Distributed Intelligent System Project

Mathilde Bensimhon, Pauline Maury Laribire, Daniel Almeida Dias, Hugo Grall Lucas
Distributed Intelligent Systems Course ENG-466, EPFL, Switzerland

***Abstract*— This report introduces a controller architecture which address the problem of distributed robot navigation based on low range local communication. The robots must form a flock while avoiding obstacles in simulation and real life. Four controllers are implemented together to perform this behaviour: Braitenberg for obstacle avoidance, Reynolds for the flock formation, Migration to go in a specific direction and Join to help re-create the flock. Their importance is set depending on the robot status. A grid search has been implemented to optimize the parameters based on metrics of relative orientation, cohesion and velocity. Finaly, a set of weights has been found for solving the two Webots scenarios. The final implementation on a set of six real e-pucks suffers from the lack of robust communication and of time for retuning the controller gains. As result, sometimes the targeted flock appears but it is quite unstable.**

I. INTRODUCTION

The goal of the project "Multi-robot navigation in cluttered and dynamic environments" is to implement a multi-robot navigation strategy using a collective movement approach. The robots are e-puck and they have to maximize a combination of 3 metrics in 2 different simulation scenarios and in a real life experiment. To optimize those metrics, the robots should have the same orientation, have close intra-group distance while going as fast as possible. There is a first scenario *obstacle avoidance* where a group of e-pucks must navigate around obstacles before re-grouping and a second scenario *crossing* where two groups of e-pucks must be able to cross each others. The controllers must be implemented both in simulation on Webots software [1] and on real e-pucks [2]. The major limitation is that no global positioning is allowed, as robots can only communicate locally with the use of infra-red sensors.

This report describes the experiments, results and conclusion of the project. The main intuitions and methods were taken from the Master's degree course "Distributed Intelligent Systems" given by A. Martinoli at EPFL [3], and especially of the laboratory 4 of the course.

II. EXPERIMENTS

A. Controllers

1) Overall controller functioning:

The controller of the robots divides the computation of the speed in four sections as can be seen in figure 1: an obstacle avoidance based on Braitenberg, a flocking behaviour regulated by Reynolds rules, a migration urge to lead in a direction and a 'Join' behaviour to help robots re-unit when straying off. Each section computes the speed of the right and left wheel speed independently. The speed applied on the wheels of the robot is then the weighted sum of each of these sections:

$$v_{wheel} = w_b * v_{Brait} + w_r * v_{Reyn} + w_m * v_{Migr} + w_j * v_{Join}$$

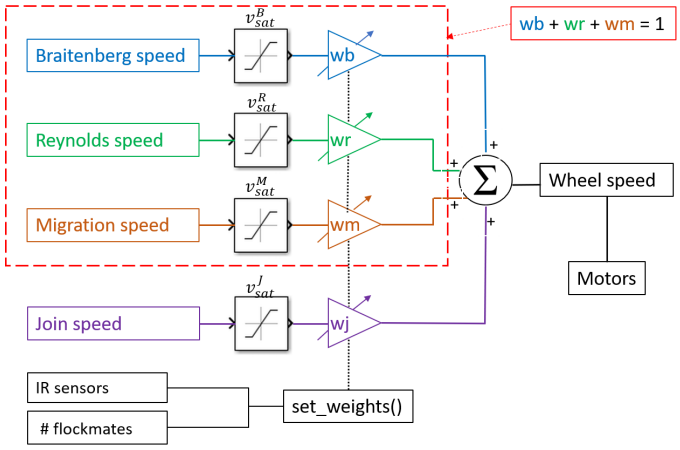

Fig. 1. E-puck controller

2) Individual controller speeds:

a) Braitenberg speed:

For the obstacle avoidance, a Braitenberg controller is implemented. The weight matrix associated to the sensors have been tuned to avoid obstacles with a bias on the left side. Indeed, if all the robots turn in the same direction, there is less chance to lose the flock formation.

$$v_{Brait} = \sum_{j=1}^{m} w_{ij}^{Brait} * I_j$$

b) Reynolds speed:

For the flock formation, the inspiration comes from the paper [4]. To keep the flock formation, the e-puck first computes the relative average location and speed of its neighbours (the robots that are in a 25cm range of him, as this is a hardware constraint). At each time step, each robot sends a message containing its id, which enables the other robots to measure their relative range and bearing to the sender, as well as its id. The range and bearing are measured directly by the message strength and direction. Then, it implements a weighted sum of the three Reynolds rules:

- Rule 1: Cohesion, the robot try to go towards the center of mass of its neighborhood. v_{cohes} is set at the relative average position of the e-puck's flockmates.
- Rule 2: Dispersion, the robots respects a minimum relative distance to its neighbours. v_{disp} has been set as

the sum of the negative relative position (instead of the inverse of the formula seen in the course) because using the inverse tends to make values explode and easily leads to saturation. Thus our controller implements the dispersion more smoothly.

- Rule 3: Consistency: the robot tries to match the speed of its neighbours. $v_{consist}$ is set at the relative average speed of the e-puck's neighbours. The velocity of the other robots is computed based on their previous and present position in the global frame[1]. It is not in the local frame of the robot as the robot is also moving between the time steps.

The 'Reynold speed' v_{Reyn} is computed for each wheel individually. Each rule has a specific weight, tuned as explained in the section II-C and they are combined in the weighted sum:

$$v_{Reyn} = w_1 * v_{cohes} + w_2 * v_{disp} + w_3 * v_{consist}$$

The norm of the resulting v_{Reyn} is then normalized to 1. This enables the values due to the Reynold speed to always be as high as possible when the wheel speed is computed.

c) Migration speed:

The robots are also subject to a migration urge which is provided as a vector in the original reference frame defined at the startup instant. A function translates this urge to a speed v_{Migr} for the current pose. To avoid losing the information due to saturation, the values are scaled back such that the change in orientation is preserved even in case of saturation.

$$v_{Migr} = R_\theta * vect_{migr}$$

with R_θ being the matrix of rotation between the referential of the robot and the migration direction.

As for the Reynold speed, the norm of the resulting v_{Migr} is then normalized to 1, wich enables the following wheel speed computation to always saturate.

d) Join speed:

It has been observed that at times the presence of an obstacle can split a flock in two despite the Braitenberg avoidance having a bias in its coefficients. When these situations arise and the robot loses all its flockmates, the nature of the proposed algorithm gives it a boost (by doing only migration) with the hope of catching up. This method however only works suitably if the lost teammates are ahead, if not, the boost can further isolate them. The Join speed approach is based on the idea that under ideal conditions the rest of the flock will likely be in the axis determined by the migration vector, hence this additional factor tends to bring back any strays inward perpendicularly. In the simple scenarios provided, this can be achieved with a proportional controller. A caveat of this approach is if the robot finds itself perpendicular to the axis, in fact the robot could loop backwards to rejoin, for this reason, the Join speed is affected only if it's current pose is in a $]-\pi/2, \pi/2[$ section facing the migration vector. Both the offset from the main axis and the pose is obtained by odometry. Finding a suitable coefficient for the controller is tricky and oscillations aren't unusual. However, given the nature of the scenarios and the optimization approach, this behaviour can become an advantage for this type of environment.

$$v_j^{right} = Y_\perp K_j \quad and \quad v_j^{left} = -Y_\perp * K_j$$

3) Weights of each controller speed:

The weight given to each part depends on the environmental context of the robot and are applied before the final wheel speed computation.

Firstly in the initialization each robot is set with a unique parameter α depending on its personal id: $\alpha = 1.5 + (id + 1) * 0.4$.[2] Afterwards, the relative importance of the migration wm and of Reynolds wr is set as $w_r = \alpha w_m$. The goal is to add some heterogeneity in the robots behaviours. Indeed, some robot give more importance to Reynolds weights and hence, help maintain the flock formation while other have strong migration weights, which enables the flock to move very fast forwards.

Secondly, the three first sections, Braitenberg, Reynolds and Migration (framed in red on figure 1) are always bounded to each other with the constraint: $w_b + w_r + w_m = 1$ and their saturation block is always the same for the three $v_{sat}^B = v_{sat}^R = v_{sat}^M$. Hence the maximum value of this part (red frame) is the speed saturation value. The speed coming from the "Join" part is independent:wj has no constraint with the other weights and the saturation speed v_{sat}^J is different (smaller) than the other saturation speed. Intuitively, it can be understood as an 'added' speed to the other speeds from Braitenberg, Reynolds and Migration. The saturation values were chosen to always stay below the maximum velocity of the robots. More details are given at the end of this section. Also, when the values lead to a value above the saturation, they are normalized keeping the difference of speed between the two wheels to avoid losing rotational speed.

To set the weights in the robot controller dynamically two conditions are observed as in figure 2.

- Condition 1: $IR_{max} > \Delta$: is the value of the maximum sensor IR_{max} above a threshold $\Delta = 350$?

If it is the case, the robot speed takes into account the obstacle avoidance with Braitenberg. This condition to be above a threshold is implemented to ignore reacting to sensor noise. Likewise, avoidance is given maximum priority over the other parameters above the threshold $B_{upper} = 2000$ ensuring the robot focuses on it even at the cost of breaking temporarily formation. Then, w_b is set logarithmic proportional and normalized by the constant B_{upper} to the value of the maximum infra-red sensor. If the ratio is above 1, wb is set to 1 anyway to avoid having too strong speed on the motors.The 'Join' weights w_j is also set to zero if there is obstacle avoidance. Indeed, if an obstacle must be avoided then this parameter is not taken in account to not decrease the controller reactivity.

[1]Global frame refers to the reference frame at the startup

[2]This formula was found through trial and errors in the optimization section.

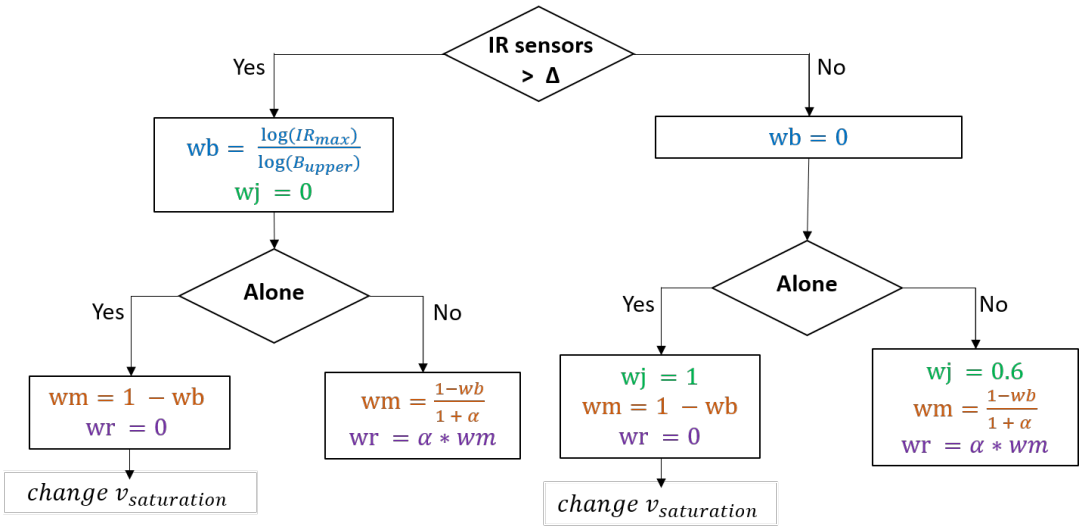

Fig. 2. Weights of each controller

Otherwise, if $IR_{max} < \Delta$, then it is assumed that the values sensed are only due to noise and they are ignored by setting $wb = 0$. In this case, w_j is not immediately set and will be determined in condition 2.

- Condition 2: $Alone$: has the robot detected a flockmates during the last 10 time steps ?

If the robot is not alone, the solution to the equation $w_b + w_r + w_m = 1$ with the constraint $w_r = \alpha w_m$ is applied. The results is the following equations:

$$w_m = \frac{1 - w_b}{1 + \alpha}; \quad and \quad w_r = \alpha w_m;$$

Still in the case where the robot is not alone and if there was no obstacle avoidance, wj is set to 0.6. This value is chosen as a compromise to continue implementing this joining behaviour while not adding to much speed to enable other robot that has 'lost' the flock to catch up.

However, if the robot is alone, its Reynolds weight is ignored ($wr = 0$) and migratory weights are increased to $w_m = 1 - w_b$. This condition stops as soon as the robot has a flockmate. This is particularly useful if the robot was set with a high α but has lost the flock: otherwise it would go very slowly because its migration speed is very small. Indeed, robot with high α have bigger Reynold weights wr than other robots and thus losing the flock has the unwanted effect to reduce their speed, making them unfit to catch up with the flock.

$$w_m = 1 - w_b; \quad and \quad w_r = 0;$$

In the case where the robot is alone and if there was no obstacle avoidance, wj is set to 1 (the maximum possible value of a weight) to add the full joining speed. Indeed, when the robot is alone, the priority is to find and return in the flock formation. Hence, the robot is given more speed range than others to fulfill this task.

Moreover in the case where the robot is alone, the saturation speeds are updated. Usually their values in simulation[3] are set as:

$$v^B_{sat} = v^R_{sat} = v^M_{sat} = 700 \quad and \quad v^J_{sat} = 100$$

However, the goal of the update when the robot is alone is to give more importance to finding the flock again. Hence, the output of other speed is decreased and the output of 'Join' increased:

$$v^B_{sat} = v^R_{sat} = v^M_{sat} = 500 \quad and \quad v^J_{sat} = 300$$

B. Difference for real implementation

The differences of implementation between the simulation on webots and the real experiment are explained in this section.

The same IR sensors are used for sending and receiving messages as well as sensing obstacles. However those sensors are noisy, sensitive to light conditions and can only be used for one purpose at a time. Hence, timers from the `librcom` library in $ircomTools.c$, are used to divide those three tasks of the IR sensor. It was observed that the robots do not receive messages of their neighbours at all time steps, even if one is only in sensing mode and the other other in receiving mode. Messages sometimes even go through other e-pucks as they are semi-transparent. Moreover, if a robot receives messages from different robots at the same time, those messages might have wrong information. To solve those issues, an example was found online *Synchronize example* that also belongs to the `librcom` library and we inspired our communication on it. Thus, at each time step, the robot

1) Send message: its id and group number
2) Listen for message: other robots id and group number

[3]on the real robots all those values were divided by two

It can happen that a robot decodes a message with an id that doesn't exist. Thus, messages are only considered if the robot id and group id is valid. Also, in real experiments, the environment light should be controlled as it has an impact on the communication and sensing.

Moreover, on real e-puck due to these sending and receiving periods, the duration of a step (time for a loop to be performed) is greater than in simulation. Hence, the update of the robot odometry and the relative velocities between robots is computed with the real time step. This time step is computed at each loop and its duration is almost equal to the sending and receiving time. Usually the duration of a step is slightly above 60 [ms] (20 [ms] to send messages and 40 [ms] for reading).

Another limitation of the `librcom` library is that the accuracy of the sensing direction is low: it is given by the position in radians of the 8 IR sensors that received the message. Hence, at best, the accuracy is $\frac{2\pi}{8}[rad]$.

Another difference is of course the command on the robot's wheels. It was advised in the project assignment to move the e-pucks slowly. Hence, the maximum speed were set lower in reality then in simulation (we divided all webots speed by two for real e-puck experiments). The errors in odometry due to wheel slip can not be avoided but might be decreased with those lower speed.

C. Optimization

As described on our choice of implementation, the strength of our strategy is based on a good tuning of our parameters. The Braitenberg weights for obstacle avoidance should allow the robot to be reactive enough when facing an obstacle, without an over-deviation from the initial orientation. The Reynolds weights should give enough flexibility and robustness to our flock in order to be able to split and reunite, with stable distances between robots when the flock is formed. The Reynolds to migration ratio needs to encourage moving the flock toward the direction of migratory urge, without restraining the Reynolds and avoidance controllers.

1) Optimization program: The optimization procedure is based on a meshgrid approach. Having established Reynolds thresholds suitable for the scenarios and imposed limitations, the task of the optimization procedure is to establish the appropriate weights for each Reynold's rule. The resolution of the meshgrid is chosen as ($\mathcal{M}(r_1, r_2, r_3) = Z^1_{[1,10]}$) in order to identify the regions of interest where a secondary meshgrid optimization, if needed, can be implemented. Each simulation is interrupted either if the simulation time exceeds a threshold or if any of the robots reaches a border of the table. This is done to prevent the performance from being polluted by robots falling off.

2) Matlab analysis: For each simulation, Matlab analyzes the mean value and standard deviation of each metric (orientation, cohesion, velocity, overall performance). Then three types of classifications will be done: for the first part, we take the highest mean value for each metric, and the four simulations providing these results are boxplotted next to each other, to compare their performances for the four metrics in the same time. We also take the smallest standard deviation value for each metric, and plot it the same way we did before. For the second classification, we add the mean value of the metrics obtained for each performance and normalize it, this will provide us a "compromise" metric. With these different statistical analysis, the idea would be to find a better correlation between the values collected and their individual influences on the flock movements, as some values might be high but without optimizing the flock behaviors in all of its aspects.

Finally, having gathered the fitness in simulation and the

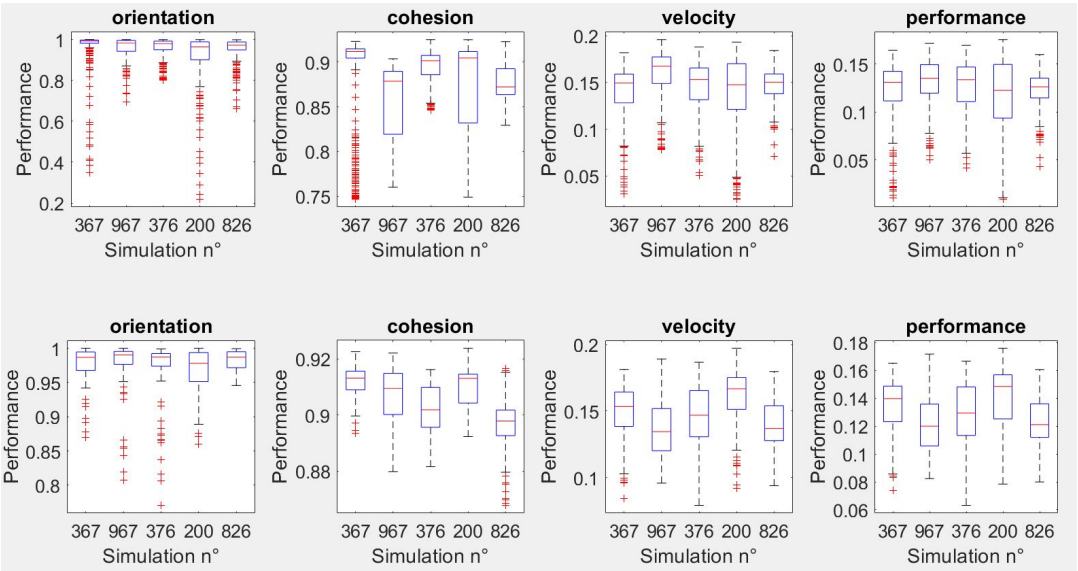

Fig. 3. Statistical results of the best simulations. Top : Mean value maximization. Down : Std minimization

corresponding instantaneous performance, one last metric has been attempted to choose the best combination of parameters, namely cumulative sum of the instantaneous performances for which the best results have been obtained. Since the aim is to have a good performance in both scenarios, the normalized sum of each has been added in order to establish the combination that yields the best results in both. Table I summarizes the best combinations for each separate scenario and the combined best, for which we have chosen: $[w_1, w_2, w_3, \alpha] = [7, 7, 4, 1.5]$ Table I shows the fitness

Scenario	**Ratio** α	**Cohesion** w_1	**Dispersion** w_2	**Alignment** w_3
Obstacles	1.5	9	8	5
Crossing	1.5	10	9	10
Combined	1.5	7	7	4

TABLE I
OPTIMAL PARAMETERS

results for each combination with the optimal parameters. The results leading to the best combined fitness were kept.

III. RESULTS

A. Meshgrid optimization

Figures 4 and 5 show the average performance metric respectively for the *obstacle* and *crossing* . The ranges of the meshgrid chosen appear to be appropriate particularly for orientation and cohesion whereas velocity is rather low. This is due to the fact that this metric considers the velocity of the center of mass which is typically far from the robot's top speed.

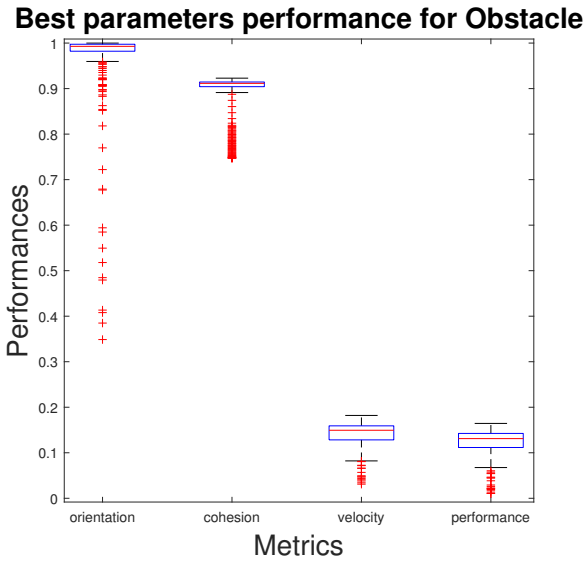

Fig. 4. Performance metrics of the optimal $[7, 7, 4, 1.5]$ choice on obstacles world

B. Scalability

After establishing the optimal parameters, the effect of varying the number of robots is displayed for both worlds

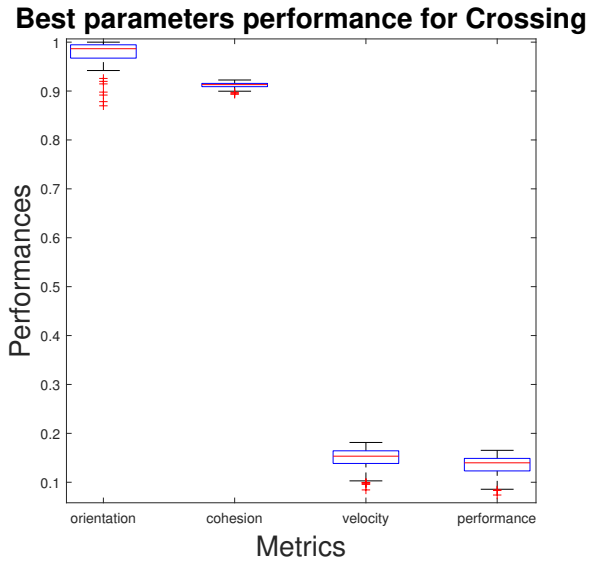

Fig. 5. Performance metrics of the optimal $[7, 7, 4, 1.5]$ choice on crossing world

in figures 6 and 7. In both cases we can see an overall high value of orientation and cohesion and a much lower velocity. Cohesion and orientation however suffer more considerably from and increase of robots, this is likely due to the increase difficulty of maintaining a broad flock in a cluttered environment.

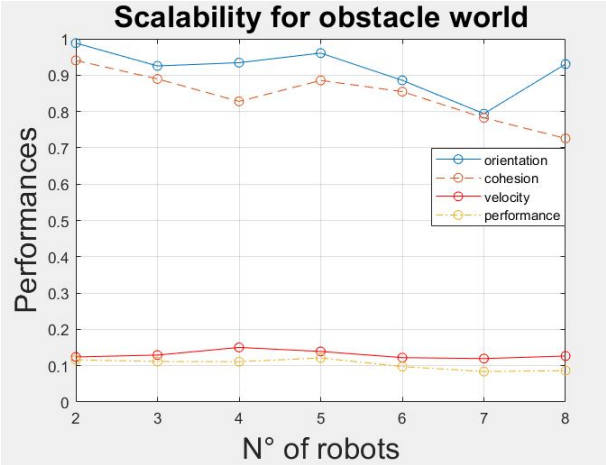

Fig. 6. Performance metrics with respect to number of robots in obstacles world

C. Real e-puck experiment

For the experiments on real e-pucks, no metrics could be computed. Hence, the behaviour of the robot depending on the implemented controllers are described in this section.[4]

- *Migration only*: In this case, the behaviour is the one desired. Different migration goal were tested and the robot was able to orient itself and move forward in those directions.

[4]Videos were taken and will be shown in the oral presentation

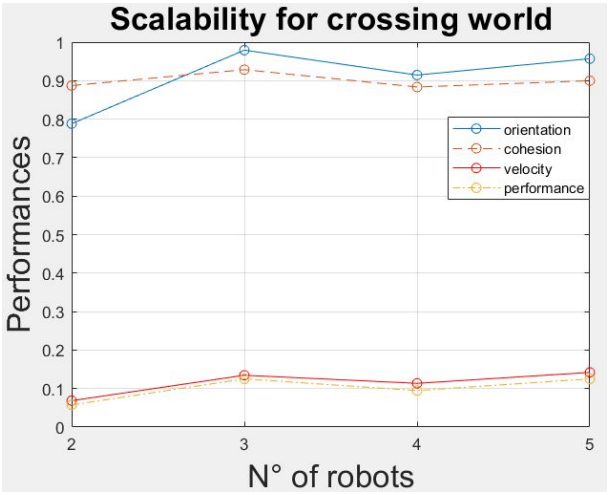

Fig. 7. Performance metrics with respect to number of robots in crossing world

- *Braitenberg only*: The obstacle avoidance alone was very reactive. It works well but has two main drawbacks. Firstly, it can react too much to an obstacle and turn more than needed. Secondly, it is not efficient to detect obstacle that are right in front of the robot (due to the IR sensors positions); obstacles must not be precisely centered in front of the robot to be avoided. This second issue is an hardware limitation.
- *Migration and Braitenberg*: Combining those controllers led to the desired behaviour. The robot moves in the migration direction until it meets an obstacle, avoids it, and then goes back to the migration direction.
- *Communication for Reynolds only*: To test communication and position detection of robots, LEDs were turned on in their direction when detecting other robots. It worked efficiently except in some cases where the reflection angle on the other robot returned on a led next to the desired one. This is again a hardware limitation.
- *Three rules of Reynolds*: With cohesion only the robots are able to re-unite; with dispersion only they are able to move away from other robots, with alignment only, they align with respect to the others. However, when the three rules are set, the flock is not very stable. This is due to not optimal parameters choice.
- *Migration and Reynolds*: The flock is more stable when implemented with migration and goes in the given migration direction.
- *Migration, Reynolds and Braitenberg*: The robots are sometimes able to form the flock and move forward while avoiding obstacles. However, the flock is unstable and doesn't follow the migratory urge precisely.
- *Full controller: Migration, Reynolds, Braitenberg and Join*: Same observations.

With the full controller, in obstacle avoidance real experiments, the robot's performance is satisfying: they form the flock, move maintaining the flock formation towards the migration urge and are able to avoid obstacles. The crossing real life experiments are less successful: robots are able to follow the migratory urge to go to the other side but they are not always successful to cross the other flock. Indeed, robots are often stuck for a period of time before being able to continue on their way.

IV. CONCLUSION

This project implemented new ideas in combination with other well known methods, seen in the course lecture, to create a controller for multi-robot navigation. Indeed, the project was greatly inspired by Reynolds for the flock formation, Braitenberg for the obstacle avoidance and the migratory urge seen in laboratories. Their implementation was then modified to best suit the need and goal of the project. The 'Join' controller is a new idea that greatly improved performances. The final controller is actually the addition of 4 individual controllers, which individual weights in the final speed computation depends on the value of infrared sensors and number of flockmates.

A great difficulty of the project of the choice of parameters to optimize the robots behaviours. In simulation, meshgrid and systematic search could be applied. This was not possible on real e-pucks due to the hardware constraint and moreover their behaviour would have been less reproducible due to their sensibility to noise, light conditions, and not always receiving messages.

Finally, the results on simulations were conclusive to obtain a flocking behavior while having a fluid motion. The results on real e-pucks are less positive and could still be improved with parameter tuning directly on the robots.

REFERENCES

[1] Webots software `https://www.cyberbotics.com/`
[2] E-puck robots `http://www.e-puck.org/`
[3] A. Martinoli, Distributed Intelligent System, EPFL ENG-466 course
[4] Reynolds, C. (1987). Flocks, herds and schools: A distributed behavioral model. In Proceedings of the 14th annual conference on computer graphics and interactive techniques (SIGGRAPH 87) (pp. 2534). New York: ACM Press
[5] libIrcom 1.0.7 library, Alexandre Campo, Alvaro Gutierrez, Valentin Longchamp, Steven Roelofsen `http://www.e-puck.org/index.php?option=com_content&view=article&id=32&Itemid=28`