Scalable and Practical Pursuit-Evasion

Marcos A. M. Vieira
Department of Computer Science
University of Southern California
mvieira@usc.edu

Ramesh Govindan
Department of Computer Science
University of Southern California
ramesh@usc.edu

Gaurav S.Sukhatme
Department of Computer Science
University of Southern California
gaurav@usc.edu

Abstract— In this paper, we consider the design and implementation of practical, yet near-optimal, pursuit-evasion games. In prior work, we developed, using the theory of zero-sum games, minimal completion-time strategies for pursuit-evasion. Unfortunately, those strategies do not scale beyond a small number of robots. In this paper, we design and implement a partition strategy where pursuers capture evaders by decomposing the game into multiple multi-pursuer single-evader games. Our algorithm terminates, has bounded capture time, is robust, and is scalable in the number of robots. In our implementation, a sensor network provides sensing-at-a-distance, as well as a communication backbone that enables tighter coordination between pursuers. Our experiments in a challenging office environment suggest that this approach is near-optimal, at least for the configurations we have evaluated. Overall, our work illustrates an innovative interplay between robotics and communication.

I. INTRODUCTION

We are motivated by practical problems in security and monitoring for large, structured, spaces (e.g., to ensure the integrity of a large building or complex). The problem we focus on is pursuit-evasion wherein robots must pursue and catch evaders.

In Pursuit-Evasion Games (PEGs), multiple robots (the pursuers) collectively determine the location of one or more evaders, and try to corral them. The game terminates when every evader has been corralled by one or more robots. Several versions of the problem exist. In certain frameworks, it is acceptable to merely "sight" an evader for it to be "located", in others, a precise coordinate must be reported. Other formulations insist on a certain speed of convergence with fewer constraints on accuracy. Finally, formulations vary depending on whether the multi-robot control algorithm is required to have provably correct behavior, whether the number of evaders is known a priori, and whether they are malicious or benign. Each variation of the problem brings with it a different set of challenges, and several of these variations have been solved to varying degrees.

We consider the class of PEGs played on a discrete graph. Specifically, we use a topological map of the environment,

whose nodes correspond to coarse-grained regions and whose links connect neighboring regions [1, 2]. Discrete graph based games are acceptable for many uses of pursuit-evasion (e.g., surveillance, finding survivors). Of course, the physical multi-robot games run on a continuous space, but we discretize the environmental for our localization and game model.

In this paper, we consider a version of the game in which p pursuers collectively attempt to capture r evaders (Section II). We are interested in the convergence time of the game (i.e., the minimum number of steps for the pursuers to capture the evaders). We decompose the multi-player game into multiple multi-pursuer single-evader games. We prove that our algorithm terminates, has bounded captured time, is robust, and is scalable in the number of robots, being suited for practical applications. Based on our previous work, where we have designed the optimal policy that pursuers should use in order to capture evaders with the minimum number of steps, we design an assignment algorithm that optimally decomposes the game.

An embedded network provides sensing, communication and computational resources to the robots. The pursuers make use of the resources of an environment-embedded network, wherein robot sensing and communication is enhanced by the network, to have full knowledge of the game.

We present results from running an implementation of our algorithm on a physical robot testbed (Section V). Please refer to [3] for more details about this paper.

II. ASSUMPTIONS, TERMINOLOGY AND DEFINITIONS

In this section, we start by stating the sensing and communication assumptions for our PEGs, then discuss the class of games we are interested in. We then lay down some terminology, and formally define the objective of our PEGs. This sets the stage for our main contribution, a scalable near-optimal algorithm for PEG, which is discussed in the next section.

We focus on PEGs in bounded, spatially complex, environments similar to today's office environments. Because such environments are obstructed, they present limited line-of-sight visibility. However, it is increasingly true that such environments are well provisioned with wireless communication capability, and that many such environments will likely have dense embedded sensing.control).

In this paper, we assume such *network-assisted* environments; these environments provide sensing-at-a-distance to circumvent line-of-sight limitations. Moreover, they provide a

network communication capability that enables much tighter coordination than would have been possible otherwise. More specifically, the network a) contains sensors that are able to *approximately* localize all participants, and b) provides a communication backbone that enables participants to exchange game state.

Based on these assumptions, we consider the class of PEGs in which all participants have complete (but possibly imprecise) knowledge of the positions of all participants. We are interested in the class of games where enough pursuers (we make this more precise in the next section) exist to guarantee termination. Finally, we also assume that pursuers and evaders move at the same speed (more precisely, we assume that they move exactly one hop in the topology at each time step); we have left a relaxation of this assumption to future work.

Within this framework, we are interested in the optimal strategy that pursuers and evaders should play, where our measure of optimality is the capture time (defined below). Before we discuss this, we lay down some terminology.

Let $G = (V, L)$ be a finite connected undirected graph with V vertices and L links or edges. There are two sets of players called pursuers P and evaders E. Initially, P and E occupy some vertices of G. In describing the algorithm, we assume that time is discrete and increments at steps of 1; in our implementation, of course, we make no such assumption. At each time step, all pursuers and evaders are given the positions of all participants. Both teams play a game on G according to the following rule. At each step, each pursuer chooses a neighboring vertex of G to move to, then the evaders do the same. They then move to the corresponding vertex in G, as defined in [4], and repeat the previous step. The team of pursuers P wins if it "captures" all evaders. If an evader can avoid capture indefinitely, then the evader team wins the game. In the literature [5], the necessary number of pursuers to capture an evader in a graph G is denoted by $(c(G))$.

Let $p=|P|$, $r=|E|$, $v=|V|$. Let P_i be the current position of the ith pursuer and E_i be the current position of the ith evader. The tuple $a = < P_0, \ldots, P_p, E_0, \ldots, E_e >$ represents the current position of all participants. We define a boolean variable $T(turn)$ to denote if it is the pursuers' turn to move or not (recall that, in our algorithm, pursuers and evaders alternate at each time step). We say that the tuple $< a, T >$ encodes the *state s* of a game.

We can now define our game more formally as follows:
Input Coarse estimated positions of p robots and r evaders in a bounded environment E.
Output Motion commands for p robots.
Goal Minimize the capture time of the evaders.
Restriction No motion model for the evaders available to pursuers.

The game terminates when a *capture state* is reached. In a capture state, at least one pursuer occupies the same vertex in which an evader resides. There exists a different definition of termination: if, during the evolution of the game, a pursuer reaches an evader's position, the evader exits the game. Our definition results in a game that is strictly harder than this variant.

III. PURSUIT-EVASION STRATEGIES

In defining the game, we have avoided mention of the particular strategy that the pursuers and evaders use. In this section, we discuss this strategy, which is the main focus of our paper. We start by discussing the optimal strategy, which is computationally intractable. We then discuss a computationally feasible strategy that, as we show experimentally is near-optimal.

A. Optimal Strategy

In our game, we have assumed that both pursuers and evaders have complete information about the positions of all players. Consider now the *optimal strategy* for the pursuers and evaders: for the former, to capture the evaders in the shortest time, and for the latter, to avoid capture for the longest possible time. To formalize this intuition, we turn to zero-sum games.

Pursuit-Evasion is a zero-sum game since the pursuers' gain or loss is exactly balanced by the losses or gains of the evader. The evader's goal is to escape as long as possible whereas the pursuers have to capture the evaders as fast as possible. Zero-sum games have been extensively studied in the game theory literature, and our solution models a PEG as a zero-sum game that uses the minimax algorithm [6]. This algorithm minimizes the maximum possible loss for each player in the game.

To describe this algorithm, consider first that the evolution of any PEG can be represented by a game graph, a directed graph with possible cycles. The start state (as defined by the starting configuration of the pursuers and evaders) has a directed edge from itself to all possible next states that the pursuers can make from the start state. (In our game, we assume that pursuers and evaders move alternatively). In turn, from each of these states, there is a directed edge to all possible next states resulting from evaders' moves from that state. The graph can thus be recursively defined. In general, a game is a traversal on this graph. If this traversal ends in a capture state, the pursuers win the game. However, it is also possible for the traversal to repeat states: such a traversal will result in a non-terminating game and the evaders win.

We construct the game graph by generating all states and all possible transitions between states. For each state, we can calculate the cost to reach a capture state using a bottom-up approach. Then, this is the strategy that pursuers and evaders follow.

> $\mathscr{S}$. Each pursuer's strategy is to move to the vertex dictated by that neighboring state whose cost is least. Each evader's strategy is to move to the vertex dictated by that neighboring state whose cost is most among all neighbors.

In previous work [7], we proved the following theorem:

Theorem 3.1: For any given topological graph G and any given initial configuration of pursuers and evaders, $\mathscr{S}$ terminates in the minimal number of steps.

In practice, to play the game, we first pre-compute a complete state transition diagram offline. This is pre-loaded on all

the robots, and each pursuer or evader makes a decentralized local state transition decision, given the current state (the positions of all the robots), to calculate next state.

The complexity of this precomputation is $O(v^{2(p+r)})$. Unfortunately, the computational cost of enumerating the state transition diagram is not practical; for instance, using a modern desktop, we have been unable to compute the robot's strategy for 9 robots.

This motivates the work presented in this paper. In the next section, we present a scalable algorithm, where we partition a PEG into multiple multi-pursuer, single-evader games.

B. Partition Strategy

In our partitioning strategy, the pursuers divide the evaders amongst themselves and play $c(G)$-1 sub-games (recall that $c(G)$ is the minimum number of pursuers required to guarantee termination on a graph G). In each of these sub-games, the pursuers and the evader each play the optimal strategy discussed in Section III-A, and the goal is still to minimize the time to capture the evader. The key insight is that computing the state transition diagram for these partitioned games is computationally feasible. This section discusses an assignment algorithm that allocates $c(G)$ pursuers to each evader.

Our assignment algorithm assumes $p >= c(G)*r$, i.e, that the number of pursuers is at least that required to ensure that $c(G)$ pursuers can be assigned to each evader. The inputs to the assignment algorithm include the graph G and the initial positions of all pursuers and evaders. The assignment algorithm outputs for each pursuer which evader to pursue (and eventually capture).

We model the assignment problem of r teams and r evaders as a matching problem. Consider the bipartite graph $G = (T,E,L)$. The set T contains nodes, each of which represents a *team* of pursuers: each team represents a distinct combination of pursuer robots. Each node in the set E represents a single evader. Finally, each edge $l = (u,v)$ from the set L represents the assignment of team u to evader v. Each edge l has a cost c_{uv} which is the time to capture evader v with team u. It is possible to compute c_{uv}, the expected time to capture evader v by team u, by evaluating a $c(G) - 1$ game. Realize that we can pre-compute this cost and we only need to verify one game to know the cost of every position. We represent the cost c_{uv} in a matrix C.

The goal is to find the assignment that minimizes the maximum time to capture all evaders.

An assignment can be represented as a matrix $X = [x_{ij}]$ where x_{ij} is equal to 1 if team i is assigned to evader j, and equal to 0 otherwise. The assignment problem is written formally as follows:

$$\min \quad \max_{i,j=1,..,N} c_{ij}x_{ij}$$

$$\text{subject to} \quad \sum_{j=1}^{n} x_{ij} = 1 \quad \forall\ i \in N$$

$$\sum_{i=1}^{n} x_{ij} = 1 \quad \forall\ j \in N$$

$$x_{ij} \in \{0,1\} \quad \forall\ (i,j) \in N\ .$$

This problem is called *linear bottleneck assignment* (LBAP) [8] and can be solved optimally by any of the polynomial-time algorithms based on network flow theory [9].

The assignment algorithm described above is summarized in Algorithm 1.

Algorithm 1 Assignment of pursuers to evaders

1: Compute a cost metric for a $c(G) - 1$ game.
2: Generate all possible team configurations, with r teams which has $c(G)$ pursuers and one pursuer does not belong to more than one team.
3: **for all** feasible team configuration **do**
4:     calculate matching cost as a LBAP.
5:     update assignment with minimum max game cost.
6: **end for**
7: **return** min max assignment.

Algorithm 1 assigns $c(G)*r$ pursuers to r evaders. If $p > c(G)*r$, the unassigned pursuers chose randomly which evader to capture. This adds robustness to our algorithm in case of robot failure.

Our algorithm solves the global optimization problem described above even with the restriction that a pursuer can participate in at most one team. Would a simple greedy algorithm have worked? A greedy assignment strategy can result in a non-terminating game. Consider a $2-2$ game where the $c(G) = 1$ and the cost matrix $C = \begin{bmatrix} 1 & 2 \\ 3 & \infty \end{bmatrix}$. The optimal assignment that minimizes the time to capture of all evaders for this matrix is pursuer 1 to evader 2 and pursuer 2 to evader 1, which gives max time to capture $c = 3$. The greedy assignment would instead assign pursuer 1 to evader 1 and pursuer 2 to evader 2, with cost $c = \infty$. Figure 2(a) illustrates such a game.

Each pursuer robot runs the assignment algorithm locally. The inputs to the algorithm are the current positions of all pursuers and evaders; this information is obtained from the network (and may, of course, be inexact because of sensing noise). Each pursuer executes the assignment only once, and sticks with the assignment until the game terminates.

1) Complexity: The number of evaluated team configurations x can be modelled as the number of ways to put p distinct balls in r identical boxes. Each distinct ball represents a pursuer and each identical box represents an evader. The boxes are identical because we do not need to determine the identity of the evaders, the matching will determine this. Moreover, each box should have $c(G)$ balls. Hence, the number of configuration x we have is $x = \frac{p!}{(c(G)!)^r*(r!)}$.

Given that $n! \simeq n^n$ (Stirling's approximation), we have $x \simeq \frac{p^p}{(c(G))^{c(G)*r}*(r^r)}$. If $p = c(G)*r$, we have the number of evaluated configuration $x \simeq r^{p-r}$. The complexity of the Linear Bottleneck Assignment Problem (LBAP) with N nodes is $O(N^2)$ [10]. Thus, this part of our algorithm is $O(x^2)$. We also need to evaluate a $c(G) - 1$ game to know the cost metric, which has complexity $O(v^{2(c(G)+1)})$ (from Section III-A). Thus, our overall complexity is $O(x^2 + v^{2(c(G)+1)})$, where $x = \frac{p!}{(c(G)!)^r*(r!)} \simeq r^{p-r}$ instead of previous $O(v^{2(p+r)})$. Since

$r << |V|$, this algorithm is significantly more computationally efficient.

2) Properties: Here we enumerate the properties of the partition strategy.

Termination: Lemma 3.2 guarantees game will terminate, assuming no robot fails.

Lemma 3.2: Our algorithm guarantees that the $p - r$ game will terminate.

Proof: Since we decompose the $p - r$ game into r parallel $c(G) - 1$ sub-games, and each of these games is guaranteed to terminate by Theorem 3.1, the overall $p - r$ game terminates. ∎

Optimality of partitioning: Our algorithm optimally partitions the evaders across the pursuers, subject to the cost metric. This property follows from the optimality of the LBAP assignment algorithm.

Bounded capture time: The completion time for the $p - r$ game is the maximum completion time across the $c(G) - 1$ sub-game. However, is it still an open question how far off from the optimal completion time this partition strategy is.

Scalability: Our algorithm scales better than the optimal strategy, since its scaling is dominate by r^{p-r}, while the optimal scales as $|V|^{2(p+r)}$, and usually $r << |V|$.

Robustness: If $p > c(G) * r$, our algorithm can assign extra pursuers to evaders to ensure robustness to robot failures.

To summarize, our algorithm works as follows. In a decentralized manner, each robot pre-computes a $c(G) - 1$ state transition diagram. Then, after being informed by the network of the positions of all robots, each robot runs the assignment algorithm described above once. Thereafter, using the pre-computed state transition diagram and network localization updates, all robots continuously play the game until all evaders are captured.

IV. Design

In this section, we discuss the hardware and network testbed which form the basis for our pursuit-evasion experiments. We then describe, in some detail, our PEG software design and implementation. This sets the stage for our system evaluation, which is discussed in the next section.

A. Platform

The Robot Platform. We use a commoditized robotics platform and made minimal modifications to it using COTS. Our platform consists of an iRobot Create and a small embedded computer mounted on top of it (Figure 1(b)).

The embedded computer, Ebox 3854, runs Linux Fedora Core 6 as the operating system. For sensing and control, we developed a Create driver for Player [11], using which we are able to move the robot, turn on/off LEDs, read the bumpers, buttons and IR sensors. We set the nominal speed to 0.2 m/s. **The Network.** The robots use the network shown in Figure 1(a). This network is deployed above the false ceiling on one floor of a large office building and consists of two tiers: an upper tier containing 6 Stargates (embedded computers

running Linux), and a lower-tier containing 56 TelosB motes (tiny commercial sensor nodes).

The network provides the robots two capabilities. The upper-tier nodes form a wireless communication backbone that can be used for inter-robot communication and coordination.The presence of a static network infrastructure ensures that robots can quickly communicate with each other, and thereby coordinate more effectively. The lower-tier nodes provide the capability of a virtual position sensor, and can sense the position of pursuers and evaders. We describe the details of this capability below.
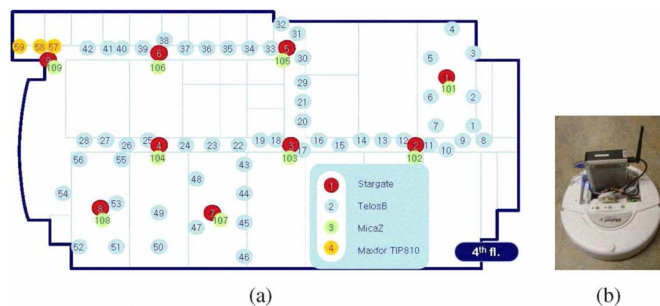

Fig. 1.   a)Layout of the network testbed b)robot platform

B. Network-Assisted Localization

Our robots do not have extra hardware such as sonar or lasers to determine robot poses. In our system, the network estimates robot position. We use a topological map as a representation of the environment. The goal of our network-assisted localization subsystem is to approximately place each robot at a node in this topological map.

This subsystem uses the signal strength of periodic "beacon" messages emitted roughly once every second (we add some randomization to the interval to reduce collisions) by the robots themselves.

The second-tier motes receive these beacons, and report all beacons whose signal strength is above a certain threshold. Tenet [12], a readily available open-source software package for programming wireless sensor networks, is the software that collects the beacon signal strength. A centralized robot location server then applies a voting scheme on a sliding window of reports to generate a location estimate.

Our architecture also allows for a decentralized location system. Instead of retrieving location information from the server, this information is flooded on the upper-tier network, of which the robots are a part. We have experimented with this version as well, and the performance of the system is comparable to the centralized system, as we show in Section V.

C. Navigation

We design a simple `wall-following` behavior for the Create to traverse the environment using the IR sensor.

The navigation component calculates the goal position, and invokes `wall-following` to move from one topological node to the adjacent one. Navigation is executed every time the robot changes its position (as well as when any evader

changes its position), so the robot continuously updates its trajectory.

The navigation component calculates the goal position given its team's position and that of its assigned evader. Using [7], we pre-compute a state transition diagram for a given team configuration. This state transition diagram is pre-loaded on all the robots, and each pursuer or evader makes a decentralized local state transition decision, given the current state. This decision tells the robot which topological node to move to next.

However, there exists an important subtlety in the navigation algorithm imposed by the minimality of our platform. Our robot has no inherent proprioception capability, it can only travel parallel to a wall, keeping the wall to its right (since the robot has only one IR wall sensor positioned on its right). As a result, the robot might actually move in a direction opposite to that intended by the navigation component. To rectify this, we add a simple *a posteriori* correction to the navigation component. If the `wall-follower` moves the robot to a node that it does not expect to arrive at, it invokes the `detach` behavior to reverse direction.

V. EVALUATION

In this section, we describe the results from several games played on the physical robot testbed described in Section IV.

We play the games on the floor plan shown in Figure 1(a), using the network whose nodes are shown in that figure.

The convergence time of a game depends on the initial configuration. We play our games using a worst-case initial configuration (there can be many). To calculate the worst-case configuration, we implemented an idealized game simulatorand exhaustively enumerated all configurations.

We can also use the simulator to compare the partition strategy with the optimal (Table I) for various topologies. The first column is the topology. The second column is the necessary number of pursuers to guarantee the termination of the game for that topology. The third column is the maximum number of steps to terminate the game (t) with 1 evader, which of course is the same in both strategies. The fourth and fifth columns give the maximum number of steps to terminate the game using optimal and partition strategy for two evaders. For these topologies, the partition strategy has the same completion time as optimal strategy.

The drawback of our partition strategy is we might use more pursuers than the minimum necessary. In a ring topology, 3 pursuers are sufficient to capture 2 evaders. In the partition strategy, we need 4 pursuers. We believe that this tradeoff is acceptable, since the partition strategy enables efficient capture.

We played a few (specifically, $2-1$, $4-2$, and $6-3$) games on our real world testbed, and we discuss the results below.

An Illustrative Game Instance. Figure 2(b) shows the trajectory of each pursuer during one instance of a $4-2$ game. The nodes and solid undirected edges connecting them represent the topological map. The solid directed lines (lines with arrows) show the pursuer path as determined by the

Topology	$c(G)$	$t(c(G)$-1 game)	$t(2c(G)$-2 game)	
			Optimal	Partition
Grid 2D 3x3	2	4	4	4
Cylinder Grid 3x3	2	3	3	3
Torus Grid 4x4	3	4	4	4

TABLE I

GAMES AND THEIR PROPERTIES

localization system. The dashed lines illustrate the evader path. The edge labels represent the time sequence of the robot. The pursuer and evader's initial positions are indicated by the corresponding icons.

In simulation, the capture time for the described game is 5 steps. Our results took 7 steps because our robots do not have a sense of direction and can only reverse direction by going in the wrong direction to determine that they have done so. This behavior is not capture in the simulator. Despite this (and other) non-idealities, our implementation works well in the real-world.

Results across Multiple Games. Finally, Figure 2(c) depicts the capture time across multiple games. The capture time is similar between the games, as one might expect (since each game is partitioned into a $2-1$ game). However, there are small differences in capture times. For a $4-2$ game, even with a pursuer failure in localization, the game terminated since only 2 robots are needed in our topology.

VI. RELATED WORK

To situate our work in the existing literature, we classify the type of PEGs using seven criteria: the *ratio* of the number of pursuers to the number of evaders; whether pursuers and evaders have full and/or global *visibility*, or whether they can only see within a threshold distance or until occluded by an obstacle (usually modelled by the edge of a polygon in 2D); what additional *information* robots have with respect to the opponents' strategy or planning algorithm; whether the *environment* is modelled as a graph (discrete) or a polygon (continuous half-space with lines in 2D as boundaries); how the evader is *captured*, whether by being surrounded, seen or sensed by the pursuer, or approached within a certain distance, or physically contacted; the relative *speed* between the pursuer and evader;and, if the *time to capture* is important.

Table II shows a classification of the related work in the literature along these dimensions. Our work is distinct from several pieces of prior work, as shown in Table II. The novelty of our proposed work is clear in several dimensions: while other work has explored theoretical bounds on eventual capture [13, 14], or pursuit-evasion under constrained geometries [15, 19] , or has examined sophisticated control strategies [17, 18], our proposed work attempts to minimize the time of captured of a multi-pursuer multi-evader under the pragmatic realization of physical multi-robot games.

In the survey presented by Alspach [20], a number of references on the necessary number of pursuers for a given graph class can be found. In [5], an algorithm to determine if K pursuers are sufficient to capture an evader is presented.
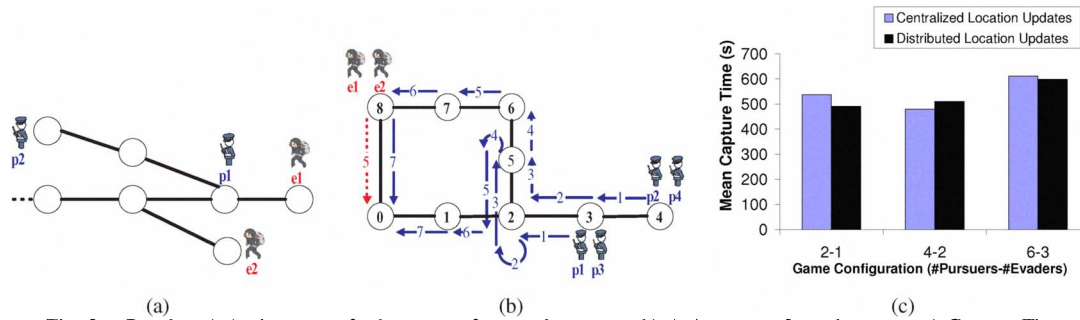
Fig. 2.   Results: a) An instance of a bag game for greedy strategy b) A 4 pursuer, 2 evader game. c) Capture Time

	[13]	[14]	[15]	[16]	[17]	[18]	Our work
Pursuer to Evader Ratio	≥ 1	1	≥ 1	≥ 1	≥ 1	1	$\gg 1$
Pursuer Visibility	full	full	local	local	local	full	full
Evader Visibility	full	full	full	full	local	none	full
Information	full	full	full	full	no	pursuer full	full
Environment	Graph	Polygon	Polygon	Polygon	Polygon	Polygon	Graph
Capture	touch	touch	see	touch	touch	touch	touch
Speed (faster entity)	same	same	same	evader	any	same	same
Time to Capture	no	no	no	no	no	no	yes
Robot Implementation	None	None	None	None	Partial	Partial	Full

TABLE II

RELATED WORK IN PURSUIT-EVASION

Aigner and Fromme [13] proved that in a planar graph G, 3 pursuers are sufficient for the pursuers to win the game.

VII. CONCLUSIONS

We presented an assignment algorithm that guarantees the game terminates, has bounded captured time, is robust, and is scalable in the number of robots, being suited for practical applications. We have reported on the design and experimental characterization of a nontraditional mobile robot-based pursuit evasion system. In our system robot sensing and communication is enhanced by an environment-embedded network. We have validated the feasibility of our algorithm by experimentally playing mobile robot-based pursuit evasion games on a physical testbed.

ACKNOWLEDGEMENTS

REFERENCES

[1] B. Kuipers and Y.-T. Byun, "A Robot Exploration and Mapping Strategy Based on a Semantic Hierarchy of Spatial Representations," Tech. Rep. AI90-120, 1, 1990.

[2] M. J. Matarić, "Integration of Representation Into Goal-Driven Behavior-Based Robots," IEEE Transactions on Robotics and Automation, no. 3, June 1992.

[3] M. A. M. Vieira, R. Govindan, and G. S. Sukhatme, "Scalable and Practical Pursuit-Evasion," Department of Computer Science, University of Southern California, Tech. Rep. 09-902, 1 2009. [Online]. Available: http://www.cs.usc.edu/Research/ReportsList.htm

[4] R. J. Nowakowski and P. Winkler, "Vertex-to-Vertex Pursuit in a Graph," Discrete Mathematics, vol. 43, no. 2-3, pp. 235–239, 1983.

[5] A. Berarducci and B. Intrigila, "On the Cop Number of a Graph," Adv. Appl. Math., vol. 14, no. 4, pp. 389–403, 1993.

[6] S. J. Russell and P. Norvig, Artificial Intelligence: A Modern Approach. Pearson Education, 2003.

[7] M. A. M. Vieira, R. Govindan, and G. S. Sukhatme, "Optimal Policy in Discrete Pursuit-Evasion Games," Department of Computer Science, University of Southern California, Tech. Rep. 08-900, 1, 2008.

[8] R. Jonker and A. Volgenant, "A Shortest Augmenting Path Algorithm for Dense and Sparse Linear Assignment Problems," Computing, vol. 38, no. 4, pp. 325–340, 1987.

[9] R. E. Burkard and E. Çela, "Linear Assignment Problems and Extensions," Handbook of Combinatorial Optimization, vol. 4, 1999.

[10] U. Pferschy, "Solution Methods and Computational Investigations for the Linear Bottleneck Assignment Problem," Computing, vol. 59, no. 3, pp. 237–258, 1997.

[11] B. P. Gerkey, R. T. Vaughan, K. Støy, A. Howard, G. S. Sukhatme, and M. J. Mataric, "Most Valuable Player: a Robot Device Server for Distributed Control," vol. 3, Maui, HI, USA, 2001, pp. 1226–1231.

[12] O. Gnawali, B. Greenstein, K.-Y. Jang, A. Joki, J. Paek, M. Vieira, D. Estrin, R. Govindan, and E. Kohler, "The TENET Architecture for Tiered Sensor Networks," in Proceedings of the ACM Conference on Embedded Networked Sensor Systems, November 2006.

[13] F. M. Aigner. M, "A Game of Cops and Robber," Tech. Rep., 1984.

[14] J. Sgall, "Solution of David Gale's Lion and Man Problem," Theor. Comput. Sci., vol. 259, no. 1-2, pp. 663–670, 2001.

[15] L. J. Guibas, J.-C. Latombe, S. M. LaValle, D. Lin, and R. Motwani, "Visibility-Based Pursuit-Evasion in a Polygonal Environment," in WADS '97: Proceedings of the 5th International Workshop on Algorithms and Data Structures.   London, UK: Springer-Verlag, 1997, pp. 17–30.

[16] V. Isler, S. Kannan, and S. Khanna, "Randomized Pursuit-Evasion in a Polygonal Environment," IEEE Transactions on Robotics, vol. 5, no. 21, pp. 864–875, 2005.

[17] R. Vidal, O. Shakernia, H. J. Kim, D. H. Shim, and S. Sastry, "Probabilistic Pursuit-Evasion Games: Theory, Implementation, and Experimental Evaluation," Robotics and Automation, IEEE Transactions on, vol. 18, no. 5, pp. 662–669, 2002.

[18] S. Oh, L. Schenato, P. Chen, and S. Sastry, "Tracking and Coordination of Multiple Agents using Sensor Networks: System Design, Algorithms and Experiments," Proceedings of the IEEE, vol. 95, no. 1, Jan 2007.

[19] R. Murrieta-Cid, T. Muppirala, A. Sarmiento, S. Bhattacharya, and S. Hutchinson, "Surveillance Strategies for a Pursuer with Finite Sensor Range," Int. J. Rob. Res., vol. 26, no. 3, pp. 233–253, 2007.

[20] B. Alspach, "Searching and Sweeping Graphs: a Brief Survey," Le Matematiche (Catania), vol. 59, pp. 5–37, 2004.