Pursuit-Evasion in 2.5d based on Team-Visibility

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, captures some aspects of 3d visibility and can include target heights. In our approach we construct a graph representation of the environment by sampling points and computing detection sets, an extended notion of visibility. Moreover, the constructed graph captures overlaps of detection sets allowing for a coordinated team-based clearing of the environment with robots that move to the sampled points. Once a graph is constructed we compute strategies on it utilizing previous work on graph-searching. This 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 sample map with multiple loops and elevation plateaus and two realistic maps, one of a village and one of a mountain range. 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. 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 we have research on 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. 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 searchers 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]. 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 introduce a novel

** 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

method for computing guaranteed clearing strategies for a team of robots searching for an evader on height maps. This is carried out by first, randomly sampling *strategic locations* on the map and computing their *detection set*, i.e., the set of locations at which an evader can be detected. The overlaps between these detection sets are computed which determines which detection set can guard the boundaries of another. The set of strategic locations and the overlaps of their detection sets are captured in a graph on which we can compute clearing strategies for robot teams. The strategy is executed by selecting for each robot at each time step an appropriate strategic location, which is then reached by planning a trajectory executable on the terrain. 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

II. Problem Description

We consider a 2.5d map represented by a height function $h : H \rightarrow \mathbb{R}^+$. 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 $\mathcal{E} \subset H$ 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 $\mathcal{E}$. All points not in $\mathcal{E}$ are considered non-traversable obstacles. The problem is to move a number of robots equipped with a target detection sensor through $\mathcal{E}$ to detect all targets that are potentially located therein. Targets are assumed to move on continuous trajectories within $\mathcal{E}$ but at unbounded speeds and are omniscient. 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. 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_r. 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 $\mathcal{E} = H$. 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 relating to h_r,h_t. 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. In a practical application 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 $2m$.

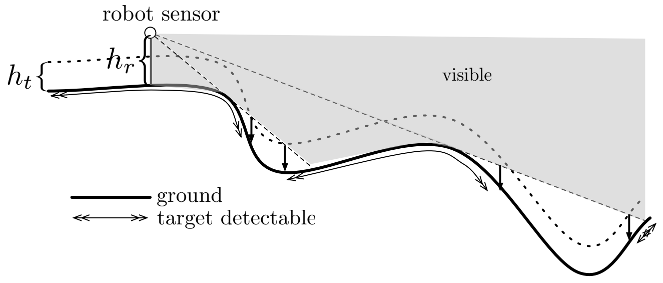

Fig. 1. An illustration how to compute target detection areas with a simple straight line visibility sensor model.

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.

We randomly select points in free space as follows. First, pick p_1 from $\mathcal{E}$ and then subsequently pick another p_i, $i = 2, \ldots$ from $\mathcal{E} \setminus \bigcup_{j=1}^{i} D(p_i)$ until $\mathcal{E} \setminus \bigcup_{j=1}^{i} D(p_i)$ is the empty set. This ensures that a target on any point in $\mathcal{E}$ 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 $p, p' \in \mathcal{E}$ 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 a robot is placed on them. Clearly, all vertices whose detection set intersect with $\delta D(p_i)$ can prevent targets from passing through aforementioned intersection. Hence, we are considering $\delta D(p_i) \cap D(p_j)$, $j \neq i$. If for a vertex $j \neq i$ we have $\delta D(p_i) \cap D(p_j) \neq \emptyset$ then a robot on v_j can guard part of $\delta D(p_i)$. For convenience let us write $G_{i,j} := \delta D(p_i) \cap D(p_j) \neq \emptyset$ and call it the guard region of v_i from v_j. From this guard region we shall now construct two types of edges. To distinguish the types we define the following condition

$$shady(v_i, v_j) := \left\{ \begin{array}{ll} 1 & \exists v_{j'} \in V, j' \neq j, i: \ G_{i,j} \subsetneq G_{i,j'} \\ 0 & otherwise \end{array} \right. \tag{1}$$

For now suppose edges have a direction and are written $[v_i, v_j]$ from v_i to v_j. The first type, a regular edge, is created from v_i to v_j, $i \neq j$, iff $G_{i,j} \neq \emptyset$ and $shady(v_i, v_j) = 0$. In colloquial terms v_i and v_j get a regular edge if and only if there is no third vertex $v_{j'}$ whose guard region of v_i completely covers the guard region of v_i from v_j. The second type, a so called *shady* edge, is created from v_i to v_j iff $G_{i,j} \neq \emptyset$ and $shady(v_i, v_j) = 1$. In this case there is a third vertex that completely covers the guard region. Hence if $G_{i,j} \neq \emptyset$, then we have an edge $[v_i, v_j]$ that is either shady or regular. To get a graph without directional edges, written (v_i, v_j), we simply add an edge if we have either $[v_i, v_j]$ or $[v_j, v_i]$ with regular edges dominating shady edges. Write $E = E_r \cup E_s$ for the set of undirected edges where E_r are the regular and E_s are the shady edges. Algorithms 1, and 2 present the above in details with pseudo-code. The reasoning behind creating two types of edges is straightforward. If a robot is placed at p_i, i.e. vertex v_i, it sees all targets in $D(p_i)$ and hence clears it. The robot can only be removed without causing recontamination if it can be guaranteed that no target can pass through $\delta D(p_i)$. This is satisfied if all vertices that are neighbors of v_i through regular edges are either clear or have a robot on them. The shady edges capture the remaining intersections between detection sets that are dominated by larger intersections.

Let us now describe the pursuit-evasion model on the graph level. 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 strategy as a solution. But apart from the addition of shady edges there is another crucial difference. In the edge-searching scenario contamination spreads through any vertex that is not guarded, but the robot on a guarded can slide along an edge and guard a new vertex without contamination spreading from the new vertex to the old. This also applies to the edge-searching variant from [4] that considers contamination in vertices instead of edges. For our problem this implies that while we move a robot from vertex v_i to v_j we would have to guarantee that no target could enter from contaminated areas of v_j to v_i. Since we cannot guarantee that the path a robot takes in our height map will continuously cover the boundaries of these region we cannot allow such sliding moves. Instead we only allow the removal and placement of agents on vertices in order not to impose additional requirements on the paths between vertices. The following modification incorporates this into edge-searching. As a basis we use the simple label-based algorithm from [12] with a modified label equation. The result of this algorithm is a contiguous strategy on a tree without recontamination, i.e. a sequence of vertices that guarantees that all clear vertices are a connected subtree. Hence we assume that we converted our graph into a tree by selecting a spanning tree T. For now we can ignore the difference between shady and regular edges. 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. $\rho_i \geq \rho_{i+1}$. The team of robots now clears the subtrees that are found at each v_i in the order $v_m, \ldots, v_1$. This leads to an overall cost represented by the next label $\lambda_{v_x}(e)$. In original edge searching the label would now be $\lambda_{v_x}(e) = \max\{\rho_1, \rho_2 + 1\}$. In our modified version the equation becomes:

$$\lambda_{v_x}(e) = \left\{ \begin{array}{ll} \rho_1 + 1 & if \rho_1 = 1 \\ \max\{\rho_1, \rho_2 + 1\} & otherwise \end{array} \right. \tag{2}$$

Where we assume that $\rho_2 = 0$ if $m = 1$. The change results from the fact that only after the first vertex of the last subtree, i.e. v_1, is cleared then the guard on v_y can be removed. Hence if $\rho_1 =$ which implies that v_1 is a leaf the label is $\rho_1 + 1$. Otherwise, if $\rho_1 > 1$, then the robot can be removed after v_1 is cleared and used subsequently in the remaining subtree beyond v_1. In edge-searching the guard on v_y can be move into v_1 earlier to clear it.

Our formulation now allows us to use the idea from the anytime algorithm, called GSST, from [4] which tries multiple spanning trees to improve the strategy for the graph. For this 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 leaving robots at their position whenever a cycle edge leads to a contaminated vertex. Finally we select the strategy across all spanning trees that leads to the least robots that are needed on the graph. In [5] it was proven that this can lead to an asymptotically optimal algorithm for graphs, i.e. if run for a sufficiently long time it will

```
$i \leftarrow 0, V \leftarrow \emptyset, P \leftarrow \emptyset$
while $E \setminus \bigcup_{j=1}^{i} D(p_j) \neq$ do
  pick any $p_i \in E \setminus \bigcup_{j=1}^{i} D(p_j)$
  $V \leftarrow V \cup v_i$, $P \leftarrow P \cup p_i$, $i \leftarrow i+1$
Return $V, P$
```

Algorithm 1: $Vertex\_Construction()$

```
$E_r, E_s \leftarrow \emptyset, E_{r,dir}, E_{s,dir} \leftarrow \emptyset$
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
      if $shady(v_i, v_j)$ then
        $E_{s,dir} \leftarrow E_{s,dir} \cup \{[v_i, v_j]\}$
      else
        $E_{r,dir} \leftarrow E_{r,dir} \cup \{[v_i, v_j]\}$
for $i = 1$ to $m$ do
  for $j = i+1$ to $m$ do
    if $[v_i, v_j] \in E_{r,dir}$ OR $[v_j, v_i] \in E_{r,dir}$ then
      $E_r \leftarrow E_r \cup (v_i, v_j)$
    else if $[v_i, v_j] \in E_{s,dir}$ OR $[v_j, v_i] \in E_{s,dir}$ then
      $E_s \leftarrow E_s \cup (v_i, v_j)$
Return $E_s, E_r$
```

Algorithm 2: $Edge\_Construction(V, P)$

find the optimal spanning tree for some of the label-based approaches. 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.

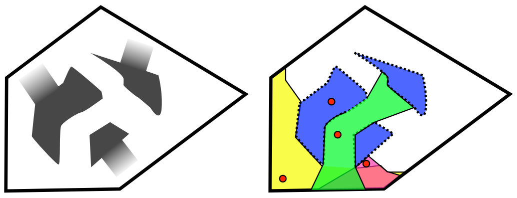

Fig. 2. The figure shows overlapping detection sets that enable two vertices to guard part of the boundary of the red detection sets. Yellow covers more of the boundary and hence receives a regular edge to red while green only gets a shady edge.

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 [13] at a resolution of up to 10 meters. On the other hand, as we have shown in previous work, higher resolutions can

```
$max\_cost \leftarrow \infty$
for $i = 1$ to $trees$ do
  Generate a spanning tree $T$ from $G$
  Compute strategy $S_T$ on $T$
  Convert $S_T$ to a strategy $S_G$ on $G$
  if $cost(S_G) < max\_cost$ then
    $best\_strategy \leftarrow S_T, max\_cost \leftarrow cost(S_G)$
Return $best\_strategy$
```

Algorithm 3: $Compute\_Strategy(G, trees)$

be achieved by traversing the terrain with a mobile robot platform [14]. 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.

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 3) and $SDown(x, a, b)$ (Equation 4), where a and b denote the desired range of the parameter.

$$SUp(x, a, b) = \begin{cases} 0 & \text{if } x < a \\ \frac{x-a}{b-a} & \text{if } a \leq x \leq b \\ 1 & \text{if } x > b \end{cases} \tag{3}$$

$$SDown(x, a, b) = 1 - SUp(x, a, b) \tag{4}$$

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 5 and Equation 6, respectively.

$$\delta h_i = \max_{j \text{ is neighbor to } i} |h_i - h_j| \tag{5}$$

$$\alpha_i = \arccos\left((0, 1, 0)^T \cdot \mathbf{n_i}\right) = \arccos\left(n_{i_y}\right) \tag{6}$$

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*, and *steep wall*.

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 [15]. 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. For discriminating more complex obstacle types, such as ramps and stairs, Markov Random Field (MRF) models, can be used [16].

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 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:

$$c(s_{i+1}) = c(s_i) + \alpha \frac{d(s_{i+1}, s_i)}{df(s_{i+1})} \tag{7}$$

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

There are a number of variations that are possible for the generation of multiple spanning trees as well as the conversion of the spanning tree strategy to the graph in Algorithm 3. In the first variant we generate a random depth-first spanning tree using all edges from E and convert the strategy from the spanning tree by considering all cycle edges from E that are not in the spanning tree. This treats all regular and shady edges equally. In the second variant we modify the conversion of the spanning tree strategy to the graph by only considering cycle edges that are regular. This is equivalent to removing all shady edges that are not part of the spanning tree since they cannot lead to recontamination. Finally, we can also bias the generation of the spanning tree to only include regular edges for the depth-first traversal. This leads to more cycle edges that are shady and can then be removed for the second variant. Hence this bias is expected to improve the cost of strategies for the second variant and is equivalent to removing all shady edges. Note that the removal of a shady edge does not necessarily imply that strategies will get better since we are considering contiguous strategies. Contiguous strategies on graphs are generally more costly than non-contiguous strategies since contiguity is an additional requirement. Hence, removing an edge may prevent us to find a good contiguous strategy because it then turns into a non-contiguous strategy. Despite this potential effect we shall show in our experiments that one can generally expect an improvement when removing shady edges.

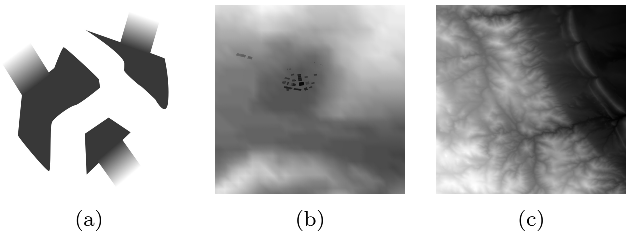

Fig. 3. (a) Sample map for testing with a three-way canyon, three plateaus with each its own ramp and several concave sections (843x768 cells). (b) Map of a small village with surrounding hills (798x824 cells). (c) Map of a mountain area located in Colorado, US (543x699 cells).

We present result on three maps seen in fig. 3. The resolution of (a) and (b) is 0.1units/pixel, and 10units/pixel for (c). Sensing ranges mentioned below are always measured in units. The height of cell in the map is given by its grey level and ranges from 0m to 10 units with 0 as white and 10 as black. Traversability classification as seen in fig. 4 is always based on a *Telemax* robot.

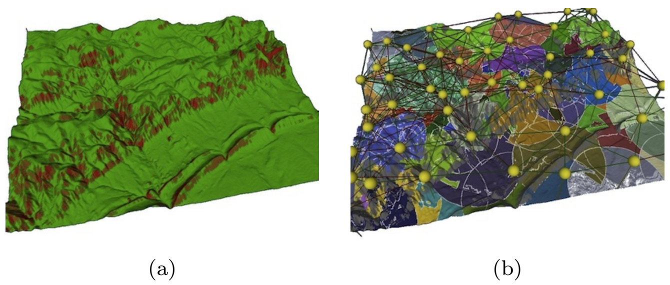

Fig. 4. (a) Terrain traversability on the Colorado map. Non-traversable regions are marked red. (b) Detection sets on the same map with according graph computed for robots with $h_r = 2.0$, $s_r = 10$, and $h_t = 1.0$.

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 yet be detected. Hence, all our tests with every configuration are run and averaged across 100 graphs generated via the random sampling within $\mathcal{E}$ Second, the strategy on the generated graph is computed by trying strategies on multiple random spanning trees. For this we conducted extensive tests to investigate the effect with our sample map seen in fig. 3. This is done with

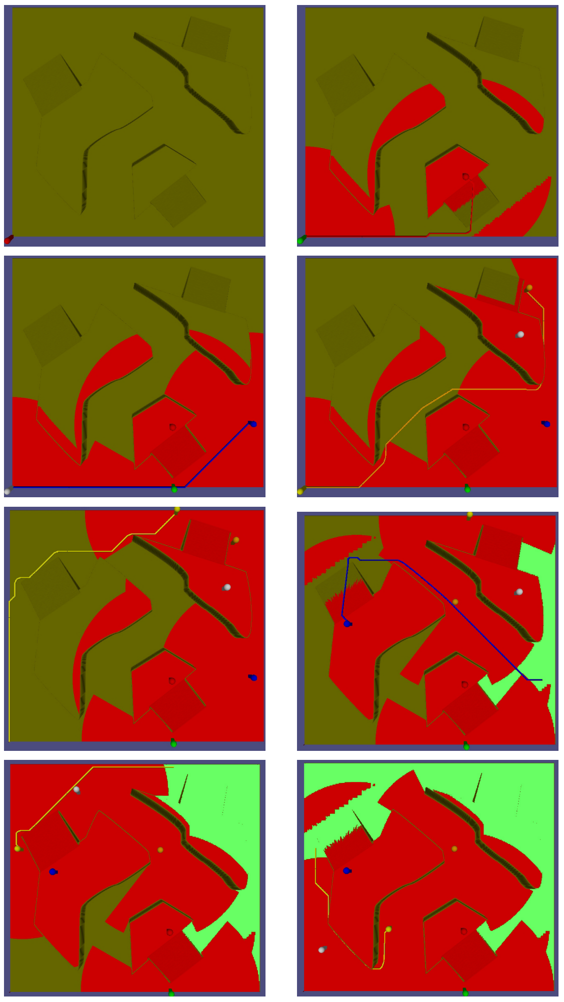

Fig. 5. A strategy for our sample map from fig. 3 with 6 robots. Detection sets are marked red and cleared areas not under observation are marked green. At step 0 on the upper left all robots are at their deployment location at the bottom left. The pictures show steps $0, 1, 3, 5, 6, 7, 10$ and 12 from 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.

the first variant that considers all edges and no bias in the generation of the spanning trees. 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 $s_r = 10$ 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

s_r	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.

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.

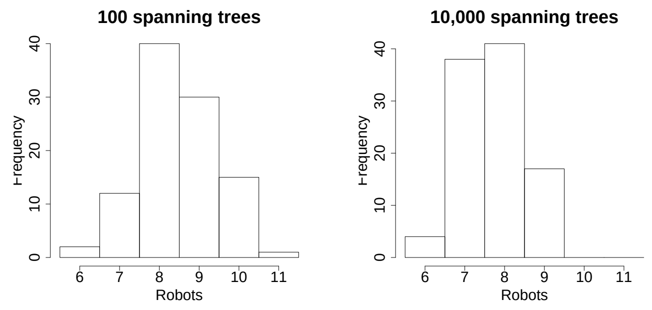

Fig. 6. Two histograms of the distributions of the number of robots needed for the 100 randomly generated graphs.

We also tested the algorithm on a realistic 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 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 1 to 0.5 requires 9 instead of 8 for the same sensing range and $h_r = 1$. A reduction of h_r from 1 to 0.5 requires 10 instead of 8 for the same sensing range and $h_t = 1$. Reducing both, h_t and h_r to 0.5 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.

The next question we investigated relates to the second variant of Algorithm 3 which only considers regular edges

s_r	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.

trees	v	bias	min	max	mean	$p-value$
100	1	no	7	12	8.73 ± 1.05	0.0324
100	1	yes	7	12	9.05 ± 1.16	
100	2	no	6	9	6.94 ± 0.56	<0.0001
100	2	yes	6	9	7.6 ± 0.61	
1000	1	no	7	11	8.54 ± 1.02	0.7855
1000	1	yes	6	11	8.58 ± 1.13	
1000	2	no	5	8	6.65 ± 0.43	0.0007
1000	2	yes	5	9	7.01 ± 0.66	
10000	1	no	6	11	8.51 ± 0.90	0.3894
10000	1	yes	7	10	8.4 ± 0.73	
10000	2	no	5	8	6.64 ± 0.45	0.2442
10000	2	yes	5	8	6.75 ± 0.43	

TABLE III
The results from experiments with $s_r = 30$, $h_r = 1$ and $h_t = 1$. The last column shows the $p - value$ from a standard T-test between two subsequent rows.

map name	s_r	Variant	min	max	mean
Sample map	10.0	2	12	17	14.3 ± 1.1
	10.0	1	13	20	16.5 ± 1.7
	20.0	2	6	17	7.5 ± 3.0
	20.0	1	7	19	9.2 ± 3.3
	30.0	2	5	9	6.7 ± 0.5
	30.0	1	6	12	8.3 ± 1.1
	50.0	2	5	8	6.0 ± 0.5
	50.0	1	6	10	7.8 ± 0.8
	70.0	2	4	8	5.9 ± 0.4
	70.0	1	5	10	7.9 ± 1.0
Village	10.0	2	15	20	17.2 ± 1.2
	10.0	1	16	23	19.1 ± 2.0
	20.0	2	9	14	11.0 ± 1.3
	20.0	1	10	18	13.6 ± 2.2
	30.0	2	7	12	9.5 ± 1.3
	30.0	1	9	15	12.1 ± 1.9
	50.0	2	6	12	8.8 ± 1.0
	50.0	1	8	15	11.6 ± 1.8
	70.0	2	6	11	8.6 ± 1.2
	70.0	1	8	16	12.1 ± 2.7
Colorado	10.0	y	12	17	14.1 ± 1.2
	10.0	n	13	20	16.3 ± 1.9
	20.0	y	11	18	14.6 ± 2.7
	20.0	n	12	20	17.1 ± 3.5
	30.0	y	12	22	16.9 ± 5.3
	30.0	n	14	25	19.1 ± 5.8
	50.0	y	14	27	19.8 ± 5.8
	50.0	n	16	30	22.0 ± 6.1
	70.0	y	15	26	20.2 ± 5.6
	70.0	n	18	29	22.8 ± 6.3

TABLE IV
Results of the experiments on the three maps from fig. 3.

as cycle edges as well as the bias on the spanning tree generation. A well chosen bias in the spanning tree generation can potentially speed up the discovery of a good spanning tree for the graph thereby reducing the number of trees that need to be tested. Furthermore, the bias should be more beneficial for the second variant than the first. Table V shows the results for these questions. They indicate that when generating 100 spanning trees the bias significantly improves the average across all strategies, although it does not have an effect on the minimum number of robots. With 1000 spanning trees we only see a significant improvement for the second variant when using the bias. The second variant, as expected, benefits more from the bias. Finally, when generating a larger number of spanning trees the positive effect of the bias diminishes. Comparing the minimum number of robots for the first and second variant in Table V shows a significant difference with a $p-value < 2.210^{-16}$ for all conditions. In all cases the minimum number of robots needed for the second variant is better by 2 or 3 robots. Also the variance of the cost of strategies across the 100 generated graphs is less than the variance for the first variant. Hence, the second variant is generally preferable. This applies particularly to larger graphs for which generating a large number of spanning trees is computationally expensive.

Finally, we tested variant one and two with a biased spanning tree generation on all three maps with sensing ranges from 10 to 70 as seen in Table V. Again variant two always outperforms variant one at all sensing ranges. This applies to the sample map and to more the realistic and very complex maps *Village* and *Colorado*. Most notably, as the sensing range increases in maps *Sample map* and *Village* the number of robots decreases, but in map *Colorado* it first improves slightly and then gets worse. Fig. 7 illustrates this. This is likely due to the more complex structure of *Colorado*. In this case an increased sensing range does not yield a much larger detection set, but a detection set with a more complex boundary due to many more occlusions. This complex boundary leads to many more edges in the graph. The plot in fig. 8 verifies that the number of edges increases for *Colorado* but not for the other maps as the sensing range increases.

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. We have pre-

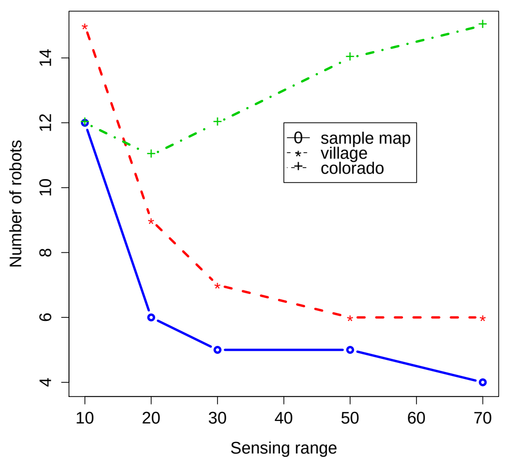

Fig. 7. A plot of the number of robots needed for the best strategy at a given sensing range for all three maps.

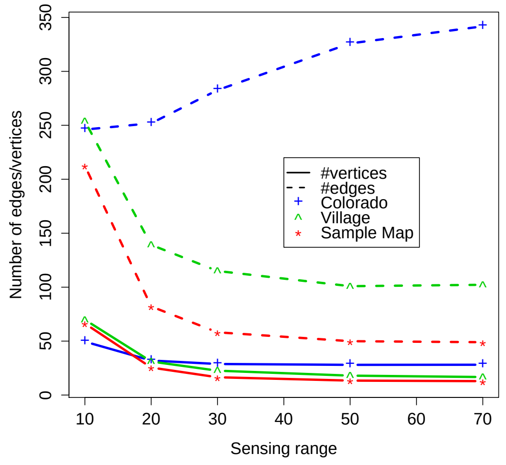

Fig. 8. A plot of the average number of vertices and edges for all three maps.

sented a novel graph structure that captures visibility information that arises in 2.5d problems and designed several variants that utilize this information differently. This graph model also poses a new set of questions for further work. One next step is to consider non-contiguous strategies which should lead to a significant improvement since all shady edges can be ignored without jeopardizing the graph strategy. Yet, despite this problem we have shown that discarding shady edges generally leads to better strategies. We have also demonstrated the effect of changing target and robot heights on strategies. Another important result relates to changes in the sensing ranges which have an effect that is highly dependent on the map. In complex maps a larger sensing range can lead to worse strategies. Finally, our approach allows us to identify a sensing range and robot height that leads to strategies that requires less robots for a particular map. 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 urban environments, such as streets and building walls.

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

[14] 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.

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

[16] 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.