sub-optimal decomposition of the problem by solving a set cover and subsequent traveling salesman problem.

First, we find a minimal set of views that covers all parts of the partition. This problem is the classical *set cover* problem. Find a minimum cardinality subset $Q_C \subseteq \tilde{X}$, where all parts are covered, i.e. $\bigcup_{\mathbf{x}_j \in Q_C} P(\mathbf{x}_j) = \bigcup_{\mathbf{x}_i \in \tilde{X}} P(\mathbf{x}_i)$.

The geometric nature of our data enables us to reduce the problem. Many views cover a unique part of the search space that is not covered by any other view and thus must be included in any set cover solution. We call those views *necessary*, and 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: Action costs 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.

We have now determined a minimum cardinality subset of views that covers the search space. Thus the only remaining problem is to find an optimal cost 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 can easily apply this formulation to the Traveling Salesman Problem by changing the goal formula to:

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

This requires all views to be visited and thus an optimal cost plan to this problem results in a minimum-cost 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 alternatively investigate applying the LKH solver [19], an efficient implementation of the Lin-Kernighan heuristic to solve the TSP.

When we use the TFD planner the solve the TSP in the decomposed formulation, we denote this as Set Cover/TSP (TFD). If we use the LKH solver, Set Cover/TSP (LKH) is given. TFD is always used to solve the set cover problem.

V. EVALUATION

We evaluated our algorithms on various different real-world data sets. The first data set contains a 3d scan taken in our lab with a range sensor in the local environment of the robot that needs to be covered by a sensor with a smaller field of view like an IR camera. Setting Lab 1 only considers manipulator movements, while Lab 2 and all other experiments include manipulation and navigation. The next two data sets demonstrate larger indoor environments. One taken in building 78 at the university of Freiburg consisting of two rooms separated by a door. The other data set is the NIST rescue arena at Jacobs university in Bremen. The last data set is a large scale outdoor model of the computer science campus at the university of Freiburg. Octrees for the indoor data sets are generated with 5 cm resolution, the outdoor data set uses 20 cm resolution. Examples of the data sets can be seen in Figure 2. In all experiments the set of cells $\mathcal{S}$ to be examined are vertical structures of the map thus aiming for a complete coverage besides floors and ceilings. The utilized robot mode comprises a ground robot with a 6-DOF manipulator and one meter reach. The sensor model is a camera with 60 degrees horizontal and 40 degrees vertical field of view. For the indoor data sets a maximum range of five meters was used, the outdoor data set was searched with a 35 meter maximum range.

A. Efficient cost computation

The computation of an optimal sequence for visiting viewpoint locations requires numerous computations of costs for traveling between different locations on the map. We base our travel cost planner on value iteration, a popular dynamic programming algorithm frequently used for robot planning [20]. As shown by Figure 3 the planner takes as input a classified elevation map in which important structural elements such as stairs and ramps are discriminated. Value Iteration computes then efficiently for each grid cell (e_x, e_y) on the elevation map a cost estimate for reaching a goal cell (g_x, g_y). These costs are composed of travel distance as well as costs for overcoming different types of terrain indicated by the classification. The

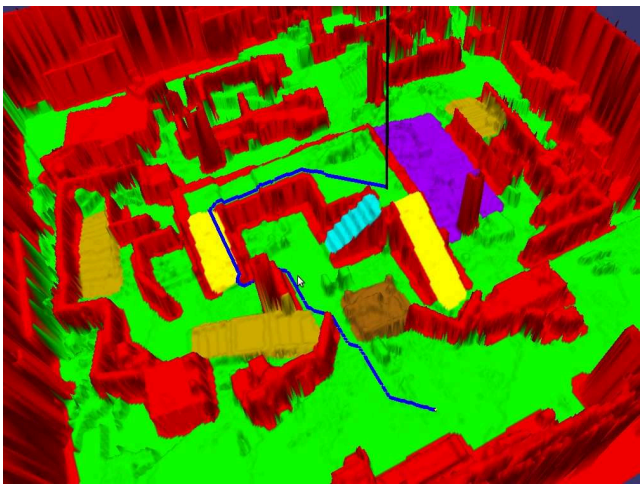

Fig. 3. Computation of plan costs on classified elevation maps (map of the arena shown by Figure 1). Showing the generated plan between two locations (blue line), traversable (green) and non-traversable (red) terrain. The classification represents structural elements such as stairs (magenta), ramps (yellow), and random step fields (violet).

resulting value function is then used by an $A*$ planner as the

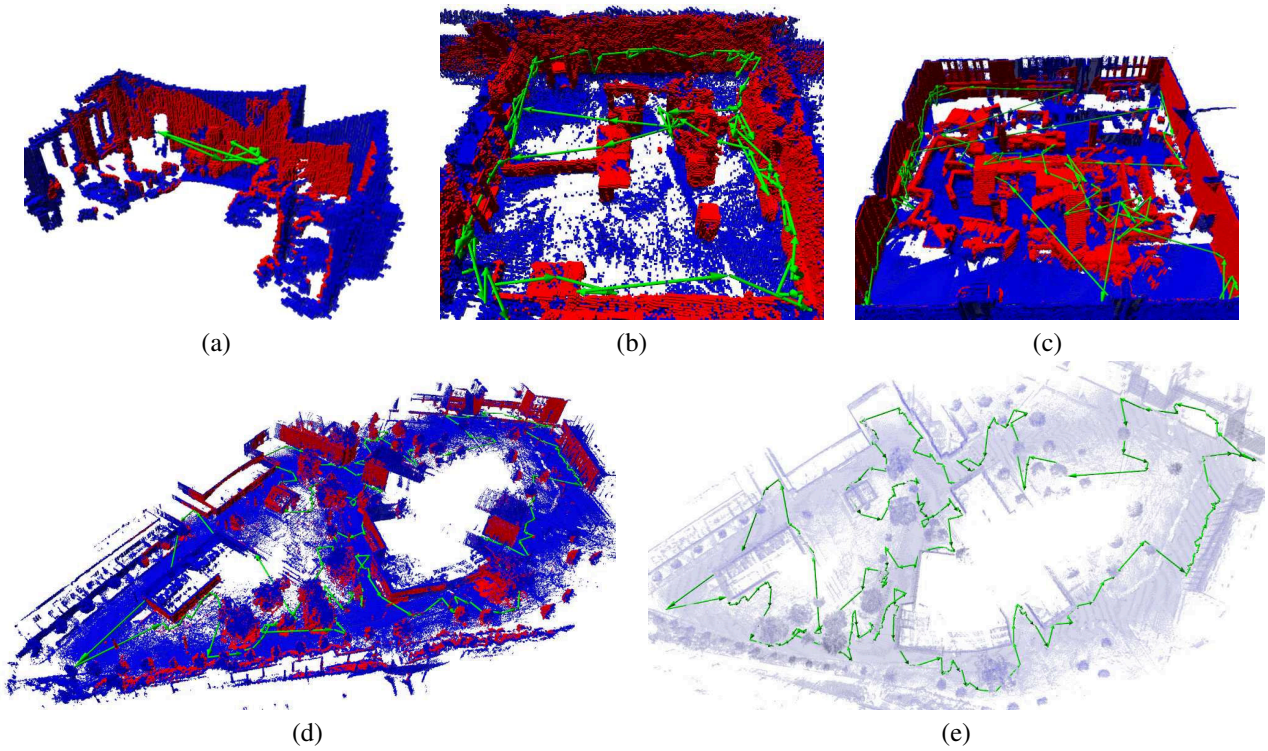

Fig. 2. This figure shows the data sets used for evaluation: The 3d scan in our lab (a), building 78 (b), the rescue arena (c) and the computer science campus of Freiburg university (d). For a better visualization of the plan in the campus a top-down view is given (e). The plans (green) are generated with the Set Cover/TSP (LKH) method. Occupied cells are displayed blue, covered cells red.

heuristic for finding shortest paths (and their costs) on the map. Besides these travel costs between robot base poses we also consider the cost to move the manipulator from one view configuration to the next based on the maximum angular displacement of any joint. The expected combined execution time defines the cost between two sensor states.

B. Coverage Search

This experiment illustrates the full application of the coverage search algorithm. We are reporting computation times and the costs of the best plans found by the algorithms on each of the maps. As the candidate generation involves randomization, we ran the algorithm ten times for each scenario and report mean values with variance. The minimum utility ϵ to accept a view as well as the number of views that need to be generated was chosen with respect to the average expected utility that depends on the sensor model and the environment. Therefore, for the outdoor environment with a 35 meter sensor, this minimum needs to be significantly higher than the one chosen for an indoor environment with limited field of view. Table I shows mainly computation times for the candidate generation and the partitioning. We also give the reduction factor achieved by the partition. This is computed as the ratio of covered cells to parts in the partition. The motivation for this is that without computing partitions, coverage computations would be performed by considering individual cells instead of larger chunks in partitions. The factor indicates the reduction of the problem size due to the partitioning.

Based on the results of the candidate generation, we ran the search algorithms presented in Section IV. Table II shows costs (execution time) and measured runtime (solution computation time) from these experiments. The TFD planner that was used during the experiments supports both numerical values and temporal actions. As we are only interested in numerical computations without requiring concurrent actions, this feature has been disabled during the experiments. All greedy algorithms were run until a solution was found. The anytime planner was set to stop after twice the time it took to find the first valid plan. The LKH solver was executed with the default parameters supplied by the software.

For the greedy variants, we observe that the base line of incremental next best view performs worst. This is not surprising as it doesn't consider costs. From the other two variants greedy-cost shows better performance, although it does not consider utility. This is possible, because the utility is implicitly contained in the selected views $\tilde{X}$ as the candidate generation chooses good views.

For the search based variants, we generally see better results at the cost of increased computation time. The results for the small Lab scenario are similar. With increasing problem size the decomposed variant becomes superior, especially when using a specialized TSP solver. Although plan costs computed for the NIST arena do not significantly differ, needed computation times are shorter. The planner operates on a grounded representation and thus the very large problems become infeasible for the planning based variants. In those

Scenario	Lab 1	Lab 2	Bldg. 78	Arena	Campus
Candidate Views [s]	2.12 ± 0.20	3.01 ± 0.45	30.80 ± 2.47	78.96 ± 4.85	201.90 ± 10.09
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	15	15	100	142	280

TABLE I
THIS TABLE SHOWS COMPUTATION TIMES FOR THE CANDIDATE GENERATION. IN ADDITION THE REDUCTION ACHIEVED BY THE PARTITION IS GIVEN.

Scenario		Lab 1	Lab 2	Bldg. 78	Arena	Campus
Next Best View	cost [s]	29.04 ± 5.55	66.45 ± 11.17	1266.34 ± 61.29	4178.80 ± 220.62	38831.17 ± 1788.91
	time [s]	0.03 ± 0.00	0.11 ± 0.02	5.29 ± 0.57	12.68 ± 1.39	113.43 ± 8.07
Cost-Based Next Best View	cost [s]	23.69 ± 3.27	52.09 ± 8.10	532.63 ± 24.73	1508.76 ± 190.85	8703.67 ± 519.57
	time [s]	0.04 ± 0.00	0.13 ± 0.02	18.77 ± 1.43	72.78 ± 7.63	765.58 ± 88.94
Greedy-Cost	cost [s]	24.48 ± 2.87	44.04 ± 6.55	337.88 ± 11.58	891.97 ± 93.01	4950.81 ± 261.66
	time [s]	0.04 ± 0.01	0.12 ± 0.03	17.54 ± 1.12	70.25 ± 8.18	775.65 ± 54.19
Set Cover/TSP (TFD)	cost [s]	16.33 ± 2.61	39.41 ± 4.38	331.92 ± 11.55	845.44 ± 98.74	-
	time [s]	1.23 ± 0.27	1.18 ± 0.28	95.12 ± 5.03	407.88 ± 110.13	-
Set Cover/TSP (LKH)	cost [s]	16.60 ± 2.43	38.39 ± 6.67	289.07 ± 7.16	870.73 ± 143.04	4056.73 ± 105.01
	time [s]	0.63 ± 0.22	0.53 ± 0.05	46.22 ± 3.12	151.82 ± 11.78	1215.75 ± 145.80
Complete Planning	cost [s]	22.07 ± 2.83	40.53 ± 5.23	333.06 ± 12.19	867.36 ± 95.30	-
	time [s]	1.68 ± 0.24	2.93 ± 0.86	1141.26 ± 206.04	1722.82 ± 321.96	-

TABLE II
THIS TABLE SHOWS RESULTS FOR THE COVERAGE SEARCH ALGORITHMS. MISSING ENTRIES FOR THE PLANNING BASED SOLUTION ON CAMPUS ARE DUE TO LIMITED MEMORY.

cases the system runs out of memory not in the search, but already during the grounding phase of the planner. Although in contrast to the decomposed approach the complete planning formulation can provide the optimal solution, this is unlikely to be reached within practical time limits. Instead, the more complex formulation takes more time to search and thus, we see higher runtimes.

C. Anytime Behavior and Optimal Solutions

This experiment is designed to show the impact of the decomposition as a separate set cover and TSP in comparison to the complete approach. We investigate, if the complete formulation enables us to find better plans in real data and how both approaches perform in an anytime setting.

We used the first run of the smallest map and execute *complete planning* and *Set Cover/TSP (TFD)* until the state space was completely explored. Using the TFD planner in both instances allows us to find and prove optimal solutions. Computing optimal solutions for the larger maps was not feasible.

We computed time and cost for the first and best plan that was found and also the time that the full run took until the best solution was proven optimal. The best of the greedy algorithms in this instance *greedy cost* is given as a reference. Table III shows the results of this experiment.

The first solution is quickly found by all algorithms. For the planning based algorithms this is also already better than the greedy cost solution, although it also took longer to compute. We can see that the best solution that the decomposed approach returns is worse with 16.83 than the optimal solution of the complete formulation with 15.78. This shows that we do indeed loose the optimal solution by our decomposition. The best plan found by *Set Cover/TSP (TFD)* is only 6.6% longer than the optimal one, but is computed significantly faster. Our data also shows that it takes for the complete formulation 4996 seconds to compute a plan that is better than the best plan of the decomposition. The gross difference in computation time can be explained by the fact that we have an NP-hard problem and due to the set cover decomposition the TSP only contained 12 views instead of 15 for the complete formulation. Even only considering permutations that is a factor of $15!/12! = 2730$. This means that in practice, the decomposed solution will produce better results within reasonable time limits.

Algorithm		first plan	best plan	full run
Greedy Cost	cost [s]	22.22	22.22	
	time [s]	0.03	0.03	
Set Cover/TSP (TFD)	cost [s]	19.34	16.83	
	time [s]	0.15	7.76	100.09
Complete Planning	cost [s]	20.79	15.78	
	time [s]	0.16	12722.88	33281.29

TABLE III
COMPARISON OF BEST RESULTS THAT THE ALGORITHMS ARE CAPABLE OF. TIME AND COST 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.

VI. CONCLUSIONS

We considered the problem of identifying and planning efficient view sequences for covering complex environments represented in 3d. For that purpose we compared several variants of our algorithm empirically for finding a good trade-off between schedule computation and execution time. Our results indicate that despite the intractability of the problem, efficient coverage in 3d is feasible.

Small size problems such as incremental vicinity exploration by a robot were solved close to real-time and thus can directly be deployed in real-world applications. For larger problems, the *greedy-cost* variant provided solutions that were competitive with the ones based on search. Although the computation times for the *greedy-cost* variant were smaller, advanced solutions are still the better choice since they result in lower costs and thus lower execution times. Among the optimization algorithms considered, the decomposition into a separate set cover and traveling salesman problem turned out to work very well, i.e., producing high-quality solutions in a short amount of time. Using the set cover solution with a specific TSP solver finally appears to be the superior method.

ACKNOWLEDGMENT

The Freiburg campus dataset was originally recorded by Bastian Steder. Building 78 is courtesy of Jürgen Hess, Felix Endres and Andreas Hertle. Both are available at http://ais.informatik.uni-freiburg.de/projects/datasets/octomap/. The rescue arena dataset was recorded by Dorit Borrmann, Jan Elseberg and Andreas Nüchter from Jacobs University Bremen gGmbH, Germany. The source data is available from http://kos.informatik.uni-osnabrueck.de/3Dscans/.

REFERENCES

[1] A. Jacoff, B. Weiss, and E. Messina, "Evolution of a performance metric for urban search and rescue robots," in *Performance Metrics for Intelligent Systems*, 2003.

[2] S. M. LaValle, *Planning Algorithms*. Cambridge, U.K.: Cambridge University Press, 2006, available at http://planning.cs.uiuc.edu/.

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

[4] D. Golovin and A. Krause, "Adaptive submodularity: Theory and applications in active learning and stochastic optimization," *Journal of Artificial Intelligence Research*, vol. 42, pp. 427–486, 2011.

[5] T. Kollar and N. Roy, "Efficient optimization of information-theoretic exploration in slam," in *Proceedings of the National Conference of Artificial Intelligence (AAAI 08)*, July 2008.

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

[7] J. Banta, Y. Zhieng, X. Wang, G. Zhang, M. Smith, and M. Abidi, "A "Best-Next-View" Algorithm for Three-Dimensional Scene Reconstruction Using Range Images," *Proc. SPIE (Intelligent Robots and Computer Vision XIV: Algorithms)*, vol. 2588, 1995.

[8] H. Gonzalez-Banos, E. Mao, J. Latombe, T. Murali, and A. Efrat, "Planning robot motion strategies for efficient model construction," in *Proc. of the 9th International Symposium on Robotics Research (ISRR)*. Springer, Berlin, 2000, pp. 345–352.

[9] T. Shermer, "Recent results in art galleries," *Proceedings of the IEEE*, vol. 80, no. 9, pp. 1384–1399, 1992.

[10] A. Kleiner, A. Kolling, M. Lewis, and K. Sycara, "Hierarchical visibility for guaranteed search in large-scale outdoor terrain," *Journal of Autonomous Agents and Multi-Agent Systems (JAAMAS)*, vol. 26, pp. 1–36, 2013. [Online]. Available: http://dx.doi.org/10.1007/s10458-011-9180-7

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

[12] R. Karp, "Reducibility among Combinatorial Problems," *Complexity of Computer Computations*, 1972.

[13] D. L. Applegate, R. E. Bixby, V. Chvatal, and W. J. Cook, *The traveling salesman problem: a computational study*. Princeton University Press, 2007.

[14] F. Zacharias, C. Borst, and G. Hirzinger, "Capturing robot workspace structure: representing robot capabilities," in *Proceedings of the IEEE International Conference on Intelligent Robots and Systems (IROS2007)*, 2007, pp. 4490–4495.

[15] A. Hornung, K. M. Wurm, M. Bennewitz, C. Stachniss, and W. Burgard, "OctoMap: An efficient probabilistic 3D mapping framework based on octrees," *Autonomous Robots*, 2013. [Online]. Available: http://octomap.github.com

[16] M. Fox and D. Long, "PDDL2.1: an extension to PDDL for expressing temporal planning domains," *Journal of Artificial Intelligence Research (JAIR)*, vol. 20, no. 1, pp. 61–124, 2003.

[17] K. M. Wurm, C. Dornhege, P. Eyerich, C. Stachniss, B. Nebel, and W. Burgard, "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)*, October 2010.

[18] C. Dornhege, P. Eyerich, T. Keller, S. Trüg, M. Brenner, and B. Nebel, "Semantic attachments for domain-independent planning systems," in *Proc. of the Int. Conf. on Automated Planning and Scheduling (ICAPS)*, 2009, pp. 114–121.

[19] K. Helsgaun, "An effective implementation of the lin-kernighan traveling salesman heuristic," *European Journal of Operational Research*, vol. 126, no. 1, pp. 106–130, 2000.

[20] W. Burgard, A. B. Cremers, D. Fox, D. Hähnel, G. Lakemeyer, D. Schulz, W. Steiner, and S. Thrun, "The interactive museum tour-guide robot," in *Proceedings of the National Conference on Artificial Intelligence*. JOHN WILEY & SONS LTD, 1998, pp. 11–18.