ON THE EFFICIENT CAPTURE OF DANGEROUS CRIMINALS

Vincenzo Gervasi
Giuseppe Prencipe
Dipartimento di Informatica, Università di Pisa
{*gervasi,prencipe*}*@di.unipi.it*

1. *Prologue*

President Hush[1] was extremely proud of his robotics troops. The search for the notorious criminal Assuma' Hex Loaded was proceeding fairly well, and the robotic troops on the ground were on the verge of surrounding and capturing the escapee. Compared to this, the thrill of his chief scientific advisor, who was babbling about how efficient the new capture algorithm was, seemed to President Hush rather incomprehensible. To him, what mattered was that the robots could reach and surround the enemies, making sure they could not escape, and keep them in check until the bomb squad could come and get rid of that nuisance. Of course, this tactic would cause the loss of all the robots as well, but in the end robots were expendable, did not vote, did not have a family awaiting them back home, and their loss was not necessarily a sad occurrence, in view of the needs of the arm lobby. Or at least, this was what his advisor Condoglianza kept saying him.

Defense Secretary Donald Duckfield had a completely different point of view. He had always felt that the extermination of the enemy had to be performed with maximal efficiency. It was not enough to be effective, the robotic troops had to be optimal. Indeed, he had allocated 7.5 quadrillion dollars to a promising project comparing the performances of different algorithmic approaches to the problem.

Of course, we could not afford to let such an important source of funding go. So, while waiting for a promised national grant of 7.5 euros, we settled to study the effectiveness of several algorithms to accomplish the desired task of reaching and surrounding an enemy unit fleeing in a hostile battlefield. This paper reports on the finding of our investigation.

2. *Formalizing the capture problem*

Consider an open area where a set of robotic troopers are parachuted at random spots to surveil a restricted area. Their task is to *capture* an hostile unit, or *enemy*, that could possibly enter the surveilled zone: the robots consider their task accomplished when they surround the enemy, so

[1]All persons and events portrayed in this paper are entirely fictional. Any coincidence with real persons or facts is purely coincidential.

that it has no means to escape. This problem can arise in a number of real-world situations beyond battlefield operations. For example, sensible areas where little or no traffic is expected, like airfields runways and aprons, or logistic compounds, could be effectively patrolled by robotic units.

In all these cases, the robots must be truly autonomous — the problem must be solved without relying on any kind of on-site infrastructure. Also, robots could conceivably be knocked off by opponents, and radio communications among them could be intercepted (thus revealing their presence) or disrupted (thus making them useless). Hence, a good solution to our problem must do away with explicit communication, relying instead only on the intrinsic capabilities of the robots; it must assume no external help, and should be able to adapt to a varying (i.e., decreasing) number of robots.

This problem has been extensively explored in a graph-oriented setting [3, 7]: the robots have to patrol an area that is described as a graph; they can move only from node to node, following the edges connecting them. In the graph there is also an enemy robot, and the patrolling units must surround him: in particular, they have to occupy all the neighborhood of the node where the enemy is. In contrast to this kind of study, in our approach the patrolled area is modelled as a two dimensional plane where our agents, as well as the enemy, can freely move.

A related problem to ours has been analyzed in [12, 13], where the robots and the enemy could move strictly inside a polygonal area (including its border): each surveilling robot could hold a flashlight that emits rays of light whose direction can be changed continuously. In their model, each robot can only see points lying on one of the rays. The goal of the robots is to detect the presence of the enemy. This is different from our problem, in that we assume the robots can always see the enemy, but we ask them to surround him rather than just detecting him.

Following the motivations that prompted previous studies ([6, 10, 11]), in this paper we adopt extremely simple units to study the problem: the robots are completely anonymous, identical (no identities are used during the computation), asynchronous, memoryless, and with no means of direct communication. We describe two algorithms, the same for all the robots, that allows them to surround the enemy, limiting his movement ability, and to keep him surrounded until some external event concludes the pursuit. Moreover, we present results of computer simulations that show the effectiveness of the proposed solution.

2.1. *Computational model*

We consider a system composed by n autonomous mobile robots. Each robot is capable of observing its surrounding, computing a destination based on what it observed, and moving towards the computed destination; hence it performs an (endless) cycle of observing, computing, and moving.

Each robot has its own *local view* of the world. This view includes a local Cartesian coordinate system having an origin (that without losing generality we can assume to be the position of the robot), a unit of length, and the *directions* of two coordinate axes, together with their *orientations*, identified as the positive and negative sides of the axes. Notice that there is no agreement among the robots on the chirality of the respective coordinate systems (i.e., the robots do not share the same concept of where North, East, South, and West are).

The robots are modeled as units with computational capabilities, which are able to freely move in the plane. They are equipped with sensors that let each robot observe the positions of the others with respect to their local coordinate system. Each robot is viewed as a point, and can see all the other fellow robots in the patrolled area, as well as the enemy.

The robots act totally *independently* and *asynchronously* from each other, and do not rely on any centralized directives, nor on any common notion of time. Furthermore, they are *oblivious*, meaning that they do not remember any previous observation nor computations performed in the previous steps.

The robots are *anonymous*, meaning that they are a priori indistinguishable by their appearances, and they do not have any kind of identifiers that can be used during the computation. They can only distinguish the enemy from a fellow robot. Moreover, there are no explicit direct means of communication; hence the only way they have to acquire information from the fellow robots is by observing their positions.

They execute the same algorithm, which takes as input the observed positions, and returns a destination point towards which the executing robot moves. A robot, asynchronously and independently from the other robots, (i) *observes* the environment (*Look*), by taking a snapshot of the positions of all other robots and of the enemy with respect to its local coordinate system[2]; (ii) It *computes* a destination point p according to its oblivious algorithm (*Compute*); the local computation is based only on the current (i.e., at the time of the previous *Look*) locations observed by the robot. (iii) Finally, the robot *moves* an unpredictable amount of space towards p (*Move*), which is however assumed to be neither infinite, nor infinitesimally small (see Assumption A1 below), and goes back to the *Look* state.

In the model, there are three limiting assumptions. The first refers to space; namely, the distance traveled by a robot during a cycle of activity.

Assumption A1 (Distance) *The distance traveled by a robot r in a* Move *is not infinite. Furthermore, it is not infinitesimally small: there exists a constant $\delta_r > 0$, such that if the destination point is closer than δ_r, r will reach it; otherwise, r will move towards it by at least δ_r.*

The reason for introducing δ_r is to ensure progress in the movement of the robots; in other words, if r aims to reach a destination point p, A1 ensures that r will reach p in a finite number of cycles. Without such an assumption, it would be impossible to prove the termination of any algorithm in a finite number of cycles. As no other assumptions on space are made, the distance traveled by a robot in a cycle is unpredictable.

The second assumption in the model refers to the duration of a cycle of activity.

Assumption A2 (Cycle of Activity) *The amount of time required by a robot r to complete a cycle of activity is not infinite. Furthermore, it is not infinitesimally small: there exists a constant $\varepsilon_r > 0$ such that the cycle will require at least ε_r time.*

[2]Since each robot is viewed as a point, its position in the plane is given by its coordinates.

The purpose of A2 is to make sure that the capturing task does not trivially fail because some robot takes an infinite time to complete one of its cycles. As no other assumption on time exists, the resulting system is truly *asynchronous* and the duration of each activity (or inactivity) is unpredictable. As a result, robots can be seen while moving, and computations can be made based on obsolete observations.

Finally, since we need to model robots that "continuously" move, we assume that

Assumption A3 (Continuous Movement) *The time spent in looking and computing is negligible compared to the time spent in moving.*

We stress that no one of the followers knows in advance the path that the enemy will follow, nor can it derive it at run-time (e.g., by observing the position of the enemy at different times or his heading in order to estimate the current direction).

2.2. *Formalization*

We consider a system of autonomous mobile robots that have to patrol a given area, modelled as an infinite plane. A distinguished independent unit *E*, the *enemy*, is also on the plane. The goal of the robots is to surround the enemy, while keeping at a certain distance from him, in order to reduce his leeway. In particular, the robots must place themselves as to minimize the maximum distance that the enemy can place between himself and the nearest robot along any escape route.

Let l_1 and l_2 be respectively the minimum and maximum distance from the enemy that we want the robots to maintain (given as constants of the problem). It is easy to see that the problem as stated is solved when the n robots place themselves uniformly spaced on a ring at a distance $l_1 \leq l \leq l_2$ from the enemy, thus forming a regular polygon of characteristic angle $\phi = 2\pi/n$ and radius l_1 (see [8] for a fully formal definition of the problem). We call *capture area* the ring $\mathcal{C}_2 \setminus \mathcal{C}_1$, with $\mathcal{C}_1$ and $\mathcal{C}_2$ the two circles centered in *E* and having radius l_1 and l_2, respectively.

Since the enemy keeps moving, it is impossible for the robots to maintain a perfect solution. In the following we will consider sub-optimal solution acceptable, as long as they are indefinitely maintained once first reached at time t_0. In this context, a sub-optimal solution is defined as having each robot at a distance between $l_1 - \varepsilon_1$ and $l_2 + \varepsilon_2$.

The constants ε_1 and ε_2 are also tied to the temporal features of the asynchronous behavior of the robots. In fact, the longer the time between two consecutive *Look*s of a robot, the more outdated the snapshot taken of the other robots' positions becomes. Hence, computationally slow robots will only be able to guarantee a sub-optimal solution for relatively large values of $\varepsilon_{1,2}$, while faster robots will be able to better approximate the optimal solution.

Finally, it is worthwhile to observe that the robots have no hope of reliably capturing an enemy faster than themselves. Therefore, a necessary condition for the solvability of the problem is that the enemy is slower than the slowest of the robots, i.e.

$$v_I < \min_i v_{r_i},$$

where v_k denotes the linear velocity of k.

3. *Algorithms*

We present here two algorithms solving our problem using two different approaches, and compare their performances in the next section. The first algorithm, that is a variation of the one introduced in [8], tackles the problem from a strictly algorithmic perspective, while the second, that is introduced here, relies on an heuristic.

An algorithm for our robots will have in input the positions of all the other robots at the time of the last *Look*, and the position of the enemy, expressed as set of points in the local coordinate system of the robot.

The algorithm must return as output the point p towards which the robots should move, also expressed in the local coordinate system. $(E.x, E.y)$ will denote the coordinates of E, and Me the current position of the robot executing the algorithm, that is $(0, 0)$ in its local coordinate system.

Note that a requirement of any capture algorithm is that the robots must have common knowledge [9] of the unit of measure. This is needed to allow them to have a common understanding of constants l_1 and l_2, and to agree on the distance they have to be to surround the enemy.

3.1. *The LAT Algorithm*

Algorithm 1 (LAT) An algorithmic solution to the capture problem.

```
1:  $Chief$ := Closest Robot to $E$;
2:  If I Am $Chief$ Then
3:      $l :=$ dist$(Me, E)$;
4:      $target := (E.x \cdot \frac{l-l_1}{l}, E.y \cdot \frac{l-l_1}{l})$;
5:  Else
6:      $\phi = 2\pi/n$;
7:      sortByAngle($Robots, E, Chief$);
8:      $k$ :=myRank();
9:      $\theta$ :=angle($MyX, [E, Chief)$);
10:     $\alpha := k \cdot \phi + \theta$;
11:     $l := \max(l_1, \text{dist}(E, Chief)) \cdot (1 + \varepsilon)$;
12:     $target := (E.x + l \cdot \cos(\alpha),$
13:                 $E.y + l \cdot \sin(\alpha))$;
14:     $\mathcal{C}$ := Circle Centered in $E$ With Radius $l$;
15:     If $[Me, target] \cap \mathcal{C} \neq \emptyset$ Then
16:         $target$ :=NonIntersTarget($E, target, l$);
17: Return $target$;
```

The idea of the algorithm is as follows. First, the closest robot to the enemy is determined (call it *chief*). The chief simply moves towards or away from the enemy, trying to maintain a

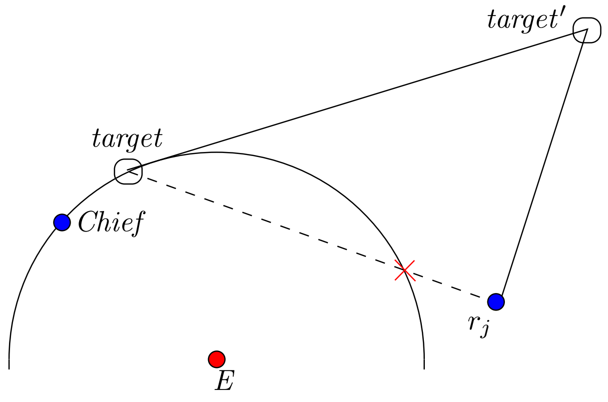


Figure 1: Sideway stepping in `NonIntersTarget()` routine.

distance l_1 from him (Lines 2–4). All the other robots aim to reach the vertices of the regular n-gon inscribed in a circle centered in the observed enemy's position and having as radius l such that $l_1 < l \leq l_2$ (Lines 6–16). Once they reach such a position, the robots' task is achieved.

In order to reach an agreement on which vertex is assigned to each robot, the robots are sorted by routine `sortByAngle()`: in particular, the chief is considered to be the first robot in the order; the other robots are sorted, in increasing order, according to the angle each of them forms with the enemy and the chief (Line 7). At this point, the targets (i.e. the positions they have to reach in order to complete the task) of the robots are computed: these are the vertices of the regular polygon having characteristic angle $\phi = 2\pi/n$, with the first vertex being on the chief's position, and inscribed in the circle $\mathcal{C}$ centered in E and having radius $l = \max(l_1, \text{dist}(E, \textit{Chief})) \cdot (1+\varepsilon)$ (Line 11). The target of the i-th robot in the ordering is the i-th vertex of the polygon. Routine `angle()` in Line 9 returns the angle between the half-line $[E, \textit{Chief})$ and the x axis in the local coordinate system of the executing robot: this angle is used to rotate the polygon to be formed so that the first vertex coincides with the *Chief* (Line 10). The reason for the targets being computed with respect to $\mathcal{C}$ and not with respect to a smaller circle of radius exactly l_1, is to reduce cases where another robot becomes chief, displacing the previous chief: in fact, such displacements would introduce some instability in the algorithm, slowing down convergence.

Also, it is possible that a robot r, to reach its target, crosses $\mathcal{C}$. This too would introduce instability in the algorithm, since in so doing r could come closer to E than the current chief, thus becoming chief itself. To avoid this effect, Line 16 of the algorithm invokes routine `NonIntersTarget()`, that forces r to take a route outside $\mathcal{C}$, so that no crossing is possible: r moves sideways until a straight path from its current position to its assigned target does not cross $\mathcal{C}$ (see the example depicted in Figure 1). In this routine, the constant ρ represents the length of the sideway step. The robot will keep stepping sideway until necessary to reach its real target without crossing $\mathcal{C}$.

```
**Routine** NonIntersTarget($E, target, r'$)
  $\beta := \arctan(target.y/target.x)$;
  $\gamma :=$angle($Me, E, target$);
  **If** $\gamma > \pi$ **Then**
    $\beta := beta + \pi/2$;
  **Else**
    $\beta := beta - \pi/2$;
  **Return** $(\rho \cdot \cos(\beta), \rho \cdot \sin(\beta))$.
```

3.2. *The HEUR-S Algorithm*

Algorithm 2 (HEUR-S) An heuristic solution to the capture problem

```
1: $l :=$ dist$(Me, E)$;
2: $target := (E.x \cdot \frac{l-l_1}{l}, E.y \cdot \frac{l-l_1}{l})$;
3: $dx := E.x \cdot \frac{l-l_1}{l}$;
4: $dy := E.y \cdot \frac{l-l_1}{l}$;
5: $cord = 2 \cdot l_1 \cdot$ sin $\left(\frac{\pi}{n}\right)$;
6: **For All** $i = 1..n$ such that I am not $r_i$ **Do**
7:     $l' :=$ dist$(Me, r_i)$;
8:     **If** $l' < cord$ **Then**
9:         $dx := dx + r_i.x \cdot \frac{l'-cord}{l'}$;
10:        $dy := dy + r_i.y \cdot \frac{l'-cord}{l'}$;
11: **Return** $(dx, dy)$;
```

The intuition behind Algorithm 2 is as follows. All robots are subject to a force, attracting them towards the enemy if they are farther than l_1 from him, or repulsing them if they are nearer. Moreover, when two robots come closer to each other than a certain distance $cord$, they repel each other. The distance $cord$ is computed as the side of an n-gon of radius l_1 (Line 5).

While the algorithm by itself does not coordinate the behavior of each robots with that of its fellows, like Algorithm 1 does when establishing a shared assignment of robots to vertices, it has as a lowest-energy equilibrium a configuration where the robots do evenly surround the enemy. In this sense, the behavior of Algorithm 2 is truly *emergent*, in that no explicit and direct solution of the problem is provided in the code (see [1, 2]).

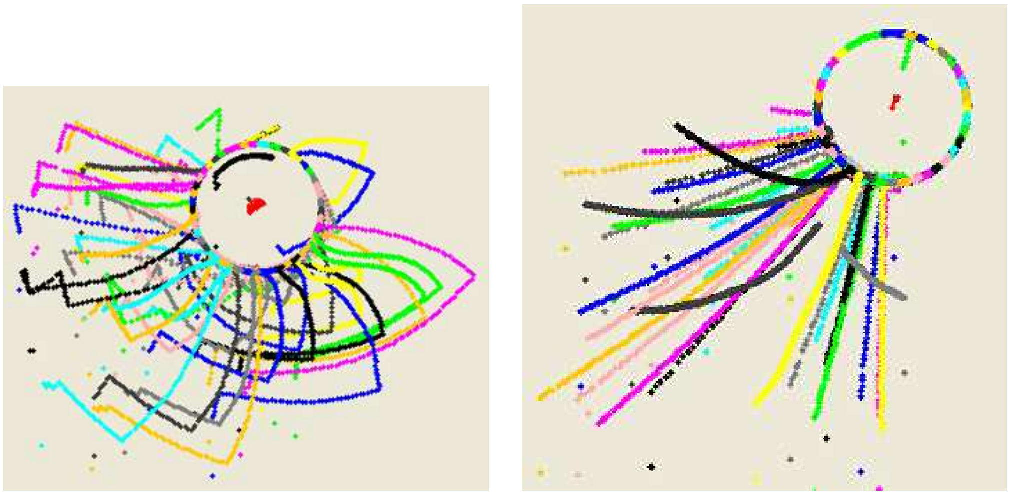

(a) (b)

Figure 2: Traces of the behavior of the robots according to (a) the LAT algorithm, and (b) the HEUR-S algorithm. The camera is fixed on the enemy, that thus appears static.

4. *Evaluation of the Algorithms*

Experimental setting. To assess the effectiveness of the two algorithms, we ran a number of tests using numerical simulations. Each run included a random[3] number of robots between 2 and 50; the enemy and the robots were initially placed at random in a 256×256 units square. The robots had their axes orientation and direction assigned randomly, and linear speed v_f between 0.5 and 5 space units per time units.

The enemy's course was determined as follows: at all times, the enemy would move forward according to its linear velocity (determined randomly). At each move, with a probability of 1/10, the enemy could start turning to its left or right, with random angular velocity less than its maximum angular velocity. If already turning, with probability 1/100 the enemy could stop and continue its course as a straight line (these parameters ensured curved, irregular trajectories).

As an example, Figures 2(a) and 2(b) show the traces of two run of the LAT and HEUR-S algorithm, respectively.

Measures. To measure the convergence features of the algorithms, we measured two parameters. The first one, ν_r, measures how many robots have reached the capture area, as a ratio of

[3]In all cases, random values were obtained from a linear distribution.

the total number of robots:

$$\nu_r = \frac{|\{r_i | l_1 \leq \text{dist}(r_i, E) \leq l_2\}|}{n}.$$

The second one, ϕ_r, measures the ratio between the largest angle between two angularly adjacent robots in the capture area, and the optimal value of such an angle ($2\pi/n$), i.e.

$$\phi_r = \frac{n \cdot \max_{i,j}\{r_i \widehat{E} r_j\}}{2\pi},$$

with $i \neq j$, such that there is no r_k in the region of the plane delimited by the half-lines $[E, r_i)$ and $[E, r_j)$ intersected with the capture area. Values of ϕ_r close to 1 indicate that the robots are close to the optimal capture configuration.

Results. In all cases the robots were able to surround the enemy, correctly solving the problem (although with sub-optimal solutions, as described earlier). The results obtained by averaging the measures above over 1000 random runs of our algorithms, with each run comprising 4000 *Look–Compute–Move* cycles are shown in Figures 3 and 4.

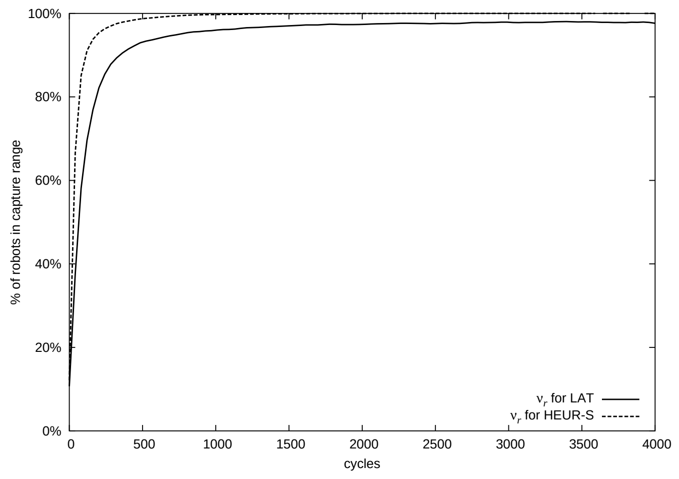


Figure 3: Average number of robots in the capture area (ν_r) in the simulations.

As can be observed in the figures, both algorithms exhibit reasonably fast and stable convergence to a good solution. In particular, Algorithm 1 (LAT) sports a slower convergence of the robots into the capture area than Algorithm 2 (HEUR-S), as shown in Figure 3. This is not surprising, as LAT directs the robots directly towards their final positions, and may have to re-route them laterally when `NonIntersTarget()` is called. In contrast, HEUR-S simply moves the

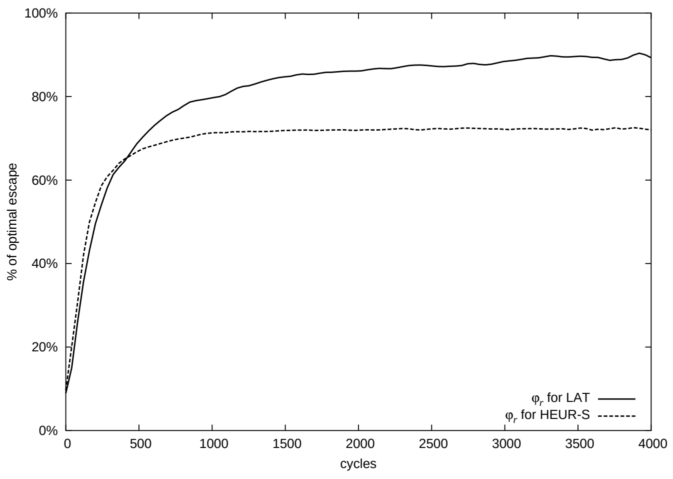


Figure 4: Average relative largest angle between two angularly adjacent robots in the capture area (ϕ_r) in the simulations.

robots towards a position distant l_1 from the enemy, leaving their uniform distribution around the enemy for a later stage (after the robots have entered the capture area). It is worthwhile to notice that, while ν_r for LAT converges more slowly than that of HEUR-S, both algorithms have essentially the same asymptotic performance, with all the robots reaching the capture area.

On the other hand, LAT behaves much better than HEUR-S in terms of ϕ_r (see Figure 4). The more precise strategy employed by LAT allows the robots to better approximate the optimal equidistribution around the enemy. Indeed, LAT reaches 90% of the optimal distribution, while HEUR-S stops at 72%.

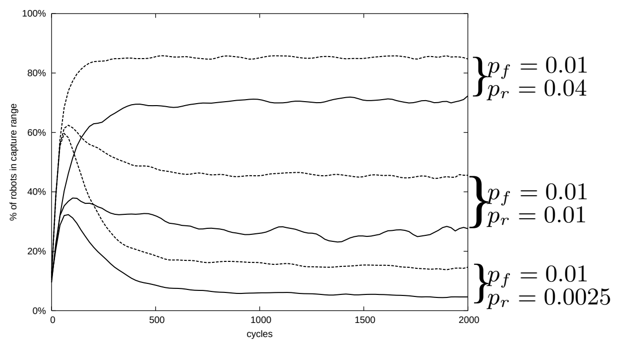
5. *Fault Tolerance*

An interesting aspect in the study of autonomous robots is the characterization of their behavior in the presence of faults. In this section, we analyze the behavior of our algorithms when a particular type of faults — transient hang-ups of the robots, during which they stop moving — can occur.

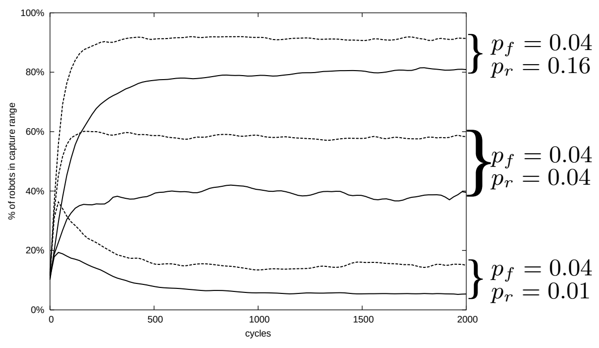
This fault model is based on two parameters, namely, the probability of occurrence of a fault p_f, and the probability of resuming the normal behavior p_r. Initially, all robots are in order. At each cycle, with probability p_f, a robot can enter its faulty state; in this case, it stops moving, but continues executing its *Look-Compute-Move* cycle. A faulty robot can recover and switch back to normal behavior with probability p_r.

It can be seen easily that the obliviousness of the robots in our model makes the system self-stabilizing in the sense of Dijkstra [4] — that is, if after a certain number of cycles K faults no

Plots of ν_r for various values of p_f and p_r.

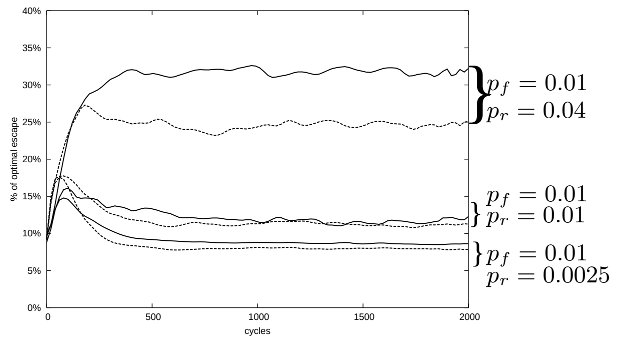


(a)

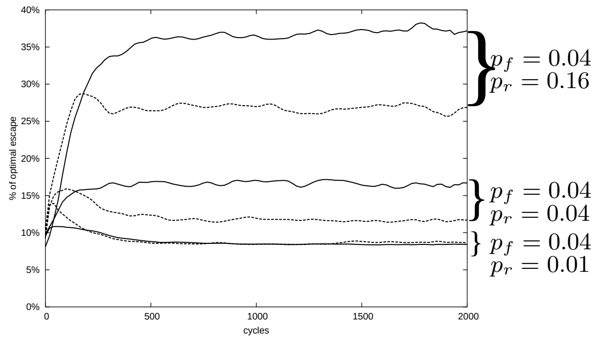


(b)

Plots of ϕ_r for various values of p_f and p_r.



(c)



(d)

Figure 5: Simulation results in the presence of faults. In all plots, the continuous line represents the measures for LAT, while the dashed line represents the measures for HEUR-S.

longer happen, the robots will solve their task correctly. Indeed, in such a setting we can imagine the configuration of the robots at cycle K as the starting one, and since our algorithms solve the problem starting from any arbitrary configuration, the self-stabilization property trivially holds.

Hence, we focus here on the performances of the algorithms when faults occur indefinitely. Figures 5(a) and 5(b) show the ν_r measured for the two algorithms with various values for p_f and p_r. As can be observed, the HEUR-S algorithm consistently beats the LAT algorithm under this measure. On the contrary, LAT performs better, even in the presence of faults, according to the ϕ_r measure, as can be observed in Figures 5(c) and 5(d).

It should be noted that, on average, the robots will be in order for $p_r/(p_f + p_r)$ of the time, and faulty in the remaining $p_f/(p_f + p_r)$ of the time. Thus, we can expect the optimal ν_r and ϕ_r to be reduced accordingly: for example, if $p_f = p_r$, we can expect half the robots to be faulty at any given time, and thus the measured ν_r and ϕ_r to be at most 0.5 (and indeed, this behavior can be observed in Figure 5(c) for $p_f = p_r = 0.01$, after the initial transient due to the fact that no robot is faulty at the start of the simulation).

This model, however, is not totally accurate, since the robots have a certain “leeway” due to several factors: first, the capture area can be large (i.e., $l_2 >> l_1$); second, the robots are faster than the enemy, thus they can make up for any delay due to a fault with their higher speed. In particular, for higher values of p_r the time spent in a faulty state can be short enough that the effects of the fault are hidden by the “leeway” effect described above. For example, this is what happens in Figure 5(b) for $p_f = p_r = 0.04$, where the HEUR-S algorithm attains $\nu_r = 60\%$ even if our reference “optimum” value would be 50%.

The data from our simulations seem to indicate that, in general, the HEUR-S algorithm is more robust than the LAT algorithm in the presence of faults, as far as ν_r is concerned. In other words, HEUR-S comes closer to the best possible performance with the given portion of faulty robots.

Both algorithms behave much worse w.r.t. ϕ_r. In fact, in no case they come close to the reference optimum value. For example, as can be observed in Figure 5(c) for $p_f = p_r = 0.01$, the value measured for ϕ_r is around 15%, compared to a reference optimum of 50%. This is not surprising, since we do not have for ϕ_r any leeway comparable to that provided for ν_r by the range (l_1, l_2). In ϕ_r, any deviation from the “right” position causes an immediate drop of ϕ_r.

6. *Conclusions*

In this paper we studied the capture problem: a number of robotic robots that patrol a restricted area have to capture an enemy that sneaked inside the area. The robots are non-communicating, asynchronous, anonymous and memoryless vehicles that can freely moves on a plane; the enemy is an external agent whose behavior is not known to the robots in advance.

We have provided two algorithms to solve the capture problem, that only assumes that the robots share a common unit of distance, but need not to have a common sense of direction (i.e., a common coordinate system).

Indeed, the algorithms we proposed exhibits remarkable robustness, and numeric simulations indicate that the enemy is efficiently captured in a relatively short time and kept surrounded after that, as desired. The solution we proposed is self-stabilizing [4, 5]. In particular, any external intervention (e.g., if one or more of the robots are stopped, slowed down, knocked out, or simply faulty) does not prevent the completion of the task.

Several variants of the algorithms we have presented are possible. In particular, both algorithms can be made to react dynamically to the detection of faults in their fellows, either by direct observation, or by considering as potentially faulty all the robots that are outside the capture area. These changes could improve the behavior of the robots when large number of units at a time is faulty (e.g., $p_f = 0.04$ and $p_r = 0.01$ in Figures 5(b) and 5(d)).

Another aspect worth studying is which kind of algorithms can be used when a bounded amount of memory is available to the robots, or when their observational capability is reduced (e.g., obstructed by other robots, or limited by distance). Moreover, these aspects would combine with the presence of faults (e.g., can we assume that robots outside of our field of vision are faulty?), giving rise to several complex settings. We intend to investigate these issues as part of our future work.

References

[1] R. C. Arkin. Motor Schema-Based Mobile Robot Navigation. *International Journal of Robotics Research*, 8(4):92–112, 1989.

[2] T. Balch and R. C. Arkin. Behavior-based Formation Control for Multi-robot Teams. *IEEE Transaction on Robotics and Automation*, 14(6), 1998.

[3] L. Barrière, P. Flocchini, P. Fraigniaud, and N. Santoro. Capture of an Intruder by Mobile Agents. In 14^{th} *Symposium on Parallel Algorithms and Architectures 2002 (SPAA 2002)*, 2002.

[4] E. W. Dijkstra. Self-stabilizing Systems in Spite of Distributed Control. *Communication of the ACM*, 17(11):643–644, November 1974.

[5] S. Dolev. *Self-Stabilization*. The MIT Press, 2000.

[6] P. Flocchini, G. Prencipe, N. Santoro, and P. Widmayer. Distributed Coordination of a Set of Autonomous Mobile Robots. In *IEEE Intelligent Vehicle Symposium (IV 2000)*, pages 480–485, 2000.

[7] N. Foukia, J. G. Hulaas, and J. Harms. Intrusion Detection with Mobile Agents. In 11^{th} *Annual Conference of the Internet Society (INET 2001)*, 2001.

[8] V. Gervasi and G. Prencipe. Robotic Cops: The Intruder Problem. In *2003 IEEE Conference on Systems, Man and Cybernetics (SMC 2003)*, pages 2284–2289, October 2003. Washington D.C., USA.

[9] J. Halpern and Y. Moses. Knowledge and Common Knowledge in a Distributed Environment. In *Proceedings of the* 3^{rd} *ACM Symposium on Principles of Distributed Computing*, pages 50–61, 1984.

[10] Y. Oasa, I. Suzuki, and M. Yamashita. A Robust Distributed Convergence Algorithm for Autonomous Mobile Robots. In *IEEE International Conference on Systems, Man and Cybernetics*, pages 287–292, October 1997.

[11] G. Prencipe. CORDA: Distributed Coordination of a Set of Autonomous Mobile Robots. In *Proceedings Fourth European Research Seminar on Advances in Distributed Systems (ERSADS 2001)*, pages 185–190, May 2001.

[12] I. Suzuki and M. Yamashita. Searching for a Mobile Intruder in a Polygonal Region. *Siam Journal on Computing*, 21(5):868–888, 1992.

[13] M. Yamashita, H. Umemoto, I. Suzuki, and T. Kameda. Searching for Mobile Intruders in a Polygonal Region by a Group of Mobile Searchers. *Algorithmica*, 31:208–236, 2001.