Swarm Robotics Project : Report

Thomas Wisniewski

ULB

Abstract. In real-world swarm robotics applications, exploration is an important behavior. During the course, we implemented the simplest exploration strategy : diffusion. It works well for simple environment, but it is not enough for complex structure. We are going to consider a solution linked with robot swarms : chaining. The solution is structured as a state machine, with different behaviors for each state. Interactions between robots and with the environment are modeled with the Lennard-Jones potential. Results show chains, but there is place to improvements.

Table of Contents

Introduction

To simulate swarms robots, we use ARGoS, which is a multi-physics robot simulator. The robots have access to several sensors to detect their environment and the others robots. To enable robots to act like a swarm, we program the behavior only on one robot. All robots execute the same controller code. Emerging a collective behavior needs use of all tools available and local interactions analysis.

First, we will describe the problem, the environment and the several tools available. Then an explanation of the different behaviors and state design and finally some analysis on the results and some ways to improve the collective behavior.

1 Description of the problem

1.1 Goal

We have to implement a chaining strategy for the following environment :

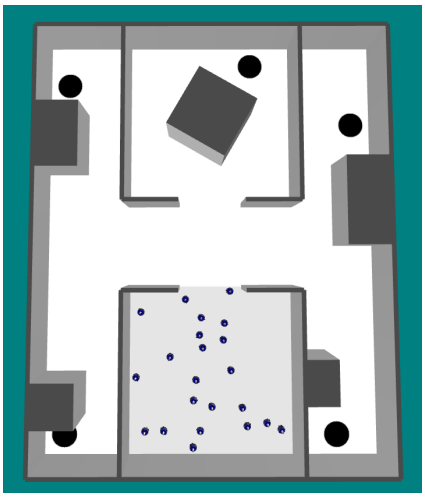

Fig. 1. Initial state of the environment

The environment loosely mimicks an indoor structure, in which five target locations are marked with black dots. The robots are

initially placed in an area called the nest. The goal is to form five chains that connect the nest with the target locations. Here is what we got for example as final state with the behavior implemented :

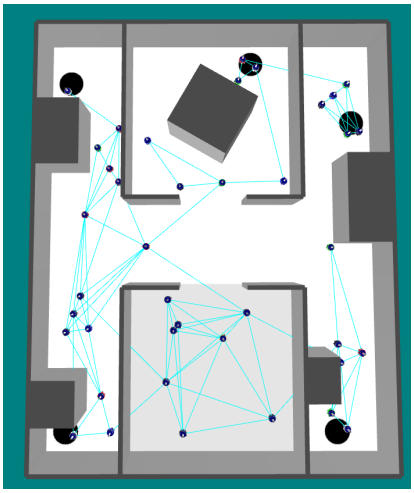

Fig. 2. Final state of the environment with a range of 300cm and 40 robots

As we can see, there is chains but the solution is not perfect. This will be discussed in details on the last section.

1.2 Technical informations

To help us implement the behavior, we have several tools available. Here is a description to understand how they works and utilisation of them on the behavior. One of the most useful tool is the range and bearing sensor. It can receive and send data and detect the position of the sender. They can exchange 10 bytes of data. The robots can exchange data only if they are in direct line of sight, if there is an obstacle between the two robots they can't communicate. We can change the range of this sensor, and we will see that this parameter affects the behavior performance. We use it to create chains by detecting other robots states and position in the chain.

The distance scanner has fours sensors : two long range and two short range. Each of them returns up to 6 values at each time step. These values are the distance and the angle of the obstacles sensed

by the sensors. The range is fixed (150 cm) but we have to set a angular rotation. We will set the maximum value to let robots have the best understand of their surrounding. We use it to detect short and long obstacles. It could be robots or walls. Furthermore we use it to attract robots to free zones, which is a zone where nothing is detected. It creates a behavior which robots go where no chain is formed.

The ground sensor has 8 sensors around the robot and detects the ground's color. It is useful for detecting zone changes. The target zone is black and the nest is grey.

Finally, we can set a color to eight leds, but it is only for visual purpose.

2 Description of the solution

The most important coding aspect of the solution is it is structured as a state machine. It means we can decouple the collective behavior, which is complex to implement into several behaviors, each one has his own reactions with the environment and others states. The complex problem is decoupled into simpler instances, whose solution is easier to develop.

2.1 State machine

The design of the states is inspired by the work of Shervin NOUYAN, in his thesis « Path Formation and Goal Search in Swarm Robotics ».

We distinguish 4 states : `chain_member`, `explorer`, `target` and `nest`. Here is a graphic representation of the states and their transitions. A circle represents a state and an arc in combination with a condition represents a switch from one state to another. Transitions are triggered by local perception and probabilistic events. Also, wait time between transition is added.

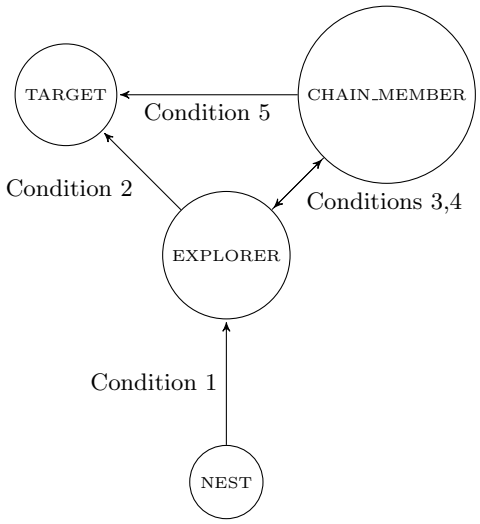

Fig. 3. State machine diagram

Nest When the experiment begins, all robots are on the nest. They are randomly placed across the nest characterized by the grey color's ground. They don't move and are easily recognized by their yellow leds. A simple threshold function models a nest stigmergy. The formula is :

$$p = \frac{n_{nest}^2}{4n_{robots}^2 + n_{nest}^2}$$

n_{nest} is the number of nest robots detected and n_{robots} is the number of robots in the experience. Then we draw a random number from the uniform law and check if this probability is lesser or equal to p (Monte-Carlo method), allowing the robot to become an explorer (Condition 1).

Explorer When the robot is on explorer state, it will explore the environment searching for a chain. Obviously, if the explorer founds a target spot, the transition to the target state is triggered (Condition 2).

Explorers have fundamental interactions with chain members. If the explorer is far enough from the nest and no chain or only one

chain member is detected, the explorer activates the transition to chain member. It will creates a new chain or aggregate to the one detected (Condition 3). A function will compute his position on the chain.

If the explorer detects two or more chain members, it means the robot is not on the chain's end, thus it will follow it. Based on the pattern of the two chain members detected (see next subsection), the robot will navigates to the next chain member. The distance between an explorer and a chain member is d_{chain}.

Transitions are controlled by the same way of the nest state, meaning with a threshold function :

$$p = 1 - \frac{n_{chain\_m}^2}{1 + n_{chain\_m}^2}$$

n_{chain_m} is the number of chain members detected.

Chain member In order to give a direction to the chain and enables robots to navigate among it, the robots in a chain are ordered in a periodic sequence of the three colours blue, green and red, as demonstrated in the following figure :

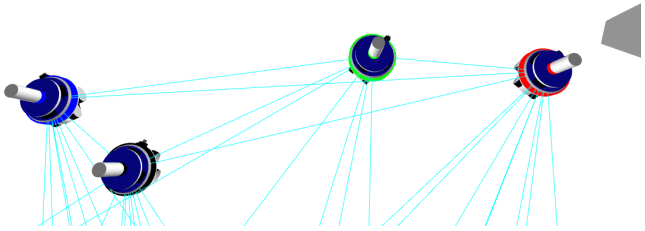

Fig. 4. Chain with 3 members, and one explorer

If the chain member founds himself on the nest by any mean, it will transition to the explorer state (Condition 4). Same way for the target transition (Condition 5). If the robot detects no explorer and no chain members, it means the robot is close to a target spot. Because of the nest behavior, a lot of explorers are sent on the same

time, and this condition will not trigger early. But the condition depends **a lot** of the number of robots.

Furthermore, a behavior will adjust the distance between chain members. Only the last member of the chain moves and adjust his distance with his predecessor to be the closest to the chain distance, which is 150 cm. Thanks to Lennard-Jones potential, a vector is computed and the robot will adjust his angle and wheels speed accordingly to it.

Finally, a merge behavior was implemented to prevent loop chains. Indeed, during tests, the adjust distance behavior implied that different chains interacted between them, creating loop. To prevent this, if two same chain members detect each other, they are attracted (Lennard-Jones function) and when the distance between reach reach a certain threshold d_{merge}, they both transition to the explorer state.

Target The final state. Triggered when a robot reach a black spot, but if a robot is already on the spot, the robot will not stop on the target spot, to prevent an agglutination of robots on the spot.

The structure of the solution was to implement behaviors only with functions that are not dealing with sensors information. In this way, we can call the states behavior are implemented on a high-level programming way. Range and bearing functions allow to detect what is going on and Lennard-Jones functions to act. Now we will explain how for example a robot can know how many chain members he detects or how to compute a vector to avoid obstacles. We can call them low-level programming functions.

2.2 Range and bearing functions

To create directed chains, we need to give robots a mean to detect chain members and their position. Each step, a robot emit his state on the first byte and his color on the second. This way a robot can loop through all messages he received and detect local behavior, and act according to this. Also nest and target detection functions are added here.

One of the most important function is the one that return the number of robots detected in the state sent in parameter, because it is used on almost every behavior and range and bearing functions. Others functions are all about detection of chain members (previous and next), same chain member and returning their information.

2.3 Lennard-Jones functions

To adjust robots' position with their environment, we use the Lennard-Jones potential. We imagine the robots immersed in a virtual potential field. The potential field is calculated through the distance scanner. The goal is to return a vector and computes the angle and speed for the wheels from it.

The computation of a process goes this way : We first compute a vector with a certain behavior : for example avoid short obstacles, long obstacles, etc and each process we talk above (merge, go to a robot,adjust distance and exploration) will combine these vectors pondered by factors. We can this way easily create new behaviors. Lennard-Jones potential computation parameters for some behaviors are taken from the work of an ancient student because they were far better than the initials ones. The computation from the vector to make the wheels move is the same as explained in TP.

3 Results

For each experimental setup, 30 repetitions were made. A bash script `data.bash` allows automatic execution and get data for each setup. Two metrics are used to characterize our solutions : Average and standard deviation time to reach all the locations (`mean_steps` and `sd_steps`) and average and standard deviation number of robots involved in the chains (`mean_chain` and `sd_chain`).

3.1 Range and bearing sensor range

We run several setups with the same number of robots (30) and we change the range each time.

range	mean_steps	sd_steps	mean_chain	sd_chain
300	3832.700	1554.882	20.00000	1.701926
250	4566.500	2383.768	21.06667	1.700575
200	5178.867	1788.659	22.33333	1.953482
175	5973.867	2836.644	22.43333	1.715715
150	6805	2742.938	22.76667	1.590561

We can see that the number of steps to reach all target locations increases with the decreasing of the range. Also, the standard deviation increases too. The number of necessary robots increases too when the range decreases.

When the sensor's range if low, robots can not detect others robots good as with a larger range. Because of that, chains take more time to be created and explorers take more time to detect chains.

3.2 Number of robots in the experiment

We run several setups with the same range (300cm) and we change the number of robots.

number of robots	mean_steps	sd_steps	mean_chain	sd_chain
40	2275.933	549.2995	23.50000	2.193250
30	3832.700	1554.8821	20.00000	1.701926
25	3655.633	1538.6646	18.40000	1.428768
22	4670.300	2239.9950	17.03333	1.299425

When the number of robots is decreasing, the time to complete the task increases. because of the distance between chain members which doesn't change. Actually, if we are looking at the chains for example with 20 robots we got this :

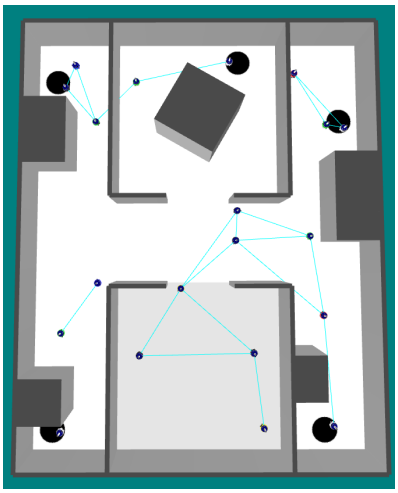

Fig. 5. Final state with 20 robots and 300cm for range

If we compare it to Figure 2, we clearly see that chains pattern is worse. This is because of the chain member behavior to explore the environment if they detect nothing. When the number of robots is low, there is a much higher probability to detect nothing than a state with 40 robots. Also, this behavior was added to ensure all states to be feasible. We can clearly see that chains are not perfect (for example some don't connect with the nest). Furthermore, during tests, robots could sometimes detect another robot even know there was an obstacle between them, causing them to messing up. A solution was to adapt the camera range each step, but the code was already implemented and it would create more trouble than good.

4 Conclusion

In conclusion, this project made me apply everything I learned during the TP. The implementation was very hard, this is the first time I made a project of this big. The solutions are not perfect, and some improvements can be done on the chain member behavior.

5 References

State machine design based on this thesis http://www.swarm-bots.org/dllink.php@id=448&type=documents

Transform Lennard-Jones vector into wheels movement http://iridia.ulb.ac.be/~gpodevijn/h-414/pattern_formation.lua

Lennard-Jones coefficients of some behaviors https://github.com/feidens/SwarmRoboticsChaining/blob/master/srprojectlow.lua#L709