Solving Pursuit-Evasion Problems on Height Maps

A. Kolling* and A. Kleiner** and M. Lewis* and K. Sycara**

***Abstract*— In this paper we present an approach for a pursuit-evasion problem that considers a 2.5d environment represented by a height map. Such a representation is particularly suitable for large-scale outdoor pursuit-evasion. By allowing height information we not only capture some aspects of 3d visibility but can also consider target heights. In our approach we construct a graph representation of the environment by sampling points and their detection sets which extend the usual notion of visibility. Once a graph is constructed we compute strategies on this graph using a modification of previous work on graph-searching. This strategy is converted into robot paths that are planned on the height map by classifying the terrain appropriately. In experiments we investigate the performance of our approach and provide examples including a map of a small village with surrounding hills and a sample map with multiple loops and elevation plateaus. Experiments are carried out with varying sensing ranges as well as target and sensor heights. To the best of our knowledge the presented approach is the first viable solution to 2.5d pursuit-evasion with height maps.**

I. Introduction

Pursuit-evasion problems are an interesting domain for multi-robot systems. The spatial distribution and flexibility they can achieve are a great advantage compared to centralized and immobile systems. This makes the multi-robot approach very viable for real world tasks that relate to pursuit-evasion. So far, however, most of the research pursuit-evasion problems have considered idealized scenarios restricted to graphs or two-dimensional environments or certain types of idealized sensors such as unlimited range target detection. Yet, considerable progress has been made and more realistic applications are now coming into reach. The purpose of this paper is a first attempt to use part of the large body of research relating to pursuit-evasion with robot teams and apply it to a challenging scenario closer to real world pursuit-evasion, namely large 3d environments represented by height maps.

Related to this effort is visibility-based pursuit-evasion in two-dimensional environments and with unlimited range sensors [1], [2]. But these methods do not extend to very large teams of robots nor limited range and only deal with two-dimensional environments. Very litte work has so far been done for three-dimensional pursuit-evasion problems. A report by Lazebnik [3] discusses the challenges and complications when extending the ideas from two-dimensional visibility-based pursuit-evasion to three-dimensions. In the two-dimensional case so called critical events that occur as a robot moves through the environment fully determine the changes in the information about the evaders possible locations. Critical events turn out to be significantly more complex in three-dimensions. Not only is the catalogue of such events larger they also lead to non-local changes in the information states. Furthermore, shadow spaces are not guaranteed to be simply-connected which causes further complications. As a consequence the problem received little attention so far. The height maps that we consider capture at least some of the structure of a three-dimensional space.

Apart from visibility-based approaches we also find a number of attempts to utilize various forms of graph-searching, i.e. pursuit-evasion problems on graphs, for robotic pursuit-evasion. In [4], [5] the edge-searching problem is modified to better suit a robotic application by considering vertex-located intruder instead of edge-located. Furthermore, labeling based approaches, frequently found in edge-searching on trees, are incorporated into an anytime algorithm that tries many spanning trees of the graph. This allows the computation of strategies on graphs from the labels computed on a spanning tree. It is shown in [5] that for some labeling approaches this leads to a probabilistically complete algorithm for graphs. An alternative graph model for robotic pursuit-evasion, called Graph-Clear, is presented in [6], [7]. Therein actions on the graph that can detect a target can require multiple robots and the restriction of contamination is achieved not through placing searcher in vertices but also on edges. Automated methods to extract a graph representation for Graph-Clear have been presented in [8] and are based on detecting narrow parts of the environment via its Voronoi Diagram. An extension to probabilistic sensing models for Graph-Clear is found in [9] and can likely be extended to the edge-searching model as well. Similarly, the ideas of the anytime algorithm from [4] can also be applied to the tree algorithms from [7]. To conclude there are a variety of graph models and algorithms available to compute strategies for robot teams from a graph representation of the environment. How to obtain a good graph representation, however, remains an open problem. Apart from [8], [10] we have [11] in which a graph is extracted from the environment through random sampling, similar to our approach for obtaining an initial graph.

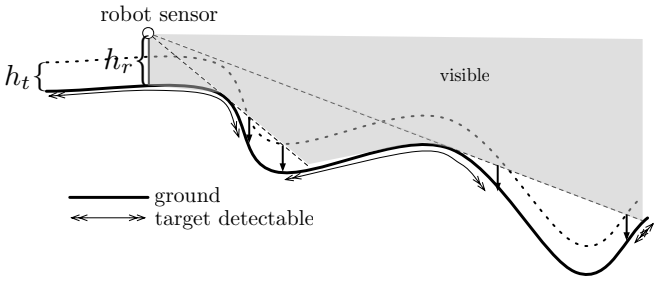
From the above we can see that a great deal of progress has been made in transporting the theory from graph-based pursuit-evasion to a robotic context. This, however, has been restricted to two-dimensional environments and in this paper we shall present the first attempt to tackle 2.5d robotic pursuit-evasion with height maps using a graph-based approach. We will first describe the problem in Section II, outline our algorithm in Section III, discuss trajectory planning on height maps in Section IV. Finally we shall present our experimental results in Section V and conclude with Section VI

** Robotics Institute, Carnegie Mellon University, 500 Forbes Ave., Pittsburgh, PA 15213 * School of Information Sciences, University of Pittsburgh, 135 N. Bellefield Ave., Pittsburgh, PA 15260

II. Problem Description

We consider a 2.5d map represented by a height function $h : H \to \mathbb{R}^+$. The domain H is continuous and $H \subset \mathbb{R}^2$ which for all practical purpose can be approximated by a 2d grid map that contains the heights as described in Section IV. We write $E \subset H$ for the free space in which robots can move and assume that it is connected, i.e. regardless of deployment a robot should always be able to move to another feasible point in E. All points not in E are considered non-traversable obstacles. The problem is to move a number of robots equipped with a target detection sensor through E to detect all targets that are potentially located therein. As in most pursuit-evasion scenarios targets are assumed to move on continuous trajectories within E but at unbounded speeds and are omniscient. Hence, robots and targets alike can only move on feasible paths in the environment and trajectories do not intersect parts of the height map that are not reachable. Additionally, targets have a minimum height h_t that can influence the visibility of a target. To capture the target detection capabilities of the robots let $D(p) \subset H$, the detection set of a point $p \in H$, be the set of all points in H on which a target is detectable by a robot located at p. In general $D(p)$ depends on the sensor model, height of the sensor h_r relative to $h(p)$ and height of targets h_t. For the problem in this paper we consider a limited range three-dimensional and omni-directional sensor. Hence, a target on $p' \in H$ detectable by a robot on p if the line segment $\{p', h(p')\}$ to $\{p', h(p') + h_t\}$ embedded in $\mathbb{R}^+$ is visible from $\{p, h(p) + h_r\}$ at distance s_{range}. Here h_t can be understood as the minimum height of any target for which we seek to guarantee a detection with the pursuit strategy. Notice that this is simply straight line visibility in the 3d space which the height map represents. Yet, even with such a simple detection model it is not guaranteed that $D(p)$ is simply-connected nor that it is connected. This applies even if the free space of the environment in which robots and targets can move is simply-connected and also when $E = H$. In this sense, our pursuit-evasion problem on height maps already captures significant complications that also arise in 3d pursuit-evasion.

The inclusion of target and sensor heights allows us to answer a variety of questions. It is rather straight forward to examine the effect of the height of a sensor, h_r on the number of robots needed to clear an environment. But the inclusion of target height introduces another interesting question. As seen in fig. 1, as h_t increases the size of $D(p)$ increases as well. With $h_t = 0$ we revert back to visibility of points on the height map, i.e., a target is seen if the ground it is on is seen. Now, we can determine how the number of agents changes as h_t and can produce strategies with less robots, given that we can impose a constraint on minimum target heights. In a practical application, for example, this means that we can inform a user that with 10 ground robots with omni-directional cameras mounted at $1m$ height we can detect all targets in a mountainous region if no target is smaller than $0.4m$ and that a further reduction to $0.3m$ means 12 ground robots or a sensor height of $2m$.



1: An illustration how to compute target detection areas with a simple straight line visibility sensor model.

III. Algorithm

Considering the difficulties visibility-based approaches face in 3d pursuit-evasion, as well as in 2d when the environment is multiply connected, we present a first attempt to solve our 2.5d pursuit-evasion by creating a graph that captures the visibility information in our environment heuristically and is directly embedded into the map. Each vertex is associated to a location which can be used as waypoints to plan the motion of the robot team. To assign these waypoints to individual robots we utilize previous work in edge-searching with a minor modification.

First, we randomly select points in free space as follows. First pick p_1 from E and then subsequently pick another p_i, $i = 2, \ldots$ from $E \setminus \bigcup_{j=1}^{i} D(p_i)$ until $E \setminus \bigcup_{j=1}^{i} D(p_i)$ is the empty set. This ensures that a target on any point in E can be detected from some point p_i. Write m for the number of points that are created during this procedure resulting in a graph with m vertices in a set V, each corresponding to a point. Fig. 2 shows a few examples of such vertices and their respective detection sets. The vertices can be understood as waypoints that will be used for the robot paths. In principle, this construction does not differ significantly from basic attempts to solve an art gallery problem for complete coverage or for 2d pursuit-evasion scenarios in which graphs are constructed at random. The main difference are the detection sets $D(p)$ which we shall now use to construct edge set E to complete the graph $G = (V, E)$. Notice that, due to our visibility model we have that if $h_t < h_r$, then for pair every $p, p' \in E$ if $p' \in D(p)$, then $p \in D(p')$, i.e. in colloquial terms they are mutually detectable. As a simple corollary for the graph construction we get that for all i, j, $i \neq j$ we have $p_i \notin D(p_j)$.

Intuitively, the edges of G should capture the neighborhood relationships between the detection sets $D(p_i)$. In a 2d scenario the analogue of our detection sets are guaranteed to be connected, allowing for simpler neighborhood relationships. In our case, however, these sets can be more complex. Consider the boundary of $D(p_i)$ written $\delta D(p_i)$. We are interested in which vertices can guard, i.e. avoid recontamination, of $D(p_i)$ if an agent is placed on them. Clearly, all vertices whose $D(p_j)$ intersect with $\delta D(p_i)$ can prevent targets from passing through aforementioned intersection. We can now add an edge between any two vertices v_i, v_j with associated

points p_i, p_j for which $\delta D(p_i)$ and $D(p_j)$ intersect. Algorithms 1,2, and 3 summarize the graph construction.

Now, at first sight it seems that we can solve a standard edge-searching pursuit-evasion problem on G, as done in [4], and use the resulting sequence of moves on the vertices as a solution. There is, however, one crucial difference. In the edge-searching scenario contamination spreads through any vertex that is not guarded. In our problem, however, we have overlapping intersections between neighboring vertices as illustrated in fig. 2. These overlaps mean that even when a neighboring vertex is not guarded contamination may not spread into $D(p)$ if sufficiently many other neighbors are guarded. This is illustrated in fig 2. Incorporating this aspect into a graph representation could potentially lead to improvements for the coordination of the robot team. Yet, it does also entail a great deal of further work regarding the theoretical foundations of such a model. In all previous graph-based models the guarded state, i.e. one in which contamination does not spread to cleared vertices is simply achieved by blocking paths on the tree. In this case an unguarded path on the graph may exist while none exists in E.

The above does not prohibit the application of edge-searching solutions, but renders them less efficient and motivates further work for a graph model that can capture this aspect. To apply an edge-searching strategy we do, however, need a modification to accommodate the fact that we cannot slide robots along an edge and guarantee that the originating vertex will not be recontaminated during the movement. This can occur when a robot has to traverse a very occluded and low area in the environment to get to its next vertex position. Hence, we can only move a robots from a vertex to another if all neighboring vertices are cleared or guarded, otherwise it will recontaminate. In contrast, original edge-searching one can move the guarding robot to one neighboring contaminated vertex if all other neighbors are cleared or guarded.

To compute strategies we use the simple label-based algorithm from [12] with a modified label equation to accomodate our modified spread of contamination. The result of this algorithm is a contiguous strategy on a tree, i.e. a sequence of vertices that guarantees that all clear vertices are a connected subtree. The label on an edge $e = (v_x, v_y)$ is directional and represented by $\lambda_{v_x}(e)$ for the direction from v_x to v_y. If v_y is a leaf then $\lambda_{v_x}(e) = 1$. Otherwise let $v_1, \ldots, v_m$ be the $m = degree(v_x) - 1$ neighbors of v_y different from v_x. Now define $\rho_i := \lambda_{v_y}(\,(v_y, v_i)\,)$ and order all $v_1, \ldots, v_m$ with ρ_i descending, i.e. $\rho_i \geq \rho_{i+1}$. In original edge searching the label would now be $\lambda_x(e) = \max\{\rho_1, \rho_2 + 1\}$. In our modified version, however, the equation simplifies to:

$$\lambda_{v_x}(e) = \begin{cases} \rho_1 + 1 & if \rho_1 = 1 \\ \max\{\rho_1, \rho_2 + 1\} & otherwise \end{cases} \quad (1)$$

This is due to the fact that only after the last subtree is being cleared the guard on v_y can be removed while in edge-searching one can move the guard from v_y to help clear the last subtree even for leaves. In our variant this is not possible for leaves since the guard on v_y can only be removed after v_1 itself has been cleared. Notice that

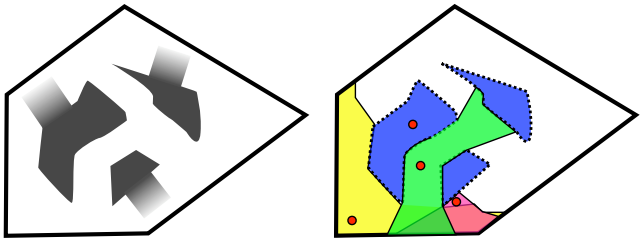
```
for all directions do
  for all cells along directions do
    Check whether a target in this cell is detectable
    from p
    if target detectable then
      add cell to S
Return S
```

Algorithm 1: $D(p)$

```
i ← 0, V ← ∅, P ← ∅
while E \ ⋃_{j=1}^{i} D(p_j) ≠ ∅ do
  pick any p_i ∈ E \ ⋃_{j=1}^{i} D(p_j)
  V ← V ∪ v_i, P ← P ∪ p_i, i ← i + 1
Return V, P
```

Algorithm 2: $Vertex\_Construction()$

the original algorithm from [12] accommodates weighted edges and vertices. For the weighted case, however, it was shown to be suboptimal in [13] and optimality on trees only holds in the unweighted case. Our formulation also allows to use the idea from the anytime algorithm, called GSST from [4], which tries multiple spanning trees to improve the strategy for the graph. Hence, we generate a number of spanning trees for our graph G and compute a strategy for each which we convert to a strategy on the graph by adding additional robots when necessary and then finally select the one with the least robots. In [5] it was proven that this leads to an asymptotically optimal algorithm for graphs, i.e. if run for a sufficiently long time it will find the optimal spanning tree. We conjecture that the result also holds for the simpler modified version presented here and the multiple spanning tree idea can be applied in a straightforward fashion. In Section V we confirm that this method works well in practice.



2: The figure shows overlapping detection sets that enable two vertices to guard the boundary at the left of the detection set marked in red. Both, green and yellow areas can guard it.

IV. Trajectory Planning on Height Maps

In this section we describe our approach for trajectory planning on height maps according to the motion model of the robot. A height map is represented by a two-dimensional array storing at each discrete location the corresponding elevation of the environment. On the one hand, height maps are widely available on the Internet as digital elevation maps (DEMs), e.g. from USGS [14]

$E \leftarrow \emptyset$
for $i = 1$ *to* m **do**
  for $j = 1$ *to* m **do**
    $I \leftarrow \delta D(p_i) \cap D(p_j)$
    if $I \neq \emptyset$ **then**
      $E \leftarrow E \cup \{v_i, v_j\}$
Return E

Algorithm 3: $Edge\_Construction(V, P)$

at a resolution of up to 10 meters. On the other hand, as we have shown in previous work, higher resolutions can be achieved in real-time on a mobile robot platform while traversing rough terrain [15]. On the mobile robot, elevation values are computed by successively integrating three-dimensional point clouds, generated by a tilted or rotated Laser Range Finder (LRF), with the 6D pose $(x, y, d, \psi, \theta, \phi)$ of the robot.

A. Classification

Height maps are classified into traversable and non-traversable terrain, which is needed for computing the pursuit-evasion graph, but also for path planing trajectories towards strategic locations encoded in this graph. The classification is carried out according to the motion model of the robot. Basically, different robot platforms have different capabilities to traverse terrain. For example, whereas a wheeled platform, such as the *Pioneer AT*, depend on even surfaces, tracked platforms, such as the *Telemax* robot, are capable of negotiating stairs and slopes up to 45°. This specific parameters are taken into account by the classifier described in the following.

For each cell of the height map, representative features are created that discriminate different structure element from the environment. We choose to use fuzzified features, which are generated by functions that project parameters, as for example, the height difference between cells, into the $[0, 1]$ interval. In contrast to binary $\{0, 1\}$ features, fuzzification facilitates the continuous projection of parameters, as well as the modeling of uncertainties. Fuzzification is carried out by combining the functions $SUp(x, a, b)$ (Equation 2) and $SDown(x, a, b)$ (Equation 3), where a and b denote the desired range of the parameter.

$$SUp(x, a, b) = \begin{cases} 0 & \text{if } x < a \\ \frac{x-a}{b-a} & \text{if } a \leq x \leq b \\ 1 & \text{if } x > b \end{cases} \quad (2)$$

$$SDown(x, a, b) = 1 - SUp(x, a, b) \quad (3)$$

For example, the features *Flat Surface*, *Wall Height* and *Ramp Angle* are build from the parameters δh_i, denoting the maximum height difference around a cell, and α_i, denoting the angle between the normal vector $\mathbf{n_i}$ and the upwards vector $(0, 1, 0)^T$, as shown by Equation 4 and Equation 5, respectively.

$$\delta h_i = \max_{j \text{ is neighbor to } i} |h_i - h_j| \quad (4)$$

$$\alpha_i = \arccos\left((0, 1, 0)^T \cdot \mathbf{n_i}\right) = \arccos\left(n_{i_y}\right) \quad (5)$$

For example, on a tracked platform, these features are defined by:

- Flat Surface = $SDown(\delta h_i, 15mm, 40mm)$
- Wall Height = $SUp(\delta h_i, 200mm, 300mm)$
- Ramp Angle = $SUp(\alpha_i, 3°, 25°) \cdot SDown(\alpha_i, 25°, 40°)$

Each time the elevation map is updated, the classification procedure applies fuzzy rules on the latest height estimates in order to classify them into regions, such as *flat ground*, *wall*, and *undecidable region*. The rules for flat ground and wall are trivial, as for example, "if δh_i is *flat surface*, then class is *flat ground*". Regions that are undecidable are extracted by "if δh_i is not *flat surface* and δh_i is not *wall height*, then class is *undecidable region*". Inference is carried out by the *minimum* and *maximum* operation, representing the logical *and* and *or* operators, respectively, whereas negations are implemented by $1-x$, following the definition given in the work of Elkan [16]. After applying the rule set to each parameter, the classification result is computed by defuzzification, which is carried out by choosing the rule yielding the highest output value.

The procedure is executed in real-time, and is the first step to classify cells into traversable and non-traversable terrain. For discriminating more complex obstacle types, such as ramps and stairs, Markov Random Field (MRF) models, can be used [17].

B. Path Planning

We employ two-dimensional A* search for trajectory planning. The A* algorithm performs informed search on graphs, which have a cost function assigned to their edges. To guide the search it uses a heuristic $h(n)$ to estimate the distance of a node n to the goal and given this heuristic is admissible, i.e., $h(n) \leq h^*(n)$ for all nodes n, with h^* being the optimal heuristic. To facilitate A* planning a graph has to be constructed from the height map. This is carried out by computing a distance map from the height map encoding in each cell the minimal distance to the next non-traversable cell.

From the distance map a plan is generated by expanding each connected traversable cell with the following cost function:

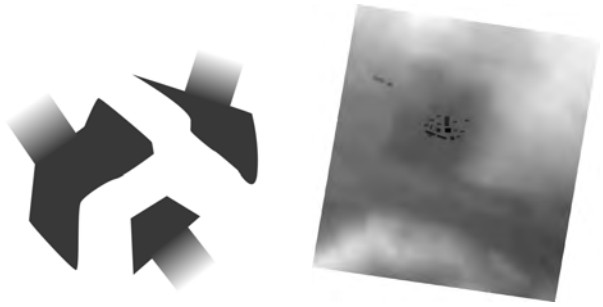
$$c(s_{i+1}) = c(s_i) + \alpha \frac{d(s_{i+1}, s_i)}{df(s_{i+1})} \quad (6)$$

Where $d(.)$ is the Manhattan distance, $df(s)$ the distance map entry for cell s, and α a factor for varying the cost for passing nearby obstacles. The heuristic used for guiding the A* search is the Euclidean distance $h = \sqrt{\delta x^2 + \delta y^2}$, which is commonly employed.

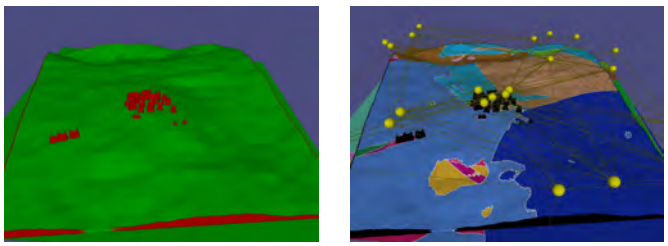
V. Experiments and Results

In this section we present result on two maps seen in fig. 3 The resolution of both is 0.1m/pixel. Sensing ranges mentioned below are always measured in meter. The height of cell in the map is given by its grey level and ranges from 0m to 10m with 0m as white and 10m as black. Fig. 4 shows the traversability classification based on a *Telemax* robot.

Recall that there are two random components to our algorithm. First, the graph that covers the map with vertices located within the map is generated by randomly sampling points from free space on which target cannot



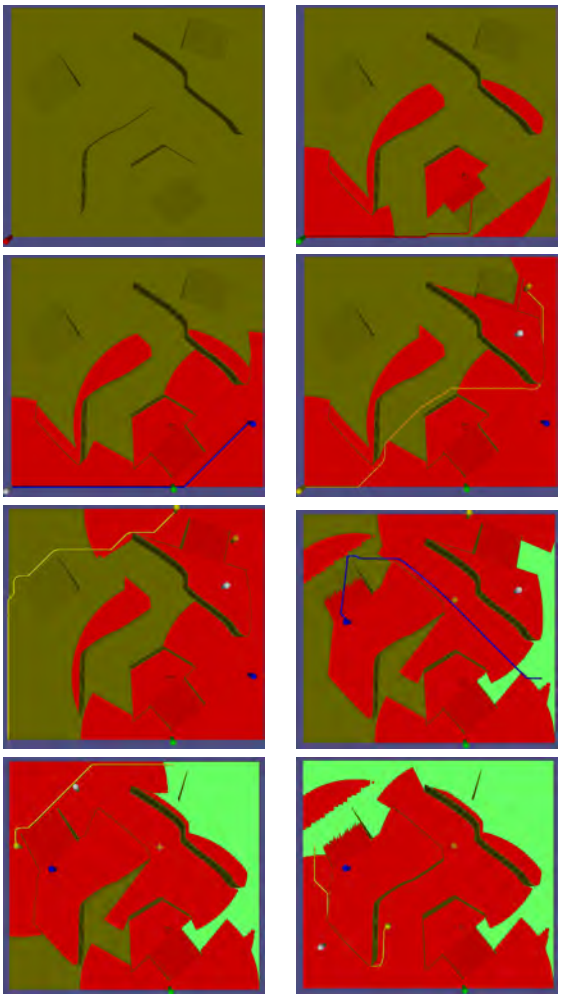
3: Left: a sample map with a three-way canyon, three plateaus with each its own ramp and several concave sections. Size of the map is 843x768. Right: a map of a small village with surrounding hills. Size of the map is is 798x824.



4: (a) Traversability, red is marked as non-traversable. (b) Detection sets with graph computed for robots with $h_r = 2.0m$, $s_{range} = 30m$, and $h_t = 0.2m$. Vertices are clustered more densely in the center of the village

yet be detected. Second, the strategy on the generated graph is computed by randomly trying strategies on multiple spanning trees. We hence conducted extensive tests to investigate the effect of the latter with our sample map seen in fig. 3. We generated 100 graphs via the random sampling within E. Then for each of these graphs we computed the best strategies based on 1) 10, 2) 100, and 3) 1000 randomly generated depth-first spanning trees, similar to [4]. Across all spanning trees we selected the one leading to the best strategy, i.e. the one needing the least robots. The results are presented in table V. Fig. 6 shows the distribution of number of robots across the 100 randomly generated graphs for 100 and 10000 spanning trees. Only for the smallest sensing range $s_{range} = 10$ the difference in the number of spanning trees had an effect on the best strategy that was found. For all other cases 100 spanning trees sufficed. Notice that smaller sensing ranges lead to more vertices and one would expect to require more spanning trees for larger graphs. Regarding the sensing range an increase from 10 to 30 reduces the number of robots needed significantly, while a further increase to 50 has no effect and to 70 only a small effect of a reduction by one. Notice that for complex environments a gain in the sensing range is mediated through the number of occlusions. With many occlusions an increase in sensing range is less likely to lead to improvements.

We also tested the algorithm on a real map from a small village also seen in fig. 3. Here we also varied the sensing range from 10 to 70 also observing a steep



5: A strategy for our sample map from fig. 3 with 6 robots. Detection sets are marked red and cleared areas not under observation are marked green. At step 0 on the upper left all robots are at their deployment location at the bottom left. The pictures show steps $0, 1, 3, 5, 6, 7, 10$ and 12 from left to right and top to bottom. At each step the path of the last robot moving is drawn. At step 1 the first robot moves to a plateau and after step 5 the robots cleared half the map. In step 6 all 6 robots are required to avoid recontamination of the graph. In step 8 the first cleared but unobserved part of the environment appears until in step 12 the entire environment is cleared.

decrease in the number of robots from 10 to 30 and then to 50 and 70 only a small changes of one. Since this map has considerably more elevation structure we also tested the effect of varying h_r and h_t. A reduction of h_t from $1m$ to $0.5m$ requires 9 instead of 8 for the same sensing range and $h_r = 1m$. A reduction of h_r from $1m$ to $0.5m$ requires 10 instead of 8 for the same sensing range and $h_t = 1m$. Reducing both, h_t and h_r to $0.5m$ needs 11 instead of 8 robots. Changes in h_r modify the set of visible cells and hence the detection sets while changes in h_t only modify the detection sets and the effect is not necessarily identical as suggested by the data.

s_{range}	spanning trees	min	max	mean	covariance
10	100	15	22	18.69	2.36
10	1000	14	20	16.8	1.58
10	10000	13	18	15.66	1.12
30	100	6	11	8.47	0.98
30	1000	6	10	7.96	0.73
30	10000	6	9	7.71	0.63
50	100	6	11	8.04	1.17
50	1000	6	11	7.70	0.98
50	10000	6	11	7.67	0.99
70	100	5	11	7.92	1.04
70	1000	5	10	7.70	0.98
70	10000	5	10	7.63	1.00

Table I: Results of the experiments on the sample map from fig. 3 with $h_p = 1.0$ and $h_t = 1.0$ and varying range and number of spanning trees. Values are from 100 randomly generated graphs. The last four columns display the minimum, maximum, mean and covariance of the number of robots needed across these 100 graphs for different ranges and numbers of spanning trees.

6: Two histograms of the distributions of the number of robots needed for the 100 randomly generated graphs. The left histogram is based on a generation of 100 spanning trees to compute the graph strategy while the right is based on the best out of 10,000 spanning trees.

VI. Conclusion

We have proposed a novel and to our best knowledge the first approach for 2.5d pursuit-evasion with height maps. Our approach is as a first baseline for the problem and as such serves for future comparisons with improved methods. The random graph generation can readily be substituted with either better sampling by biasing selection towards points with large detection sets or geometric methods that construct graphs such as in [8] based on visibility information. Also the graph model poses a new set of questions for further work. The edge-searching model with the modified spread of contamination is not entirely satisfactory as discussed in Section III. At this point it is unclear how amend this with a different graph model, but it will most certainly have to involve a different concept of guarding than previous graph-based model. Despite the fact that the presented approach is based on heuristics we have demonstrated that it already performs reasonably well in complex environments with loops and many occlusions and height differences. Its simplicity also makes it readily applicable to a variety of environments, even those with structures that resemble indoor environments, such as streets and building walls.

s_{range}	h_r	h_t	min	max	mean	covariance
10	1	1	16	22	19.46	1.71
30	1	1	9	17	12.14	2.69
50	1	1	8	15	11.76	2.10
70	1	1	8	16	11.66	2.47
50	0.5	0.5	11	21	15.46	3.44
50	0.5	1	10	17	12.82	2.23
50	1	0.5	9	18	14.52	2.57

Table II: Results of the experiments on the village map from fig. 3 with varying range and h_c,h_t. Values are from 100 randomly generated graphs.

References

[1] S. Sachs, S. Rajko, and S. M. LaValle, "Visibility-based pursuit-evasion in an unknown planar environment," *International Journal of Robotics Research*, vol. 23, no. 1, pp. 3–26, Jan. 2004.

[2] L. J. Guibas, J.-C. Latombe, S. M. LaValle, D. Lin, and R. Motwani, "A visibility-based pursuit-evasion problem," *International Journal of Computational Geometry and Applications*, vol. 9, pp. 471–494, 1999.

[3] S. Lazebnik, "Visibility-based pursuit-evasion in three-dimensional environments," University of Illinois at Urbana-Champaign, Tech. Rep., 2001.

[4] G. Hollinger, A. Kehagias, S. Singh, D. Ferguson, and S. Srinivasa, "Anytime guaranteed search using spanning trees," The Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, Tech. Rep. CMU-RI-TR-08-36, August 2008.

[5] A. Kehagias, G. Hollinger, and A. Gelastopoulos, "Searching the nodes of a graph: theory and algorithms," Carnegie Mellon University, Tech. Rep. ArXiv Repository 0905.3359 [cs.DM], 2009.

[6] A. Kolling and S. Carpin, "Multi-robot surveillance: an improved algorithm for the Graph-Clear problem," in *Proceedings of the IEEE International Conference on Robotics and Automation*, 2008, pp. 2360–2365.

[7] ——, "Pursuit-evasion on trees by robot teams," *IEEE Transactions on Robotics*, vol. 26, no. 1, pp. 32–47, 2010.

[8] ——, "Extracting surveillance graphs from robot maps," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2008, pp. 2323–2328.

[9] ——, "Probabilistic Graph-Clear," in *Proceedings of the IEEE International Conference on Robotics and Automation*, 2009, pp. 3508–3514.

[10] ——, "Surveillance strategies for target detection with sweep lines," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2009, accepted for publication.

[11] M. Moors, T. Röhling, and D. Schulz, "A probabilistic approach to coordinated multi-robot indoor surveillance," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2005, pp. 3447–3452.

[12] L. Barrière, P. Flocchini, P. Fraigniaud, and N. Santoro, "Capture of an intruder by mobile agents," in *Proceedings of the Fourteenth Annual ACM Symposium on Parallel Algorithms and Architectures*. New York, NY, USA: ACM Press, 2002, pp. 200–209.

[13] A. Kolling and S. Carpin, "On weighted edge-searching," School of Engineering, University of California, Merced, Tech. Rep. 01, 2009.

[14] (2010) U.S. Geological Survey (USGS). Http://www.usgs.gov/. [Online]. Available: http://www.usgs.gov/

[15] A. Kleiner and C. Dornhege, "Real-time localization and elevation mapping within urban search and rescue scenarios," *Journal of Field Robotics*, vol. 24, no. 8–9, pp. 723–745, 2007.

[16] C. Elkan, "The paradoxical success of fuzzy logic," in *Proceedings of the Eleventh National Conference on Artificial Intelligence*. Menlo Park, California: AAAI Press, 1993, pp. 698–703.

[17] C. Dornhege and A. Kleiner, "Behavior maps for online planning of obstacle negotiation and climbing on rough terrain," in *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots & Systems (IROS)*, San Diego, California, 2007, pp. 3005–3011.