Traffic Control for a Swarm of Robots: Avoiding Target Congestion

Leandro Soriano Marcolino and Luiz Chaimowicz

***Abstract*— One of the main problems in the navigation of robotic swarms is when several robots try to reach the same target at the same time, causing congestion situations that may compromise performance. In this paper, we propose a distributed coordination algorithm to alleviate this type of congestion. Using local sensing and communication, and controlling their actions using a probabilistic finite state machine, robots are able to coordinate themselves to avoid these situations. Simulations and real experiments were executed to study the performance and effectiveness of the proposed algorithm. Results show that the algorithm allows the swarm to have a more efficient and smoother navigation and is suitable for large groups of robots.**

I. Introduction

The use of large groups of simple and inexpensive robots to perform complex tasks has become an important research topic in robotics. Generally called *swarms*, these groups bring several advantages over single robot solutions. The division of work among the team generally improves the efficiency of the system. Moreover, robustness is also increased, since with a large number of robots it is easier to have redundancy and therefore to design fault-tolerant systems. However, there are many challenges when working with swarms of robots. Generally, they must work in a distributed fashion and use limited communication resources. Hence, new algorithms must be developed to coordinate these large groups of robots.

A key requirement for swarms is to be able to efficiently navigate in different scenarios. One of the main challenges in swarm navigation is congestion: a large number of robots moves towards the same region of the environment in the same time interval, causing conflicts that waste time and resources. This problem may appear when groups of robots move in opposite directions and encounter while navigating or when a specific region is a target for many robots. This second case, in particular, appears very often. For example, during *waypoint navigation* there might be a critical waypoint that will be used by many robots. Particularly, in the methodology proposed in [1], we observed several congestion situations as robots had to navigate through the same waypoints to overcome local minima. Other potential conflicting targets in swarm navigation may include a recharge station to which several robots need to move at the same time or some narrow passage that only allows few robots at a time.

This work is partially supported by Fapemig and CNPq. The authors would like to thank Renato Garcia for his help with the localization system and the development of the epuck driver.

The authors are with the Vision and Robotics Laboratory (VeRLab), Computer Science Department, Federal University of Minas Gerais, Brazil. emails: {soriano,chaimo}@dcc.ufmg.br

However, this problem is not easy to solve. In general scenarios, robots may come from any direction, making solutions based in delimited lanes (such as roads and crossings) not applicable. Besides, as fault-tolerance is desirable, it is not a good idea to have a centralized server or design leaders to coordinate the movement towards the goal. But unfortunately, without a centralized server, it is harder to find an adequate time schedule for all robots. Common solutions for shared resources, such as a token ring, may be not feasible since robots are constantly moving and have a limited communication range. Regular collision avoidance algorithms do not solve the problem either, because avoiding collisions does not mean avoiding congestions. So, even using collision avoidance algorithms, we can still have performance problems when a large number of robots must navigate to a common target. As can be seen, it is necessary to design a robust and decentralized solution that does not depend on a structured environment. Moreover, the solution must be efficient and guarantee that eventually all robots will reach the target, i.e., there will be no deadlock situations.

Hence, the objective of this paper is to investigate and develop methodologies to control the traffic of large groups of robots when they are moving to a common target in unstructured environments. We propose a distributed coordination algorithm that makes some robots wait while others move towards the common target, alleviating congestion and improving performance. Robots control their actions using a probabilistic finite state machine and rely on local sensing and communication to coordinate themselves. We analyze the performance of the algorithm and show its effectiveness by executing a series of experiments using both a simulator and a group of real robots and also develop proofs that the algorithm is able to solve the proposed problem. In a companion paper [2], we also developed a solution for the congestion situation where groups of robots move into opposite directions.

II. Related Work

The traffic control problem is an important research topic. In [3], it is characterized as a *resource conflict* problem and the importance of its study is emphasized. Works dealing with traffic control started to appear in the late 1980s. In [4], for example, many policies are presented to avoid congestion of robots in a factory, and in [5] traffic rules are shown to navigate a group of robots. 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, in general, is executed only in these intersections.

More recent works can be found both in the cooperative robotics field and in the multi-agent systems field. Some works use a manager agent to administrate the traffic at intersections where congestion may happen, as in [6]. A similar approach, in the robotics field, can be seen in [7], where a sensor network is used to coordinate the traffic of a group of robots. Others are working in manager free scenarios, as in [8], which presents a completely distributed algorithm that, based on a spatial temporal pattern, coordinates the movement of robots into intersections or junctions. 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.

In [9], a mechanism is proposed to avoid congestion in crowd simulations. The authors propose an approach in which agents plan early to avoid a congestion, enabling smoother trajectories than when using local repulsion forces. The method, however, is too centralized to be used with 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.

Instead of dealing with traffic control, there are works that try to find more efficient approaches to collision avoidance than using local repulsion forces. In [10], an algorithm is proposed in which robots coordinate their velocities in order to avoid a collision. 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. Other works that deal with collision avoidance are [11], [12]. However, as mentioned, collision avoidance algorithms may not be sufficient for preventing congestion situations mainly when a large number of robots converge to the same place. Hence, even with a good collision avoidance behavior, the system may still become cluttered and inefficient.

Therefore, although there are many works dealing with traffic control and collision avoidance, to the best of our knowledge there is no algorithm that deals directly with the proposed problem, in which many robots converges to a common target in a unstructured environment and must coordinate themselves in a distributed, robust and fault-tolerant fashion. This problem often happens in swarm navigation, for example, when they are using waypoints. We propose a decentralized coordination mechanism based on a probabilistic finite state machine that allows a swarm of robots to prevent congestion in waypoint navigation, without assuming the use of delimited lanes nor needing an external infra-structure to control the traffic. This is the main contribution of this paper.

III. Methodology

The general idea of the coordination algorithm is to force some robots to wait while others converge to the target. Therefore, a small number of robots try to reach the same target at the same time, decreasing the congestion problem.

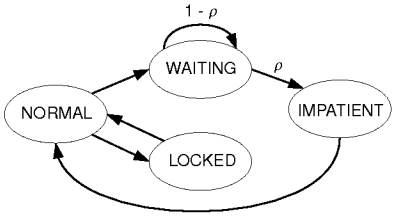

Fig. 1. Probabilistic finite state machine showing the possible states and transitions for each swarm member.

Note that we will not prevent that two or more robots head towards the target at the same time interval, we only want to reduce the number of robots that try to do it simultaneously. With few robots at the target region, common collision avoidance techniques are able to work accordingly.

We modeled our solution as a *Probabilistic Finite State Machine* [13], in which edges are annotated with probabilities that define which transition will be taken. Figure 1 shows the probabilistic finite state machine used in this paper. As can be seen, robots in the swarm can be in one of 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 on the same state with probability $1 - \rho$.

We will begin by explaining general ideas about the algorithm and making some necessary definitions. We define a *free* region as a circular region with radius σ around the target. Around this region, we define a *danger* region as a ring-shaped region with inner radius σ and outer radius γ. The general idea is that the robots that reach the *danger* region will coordinate so that only few of them will enter the *free* region at the same time. Upon entering the *free* region, robots will move straight to the target.

We also 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 Figure 2). As will be explained later in this section, this α-area will be used to detect other robots that may interfere with the navigation to the target.

We consider that a robot is able to detect the presence of another and avoid collisions when the distance between them is lower than δ. Every time that 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 informing its target 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.

The coordination algorithm works as follows: a *normal* robot, j, moves in the direction of the target while avoiding collisions. When it 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

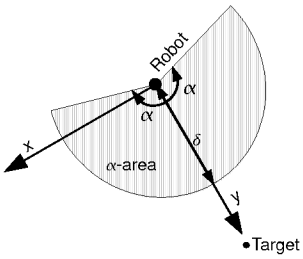

Fig. 2. Sensing area (α-area) considered by a robot to change its state.

are true, j will change its state to *waiting*. This situation can be seen in Figure 3(a).

A *waiting* robot will not move in the direction of the target. It will try to remain stationary in the point where it changed its state while at the same time avoiding collisions. Avoiding collisions 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. At every η interactions, 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 in *waiting* with probability $1 - \rho$.

A *normal* robot may also change its state to *locked*. This happens when the robot detects a *waiting* or a *locked* robot with the same target as its own. However, this time the robot can change its state even outside the *danger* region. In this case, the α-area is narrower, i.e., a smaller α is used to check whether a certain neighbor should be considered to change the robot's state. We will call this value α_l. This situation can be seen in Figure 3(b) and (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. 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. However, the transition from this 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.

Finally, 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 to *waiting* nor *locked*. Only after the robot reaches the target, it will change its state back to *normal*. This situation can be seen in Figure 3(d), where robot j resumed 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 changed their state to *waiting* upon reaching the *danger* region and, therefore, will not impose difficulties for robot j to reach the target and leave its region, enabling a smoother navigation.

We can use an analogy with a car traffic jam to better understand the algorithm. Consider a five lane highway, in which a mud slide closed four lanes. The first cars approaching the region will stop (enter the *waiting* state). Eventually, one of them will get *impatient* and pass first.

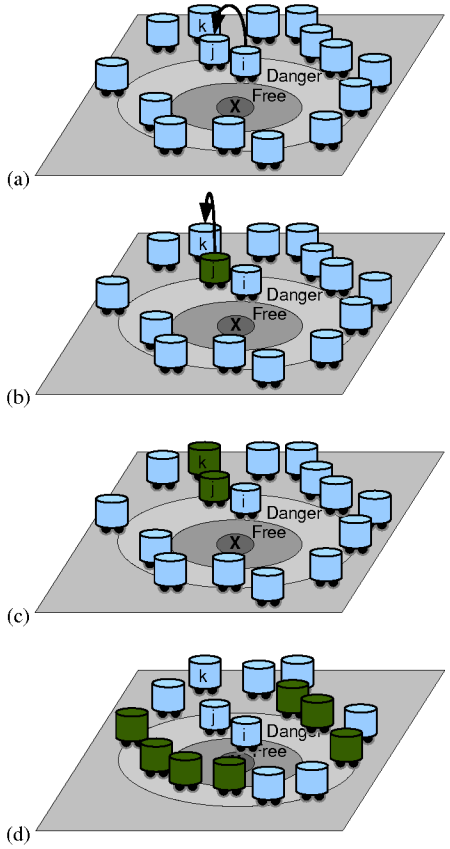

Fig. 3. Steps of the execution of the proposed coordination algorithm. Green (dark) robots are in the *waiting* or *locked* states. The arrows indicate message transmission.

We can consider this decision as a random process, in which each driver has a probability ρ of getting *impatient*. The other cars coming on the highway will see the first cars stopped in front of this accident and will also stop (enter the *locked* state) forming several lines. As the cars on the front row pass through the single lane, the *locked* cars move forward and approach the accident, waiting their turn to go through the single lane. This type of approach works without the need of a traffic guard or any kind of centralized mechanism.

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. The robots could estimate it using an on-board sensor such as a stereo camera or a laser. Therefore, the coordination algorithm does not impose additional requirements to the system besides the ability to locally sense and communicate with neighbors.

IV. 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 changes 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 $alpha_l < 90^o$). 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 graph dependency, 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 congestions 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 give up 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. So it is necessary to find a good compromise point.

V. EXPERIMENTS

We ran a series of simulations and real experiments to study the performance and feasibility of the proposed algorithm. For the simulations, we used the Player/Stage framework [14], a well known framework for robotics programming and simulation. The real experiments were performed using a dozen *e-puck robots*. The e-puck is a small-sized (7cm diameter) differential drive robot that is very suitable for swarm experimentation [15]. Each robot is equipped with a ring of 8 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 robot to robot communication and remote control. Figure 4 shows the robots used in the experiment.

Fig. 4. Dozen *e-puck* robots used in the experiments.

In our experiments, robots were controlled using a common potential field algorithm: an attractive force moved them towards the goal while local repulsion forces were used to avoid collisions among the group. We decided to use that collision avoidance technique because it is very common in works dealing with a large number of robots, for example [16], [17], [18]. However, the coordination method could be tested with other controllers, as it does not directly depend

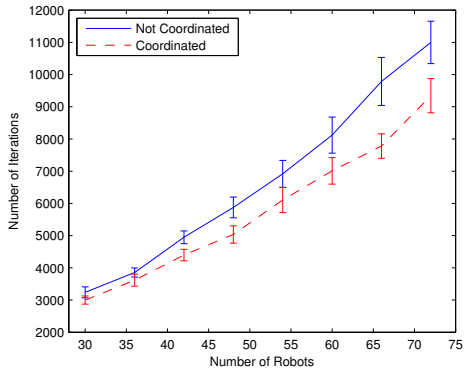

Fig. 5. Time used by both algorithms. The bars show the confidence interval of the results, with 95% level of confidence.

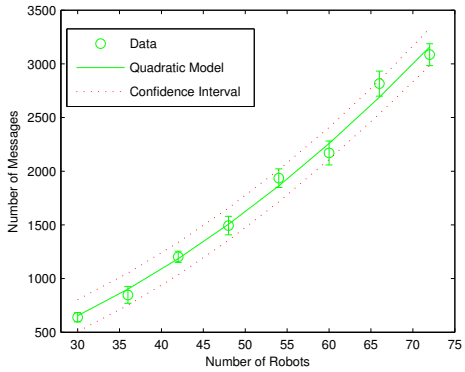

Fig. 6. Number of messages sent for a varying number of robots. The confidence interval corresponds to a level of confidence of 95%.

on potential fields to work. Both in simulations and in real experiments we used non-holonomic robots, with a control equation similar to the one presented in [19].

A. Simulations

In order to evaluate the proposed coordination algorithm we ran a series of simulations using the algorithm (Coord) and not using it (NotCoord). We consider a scenario where robots should move to a common target and leave in another direction after that. In every execution, the robots were randomly positioned in the scenario outside the *danger* and the *free* region. We varied the number of robots and measured the execution time and the number of messages sent. As a measure of time, we used the number of iterations necessary for the last robot to reach the target. Each simulation was run 20 times and the mean results were considered. We used the following values for the main constants: $\delta = 2m$, $\epsilon = 25$, $\gamma = 3.5m$, $\sigma = 1.5m$, $\alpha_w = 95^o$, $\alpha_l = 45^o$, $\eta = 40$, $\rho = 0.15$.

Figure 5 shows the execution time for a varying number of robots. As can be observed, the proposed algorithm has a better performance when the number of robots increases. In fact, we executed a *t-test* that showed that the Coord algorithm was better in all analyzed points with more than 42 robots with 95% level of confidence. The performance improvement reached 20% with the use of the proposed algorithm. We also computed the standard deviation of the results, which showed that with more than 42 robots the Coord algorithm has a smaller deviation from the mean.

In Figure 6 we can see the number of messages used by the proposed algorithm for a varying number of robots. The best model found for the curve was the quadratic $y = 0.5107x^2 + 7.4987x - 30.2645$, with a coefficient of determination (R^2) of 0.9964. Although the model is a quadratic function, we can see that the quadratic term is small. This result shows that the algorithm scales well and is suitable for large groups of robots.

A visual log of one simulation with 48 robots is presented in Figure 7. Robots are represented by different shapes according to their states: *normal* ($+$), *waiting* ($\circ$), *locked* ($\triangle$) and *impatient* ($\times$). Robots in the *normal* state that have already reached the target and are moving to another one are represented by the symbol (*). The outer circle represents the *danger* region while the inner one represents the *free* region. As can be observed, the *waiting* robots form a barrier in the *danger* region, while the *locked* robots tend to wait outside that region. That enabled all robots to reach the target in a smoother fashion, as the number of disputes is a lot smaller in comparisson to the no coordinated version.

B. Real Robots

As mentioned, we also tested the proposed algorithm using a dozen *e-puck* robots. These experiments are important to show the feasibility of the algorithm in real scenarios, with all the uncertainties caused by sensing and actuation errors, communication failures, etc.

To simplify the implementation, we used a localization system specifically designed for swarm localization in indoor environments [20], although, as mentioned, the algorithm does not depend on global localization. Also, as the IR sensors of the e-pucks have a very small range, we implemented a virtual sensor based on the localization system to detect neighbors.

We ran many scenarios, varying the initial position of the robots and the value of parameter ρ. The sequence of snapshots of one execution can be seen in Figure 8 (a short video is accompanying the paper). E-pucks with all LEDs on are in the *waiting* or *locked* state, while e-pucks with all LEDs off are in the *normal* or *impatient* state. The graphs on the bottom depict the robots' positions and states, as well as the *danger* and the *free* regions, as in Section V-A. We used the following values for the main constants: $\delta = 0.18m$, $\epsilon = 2$, $\gamma = 0.3m$, $\sigma = 0.1m$, $\alpha_w = 115^o$, $\alpha_l = 45^o$, $\eta = 60$, $\rho = 0.045$.

Twelve e-pucks are distributed around the target region (indicated by a small mark in the snapshots) in groups of three. After reaching the common target, each robot must move to its own individual target in the upper or bottom side of the scenario. In Figure 8(a) we can see the initial position of the robots (numbered from 1 to 12). Upon entering the *danger* region, robots change their state to *waiting* ($\circ$) as soon as they detect another robot with the same target in

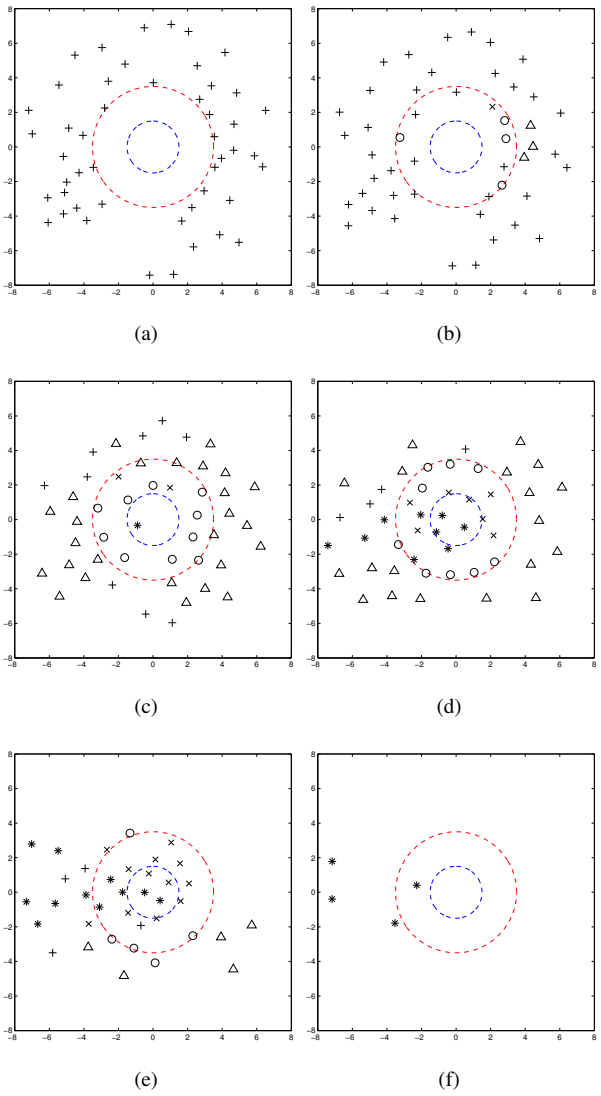

Fig. 7. Simulation results using the coordination algorithm.

their α-area (Figure 8(b)). Robot 10 changes its state to *impatient* ($\times$), and starts moving towards the target (Figure 8(c)). Robots 3 and 6, which are outside the *danger* region, upon detecting *waiting* robots in their α-area, change their state to *locked* ($\triangle$) (Figure 8(d)). Their state change back to *normal* ($+$) only when they detect no other *waiting* robots in their α-area (Figure 8(e)). As time passes, robots change their state to *impatient* and approach the target (Figure 8(f)). Soon, many of them succeed at reaching the common target (*) and are heading towards their second objective (Figure 8(g)), leading to the final state where all robots completed the specified task (Figure 8(h)).

As can be seen, using the proposed algorithm the robots were able to complete the task in a smooth and efficient manner. The total time of this execution was 7 minutes. We also ran the same scenario using only local repulsion forces, which needed 9 minutes for a complete task execution. Thus, the convergence time gain was 22%, a better result from what we found in the simulations, as it was achieved with a smaller number of robots. With more robots, the convergence time gain might be even better. Therefore, these proof of concept experiments indicate that the algorithm can work well to coordinate a swarm of robots, allowing them to smoothly reach a common target.

VI. Conclusions

In this work, we proposed an algorithm to control the traffic of a swarm of robots, avoiding congestion situations. We focused on the case where many robots try to reach the same target, a situation that often appears in robotics.

To study the algorithm, we mathematically proved its efficacy and executed simulations and real experiments. We ran executions with and without the proposed algorithm in order to evaluate the impact of its presence. The results showed that, besides allowing a smoother navigation, the proposed algorithm has a better performance when the number of robots increase. We noticed a quadratic tendency in the number of messages used by the algorithm, but the quadratic term was small. We believe, therefore, that this algorithm is scalable to a large number of robots. Real experiments were successfully executed with a dozen e-puck robots, showing the effectiveness and applicability of the proposed approach.

We intend to investigate the common target problem even further, and improve the algorithm to obtain lower convergence times. Specifically, we noticed that sometimes robots that already reached the common target have difficulty leaving the *danger* region because of conflicts with other robots. A better coordination in this situation might lead to even higher improvements in the common target case.

References

[1] L. S. Marcolino and L. Chaimowicz, "No robot left behind: Coordination to overcome local minima in swarm navigation," in *Proceedings of the 2008 IEEE International Conference on Robotics and Automation*, 2008, pp. 1904–1909.

[2] ——, "Traffic control for a swarm of robots: Avoiding group conflicts," in *Proceedings of the 2009 IEEE International Conference on Intelligent Robots and Systems*, 2009.

[3] U. Y. Cao, A. S. Fukunaga, and A. B. Kahng, "Cooperative mobile robotics: Antecedents and directions," *Autonomous Robots*, vol. 4, no. 1, pp. 7–23, March 1997.

[4] D. Grossman, "Traffic control of multiple robot vehicles," *Journal of Robotics and Automation*, vol. 4, pp. 491–497, 1988.

[5] S. Kato, S. Nishiyama, and J. Takeno, "Coordinating mobile robots by applying traffic rules," in *Proceedings of the IEEE International Conference on Intelligent Robots and Systems*, 1992.

[6] K. Dresner and P. Stone, "Multiagent traffic management: an improved intersection control mechanism," in *AAMAS '05: Proceedings of the fourth international joint conference on Autonomous agents and multiagent systems*. New York, NY, USA: ACM, 2005, pp. 471–477.

[7] K. Viswanath and K. M. Krishna, "Sensor network mediated multi robotic traffic control in indoor environments," in *Proceedings of the International Conference on Advanced Robotics*, 1997.

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

[9] A. Treuille, S. Cooper, and Z. Popović, "Continuum crowds," in *SIGGRAPH '06: ACM SIGGRAPH 2006 Papers*. New York, NY, USA: ACM, 2006, pp. 1160–1168.

[10] K. M. Krishna and H. Hexmoor, "Reactive collision avoidance of multiple moving agents by cooperation and conflict propagation," in *Proceedings of the 2004 IEEE International Conference on Robotics and Automation*, 2004, pp. 2141–2146.

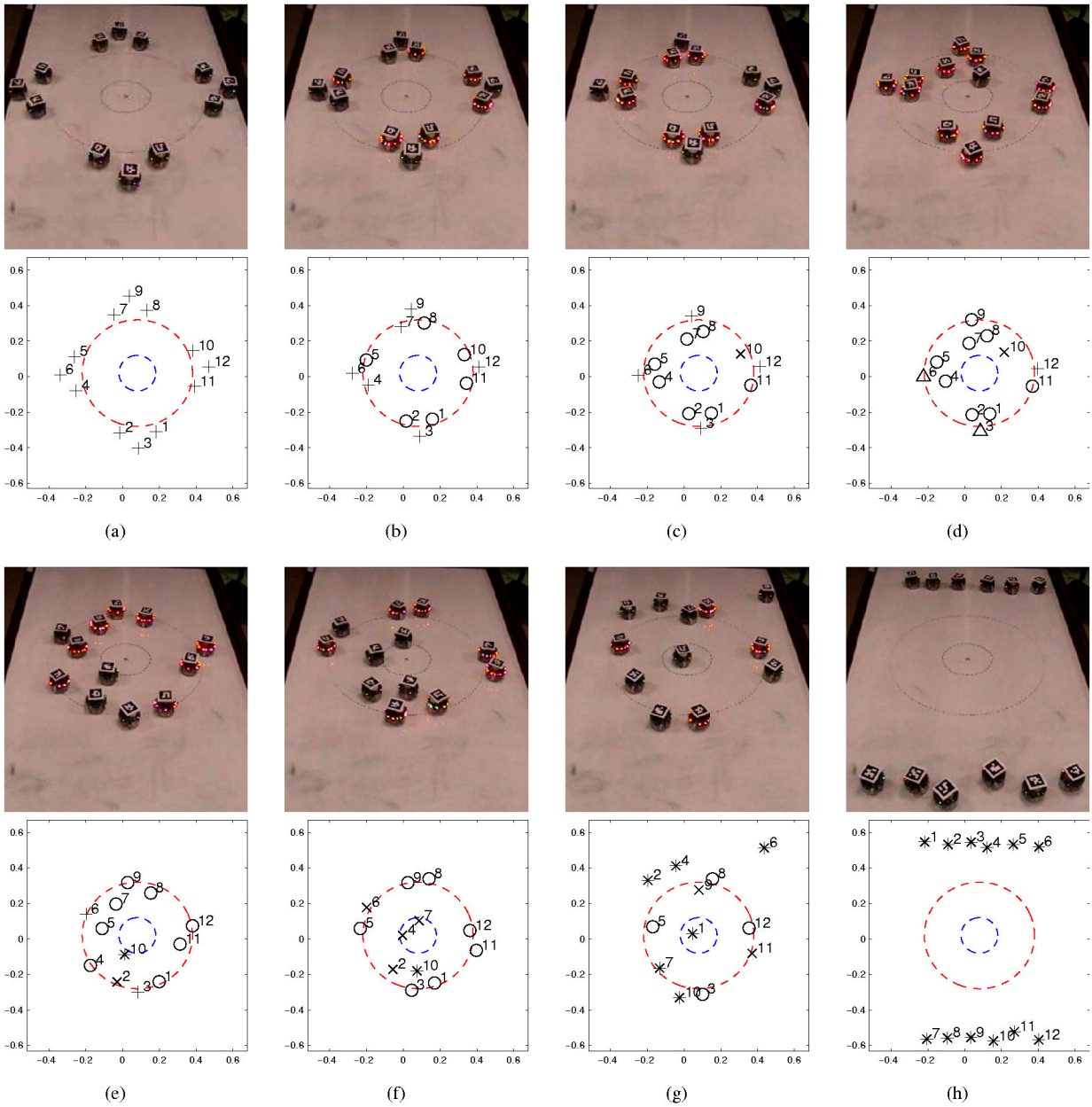

Fig. 8. Real execution using the coordination algorithm.

[11] A. Yasuaki and M. Yoshiki, "Collision avoidance method for multiple autonomous mobile agents by implicit cooperation," in *Proceedings of the 2001 IEEE International Conference on Intelligent Robots and Systems, IROS 2001*, Maui, USA, 2001, pp. 1207–1212.

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

[13] E. Vidal, F. Thollard, C. de la Higuera, F. Casacuberta, and R. C. Carrasco, "Probabilistic finite-state machines-part i," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 27, no. 7, pp. 1013–1025, 2005.

[14] B. Gerkey, R. T. Vaughan, and A. Howard., "The player/stage project: Tools for multi-robot and distributed sensor systems." in *Proceedings of the 11th International Conference on Advanced Robotics*, Coimbra, Portugal, June 2003, pp. 317–323.

[15] C. M. Cianci, X. Raemy, J. Pugh, and A. Martinoli, "Communication in a Swarm of Miniature Robots: The e-Puck as an Educational Tool for Swarm Robotics," in *Simulation of Adaptive Behavior (SAB-2006), Swarm Robotics Workshop*, ser. Lecture Notes in Computer Science (LNCS), 2007, pp. 103–115.

[16] M. A. Hsieh, L. Chaimowicz, and V. Kumar, "Decentralized controllers for shape generation with robotic swarms," *Robotica*, vol. 26, pp. 691–701, September 2008.

[17] A. E. Turgut, H. elikkanat, F. Gke, and E. ahin, "Self-organized flocking in mobile robot swarms," *Swarm Intelligence (Special Issue: Swarm Robotics)*, vol. 2, no. 2-4, pp. 97–120, 2008.

[18] I. Navarro, J. Pugh, A. Martinoli, and F. Matia, "A distributed scalable approach to formation control in multi-robot systems," in *Proceedings of the International Symposium on Distributed Autonomous Robotic Systems*, Tsukuba, Japan, 2008.

[19] A. De Luca and G. Oriolo, "Local incremental planning for nonholonomic mobile robots," in *Proceedings of the 1994 IEEE International Conference on Robotics and Automation*, 1994.

[20] R. Garcia, P. Shiroma, L. Chaimowicz, and M. Campos, "A framework for swarm localization," in *Proceedings of VIII SBAI - Brazilian Symposium on Intelligent Automation*, October 2007, in Portuguese.