RoSinA project report

Robotic Swarm in ARGoS

`daniele.schiavi@studio.unibo.it`
`marco.canducci@studio.unibo.it`

A.Y. 2018/2019

Contents

1 Motivation and project goal **2**

2 Design **2**
2.1 State diagram . . . 2
2.2 Motor schemas . . . 4
2.2.1 Stay still . . . 4
2.2.2 Move randomly . . . 4
2.2.3 Go straight . . . 4
2.2.4 Avoid collisions . . . 4
2.2.5 Move perpendicular . . . 4
2.2.6 Follow chain direction . . . 5
2.2.7 Align . . . 6
2.2.8 Adjust distance . . . 6
2.3 Miscellaneous design choices and considerations . . . 7

3 Testing and results **8**

4 Further developments **10**

Bibliography **11**

1 Motivation and project goal

The motivation behind this project was to investigate the increasingly relevant topic of swarm robotics. To do this, we chose a well-known case study [2] concerning robotic chain formations between a starting station (*nest*) and an arrival station (*prey*).

Our aim is to reproduce the experiment presented in the paper inside the ARGoS environment [3], adapting the proposed solution to the features of the chosen simulator.

2 Design

The controller is based on a behaviour-based architecture, consisting of 6 different states. Each state corresponds to a different behaviour. A behaviour is realized following the motor schema paradigm.

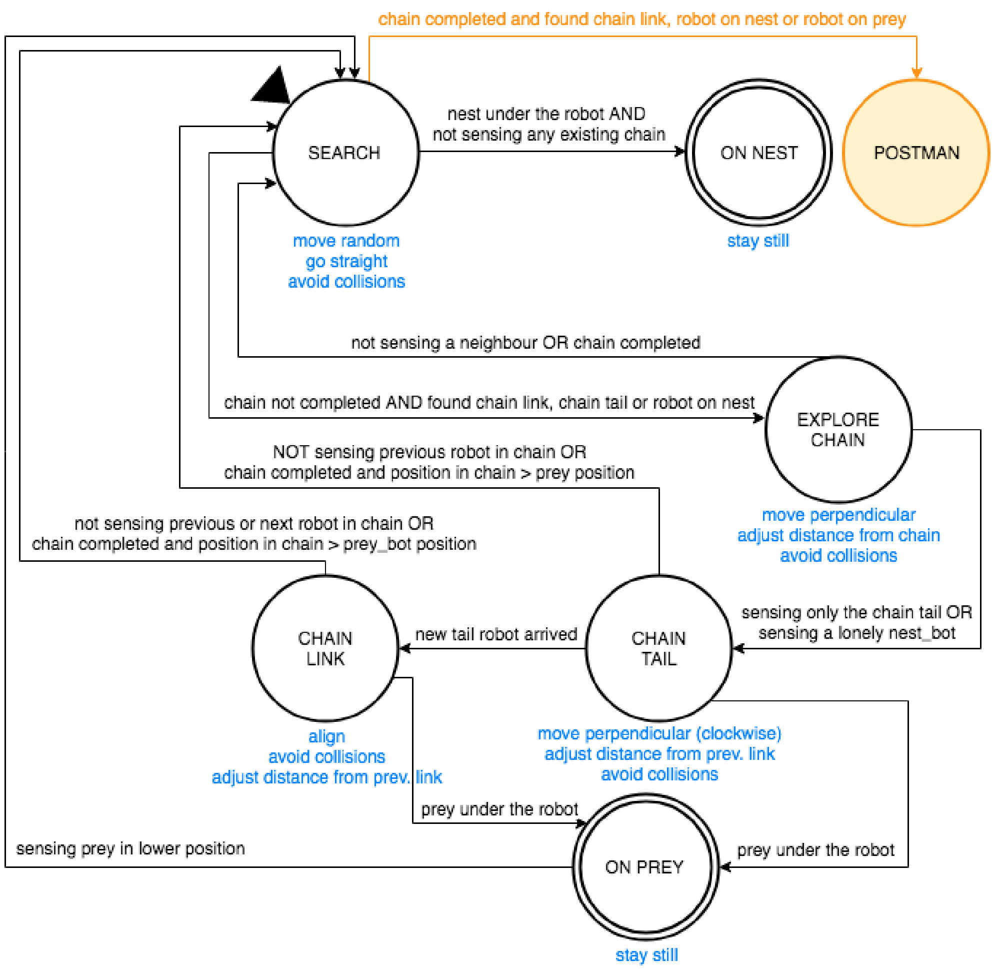
At each time step only one behaviour is considered; for each behavior all of its motor schemas are active in parallel. Each motor schema outputs a vector denoting the desired direction and intensity of motion. The sum of the active motor schemas is then translated into motor activation.

2.1 State diagram

The robots are initially located at random position and orientation in a SEARCH state. While the complete state diagram is presented in Figure 1, we try to briefly describe what condition we wanted to represent with each state and, therefore, the desired behaviour:

- ➢ A bot SEARCHes for a nest or a chain performing a random walk until it perceives one of them.
- ➢ A bot ON NEST has to just stay still while the chain forms and rotates around it.
- ➢ A bot in EXPLORE CHAIN state has to circumnavigate the chain moving towards the tail with the purpose to become the new tail.
- ➢ A CHAIN TAIL bot needs to keep a certain distance from the previous link bot while moving perpendicularly from it in order to trigger chain rotation.
- ➢ CHAIN LINKs are the bots forming the chain portion between the nest and the tail; every time a new tail is added to the chain the previous tail becomes a link. Every chain link must align and keep a constant distance from the previous and the next bots belonging to the chain.
- ➢ A bot ON PREY has to stay still and inform the other members that the chain is completed.
- ➢ POSTMAN bots will circumnavigate the chain once it is completed in order to retrieve food.

Note: if a robot in the CHAIN LINK state perceives the prey it must change its state to ON PREY and, consequently, all the bots in following positions inside the chain must leave it, changing their state to SEARCH.



chain completed and found chain link, robot on nest or robot on prey
SEARCH
nest under the robot AND not sensing any existing chain
ON NEST
POSTMAN
move random
go straight
avoid collisions
stay still
not sensing a neighbour OR chain completed
chain not completed AND found chain link, chain tail or robot on nest
EXPLORE CHAIN
NOT sensing previous robot in chain OR chain completed and position in chain > prey position
not sensing previous or next robot in chain OR chain completed and position in chain > prey_bot position
move perpendicular
adjust distance from chain
avoid collisions
CHAIN LINK
new tail robot arrived
CHAIN TAIL
sensing only the chain tail OR sensing a lonely nest_bot
align
avoid collisions
adjust distance from prev. link
move perpendicular (clockwise)
adjust distance from prev. link
avoid collisions
prey under the robot
sensing prey in lower position
ON PREY
prey under the robot
stay still

Figure 1: The controller's state diagram. Each circle represent a state (i.e. a behaviour). Arrows are labelled with the conditions that trigger a state transition. For each state the corresponding active motor schemas are presented in blue under the circle. The starting state is SEARCH, while ON NEST and ON PREY are intended as arrival states. Orange components show a possible extension of the project; refer to the *future developments* section for more information.

2.2 Motor schemas

2.2.1 Stay still

A motor schema thought for footbots that have reached a *final state* such as ON NEST or ON PREY. In these cases the desired behavior is to just remain stationary, therefore the motor vector will be null.

2.2.2 Move randomly

Motor schema which introduces a random component to the robots' movements; it's designed for the SEARCH state, in which footbots do not have a target yet and only need to look for a nest or a chain. The motor vector has a random module between 0 and 1 and a random direction between -π and +π.

2.2.3 Go straight

This simple behavior was designed to avoid the Brownian motion given by the solely presence of the random component. It consists of an unitary vector that will lead the robot to go straight; it always accompanies the *Move randomly* motor schema.

2.2.4 Avoid collisions

This motor schema is used to avoid collisions between robots or other objects like obstacles or walls. Two possible implementations have been devised: the former considering only the proximity sensor with the maximum value; the latter considering the sum of the proximity sensors exceeding a certain threshold. The resulting force's module will be inversely proportional to the distance and with a direction opposite to the sensor (in the first solution) or to the sum vector (in the second solution).

2.2.5 Move perpendicular

Consists of a constant vector perpendicular to the straight line passing through the footbot and another footbot within its range of sensing. This motor schema is useful in two different situations:

- a bot in EXPLORE CHAIN state that senses one (and only one) footbot nearby belonging to the chain;
- applied to CHAIN TAIL bot with respect to the previous link in order to enable the chain rotation around the nest - all the chain links will follow accordingly thanks to their *Align* motor schema.

In the first case the rotation is made following the most natural direction (if the sensed robot is on the left it's convenient to rotate counterclockwise, otherwise clockwise). In the second case rotation is always in the same direction because we want the chain to rotate always clockwise (or always counterclockwise) in order to cover a greater surface in the same amount of time.

2.2.6 Follow chain direction

In the EXPLORE CHAIN state the footbots must circumnavigate the chain until they reach the tail. The *Follow chain direction* motor schema enables this behaviour by providing a vector oriented towards the chain expanding direction. To do so we took advantage of the position number that every chain bot transmit to its neighbours through the range-and-bearing actuator. To apply this motor schema it's necessary to sense at least two chain bots because the resulting vector (C) will be evaluated by subtracting the directional vector of the bot with the highest position number (A) from the directional vector of the previous chain bot (B).

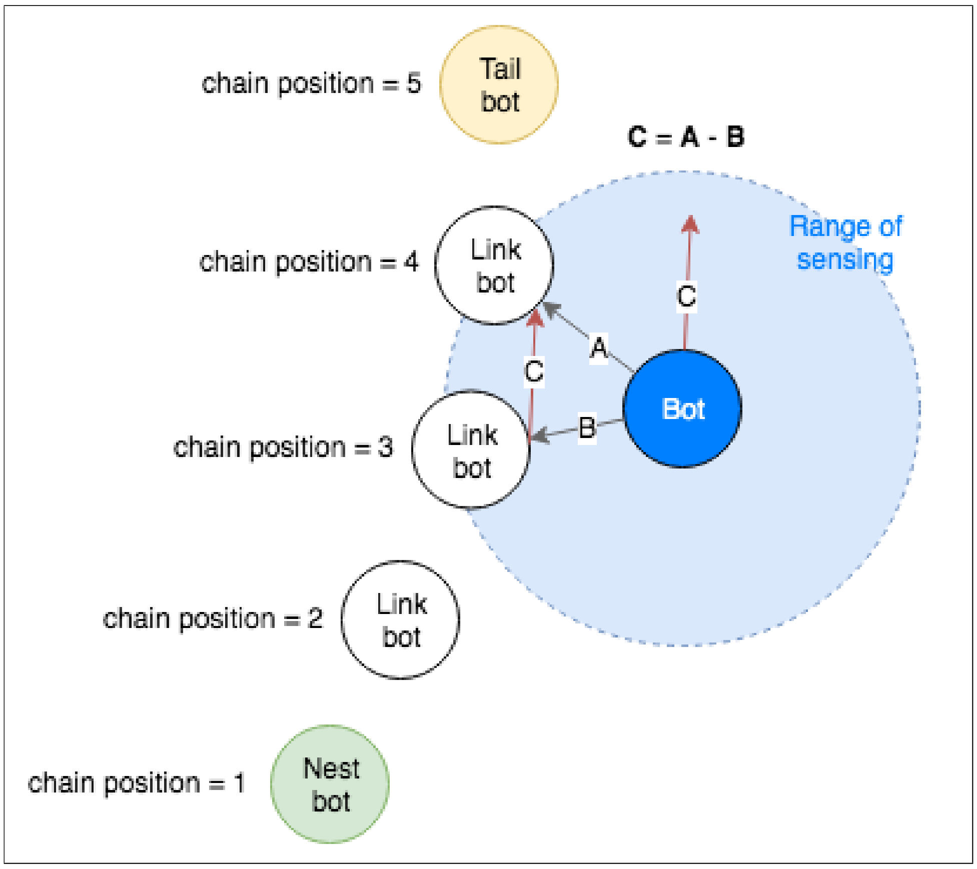


Figure 2: *Follow chain direction* motor schema. The blue bot is the one following the chain direction; the Nest bot, Link bots and Tail bot form the chain and everyone is enumerated from the Nest bot through the Tail.

2.2.7 Align

To guarantee a straight chain while rotating around the nest it's useful that each link aligns itself with the previous and the next bots. Doing so makes it possible to just turn the tail, the rest of the chain will follow. Two constant vectors directed towards the previous and the following chain bots are added together in order to obtain the desired motor vector.

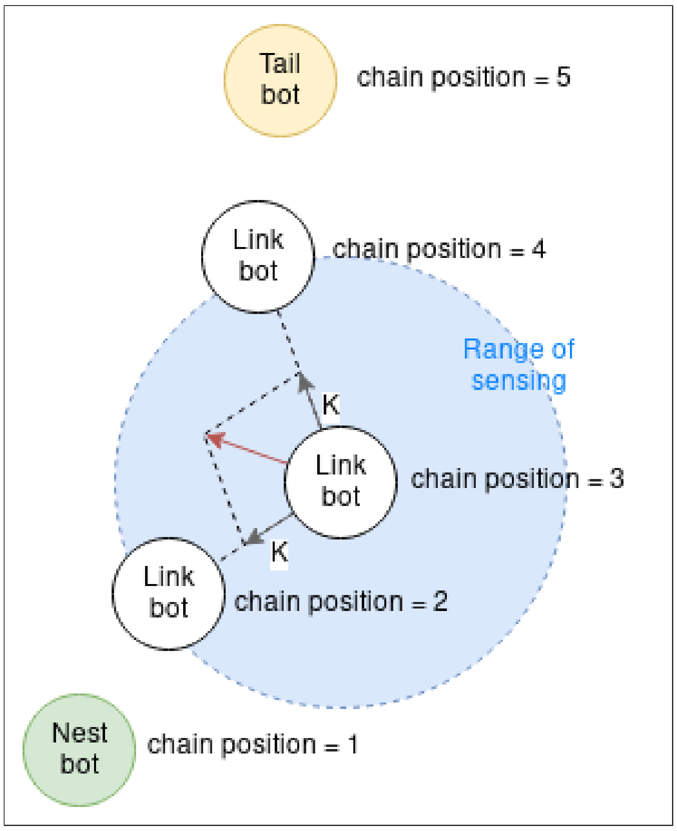


Figure 3: The *align* motor schema.

2.2.8 Adjust distance

Motor schema that aims to keep one footbot at a fixed distance from another. This enables (1) the chain bots to keep a constant distance from adjacent chain bots and (2) the robots in EXPLORE CHAIN state to keep distance from the chain. The desired distance is designed in order to avoid clashes and to maximize the covered area. The motor vector will be of attractive nature when the distance between the footbots is greater than the desired one, null if the distances are the same, repulsive otherwise. It's useful to add a tolerance to the desired distance in order to avoid oscillating behavior.

2.3 Miscellaneous design choices and considerations

- We decided to put a minimum threshold for the proximity sensor in order to avoid false activations due to noise.
- We have reduced the original RaB's range of sensing to impose a localized perception with the aim of enforcing locality of sensing, in style with swarm robotics programming [1].
- The bot ON NEST will transmit its chain position - always equal to 1 - through its range-and-bearing actuator. Whenever a new bot in EXPLORING CHAIN state will become the new tail it will transmit a chain position equals to the old tail incremented by one; note that if the chain is composed just by the nest it's considered as the degenerate case where the nest is also the tail.
- Contrary to the original paper's design, we decided to separate the CHAIN TAIL state from the CHAIN LINK state depending on whether the bot is in the last position of the chain or not. We think it's more clear this way since the two behaviours must be different.
- Each motor schema's vector module has been chosen based on the results of several manual tests.
- We have chosen the distance between chain links with the aim of obtaining a good compromise between exploration radius and ease of chain circumnavigability.
- To ensure that the new tail bot would position itself aligned with the rest of the chain we triggered the state transition from EXPLORE CHAIN to CHAIN TAIL when the last chain link becomes undetectable by the RaB sensor because hidden behind the tail.
- We based the *adjust distance* motor schema on the RaB sensor instead of the proximity sensor because it's always necessary to keep a certain distance between robots that are forming or exploring the chain.
- To speed up the searching process it's convenient to have a chain that rotates as quickly as possible. To get a high rotational speed without the chain crumpling on itself we had to introduce an optimization to the *align* motor schema, thanks to which we were able to double the chain's initial speed. The optimization consists in making the strength of the alignment of the chain links proportional to their positional index: this is necessary because the more the bot is away from the nest and the more it will have to rotate quickly to remain aligned.
- To achieve a better chain rotational behaviour it would ideal that the tangential force applied to the tail was perpendicular to the straight line passing through the

tail and the nest. Though, due to the limited perception of the robots, the force is designed to be perpendicular to the line passing through the tail and its closest link. We believe this is a good approximation when the chain is “adequately” straight. An alternative solution would have been to let every chain link transmit the angle between its previous and its next bot in order to evaluate the angle of the bot positioned on the nest as the sum of all the deltas. We decided to discard this solution because it adds complexity while it doesn’t guarantee to bring improvements to the current implementation:

- ➢ it’s cumbersome to transmit angle values through the RaB’s sensors because transmitted data must be a number in the range [0,255];
- ➢ the noise levels would add up and propagate through the links, worsening the accuracy as the chain lengthens.

3 Testing and results

To verify the effectiveness of our work we carried out 60 tests divided in blocks of 10 tries per different condition. “Different conditions” means a different swarm size and presence/absence of noise levels. In every test block we verified the number of steps used to form a completed chain from the nest to one of the two preys. The different tests carried out are, in detail with:

- ➢ 15 footbots without any kind of noise;
- ➢ 15 footbots with 0.3 noise level on every sensor and actuator;
- ➢ 30 footbots without any kind of noise;
- ➢ 30 footbots with 0.3 noise level on every sensor and actuator;
- ➢ 50 footbots without any kind of noise;
- ➢ 50 footbots with 0.3 noise level on every sensor and actuator.

Note: we’ve chosen a noise value of 0.3 because it seemed the sweet spot between introducing some level of uncertainty and a satisfying swarm behaviour.

We decided to stop at 50 robots for two main reasons:

- ➢ the chain started to collide with the walls while rotating due to the small size of the arena;
- ➢ too many concurrency problems arose with bigger swarm size.

As expected, the number of steps necessary for the chain completion decreases considerably with the increase in the number of robots. Interestingly, noise levels seem to decrease overall performance only when there are many robots and it does never prevent the swarm to achieve its goal in a reasonable time. The last thing we want to point out regards the variance of the test results per block: it’s much higher with 15 footbots probably due to the greater impact that the luck factor has on smaller swarms.

Tries	15_bots	15_bots_noise	30_bots	30_bots_noise	50_bots	50_bots_noise
1st	3034	1797	1911	1664	1796	1124
2nd	3042	2430	1663	1791	678	1666
3rd	4536	3261	1066	3207	1096	1485
4th	6400	3114	2977	3275	810	567
5th	3049	4205	1438	1958	1379	978
6th	4935	5221	1583	1750	763	1384
7th	947	3643	1794	1131	1469	1861
8th	3450	3106	1611	2971	677	1159
9th	2390	2701	1602	3220	1558	841
10th	2701	3520	1112	2655	1120	1841
μ	3448.4	3299.8	1675.7	2362.2	1134.6	1290.6
σ	1510.6	950.1	529.9	789.3	401.6	432.6

Table 1: Test results - steps required to form a completed chain

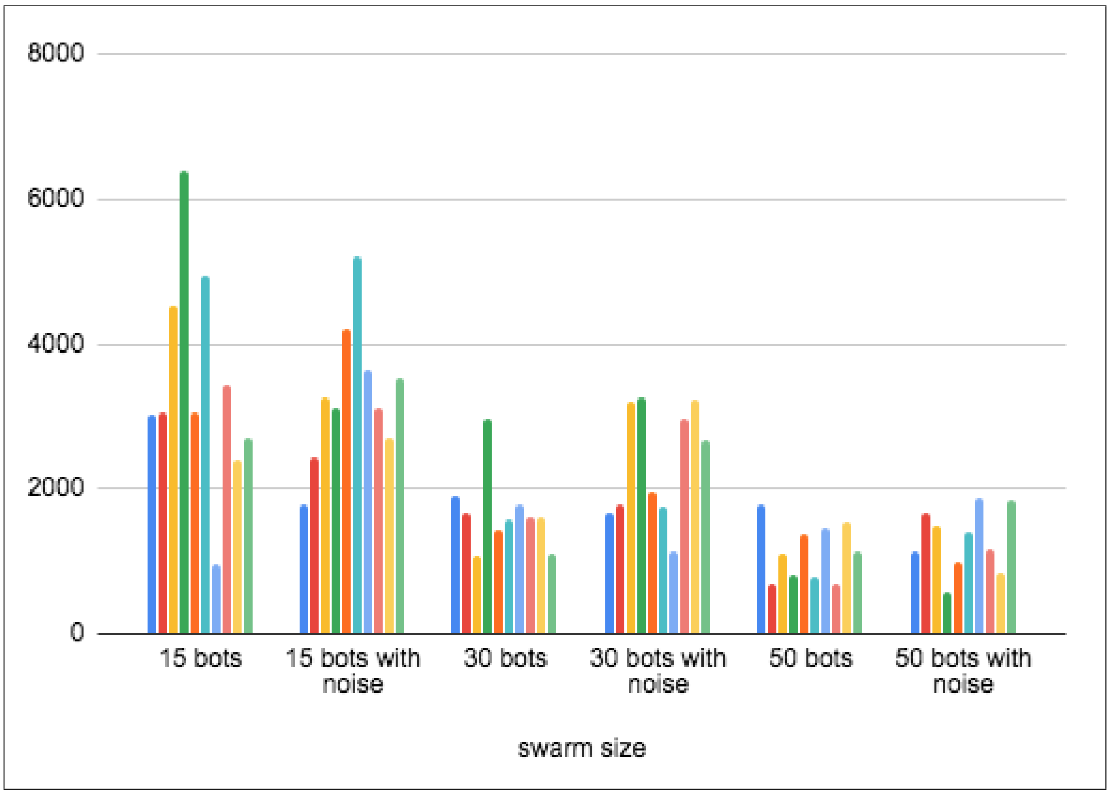


Figure 4: Comparative chart related to the data reported in Table 1

4 Further developments

There are some solutions and enhancements that haven't been implemented due to time constraints; we'd like to at least mention them and give the reader some ideas for possible future developments.

- The experiment ends with the creation of the complete chain (from nest to prey). The purpose of these chain formations is to mark the path to be traveled by other agents, so we could think to introduce different robots or even exploit the footbots still wandering in the arena for the food retrieval task, see the POSTMAN state in Figure 1.
- We have tried to start chain expansions from the nest and from the preys simultaneously in order to accelerate the chain completion process, unfortunately though this added too much complexity (for the time invested on the project) due to the presence of two rotating chains that should have merged together in a clean fashion. Nonetheless, this could be a cool feature to add in the future.
- Unfortunately, with the presence of too many robots inside the arena, two tails can form at the same time ending up forking the chain in two and resulting in unexpected behaviours and malfunctions. This concurrency problem could be solved by letting every bot broadcast its identifier and enforcing a protocol describing a priority rule over the identifiers. Eg: if a bot in CHAIN TAIL status senses another tail nearby it must leave the chain if its identifier is higher than its neighbour's.
- Finally, it would be a great exercise trying to introduce obstacles in the arena and designing feasible solutions to overcome all the new problems that will inevitably arise.

Bibliography

[1] M. Dorigo, M. Birattari, and M. Brambilla. “Swarm robotics”. In: *Scholarpedia* 9.1 (2014). revision #138643, p. 1463. DOI: `10.4249/scholarpedia.1463`.

[2] Shervin Nouyan, Alexandre Campo, and Marco Dorigo. “Path formation in a robot swarm”. In: *Swarm Intelligence* 2.1 (Mar. 2008), pp. 1–23. ISSN: 1935-3820. DOI: `10.1007/s11721-007-0009-6`. URL: `https://doi.org/10.1007/s11721-007-0009-6`.

[3] Carlo Pinciroli et al. “ARGoS: a Modular, Parallel, Multi-Engine Simulator for Multi-Robot Systems”. In: *Swarm Intelligence* 6.4 (2012), pp. 271–295.