2010 IEEE International Conference on Robotics and Automation
Anchorage Convention District
May 3-8, 2010, Anchorage, Alaska, USA

Multi-Robot Pursuit-Evasion without Maps

Andreas Kolling and Stefano Carpin

Abstract— **We propose a distributed algorithm enabling a large team of robots to detect all intruders within a large planar environment. Each robot can only detect intruders and communicate with other robots within a limited range. No map of the environment is given, and none is built during the process. Robots are only capable of following walls and other robots that are nearby. The algorithm puts together elementary behaviors giving robots the means to coordinate their movement in order to cover lines between opposite walls with their sensors and discover nearby new walls. A line has leading robots at its endpoints that follow walls and hence move the line of robots forward. Multiple such lines move through the entire assigned area in order to detect all intruders. The movement of multiple lines is coordinated by using a graph representation of the environment that describes possible line movements and their associated costs in terms of robots. This coordination requires only local communication between the leaders of different robot lines when they meet. Finally, we demonstrate how the algorithm can be implemented using elementary wall following and obstacle discovery behaviors.**

I. Introduction

The utilization of robotic systems for search, surveillance, and reconnaissance offers various advantages to the use of human personnel. Particularly, applications involving a significant risk such as search and rescue operations or surveillance of hostile territories benefit from robotic systems. Robots make these operations feasible by either being expendable, or designed to be more robust against the particular risk. Apart from reducing risk, they can also enable operations that are not cost effective otherwise, such as surveillance of large areas for security purposes. This relates to another important problem for the application of robotic systems for those tasks, namely cost-effective scalability. A multi-robot system in which each component is reasonably cheap, but as a consequence also less powerful, can help to solve this problem. The coordination of these components to achieve the required capabilities for the entire system is however not a trivial task.

In this paper we focus on the design of a distributed algorithm for the detection of all intruders within an unknown environment with a large team of robots. Hereby, each robot has limited capabilities and can only sense and communicate within a very limited range[1]. Intruders are assumed to be omniscient and capable of moving at unbounded speed. In particular, they are assumed to have constant and full knowledge about the robots' positions. This assumption of a worst-case adversary turns the task into a pursuit-evasion problem. In these, the possibility of an intruder being located in part of the environment is often represented by contamination, and the task of detecting all intruders becomes equivalent to clearing all contamination.

Given that robots only have limited sensing and communication range, we assemble them onto lines between obstacles to prevent intruders from crossing between them. We then move these robot lines through the environment to clear it from contamination and to discover new obstacles. The exploration is hence taking place in parallel to the clearing. The main parts of the algorithm consist of moving these lines forward and coordinating many such lines to clear the environment with a given number of robots, or to determine that more robots are necessary to complete the task. The coordination is achieved with the help of a graph representation of the already explored part of the environment. This graph is known as a surveillance graph and is described later on in more detail. No metric map is created during this process nor need the robots have the capability to create one.

The algorithm presented in this paper extends our previous work on on pursuit-evasion with limited range sensors and robots moving on lines [10]. Therein a map was required in order to extract a surveillance graph from it, and the graph was then used to schedule the movement of lines. In [6] the problem is more rigorously formalized and improved algorithms are presented that coordinate lines better but still require a map. In this manuscript we provide three original contributions that significantly generalize our former findings:

1) we remove the demanding requirement that a map has to be given up-front;
2) we identify the minimal capabilities needed for each robot to implement elementary building blocks to successfully clear an environment;
3) we provide implementations of these procedures, focusing in particular on wall-following, searching for new obstacles close to an existing line, and splitting an existing line into two new lines.

II. Related Work

The work presented in this paper connects to a variety of previous research, namely visibility-based pursuit-evasion, pursuit-evasion on graphs, and sweeping environments with lines. We will shortly discuss a small selection of papers from these three areas.

Andreas Kolling is with the School of Information Sciences, University of Pittsburgh, PA, USA.

Stefano Carpin is with the School of Engineering, University of California, Merced, CA, USA.

[1]The concept of limited range obviously depends on the environment that is being considered. In our context, limited range means that a single robot generally cannot cover the distance between any two opposite walls.

Visibility-based pursuit-evasion focuses on robots with unlimited range sensors in 2d environments. The body of work in this area is vast and includes many variations with respect to the exact type of environment, motion constraints, field of view of the sensor, and number of robots available. A great deal is concerned with finding a solution for only one robot, which can already clear rather complicated environments, thanks to the generous sensing range. Much of the research culminates into the paper [11] in which an online algorithm is presented that can clear an unknown, simply-connected, piecewise-smooth planar environment. The robot can only sense depth-discontinuities, has imperfect control, and follows only simple motion primitives. The approach incrementally builds a navigation graph based on the motion primitives. The information state about possible locations of the invader is superimposed on this graph forming the so called information graph. An online version is achieved by envisioning preliminary solutions in the information graph. It is a complete algorithm that enables the robot to clear the same environments than a pursuer with a complete map, perfect localization, and perfect control can clear. One of our goals is to present a similar online version for our scenario with limited range sensors and large robot teams by entangling exploration with our version of motion primitives to build a navigation graph on the fly while also envisioning temporary solutions. We do not borrow any techniques from this line of research, however, since the unlimited range assumption changes the problem significantly.

Pursuit-evasion on graphs is an even larger area of research and we shall only mention two relevant publications concerning weighted graphs, namely weighted edge-searching [1] and Graph-Clear [9]. Both problems are defined on a graph with weights on edges and vertices. The formulations of the two problems differ primarily in the way they address the avoidance of recontamination. In weighted edge-searching contamination is avoided by guarding vertices while in Graph-Clear edges are blocked instead. Details on the distinction are discussed in [9]. For our purposes we will use the Graph-Clear formulation to enable us to coordinate multiple moving lines using the graph built during the exploration. This is discussed in more detail in Section IV. For Graph-Clear we also have access to a probabilistic variant from [8] which enables the integration of a failure model for the target detection.

The idea of sweeping environments with robots on lines has also been discussed in [3], [5], [10] and [6]. In [3] robots operate in an environment without obstacles and attempt to trap a possibly faster intruders by forming a so-called trapping chain which surrounds and captures an intruder once its is discovered in the center of the chain. Very precise control laws that governing this behavior are given, and they enable a precise description of its properties. A similar description can be used for the line-following behavior in this paper along the line of [4]. This, however, remains subject to further work. More closely related is the work in [5]. Therein robots sweep the environment with one polygonal chain which requires that they are always mutually visible. Robots are required to have unlimited range. There is only one chain of robots and this chain sweeps the environment which is a simply-connected polygon. The algorithm to coordinate the motion of the robots in the line is based on a so-called link diagram which represents the minimum number of links a path between two points on the boundary of the environment. Our approach allows multiple lines and focuses on the coordination of when to split these. In general it can be applied to multiply-connected environments and associates a different cost to lines, one based on length due to the limited range and not on the number of links[2]. In this it is an extension of our previous work [10]. Therein the idea of sweep lines with a cost defined by length and the possibility of splitting these lines has been discussed. To coordinate line movement a Voronoi Diagram of the environment has to be computed and converted to a surveillance graph for Graph-Clear. This leads to a partitioning of the environment based on the associations of obstacles according to the Voronoi diagram. A formalization of these ideas as so called *sweep lines* is presented in [6]. The resulting problem to coordinate sweep lines to clear an environment with lowest cost is called *Line-Clear*. Based on the formalization it is shown that the algorithm from [10] does not yield an optimal coordination of sweep lines, a so-called sweep schedule, and improved algorithms are presented. The key part of these algorithms is the selection of obstacle boundaries on which to split the lines and to find an optimal sequence of these splits. In this paper we develop a method that coordinates sweep lines without requiring a map. In this case it is impossible to compute the sequence of splits in advance and instead they are discovered as the lines move through the environment. Additionally we present a routine that allows robots to follow sweep lines and thereby coordinate the motion of an actual robot team.

III. Algorithmic details

The goal of the algorithm presented in this paper is to coordinate the movement of robots along lines to simultaneously clear and explore an unknown environment. First, the deployed robot team has to find an obstacle boundary and form a first line starting and ending at this boundary. After this line is set up, robots move it until a new obstacle is encountered. When an obstacle is encountered the line is split into two new lines and robots have to choose which line to continue moving. If the team runs out of robots to cover the line and does not encounter a new obstacle, a search procedure is executed with which robots attempt extend their reach towards unknown obstacles. Finally, robots have to coordinate which line moves at what time and associate costs to a sequence of moves. In the following we will first define the capabilities of the robots that are needed for the algorithm, describe how robots move a line forward, and then describe the entire algorithm in more detail in the following sections.

[2]With unlimited range the number of links or bends in the line determines the cost in terms of robots.

We assume that robots are holonomic, can detect targets within a range r_{detect}, and communicate with other robots within a range $r_{comm} > 2 \cdot r_{detect}$. Further, they can sense obstacles and follow the boundary of an obstacle at distance $r_{obstacle}$ by sensing the tangent of the boundary. Finally, they can detect robot neighbors within a range r_{detect}, and determine their relative position. The desired distance between robots on a line is denoted by $r_{follow} \leq 2 \cdot r_{detect}$. Free space of the environment is bounded and is denoted by $\mathcal{E} \subset \mathbb{R}^2$. Let n be the number of all robots which are initially deployed together, i.e. so that their communication graph is connected. This allows the formation of a first sweep line that can then be moved forward. Given that robots closer than $r_{follow} < r_{comm}$ can communicate, we assume that robots forming a line with distance r_{follow} between neighbors can maintain a set of shared variables while operating.

A. Moving a line

The basic capability needed by robots is to jointly move a line forward. Lines are simply chains of robots that cover an area between two obstacles with their sensors, as in [10] and [6]. Fig. 1 shows a line covered by robots and additional robots following the line as a reserve. The robots on its endpoints control the forward movement of the line by following the obstacle boundary. More precisely, they sense the two tangents at the two obstacle boundaries between which the line of robots is spanned. From these tangents one can determine whether the length of this line grows or shrinks. Let us assume that the line of robots is spanned between points $l_1(t)$ and $l_2(t)$. At these points let the tangent along the obstacle boundary be given by $\hat{T}l_1(t)$ and $\hat{T}l_2(t)$ respectively for each endpoint. The relative angle between these tangents and the line determines whether a forward movement of l_1 or l_2 shrinks or expands the line. The rules for moving the line are straightforward. If both tangents determine a decrease in length, then both robots at the end follow that tangent. If one is decreasing, then only the robot at that end moves. If both are increasing, then only the one with less increase moves. Since robots can communicate with their neighbors on the line and sense their relative position, we can assume that they all have access to the relative positions of the end robots $r_1(t)$ and $r_2(t)$, and can use this information to position themselves on the line between $r_1(t)$ and $r_2(t)$. As seen in Fig. 1, robots that are currently not needed to cover the line, denoted as reserve, just follow and join in when the length of the line increases.

Robots moving along a line execute one of three algorithms, depending on their role. They execute Algorithm 1 if they are at one of the endpoints, denoted as line leaders, Algorithm 2 if they are in the center of the line, and Algorithm 3 if they serve as reserve of the line. Pseudocode for these algorithms is provided in Algorithm 1, 2 and 3 and they are here shortly commented. The shared variable r_{all} is the number of robots associated to the line, i.e. the number of robots currently running Algorithm 1, 2 or 3. The shared variable r_{line} is instead the number of robots running only Algorithm 1 or 2. Furthermore, each robot has access to its relative position in the line via p, i.e. it is the p-th robot in the line starting from the leftmost robot with $p = 1$. The function a returns the smaller angle between two lines; $wait()$ simply leaves the robot stationary; $move\_robot(to)$ moves the robot to point to; $move\_along(d)$ moves it in the direction d; $follow\_line()$ lets the robot follow r_2; $sense\_obstacles()$ returns true if a new obstacle is encountered and sets shared variable $new\_obstacle$ to true; $first\_in\_reserve$ returns true if the robot is the designated first amongst all those running Algorithm 3. Four functions trigger events that are caught on a higher level, $trigger\_dissolve\_line()$, $trigger\_search()$, $trigger\_line\_shrinks()$ and $trigger\_line\_grows()$ are executed when discrete events occur. They are respectively called when a line is no longer necessary, when a new obstacle must be sought because a line cannot be further stretched, and when a line shrinks by one robot or grows by one robot.

Algorithm 1 $Line\_Leader()$

```
if p = 1 then
  x ← 1, y ← 2
else if p = r_line then
  x ← 2, y ← 1
end if
while ¬new_obstacle do
  l ← r_y(t) − r_x(t)
  if a(T̂l_x(t), l) < π/2 OR a(T̂l_x(t), l) ≤ a(T̂l_y(t), −l)
  then
    move_along(l_x(t))
  else
    wait()
  end if
  if length(l) < ε then
    trigger_dissolve_line()
    return
  else if length(l) > (r_all) · r_follow then
    trigger_search()
    return
  end if
end while
return
```

Algorithm 2 $Line\_Center()$

```
while ¬new_obstacle do
  l ← r_2(t) − r_1(t)
  m ← (p − 1) · r_follow / length(l)
  if m > length(l) then
    trigger_line_shrinks()
    Line_Reserve()
  else
    g ← r_1(t) + m · l
    move_robot_to(g)
  end if
end while
return
```

Algorithm 3 $Line\_Reserve()$

```
while true do
  if first_in_reserve() then
    l ← r_2(t) − r_1(t)
    m ← length(l) / (r_line − 1)
    if m ≤ r_follow then
      trigger_line_grows()
      return
    end if
    follow_line()
  end if
end while
```

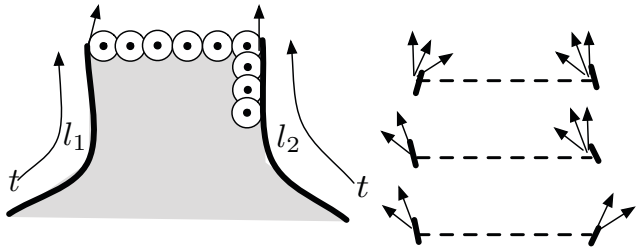

Fig. 1. A generic illustration of the main line moving forwards to extend the cleared area marked in grey. On the right hand side are possible configurations for the tangent of the line. A tangent at an endpoint away from line leads to its length increasing while a tangent inwards leads to a decrease. The three cases depicted are one where 1) both sides lead to decrease, 2) one side leads to a decrease and one to an increase 3) both sides lead to an increase in length. On the left figure some robots in the gray area serving as reserve are shown.

B. Obstacle Search

The purpose of an obstacle search is to extend the reach of the robots that form a line. It triggers when the robots try to move a line forward but cannot continue due to an increase in its length and a lack of robots in the reserve. It entails moving the sweep line backwards to a narrower position between the two obstacles in order to free up robots that then join the reserve. Once at least one robot joins the reserve, a search is executed. Fig. 2 illustrates how additional robots can be used to split the original robot line into two line segments, s_l on the left and s_r on the right, to extend the reach. All possible such extensions can be described by the number of robots allocated to each segment. Given r_{all} robots and r_{line} robots required for the original line, we can allocate $i+1$ robots to s_l with $i \in \{1, \ldots, r_{all}-2\}$ which leads to $r_{all}-i$ robots on segment s_r. Note that one robot is the end point on both segments. Fig. 3 shows the simple trigonometry involved in finding the outmost position for every $i \in \{1, \ldots, r_{all}-2\}$. Since the lengths of the triangle formed by each point p_i are known, we can find p_i and move the robots to cover s_l and s_r for every choice of i. Even though method depicted in the figure ensures that we get extended reach for a given number of robots in the line, we are not able to compute the best positions for the original line. Hence, it is desirably to search through all p_i every time another robot becomes available. However, we cannot claim this ensures that an obstacle will be found even if it is theoretically possible to reach one with the available number of robots while maintaining a line between obstacles l_1 and l_2. If the backwards movement of the line does not free up a robot but requires one more, then the search has failed and it triggers a search failure.

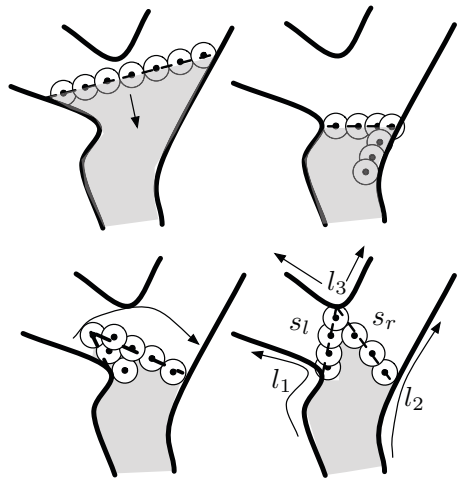

Fig. 2. Top left: a line runs out of robots and cannot move further (the arrow indicates how it will move back). Top right: it then moves back. Bottom left: it then searches for a new obstacle. Bottom right: the line is split in two and each of them can individually move.

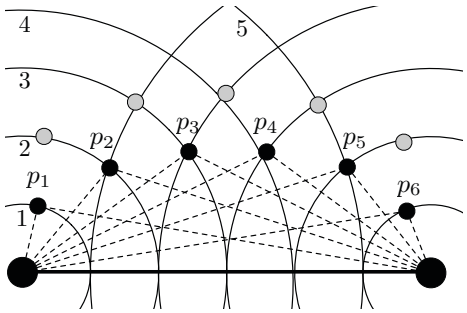

Fig. 3. The radii of the circles are multiples of r_{follow}. The line from which the search originated is marked as a thick black line with thick circles representing the left and right line leader. The original line requires 7 robots and the small black dots show which points can be reached with a total of $r_{all}=8$ robots by separating them onto the two new line segments into, i.e. for $i=1$ there are $i+1$ robots on the left segment and $8-i$ robots on the right segment. The grey circles indicate the points that could be reached if we had $r_{all}=9$.

If the extensions or the search procedure is successful and a new obstacle is encountered the line splits into a left and right side. Each side needs sufficiently many robots to be able to span a line between the found point on the third obstacles l_3 and l_1 and l_2 respectively as seen in fig. 2. The team can try to improve both sides of the split locally by moving the endpoints towards the direction into which the line length is decreasing. A split is also triggered when a robot encounters a new obstacle while following a line. If the search procedure is not successful, then the robots track back to the previous split. Executing a split and finding a previous

split is discussed in IV. Since all the previous motions are reversible moving to the previous split is possible.

IV. Surveillance Graphs and Line Coordination

The procedures introduced in the previous section can been seen as local behaviors of the robot team that enable the movement of robot lines through the environment and as such are motion primitives. As the robot team moves lines and searches for obstacles, it explores the environment. The exploration is captured by a surveillance graph that represents the discovered topology of the environment, and the cost in terms of robots to form a line and moving it forward. We can associate the motion primitives to this surveillance graph and construct it on the fly. Here we discuss how to create such a surveillance graph and how to use it to coordinate the motion primitives and scale them to clear a larger environment. Formally, a surveillance graph is a triple $G = (V, E, w)$ with undirected edges and the weight function w defined on both edges and vertices $w : V \cup E \rightarrow \mathbb{N}^+$. Two types of actions can be executed on the graph, namely *blocking* on edges and *sweeping* on vertices. The state of G is given by a description of which vertex and edge is clear, contaminated or blocked. A sweeping action clears a vertex while a blocking action clears an edge and prevents contamination from spreading through it. A surveillance graph that represents all previous line movements and their costs is easily constructed as follows. First, a vertex is created and given the maximum cost of the starting line. Every successful encounter of a third obstacle leading to a split creates three new vertices. The first receives as cost the number of robots that are needed to execute the split on the third obstacle into two sweep lines, and is connected to the graph trough an edge added to the previous vertex the line encountered with the weight of the minimum line width[3] between these two positions. The next two vertices for the encountered obstacle are created for each new line and receive as weight the cost of the respective new line. For each of these new vertices, an edge to the first vertex of the encountered obstacle is added, and it receives the same weight as the vertex. Furthermore, a number of consecutive increases in the number of robots while moving the line forward followed by one decrease creates a vertex, representing the cost of passing through a narrowing region. Naturally, edges are created for every two vertices that are encountered consecutively by one moving line. Their weight is always given by the number of robots needed to cover the shortest line encountered during the movement between vertices. We call this line *blocking line*, since it represents the cost of a blocking action of the surveillance graph since it has the lowest cost of preventing contamination between the vertices with a robot line. Built this way, the surveillance graph is a record of the line movements. A traversal of a vertex in the graph can be associated to moving a line from a blocking position to either another blocking line or a split. It hence captures the motion primitives that the robot team can jointly execute. Fig. 4 illustrates this idea and shows how vertices and edges are created as lines move and split.

[3]Technically, this is the number of robots, however line width and number of robots necessary to cover it are proportionally related, so the two terms are considered synonym.

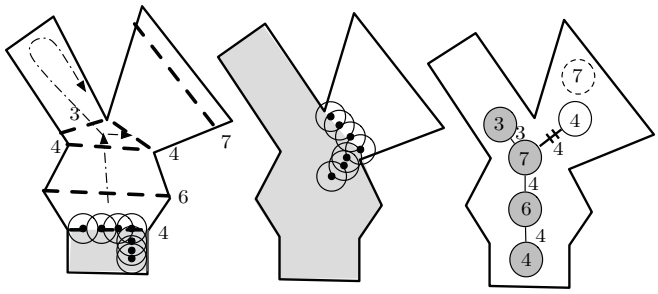

Fig. 4. Example of the construction of a graph as a result of the exploration with lines. The cleared and known vertices are marked in grey. The lines leading to their creation are marked as thick dashed lines with the associated number of robots needed. The robots explore the environment, following the arrows and stop before discovering the vertex with weight 7. Hence the neighboring vertex 4 is not considered explored.

Given a sufficiently large number of robots for the environment being cleared, one can incrementally discover the whole graph by just choosing to continue on an arbitrary side, left or right after a split, and coming back once the side is entirely cleared. If a successful search is triggered, the robot team can similarly continue. However, the sequence in which vertices are encountered can be expected to lead to a much higher cost in terms of robots than the best possible sequence. In other words, the discovery of vertices leads to a sequence in which vertices were cleared, a so-called strategy, and this strategy is not necessarily a sub-strategy of an optimal strategy of the entire, partially unknown, graph. The discovery of more vertices may require all previously cleared vertices to be recontaminated to reach a state of the graph that is in fact a state reached by an optimal strategy. In the worst case the discovery of the last vertex can require the recontamination of all previously cleared vertices. This is an unavoidable problem, also occurring in [11] which also computes and executes partial solutions that can require to start from scratch once the entire environment is explored.

An overview of the algorithm is given in fig. 5. The exploration consists of moving only one line, denoted as the main line, which has all available robots in its reserve. Once a split is encountered the robots choose with which direction, s_l or s_r (left or right), to continue as the main line while the other side will remain stationary until the robots from the main line return and activate it. Once the length of the main line reduces to zero, i.e. the two endpoints meet, it is dissolved and all its robots follow either the left or right wall to go back to the next stationary line. The same happens when the main line encounters a stationary line which joins the robots of both lines and sends them to the next stationary line. For now we shall ignore this event and merely note that it can be dealt with by adding a cycle edge into the graph and then simply dissolve the two lines consecutively. Every stationary line is always associated to an unexplored vertex and hence one can find the next stationary line by searching through the graph. As vertices are added during

the exploration they are marked as unexplored until the line passing through them has lead to the addition of another vertex or triggered a dissolution of the line. The choice of the next stationary line to become a main line after a split is in principle arbitrary, but as a heuristic one can choose to extend the line with more robots. Similarly, after the dissolution of the line the choice is also arbitrary. As a heuristic we minimize the graph distance, but other criteria can be applied, such as picking the longest stationary line.

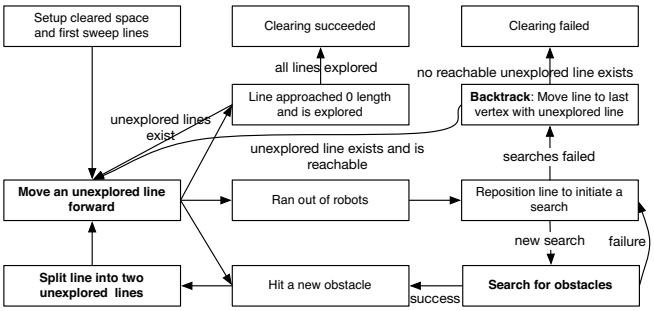

Fig. 5. A diagram of states of the robot team.

If this process continues without unsuccessful obstacle searches occurring, it is in principle a depth first traversal of a tree, if the environment is simply connected, or a graph, if it is multiply-connected. If the main line runs out of robots and cannot proceed, it initiates an obstacle search. If the search finds an obstacle the main line splits and continues as before. If it is not successful, the robots continue moving the main line backwards until they reach the cost of the last blocking line to free up as many robots as possible without recontaminating a vertex. The main line then becomes stationary and the remaining robots choose the next closest stationary line on the graph and attempt to extend it. Once all attempts to extend every stationary line have failed, the robots need to resort to allow recontamination of already cleared vertices. This enables further exploration by freeing up robots that are currently locked into stationary lines. The Graph-Clear problem, defined on surveillance graphs, is to find strategies that have minimal cost and hence use the least robots. This problem is shown to be NP-hard in [9], but for surveillance graphs that are trees we can compute optimal strategies in polynomial time using the algorithm presented in [9]. For graphs one can adapt strategies that are created by removing all edges of cycles and then uses these as presented in [7], but these are not guaranteed to be optimal.

Given the partially explored graph we need to pick an unexplored vertex and then compute a state of the partial graph that allows more robots to be freed so that they can join the main line. There is a variety of possibilities how to achieve this. The brute force approach would be to recontaminate all known vertices and move all robots to the chosen unexplored vertex. Obviously, a totally contaminated state of the graph does not require any robots to block contamination and hence frees all. To avoid as much recontamination as possible one could resort to the algorithm in [9] for optimal strategies on trees. It computes so-called cut sets that are associated to edges and can be used to identify states of the graph which have low blocking cost but also a larger number of cleared vertices. Unfortunately, in the worst case it may well be that the unexplored vertex has a cost that requires all robots. The environment could still be cleared with the given number of robots, but only if starting at the high cost vertex. With plenty of excess robots and if the total time needed for clearing is an issue, then cut sets should be used to identify which vertices to recontaminate and free robots one by one. But for our current purposes the brute force approach suffices.

The algorithm could run on one main robot that collects all the information and takes part in the movement of every line. But it may also be done distributively with relevant information being transferred between robots when they meet the main line. Its primary function is to catch discrete events that occur as the lines are moving and update the graph structure and determine which line is moving next.

V. Implementation and Results

Most of the procedures presented above can be implemented in a variety of ways, depending on the robot platform. In particular, if the robots are non-holonomic, wall and line following behavior has to be designed differently. To demonstrate the feasibility of the proposed algorithm we implemented a specific wall following routine that utilizes a laser sensor to estimate the obstacle tangent. Robots are simulated with the Player/Stage software [2] and each robot ran its own control program communicating with others through the network interface. The range of the laser is set to $0.7m$ and the desired distance between robots is $1m$ allowing for an overlap of $0.4m$ to accomodate for synchronization and control errors. Fig. 7 shows the environment which is a $457x458$ pixel bitmap at a resolution of $0.032m$ per pixel leading to $14.5m$ width and height. It also shows the resulting surveillance graph of a test run that successfully cleared the environment with 9 robots and is presented in the accompanying video. For this run robots are placed at the bottom of the environment. Starting at the same spot with 8 robots leads to a failure for the first obstacle search that is seen executed with 9 robots in the accompanying video. The algorithm from [10] applied to the same map with equivalent sensing range yields a solution that requires 7 robots. Hence, the cost of not knowing the optimal starting position and optimal positions for splits is two additional robots. Running the algorithm with robots initially placed on the top right, however, clears the environment successfully with 8 robots. Reducing the number of robots to 7 leads to problems with final obstacle search on the very top of the environment. It succeeded in one out of three trials with 7 robots due to position of the line when the obstacle search was initiated. It is hence possible to clear this environment with 7 robots if one resets the starting position during clearing and retries multiple obstacle searches. Needless to say, the surveillance graph created with 7 or 8 differs from the one created by 9 robots since the exploration proceeds differently when more robots are available. This can be seen in the top part of the

environment for which 9 robots leads to a vertex of cost 9 while 8 robots created an additional vertices by executing one more split.

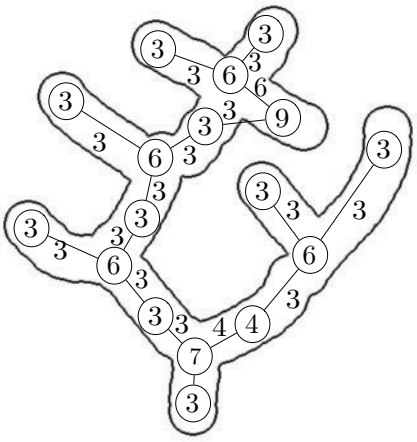

Fig. 6. The simply-connected sample environment and its associated surveillance graph that was created by clearing the environment with 9 robots placed at the bottom of the environment.

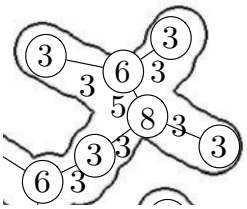

Fig. 7. The surveillance graph created for 8 robots of the top part of the environments shown in the figure differs from the one with 9 due to an additional split that occurs as a result from an obstacle search.

The algorithm can deal with cycles. If a cycle is present a stationary line will be met by a moving line. The latter can treat the stationary line just like an obstacle and once the moving line moved past or dissolved on it the stationary line is dissolved and its robots merge into the reserve. For every cycle encountered the associated surveillance graph also receives a cycle. The complications arising from cycles for strategies on the graph are identical to those of Graph-Clear and Line-Clear, as discussed in [6]. The experiments presented are a qualitative investigation and encourage further improvements of the methods presented here. A companion video offers more details about the different stages of the algorithm.

VI. Discussion and Conclusion

This paper presents the first attempt to connect the rigorous pursuit-evasion theory on graphs with low-level behaviors executed with robots with minimalist capabilities. As such, there are numerous improvements and future developments that are readily envisioned for this work. On the practical side the weights of the graph, particularly the costs of the split, should be improved by locally optimizing the split position. An important question here is whether one can improve the algorithm to always find the optimal split that is found by the algorithms that are provided with a map. Similar questions can be asked about the obstacle search and backtracking. We can now attempt to design these parts of the algorithm with the goal that they provably guarantee to discover the same strategies that the algorithms with known maps from [6] compute. The hardness of this question remains open for now, but the preliminary results from simulations are encouraging. Obviously, a more rigorous formulation as well as improvements to the methods presented here are required before such formal claims can be made and proven. Once such a formalization, potentially in form of a hybrid algorithm, is developed then a suite of experiments can be carried out to quantify the trade-off in terms of recontamination and longer searches between algorithms with and without a map and to validate proven claims in practice.

Furthermore, we presented a minimalist approach that does not require robots to localize themselves nor create a map. Only walls and neighbors need to be sensed. In this sense the contribution of this paper is also a practical one since it enables a robot team with minimalist capabilities and without a map to attempt to clear an environment. The general ideas of the algorithm can be applied to any robot team so long as they can form lines between obstacles, move these lines, can communicate locally and sense obstacles. One modification of interest is to adapt the basic behaviors and implement them for non-holonomic platforms.

References

[1] 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*, pages 200–209, New York, NY, USA, 2002. ACM Press.

[2] G. Biggs, T. Collett, B. Gerkey, A. Howard, N. Koenig, J. Polo, R. Rusu, and R. Vaughan. Player/Stage project. http://playerstage.sourceforge.net, 2005.

[3] S. D. Bopardikar, F. Bullo, and J. P. Hespanha. Cooperative pursuit with sensing limitations. In *American Control Conference*, pages 5394–5399, 2007.

[4] F. Bullo, J. Cortés, and S. Martínez. *Distributed control of robotic networks*. Applied Mathematics Series. Princeton University Press, 2009. To appear. Electronically available at http://coordinationbook.info.

[5] A. Efrat, L. J. Guibas, S. Har-Peled, D. C. Lin, J. S. B. Mitchell, and T. M. Murali. Sweeping simple polygons with a chain of guards. In *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms*, pages 927–936, 2000.

[6] A. Kolling. *Multi-Robot Pursuit-Evasion*. PhD thesis, University of California, Merced, December 2009.

[7] A. Kolling and S. Carpin. The GRAPH-CLEAR problem: definition, theoretical properties and its connections to multirobot aided surveillance. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1003–1008, 2007.

[8] A. Kolling and S. Carpin. Probabilistic Graph-Clear. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 3508–3514, 2009.

[9] A. Kolling and S. Carpin. Pursuit-evasion on trees by robot teams. *IEEE Transactions on Robotics*, 2009. accepted for publication.

[10] A. Kolling and S. Carpin. Surveillance strategies for target detection with sweep lines. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5821–5827, 2009.

[11] S. Sachs, S. Rajko, and S. M. LaValle. Visibility-based pursuit-evasion in an unknown planar environment. *The International Journal of Robotics Research*, 23(1):3–26, January 2004.