Distributed Systems

Roberto Reda, Giuseppe Giorgino
{roberto.reda,giuseppe.giorgino}@studio.unibo.it

Ingegneria e Scienze Informatiche - Cesena - A.A. 2015-2016

Abstract. This is the final report about the project for the exam of Distributed Systems held by professor Andrea Omicini at University of Bologna. The aim of this work is to study the concepts and the issues of distributed systems applied to the field of robotics. Using ARGoS simulator, we implemented some swarm robotics algorithms from literature about pattern formation and spatially communication. Furthermore we tested the solutions proposed under faulty conditions.

Keywords: Distributed robotic systems, Multi-agent systems, Swarm robotics, Pattern formation, ARGoS simulator

Table of Contents

Introduction

The aim of this project is to study the concepts and the issues of distributed systems applied to the field of robotics. We chose the topic of Swarm robotics because it is inherently a decentralised system, that is the emerging behaviour of the total system is given by the contributes of a multitude of agent which work asynchronously using only local information and some kind of communication among each other. In particular, we focused our attention to the topic of pattern formation and spatially target communication in swarm robotics. We selected from the literature some algorithms and studies about it and we implemented and tested them using Lua language and ARGoS simulator.

In the first section, we provide an overview of the topic of distributed robotics systems and swarm robotics. In the second section, we introduce the simulation environment used to carry out our experiments. In the third section we provide, for each algorithm, a general introduction to the particular task and a detailed report about how we implemented the solution to solve it. In the final section, after a general discussion about the fault tolerance in swarm robotics, we consider these issues relative to the algorithm we implemented.

1 Distributed robotics system

Distributed robotics is an interdisciplinary and rapidly growing area, combining research in computer science, communication, control systems and electrical and mechanical engineering. Distributed robotic systems can autonomously solve complex problems while operating in highly unstructured real-world environments. The environments in which robots operate include factories, offices, and homes.

Distributed autonomous systems and *multi-agent systems* are characterised by incomplete information or capabilities for solving the problem in each agent, no system global control, decentralised data, and asynchronous computation. Because various tasks are required for robots, cooperation becomes important and group performance cannot be programmed directly. Designs are limited to the individual performance of each agent. The performance of multi-agents emerges from that of individual agents. Because multi-agent robot systems are part of multi-agent systems (distributed autonomous systems), the above-mentioned characteristics can also apply to multi-agent robot systems. However, multi-agent robot systems have the following special characteristics: since a robot has its own physical parts, including sensors, processors, actuators, and communication devices, its limitations must be considered, such as those related to calculation power, sensing errors, and errors in actuating and communicating devices; and robots move in actual environments; therefore, geometry must be considered. In other words, physical interaction, such as collision avoidance and

the manipulation of physical objects by the cooperation of robots, must be taken into account.

Multi-agent robot systems deal with many kinds of pattern. Tasks by multiple robots can be classified based on the dimension of the goal state and the number of the iteration performed as prosed in [6]. The dimension of the goal state can be a point-reaching (zero-dimension), region-sweeping (more than one-dimension) or compound. Point-reaching tasks are those in which the target is expressed with a specific goal configuration of the robot. The goal configuration is expressed as a specific point in the configuration space of the robots. Region-sweeping tasks are those in which the target is expressed with a specific goal region. Compound tasks are those in which the two types of tasks aforementioned are combined. The number of the iteration of tasks can be one-time or many-times. Ordinary tasks should be performed only once. Some tasks, however, must be performed many-times. Here, 'many-times' means that we expect the effect of adaptation, learning, and self-organisation to occur as a result of several trials. Tasks can be classified into six classes by a combination of the two categories given above.

	One-time	Many-times
Point-reaching	Motion-planning Cooperative handling of a large object	Coming and going between two positions
Region-sweeping	Pattern Formation Sweeping Map generation	Periodical cooperative sweeping
Compound	Cooperative transportation in unknown environments	Collecting objects/ foraging Robot soccer

Table 1. Task classification of multi-agent robots [6]

1.1 Swarm robotics

Swarm robotics is a new approach to the coordination of multi-robot systems which consist of large numbers of mostly simple physical robots. It is supposed that a desired collective behaviour emerges from the interactions between the robots and interactions of robots with the environment. This approach in the field of artificial swarm intelligence, as well as the biological studies of insects, ants and other fields in nature, where swarm behaviour occurs. Unlike distributed robotic systems in general, swarm robotics emphasises a large number of robots, and promotes scalability, for instance by using only local communication. That local communication for example can be achieved by wireless transmission systems,

like radio frequency or infra-red [3]. Potential applications for swarm robotics is indeed huge. It includes tasks that demand for miniaturisation (nanorobotics, microrobotics), like distributed sensing tasks in micro machinery or the human body. One of the most promising uses of swarm robotics is in disaster rescue missions. Swarms of robots of different sizes could be sent to place rescue workers can't reach safely to detect the presence of life via infra-red sensors. On the other hand, swarm robotics can be suitable for tasks that demand cheap designs, for instance mining tasks or agricultural foraging tasks. Also, some artists use swarm robotic techniques to realise new forms of interactive art. Another large set of problems may be solved using swarms of micro aerial vehicles, which are also broadly investigated nowadays. Swarms can be also be used in military to form an autonomous army.

2 ARGoS simulator

ARGoS (*Autonomous Robot Go Swarming*) [7] is a multi-robot simulator designed for real-time simulation of large heterogeneous swarms of robot. The multi-thread architecture of ARGoS provides scalability, low run-times and high speed up on multi-core CPUs. Simulation is central to the study of swarm robotics for several reasons. Simulation allows for cheaper and faster collection of the experimental data without the risk of damaging the real life platforms which are often very expensive, simulated experiments can potentially involve quantity of robots that would be impossible to manufacture for reasons of cost. Robots, sensor, actuators, visualisation and physic engines are implemented as user-defined module. The user can choose which module to utilise through an XML configuration file. There are many possible models for each specifics sensor or actuator characterised by differences in accuracy and computational cost. ARGoS simulator is *physics-based* –robot bodies, sensors, and actuators are simulated using physics models. ARGoS is also *discrete-time* which means that the execution proceeds synchronously in a constant step-wise fashion. ARGoS is open source and it runs under Linux and Mac OS X (these two operating systems have been utilised to run the algorithms we implemented).
In our project all the experiments were carried out using the Foot-Bot robot. The Foot-Bot is an autonomous robot that is modular at low-level: mechanics, electronics, and software. The Foot-Bot has differential motion control which uses a combination of tracks and wheels to provide mobility. The base of the Foot-Bot includes IR sensors for obstacles detection, a radio frequency identification reader and writer with an antenna situated on the bottom of the robot, a LED included inside which can be used for color-based communication with other Foot-Bot, it also has a omnidirectional camera.
In ARGoS to tell the robots what to do, you need to write a script using *Lua* as language.

3 Algorithms

We chose from the literature some scientific articles about the pattern formation and spatially target communication and when it was possible, we implemented the solutions proposed without any significant variation, however sometimes some adjustment or little changes were required. Every algorithm was implemented using *Lua* language and tested on *ARGoS* simulator.

3.1 Pattern formation

One of the main collective behaviours in swarm robotics studied in the literature is spatially-organizing behaviour. Spatially-organizing behaviours focus on how to organize and distribute robots and objects in space. Robots can be organized and distributed in space in several possible ways: aggregates, patterns, chains and structures of physically connected robots. Moreover, robots can also physically move objects to create clusters and structures. [2]

Pattern formation aims at deploying robots in a regular and repetitive manner. Robots usually need to keep specific distances between each other in order to create a desired pattern. Multi-robot pattern formation has received much attention in recent years, which can be the basics of some advanced application like multi-robot construction, collective movement, and cooperative transport. Pattern formation can be found both in biology and in physics. Some biological examples are the spatial disposition of bacterial colonies and the chromatic patterns on some animal's fur. Some physics examples are molecules distribution and crystal formation, and cells.

3.1.1 Aggregation

General description. The goal of aggregation is to group all the robots of a swarm in a region of the environment. Despite being a simple collective behavior, aggregation is a very useful building block, as it allows a swarm of robots to get sufficiently close one another so that they can interact. [2] In swarm robotics, aggregation is usually approached in two ways: probabilistic finite state machines (PFSMs) or artificial evolution. The most common approach is based on PFSMs: the robots explore an environment and, when they find other robots, they decide stochastically whether to join or leave the aggregate. In this approach, a stochastic component is often used in order to ensure that eventually only a single aggregate is formed.

Implementation. In our project, we use a simple PFSM-based aggregation behavior as described in [1]. The PFSM representing the robot controller has two states and their corresponding behaviors: **random walk** and **wait**. The robots start in random walk state wandering aimlessly in the environment. When a robot in the random walk state (a searching robot) detects another robot, it switches to the wait state. The transition from the wait state to the random

walk state can occur in two ways. First, if a waiting robot no longer detects other robots around itself, it switches back to random walk state. Second, if a waiting robot has one or more neighbors, it switches to the random walk state with probability P_{leave}. In order to prevent oscillations between aggregates and searching robots, when a waiting robot switches to the random walk state, the transition to the wait state is disabled for certain time period (T_{leave}). Inside the wait state, robots spatial disposition is obtained using a *social force* as described in [8].

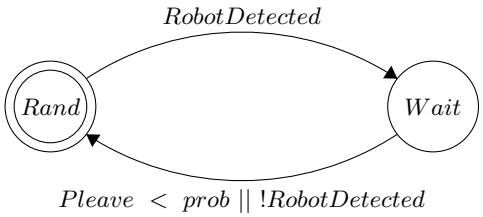

Fig. 1. PFSMs of robot aggregation [1].

3.1.2 Robot formation

General description. A robot formation can be defined as a group of robots that moves as a collective maintaining predetermined positions and orientations among its members. Having a predefined global shape is not always a requirement, and sometimes only relative positions between the members are defined. Formations with a triangular lattice structure are created using distributed control rules, using only local information on each robot. The overall direction of movement of the formation is not pre-established but rather results from local interactions, giving all the robots a common, self-organized heading. [5]

There are many applications in which robot formations can play an important role and be an advantage as compared to a single robot. For instance, formations of robots organized in a lattice can act as sensor arrays, allowing them to collect rich spatial information about their environment. Thus, formations can be very useful in search tasks, especially those in which the spatial pattern of the source can be complex as in the case of sound or odor. They can also be quite useful in mapping tasks. Redundancy in the measurements of both the environment and the relative positions among the robots might allow them to build more accurate maps than those generated using individually operating robots.

Robot formations in which only relative positions of nearby robots are known are classified by [2] into two groups: *Leader-referenced* and *Neighbor-referenced*. In the second type of formation, robots react to the positions of their neighboring robots, creating among the robots a regular pattern made up of squares,

rectangles or triangles. No leaders are designated, and in principle the global shape of the formation is not pre-specified, and is determined solely by local interactions. The main advantage of this type of architecture is that it scales well with an increasing number of robots since the position error doesn't propagate cumulatively from leader to follower.

Implementation. The developed algorithm is based on the *Physicomimetics Framework* (PF) that allows for the creation of a self-organized formation by using control laws inspired by physics [5]. The controller is fully decentralized; each robot perceives the relative positions of its neighbors and reacts to attractive or repulsive virtual forces given by these positions, making the system scalable with the number of members. Using these forces, robots can position themselves to form triangular lattices. The algorithm is implemented as a Finite State Machine (FSM) with the following four states: *Positioning*, *Alignment*, *Moving*, and *Dispersion*. States are innate to individual robots, so different robots may be in different states at the same time. The *Positioning* state works in a similar way to the PF, resulting in a triangular lattice. Once each robot considers itself well-positioned with respect to neighboring robots, it enters into the *Alignment* state and aligns its heading with its neighbors. When it has achieved proper alignment it can enter into the *Moving* state that moves the group in a common direction. While in the *Positioning* state, each robot continuously evaluates whether it is stuck in a cluster. *Clustering phenomena* occur when robots are stuck in the middle of the lattice and have insufficient force to push the other robots around them. If this is the case, the robot enters into the *Dispersion* state that allows it to get out from this deadlock situation using a variable that represents an estimation of the probability of being in a cluster. The probability of being part of a cluster is increased if the average distance to the closest neighbor is below a certain threshold and keep decreasing over successive time steps, otherwise this probability is decreased.

3.1.3 Circle

General description. A large number of robots form an approximation of a circle shape and also distribute themselves nearly uniformly within the circle by executing an identical simple algorithm.

A centralized solution for this problem may consist of the following steps: (1) Count the total number of robots, (2) obtain the current position of each robot, (3) determine a final position for each robot, (4) generate a schedule for moving the robots from their current positions to their respective final positions, (5) broadcast the schedule to the robots, and (6) let the robots start executing the schedule simultaneously. Another centralized solution would be to let the robots communicate regularly and maintain the same information at all times, so that each robot can generate a plan for the entire set of robots by using the same information and then execute the portion of the plan applicable to itself. It is conceivable, however, that such solutions may not always be desirable for the following reasons:

1. If the number of robots is large, the communication overhead required for collecting all relevant information may be prohibitively large. Also, it may be computationally infeasible to generate a schedule for the entire set of robots in real time.
2. A new schedule for the entire set of robots may have to be generated every time a robot becomes faulty.

Furthermore, the centralized solution can be considered to be rather "unnatural", since an exact final position of every robot must be determined before a schedule is generated, whereas none of the problems stated above requires that any specific robot be moved to any specific position. The algorithm that we implemented, as described in [9], is fully distributed in the sense that each robot plans its motion individually based upon a given goal of the group and the observed positions of other robots.

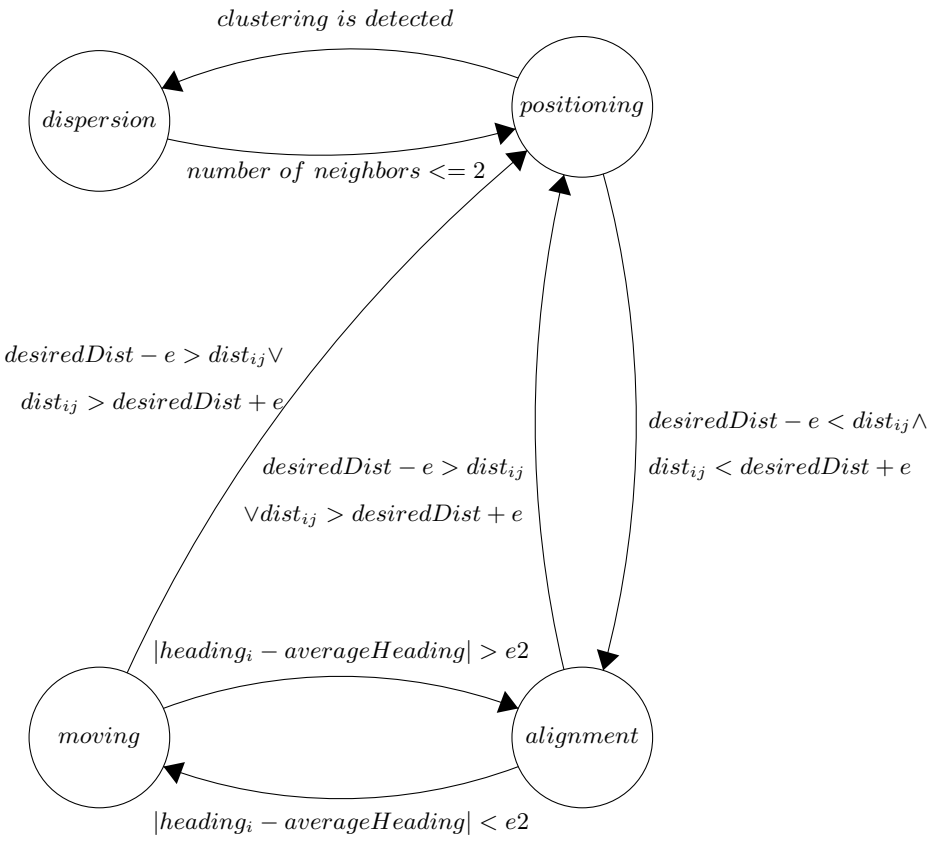

Fig. 2. FSM of robot formation [5].

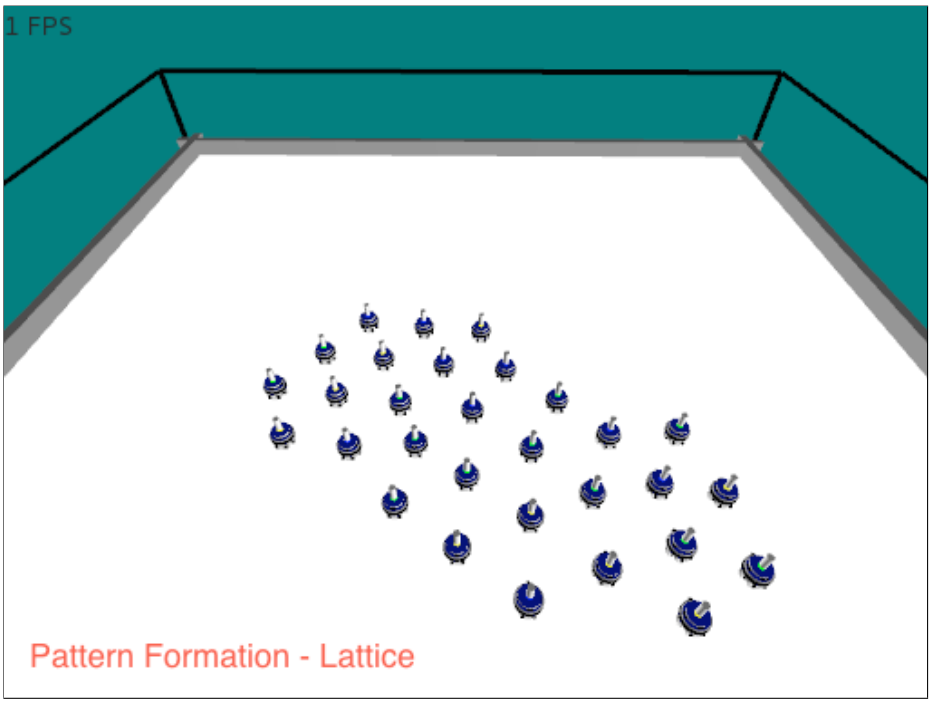

Fig. 3. Result of robot formation experiment with ARGoS

The basic idea is to let each robot execute a simple algorithm and generate its own schedule adaptively based upon the given goal and the positions of certain other robots. No explicit communication among the robots is necessary, and usually most of the robots execute the same algorithm. The algorithms are designed to work satisfactorily even if (1) the total number of robots is unknown and (2) a small number of robots become faulty during the operation, as long as non faulty robots can determine which robots are non faulty. Therefore, our solutions do not have the drawbacks of centralized solutions mentioned earlier.

Implementation. The objective is to move the robots so that they will form an approximation of a circle having a given diameter D.

The algorithm **CIRCLE** will be executed by each robot R, which continuously monitors the positions of a farthest robot R' and a nearest robot R'', breaking ties arbitrarily, and moves as follows in real time, where d is the distance between R and R':

Case 1: If $d > D$, then R moves toward R'.
Case 2: If $d < D - \delta$, then R moves away from R'.
Case 3: If $D - \delta < D$, then R moves away from R''.

Where $\delta > 0$ is a small constant.

Case 3 of **CIRCLE** has the effect of distributing the robots more uniformly. Cases 1 and 2 will ensure that the distance between any robot and a robot

farthest from it will be approximately D, so that the width of the resulting distribution of the robots will be D in any direction. The algorithm does not have any built-in mechanism for terminating its execution.

Algorithm 1 Circle

Require: R current robot
Require: R' farthest robot
Require: R'' nearest robot
Require: d distance between R' and R''
Require: D desired diameter
  if $d > D$ **then**
    R moves toward R'
  end if
  if $d < D - \delta$ **then**
    R moves away from R'
  end if
  if $D - \delta < D$ **then**
    R moves away from R''
  end if

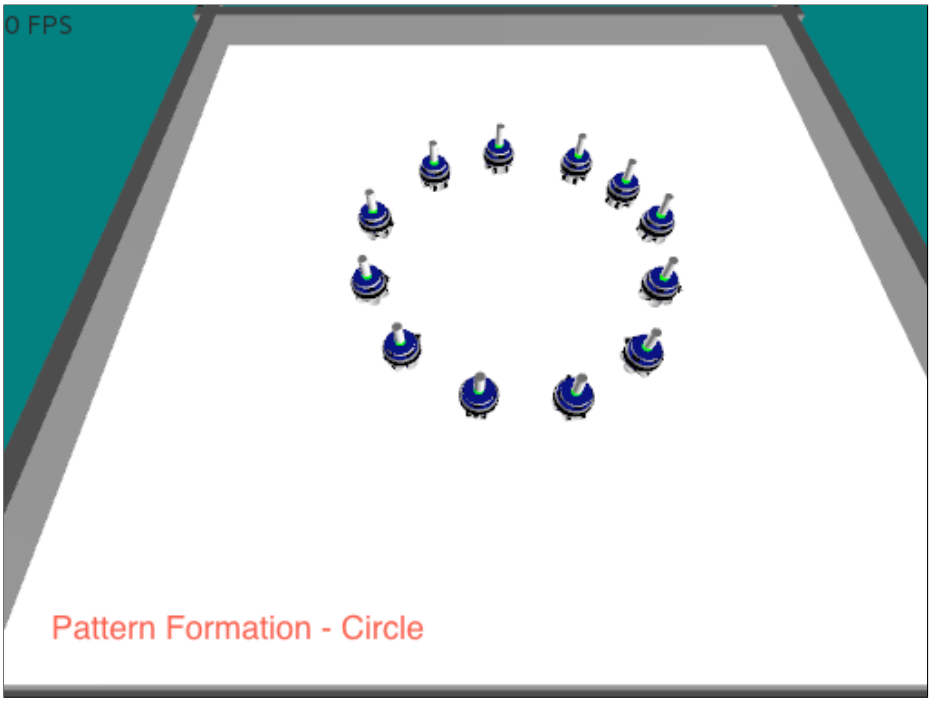

Fig. 4. Result of robot formation experiment with ARGoS

3.1.4 Fill circle

General description. Robots are distributed nearly uniformly within (an approximation of) a circle having diameter D.

Implementation. As described in [9], robot R continuously monitors the positions of a farthest robot R' and a nearest robot R'', breaking ties arbitrarily, and moves as follows in real time, where d is the distance between R and R'

Case 1: If $d > D$, then R moves toward R'.
Case 2: If $d < D$, then R moves away from R''.

Case 1 of FILLCIRCLE ensures that the distance between any two robots will be at most D. Case 2 has the effect of distributing the robots more uniformly. As is the case with algorithm CIRCLE, we can move the robots in any direction without destroying the distribution by moving any one robot manually.

Algorithm 2 Fill circle

Require: R current robot
Require: R' farthest robot
Require: R'' nearest robot
Require: d distance between R' and R''
Require: D desired diameter
  if $d < D$ **then**
    R moves toward R'
  else
    R moves away from R''
  end if

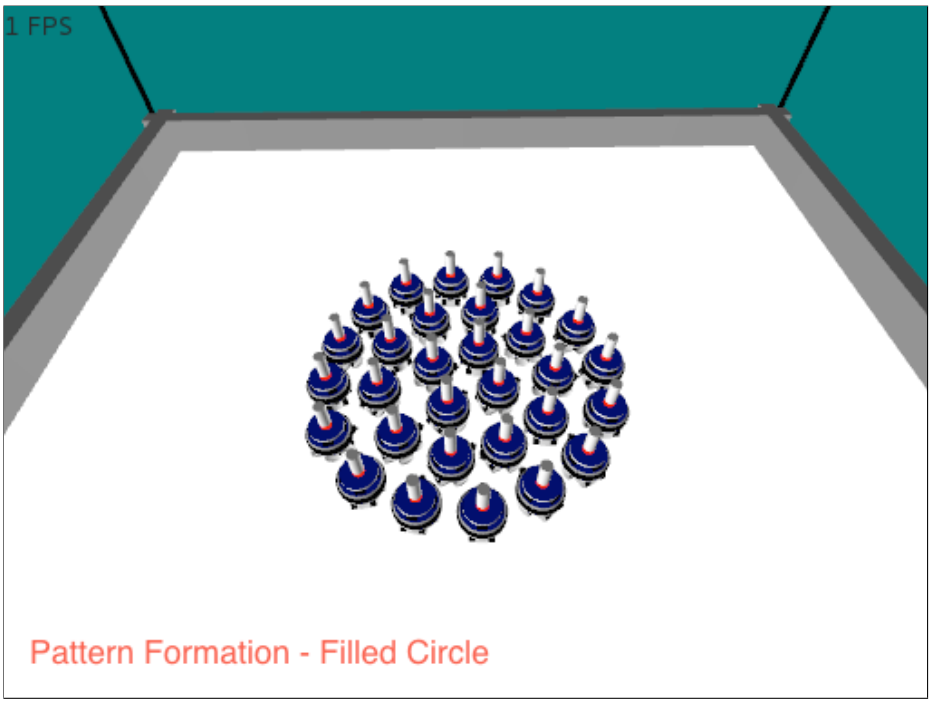

Fig. 5. Result of fill circle experiment with ARGoS

3.1.5 Following/Splitting

General description. This is a method for dividing N robots into m groups of approximately equal sizes, where m is much smaller than N. First, let the robots execute algorithm CIRCLE. Next, m robots which will be the "leaders" of the m groups are selected (i.e. using a leader election algorithm). To make the sizes of the groups approximately the same, the m robots must be chosen so that they are nearly evenly distributed. Finally, we do the following, possibly concurrently.

1. Control the m robots manually and move them away from each other (we used a random walking in our project).
2. Let all other robots R execute FOLLOW algorithm given below.

Implementation. As described in [9], we implemented in ARGoS the FOLLOW algorithm (Fig. 6): Robot R stays in the current position until one of $I(R)$ and $r(R)$ moves, and then "follows" the one, call it R', which has started to move first (breaking a tie arbitrarily) as follows: R memorizes the initial distance d between R and R', continuously monitors the position of R', and moves, in real time, toward R' if the distance between R and R' becomes greater than d. The motion detection function (see Algorithm 3.) is need to keep track of the first

robot in front of R (which is also the robot to follow). It works by checking any variation of the angle (inside a certain range) or distance with the robot to follow.

Algorithm 3 Motion Detection

Require: R robot to check
Require: S set of visible robots
  for all $i \in S$ **do**
    $angle_{diff} \leftarrow (R_{angle} - angle_i + 180 + 360) \mod 360 - 180$
    if $angle_{diff} \leq 45 \wedge angle_{diff} \geq -45$ **then**
      min $fromRobot$
    end if
  end for
  if $|R$ - distanceFromMinRobot$| > \epsilon$ **then**
    return TRUE
  else
    return FALSE
  end if

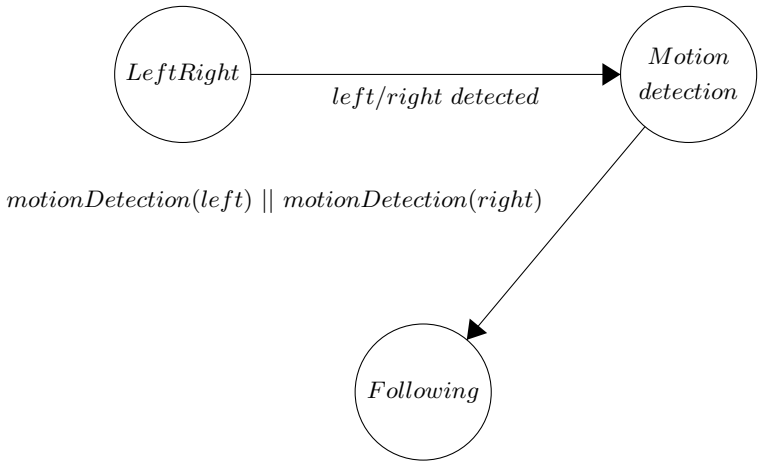

Fig. 6. FOLLOW FMS

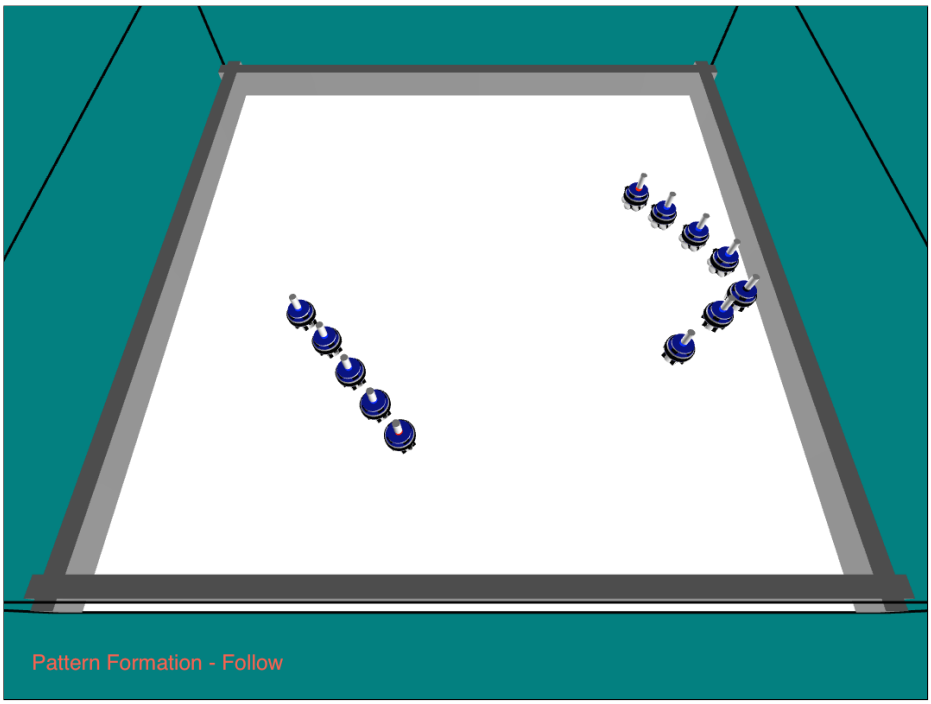

Fig. 7. Result of FOLLOW experiment with ARGoS

3.2 Spatially targeted communication

Spatially targeted communication enables one robot to establish dedicated communication links with individual robots or with selected groups of robots based on their position in the environment. The system does not rely on any form of global information.[4]

Spatially targeted communication has been achieved using protocols built on top of situated communication modalities. In situated communication modalities, localized information about the sender is implicit in the message delivery mechanism. As an example of situated communication, consider the case in which a robot receives the message "stay away, I am near danger". This message is only meaningful if the communication is situated, that is, if the receiving robot can estimate the location of the sender. We implemented an algorithm which permits a robot to establish a spatially targeted communication link with a particular other robot among a group using situated communication based on on-board LEDs and camera. We use a binary selection process, whereby sender robot initially communicates with all others robots within visual range and iteratively eliminates robots, until only the selected robot is left. We show how an established one-to-one communication link can be expanded to a one-to-many communication link with a group of co-located wheeled robots. The approach is

applicable to any communication modality that is situated and supports at least three distinct signals.

3.2.1 One to one

General description. We describe the approach used to establish a one-to-one communication link between robots. Given a set $C := red, blue, green$ of distinctive signals available to both robot types (leader and target), communication can be established by the leader robot with a particular target robot by the means of an iterative selection process. We assume that the leader robot has already selected a particular target robot with which it wishes to communicate. The leader robot first attracts the attention of all robots in visual range by signaling c_1 = red, the SOS signal. All robots able to perceive the SOS signal register to the iterative selection process by replying with c_2 = blue. The leader robot responds to this initial registration with a matching handshake using c_2 = blue. After this handshake, the iterative selection process starts. At each iteration, every robot that is still part of the selection process randomly chooses and illuminates a color from the set Cs. At each iteration, the leader robot illuminates its LEDs to match the color chosen by the selected target robot with which it wishes to communicate. At the end of every iteration, only those robots whose color match with the color of the leader robot remain part of the selection process. The robots which are not part of the selection process do not illuminate any color. This iterative selection process continues until the selected target robot is the only illuminated robot. In this case, the leader robot indicates the termination of the selection process to the target robot by repeating c_1 again. The remaining robot acknowledges this by matching the leader robot's color. The leader robot and the remaining robot have now established a spatially targeted communication link.

Implementation. We implemented, as described in [4], two controllers: one for the leader robot (Fig. 8 (a)) and one for the other robots (Fig. 8 (b)). Both controllers are behavior-based, completely distributed, homogeneous and are represented by a finite state machine (FSM). Both leader and other robots use three colors to communicate: red, blue and green.

Once the leader robot has determined that it needs to communicate with a particular robot, it enters the SOS state (red color). The transition from SOS to SP is triggered when the selected robot acknowledges the SOS. While in state SP, the leader robot keeps matching the color displayed by the wheeled robot. If the selected wheeled robot is the only robot displaying any color, the leader robot changes its state to CE to confirm the establishment of the communication.

The FSM implemented on the other robots (which aren't the leader) consists of the three states ACK (acknowledge), SP (selection process), CE (communication established) and an end state which causes the robot to terminate. The state ACK is associated with the predefined color blue and the state CE is associated with red. The state SP is given two colors, namely blue and green. The

robot enters the state ACK as soon as an SOS color is perceived on the leader robot. In case the ACK color is matched by the leader robot, the transition to state SP is triggered. When entering the state SP, each robot randomly selects and displays a color from the set of colors provided to the state. At the same time, each robot starts incrementing an internal timer t. Whenever this timer t exceeds a fixed threshold τ, the robot examines the color displayed on the leader robot to determine whether to remain in state SP or to leave the state and terminate the behavior. The timer mechanism provides the leader robot sufficient time to perceive, process and react to the colors displayed by the robots. When a robot is in state SP and detects the CE color on the leader robot, the robot can safely assume that it is the robot with which the leader robot wishes to communicate. In this case, the robot confirms the termination of the selection process by displaying its CE color.

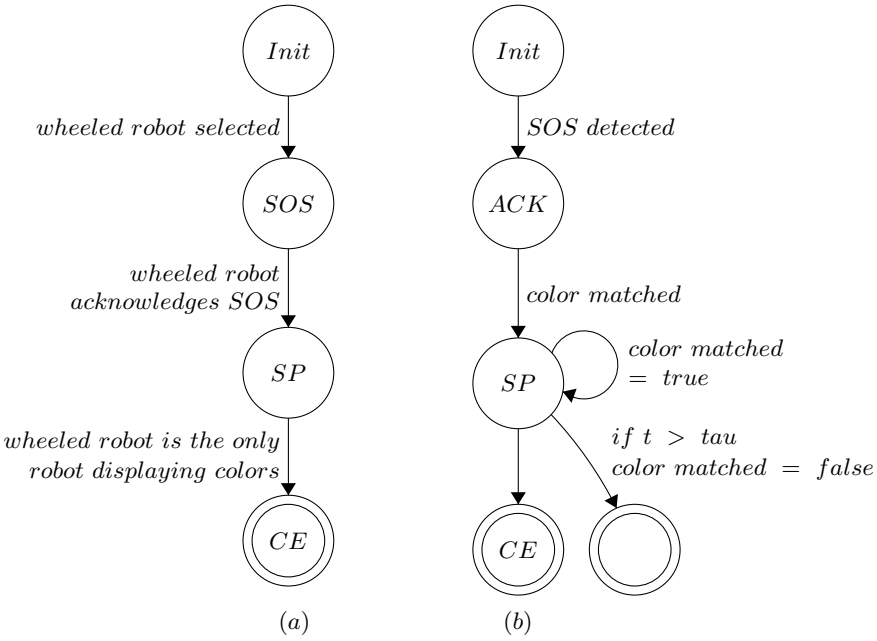

Fig. 8. FSMs of one-to-one communication [4].

3.2.2 One to many

General description. We describe how an already established one-to-one communication link can be expanded to become a one-to-many communication link between a leader robot and a group of co-located robots. The approach used works by iteratively growing up a group of robots around a seed robot with which a one-to-one communication link has already been established. In the first iteration, the seed robot is the sole member of the group. In each subsequent iteration, the leader robot may send a request to increase the size of the group. Only wheeled robots that are within visual range of an existing group member process this request. We refer to the robots in this range as candidate robots. Robots that are not directly adjacent to the existing group (i.e., they detect other robots between themselves and the group) eliminate themselves as potential candidates. We refer to the remaining candidate robots as the closest candidate robots. These closest candidate robots now signal their candidacies to the leader robot. The leader robot completes the iteration by granting group membership to some or all the closest candidate robots, depending on the type of growth required. To achieve an exact group size, the leader robot can request the closest candidate robots to relinquish their candidacies probabilistically.

Implementation. We implemented the algorithm, as explained in [4], with some changes, we will describe later in this section, to overcome possible prob-

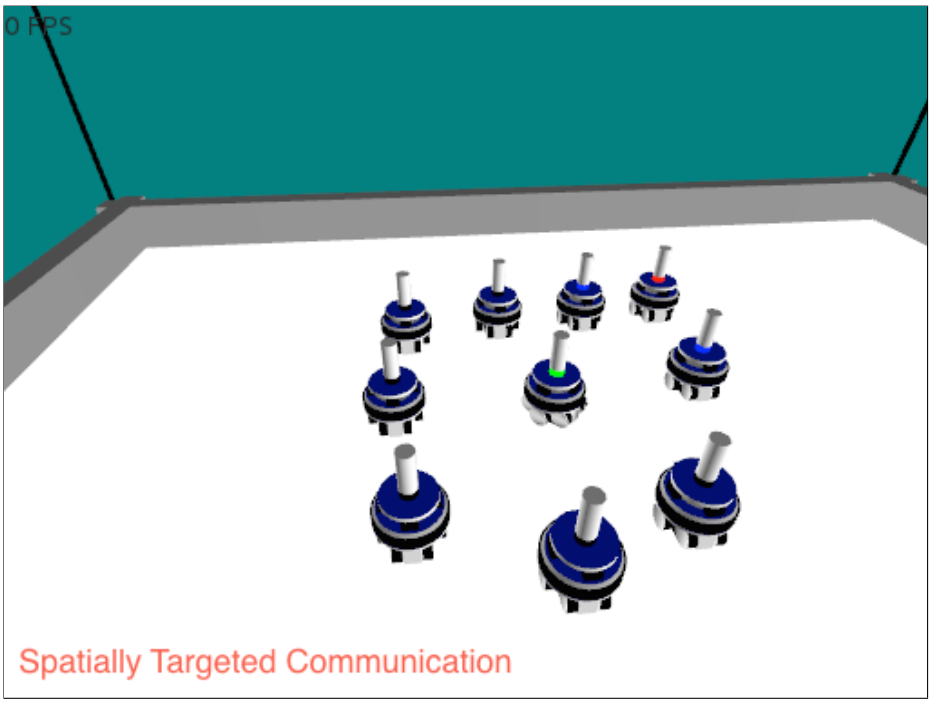

Fig. 9. Result of one-to-one communication experiment with ARGoS

lems like deadlock in particular spatially distribution of the robots (i.e. when there's only one robot next to the seed the condition to trigger ta2, as described in the article, can't happen). The leader's FSM (Fig. 10 (a)) implemented considers two states STA (stable group size) and ADD (add members). The state STA displays the color red and the state ADD the color green. The transition *ta1* is triggered if the number of robots in red (in the group) is smaller than the desired group size. The transition *ta2* is triggered when the number of robots displaying red or green is less or equal to the desired group dimension, this happens only every 20 steps of the simulator because of possible deadlocks when a not sufficient number of robots in the CNN state (green led on) can be turned on. When the group size is reached the transition *ta5* terminates the behavior. The state LEA and the transitions *ta3* and *ta4* allow the leader robot to request the closest candidate robots to relinquish their candidacies. The leader robot displays the color blue while in state LEA. If the leader robot is in state ADD and the sum of the group members and the closest candidate robots exceeds the desired group size, the transition *ta3* is triggered. On the other hand, if the leader robot is in state LEA and the sum of the group members and closest candidate robots is below or equal to the desired group size, the transition *ta4* is triggered and the control program returns to state STA.

The FSM implemented on the robots (which are not the leader) (Fig. 10 (b)) consists of 4 states HIB (hibernate), CAN (candidate), CCN (closest candidate) and MEM (group member). The LEDs are switched off while in state HIB, whereas blue is displayed while in state CAN, green while in state CCN, and red when in state MEM. The state transition *tw1* is triggered if the leader robot illuminates green, at least one non-leader robot in the visible range displays red, no other robot displaying blue is perceived closer to the group and no robot in the visible range displays green. The transition *tw2* is triggered if a candidate robots sees another candidate robot closer to the group than itself. While in state CAN, a timer t is incremented. Whenever this timer t exceeds a given threshold τ , transition *tw3* is triggered. The triggering of transition *tw3* depends on the outcome of a Bernoulli trial with probability $p_j = 0.5$: if successful, the transition is triggered otherwise the timer t is reset. This timer mechanism provides the robots sufficient time to determine the closest candidate robots. Finally, the transition *tw5* is triggered when the leader robot grants the membership to the group by displaying red. If a robot is in state CCN and the color blue is displayed by the leader robot, a timer t is incremented. Whenever this timer t exceeds a given threshold τ, transition *tw4* is triggered. The robots utilize the outcome of a Bernoulli trial with probability $p_l = 0.5$ to decide whether to trigger transition *tw4* or to reset the timer t. These extended control programs allow the leader robot to grow a group containing an exact number of robots by iterating between the states STA, ADD and LEA until the desired group size is reached.

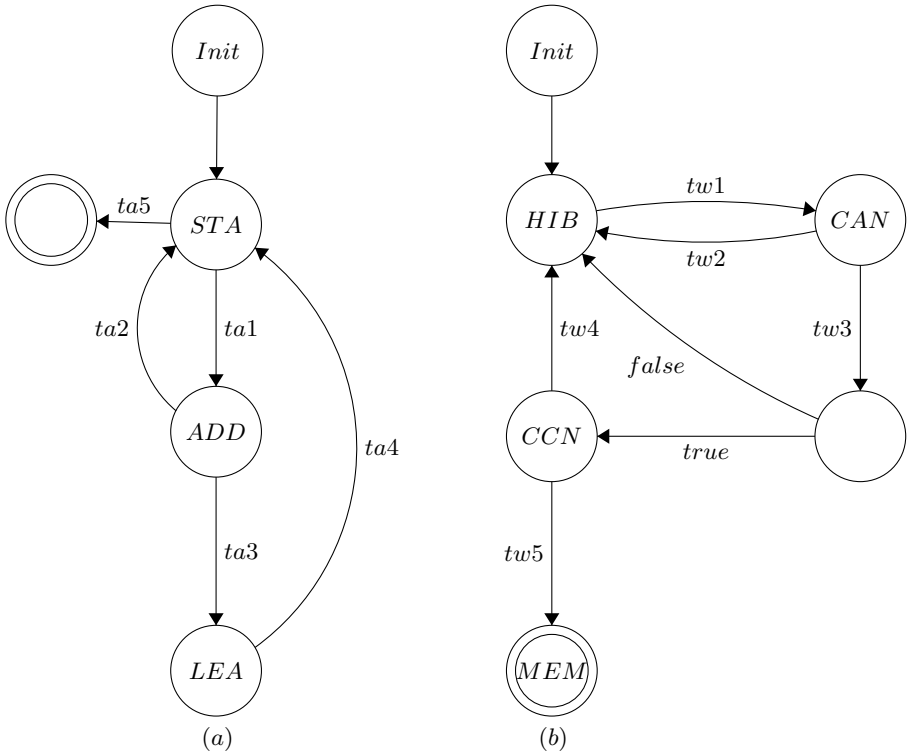

Fig. 10. FSMs of one-to-many communication

3.2.3 Leader election

General description. One of the main features of swarm robotics is the local character of interaction of robots, with each other and with their environment. This kind of interaction is called implicit communication which means that each robot in the group directly interacts only with neighbours within some limited visibility range. In such systems, it usually follows that robots make decisions independently on further actions, guided by some simple rules of local interaction. However, the overwhelming majority of examples of task solution in the field of swarm robotics concern the coordinated movement of a swarm. As a solution to the basic problem of coordinated movement of robots, it is sufficient that there is a leader. However, more complex challenges solved by a group of robots require differentiation of their functions and, generally speaking, distribution of tasks between robots; this is perhaps one of the most problematic concepts of swarm robotics. Leader election is the process of designating a single robot as the organizer of some task distributed among several other robots.

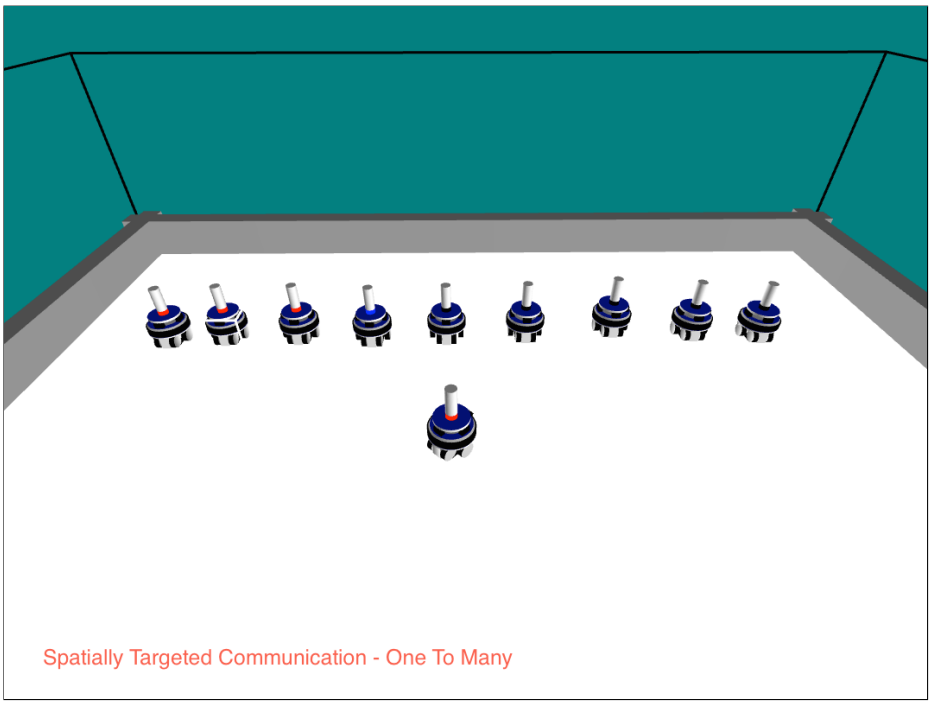

Fig. 11. Result of one-to-many communication experiment with ARGoS

Implementation. To implement the leader election algorithm on the Foot-Bot, we got inspired from the *Random Competition based Clustering*(RCC) algorithm [10]. The FSM (Fig. 13) implemented on the robots consists of 3 states INIT, LDR and SLV. The main idea is that each robot, which isn't currently a leader, can initiate the process to request the leadership by broadcasting a red colour signal (INIT state). The first robot that broadcasts the red signal will be elected as the leader (LDR leader state). All its neighbour robots after perceiving this signal give up the right to become the leader and turn the blue led on (SLV slave state). The robots with the blue led on periodically check the leader presence and in case they don't see any red light the process starts again. To reduce concurrent broadcasts, we introduced a random timer and each robot defers a random time before its leader claim. If a robot sees a red light on inside the INIT state or LDR state, it gives up its broadcast and goes directly into SLV state.

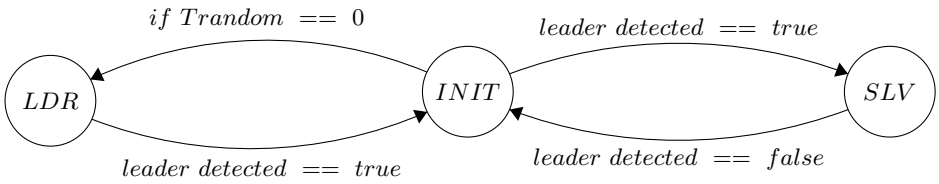

Fig. 12. FSM of leader election

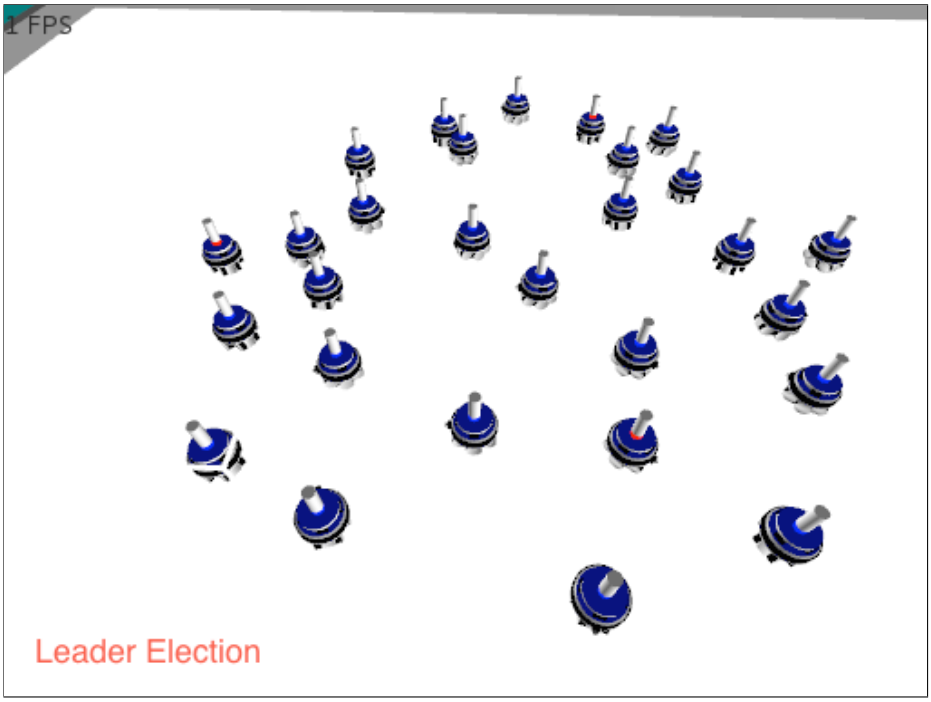

Fig. 13. Result of leader election experiment with ARGoS

4 Fault tolerance

Collective robotic systems (or, equivalently, multi-robot teams) have many potential advantages over single-robot systems, including increased speed of task completion through parallelism, improved solutions for tasks that are inherently distributed in space, time, or functionality; cheaper solutions for complex applications that can be addressed with multiple specialized robots, rather than all-capable monolithic entities and increased robustness and reliability through redundancy. Systems with these characteristics could, potentially, exhibit very high levels of robustness, in the sense of tolerance to failure of individual agents, much higher levels of robustness than in complex distributed systems based on traditional design approaches. However, that robustness comes at a price. Complex systems with swarm intelligence might be difficult to control or mediate if they started to exhibit unexpected behaviours. Further, collective robotic systems may require increased communication to coordinate all the robots in the system. Increasing the number of robots can lead to higher levels of interference, as the robots must act without complete knowledge of their teammates' intentions. Additionally, collective systems will often experience increased uncertainty about the state of the system as a whole. If not properly handled, all of these challenging issues can lead to a collective system that is unreliable and faulty. We discuss the problem of reliability and fault tolerance in collective robot systems and general mechanisms for fault detection and recovery and we provide an overview of the possible approaches that may be appropriate for achieving reliability and fault tolerance in a variety of multi-robot systems.

Even the most carefully designed and tested robots may behave abnormally in some situations. In some cases robot collectives may be designed to be inherently robust, the cooperative control is designed to work in spite of certain failures, which may not have to be explicitly identified or diagnosed. Nevertheless, even in such systems, the designers typically have to analyze and address the many types of faults that may occur in the robot collective, so as to design robust control solutions.

The main common terms used to discuss unreliability and faulty systems in robot collectives are defined as follows:

- Fault: a deviation from the expected behavior of the robot system. In some applications, the determination of a fault is a binary designation in which a fault has either occurred or it has not. In other applications, however, it may be more meaningful to measure degrees of faults, along a continuum from a completely nominal system to a completely failed one (in our experiments we used noise added to sensor and actuators).
- Reliability: the probability that a device, system, or process will perform its prescribed duty without failure for a specified period of time when operated correctly in a specified environment. Different measures of reliability can be given to individual robot components, to individual robots, or to the entire collective of robots. Of particular interest is measuring the reliability of the robot team as a whole, regardless of the reliability of the individual components or robot team members.

– Fault Tolerance, or Robustness: the capability of a system to continue operation, perhaps at a reduced level (i.e., graceful degradation), even when undesired changes in the internal structure or external environment occur. In the context of robot collectives, fault tolerance and robustness typically refer to the ability of the system to deal with failed robots.

One of the key motivations building for collective robot systems is to achieve increased overall reliability through the redundancy of multiple robots. The idea is that several individual robot failures could be overcome simply through redundancy, under the assumption that even if some number of robots fail, sufficient numbers of properly functioning robots will be available to successfully accomplish the application (in the next section we show how these concepts are applied to our experiments). To realize this objective, the system must be designed with these faults in mind:

– *Individual robot malfunctions*: many different causes of failure can be roughly categorized as physical failures, software control failures, or human control errors. The most common causes of failure in multi-robot system are: effector failures, unstable control systems, platforms designed only for a narrow range of operating conditions, limited wireless communication range, and insufficient bandwidth.
– *Local perspectives that are globally incoherent*: by definition, distributed systems are composed of individual entities that maintain only a local perspective. While some non local information may be shared between entities in the system, no individual entity will have complete knowledge across the entire domain of the collective system. Actions taken based only on local information could lead to globally incoherent solutions. For example, the interaction of the local control approaches of individual entities may cause unexpected emergent consequences that are undesirable and the result is a collective system that does not behave in a globally coherent fashion.
– *Interference*: any system with multiple robots sharing the same physical space must deal with contention, or interference, that may arise when robots operate near each other. Without properly addressing physical workspace contention, the benefits of multiple robots can be erased by harmful inter-robot interference.
– *Software errors or incompleteness*: complex software systems are notoriously difficult to validate. Large collective robot systems consist of multiple robots, each of which may be rather complex. Ensuring that the autonomous control software for these systems is accurate for all possible situations and environments to which the collective may be applied is currently beyond the state of the art. Thus, providing the robot collective with an ability to deal with not modeled events is important for achieving robust solutions.
– *Communications failures*: many collective robot systems can achieve increased performance by enabling robots to share partial information about their local state or environment, even a small amount of shared state can significantly improve team performance. However, collective systems must also

be designed to properly respond even when this communicated information is not available.

4.1 Pattern formation

To study the robot behaviour under real conditions of actuators and sensors, we introduced noise using values between the range $[0; 6]$, whose extremes correspond respectively to an ideal condition (in simulator only) and to the worst case possible.

Robot formation. We tested the pattern formation algorithm adding some noise to the actuators and to the sensors using values from the range above. We first tried to only actuators noise, which means that the wheels add a random movement to the desired one and we saw that the pattern was still stable due to tolerance in the software measurements, and the feedback the robot obtain using the camera to sense the neighbours robot position. Experiments with sensors noise yield to the same result as above, since a single robot measure multiple points to adjust and maintain its positions, (two and more neighbour robot positions are detected) so that a single wrong information can't influence a lot the entire process. In case of total unavailability of the motors of very few robots relative to the total number of the group, the flock still working fine, because the lack of a single robot don't condition the behaviour of the other non-faulty robots. The general behaviour can be seen as the sum of many very small contributions which can't alter the overall system by it-selves. In the case of led failure (led turned off) the robot is simply ignored and seen (if an obstacle detection is active) as a normal obstacle.

Circle. For small values $[0; 4]$ of noise the circle formation still worked fine for the same reasons described for the robot formation (even if only the farthest and the nearest robot are detected is sufficient to guarantee enough position adjustment), on the other hand for high values of noise $[5 - 6]$ the system couldn't converge to a stable form. There is a possibility that the algorithm proposed can be made fault-tolerant in the following sense: it should be easy to see that even if a small number of robots become faulty during the execution, the non-faulty robots can continue to form and maintain a desired distribution, as long as they can determine which robots are non-faulty and ignore the faulty robots. On the other hand, we also note that if robots can be faulty and remain operational without being detected to be faulty by other robots, then a single faulty robot can defeat such distributed algorithm.

Following/splitting. A robot with motor failure can compromise the entire system depending on its position in the chain. Robots follow the leader by following only the nearest robot in front of them, if a motor failure occur in the last one all the robots behind him stop running. A fault detection system is strictly necessary to overcome this kind of issue, because the algorithm is not robust enough to overcome the problem. A led failure in a single robot doesn't

compromise the entire chain, as long as a non-faulty robot is inside the view range of the robot immediately behind the faulty one. Sensor noise inside the reference range doesn't influence the correct work of the swarm, due to the high tolerances possible in the measurements.

4.1.1 Robot replacement

In the original Circle algorithm, we saw that there was already an intrinsic robustness to tolerate the faulty robots that is the robots which turn off their led are ignored and the process still works fine. Nevertheless we wanted to improve the system in order to maintain a fixed number of robots forming the circle even when a robot (or more than one) goes faulty. We modified the original algorithm by adding a new WAIT state to the FSM robot controller (Fig. 14). In the WAIT state robots don't move but continuously broadcast a message containing their ID and wait for an SOS signal from a faulty robot. The robot ID is a random number of 10 byte which is generated by each robot at the initialisation phase. In the initial spatial configuration now there are some robots in WAIT state and the other ones which perform the Circle algorithm as usual. A robot which is actively forming the pattern, as soon as it detects a fault, turns off its led, performs a random walking (to keep itself away from the non faulty robots) and scans for the radio messages(from robots in the WAIT state). The idea is to select the nearest robot ID in WAIT state and then send to it an SOS signal. The SOS signal is an exact copy of the robot ID which has to replace the faulty one. A robot in the WAIT state when receives back its ID number, immediately leaves the WAIT state and starts participating to circle formation so that the number of robots performing the main task is mostly the same.

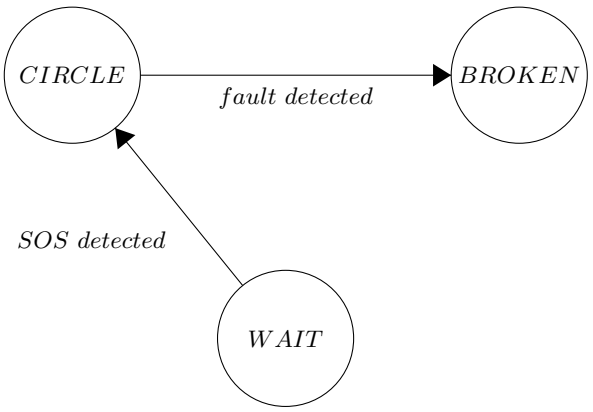

Fig. 14. FSM of CIRCLE algorithm with robot replacement

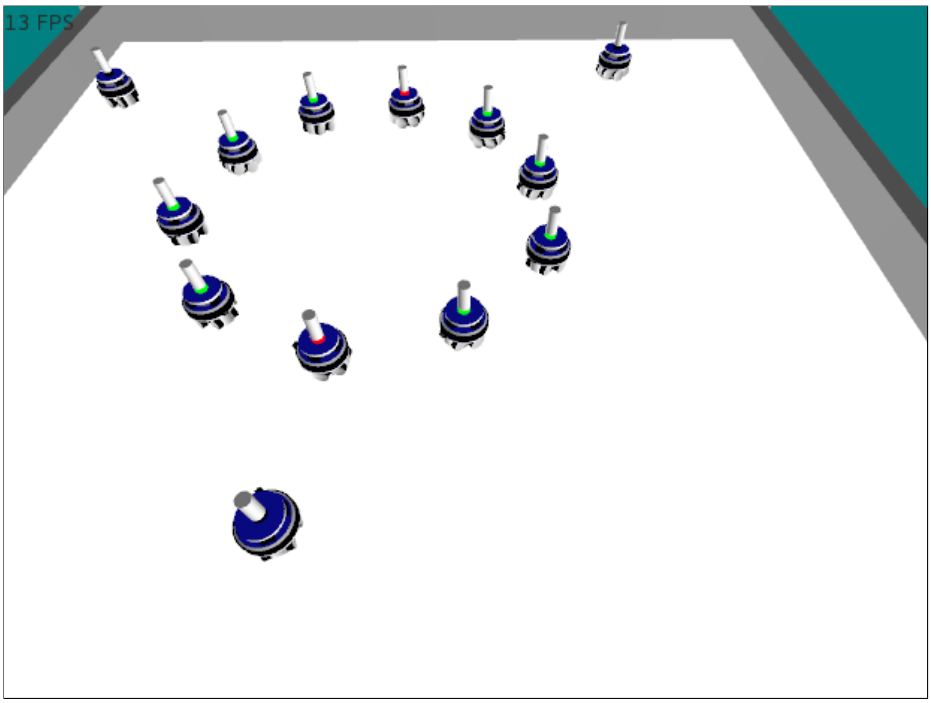

Fig. 15. Robot replacement in Circle algorithm

4.2 Communication

One-to-one communication. In the communication algorithms we can distinguish several scenarios of led fault in the slave robots. If the led breaks down before the ACK state, it won't be perceived during the selection process by the leader robot. If the led breaks down in a non-chosen robot, the system still keep working fine because it would have turned off anyway at a certain point of the process. The only critical case is when the led failure occurs in the chosen robot and the communication can't terminate. In this last case, only the leader can detect the failure.

The slave robots can detect a leader failure because when the process begins they memorize its relative position. If the led fault happens to the leader robot the only way to counteract this failure is choosing another leader to substitute the faulty one.

Leader election. In case of led failure, we can distinguish two possibilities: the led failure happens to a robot before the election process has terminated or it may happen to the leader robot. In the first case the faulty robot simply won't

compete for the election and in the second case the other robots will start again the process to choose a new leader.

4.3 Communication fault recovery

The most part of the algorithms we implemented relied on light signals using led to communicate with the neighbour robots. The led permits the camera to determine the distance and the angle of the robot that has its led turned on. In case of led fault, to keep working, robots could switch to another form of communication using the range and bearing module to exchange the same information that is using another channel, in this case a radio-frequency transmitter-receiver instead of using light signals.

The FSM (Fig. 16) describes, at the highest level, the fault recovery system of communication channel switching. In the *Led* state the robot are operating using the omnidirectional camera and the the tower led to emit the light signals, whenever a robot detects a led fault it broadcasts an *SOS signal* to the other robots nearby and then switches to the radio communication modality. A robot which is still working in Led modality when receives an *SOS signal* switches directly to radio modality even though it hasn't detected any led fault.

We implemented end tested this solution to the Circle and Fillcircle algorithm and to the One-To-One spatially targeted communication algorithm.

Circle/Fillcircle. In this section, we explain in more detail how we applied the concept of channel communication switching described above to the Circle and Fillcircle algorithm. The original algorithm implementation doesn't consider the faulty cases in which robots could fall. It may happen that during the pattern formation process the communication through light signals is not still possible, due to led faults or too much light noise in the environment from other sources. As soon as robots detect bad working conditions, they signal to the others the

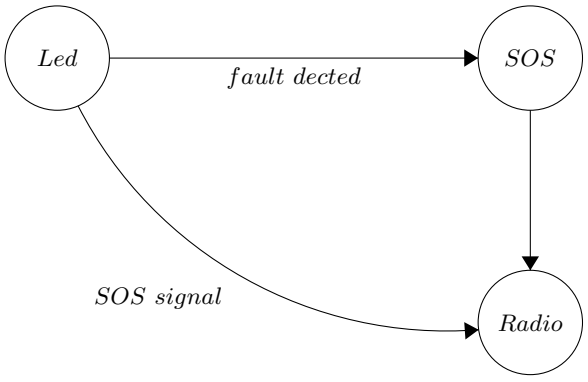

Fig. 16. FSM of communication fault recovery

necessity to actuate a communication channel switching by broadcasting an SOS message using range-bearing radio module instead of blocking the entire job. To make sure that a robot of the group involved is informed of the issue, each robot first broadcasts the SOS signal again and then keeps doing the task by getting the position and information relative of its neighbours using the radio module.

The pseudo-code 4 shows how we implemented the solution that we proposed, with minimal changes to the original Circle (and Fillcircle) algorithm. The most important adding is the function 5 that deals with the SOS signal management.

To simulate a faulty condition in a specific robot, we modified the FSM which models the robot behaviour by adding a special state *broken*. When a robot is inside a broken state it starts acting as if a faulty event happened. For example in this state we turn off the tower led in case of led fault or we stopped the wheel motor in case of motor failure.

Algorithm 4 Circle Fault Tollerance

Require: R current robot
Require: R' farthest robot
Require: R'' nearest robot
Require: d distance between R' and R''
Require: D desired diameter
  if $CheckInternalFault \vee CheckSOSMessage$ **then**
    $broadcast \leftarrow SOSmessage$
    $changeTypeCommunication$
  end if
  if $d > D$ **then**
    R moves toward R'
  end if
  if $d < D - \delta$ **then**
    R moves away from R'
  end if
  if $D - \delta < D$ **then**
    R moves away from R''
  end if

Function 5 CheckSOSMessage

Require: R set of robots inside radio communication range
Require: SOS fault message
  for all $i \in R$ **do**
    if $message_i == SOS$ **then**
      return TRUE
    end if
  end for
  return FALSE

One-to-one communication. The FSM (Fig. 17) shows how we modified the one-to-one communication algorithm, to adapt it to switch communication channel in case of led fault or when the environment doesn't permit information exchange through light signals. Except for the final state both the leader robot and the slave robots can do the communication switching at any time. Similar to the Circle and Fillcircle algorithm, when a robot operating in led mode detects a fault or receives an SOS message, it immediately changes the communication device and keeps finishing its job. In one-to-one communication algorithm it's essential for a robot to know not only the distance and the angle between its x-axis and its neighbours but also knowing the neighbours led color. We needed to encode this information by choosing a simple color exchange protocol. The Foot-Bot radio module permits robots to broadcast a 10 byte length message, in our implementation we used 3 byte (red, green, blue) to signal the current robot color via radio transmission.

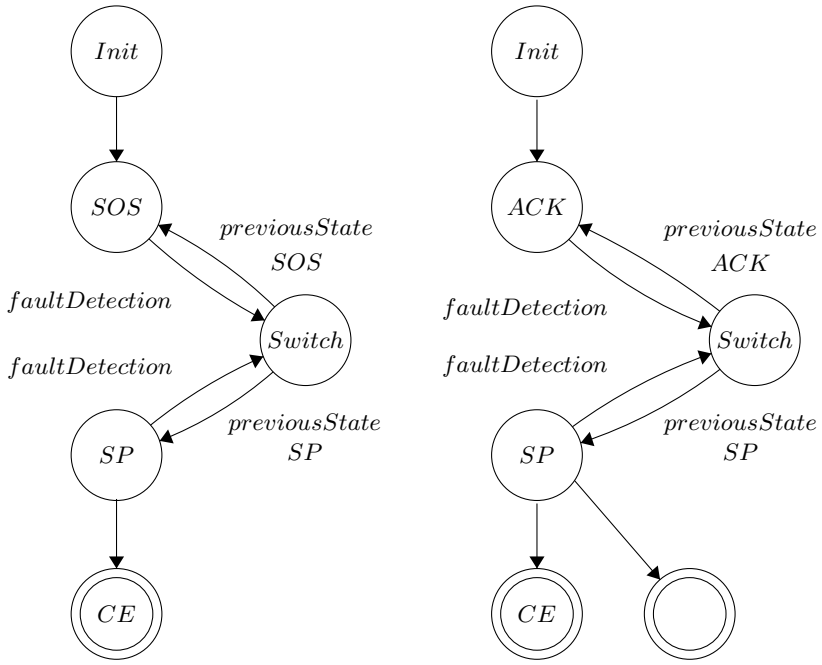

Fig. 17. FSMs of one-to-one communication with fault tolerance

Bibliography

[1] L. Bayindir and E. Sahin. Modeling self-organized aggregation in swarm robotic systems. In *Swarm Intelligence Symposium, 2009. SIS'09. IEEE*, pages 88–95. IEEE, 2009.

[2] M. Brambilla, E. Ferrante, M. Birattari, A. F. D. Roosevelt, M. Brambilla, E. Ferrante, M. Birattari, and M. Dorigo. Swarm robotics: a review from the swarm engineering perspective. *Swarm Intelligence*, page 2013.

[3] N. Correll and D. Rus. Architectures and control of networked robotic systems. *Handbook of collective robotics: fundamentals and challenges, Pan Stanford Publishing Pte*, pages 81–103, 2013.

[4] N. Mathews, A. L. Christensen, E. Ferrante, R. O'Grady, and M. Dorigo. Establishing spatially targeted communication in a heterogeneous robot swarm. In *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems: volume 1-Volume 1*, pages 939–946. International Foundation for Autonomous Agents and Multiagent Systems, 2010.

[5] I. Navarro, J. Pugh, A. Martinoli, and F. Matía. *A Distributed Scalable Approach to Formation Control in Multi-robot Systems*, pages 203–214. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009.

[6] J. Ota. Multi-agent robot systems as distributed autonomous systems. *Advanced engineering informatics*, 20(1):59–70, 2006.

[7] C. Pinciroli, V. Trianni, R. O'Grady, G. Pini, A. Brutschy, M. Brambilla, N. Mathews, E. Ferrante, G. Di Caro, F. Ducatelle, et al. Argos: a modular, multi-engine simulator for heterogeneous swarm robotics. In *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5027–5034. IEEE, 2011.

[8] J. H. Reif and H. Wang. Social potential fields: A distributed behavioral control for autonomous robots. *Robotics and Autonomous Systems*, 27(3):171–194, 1999.

[9] K. Sugihara and I. Suzuki. Distributed motion coordination of multiple mobile robots. In *Intelligent Control, 1990. Proceedings., 5th IEEE International Symposium on*, pages 138–143. IEEE, 1990.

[10] K. Xu and M. Gerla. A heterogeneous routing protocol based on a new stable clustering scheme. In *MILCOM 2002. Proceedings*, volume 2, pages 838–843. IEEE, 2002.

Appendix: how to run the experiments

To run the experiments:

- first you need to download and install ARGoS simulator following this link: `http://www.argos-sim.info/core.php`
- get the source code from `https://github.com/tonygi/Distributed_Systems_exam`
- using a terminal session, inside an experiment directory, type the following command:

  ```
  argos3 -c arena.argos
  ```

- once the simulator is running, in the *Lua editor* window, click on *Open a file* then select the Lua script inside the same directory
- click on *Execute code* and then on *Play experiment* button in the simulator window