Auton Robot (2017) 41:1297–1320
DOI 10.1007/s10514-016-9577-x

Avoiding target congestion on the navigation of robotic swarms

Leandro Soriano Marcolino[1] · Yuri Tavares dos Passos[2] · Álvaro Antônio Fonseca de Souza[3] · Andersoney dos Santos Rodrigues[2] · Luiz Chaimowicz[3]

Received: 24 August 2015 / Accepted: 19 May 2016 / Published online: 10 June 2016

Abstract Robotic swarms are decentralized systems formed by a large number of robots. A common problem encountered in a swarm is congestion, as a great number of robots often must move towards the same region. This happens when robots have a common target, for example during foraging or waypoint navigation. We propose three algorithms to alleviate congestion: in the first, some robots stop moving towards the target for a random number of iterations; in the second, we divide the scenario in two regions: one for the robots that are moving towards the target, and another for the robots that are leaving the target; in the third, we combine the two previous algorithms. We evaluate our algorithms in simulation, where we show that all of them effectively improve navigation. Moreover, we perform an experimental analysis in the real world with ten robots, and show that all our approaches improve navigation with statistical significance.

✉ Leandro Soriano Marcolino
sorianom@usc.edu

Yuri Tavares dos Passos
yuri.passos@ufrb.edu.br

Álvaro Antônio Fonseca de Souza
alvaro.souza@dcc.ufmg.br

Andersoney dos Santos Rodrigues
andersoneyrodrigues@gmail.com

Luiz Chaimowicz
chaimo@dcc.ufmg.br

[1] Teamcore Laboratory, Computer Science Department, University of Southern California, Los Angeles, CA, USA

[2] Centro de Ciências Exatas e Tecnológicas, Universidade Federal do Recôncavo da Bahia, Cruz das Almas, BA, Brazil

[3] VeRLab - Vision and Robotics Laboratory, Computer Science Department, Universidade Federal de Minas Gerais, Belo Horizonte, MG, Brazil

Keywords Robotic swarms · Traffic control · Navigation · Distributed coordination

1 Introduction

Robotic swarms are systems formed by a large number of relatively simple robots, placed in the same region and interacting to fulfill a common goal. The inspiration of robotic swarms arises from insect colonies behavior, such as ants and bees. In these colonies, individuals interact using only local communication and there is no central unit controlling each and every individual. Even so, they are able to perform complex global behavior and solve hard problems in the real world.

In an environment where there are multiple robots, the tendency for a robot to interfere with other robots execution is greater (Lerman and Galstyan 2002). A common problem in the navigation of swarms is congestion, which happens when a large number of robots must move towards the same region simultaneously. For example, it is common that robots have a common target, as in waypoint navigation (Marcolino and Chaimowicz 2008). Even if they have distinct targets, these might be located close to each other, in the same region.

Besides waypoint navigation, we can also see this kind of congestion when a swarm of robots is solving a foraging problem (Sahin et al. 2008; Sahin 2004) (i.e., when robots must move to locations in the environment to collect items and transport them to a specific location). The foraging problem has many practical applications in the real world, such as transportation of toxic material or debris from landslides.

The congestion problem could be solved by using a central processing unit to compute the best trajectory for each robot. However, there is a drawback: the system becomes dependent

on this central unit and the solution is not scalable to a large number of robots.

We can see many works in the robotics and in the multi-agent systems literature dealing with traffic control in a distributed fashion, but they focus mainly in scenarios where agents navigate in delimited lanes and meet in the intersections of these lanes (Carlino et al. 2013; Ikemoto et al. 2004; Ferrati and Pallottino 2013; Hoshino 2011). Here, however, we are dealing with a problem where robots *may arrive from anywhere*, and *must go towards a specific target*, not simply pass through an intersection.

There are also many collision avoidance algorithms (Alonso-Mora et al. 2015; van den Berg et al. 2011; Franchi et al. 2015; Krontiris and Bekris 2011), but avoiding collision does not necessarily lead to a better performance in the common target problem, as the system can still become cluttered and inefficient even with a good collision avoidance algorithm.

In this work we present three fully distributed algorithms for congestion control of a swarm of robots that move towards a common target. Our first algorithm, originally proposed in Marcolino and Chaimowicz (2009), uses a probabilistic finite state machine to coordinate robot navigation. Robots near the target stop moving towards their objective for a random number of iterations, in order to help other robots reach the target. However, although this algorithm alleviates the congestion when robots are reaching the target, robots still have problems to exit the target area. Therefore, we propose a novel algorithm that divides the target area in two regions: one for the robots to move towards the target, and another for the robots to exit the target area. If a robot is moving towards the target and finds itself in the wrong area, it will move towards its appropriate region. Finally, in our third algorithm we combine the ideas from the two previous ones, effectively alleviating the congestion both for robots entering and for robots exiting the target area.

We evaluate our algorithms in simulation, where we show that we can improve the time of the executions in 76 %, clearly outperforming the baseline approach of using only collision avoidance. We also show that ORCA, a state of the art collision avoidance mechanism (van den Berg et al. 2011), is unable to handle the common target problem, as it reaches an equilibrium where the robots circulate around the target and do not move towards it. Additionally, we analyze our algorithms with real robots, and show that all our approaches improve the navigation with statistical significance for a group of ten real *e-puck* robots.

This paper is organized as follows: in the next section we present relevant related work. In Sect. 3 we present our first proposed algorithm, while in Sect. 4 we explain our second algorithm, and how both algorithms can be combined into a single one. We show and discuss our results in Sect. 5, where we study simulations and executions with real robots. Finally, Sect. 6 presents our conclusions and possible directions for future work.

2 Related work

We can find related works about traffic control not only in the robotics literature, but also in multi-agent systems, and even in the computer graphics literature. Besides, biological studies also show traffic control in animals and bacteria swarms. In this section we discuss these in detail. We will begin by discussing the works in robotics.

The problem of traffic control can be considered as a *resource conflict* problem. In Cao et al. (1997), it is denoted a "resource conflict" when a single and indivisible resource is requested by various robots. In the context of this work, the resource in question is the common target, represented by a small region of space. A theoretical study of spatial conflicts is presented in Savchenko and Frazzoli (2005), where a conflict is defined as an event generated when two robots get closer than a velocity-dependent safety distance. In their work, they prove that the time needed to transfer each robot from its origin to its destination, chosen arbitrarily, takes $\Theta(\bar{L}\sqrt{n})$ time to complete, where n is the number of robots and $\bar{L}$ is the average distance between origins and destinations.

The problem of organizing the traffic of a group of robots has been studied since the 80s. In Grossman (1988), an algorithm for traffic control of automated guided vehicles is presented, but his control is made in a centralized manner. The works of Kato et al. (1992) and Caloud et al. (1990) treat this problem in a decentralized manner using traffic rules that each robot obeys. However, they do not consider the specific problem where every robot must go towards a common target. In general, these works assume that the robots navigate in delimited lanes (like streets or roads). These lanes meet in intersections, where congestion may happen. The traffic control, normally, is executed only at these intersections.

Although traffic control has been studied for a long time, the problem of controlling the traffic of a swarm of robots that navigate towards a common goal has not been well studied. In the works of Sahin (2004), Sahin et al. (2008), Barca and Sekercioglu (2013), Brambilla et al. (2013), and Bayindir (2015), which present thorough reviews about swarm robotics, this problem is not specifically discussed.

Many works in robotics, however, study how to find efficient paths for a group of robots. For example, Olmi et al. (2009) develop an algorithm to coordinate predefined paths for a group of robots in industrial environments, and a central processing unit changes the path of each robot in case there is a possible collision. In Guo and Parker (2002), a distributed algorithm is designed for motion planning for multiple robots, but their simulation experiments involve at

most three robots. The work of Peasgood et al. (2008) deals with the trajectory collision problem for several robots, but in a context where the targets of each robot is different. Their approach also needs a *roadmap* of the environment. In Hoshino (2011), the congestion problem is dealt in a simplistic scenario: the robots navigate in lanes, and they are only allowed to move in one direction, without passing other robots. Ferrati and Pallottino (2013) propose an algorithm for distributed traffic management of a group of mobile collaborative vehicles moving within a shared environment, however they did not treat the common target problem. Ikemoto et al. (2004) show a completely distributed algorithm that, based on a spatial temporal pattern, coordinates the movement of robots into intersections or junctions. Overall, the works in robotics focus on the situations where robots navigate in delimited lanes, or present only results in simulation environments, or do not show results for a large group of robots.

Instead of dealing explicitly with traffic control, there are works that focus on finding more efficient approaches to collision avoidance than using local repulsion forces. In Krishna and Hexmoor (2004), an algorithm is proposed in which robots coordinate their velocities in order to avoid collisions. The coordination may entail not only the robots directly involved in the probable collision, but the robots in the neighborhood as well, which might have to change their velocities to help the robots involved. The work presented in Krontiris and Bekris (2011) deals with the collision resolution problem in a decentralized fashion, using an extension of the obstacle prevention policy named *Generalized Roundabout Policy* (Pallottino et al. 2007). Franchi et al. (2015) study the problem of encircling a moving target while guaranteeing collision avoidance between the robots. Other examples of works dealing with collision avoidance are Yasuaki and Yoshiki (2001), Cai et al. (2007), van den Berg et al. (2008), van den Berg et al. (2011) and Alonso-Mora et al. (2015). However, as mentioned, collision avoidance algorithms may not be sufficient for preventing congestion situations when a large number of robots converge to the same region. Hence, even with a good collision avoidance behavior, the system may still become cluttered and inefficient.

Traffic control is also an important issue in biology, when studying animal and bacteria swarms. For example, Bazazi et al. (2012) show that tadpoles form vortexes when foraging for food. Shapiro (1988) comments that the *Myxococcus xanthus* bacteria constructs spherical colonies containing millions of individuals, surrounding a common target in order to feed. Couzin and Franks (2002) study the behavior of army ants, showing that they form massive three-lane structures while transporting resources, minimizing traffic congestion.

Concerning robotic swarms, Santos et al. (2014) study the case where subgroups of a swarm meet during navigation, and each subgroup must remain segregated from the others while navigating. The navigation of different groups is also studied by Santos and Chaimowicz (2011), where a hierarchical approach is used in order to control each robotic group.

In Ducatelle et al. (2011a), self-organized cooperation between two different groups is studied, one formed by wheeled robots and the other by flying robots that can attach to the ceiling. Using only simulation, it is shown how this system is able to find efficient paths in complex environments. Ducatelle et al. (2011b, 2014) show an algorithm where the swarm members cooperate to dynamically find efficient paths between two targets. However, they still face congestion problems near the common targets when the number of robots is large, decreasing performance. In a very recent work, Demir et al. (2015) study how to use Markov chains to control a swarm to a certain desired spatial distribution, but such approach does not apply for the common target problem studied in this paper.

Some works, in the multi-agent systems field, use a manager agent to administrate the traffic at intersections where congestion may happen, as in Dresner and Stone (2005). Other works are dealing with manager free scenarios; Carlino et al. (2013), for instance, propose the use of auctions to manage traffic congestion for a large group of autonomous vehicles at intersections. However, these methods do not solve the problem discussed in this paper. In the common target case, robots may arrive from and depart to any direction. Besides, this target can be located in any place of an unstructured environment, not only in fixed locations such as intersections or junctions.

We also found related works in computer graphics, in the research of crowd simulations. For example, in Treuille et al. (2006), a mechanism is proposed to avoid congestion among people during the simulations of a big crowd. The authors propose an approach where agents plan early to avoid a congestion, enabling smoother trajectories than using only local repulsion forces. The method, however, is not completely decentralized, making it hard to be implemented for a swarm of robots. Besides, it focuses on the case where agents move in opposite directions, not on the case where many robots try to reach the same target.

As we can see, although there are many works dealing with traffic control and collision avoidance, there are not so many works dealing directly with the proposed problem, in which many robots converge to a common target in an unstructured environment, and must coordinate themselves in a distributed, robust and fault-tolerant fashion. This problem often happens in the navigation of swarms, for example, when they are using waypoints or foraging. In particular, we first noticed this problem during the experiments in Marcolino and Chaimowicz (2008), and it is also mentioned as a problem during the navigation of swarms in Ducatelle et al. (2011b, 2014).

We propose two decentralized coordination mechanisms that prevent congestion for a swarm of robots in the common target problem, without assuming the use of delimited lanes nor needing an external infra-structure to control the traffic. This is the main contribution of our work. Our first approach, originally introduced in Marcolino and Chaimowicz (2009), is based on a probabilistic finite state machine (Vidal et al. 2005). In this paper, firstly we revisit our algorithm, but we present novel and detailed experiments with a large number of robots that allow us to further understand and evaluate the algorithm. We also introduce a novel algorithm that surpass our original one, based on the creation of two areas that alleviate the bottleneck when robots are leaving the target region. Finally, we study the combination of both approaches, and present an extensive experimentation in the real world, with ten *e-puck* robots.

We execute many simulations in order to study the impact of the algorithms' parameters. There is a large body of work that presents microscopical and/or macroscopical models of a swarm (Lerman and Galstyan 2002; Martinoli et al. 2004; Correll and Martinoli 2006), which can be used to find optimal algorithm parameters analytically; or at least in a much faster fashion than performing simulations. However, the assumptions that are common in such models generally do not hold when the system is cluttered (Martinoli et al. 2004); thus modeling a swarm in the common target scenario is still an open problem.

3 Probabilistic congestion control

Our first solution for this problem is called Probabilistic Congestion Control Algorithm (PCC). The objective of the algorithm is to avoid that all robots be "selfish", simultaneously trying to move towards the target. In order to achieve this, each robot uses a probabilistic state machine to coordinate within a region around the common target. This solution was first proposed in Marcolino and Chaimowicz (2009), and here we refine the algorithm and perform a deeper and thorough analysis.

The basic idea is that some of the robots choose to wait before going to the target, in order to minimize the chance of robots interfering with the other members of the swarm. Therefore, a smaller number of robots try to reach the target at the same time, decreasing the congestion problem. Note that we do not prevent that two or more robots head towards the target at the same time, we only want to reduce the number of robots that try to go simultaneously. With a small number of robots at the target region, collision avoidance techniques should work well.

The solution is modeled using a *Probabilistic Finite State Machine* (Vidal et al. 2005), in which some edges are annotated with probabilities that define which transition will be

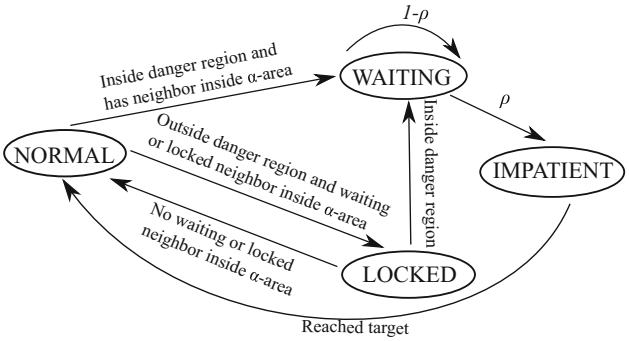


Fig. 1 Probabilistic finite state machine of the PCC algorithm

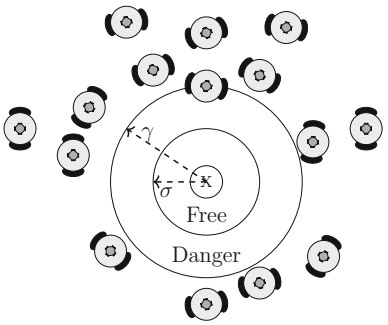


Fig. 2 *Free* and *danger* regions. "X" indicates the position of the target

taken. The state machine, presented in Fig. 1, shows the possible behaviors for each robot. There are four different states: *normal*, *waiting*, *locked* and *impatient*. From the *waiting* state, the robot can switch to the *impatient* state with probability $\rho > 0$ or stay in the same state with probability $1 - \rho$.

In order to describe these behaviors, first we have to make a few definitions. We start by defining two regions around the target: a *danger* region, with a large radius, and a *free* region, with a small radius (Fig. 2). The general idea is that the robots that reach the *danger* region must coordinate so that only few of them enter the *free* region at the same time. Upon entering the *free* region, they should move straight to the target, otherwise they could obstruct other robots, as they would be too close to the target to wait. Therefore, we define the *free* region as a circular region with radius σ around the target. Around this region, we define the *danger* region as a ring-shaped region with inner radius σ and outer radius γ.

For each robot, it is also necessary to define which other robots will be considered neighbors and hence influence its navigation to the target. Therefore, we define a sub-area in the robot's sensor region as an α-area. Considering a coordinate system centered at the robot's position with the y axis pointing towards the target, the α-area will be defined by the circular sector $[-\alpha, \alpha]$ centered in y with radius δ (see Fig. 3). The importance of defining an α-area will be explained later in this section.

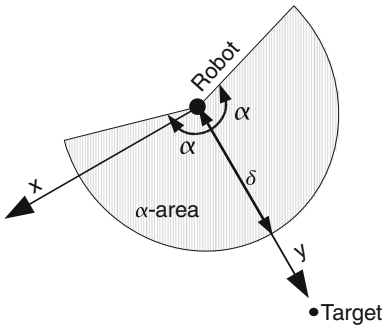


Fig. 3 Sensing area (α-area) considered by a robot to change its state

We consider that a robot detects the presence of another (and avoid collisions) when the distance between them is lower than δ. Every time a robot, i, detects the presence of another, j, it sends a message saying its target and its current state. In order to decrease the number of messages, each robot can send only one message at every ϵ iterations. Moreover, a robot will only send a message if it is inside the *danger* region or if it is in the *locked* state, which will be described later.

Finally, we introduce our algorithm, by describing the behaviors of each state in the probabilistic finite state machine. A formal description of the algorithm is shown in Algorithm 1. It works as follows: a *normal* robot j moves in the direction of the target while avoiding collisions. It can follow a simple potential field controller, such as:

$$\mathbf{u}_j = a \cdot \mathbf{t}_j - b \cdot \sum_{i \in N_j} \mathbf{r}(i, j), \tag{1}$$

where $\mathbf{u}_j$ is the control input for robot j, a and b are positive constants, $\mathbf{t}_j$ is a function that drives the robot towards the target, N_j is the set of robots in the neighborhood of j and $\mathbf{r}(i, j)$ is a repulsive function that drives robot j away of its neighbor i. The proposed algorithm does not depend on which specific functions $\mathbf{t}_j$ and $\mathbf{r}(i, j)$ are used.

When a *normal* robot j is in the *danger* region, and detects another robot i, it will check if that robot is within its α-area and if they have the same target. The constant α used in the area verification will be called α_w. If both conditions are true, j will change its state to *waiting*. This situation can be seen in Fig. 4a. It is important to define an α-area, instead of considering the full sensing range of the robot, in order to avoid that robots stop moving towards the target because of other robots behind them, in a situation where they could easily reach the target with a low risk of congestion.

A *waiting* robot will try to remain stationary in the point where it changed its state while at the same time avoiding collisions. Collision avoidance must have a higher priority than staying at the same place where it changed state; we only want to prevent that the robot changes too much its position due to the influence of other robots. Therefore, its control equation can be given by:

$$\mathbf{u}_j = 0 \cdot \mathbf{t}_j - b_1 \cdot \sum_{i \in N_j} \mathbf{r}(i, j) + b_2 \cdot \frac{\mathbf{w}_j - \mathbf{q}_j}{||\mathbf{w}_j - \mathbf{q}_j||}, \tag{2}$$

where $b_1 > b_2$ are constants, $\mathbf{q}_j$ is the current position of robot j and $\mathbf{w}_j$ is the point where the robot j changed its state to *waiting*. We kept the term $0 \cdot \mathbf{t}_j$ just to emphasize that the function driving the robot towards the target will be ignored.

```
/* Control equations                                          */
if state = normal or impatient then
    // Move towards target, while avoiding collisions
    u_j = a · t_j − b · Σ_{i∈N_j} r(i, j);
end
if state = waiting or locked then
    // Try to remain stationary, while avoiding collisions
    u_j = −b_1 · Σ_{i∈N_j} r(i, j) + b_2 · (w_j − q_j)/||w_j − q_j||
end
/* State changes                                              */
if state = normal then
    if inside danger region and has neighbor in α_w-area then
        state := waiting;
    end
    if outside danger region and has neighbor i in α_l-area then
        if state_i = waiting or locked then
            state := locked;
        end
    end
end
if state = waiting then
    if mod(iteration,η) = 0 and rand() < ρ then // Changes state after η iterations with probability ρ
        state := impatient;
    end
end
if state = locked then
    if inside danger region then
        state := waiting;
    else if there is no neighbor i in α_l-area where state_i = waiting or locked then
        state := normal;
end
if state = impatient then
    if dist_target < ε then
        state := normal;
        target := next target; // This robot will not be considered by the other robots anymore - except for collision avoidance
    end
end
```

Algorithm 1: PCC algorithm

At every η iterations, a *waiting* robot will check if it can change its state. As mentioned, the robot will change its state to *impatient* with probability ρ and will keep its state as *waiting* with probability $1 - \rho$.

An *impatient* robot moves in the direction of the target, in a similar way as a *normal* robot. However, an *impatient* robot will not stop anymore, i.e., it cannot change its state until it reaches the target. Only after the robot reaches the target, it will change its state back to *normal* (and it will move towards its next destination).

Therefore, by following this algorithm, we are able to make the robots stop and wait around the borderline of the *danger* region, decreasing the congestion. However, the robots that are outside the *danger* region would still try to navigate towards the target, and, due to the repulsion forces, would push the *waiting* robots towards the target, causing a congestion again. It is necessary to have a mechanism to force the robots outside the *danger* region to also wait before they can enter in the region.

In order to solve this problem, a *normal* robot can also change its state to *locked*. This transition will happen when the robot is outside the *danger* region and detects a *waiting* or a *locked* robot with the same target as its own. The robot still uses the α-area to evaluate which robots it should consider to change its behavior. This α-area does not need to be the same as the one in the last case (when a *normal* robot changes to *waiting*), so we will call its defining constant as α_l. In this case, we recommend the α-area to be narrower, i.e., a smaller α can be used to check whether a certain neighbor should be considered to change the robot's state. The reason is that in this case the robot only should stop moving if a robot immediately in front of it is *waiting* or *locked*.

In the *locked* state, the robot behaves in the same way as a *waiting* robot: it will not move in the direction of the target. This situation can be seen in Fig. 4b, c, where robot k stops moving in the direction of the target because robot j is in the *waiting* state in the α-area of k. The transition from the *locked* state does not depend on probabilities. A *locked* robot will switch back to *normal* when there are no more *waiting* or *locked* robots in its α-area. Here, we implement a slight variation over the algorithm proposed in Marcolino and Chaimowicz (2009): if a *locked* robot is pushed into the *danger* region by the other robots, it changes to the *waiting* state. This modification gives a slightly better performance for large groups of robots.

We can see how the system proceeds in Fig. 4d. After some iterations, the robot j resumes its movement towards the target in the *impatient* state. Moreover, robot k changes its state to *normal* and also starts to move in the direction of the target. We can see in the figure that other robots change their state to *waiting* upon reaching the *danger* region and, therefore, will not impose difficulties for robot j to reach the target, enabling a smoother navigation.

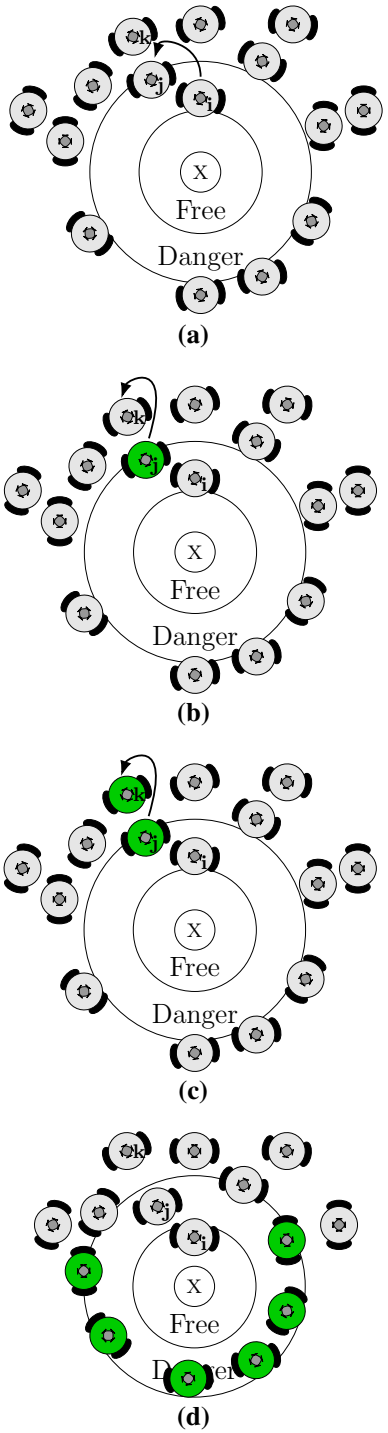


Fig. 4 Steps of the execution of the proposed coordination algorithm. *Green* (*dark*) robots are in the *waiting* or *locked* states. The *arrows* indicate message transmission (Color figure online)

It is important to mention that the proposed coordination algorithm does not depend on the knowledge of the global position of the robots. A robot only needs to know the direction and the distance to its target in order to detect whether it is in the *danger* region or in the *free* region, and must be

able to locally sense if a neighbor is in its α-area. As this algorithm is an improvement on methods where robots must move towards a target, they would already have an estimate of the direction and distance to the target in order to be able to converge to it. It is also necessary for the *waiting* robots to try to stay around the same area where they stopped. This can be achieved with a relative positioning system or even approximate methods could be used. We only want to avoid that the *waiting* robots are pushed towards different areas of the scenario due to the other robots, but it is not strictly necessary that they stay in the same place. Therefore, the coordination algorithm does not impose additional requirements to the system besides the ability to locally sense and communicate with neighbors.

The communication requirements of the algorithm can also be relaxed. We can implement this algorithm using only two kinds of messages: *warning* and *stop* messages. Messages of the type *warning* are sent by *normal* robots, to warn other robots in the *danger*-region that they might have to change their state to *waiting*; while messages of the type *stop* are sent by *waiting* and *locked* robots, to tell other robots outside the *danger* region that they might have to change their state to *locked*. This facilitates the implementation when using alternative ways of communication (such as light) instead of network packets, as only a binary information is necessary for the execution, instead of communicating all the three possible states. It would still be necessary to find a way to let the robots know the target of their neighbors, or the robots could assume that all robots have the same target (this variant, however, was not tested in our implementation).

3.1 Analysis

In this section we are going to analyze some aspects of the proposed algorithm. First, we are going to prove two important characteristics: (i) the system is effective in preventing that many robots go to the target at the same time interval; (ii) all robots eventually go to the target.

Before developing the proofs, we need to find an appropriate model for the system. The situation in which a *waiting* robot might change its state to *impatient* with a probability $\rho > 0$ or remain in the *waiting* state with a probability $1 - \rho$ can be considered a *Bernoulli trial*. Therefore, the number of robots that will change their state to *impatient* in a set of *n waiting* robots can be modeled as a binomial distribution. Let X be a random variable that defines the number of robots that change their state to *impatient* and $Pr(X)$ be the mass distribution function of the binomial distribution with n trials and probability ρ.

The robots are not necessarily synchronized, but the interval between attempts to change state is approximately equal for all robots. Hence, we will consider that, in a given time interval, all *waiting* robots will make exactly one attempt to change state. This time interval will be called an *iteration*.

Proposition 1 *Given a set of n* waiting *robots, the probability that r robots go to the target at the same* iteration *converges to zero as r gets higher.*

Proof The probability that the number of robots that will change their state to *impatient* in a given *iteration* is higher than r is given by $1 - Pr(X \leq r)$. The second term is the cumulative distribution function of the binomial, that tends to 1 as r increases. Hence, this clearly tends to zero. □

Therefore, we showed that the system is effective in preventing that many robots go to the target at the same time interval. Now we are going to show that all robots eventually go to the target.

Proposition 2 *Given a set of n* waiting *robots, the probability that all robots remain in the* waiting *state converges to zero as the number of* iterations *gets higher.*

Proof The probability that all robots will remain in the *waiting* state is given by $Pr(X = 0)$. After m *iterations*, the probability that all robots will remain in the *waiting* state is given by $Pr(X = 0)^m$, which clearly tends to zero as m gets higher since $Pr(X = 0) < 1$. □

We did not consider *locked* robots in our analysis because they will eventually move after *waiting* or *locked* robots in their α-area move. We can model this situation as a directed graph, showing the dependencies between the robots. A robot can depend on robots in front of it to move, but cannot depend on robots behind it (given that $\alpha_l < 90°$). Besides, all the α-areas of the robots are directed towards the same target, avoiding situations where an indirect cycle would be formed. As we can see, there is no cycle in the dependency graph, thus no deadlock situations will happen.

It is also important to discuss some aspects concerning the selection of the parameters. One of the most important parameters in the definition of the system behavior is ρ, the probability that a robot will leave the *w*aiting state. If it is low, the system will be "conservative" and robots might remain stationary longer than necessary. If it is high, the system will be "aggressive" and congestion situations might happen. Between these two extremes, there is a value that will minimize the time needed for task execution. This point can be estimated by an experimental evaluation. As a general guideline, if the designer expects a large number of robots trying to reach a certain target, it is better to use a smaller value of ρ. If the designer expects a small number of robots trying to reach a certain target, it is better to use a larger value of ρ.

As for the size of the *free* region, if it is small we might have a lot of *waiting* robots too near the target, which makes

it more difficult for other robots to reach and leave the target region. If it is large compared to the size of the *danger region*, the area in which robots might change their state to *waiting* will be small and congestion might happen. A similar analysis can be made for the size of the *danger* region. If it is large, robots that are far away from the target will unnecessarily cease their attempt to reach it. If it is small we will not have enough *waiting* robots to decrease the congestion problem, and they might stop too near the target, making the movement of *normal* and *impatient* robots harder. Hence, it is necessary to find a good compromise point. In Sect. 5 we perform an experimental study on the impact of all these parameters.

This algorithm focuses on avoiding congestion when entering the target. However, we noticed that there is still a problem when the robots have to exit the target and move towards their next objective. The large number of robots surrounding the *danger* region makes it hard for the robots to exit the area, and they end up causing a congestion inside the *danger* region. Therefore, we propose a new algorithm where the robots leave open areas around the target to facilitate the exit of their teammates from the target area. We are going to explain this algorithm in the next section.

4 Entrance and exit regions

4.1 Basic algorithm

Our next algorithm can either work as a complement of the previous one, or by itself, as a new approach. First, we are going to explain the part that is independent of the PCC algorithm, and later, in the next section, how the PCC algorithm can be extended. We will call our basic algorithm as Entrance and Exit Regions (EE).

Our objective is to decrease the congestion not only among the robots arriving in the target, but also exiting from it. The main idea of the algorithm is to divide the area around the target in *entry* and *exit* regions.

Hence, a circular area centered in the target is divided in four circular sectors: two defined by angle ω and two by angle β. We also define a circular area with radius γ around the target, as shown in Fig. 5. Note in the figure that the circle is divided by two lines: l, and m, as we will refer to them later in the text.

A formal description of the algorithm is shown in Algorithm 2. The algorithm affects the behavior of the robots when they are in the ring-shaped region defined by the inner radius γ and outer radius D. The area of the sectors defined by angle ω will be used as an entrance to the target region, so it will be called *entry* region. The area of the sectors corresponding to angle β will be used as an exit of the target region, and will be the *exit* region. We expect that the robots will be able to enter and exit the target region without congestion by using

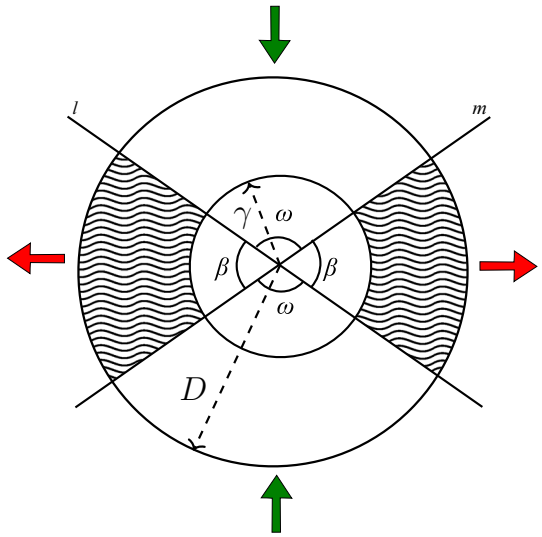


Fig. 5 Division of the region around the target in *entry* (*white*) and *exit* (*shaded*) areas

if $dist_{target} < D$ **then**
  if $dist_{target} > \gamma$ *and not in entry region* **then** // Tested using Equation 3
    $\mathbf{t}_j :=$ vector towards (x_W, y_W); // Equations 5, 6
    $\mathbf{u}_j = a \cdot \mathbf{t}_j - b \cdot \sum_{i \in N_j} \mathbf{r}(i, j)$; // Go towards entry region, while avoiding collisions
  else
    $\mathbf{t}_j :=$ vector towards target;
    if *in entry region and* $dist_{target} > \gamma$ *and not reached target* **then**
      $\mathbf{u}_j = a \cdot \mathbf{t}_j - b \cdot \sum_{i \in N_j^1} \mathbf{r}(i, j) - \frac{b}{2} \cdot \sum_{i \in N_j^2} \mathbf{r}(i, j)$; // Forces pushing the robot away of the entry region are divided by half. N_j^2 is the set of neighbors whose repulsive force pushes the robot away of the entry region, and N_j^1 is the rest of the neighbors.
    else
      if *reached target* **then**
        $\mathbf{u}_j = a \cdot \mathbf{t}_j - \frac{b}{2} \cdot \sum_{i \in N_j} \mathbf{r}(i, j)$; // Repulsive forces are divided by half
      else
        $\mathbf{u}_j = a \cdot \mathbf{t}_j - b \cdot \sum_{i \in N_j} \mathbf{r}(i, j)$; // Go towards target, while avoiding collisions
      end
    end
  end
else
  $\mathbf{t}_j :=$ vector towards target;
  $\mathbf{u}_j = a \cdot \mathbf{t}_j - b \cdot \sum_{i \in N_j} \mathbf{r}(i, j)$; // Go towards target, while avoiding collisions
end

Algorithm 2: EE algorithm

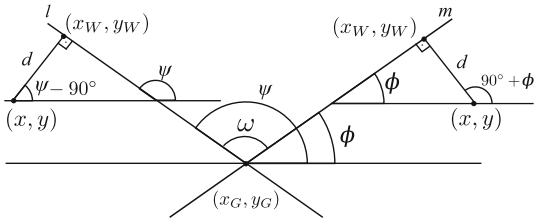


Fig. 6 Nearest distance between a point in the *exit* region and the *entry* region boundary

these different areas. The robots within a distance γ from the target will behave normally, i.e., they will either move towards the target or exit the target area towards their next destination.

When a robot is moving towards the target, and is within a distance D from it, the robot verifies whether it is outside the *entry* region. This test can be executed considering the target's position and the current robot's position, under a common global frame.

We call the target coordinates as (x_G, y_G), and the current robot's position as (x, y). We also define $\phi = 90° - \omega/2$, and $\psi = 90° + \omega/2$, as illustrated in Fig. 6. Therefore, if the distance between the robot and the target is higher than γ, the following condition determines whether the robot is inside the *entry* region:

$$\begin{cases} (y - y_G - \tan(\phi)(x - x_G) \geq 0) \wedge \\ (y - y_G - \tan(\psi)(x - x_G) \geq 0) \quad \text{if } y > y_G \\ \\ (y - y_G - \tan(\phi)(x - x_G) \leq 0) \wedge \\ (y - y_G - \tan(\psi)(x - x_G) \leq 0) \quad \text{if } y \leq y_G \end{cases} \tag{3}$$

If the robot is not in the *entry* region, it is compelled to move to the nearest point in the border of the *entry* region. The relative distance d between its current position and the nearest point is given by:

$$d = \frac{y - \tan(\theta)(x - x_G) - y_G}{\sqrt{(\tan^2(\theta) + 1)}}, \tag{4}$$

where:

$$\theta = \begin{cases} \phi \text{ if } ((y > y_G) \wedge (x > x_G)) \vee \\ \quad ((y \leq y_G) \wedge (x \leq x_G)) \\ \psi \text{ otherwise} \end{cases}$$

Equation 4 can return positive or negative values, depending on the robot's position in relation to the nearest border. That is, a negative relative distance will make the robot move towards the left, while a positive relative distance will make the robot move towards the right.

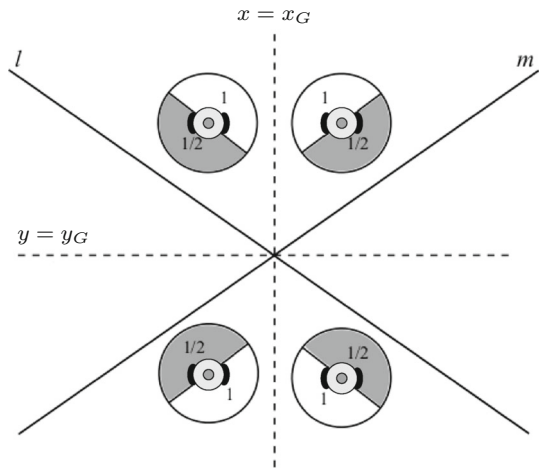


Fig. 7 Forces that push the robots away of the *entry* region are divided by half

As illustrated in Fig. 6, the point (x_W, y_W) to which the robot must move is given by:

$$x_W = x + d\cos(\vartheta) \tag{5}$$

$$y_W = y_G + \tan(\theta)(x_W - x_G), \tag{6}$$

where:

$$\vartheta = \begin{cases} 90° + \phi \text{ if } ((y > y_G) \wedge (x > x_G)) \vee \\ \quad ((y \leq y_G) \wedge (x \leq x_G)) \\ \psi - 90° \text{ otherwise} \end{cases}$$

Similarly as in the previous algorithm, each robot can follow a simple potential field controller, such as the one defined in Eq. 1. Hence, the function $\mathbf{t}_j$ will either drive the robot towards the target (if the robot is in the *entry* region) or drive the robot towards the point (x_W, y_W) (if a robot that was going towards the target happened to be in the *exit* region). As before, the proposed algorithm does not depend on which specific functions $\mathbf{t}_j$ and $\mathbf{r}(i, j)$ are used.

As expected from Eq. 1, in order to avoid collisions, the robots must also react to repulsive forces relative to their neighbors. If a robot is inside the *entry* region, the repulsive forces applied to it are divided by half if they push the robot outside the *entry* region. This reduction happens by verifying if the direction of the repulsive force vector crosses the nearest delimiting line, from inside of the *entry* region to outside (Fig. 7). This reduction does not prevent robots from exiting the *entry* region, but decreases the number of robots that are pushed away of it.

Finally, after a robot arrives at the target, the repulsive forces induced by other robots are also divided by half. This is done to facilitate the exit as the robot may come across others arriving at the target.

4.2 Extending PCC

Now we are going to explain how the PCC algorithm can work together with the EE algorithm. This version, with both algorithms running at the same time, will be called PCC-EE.

We still define the *danger* and the *free* regions of the PCC algorithm, as well as the *exit* and *entry* regions of the EE algorithm. The circular region of radius γ of the EE algorithm corresponds to the *danger* region (of same radius) of the PCC algorithm.

if $dist_{target} < D$ *and* $dist_{target} > \gamma$ *and not in entry region*
  then // tested using Equation 3
    $\mathbf{t}_j :=$ vector towards (x_W, y_W); // Equations 5, 6
    $\mathbf{u}_j = a \cdot \mathbf{t}_j - b \cdot \sum_{i \in N_j} \mathbf{r}(i, j)$; // Go towards entry region, while avoiding collisions
else
    run PCC; // The robot follows the PCC algorithm, but using the same rules for the repulsive forces as Algorithm 2: (i) forces that push the robot away of the entry region are divided by half; (ii) all repulsive forces suffered by robots exiting the target region are divided by half.
end

Algorithm 3: PCC-EE algorithm

The formal description of PCC-EE can be seen in Algorithm 3. Robots follow both algorithms in the *entry* region, but only the EE algorithm in the *exit* region. Therefore, while they are moving towards the target they must also verify whether they are in the *entry* region. If they are outside, they will move towards the nearest point in the *entry* region, following Eqs. 5, 6. Inside the *entry* region, the robots also follow the PCC algorithm. Therefore, they can change their state to *waiting* in the *danger* region, or to *locked* if there is a robot in the *waiting* or *locked* state in front of them.

We expect that the PCC-EE algorithm will be the best one in alleviating congestion, as it deals with the congestion caused by robots arriving in the target region, by using the PCC algorithm, and with the robots exiting the target region, by using the EE algorithm. In the next section we evaluate how these algorithms perform when a large number of robots must arrive in the same target, both in simulation and in real experiments.

5 Results and discussion

In order to evaluate our algorithms, we executed simulations and experiments with real robots. The proposed algorithms were tested in simulation using the *Stage* robot simulator

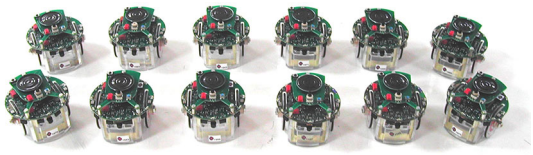


Fig. 8 *E-puck* robots used in the experiments

(Gerkey et al. 2003). The real experiments were performed using ten *e-puck robots*. The e-puck is a small-sized (7 cm diameter) differential drive robot that is very suitable for swarm experimentation (Cianci et al. 2007). Each robot is equipped with a ring of eight IR sensors that allows proximity sensing and a group of colored LEDs to indicate robot status. Local processing is performed by a dsPIC microprocessor and a bluetooth wireless interface allows remote control. Figure 8 shows the robots used in our experiments. Both in the simulations and in the real experiments, we consider non-holonomic robots using control equations based on Luca and Oriolo (1994). Also, in our implementation, we considered repulsive forces generated by the following equation (Siegwart and Nourbakhsh 2004):

$$\mathbf{F} = \begin{cases} -K \times \left(\frac{1}{d} - \frac{1}{I}\right)\left(\frac{\mathbf{o}-\mathbf{p}}{d^3}\right) & \text{if } d < I \\ 0 & \text{otherwise} \end{cases}$$

where $K > 0$ is a constant, $\mathbf{p} = [x, y]^T$ the current robot's position, $\mathbf{o} = [x_o, y_o]^T$ the neighbor's position, $d = ||\mathbf{o}-\mathbf{p}||$ the euclidean distance between $\mathbf{o}$ and $\mathbf{p}$, and I the influence radius, i.e., the maximum distance that a robot can detect its neighbor.

5.1 Collision avoidance

Before introducing our results, we first study the behavior of optimal reciprocal collision avoidance (ORCA), a state of the art collision avoidance algorithm (van den Berg et al. 2011). We used the open source library RVO2 in our implementation (van den Berg et al. 2015). The objective of this study is to motivate the need for our algorithms, and show that local repulsion forces is a reasonable baseline for comparison.

The basic idea of ORCA is to define the collision avoidance problem as a linear optimization problem. By inferring the velocities of the neighbors, each robot is able to solve a linear program in order to calculate its own velocity, guaranteeing a collision-free navigation (assuming that all robots are also doing the same).

Unfortunately, ORCA is not able to perform well in the common target problem. We tested two different parametrizations, shown in Table 1. The description of these parameters are available in the documentation of the RVO2 library. The "safe" parametrization allows the robots to prop-

Table 1 Parameters used in the ORCA executions

Parameter	Safe	Unsafe
Neighbor distance	3.9	2.4
Maximum neighbors	60	23
Time horizon	2	2
Robot radius	1.8	1.8
Maximum speed	2.5	2.5

erly avoid collisions, but the system reaches an equilibrium state where no robot is able to converge to the target. The "unsafe" parametrization has many collisions, but all robots are able to reach the target. We parametrized for 20 robots. Note that although ORCA theoretically guarantees an execution free of collisions, in practice we still have collisions in our executions, due to our parametrization. However, even with a parametrization that still has a few collisions ("safe"), the system was already not able to converge.

For instance, in one of our executions we found that in the *unsafe* case, all robots were able to reach the target after 5742 iterations, with 12 collisions. In the *safe* case, all robots were still *not able* to reach the target after 696,817 iterations, with no collisions. For comparison, using only local repulsion forces all robots are able to reach the target with only 2472 iterations, and no collisions.

In order to better understand ORCA's behavior, we show in Fig. 9 screenshots of one execution with 40 robots, in the *safe* case (red robots are moving towards the target, while blue robots already reached the target). Figure 9a shows the initial position (the target is in the center of the scenario). The robots move towards the target (Fig. 9b). We can see that some robots form a circle around the target, while others move towards the target, inside the circle. The robots inside the circle are able to eventually reach the target and move towards their next goal (Fig. 9c, d). However, the other robots maintain a circular motion around the target area, and no robot moves towards the target anymore (Fig. 9e, f). Besides,

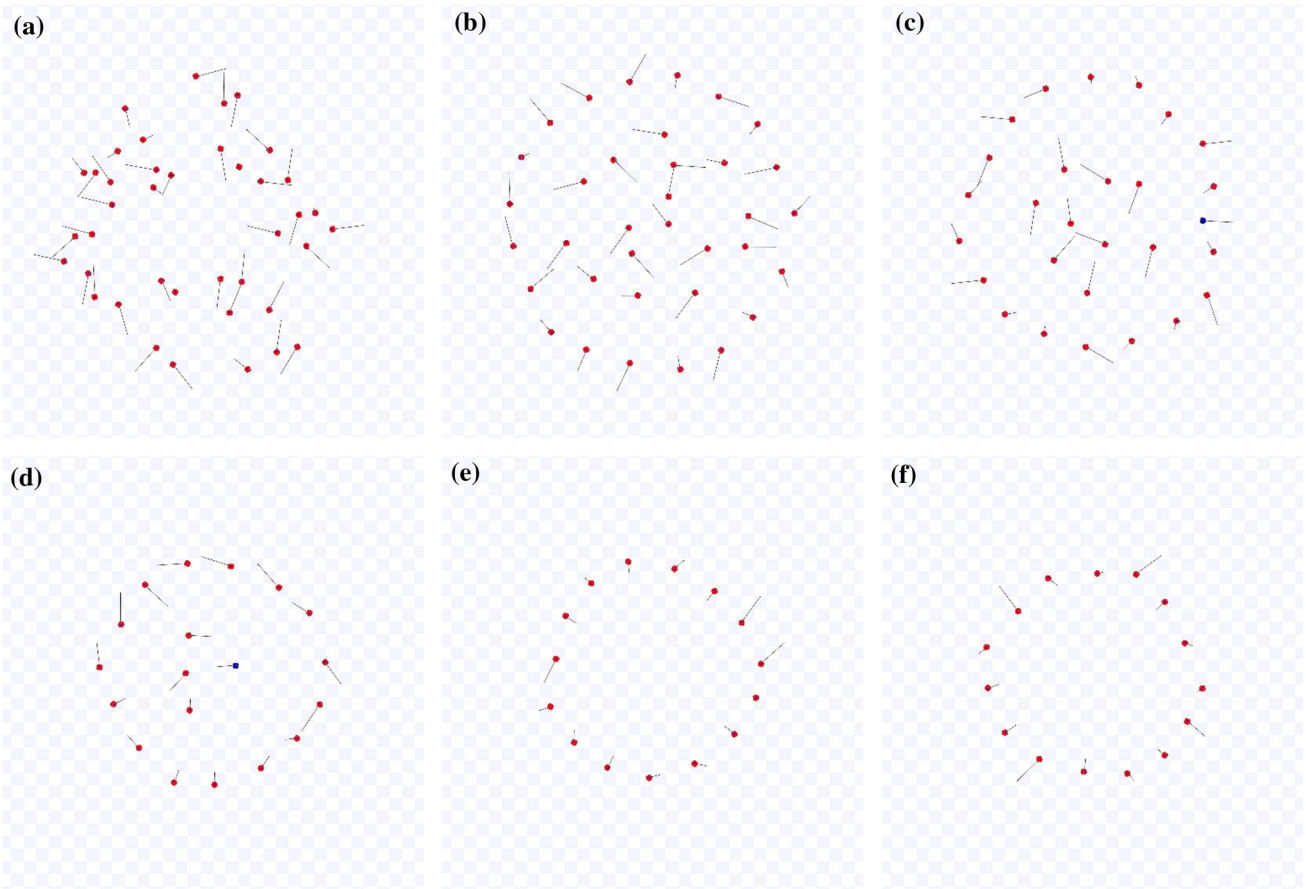


Fig. 9 Execution screenshots of the ORCA algorithm (video available at https://youtu.be/gDXrHgrb7q4). **a** *0 s* Initial position. **b** *5 min 28 s* Some robots approach the target area, while others start forming a *circle* around the target. **c** *10 min 57 s* The robots in the center of the *circle* are able to reach the target eventually. We see one of them leaving the region in the *right hand side*. **d** *16 min 26 s* More robots are able to leave the target region, and the center becomes less crowded. **e** *21 min 56 s* After the center is empty, the other robots still circulate around the target. **f** *27 min 25 s* Robots keep circulating around the target instead of converging to it

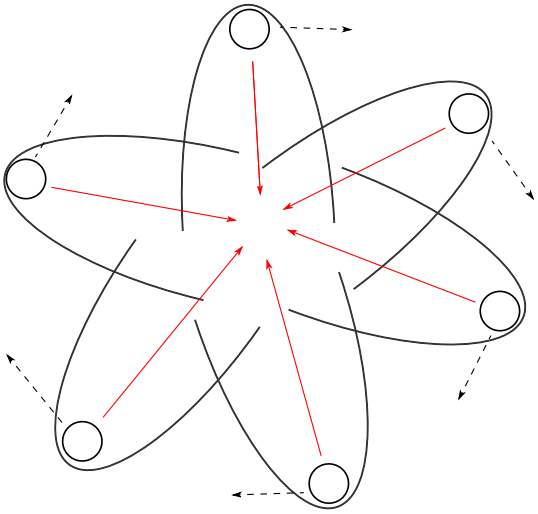


Fig. 10 ORCA reaches an equilibrium state in the common target problem

Table 2 Parameters used in the simulations

Parameter	Meaning	Value
I	Influence radius	2 m
—	Communication radius	3 m
γ	Radius of *danger* region	3.5 m
σ	Radius of *free* region	1.5 m
D	Radius of region where EE is applied	10 m
ω	Angle of *entry* region	120°
α_w	Angle of α-area for *waiting* robot	115°
α_l	Angle of α-area for *locked* robot	45°
δ	Radius of α-area	3 m
ϵ	Number of cycles before sending a message	25
η	Number of cycles for testing if a *waiting* robot will change state	40

we found that even in the *unsafe* case, most of the executions were not successful with 40 robots, due to a high number of collisions between the robots.

We explain briefly why ORCA does not work well in the common target problem. In Fig. 10 we show a diagram with 6 robots, represented by the circles. The red (continuous) arrows show the robots desired velocities (towards the target). We represent by the parabolic region the set of velocity vectors that will be avoided by the robots in order to avoid collisions, according to the ORCA algorithm. Hence, the resulting velocity vector will be perpendicular to the desired velocity, for each robot. As this process continues at each iteration, all robots will execute a circular motion around the target, instead of moving towards the target.

As we can see, although ORCA is a state of the art collision avoidance algorithm, it is not suitable for the common target problem. Hence, in our next experiments, we will compare our algorithms against executions using only local repulsion forces.

We also tested using ORCA *with* our algorithms, in order to avoid collisions while the robots run our coordination methodologies (instead of using local repulsion forces to avoid collisions in our algorithms). However, we found that all robots are still not able to converge to the target. Hence, we will use local repulsion forces to avoid collisions when running our algorithms. For the interested readers, we show in the 'Appendix' a screenshot of the PCC-EE algorithm with ORCA.

5.2 Simulations

We considered scenarios where the robots are initially in random positions, but distant from the *danger* region. After reaching the common target, the robots will move towards a next target, that will be either to the left or to the right of the common one. This decision is taken based on a uniform probability, so we can expect that about half of the robots go to the left, and half to the right. The new targets are aligned with the common one, but far away in the x axis. In Table 2 we show all the parameters of the simulations. Besides, we always use a normalized force towards the target, with norm equal to 2.5. The repulsion forces are proportional to the relative distances, and multiplied by 0.5.

Before presenting our experimental analysis, we will show execution screenshots of all algorithms, with 140 robots. We first show, in Fig. 11, an execution without using any coordination algorithm, only potential fields. Robots that are going towards the target are shown in red, robots that reached the target in yellow and robots that completed the execution (i.e., are further than 10m from the target) in black. In the beginning of the execution, the robots are initially located encircling the *danger* region, as shown in Fig. 11a. As we can see from Fig. 11a–f, the robots surround the common target but the execution is very inefficient, as they compete to reach the target and are repelled by their repulsive forces. Moreover, it is hard for robots that already arrived in the target to exit the area.

In Fig. 12, we present an execution of the PCC algorithm. Robots in *normal*, *waiting*, *locked*, and *impatient* states are represented by the colors red, green, cyan, and blue, respectively. Figure 12b shows when the first robots enter the *waiting* state. Soon the robots behind them change to *locked*, and also stop moving towards the target, while one robot becomes *impatient* (Fig. 12c). In Fig. 12d, we can see that some robots are able to reach the target, while more robots change to *impatient*. However, it is hard for the robots to leave the area of the target; we can see in Fig. 12e that many *impatient* robots end up accumulating in the *danger*

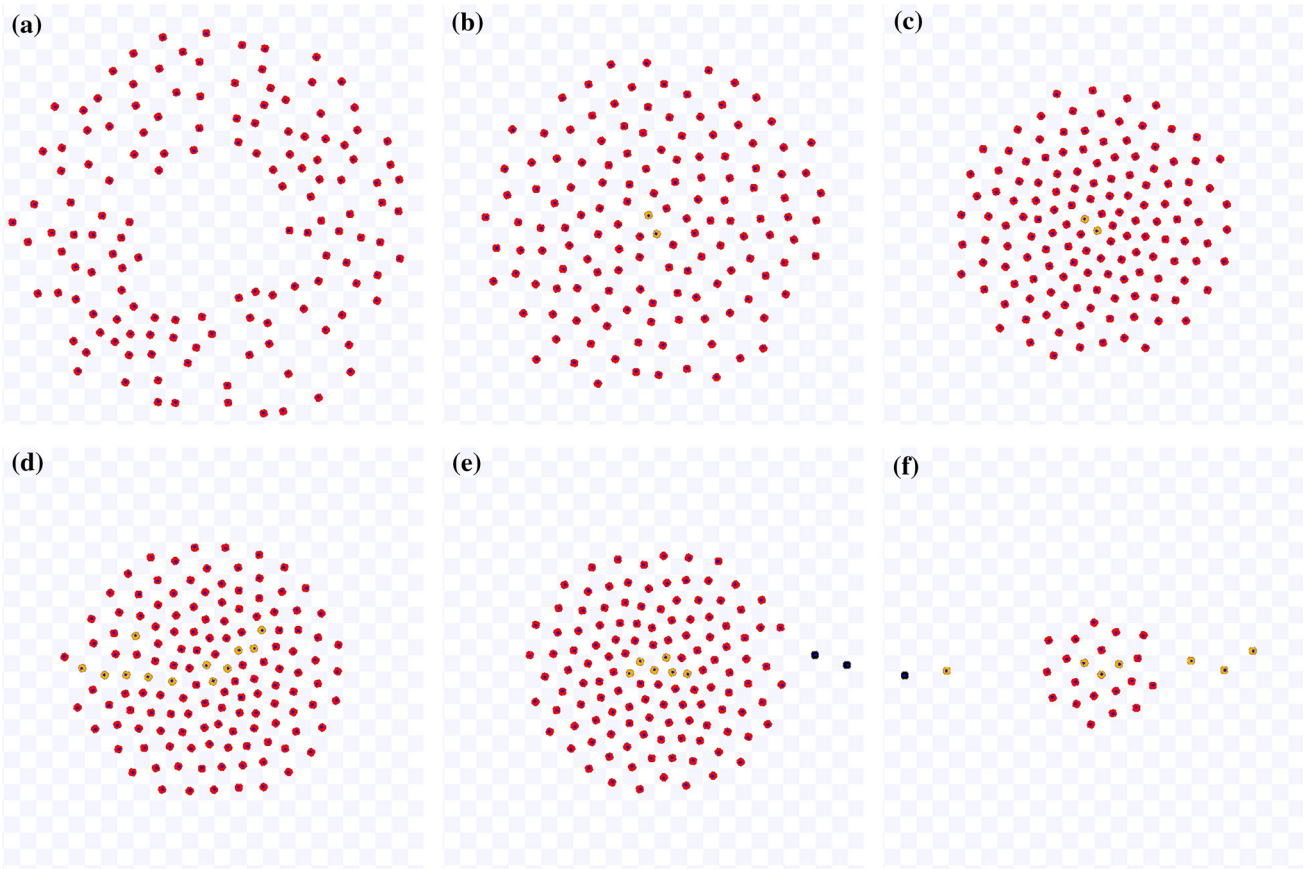


Fig. 11 Execution screenshots without any coordination algorithm (video available at https://youtu.be/4tE4aka24QE). **a** *0 s* Beginning of the execution. **b** *12 s* Robots simultaneously try to reach the target, forming a circular shape. **c** *28 s* Shape gets more compact, but still no robot is able to exit. **d** *9 min 59 s* After many iterations, robots can finally exit. **e** *13 min 34 s* Other robots still get stuck in the central area. **f** *1 h 3 min 19 s* Towards the end, robots are still competing for the center

region. Eventually the robots are able to reach the target, and move to the next one (Fig. 12f).

We now show screenshots of the EE algorithm (Fig. 13). As we can see, the robots quickly open the *exit* region (Fig. 13b), and the first robots are able to reach the target (Fig. 13c). The robots can easily exit the area (Fig. 13d, e), and soon there are only a few robots left (Fig. 13f).

Finally, Fig. 14 shows an execution of the PCC-EE algorithm. As before, robots in *normal*, *waiting*, *locked*, and *impatient* states are represented by the colors red, green, cyan, and blue, respectively. In Fig. 14a, we have the beginning of the execution. Some iterations later, as we see in Fig. 14b, the *entry* and *exit* regions are starting to get formed, since robots move away of the east and west areas and concentrate in the north and south areas. At the same time, we can notice that the robots in the *danger* region change to the *waiting* state, and the ones outside the *danger* region to *locked*. In Fig. 14c, we can see that the division in *entry* and *exit* regions gets more pronounced, while some robots could already reach the target and are now trying to exit the area around it. As we can see in Fig. 14d, e, they can easily exit the area and soon we reach the situation in Fig. 14f, where there are only a few robots left.

We now present our experimental analysis. We measure the number of iterations needed for the last robot to go to the target and reach at least D meters away from it. All simulations were executed 40 times, and we calculated the mean and the confidence interval of the results, with p value equals to 0.01 (shown by the error bars in the graphs, unless otherwise noted).

We performed an extensive experimental study of the impact of some of the algorithms' parameters, which we will present later in this section. First, we present the performance of the proposed algorithms as the number of robots increase (measured in terms of number of iterations to exit the target area, as mentioned). We used $\rho = 0.035$ in PCC, the best parameter found for 100 robots. For the PCC-EE algorithm, we used $\rho = 0.15$, again the best found for 100 robots. We can see the results in Fig. 15. As can be observed, all algorithms are significantly better than an approach without

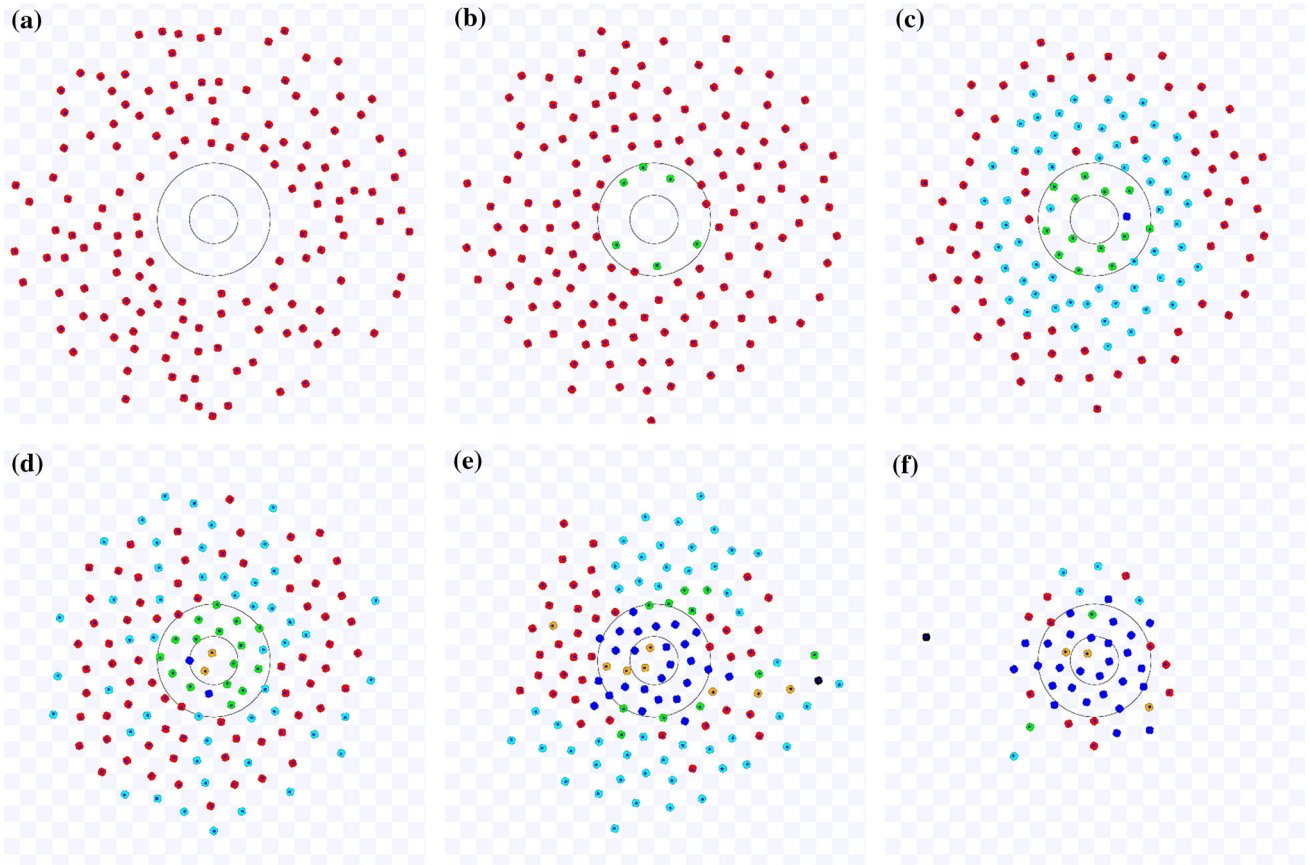


Fig. 12 Execution screenshots of the PCC algorithm (video available at https://youtu.be/wutQn7laEOU). **a** *0 s* Beginning of the execution. **b** *5 s* Some robots change to *waiting*. **c** *10 s* Robots around *danger* region change to *locked*, while one robot becomes *impatient*. **d** *36 s* More robots become *impatient*, while a few reach the target. **e** *5 min 57 s* Robots have difficulty to exit the target region, so the area still gets cluttered. **f** *34 min 19 s* Later execution, after most of the robots could reach the target

coordination (labeled as "NoCoord"), after a certain number of robots. For 140 robots, the PCC algorithm had an improvement of about 40 %, while the EE and the PCC-EE could reach an improvement of about 76 %. We can also note that both the approach without any coordination and the PCC algorithm tended to increase exponentially as the number of robots increase, while the EE (and the PCC-EE) increase in a more linear fashion.

As can be seen, the performance of the PCC-EE and the EE algorithm was very similar, but the PCC-EE seems to get better than the EE as the number of robots increase. In fact, if we use the best ρ value found for 140 robots ($\rho = 0.08$, as we will show later), the PCC-EE performs about 7 % better than the EE algorithm, with statistical significance (p value equals to 9.349×10^{-7}).

Figure 16 shows the number of messages sent during the execution of all algorithms. The number of messages of the PCC and the PCC-EE algorithm seems to increase in a quadratic way, as the number of robots increase. The best quadratic model found for the curves was $y = 0.96635x^2 + 9.11628x + 503.93571$ for the PCC algorithm, and $y = 0.3337x^2 + 8.6183x - 19.3714$ for the PCC-EE algorithm. Both fittings have an adjusted coefficient of determination (adjusted R^2) of 0.9978 and 0.9995, respectively. For the PCC algorithm, the best linear model was $y = 163.73x - 4134.54$, while for the PCC-EE was $y = 62.013x - 1621.214$. This time, the adjusted coefficient of determination (R^2) of the curves was 0.9501 and 0.9593, respectively. Even though the best model seems to be a quadratic, we can see that the quadratic term is quite small. Therefore, these algorithms should scale well. As in the PCC-EE algorithm the robots can exchange messages only inside the *entry* region, the number of messages was much lower than in the PCC.

The EE algorithm does not require message exchange during its execution. So, it is equivalent to the approach without any coordination in terms of the number of messages, but it is almost as efficient as the PCC-EE, and much more efficient than the PCC. Therefore, the EE algorithm is highly efficient and scalable. In order to verify that the performance improve-

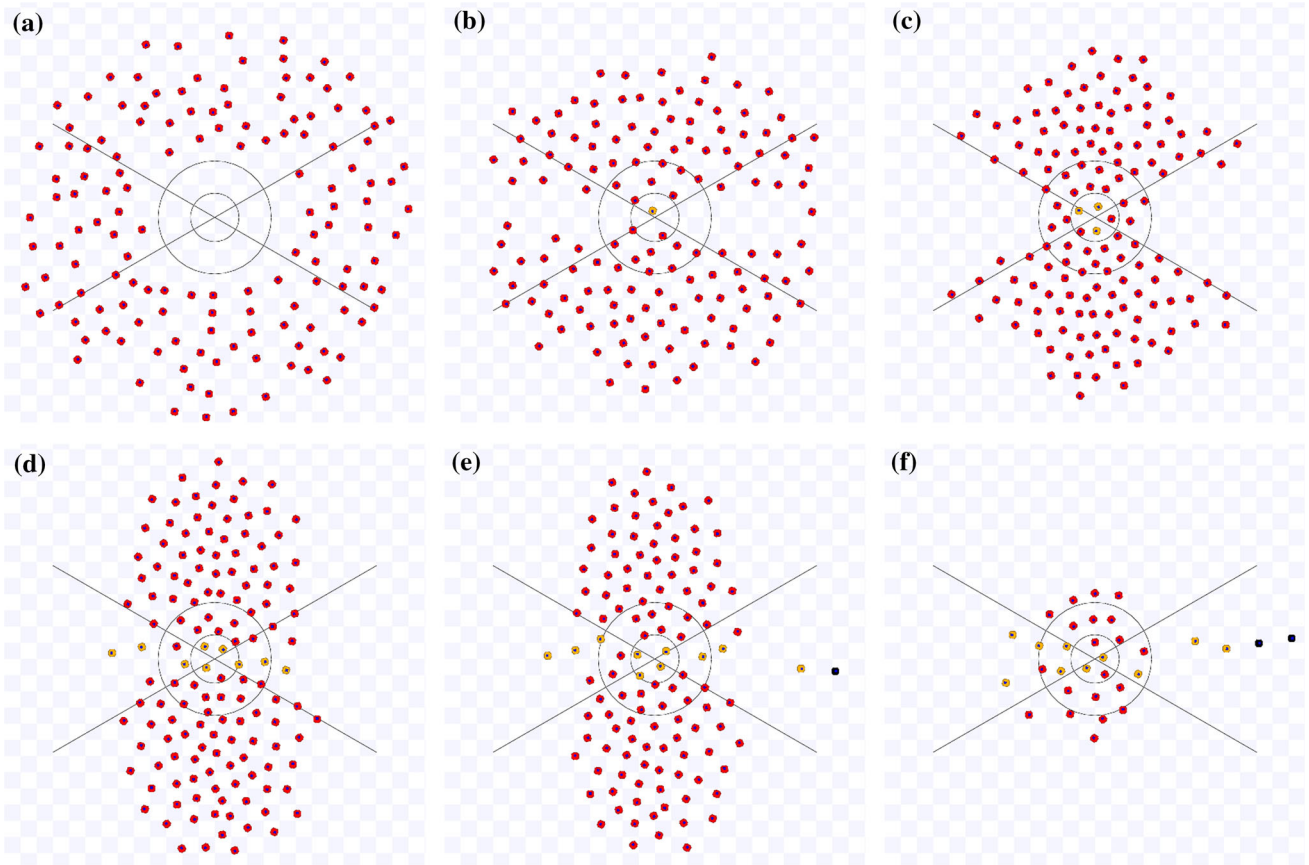


Fig. 13 Execution screenshots of the EE algorithm (video available at https://youtu.be/jte3NRS9pQg). **a** *0 s* Beginning of the execution. **b** *13 s* Formation of *entry* and *exit* regions. **c** *35 s* A few robots reach the target. **d** *1 min 25 s* Robots can easily exit the target area. **e** *4 min 09 s* Robots keep exiting easily. **f** *17 min 31 s* Later execution, only a few robots are left

ment of the EE algorithm is caused mainly by a faster exit of the robots in the target area, we calculated the mean and the standard deviation of the number of iterations each robot takes to reach a distance D for one execution, after arriving at the target. The result can be seen in Fig. 17, where this time the bars show the standard deviation of the results. As can be observed, the robots in the EE algorithm could leave faster the target region.

The previous results were obtained after performing experiments to study the impact of the algorithms' parameters. We present now our results involving ρ in the PCC and PCC-EE algorithm. This experimentation consists in measuring the number of iterations to complete execution for different ρ values.

We can see the result for the PCC algorithm in Fig. 18. As expected, there is a local minimum in the graph, and either a bigger or a smaller ρ leads to a worse performance.

We executed the same experiment to determine the optimal value of ρ in the PCC-EE algorithm. This time, we also studied the case with 140 robots, in order to more clearly identify if the PCC-EE could overcome the EE algorithm (as mentioned earlier). The result can be seen in Fig. 19. As we can see, ρ has a lower influence in the algorithm's performance than in the PCC case. Even with $\rho = 1$, the robots present a very good performance, and the presence of a local minimum is not as clear as in the last case. We can show, however, that $\rho = 0.15$ for 100 robots is about 6 % better than $\rho = 1$ with p value equals to 0.0002235.

We also studied the impact of the size of the *danger* region on the PCC and the PCC-EE algorithms, which are shown in Fig. 20a and b, respectively. For the PCC algorithm, a radius (γ) of size 4 is 13, 4 and 10 % better than a radius of size 2, for 80, 100 and 120 robots, respectively. Two of these results are statistically significant, with the following p values: 4.334×10^{-6}, 0.2399, 8.461×10^{-4}, respectively. For the PCC-EE, the impact is less significant. A radius of size 4 is only 5, 8 and 5 % better, for 80, 100 and 120 robots, respectively. The p values are, respectively: 0.03773, 2.76×10^{-5}, 0.008034.

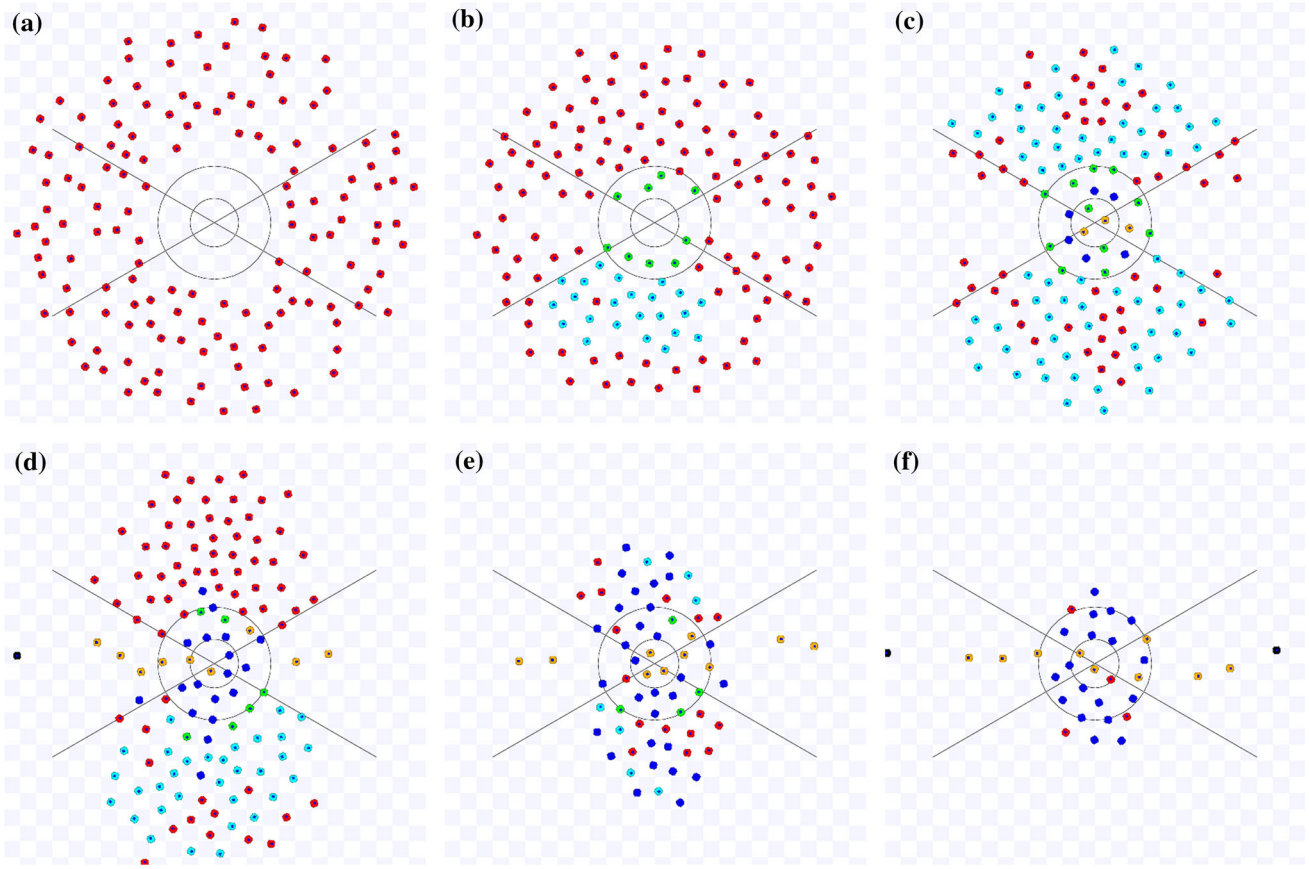


Fig. 14 Execution screenshots of the PCC-EE algorithm (video available at https://youtu.be/V_cqJRFcwvo). **a** *0 s* Beginning of the execution. **b** *10 s* Formation of *entry* and *exit* regions. **c** *30 s Exit* region gets more pronounced. **d** *2 min* Robots easily exit the target area. **e** *11 min 3 s* Robots keep exiting easily. **f** *15 min 35 s* Later execution, only a few robots are left

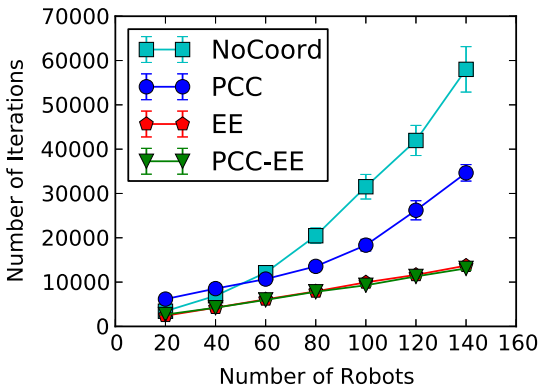


Fig. 15 Execution time for the algorithms

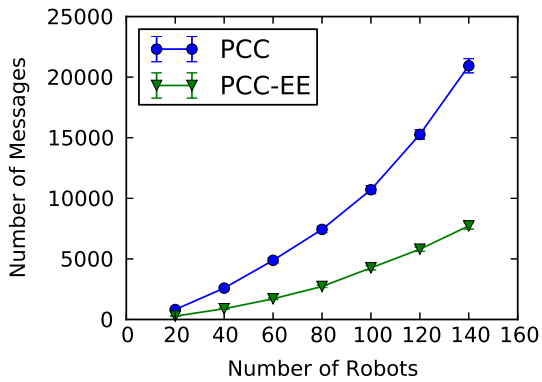


Fig. 16 Number of sent messages

5.3 Real experiments

The proposed algorithms were thoroughly tested with the *e-puck* robots, in order to evaluate them in a real life environment, with all the localization, communication and actuation errors.

We used the system for robotic swarms localization inside internal environments by Garcia and Chaimowicz (2009). With this system, a robot's position can be considered in a global referential, making straightforward the implementation of the algorithms in the way described in Sects. 3, 4. Unfortunately, the *e-pucks*'s infrared sensors have a low range. Due to this, we implemented "virtual sensors" from

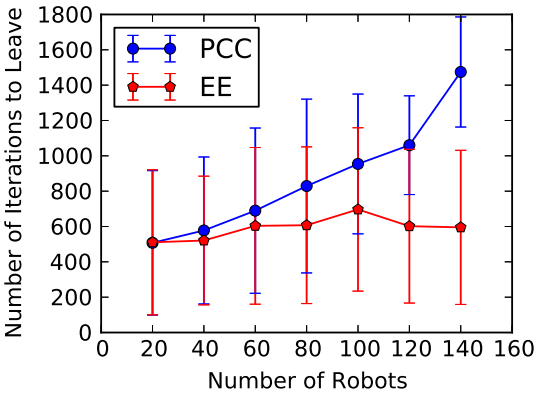


Fig. 17 Time used by the robots to leave the target region. The *bars* show the standard deviation

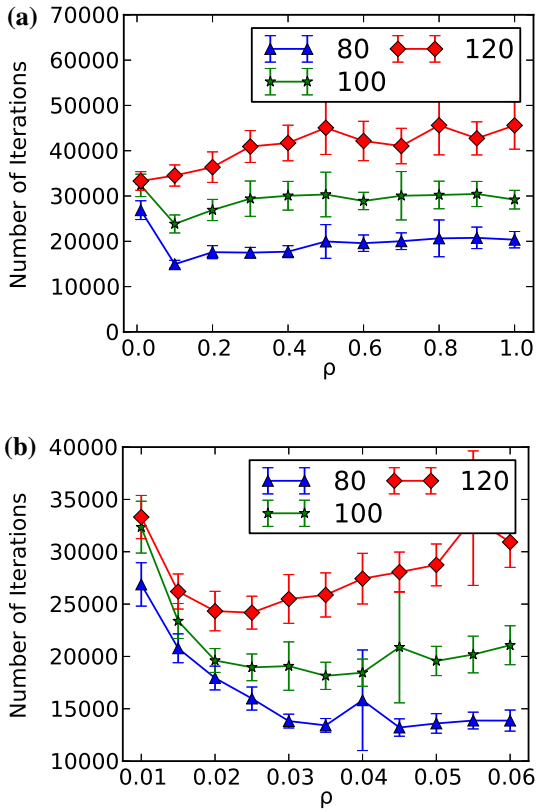


Fig. 18 Execution time for the PCC algorithm, varying ρ and number of robots. **a** From $\rho = 0.01$ to $\rho = 1.0$. **b** From $\rho = 0.01$ to $\rho = 0.06$

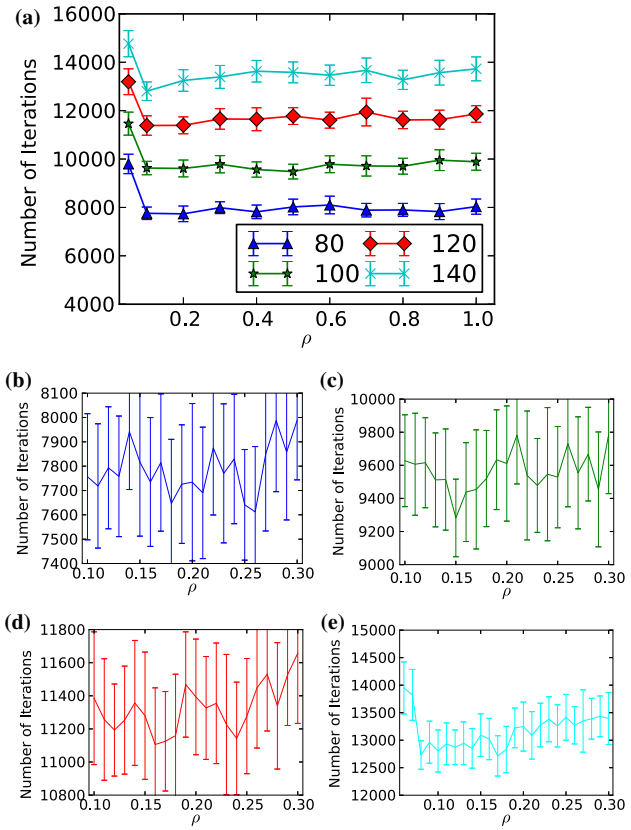


Fig. 19 Execution time for the PCC-EE algorithm, varying ρ and number of robots. **a** From $\rho = 0.05$ to $\rho = 1.0$. **b** 80 Robots, from $\rho = 0.10$ to $\rho = 0.30$. **c** 100 Robots, from $\rho = 0.10$ to $\rho = 0.30$. **d** 120 Robots, from $\rho = 0.10$ to $\rho = 0.30$. **e** 140 Robots, from $\rho = 0.06$ to $\rho = 0.30$

the positions obtained by the localization system. The parameters used in the real experiments can be seen in Table 3. Besides, we always use a normalized force towards the target, with norm equal to 2.5. The repulsion forces are proportional to the relative distances, and multiplied by 5.

We start by showing some example executions, and then we are going to analyze the performance after many samples. Figure 21 shows some images of the execution of the algorithm using only local repulsion forces. We show by a dashed yellow circle the robots that were able to reach the target, and are now moving towards their next waypoint (we do not put a circle in the robots that are already in the next waypoint or are very near it). We can see the initial configuration in Fig. 21a. Figure 21b presents all robots going towards the common target. Because this algorithm has no coordination, they move towards the target using only repulsive forces to avoid collision with their neighbors. Hence, the system soon becomes cluttered, as we can see in Fig. 21c. Even after some robots are able to reach the target, it is hard for them to exit from the target region (Fig. 21d, e). Finally, Fig. 21f presents the stage when most of the robots were able to reach the target.

Now we show in Fig. 22 some images of an execution of the PCC algorithm. Robots in the *waiting* mode are indicated by a green circle, and *impatient* robots by a blue star. We can see the initial state in Fig. 22a. Figure 22b shows some seconds later, when the first robots changed to the *waiting* state. As more robots approach the *danger* region, we soon have two more robots *waiting*, while others move towards the target, as we see in Fig. 22c. In Figure 22d, we see robots in the *impatient* state moving towards the target, while others that already reached the target try to move away towards their

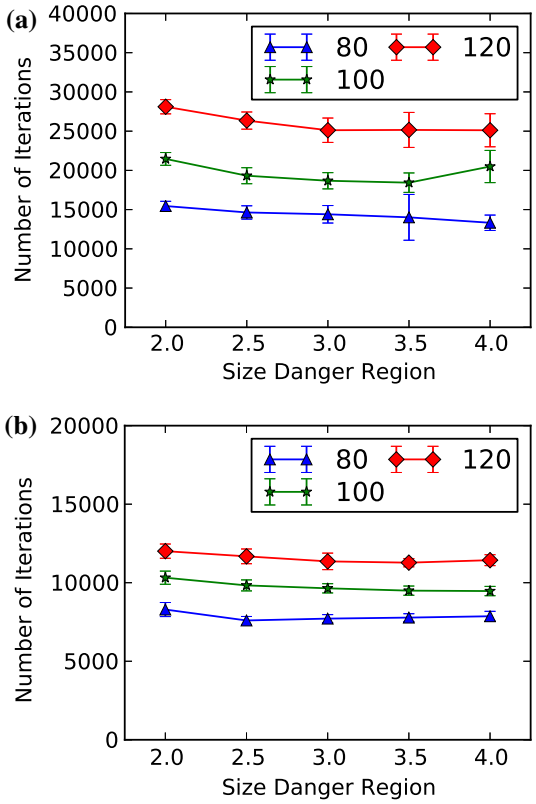


Fig. 20 Influence of the size of the *danger* region

Table 3 Parameters used in the executions with real robots

Parameter	Meaning	Value
I	Influence radius	30 cm
—	Communication radius	30 cm
γ	Radius of *danger* region	40 cm
σ	Radius of *free* region	10 cm
D	Radius of region where EE is applied	70 cm
ω	Angle of *entry* region	90°
α_w	Angle of α-area for *waiting* robot	115°
α_l	Angle of α-area for *locked* robot	45°
δ	Radius of α-area	30 cm
ϵ	Number of cycles before sending a message	2
η	Number of cycles for testing if a *waiting* robot will change state	4

next objective. One of them is finally able to exit the region, as we see in Fig. 22e. Meanwhile, others still try to exit the target area while *impatient* robots move towards the target. Finally, Fig. 22f illustrates a later stage when most of the robots already reached the target.

The execution with the EE algorithm can be seen in Fig. 23. We show by a dashed red square the robots in the *exit* region that are moving towards the *entry* region. Figure 23a

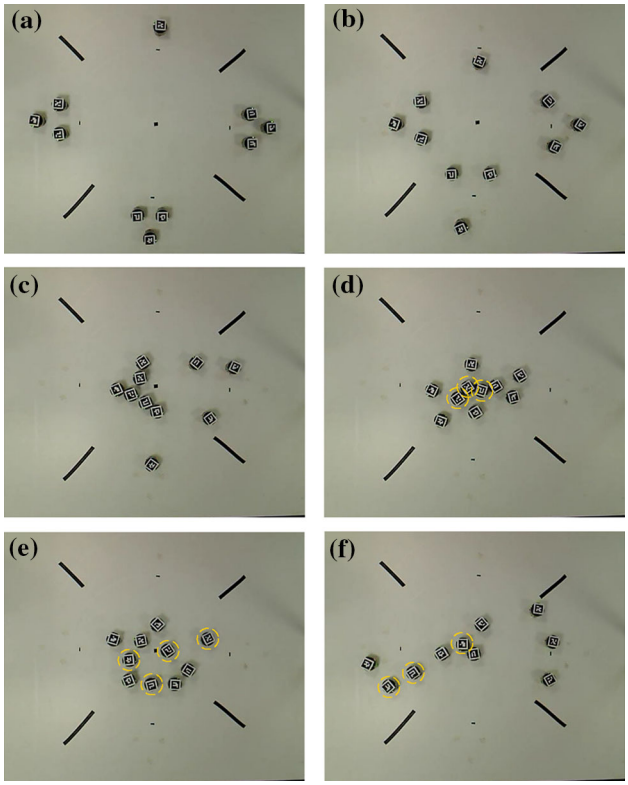


Fig. 21 Experiment with *e-pucks* using only local repulsion forces (video available at https://youtu.be/wl90yB9qla8). **a** *0 s* Initial position. **b** *23 s* Robots move towards target. **c** *41s* System gets cluttered, as all robots try to reach the target. **d** *1 min 03 s* Robots still compete for the center. **e** *3 min 35 s* It is hard for the robots to exit the target area. **f** *6 min 11 s* Later stage, when most robots could reach the target

shows the initial state, while Fig. 23b shows four robots in the *exit* region moving towards the *entry* region. With the area around the target more free, the robots that reach the target can easily move towards their next objective, as we show in Fig. 23c–e. Finally, the state where most of the robots could complete the execution is shown in Fig. 23f.

Finally, the execution of the PCC-EE algorithm can be seen in Fig. 24. Again, the initial configuration is shown in Fig. 24a. Figure 24b shows some seconds later, when five robots go towards the *entry* region, while at the same time two robots already in the *entry* region change to the *waiting* state. More robots change to *waiting*, while only two are left in the *exit* region, as we see in Fig. 24c. With the *exit* region free, we can see a robot easily going towards its next objective, in Fig. 24d, while *impatient* robots approach the target. In Fig. 24e, we can see more robots easily exiting the area, while other robots wait nearby. Finally, Fig. 24f illustrates a later stage, after almost all robots completed the execution.

We also executed an experimental analysis of all algorithms with real robots. We repeated 10 times the execution of each algorithm, using the same parameters as in the pre-

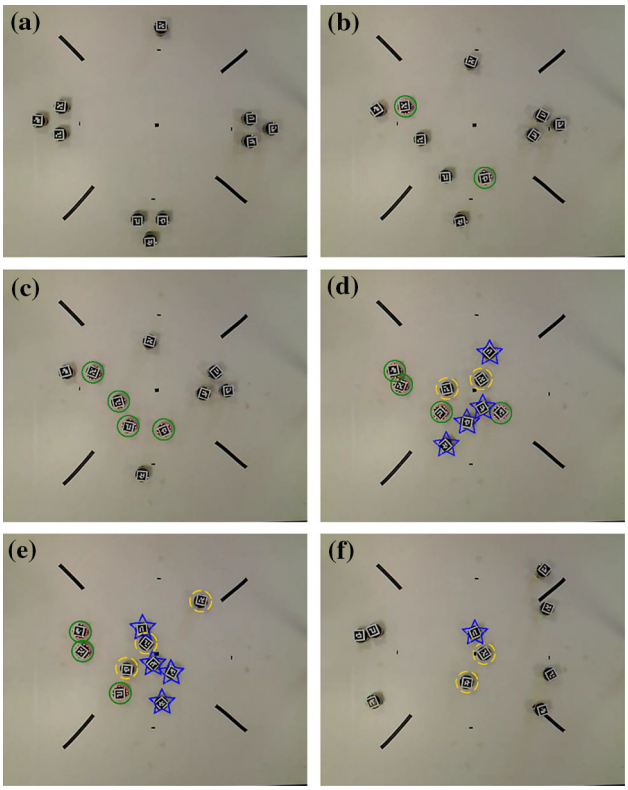


Fig. 22 Experiment with *e-pucks* using the PCC algorithm (video available at https://youtu.be/Rabf5Jrbd1A). In the video, robots with all the leds on are in the waiting or locked state. **a** *0 s* Initial position. **b** *26 s* Some robots change to *waiting*. **c** *31 s* More robots change to *waiting*, while others approach the target. **d** *1 min 29 s* Two robots are able to reach the target, while more get nearby. **e** *2 min* It is hard for the robots to exit the area of the target, but one approaches its next objective. **f** *6 min 33 s* Later stage, when most robots could reach the target

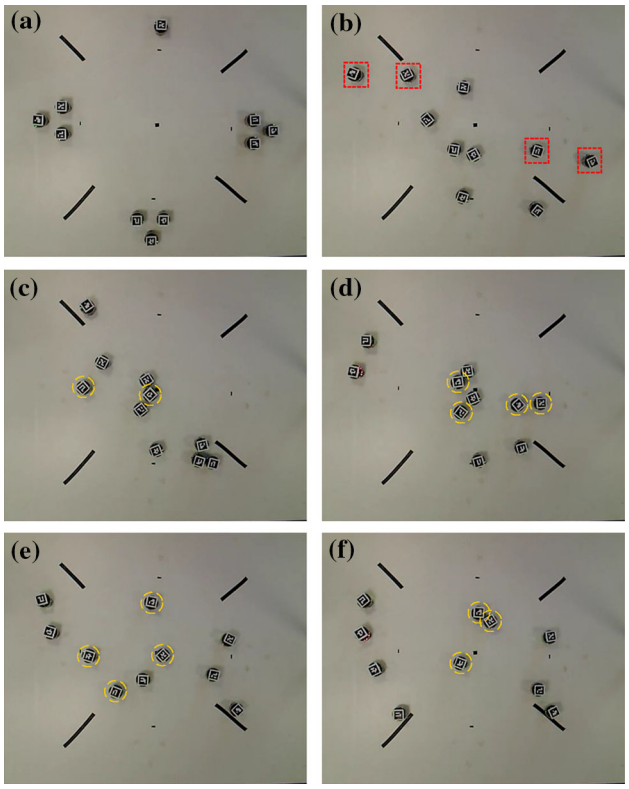


Fig. 23 Experiment with *e-pucks* using the EE algorithm (video available at https://youtu.be/XWgF4a4SdVs). **a** *0 s* Initial position. **b** *47 s* Robots move towards the *entry* region. **c** *1 min 12 s* Two robots are able to reach the target, and move towards their next objective. **d** *3 min 0 2s* More robots reach the target, and can easily use the *exit* region. **e** *4 min 57 s* Almost all robots are already in the next objective or moving towards it. **f** *5 min 16 s* Later stage, when most robots completed execution

vious example executions. However, for the PCC and the PCC-EE algorithms, we run executions with $\rho = 0.06$ and with $\rho = 0.8$ (we choose $\rho = 0.8$ since it seemed to give the best result in our preliminary experiments). The result is shown in Fig. 25, where the bars indicate the confidence intervals with p value 0.01. As we can see, all algorithms had a better performance than only using local repulsion forces, especially when $\rho = 0.8$. We can show that the PCC, PCC-EE and EE algorithms are 22, 14 and 17 % better than using only local repulsion forces, respectively, with p values equal to 0.001198, 0.02873, 0.008517.

Concerning a comparison between the proposed algorithms, we find that EE is statistically significantly better than the PCC and the PCC-EE algorithms with $\rho = 0.06$ (p values equal to 0.007761 and 0.02472, respectively); however the difference between the EE and the PCC-EE with $\rho = 0.8$ is not statistically significant (p value equal to 0.5709). It seems that the PCC with $\rho = 0.8$, however, is better than the EE, but the p value is 0.1622. Hence, with a p value <0.1 the difference between the algorithms is still not statistically significant.

Therefore, in the real executions the PCC algorithm seems to have the best performance, even though in our simulations the PCC-EE and the EE algorithms performed significantly better. One possible explanation is that the focus of the EE algorithm is in avoiding congestion when the robots are exiting the target region, but this problem does not affect the execution much if the number of robots is not large. Notice, for example, that as shown in Fig. 17, the robots using the PCC and the EE algorithms needed a very similar number of iterations to leave the target area for an execution with 20 robots, and the difference between the algorithms increased as the number of robots increased. Hence, as our real experimentation was with 10 robots, we still could not observe an improvement of the EE over the PCC algorithm (and, consequently, also of PCC-EE over PCC).

Nevertheless, the performance of the EE and the PCC algorithms are very similar, but the EE has the advantage of

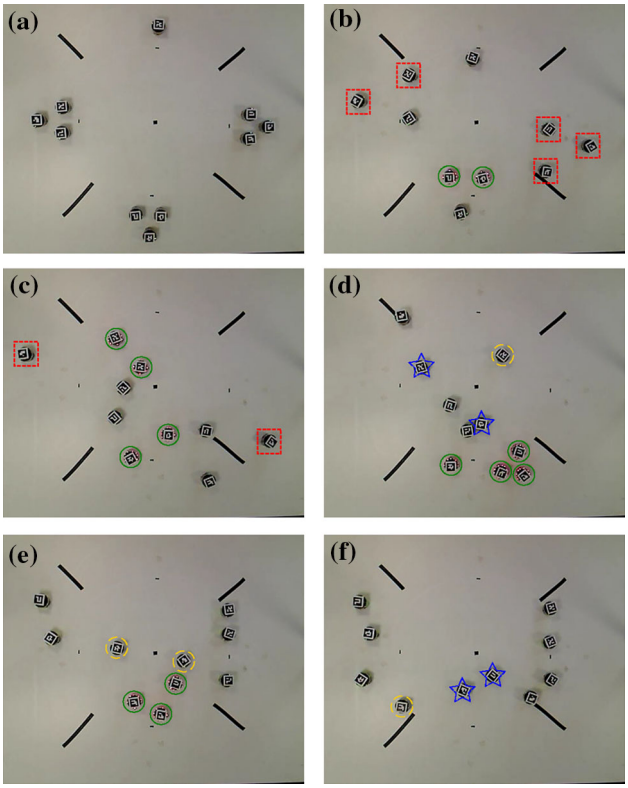


Fig. 24 Experiment with *e-pucks* using the PCC-EE algorithm (video available at https://youtu.be/jyjC3bffkdQ). In the video, robots with all the leds on are in the *waiting* or *locked* state. **a** *0 s* Initial position. **b** *21 s* Robots move towards the *entry* region, while others wait near the target. **c** *36 s* More robots change to *waiting*, while two are still left in the *exit* region. **d** *1 min 23 s* One robot reaches the target, and can easily exit. **e** *4 min 46 s* More robots go towards their next objective. **f** *6 min 17 s* Almost all robots completed execution

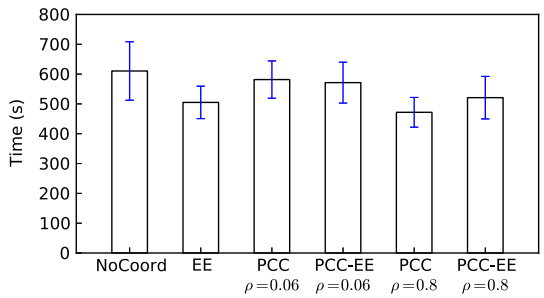


Fig. 25 Results in the *real world* executions

not needing communication between the robots nor needing to find a good parametrization for ρ.

We also studied the total number of messages sent by all robots during the PCC and PCC-EE executions. We can see the results in Fig. 26 (where, again, the bars show the confidence interval with p value 0.01). For the executions with $\rho = 0.06$, both algorithms used a similar amount of messages, and the result is not statistically significant. However, for $\rho = 0.8$, PCC uses about 25 % less messages than

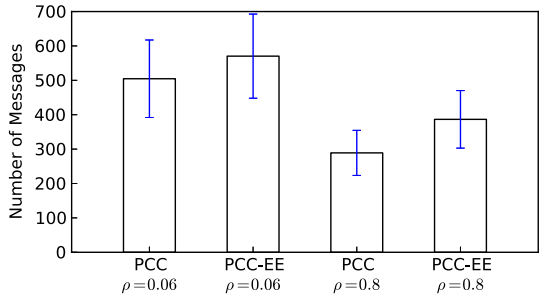


Fig. 26 Total number of messages sent in the *real world* executions

the PCC-EE algorithm, with statistical significance (p value 0.008369). This is a surprising result, since we would normally expect PCC-EE to use less messages (as how we found in the simulations). One explanation could be that, because the number of robots is small, the most important factor in determining the number of messages could have been the execution time, and PCC performed better than PCC-EE with $\rho = 0.8$.

6 Conclusion

In this paper, we presented three algorithms to alleviate congestion of a swarm of robots when they move towards a common target. Our first algorithm, PCC, uses probabilistic finite state machines; our second, EE, divides the area around the target in entry and exit regions; and our third approach, PCC-EE, is a combination of the two previous algorithms. We presented experiments in simulation and with real robots.

Our simulations show that all algorithms present a significant improvement compared to an execution using only local repulsion forces. The EE and the PCC-EE algorithm had the best performance, with the PCC-EE being only slightly better than the EE. The EE algorithm, however, does not require the robots to exchange messages, and is much easier to configure, as it is not necessary to set up the variable that defines how long the robots are expected to wait (ρ). In the other algorithms, we had to determine a good value for this parameter by performing many executions. We could also show that in the EE algorithm the robots can exit the target area in a much more efficient way, and that seems to be the reason for its excellent performance.

We also performed many real world executions, with a team of 10 *e-puck* robots. Based on that we could show that all proposed algorithms are better than using only local repulsion forces with statistical significance in the real world. PCC seems to have the best performance, which could be explained by the number of real robots not being very large in comparison with our experiments in simulation. EE, however, had a very similar performance (the difference between the algorithms was not even statistically significant), without

the need of control messages nor the need to parametrize ρ, as mentioned.

Additionally, we studied the performance of ORCA, a state of the art collision avoidance mechanism. We showed that ORCA is not able to handle the common target problem, as the robots circulate around the target instead of moving towards it. Hence, we show that local repulsion forces is, indeed, a reasonable baseline for comparison with our algorithms.

As the EE algorithm assumes that we can divide the environment in two global regions, *entry* and *exit*, some readers may wonder about the performance of other approaches that use the global coordinated frame, such as forming an attraction vortex around the target, or forcing the robots to move to one side of the target region and form a lane. After performing initial experiments, however, we found that such approaches force the robots to move more in the environment than the algorithms presented, and have a worse performance than using only local repulsion forces.

Additionally, the need of a global coordinated frame could be relaxed in the EE algorithm. Some environmental marks could be used to help the robots determine when they are in the *exit* region, or the robots in the *exit* region could move towards the *entry* region in a more relaxed fashion, instead of moving towards a specific point. In case environmental marks are not feasible, the robots would only need to know their global angle in relation to the target (e.g., using a compass); the full global position is not necessary. If this is still unfeasible for a given application, then a designer should select the PCC algorithm (which in fact had the best performance in the real world executions).

There are many possibilities for future works dealing with the problem of congestion for a swarm of robots. It would be nice to develop a model for the common target problem, in order to analytically find the optimal values for the algorithms' parameters, instead of performing extensive experimentation. For instance, for the PCC and PCC-EE algorithm, determining ρ is important for a good execution, and we had to run many simulations to study the impact of different values of this parameter. It is still a challenge, however, to model cluttered systems. An alternative could be to have a dynamically changing ρ or let the robots learn the best value during the execution.

Another important research direction would be to study what would be the theoretically optimal amount of time that the robots should spend to reach the target, in order to study how far our current algorithms are from this theoretically optimal solution.

As we move towards more and more executions with a great number of robots in the real world, and as society finds more applications for swarm robotics, the impact of congestion problems will increase. Hence, it is necessary to find now efficient ways to alleviate it in order to effectively have a swarm of robots acting in the real world in an useful way.

Acknowledgements This work was partially supported by CAPES, CNPq, and FAPEMIG.

Appendix: PCC-EE with ORCA

In Fig. 27 we show screenshots of the PCC-EE algorithm using ORCA to avoid collisions (instead of using local repulsion forces). Figure 27a shows the initial position of the robots. Robots in the *exit* region move towards the *entry* region, while the robots in the *entry* region follow the PCC algorithm (Fig. 27b). We notice in Fig. 27c that some robots are able to reach the target, but others form an arc in the *entry* region. All robots that were not in the arc are able to reach the target. However, the robots in the arc stay in equilibrium, and are not able to leave anymore (Fig. 27d, e, f).

This situation is similar to the one discussed in the main paper: as all velocity vectors point towards the target, the resulting velocity vector of all robots in the arc points towards the perpendicular of the preferred velocity vector (towards the target). This time, however, the robots in the borderline of the *entry* region are not able to leave the area, as they immediately return to the *entry* region due to the PCC-EE algorithm. Hence, instead of circulating around the target area, the robots stay locked in arcs around the target area.

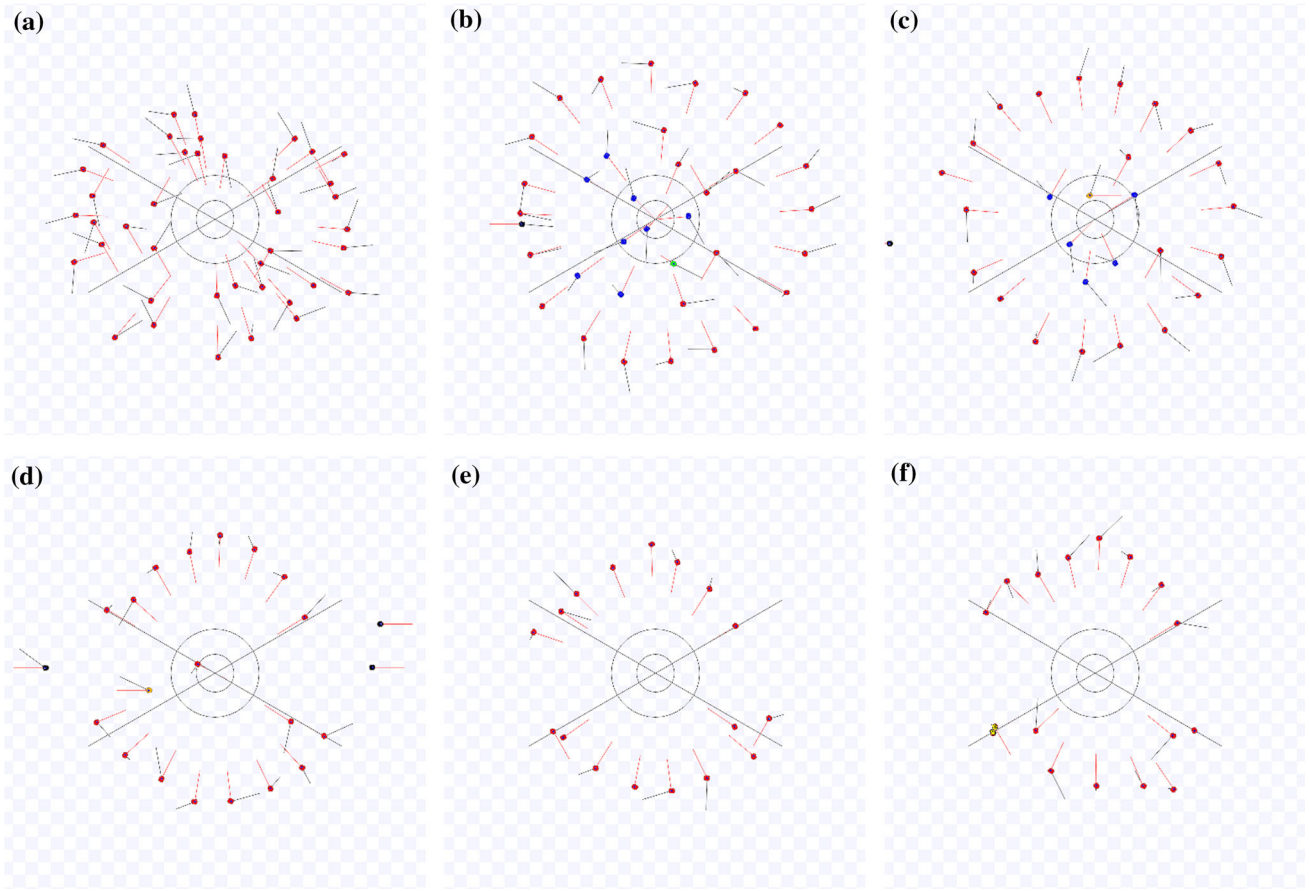


Fig. 27 Execution screenshots of the PCC-EE algorithm, using ORCA to avoid collisions (video available at https://youtu.be/ch0v2jje56E). **a** *0 s* Beginning of the execution. **b** *4 min 30 s* Robots move towards the *entry* region, following the PCC algorithm inside it. **c** *9 min 0 s* Some robots are able to reach the target, but others form an arc in the *entry* region, surrounding the target. **d** *13 min 30 s* The robots that were not in the arcs around the target are able to reach the target. **e** *18 min 0 s* The robots in the arcs still do not converge towards the target. **f** *22 min 31 s* After many iterations, the robots still do not go towards the target, locked in the arcs in the *entry* region

References

Alonso-Mora, J., Naegeli, T., Siegwart, R., & Beardsley, P. (2015). Collision avoidance for aerial vehicles in multi-agent scenarios. *Autonomous Robots*, *39*, 101–121.

Barca, J. C., & Sekercioglu, Y. A. (2013). Swarm robotics reviewed. *Robotica*, *31*, 345–359.

Bayindir, L. (2015). A review of swarm robotics tasks. Neurocomputing (in press).

Bazazi, S., Pfennig, K. S., Handegard, N. O., & Couzin, I. D. (2012). Vortex formation and foraging in polyphenic spadefoot toad tadpoles. *Behavioral Ecology and Sociobiology*, *66*(6), 879–889.

Brambilla, M., Ferrante, E., Birattari, M., & Dorigo, M. (2013). Swarm robotics: A review from the swarm engineering perspective. *Swarm Intelligence*, *7*(1), 1–41.

Cai, C., Yang, C., Zhu, Q., & Liang, Y. (2007). Collision avoidance in multi-robot systems. In *Proceedings of the 2007 IEEE International Conference on Mechatronics and Automation* (pp. 2795–2800). Harbin, China.

Caloud, P., Choi, W., Latombe, J. C., Le Pape, C., & Yim, M. (1990). Indoor automation with many mobile robots. In *Proceedings of the IEEE International Workshop on Intelligent Robots and Systems* (pp. 67–72). IROS.

Cao, Y. U., Fukunaga, A. S., & Kahng, A. B. (1997). Cooperative mobile robotics: Antecedents and directions. *Autonomous Robots*, *4*, 226–234.

Carlino, D., Boyles, S. D.,& Stone, P. (2013). Auction-based autonomous intersection management. In *Proceedings of the 16th International IEEE Conference on Intelligent Transportation Systems* (pp. 529–534.). ITSC.

Cianci, C. M., Raemy, X., Pugh, J., & Martinoli, A. (2007). Communication in a swarm of miniature robots: The e-Puck as an educational tool for swarm robotics. In *Proceedings of Simulation of Adaptive Behavior (SAB-2006), Swarm Robotics Workshop, Lecture Notes in Computer Science (LNCS)*, vol. 4433 (pp. 103–115).

Correll, N., & Martinoli, A. (2006). Towards optimal control of self-organized robotic inspection systems. In *Proceedings of the 8th International IFAC Symposium on Robot Control*.

Couzin, I. D., & Franks, N. R. (2002). Self-organized lane formation and optimized traffic flow in army ants. *Proceedings of the Royal Society of London B*, *270*, 139–146.

Demir, N., Eren, U., & Açikmeşe, B. (2015). Decentralized probabilistic density control of autonomous swarms with safety constraints. Autonomous Robots (in press—Published online).

Dresner, K., & Stone, P. (2005). Multiagent traffic management: an improved intersection control mechanism. In *Proceedings of the*

fourth international joint conference on autonomous agents and multiagent systems (pp. 471–477). AAMAS, New York, NY, USA: ACM.

Ducatelle, F., Di Cari, G., Förster, A., Bonani, M., Dorigo, M., Magnenat, M., et al. (2014). Cooperative navigation in robotic swarms. *Swarm Intelligence*, *8*(1), 1–33.

Ducatelle, F., Di Caro, G. A., Pinciroli, C., & Gambardella, L. M. (2011a). Self-organized cooperation between robotic swarms. *Swarm Intelligence*, *5*(2), 73–96.

Ducatelle, F., Di Caro, G. A., Pinciroli, C., Mondada, F., & Gambardella, L. (2011b). Communication assisted navigation in robotic swarms: Self-organization and cooperation. In *Proceedings of the 24th IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS* (pp. 4981–4988). San Francisco, USA, September 25–30.

Ferrati, M., & Pallottino, L. (2013). A time expanded network based algorithm for safe and efficient distributed multi-agent coordination. In *Proceedings of the IEEE 52nd Annual Conference on Decision and Control, CDC* (pp. 2805–2810).

Franchi, A., Stegagno, P., & Oriolo, G. (2015). Decentralized multi-robot encirclement of a 3D target with guaranteed collision avoidance. Autonomous Robots (in press—Published online).

Garcia, R. F., & Chaimowicz, L. (2009). Uma infra-estrutura para experimentação com enxames de robôs. In *Proceedings of the IX Simpósio Brasileiro de Automação Inteligente, SBAI* (In Portuguese).

Gerkey, B. P., Vaughan, R. T., & Howard, A. (2003). The Player/Stage project: Tools for multi-robot and distributed sensor systems. In *Proceedings of the 11th International Conference on Advanced Robotics, ICAR* (pp. 317–323).

Grossman, D. (1988). Traffic control of multiple robot vehicles. *IEEE Journal of Robotics and Automation*, *4*(5), 491–497.

Guo, Y., & Parker, L. E. (2002). A distributed and optimal motion planning approach for multiple mobile robots. In *Proceedings of IEEE International Conference on Robotics and Automation, ICRA* (pp. 2612–2619).

Hoshino, S. (2011). Multi-robot coordination methodology in congested systems with bottlenecks. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS*. (pp. 2810–2816). IEEE.

Ikemoto, Y., Hasegawa, Y., Fukuda, T., & Matsuda, K. (2004). Zipping, weaving: Control of vehicle group behavior in non-signalized intersection. In *Proceedings of the IEEE International Conference on Robotics and Automation, ICRA* (pp. 4387–4391). New Orleans, USA.

Kato, S., Nishiyama, S., & Takeno, J. (1992). Coordinating mobile robots by applying traffic rules. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS*, 1535–1541.

Krishna, K. M., & Hexmoor, H. (2004). Reactive collision avoidance of multiple moving agents by cooperation and conflict propagation. In *Proceedings of the IEEE International Conference on Robotics and Automation, ICRA* (pp. 2141–2146).

Krontiris, A., & Bekris, K. E. (2011). Using minimal communication to improve decentralized conflict resolution for non-holonomic vehicles. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS* (pp. 3235–3240). IEEE.

Lerman, K., & Galstyan, A. (2002). Mathematical model of foraging in a group of robots: Effect of interference. *Autonomous Robots*, *13*, 127–141.

Luca, A. D., & Oriolo, G. (1994). Local incremental planning for nonholonomic mobile robots. In *Proceedings of the IEEE International Conference on Robotics and Automation, ICRA* (pp. 104–110).

Marcolino, L. S., & Chaimowicz, L. (2008) . No robot left behind: Coordination to overcome local minima in swarm navigation. In *Proceedings of the 2008 IEEE International Conference on Robotics and Automation, ICRA* (pp. 1904–1909).

Marcolino, L. S., & Chaimowicz, L. (2009). Traffic control for a swarm of robots: Avoiding target congestion. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS* (pp. 1955–1961).

Martinoli, A., Easton, K., & Agassounon, W. (2004). Modeling swarm robotic systems: A case study in collaborative distributed manipulation. *The International Journal of Robotics Research*, *23*(4–5), 415–436.

Olmi, R., Secchi, C., & Fantuzzi, C. (2009). A coordination technique for automatic guided vehicles in an industrial environment. In *Proceedings of the 9th IFAC International Symposium on Robot Control, SYROCO* (pp. 359–364).

Pallottino, L., Scordio, V. G., Bicchi, A., & Frazzoli, E. (2007). Decentralized cooperative policy for conflict resolution in multivehicle systems. *IEEE Transactions on Robotics*, *23*(6), 1170–1183.

Peasgood, M., Clark, C., & McPhee, J. (2008). A complete and scalable strategy for coordinating multiple robots within roadmaps. *IEEE Transactions on Robotics*, *24*(2), 283–292.

Sahin, E. (2004). Swarm robotics: From sources of inspiration to domains of application. In *SAB 2014 International Workshop on Swarm Robotics—Revised Selected Papers, Lecture Notes in Computer Science*, vol. 3342 (pp. 10–20). Springer.

Sahin, E., Girgin, S., Bayindir, L., & Turgut, A. E. (2008). Swarm robotics. In *Swarm Intelligence, Natural Computing Series* (pp. 87–100). Springer.

Santos, V. G., Campos, M. F. M., Chaimowicz, L. (2014). On segregative behaviors using flocking and velocity obstacles. In M. Ani Hsieh & G. Chirikjian (Eds.), Distributed Autonomous Robotic Systems: The 11th International Symposium (pp. 121–133). Berlin, Heidelberg: Springer.

Santos, V. G., & Chaimowicz, L. (2011). Hierarchical congestion control for robotic swarms. In *Proceedings of the IEEE/RJS International Conference on Intelligent Robots and Systems, IROS* (pp. 4372–4377).

Savchenko, M., & Frazzoli, E. (2005). On the time complexity of conflict-free vehicle routing. In *Proceedings of the American Control Conference*, *5*, 3536–3541.

Shapiro, J. A. (1988). *Bacteria as multicellular organisms. Scientific American* (pp.82–89). Nature America.

Siegwart, R., & Nourbakhsh, I. R. (2004). *Introduction to Autonomous Mobile Robots*. Scituate, MA: Bradford Company.

Treuille, A., Cooper, S., & Popović, Z. (2006). Continuum crowds. In *Proceedings of the 33rd International Conference and Exhibition on Computer Graphics and Interactive Techniques* (pp. 1160–1168). SIGGRAPH. ACM: New York, NY, USA.

van den Berg, J., Guy, S. J., Lin, M., & Manocha, D. (2011). Reciprocal n-body collision avoidance. In C. Pradalier., R. Siegwart & G. Hirzinger (Eds.), *Robotics Research: The 14th International Symposium ISRR, Springer Tracts in Advanced Robotics*, vol. 70 (pp. 3–19). Springer-Verlag.

van den Berg, J., Guy, S. J., Snape, J., Lin, M. C., & Manocha, D. (2015). RVO2 library: Reciprocal collision avoidance for real-time multi-agent simulation. http://gamma.cs.unc.edu/RVO2/publications/.

van den Berg, J., Lin, M. C.,& Manocha, D. (2008). Reciprocal velocity obstacles for real-time multi-agent navigation. In *Proceedings of the 2008 IEEE International Conference on Robotics and Automation*, ICRA (pp. 1928–1935).

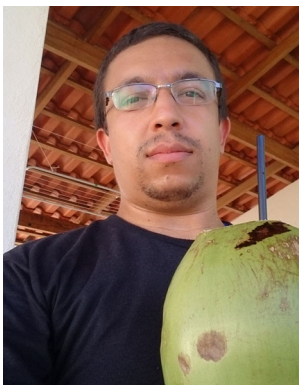
Vidal, E., Thollard, F., de la Higuera, C., Casacuberta, F., & Carrasco, R. C. (2005). Probabilistic finite-state machines—Part I. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *27*(7), 1013–1025.

Yasuaki, A., & Yoshiki, M. (2001). Collision avoidance method for multiple autonomous mobile agents by implicit cooperation. In

Proceedings of the IEEE International Conference on Intelligent Robots and Systems, IROS. Maui, USA, (pp. 1207–1212).

Leandro Soriano Marcolino is a Ph.D. student at University of Southern California (USC). He has published in several prestigious conferences in AI, robotics and machine learning, such as AAAI, AAMAS, IJCAI, NIPS, ICRA and IROS. He received the best research assistant award from the Computer Science Department at USC, had a paper nominated for best paper from the leading multi-agent conference AAMAS, and had his undergraduate work selected as the best one by the Brazilian Computer Science Society. He has been researching continuously about teamwork and cooperation, and obtained his masters degree in Japan, with the highly-competitive Monbukagakusho scholarship. Over his career, Leandro has published about a variety of domains, such as swarm robotics, computer Go, social networks, bioinformatics and architectural design.

Yuri Tavares dos Passos received his bachelor's degree in Computer Science at Federal University of Sergipe (2008) and master's degree in Computer Science at Federal University of Minas Gerais (2012). He is currently assistant professor in computer engineering courses and he teaches Formal Languages, Theory of Computation and Compilers. His research interests include robotics, artificial intelligence and computer vision.



Álvaro Antônio Fonseca de Souza is an undergraduate Computer Science student at Universidade Federal de Minas Gerais (UFMG) - Brazil. He is currently researching about robotic swarms at UFMG's Computer Vision and Robotics Laboratory (VeRLab). His experience includes the development and execution of algorithms for the e-puck robots, under different experimental frameworks (encompassing not only remote control algorithms, but also embedding algorithms/drivers in the robots themselves). His research interests includes robotics, artificial intelligence and computer networks.

Andersoney dos Santos Rodrigues is undergraduate student in Bachelor of Exact and Technological Sciences at the Universidade Federal do Recôncavo Baiano since 2014. He is currently in a CNPQ undergraduate research internship with Yuri Tavares dos Passos, working in a project concerning the comparison of swarm robotics algorithms. His research interests are machine learning, swarm robots, nanotechnology and data encryption models.

Luiz Chaimowicz is an Associate Professor at the Computer Science Department of the Universidade Federal de Minas Gerais (UFMG) - Brazil. He received his Ph.D. degree in Computer Science from this same university in 2002, and from 2003 to 2004, he held a Postdoctoral Research appointment with the GRASP Laboratory at University of Pennsylvania. He co-directs UFMG's Computer Vision and Robotics Laboratory (VeRLab), which conducts research on several aspects of computer vision and mobile robotics. His recent research focuses on the coordination and control of large groups of robots and the development of collision free navigation strategies for these groups.