Multi-UAV Motion Planning for Guaranteed Search

Andreas Kolling
Dept. of Computer and Information Science
Linköping University
Linköping, Sweden
andreas.kolling@liu.se

Alexander Kleiner
Dept. of Computer and Information Science
Linköping University
Linköping, Sweden
alexander.kleiner@liu.se

ABSTRACT

We consider the problem of detecting all moving and evading targets in 2.5D environments with teams of UAVs. Targets are assumed to be fast and omniscient while UAVs are only equipped with limited range detection sensors and have no prior knowledge about the location of targets. We present an algorithm that, given an elevation map of the environment, computes synchronized trajectories for the UAVs to guarantee the detection of all targets. The approach is based on coordinating the motion of multiple UAVs on sweep lines to clear the environment from contamination, which represents the possibility of an undetected target being located in an area. The goal is to compute trajectories that minimize the number of UAVs needed to execute the guaranteed search. This is achieved by converting 2D strategies, computed for a polygonal representation of the environment, to 2.5D strategies. We present methods for this conversion and consider cost of motion and visibility constraints. Experimental results demonstrate feasibility and scalability of the approach. Experiments are carried out on real and artificial elevation maps and provide the basis for future deployments of large teams of real UAVs for guaranteed search.

Categories and Subject Descriptors

I.2.9 [**Artificial Intelligence**]: Robotics—*Autonomous vehicles*

General Terms

Algorithms, Experimentation, Performance

Keywords

Multi-UAV; guaranteed search; pursuit-evasion; multi-robot

1. INTRODUCTION

Recent years have seen tremendous progress with regard to affordability and capability of a wide range of robotic hardware. Especially unmanned aerial vehicles (UAVs) are becoming more capable and abundant. They have even found their way into the hands of consumers, most notably the AR.Drone [**?**], leading to further reduced costs. This development lets us envision new applications, such as deploying large-scale multi-UAV teams for search and rescue, security, or surveillance. These applications benefit greatly from the mobility and low cost of UAVs. In order to scale to large environments proper coordination of many UAVs becomes paramount, especially when detection guarantees are required.

In this paper, we present a comprehensive and practical solution to the problem of coordinating a guaranteed search with a team of UAVs searching for ground targets in environments represented by elevation maps. This is achieved by coordinating the motion of all UAVs through the environment with a line-based abstraction which aggregates multiple UAVs on a line spanned between obstacles. The environment is then cleared by moving these lines of UAVs in a synchronized manner. The goal is to minimize the number of UAVs needed for clearing the entire environment and to compute the line trajectories. In order to compute these low cost trajectories we present an algorithm that computes the best possible motion through a simply-connected polygonal environment and then adapt the resulting strategies to multiply-connected and 2.5D environments. This adaptation considers visibility issues arising in 2.5D regarding the detection of targets. To determine the cost of covering lines in 2.5D we compute detection sets, i.e. an area around a location of a UAV on which targets are detectable, to locally cover the lines. The goal is to identify the approximate close-to-minimal set of locations needed to cover the entire sweep line. Figure 1 depicts an elevation map generated at the campus of the University of Freiburg and a snapshot during coordinated search by UAVs.

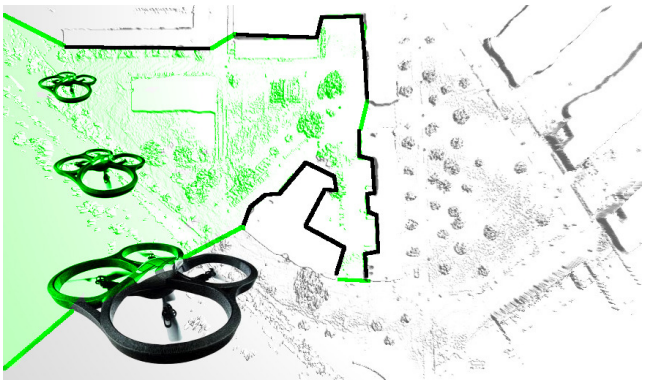

Figure 1: Motivating picture: Coordinated search for evaders by a team of UAVs to guarantee the detection of any intruder.

The primary objective of this paper is to demonstrate how to combine solutions to the combinatorial, geometric, and path planning issues that arise with guaranteed search problems in realistic environments, so that a working, scalable, and efficient system can be built on top of these solutions. Along the way we will introduce new algorithms but leave many challenges open. The remainder of

Appears in: *Proceedings of the 12th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2013), Ito, Jonker, Gini, and Shehory (eds.), May, 6–10, 2013, Saint Paul, Minnesota, USA.*

this paper is organized as follows. In Section 2 related work is discussed. Section 3 introduces the guaranteed search problem. The first part of our approach, concerned with a simplified 2D variant, is described in Section 4. In Section 5 issues arising with complex 2.5D environments are addressed. In Section 6 results from experiments on a variety of maps are presented and we finally conclude with Section 7.

2. RELATED WORK

Guaranteed search, pursuit-evasion and other search problems are at the cross-section between robotics, graph theory, computational geometry and probability theory. Consequently, a wide range of models with different assumptions regarding environments, sensors, motion of robots or motion of targets exist and each contribution usually emphasizes a particular aspect of the problem from theoretical to practical. We shall briefly review a few of the most closely related works. An overview of guaranteed search and pursuit-evasion from a robotics perspective is given in the survey [**?**]. A survey, in form of an annotated bibliography, from the a graph theory perspective can be found in [**?**].

Real robotic systems for large scale guaranteed search with multiple searchers are still rare and we are not aware of such systems with UAVs and 2.5D or 3D environments. Combining solutions to problems arising when using real robots with the more abstract combinatorial and geometric methods has proven to be a considerable challenge. Smaller systems have been demonstrated such as in [**?**]. The largest demonstration to date, with eight searchers in a 2.5D outdoor environment was presented in [**?**], although the searchers were humans equipped with iPads and not actual robots. The work in [**?**] combined solutions to graph-theoretic pursuit-evasion problems [**?**] with a sampling-based analysis of the geometry and visibility of the 2.5D environment. In addition, to execute the search faster a time optimization was considered. In contrast to our work, [**?**] employed a graph-based approach and considered static guarding locations while we consider a line-based approach with moving lines and trajectories. Most contributions emphasize optimizing the number of searchers. A notable exception is [**?**] which deals with the computational complexity of optimizing the time to execute search strategies on graphs. Yet, much further work in this domain remains to be done.

Another issue that did not receive much attention is that of graphs with cycles. Trees are fairly well understood for many of the pursuit-evasion problems [**?**, **?**, **?**] and in some case can be solved optimally in polynomial time. Consequently, practical robotic application that deal with complex environments with cycles turn to heuristics that convert graphs to trees and assume that additional searcher can be used to emulate this conversion in real environments. Such approaches have been used in [**?**, **?**, **?**] amongst others. In our work we will also follow this approach.

Problems involving 2.5D and 3D environments, in conjunction with UAVs are also becoming more popular. In [**?**] and [**?**] UAVs were considered search with a single searcher and for tracking. For our purposes the closest related work in 2.5D is [**?**]. Therein Kleiner et al. consider the efficient computation of so called detection sets that determine on which locations a UAV can see targets in a 3D environment represented by a 2.5D elevation map.

Other related works deal with moving boundaries and lines of robots, usually ground robots, for example the coordination of security sweeps for which market-based methods have been used [**?**] in 2D environments. Another 2D approach, from a control-theoretic perspective, is presented in [**?**]. Therein Durham *et al.* presented a distributed algorithm guaranteeing complete coverage of the frontier between cleared and contaminated areas during expansion [**?**]. Their algorithm can be applied to multiply-connected planar environments which may be non-polygonal, but no minimization of the number of robots is provided. Another approach using sweep lines for clearing an environments by coordinating lines on a Voronoi diagram has been presented by Kolling *et al.* [**?**]. The Voronoi Diagram of the environment induces a surveillance graph for which the weights, i.e. costs in terms of UAVs, of vertices are given by the distances to the three closest and equidistant obstacles. Lines then either wait on a Voronoi edge or move towards a new obstacle that is associated to a Voronoi vertex. This approach was shown in [**?**] to result in suboptimal coordination of lines for some problem instances, which motivated our generalized approach presented here. In contrast to [**?**] we consider more possibilities for the expansion of the lines and provide a more flexible data structure for the analysis of the environment leading to an improved algorithm. In addition we consider the adaptation of line-based strategies to 2.5D environments with complex visibility constraints.

Another closely related paper by Efrat *et al.* [**?**] considers an approach where multiple robots, each with an unlimited range sensor, are arranged in a single movable polygonal chain operating in a simply connected environment. Their algorithm for computing motion strategies runs in $O(n^3)$, an improved version of the algorithms runs in $O(n \log n)$ time [**?**]. Our work in contrast considers UAVs with a limited sensing range, e.g. a downward facing camera, and multiple simultaneous lines in the environment. We further allow the number of lines to vary as the search mission unfolds.

3. PROBLEM DESCRIPTION

In this section we describe our search problem in more detail. We make similar assumptions as most pursuit-evasion problems with regard to target characteristics, namely a worst-case mobile target that can move with unbounded speed, is omniscient, and optimally evasive. For robotic applications these assumptions have the advantage that one can search for targets with unknown properties and still retain a guarantee of detection. From a theoretical perspective we get the advantage that we can represent the possibility of a target being present at a location with the concept of *contamination*. This then turns the detection of all worst-case targets into the problem of removing all contamination and interesting formal questions with regard to re-contamination can be addressed [**?**].

Our assumptions about the environments and searchers differ from much of the prior work, with the exception of [**?**] from which we adapted our models. We assume a 2.5D environment $\mathcal{E}$ represented by an elevation map $h : H \to \mathbb{R}^+$. The domain H is continuous and $H \subset \mathbb{R}^2$ which, for all practical purposes, we will approximate with a 2D grid map that contains in each discrete grid cell the corresponding height value. Let us write $\mathcal{E} \subset H$ and assume $\mathcal{E}$ is connected and every point in $\mathcal{E}$ is reachable by targets from any other point in $\mathcal{E}$. Targets are required to move on continuous paths in $\mathcal{E}$, albeit as noted above with unbounded speed. We define a path to be a continuous function $\pi : [0, 1] \to \mathcal{E}$. Hence, targets can be thought of as moving on the ground level of the elevation map. The searchers are UAVs flying at a specified height h_r above the ground. The sensor model is a limited range disc of a given sensing radius, representing a downward facing camera. Let $\mathcal{R} = \{R_1, R_2, \ldots, R_n\}$ represent the n UAVs in $\mathcal{E}$. Every R_i is described by a path $\pi_i : [0, 1] \to H$, i.e. locations $\pi_i(t) \in H$ at time t, and an associated height h_r above $h(\pi_i(t))$[1].

For every UAV R_i let $D(\pi_i(t)) \subset \mathcal{E}$ be its sensor footprint, i.e. the set of locations in $\mathcal{E}$ on which the UAV can detect targets. In general, $D(\pi_i(t))$ depends on the sensor model, height of the UAV

[1]We assume all UAVs fly at the same height above the ground.

h_r relative to $h(\pi_i(t))$ and height of targets h_t. In this paper we consider a limited range, three-dimensional, and omni-directional sensor with a sensing range of s_r, the same model as described in [**?**]. A target at $p' \in H$ is hence detectable by a UAV R_i located at $\pi_i(t)$ at time t if at least one point on the line segment from $\{p', h(p')\}$ to $\{p', h(p') + h_t\}$ embedded in $\mathbb{R}^3$ is visible from $\{p_i, h(p_i) + h_r\}$ at distance s_r. Let $D(t) := \bigcup_{i=1,\ldots,n} D(\pi_i(t)))$ be the joint sensor footprint of all UAVs at time t. We can now define the concept of cleared and contaminated points in $\mathcal{E}$.

DEFINITION 1 (CLEARED AND CONTAMINATED POINTS). *A point $x \in \mathcal{E}$ is cleared at time t if $x \in D(t)$. Furthermore, a point x cleared at time t is also cleared at time $t' > t$ if $\nexists$ a path $\pi : [0,1] \to \mathcal{E}$ from x to a contaminated point $y \in \mathcal{E}$ at any time $t'' \in [t, t']$ that does not intersect $D(t)$. If x is not clear it is called contaminated. At $t = 0$ all points in $\mathcal{E} \setminus (D(0))$ are contaminated.*

The problem now is to find the paths π_i and the minimum number of UAVs n, so that there exists for each UAV R_i a path $\pi_i : [0,1] \to \mathcal{E}$ such that at $t = 1$ an initially contaminated environment $\mathcal{E}$ is *cleared*. Since many of the pursuit-evasion problems in simpler environments, such as graphs or even some variants on trees, are already NP-hard we expect this problem to computationally difficult and are not going to be concerned with assessing its complexity. The remainder of the paper focuses on a multi-step approach to use simplified versions of the problem, address them separately, and then successively combine them to finally lead to paths π_i. These steps involve a coordination of motion on lines in 2D, the generation of coverage location for lines in 2.5D, and the computation of trajectories from these locations.

4. 2D COORDINATION

To tackle the problem of computing paths $\pi_1, \ldots, \pi_m$ that clear all of $\mathcal{E}$ we first take a more abstract and simplified perspective in 2D. Instead of individual UAVs we consider the joint sensor footprints of multiple UAVs and represent these with sweep lines that move through the environment. This representation of multiple UAVs as sweep lines enables us to coordinate their paths in order to minimize the number of UAVs needed for clearing the entire environment. For this, we assume that we are given a representation of $\mathcal{E}$ in form of a simply-connected and simple polygon, $\mathcal{P} = \{v_1, \ldots, v_n\}$, with n vertices and edges, written $e_i = [v_i, v_{i+1}]$, $i = 1, \ldots, n$. The edges e_i of the polygon are then the obstacles of the environment and we also interpret them, with slight abuse of notation, as straight line segments $e_i \subset \mathbb{R}^2$. In Section 5 we discuss how to deal with multiply-connected environments and how to polygonize the elevation map h introduced earlier. Throughout this section the obstacle indices, i.e. the polygon edges, are assumed to be circular, i.e. we identify $i + n$ with i.

The basic idea of coordinating the motion of UAVs on sweep lines is quite simple. A sweep line can be spanned between any two obstacles and it has an associated cost that represents the number of UAVs needed to cover the area of the line with their sensors. The goal is to coordinate the motion of multiple such lines moving through the environment while minimizing the maximum cost that occurs at any point in time. The movement of these lines then clears the entire environment and every line separates the contaminated from the cleared area. A more formal definition of the problem of finding a motion schedule for many sweep lines in a 2D environment was formulated in [**?**] and we shall present a shortened and less formal introduction.

Initially, the interior of P is contaminated and one sweep line has to start at some point on the obstacle boundary. Its initial motion then clears the contamination in the area it sweeps over. This is illustrated in Fig. 2. The role of these lines is either to continue sweeping the environment and expand the cleared area or to block contamination from entering the already cleared parts. Let $\mathcal{C}$ be the cleared interior of P, then the intersection of the boundary of $\mathcal{C}$ with interior of P has to have sweep lines on it that block contamination, i.e. $\delta\mathcal{C} \cap interior(P)$ is covered by lines with UAVs on them. Otherwise $\mathcal{C}$ will get recontaminated. The problem is to find lines and move them through the environment to clear it at the lowest cost in terms of UAVs without allowing the recontamination of any point in $\mathcal{C}$. For expanding the cleared area $\mathcal{C}$ and for blocking recontamination only sweep lines that are spanned between two obstacles are useful. Note that we also consider sweep lines that are composed of multiple straight line segments. Let us write $l_{i,j}$ for a sweep line between obstacles e_i and e_j. The cost of this sweep line is given by $c(l_{i,j})$ and represents the number of UAVs needed to cover the line. This cost function can defined appropriately for a given environment or type of robot, but for our purpose we assume that $c(l_{i,j}) = \frac{|l_{i,j}|}{2 \cdot s_r}$, where s_r is the sensor range of the UAVs and $|l_{i,j}|$ is the length of the line (Euclidean).

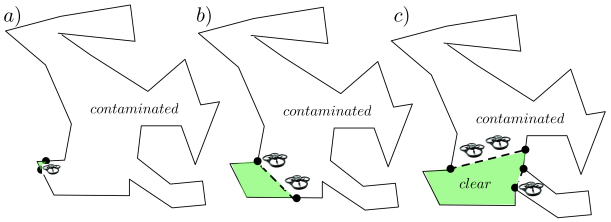

Figure 2: The beginning of a sweep through an environment using lines. In a) the sweep starts with one line, extends by moving the line through the interior and then splits into two lines, each guarding two separate boundaries of the cleared area.

Obviously, we need to find short lines in P between the obstacles e_i and e_j. For this we assume that we have two basic functions. One that computes the shortest line between two edges e_i, e_j in P, called $shortest(e_i, e_j)$, and one that computes the shortest line between a point $p \in P$ and an edge e_i in P, called $shortest(e_i, p)$. In a polygon, these shortest lines are simple to compute. We refer the reader to [**?**] Chapter 6.2.4 for details on planning shortest paths between two points in polygons.

To keep track of $\mathcal{C}$ we call an edge e_i of P cleared if $\mathcal{C} \cap e_i \neq \emptyset$, i.e. if any point on e_i is cleared. Now all cleared e_i that have an adjacent obstacle that is still contaminated must have a sweep line starting on it to block contamination. Let e_i and e_j be two segments which have a blocking sweep line between them and let $l^b(i,j) = shortest(e_i, e_j)$ denote the lowest cost line that blocks contamination.

To clear more of P and expand $\mathcal{C}$ one of the blocking sweep lines on the boundary of $\mathcal{C}$ has to be moved forward and at some point clear a new obstacle. Let e_o be the first obstacle that is cleared in this process and let $l^b_{i,j}$ be the line moved forward to clear e_o. The lowest possible cost of this process can be computed by considering the cost of splitting the line $l^b_{i,j}$ onto e_o. Fig. 3 shows such a split. This lowest cost is given by:

$$c(o|i,j) := min_{p \in e_o}\{|shortest(e_i, p)| + |shortest(e_j, p)|\} \quad (1)$$

If the shortest sweep line is a straight line, then this can be rewritten as $min_{p \in e_o}\{|p_i - p| + |p_j - p|\}$ with p_i, p_j, p subject to the constraints given by the linear equations of the lines through their respective edge segment and the bounds on their coordinates. While the constraints are linear the equation itself is non-linear without an

easy analytical solution. For our purposes we assume an oracle for Eq. 1 that simply returns the point p. A sample implementation for such an oracle is given in a later section.

Once a line is split on a new obstacle, as illustrated in Fig. 3, we get blocking lines between e_o and e_i and e_o and e_j. The cost to maintain these lines can be reduced by moving them to a locally shorter line, namely $l^b_{i,o}$ and $l^b_{j,o}$, as shown in Fig. 3. Notice that if the index o is adjacent to either j, i, or both, then the length of $l^b_{i,o}$, $l^b_{j,o}$, or both is zero. These local operations, to find

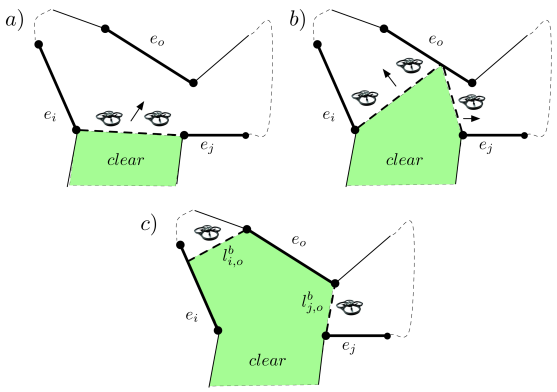

Figure 3: A line between e_i and e_j in a) splitting on e_o in b) and moving towards the blocking lines $l^b(i,k)$ and $l^b(j,k)$ in c).

a low cost blocking line or a low cost split, are relatively simple. The difficulty lies in coordinating many such operations in a larger environment and determining which line should split onto which obstacles and in which sequence. Every choice of split further determines the possible choices and costs for future splits. A sequence of all obstacle indices $o_1, \ldots, o_n$ completely determines when and where to do a split. The first splits on o_1 and o_2 simply set up the first line, possibly a zero length line if o_1 and o_2 are adjacent. The s-th split, $s > 2$, is then given by o_s. The line that is split on o_s is the blocking line between two indices from $o_1, \ldots, o_{s-1}$. These indices are next smaller and next larger index to o_s in $o_1, \ldots, o_{s-1}$, i.e. $o_l = \text{argmax}_{o \in \{o_1,\ldots,o_{s-1}\}}\{o < o_s\}$ and $o_r = \text{argmin}_{o \in \{o_1,\ldots,o_{s-1}\}}\{o > o_s\}$. So the line $l^b(o_l, o_r)$ is split on o_s while all other lines already present in P wait. The overall cost can be calculated by summing up all other blocking lines and the cost of the split onto o_s. A different obstacle sequence $o_1, \ldots, o_n$ will lead to a different sequence of splits, hence a different coordination of the motion of lines and potentially different cost. In the next section we show how to compute obstacle sequence without having to consider all possible sequences. This leads to choices of the splits so that the overall cost is minimized and based on this we can compute a low cost coordination of lines.

4.1 Choice Sets

In order to keep track of possible blocks and splits and their costs we use the concept of a choice set, first introduced in [**?**]. Every choice set represents a consecutive set of indices that are still contaminated. These indices are the possible choices for splits of the blocking line that prevents the contamination of $\mathcal{C}$. Now, suppose we have a contaminated area circumscribed by k consecutive edges starting at edge e_i, i.e. the sequence of edges with indices $\{i, i+1, \ldots, i+k-1\}$. Write $T^i_k := \{i, i+1, \ldots, i+k-1\}$, call T^i_k a choice set, and let it represent the contaminated area circumscribed by $e_i, \ldots, e_{i+k-1}$. Write $T^i_0 = T_0 = \emptyset$ for all i. This contaminated area is separated from the other parts of the environment by a blocking line, written $l^b(T^i_k) := l^b_{i-1,i+k}$, going from edge e_{i-1} to edge e_{i+k}. As a shorthand write $b(T^i_k) := c(l^b(T^i_k))$, the blocking cost for the contaminated T^i_k.

To continue clearing the area of T^i_k we can choose a split on any index $o \in T^i_k$. Write $c(o|T^i_k) := c(o|i-1, i+k)$ using Eq. 1. Splitting on o will split the contaminated area for T^i_k into two disconnected contaminated areas. Each of these areas is again represented by a choice set, namely T^i_{o-i} to the left of o and $T^{o+1}_{i+k-o-1}$ to the right of o. For convenience we shall write $T^l = T^i_{o-i}$ and $T^r = T^{o+1}_{i+k-o-1}$. Each of these in turn have their own blocking lines and choices at a given cost. We can use the relationship of T^i_k to T^r and T^l to construct the cost of obstacle sequences recursively since $T^r, T^l \subset T^i_k$ and their low cost obstacle sequences are subsequences for low cost obstacles sequences of T^i_k. Using this recursive construction we will take advantage of identifying subsequences that cannot possibly improve obstacles sequences in larger choice sets. For this we will have to consider a fundamental problem for guaranteed search and pursuit-evasion. One of the key differences to other kinds of search problems is that once a certain search state, i.e. a cleared area, is reached there is a cost associated with maintaining, i.e. blocking, it. For guaranteed search we have to not only find lowest cost for extending the search state, but also to keep track of the resulting maintenance costs. Hence, in our recursive construction of obstacle sequences we not only have to consider sequences that have a low cost to clear a choice set, but also those that have intermediate steps with a low blocking cost, but overall higher clearing cost. The reason for this is that in combination with other obstacle sequence during the recursive construction, not only the clearing cost, but the intermediate blocking cost may determine the overall cost of the combined sequence. This can be the case if for T^r we have high clearing costs and in T^l we can reach a low blocking costs before clearing T^r. This means we can choose an obstacle sequence in T^l with higher individual clearing cost, but lower intermediate blocking cost, in order to reduce the total cost for clearing the more expensive area for T^r. In [**?**] a similar reasoning was used to develop the concept of full cut sequences and compute optimal pursuit-evasion strategies on weighted trees. These full cut sequences are used to keep track of useful states and sequences in which intermediate costs for blocking are low and that can be reached with a low clearing cost. In order to do this for obstacle sequences we first introduce their clearing and blocking costs.

Let $O = \{o_1, \ldots, o_k\}$ be an obstacle sequence using all indices from T^i_k, i.e. $o_s \in T^i_k$ and $o_s \neq o_{s'} \iff s \neq s'$. Write $\bar{O}$ for all such obstacle sequences. Computing the costs for the resulting sweep lines of any $O \in \bar{O}$ is easily done by executing the splits in the given order and keeping track of the resulting costs for blocking all choice sets until they are all cleared. Let $\mu_s(O)$ be the cost of all sweep lines located in the area for T^k_i when splitting on o_s and let $b_s(O)$ be the cost of all sweep lines after splitting on o_s, i.e. the blocking cost after step s. Let $T^{i_s}_{k_s}$ be the choice set so that o_s splits the line l_{i_s-1,i_s+k_s} onto e_{o_s}. Note that $i_s = \text{argmax}_{o \in \{o_1,\ldots,o_{s-1}\}}\{o < o_s\}$ and $k_s = \text{argmin}_{o \in \{o_1,\ldots,o_{s-1}\}}\{o > o_s\} - i_s - 1$. Now, $\mu_s(O)$ and $b_s(O)$ are given by:

$$\mu_s(O) = b_{s-1}(O) + c(o_s|T^{i_s}_{k_s}) - b(T^{i_s}_{k_s}) \quad (2)$$

$$b_s(O) = b_{s-1}(O) - b(T^{i_s}_{k_s}) \quad (3)$$

$$+ b(T^{i_s}_{o_s-i_s}) + b(T^{o_s+1}_{i_s+k_s-o_s-1}) \quad (4)$$

with $b_0(O) := c(l_{i-1,i+k})$, i.e. the cost of the blocking line for

Algorithm 1 $Combine\_Obstacle\_Sequences(k, i, o))$

$T^l \leftarrow T^{i+1}_{o-i}, T^r \leftarrow T^{o+1}_{i+k-o-1}$
$\tilde{O}(T^i_k) \leftarrow \emptyset$
$o_1 \leftarrow o$
for all $(O^l, O^r) \in \tilde{O}(T^l) \times \tilde{O}(T^r)$ **do**
  $s_l \leftarrow 1, s_r \leftarrow 1, s \leftarrow 2$
  while $s \leq k$ **do**
    if $\rho^l_{s_l} < \rho^r_{s_r}$ **then**
      $o_s \leftarrow o^l_{s_l}, s_l \leftarrow s_l + 1$
    else
      $o_s \leftarrow o^r_{s_r}, s_r \leftarrow s_r + 1$
    end if
    $s \leftarrow s + 1$
  end while
  if $\neg\, is\_dominated(O, \tilde{O}(T^i_k))$ **then**
    $\tilde{O}(T^i_k) \leftarrow \tilde{O}(T^i_k) \cup \{o_1, \ldots, o_k\}$
  end if
end for
return

T^i_k. Note that in the above equation $T^{i_s}_{o_s - i_s}$ and $T^{o_s+1}_{i_s+k_s-o_s-1}$ are simply the left and right choice set (recall T^r and T^l) from which an index is chosen at step s to do a split on and the two choice sets created by the split. Here, the left and right choice sets represent the contaminated areas to the left and to the right of o_s.

In order to quantify the tradeoff between the cost clearing everything until step s, given by $\max_{s' \leq s}\{\mu_{s'}(O)\}$, and the resulting blocking cost $b_s(O)$ we define:

$$\rho_s(O) = \max_{s' \leq s}\{\mu_{s'}(O)\} - b_{s-1}(O). \tag{5}$$

A large value for $\rho_s(O)$ indicates a high cost at step s but with a low prior blocking cost. We shall see the relevance of $\rho_s(O)$ in the next section for combining obstacle sequences from a left and right choice set.

4.2 Combining Obstacle Sequences

In order to compute obstacle sequences in a recursive manner we need to combine all useful sequences from the left and right choice set of a split. The parameter $\rho_s(O)$ we introduced above will help with this. Recall that choosing $o \in T^i_k$ splits the contaminated area into $T^l = T^i_{o-i}$ to the left of o and $T^r = T^{o+1}_{i+k-o-1}$ to the right of o. The possible index sequences for T^i_k that start with the choice of o can now be constructed by considering combinations from the index sequences $\bar{O}(T^l)$ and $\bar{O}(T^r)$. Given the two sequences $O_l = \{o^l_1, o^l_2, \ldots, o^l_{o-i}\} \in \bar{O}(T^l)$ and $O_r = \{o^r_1, o^r_2, \ldots, o^r_{i+k-o-1}\} \in \bar{O}(T^r)$ there are still a large number of possibilities how to combine them into a sequence for T^i_k, since we would have to consider all possible sequences in $\bar{O}(T^i_k)$ that have O_l and O_r as subsequences, e.g. $\{o, o^l_1, o^r_1, o^r_2, o^r_3, o^l_2, o^r_4, o^r_5, \ldots, o^l_{o-i}, \ldots, o^r_{i+k-o-1}\}$, and so on.

Fortunately, only one of the many possible combinations of O_l and O_r needs to be considered. Namely, the combination we obtain by ordering all o^l_s and o^r_s by ρ^l_s and ρ^r_s, i.e. the sequence $O = \{o_1 = o, o_2, o_3, \ldots, o_k\}$ which satisfies: if $o_s = o^l_{s'}$, then $\rho^l_{s'} < \rho^r_{s''}$ for all $o^r_{s''} = o_{s^*}$ with $s^* > s$ and vice versa for r. In colloquial terms, ordering all obstacles indices from the left and right according to their tradeoff value ρ determines the lowest combination. Going through the equations that determine the cost of O from the costs from the costs $\mu^l_s(O^l)$, $\mu^l_s(O^r)$, $b^l_s(O^l)$, and $b^l_s(O^r)$ confirms this. A formal statement and proof of this claim is beyond the scope of this paper, but it follows a similar line of reasoning as presented in [**?**] for full cut sequences and the basic idea is very simple. Namely, the obstacle indices should be combined so that one adds indices that can reduce the blocking cost before adding the indices that have a high clearing cost. Using this we can now compute the obstacle sequences for a choice set as shown with Alg. 1. In order to keep track of the relevant obstacle sequences we introduce $\tilde{O}(T^i_k) \subset \bar{O}(T^i_k)$ which only contains obstacle sequences constructed in Alg. 1 that are relevant for subsequent constructions.

Algorithm 2 $is\_dominated(O, \tilde{O})$

for all $O' = \{o'_1, \ldots, o'_k\} \in \tilde{O}, O' \neq O$ **do**
  $b_{min} \leftarrow b_1(O), b'_{min} \leftarrow b_1(O')$
  $\mu_{max} \leftarrow \mu_1(O), \mu'_{max} \leftarrow \mu_1(O')$
  while $s' \leq k$ AND $s \leq k$ **do**
    if $\mu_{max} < \mu'_{max}$ **then**
      if $b_{min} \leq b'_{min}$ **then**
        return $false$
      end if
      $\mu_{max} \leftarrow \max\{\mu_{max}, \mu_s(O')\}$
      $b_{min} \leftarrow \min\{b_{min}, b_s(O)\}$
      $s \leftarrow s + 1$
    else
      if $\mu_{max} = \mu'_{max}$ AND $b_{min} < b'_{min}$ **then**
        return $false$
      end if
      $\mu'_{max} \leftarrow \max\{\mu'_{max}, \mu_{s'}(O')\}$
      $b'_{min} \leftarrow \min\{b'_{min}, b_{s'}(O')\}$
      $s' \leftarrow s' + 1$
    end if
  end while
  return $true$
end for

Note that we still have to consider all possible combinations from sequences from $\tilde{O}(T^l)$ and $\tilde{O}(T^r)$. In order to reduce the number of sequences in $\tilde{O}(T^i_k)$ used for later constructions for larger choice sets we prune all sequences that are dominated. The domination criteria is described in Alg. 2. It prunes all sequences that are dominated by another sequence, i.e. they never have a lower blocking μ cost that can be reached with at a lower overall cost. In other words, whenever a dominated sequence reaches a low blocking cost, a dominating sequence already did so but with a lower or equal overall cost. The final blocking cost of a sequence for a choice set is always zero, since the entire choice set is then cleared.

At this point, it is still an open question whether the domination criteria suffices for reducing the growth of $\tilde{O}(T^i_k)$ for larger choice sets. We will briefly discuss this issue for the experimental results in Section 6. Once all obstacle sequences in all choice sets are computed the best sequence can be found by choosing the best sequence from all $\tilde{O}(T^i_{n-1}), i = 1, \ldots, n$.

4.3 Complexity

The fact that the algorithm is complete is trivial to verify. It produces a valid strategy that clears the environment by construction. Its computational complexity, however, is more difficult to determine. For a polygonal environment with n vertices there are $O(n^2)$, in fact $n \cdot (n-1)$, choice sets. The total number of choices for all choice sets is $\frac{n^3}{2}$. The Alg. 1 is called $\frac{n^3}{2}$ times in Alg. 3. The main problem is assessing the growth of $\tilde{O}(T^i_k)$. Its growth depends on how many sequences are dominated during the construc-

Algorithm 3 $Compute\_All\_Choice\_Sets())$

```
1: for all k = 1, ..., n − 1 do
2:   for all i = 1, ..., n do
3:     for all o = i, ..., i + k − 1 do
4:       Combine_Obstacle_Sequences(k, i, o)
5:     end for
6:   end for
7: end for
```

tion by the criteria in Alg. 2. In the worst case there may be no dominated sequences and the number of sequences is given recursively by: $|\tilde{O}(T_k^i)| = \sum_{o=i,\ldots,k+i-1} |\tilde{O}(T_{o-i}^i)| \cdot |\tilde{O}(T_{i+k-o-1}^{o+1})|$ with $\tilde{O}(T_0^i) := 1$ for all i. Since

$$\sum_{o=i,\ldots,k+i-1} |\tilde{O}(T_{o-i}^i)| \cdot |\tilde{O}(T_{i+k-o-1}^{o+1})| \geq 2 \cdot |\tilde{O}(T_{k-1}^i)|$$

the number of obstacle sequences can grow exponentially $\tilde{O}(T_k^i)$ for larger k. It is still an open problem to determine whether there exist environments that require keeping track of exponentially many obstacle sequences or whether there are stricter pruning criteria that can be shown to lead to only polynomial growth of the number of obstacle sequences. For all maps tested in Section 6, however, the average number of obstacles sequences in $\tilde{O}(T_k^i)$ was 3 or fewer.

4.4 Polygonization and Multiply-Connected Environments

In order to apply the 2D coordination strategies to our original 2.5D problem we have to obtain a polygonized and simply-connected representation of the environment. For this every position in the domain H of h that is traversable by a ground target is marked as free space. All free space reachable from a given starting position then provides the connected free space that forms our search environment $\mathcal{E}$. In our implementation this connected set $\mathcal{E} \subset H$ environment is then polygonized by computing an α-shape using the CGAL library [**?**]. These shapes are frequently used to reconstruct the shape of a dense set of points. From the α-shape we construct a polygon boundary and interior polygons. On these we apply the Ramer-Douglas-Peucker line-simplification algorithm [**?**] to strip redundant vertices and get a polygons with fewer edges. The loss of precision is given by a parameter ε which determines the degree of the simplification.

To obtain a simply-connected polygon the interior polygons are connected to the outer polygon boundary by connecting the closest polygon to the boundary first and then each subsequently closest polygon to the new combined boundary. These new artificial edges require additional searchers that will have to cover the edge whenever either side of the artificial edge is in a different state, i.e. one cleared and the other contaminated. This process is similar to the computation of graph searching strategies on graphs with cycles by removing all edges that cause cycles, computing a strategy on the tree and then adapting that strategy back to the graph. This approach has been discussed extensively in [**?**] and [**?**]. Clearly, further work in dealing with cycles in edge-searching, as well as clearing with lines, is warranted as it is expected to lead to reasonable improvements of the resulting strategies. At some points during the strategy multiple artificial edges may be covered while at other times there might be none, as seen in Fig. 4.

5. FROM 2D TO 2.5D

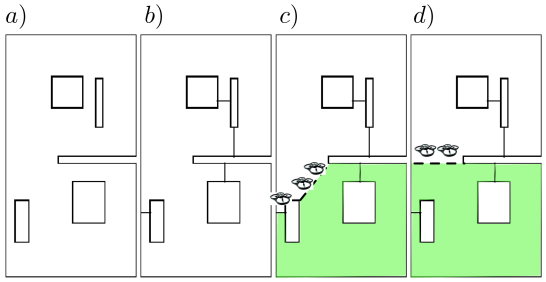

Figure 4: A multiply-connected polygon from a) is converted to a simply-connected polygon in b) by introducing artificial edges. For a clearing strategy only artificial edges between cleared and contaminated areas have to be covered as seen in c). In d) none of the artificial edges have to be covered.

For a 2D environments the blocks and splits computed in the previous section would suffice to determine the locations of UAVs on the lines by simply placing them on the line at distance $2 \cdot s_r$ to each other. Computing the continuous motion of the lines starting at the blocks and towards the splits is also trivial. For the 2.5D case, however, covering the sweep line is not as simple. When covering the line with the detection sets of UAVs one has to consider additional visibility constraints that can restrict the detection set. To solve this problem we first discretize the moving sweep lines with a given granularity to obtain a set of lines between every block and split. This discretization allows us to apply simple sampling-based approach to cover these discrete lines. Effectively, this gives us a set of lines $L(t) = \{l_1, l_2, \ldots, l_{k(t)}\}$ for every discrete time step $t = 1, \ldots, T$.

On the sampled set of lines $L(t)$ at time t we can then compute locations for UAVs that cover the line. The underlying terrain may have elevation changes that affect visibility considerably. Note that there may be elevation changes in open space that are not classified as non-traversable by the ground target and that these can affect visibility as well. We propose a simple greedy method to cover the lines. Given a line with endpoints $[p_l, p_r]$ we compute the detection set $D(p)$ for all points $p \in [p_l, p_l + s_r \cdot (p_r - p_l)]$ and chose the point p with the longest contiguous segment in $D(p) \cap [p_l, p_r]$ that contains p_l. Then the new left endpoint p_l is set to the end of this contiguous segment and the procedure is repeated until all of $[p_l, p_r]$ is covered. Using this procedure we convert $L(t)$ to a set of locations $P(t) = \{p_1, \ldots, p_{m(t)}\}$, $p_1 \in H$ for every $t = 1, \ldots, T$. These locations are effectively the 3D poses that have to be occupied by a UAV at time t. Finally, we use this discretized set of poses to compute trajectories for the UAVs. Here one still has to determine which UAV moves from its location at time t to one of the new required locations at time $t + 1$. To assign UAVs from their current location to a new location we use the Hungarian method [**?**] with the travel time to the new locations as the costs. This assigns UAVs to locations by minimizing the average cost, which can be distance or time. Any motion model or planner can be used to determine the cost.

6. RESULTS AND DISCUSSION

The source code used for the experiments in this paper is published online at http://code.google.com/p/guaranteed-search/ under a GNU GPLv3 license. In the following we refer to the results from the 2D simply-connected strategy, the 2D multiply-connected strategy, and the adaptation of the 2D simply-connected strategy

to 2.5D paths. More precisely, using the methods described in the above sections, we carried out experiments relating to the following questions: 1) growth of $\tilde{O}(T_k^i)$; 2) feasibility and demonstration in a large real elevation map and the cost of adapting 2D strategies to 2.5D; 3) cost of adapting strategies from simply-connected to multiply-connected polygons; 4) scaling of the number of robots with increasing environment size. To address these questions we used one realistic elevation map of the University of Freiburg seen in Fig. 5, and four mazes of different size seen in Fig. 6. The results are summarized in Table 1.

1) Growth of $\tilde{O}(T_k^i)$: An investigation of the growth of $\tilde{O}(T_k^i)$ showed that $|\tilde{O}(T_k^i)|$ does not grow exponentially, but is constant with an average around 3 for all the maps considered here. This shows that for the maps presented here most sequences are dominated by some few obstacle sequences with low clearing and blocking costs. We conjecture that one can, in fact, show that the simply-connected 2D strategies are optimal and that the algorithm is polynomial. A proof of this conjecture is a promising direction for further work.

2) Feasibility and demonstration in 2.5D: In order to demonstrate the practicability and feasibility of our approach we used an elevation map of the University of Freiburg and computed a 2D simply-connected strategy that clears the environments with 6 UAVs. To evaluate the adaptation to 2.5D and compare the greedy method to a baseline we also consider the random sampling of 2.5D detection set to cover the 2D lines. Adapting the simply-connected 2D strategy using randomly sampling of coverage positions on the line leads to an increase of the number of UAVs needed from 6 to 11. The greedy method leads to an increase to only 9. The other maps, which do not have interesting terrain, apart from that classified as obstacles, have an identical cost for the simply-connected 2D and 2.5D strategies with the greedy method. In Fig. 5 (d) the remaining white area is the area classified as outside of $\mathcal{E}$ and considerable elevation differences are present within the free space, such as trees, benches, smaller containers, and other objects.

3) Adapting strategies: The adaptation from simply-connected to multiply-connected polygons also requires a considerable number of additional robots. The cost of the base strategy with 6 robots for the Freiburg map increases to 16 with 12 robots being busy blocking cycles in the environment. This suggests, in unison with the work carried out in [**?**] and [**?**], that it is worth investigating the computation of strategies that consider cycles explicitly. This problem, however, is NP-hard on graphs, but we can aim for approximation algorithms or heuristics that may work well in practice. For the purposes of clearing a large environment with a real team of UAVs, the strategies on the Freiburg map provide an encouraging starting point. Snapshots of the resulting strategy are shown in Figure 5.

In addition, to determine whether the line coordination enables the robot team to exploit narrow corridors, we compared the maps Maze and Narrow Maze from Fig. 6. With a sensing range of 30 units both maps can be cleared with a similar number of UAVs, namely 5 and 4 respectively. For a smaller sensing range, at 5 units, the difference becomes more pronounced, namely 32 and 19 as seen in Table 1, and the narrow corridors are in fact exploited well by the strategy as blocking states with a low cost. Both maps have a similar increase in the number of robots when adapting the strategy from the simply-connected to the multiply-connected polygon.

4) Scaling to larger environments: Finally, a comparison between the three differently sized maze maps: Maze, Half Maze, and Quarter Maze shows a small difference in the number of robots from 5 to 4 and 4 respectively. The adaption to the multiply-connected polygon, however, shows that the larger map suffers much more from cycles in the environments, as is expected. Overall, this further supports the observation that in order to scale to much larger environments the issue of cycles should be addressed explicitly and that for simply-connected environments the approach already scales well.

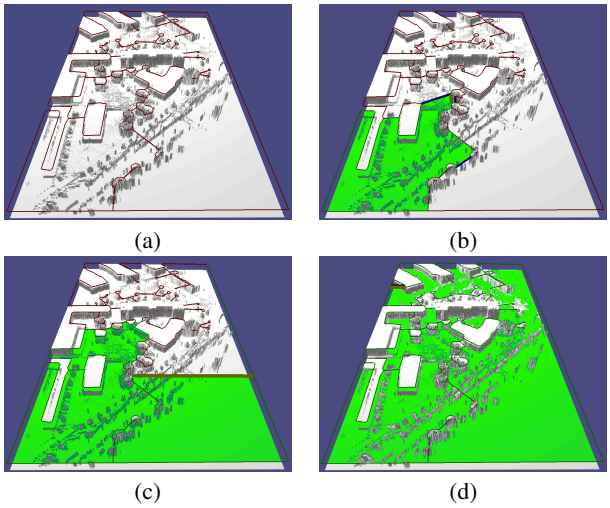

Figure 5: A sweep through the map of the Freiburg campus.

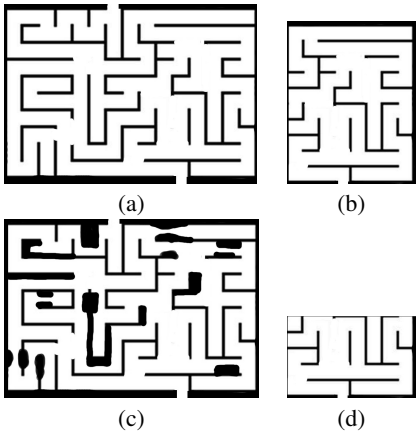

Figure 6: The four different maze environments used. In (a) we have the Maze, in (b) the Half Maze, in (c) a maze with narrow corridors (Narrow Maze), and in (d) the Quarter Maze.

7. CONCLUSIONS AND FUTURE WORK

We addressed the problem of coordinating a large team of UAVs searching for fast and smart ground targets in complex 2.5D environments with detection guarantees. The primary goal of this paper is to enable the building of a system of UAVs for guaranteed search in complex and large environments. Hence, we presented a comprehensive approach that addressed geometric, combinatorial, visibility, and path planning issues in 2D and 2.5D, but without rigorously addressing many theoretical and computational issues that arise on the different levels of abstractions. While such questions are certainly worth to be addressed on their own merits, the development of a working prototype with real UAVs will act as an even stronger motivator.

Apart from the obvious open questions there also remains the fast and parallel execution of a search. Especially with quad-rotors

Map	s_r	c_1	c_2	Size
Freiburg	30	6	16	610×1031
Narrow Maze	30	4	9	960×608
Maze	30	5	11	960×608
Narrow Maze	5	19	29	960×608
Maze	5	32	42	960×608
Half Maze	30	4	6	467×593
Quarter Maze	30	4	5	462×293

Table 1: Results for the cost of strategy in the constructed simply-connected polygon (c_1), the strategy adapted to the multiply-connected environment (c_2). All results are shown at a sensor range s_r with radius 30 or 5.

and their energy constraints such a fast execution will be crucial. Also the integration of motion models with more severe motion constraints, such as fixed wing UAVs, is a promising next step.

Even though our approach relies heavily on heuristics we demonstrated a successful application to a set of maps and provided an end-to-end implementation that delivers a set of trajectories. We further demonstrated the feasibility and scalability of our approach and successfully computed coordination strategies for a complex and large environment. A natural next step is to apply this to a real system and demonstrate powerful search capabilities with a real team of UAVs.

8. ACKNOWLEDGMENTS

This work is partially supported by grants from 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, grand agreement 600958.

9. REFERENCES