Optimized simulated flocking algorithm for e-pucks

Diego Antognini · Damien Doy · Martin d'Hoffschmidt

***Abstract*—In this report, the flocking of a swarm of mobile robots is studied. Two flocking strategies are investigated and compared. The platform of test used to perform the experiments is Webots. Three flocking metrics will also be presented which are useful to evaluate the performance of the flock. Finally, Particle Swarm Optimization is used to tune the parameters of both flocking control strategies in order to achieve the highest possible performance.**

***Index Terms*—Swarm, Flocking, Reynolds, Particle Swarm Optimization.**

I. Introduction

The flocking of individuals in nature can easily be observed. Collective interaction of individuals through flocking can provide them various advantages. For instance, flock of birds are formed for migration such as to improve the energy efficiency. This report aims at investigating how to reproduce a flocking behaviour between e-pucks. Two different flocking rules are implemented and tested on the Webots platform.

The first implemented flocking strategy is a Reynold-based rules strategy. The first rule promotes the cohesion or aggregation of the flock. The second rules aims at preventing the flock mates of being too close from one another and is called the dispersion rule. The third rule is called the consistency rule and encourages the flock mate to move fast. In addition to that, the individuals of the flock should also be able to avoid obstacles. To achieve this, a Braitenberg controller is implemented for each individual.

The second flocking strategy is based on the work of [1]. We will refer to it below as the *Kikker* controller. It will be assumed there that each individual is equipped with a range and bearing. The idea of the strategy is to control each individual of the flock based on personal and shared information with its neighbours. A desired heading vector is computed such that each individual knows the direction it should follow in order to achieve a coherent flocking. Like the first controller, the desired heading vector is made of different rules similar to cohesion and separation rules.

Third, it has also been investigated how to deal with the fact the each individual of the flock can only communicate with a limited number of flock mates. This problem arises with flocks that have a high number of individuals. The idea here is that each flock mate communicates its local info but also the info of its neighbours that other neighbours would not be able to receive directly.

Finally, three flocking metrics were introduced to evaluate the performance of the two flocking strategies. Indeed, the two flocking strategies introduce several parameters that can highly influence the performance of the flock and need therefore to be tuned. To do so, Particle Swarm Optimization will be performed on them such as to find the couple of parameters that maximizes the metrics.

II. Experiments

In order to test the flocking behaviour of our controllers, we use two worlds in Webots. The first one is a world containing a variety of static obstacles with different configurations. This allows us to test the obstacle avoidance strategy in different situations as well as the flocking behaviour. The flock is composed of five robots and each flock member has the same migration urge point. It has been chosen in a way to force the flock to go through the obstacles.

The purpose of the second world is to observe how our controllers act in presence of another flock. In this world there is no obstacles because we consider the other flock to be the obstacles. So, in this situation the obstacles won't be static any more but they will be moving. Each flocks of five members start at each side of the arena and their migration urge point is defined behind the other flock. In this manner, we force the two flocks to cross themselves.

In order to evaluate the efficiency of our controllers, we measure the performance with some defined metrics that will be discussed below. This will allow us to choose the parameters accordingly. Finally, in order to find the best parameters, we will use the Particle Swarm Optimization algorithm with noise-resistance using our metrics as fitness.

III. Flocking behaviour, Reynolds unlimited

The first controller is a simple and established flocking algorithm based on Reynolds flocking rules. It works by having an entity that knows the position of all the robots in a world and compute, for each robot, a direction vector. Flocking is achieved by each robot following its direction vector. The direction is computed such that the robots want to get together. At the same time, they should also not collide. This direction vector is the sum of multiples sub vectors that ensure the flock behaviour:

- Cohesion: move toward the centre of mass of the flock
- Dispersion: move away from other robots
- Consistency: move at the same speed as the flock
- Migration: move toward an arbitrary point

These different parameters show also one of the major drawback of the Reynolds algorithm: there must be an entity who knows the position and direction of every robots in order to calculate each of desired components. Usually this is the role of a supervisor to receive the position of each robots, calculate each of the component for each robot and send back the direction and speed to each robot.

The inital Reynolds variant we worked with was the **Unlimited Range Reynolds** where there is **no supervisor**. Instead, each robot broadcasts its id, and the robots who get the message will be able to calculate the relative position of the sender using its hardware receiver. Each robot keeps in

memory a matrix containing the position and speed of the other robots which will be used to calculate the personal direction toward the center of mass and dispersion.

This algorithm is taken from the *Laboratory 04 - part 02* of the course. The Unlimited Reynolds controller has a performance equal to the Reynolds with a supervisor, at the cost of more communication between robots, more computation for each robots, since the center of mass is calculated multiple times, which makes the Unlimited Reynolds algorithm less scalable than the centralized one. We tested this controller through the *obstacles.wbt* world and the flock was able to navigate through the obstacles without apparent problem. We used the results of the performance of the flock with this algorithm as a baseline for other experiments.

IV. Flocking behaviour, Reynolds limited

The unlimited range Reynolds algorithm is good at creating a robust flocking behaviour but it suffers from a major drawback. The unlimited range for communication is unrealistic at many different levels.

An infinite or great range means that the communication module on a real e-puck will drain a lot of energy from the battery. Assuming infinite battery capacity, the communication can not be perfect between robots in a flock across obstacles. Sometimes robots will be out of sight from the flock for a short moment. Unlimited communication is not scalable and as the number of robots increases, so does the number of messages received for each robot. Resulting in a accordingly increasing cost to handle them. Also, the communication medium (air) will get polluted and a lot of noise will be observed.

To solve this problem, it is in fact possible to represent the flock as a connected graph. Where robots are the vertices and where the edges represent the communication between robots. The unlimited communication can be represented as a complete graph and the limited communication as a loosely connected graph. This graph representation explain why an unlimited range communication is not viable in a flock: the number of edges in a complete graph is $\frac{N(N-1)}{2}$, where N is the number of vertices. To allow for a scalable and reactive flock, we have to reduce this number of edges and this is where a limited range communication model will allow us to optimize the scalability of the classical Reynolds flock algorithm.

This graph representation of the communication between robots is analogous to a switched network like the internet. Our flock, like a switched network, want to know a global state of the system without being connected to all elements in order to find the centre of mass of the flock. Since this problem is an important one in the switched network for routing, a lot of optimized algorithms exists and we can use one of them from this area for our purpose.

The algorithm that has been chosen is the Bellman Ford, which is a routing algorithm used in networks to know the lowest latency route. The huge advantage of this algorithm is that it uses local information and communication to create a general state. A node only has to know about its neighbours and exchange information with them to understand the global state of the system.

The fact that Bellman-ford is used in huge networks like Internet means that it is resistant to node failures. In our flock a failure can be represented by a robot getting out of sight (because of an obstacle) from the flock. Our algorithm being resistant is important to avoid having the flock wait for a robot that would be lost forever. Another huge advantage of an algorithm such as Bellman-Ford is that it is scalable. Indeed the algorithmic complexity of the Bellman-Ford algorithm is $\mathcal{O}(|E| \cdot |V|)$, that means that the algorithm should be scalable as long as we limit the number of edge in the graph, this can be done by a really short range.

A. Algorithm

Algorithm 1 Limited distance communication

1: $timestamps[my_id] \leftarrow current_timestamp + 1$
2: $broadcast(position(0,0), timestamps[my_id])$
3: **if** $messagereceived$ **then**
4: **if** $timestamps[id_sender] < packet(timestamp)$ **then**
5: $timestamps[id_sender] \leftarrow packet(timestamp)$
6: $position[id_sender] \leftarrow packet(position) + position_emitter_packet$
7: $broadcast(position[id_sender], timestamps[id_sender])$
8: **end if**
9: **end if**

Each robot sends a ping with its information ($timestamp$) to other robots, they also send their current position which is equals to $(0, 0)$. These robots will have matrices to store the position of other robots and their timestamp. Every time a robot receives a packet, it will forward it by broadcasting it (in the limits of its range). To avoid that a robot broadcasts an outdated information or just rebroadcasts a packet it has just send, we store the timestamp and increment it every time a robot sends an information. The timestamp also allows a flock to "forget" a robot that has wander out of the flock or is stuck. This can be done by not taking robots with an old timestamp in the calculation of the centre of mass.

The emitters range of each robots has been set to 10 cm. It means that e-pucks have to be really close to each other in order to communicate. The problem with this approach is that e-pucks have to start near one another. It will be seen in the results section that limited range algorithm actually has a similar performance to the unlimited range one.

V. Flocking behaviour, another approach : Kikker

The flocking behaviour described in this section is based on the work of Ali et al. [1]. Some modifications have been brought however in order to adapt the controller the our case of simulation. We decided to call this new controller Kikker.

A. Flocking behaviour

The flocking behaviour is obtained through decisions taken based on the knowledge of the neighbours for each individual of the flock. This decision might either be global or local. It

is given in the form of a desired direction $\mathbf{a}$ computed locally by each individual as the weighted sum of heading alignment vector $\mathbf{h}$, the proximal control vector $\mathbf{b}$, the cohesion vector $\mathbf{c}$ and the migration urge $\mathbf{d}$.

$$\mathbf{a} = \frac{\mathbf{h} + \beta\mathbf{b} + \gamma\mathbf{c} + \delta\mathbf{d}}{\|\mathbf{h} + \beta\mathbf{b} + \gamma\mathbf{c} + \delta\mathbf{d}\|} \tag{1}$$

The desired direction vector $\mathbf{a}$ represents then the direction the e-puck should follow in its reference frame. This last is defined according to a real and complex axis which we will respectively refer to as the x and y axis. The x axis, or the real axis, corresponds to the forward heading of the e-puck. The y axis, or the imaginary axis, corresponds to the left wheel axle. We will refer below to this frame as the body-fixed reference.

B. Heading alignment

All the individuals of the flock have to adapt their heading direction according to their neighbours. Therefore, the heading alignment vector quantifies averaged heading for all the neighbours of one individual.

$$\mathbf{h} = \frac{\sum_{k\in\mathcal{N}_r} \exp(j\theta_k)}{\|\sum_{k\in\mathcal{N}_r} \exp(j\theta_k)\|} \tag{2}$$

The angle θ is the heading angle of a neighbour in the body-fixed reference frame of the current robot. The set of considered neighbours is expressed as $\mathcal{N}_r$. It is equal to the flock-size if the communication between the robots is global. If the communication is range-limited, the quantity $\mathcal{N}_r$ accounts for the number of visible neighbours in the limited range.

C. Proximal control

The obstacle avoidance plays an important role in the flocking behaviour. The infra-red sensors of the individuals are used to take care of the obstacles that might be encountered on the way. When an obstacle is detected by one of the sensors, a virtual force is computed and acts on the individual in the opposite direction with respect to the sensor's position. The virtual force is denoted by the quantity f_k computed as,

$$f_k = -\frac{(o_k)^2}{C} \tag{3}$$

where o_k is the sensed value from the *kth* sensor. The virtual force is computed only if the sensed value is above a minimum threshold denoted by o_{min}. This result is scaled by a constant C which is equal to the sensors' maximum detected value.

The proximal control vector is computed according the the position angle ϕ_k of the *kth* sensor in the body-fixed reference frame,

$$\mathbf{p} = \frac{1}{N_s}\sum_{k=1}^{N_s} f_k \exp(j\phi_k) \tag{4}$$

where N_s is the number of sensors.

D. Cohesion

The cohesion vector aims at keeping the individuals of the flock together. In same way as for the proximal control, a virtual force g_j is computed as to prevent the individuals from being too close or too far from each others. To do so, the distance d_j and angle ψ_j from the *jth* neighbour are computed. The cohesion vector is computed as a sum of all the forces with their respective direction for all the neighbours.

$$\mathbf{c} = \sum_{j\in\mathcal{N}_r} g_j \exp(j\psi_j) \tag{5}$$

The angle ψ_j corresponds to the heading of the *jth* neighbour with respect to the north. It is sent to the current robot which can then express this angle in its body-fixed reference frame knowing its own heading with respect to the north. The distance d_j is computed from the emitters that equips each robot. One defines also a reference distance called d_{ref} which is the target distance that should separate each robot from one-another.

$$g_j = \begin{cases} +(d_k - d_{ref})^2 & \text{if } d_k \geq d_{ref} \\ -(d_k - d_{ref})^2 & \text{if } d_k < d_{ref} \end{cases} \tag{6}$$

E. Migration control

The migration control vector $\mathbf{r}$ aims at indicting the way all the individuals of the flock should follow. Therefore, the error on the migration is computed as the difference between the current heading angle of the individual with respect to the North and the migration angle.

$$\mathbf{r} = \exp\left(j(\theta - \Phi)\right) \tag{7}$$

F. Motion control

The motion control uses the the desired heading vector $\mathbf{a}$ in order to compute the the forward velocity denoted by u and the angular velocity denoted by ω. The forward velocity is computed as follows.,

$$u = \begin{cases} \mathbf{a}\cdot\mathbf{a_c} & \text{if } \mathbf{a}\cdot\mathbf{a_c} \geq 0 \\ 0 & \text{otherwise} \end{cases} \tag{8}$$

where the vector $\mathbf{a_c}$ represents the current heading vector of the individual in the body-fixed reference frame. In other words, $\mathbf{a_c}$ as a unity component in the x axis and zero along the y axis.

The angular velocity of the robot is the result of a proportional controller with coefficient K_p applied on the error between the phase of vectors $\mathbf{a_c}$ and $\mathbf{a}$. Therefore, the higher the error between the desired direction and the current direction, the higher the angular velocity.

$$\omega = K_p\left(\angle\mathbf{a_c} - \angle\mathbf{a}\right) \tag{9}$$

The forward velocity and the angular needs then to be converted into useful data input for the robot. The rotational speed of the right of left wheel can then be respectively computed as follows.

$$N_R = \left(u + \frac{\omega}{2} l\right) \frac{1}{2\pi R_w} \quad (10)$$

$$N_L = \left(u - \frac{\omega}{2} l\right) \frac{1}{2\pi R_w} \quad (11)$$

where R_w is the radius of the wheel of the robot and l denotes its track.

Note that when the desired direction is opposite to the robot's current direction the forward velocity is limited to zero. It makes it then possible for the robot turn on itself to get back on the right track.

VI. Flocking metrics

There are several ways to evaluate the performance of a flock. In this project, it has been decided to measure the performance of the flock according to three criterions. The flocking individuals should be aligned toward a common direction, move on a compact and coherent way and they should move fast.

The first metric measures how the flock mates are oriented with respect to the others and is given by equation 12. The metric is normalized according to the size of the flock given by N.

$$o[t] = \frac{1}{N} \left| \sum_{k=1}^{N} \exp\left(j\psi[t]\right) \right| \quad (12)$$

Second, the velocity of the flock is evaluated as the average displacement velocity of the geometric center of mass along the direction of the migration urge. This last is represented by Φ.

$$v[t] = \frac{1}{v_{max}} \max\left[\text{proj}_{\Phi}(\overline{x}[t] - \overline{x}[t-1])\right] \quad (13)$$

The third metric, called the entropy, measures the positional disorder of the flock. It is computed as,

$$s[t] = \int_0^{\infty} \sum_{k=1}^{M} p_k(h) \log_2(p_k(h)) dh \quad (14)$$

where p_k represents the proportion of individuals in the *kth* cluster and M is the amount of clusters for a given distance h. Two robots are considered to be in the same cluster if the distance between then is less or equal to h.

Finally, the instantaneous performance $p[t]$ of the flock at a given time t is computed as the product of the three above given metrics.

$$p[t] = o[t]\, v[t]\, s[t] \quad (15)$$

Note that all the metrics are computed such that the result of each of them yields a value between 0 and 1. The overall performance of the flock for a given simulation can be evaluated as the average of instantaneous performance over the time simulation T.

$$\overline{p} = \frac{1}{T} \sum_{t=1}^{T} p[t] \quad (16)$$

VII. Particle Swarm Optimization

In order to optimize the parameters of our controllers, we have used the Particle Swarm Optimization algorithm. This meta-heuristic technique can be used to search candidate solutions in large combinatorial space. Moreover, it can be adapted to be noise-resistant.

We have used it to optimize the Braitenberg parameters for the Reynolds limited controller, and the parameters of the Kikker controller we have implemented. We didn't used PSO to optimize the flocking parameters of the Reynolds controller because it has been considered that they were good enough and the limitation of this controller were the Braitenberg weights.

A. Experiments

In order to evaluate candidate solutions for both controllers, we have created a special world in Webots where the flock will go toward a long path containing various obstacles. In this way, the obstacle avoidance strategy is tested several times for each simulation. The simulation is stopped when a maximum number of steps is completed. This number should be enough to allow the flock to cross the arena of obstacles.

A run lasts around 5 minutes in Webots time. The fitness used is the same as described in section "VI Flocking Metrics". Moreover, each particle is evaluated several times in order to have an average of its fitness. Indeed, a single evaluation might not be representative of a good solution due to noise. Also, we have run four instances of Webots in our machines using multiple cores in order to accelerate the optimization process.

B. Reynolds Limited controller

Considering that the flocking parameters were already good enough, it has been decided to take into consideration only the Braitenberg weights of the obstacle avoidance controller. Indeed, it has been observed that our flocks had some difficulties to handle obstacles correctly. Two approaches have been experienced. The first one was to use a recursive feed forward neural network, where a particle size is 26 (8 sensors, 2 recurrent weights and 1 bias, for each wheel). The second one was assigning directly the weights in the Braitenberg matrix with null values for the two back sensors. In this case, a particle has a size of 12 (6 weights for each wheel).

The expected results were no met. The main reason is that the evaluation of one particle is very time expensive, around 5 minutes in Webots time (around 20 seconds in our computers). Considering that certain number of particles (e.g. 10) have to be evaluated several times (e.g. 3), we have for a single iteration around 10 minutes of computational time. In order to find a good solution, hundreds of iterations might be necessary and unfortunately we were only able to compute approximately 20 iterations.

C. Kikker controller

The parameters that were taken into account to perform the optimization of flocking behaviour are essentially the weights introduced in the computation of the desired heading direction $\mathbf{a}$. More specifically it means that one desires to

tune on an optimal way the coefficient β, γ and δ such that respectively the contribution of the obstacle avoidance, cohesion and migration urge are well scaled.

One will also set the proportional gain of the angular velocity controller as an optimization parameter. Indeed, its value is expected to play a significant role in the performance of the flocking. In addition, the parameter d_{ref} is also set as an optimization parameter.

The table I gives to bounds considered on each parameter. To run the algorithm, each optimization parameter has been scaled between 0 and 1 according to the given bounds. It was then specified in the algorithm that values respectively below and above 0 and 1 are forbidden.

Name	lower bound	upper bound
β	1	20
γ	1	20
δ	0.01	2.0
K_p	0.1	1.5
d_{ref}	0.05	0.50

Table I
OPTIMIZATION PARAMETERS

It can be noticed that the number of parameters introduced in PSO is now less than the ones introduced in the case of the Reynolds controller. It was therefore expected to reach good results in less iterations with the Kikker controller. Also, the fact that the parameters are normalized should make it easier for the PSO algorithm to find a solution.

After several simulations, interesting parameters were found within the considered bounds. However the performances of the flock were still not very satisfactory. In fact the same issue as the other controller has probably been met and a higher amount of iterations should be performed to come to a better solution.

D. Results

In order to compare our baseline with parameters found with PSO, we made 20 runs with our flock on the world *obstacles.wbt*. A run consist of an experiment where the flock go through the world until reaching the other side. Table II we reported the mean and the standard deviation.

Parameters	performance
Baseline	0.124 ± 0.018
$\beta = .38, \gamma = .33, \delta = .69, K_p = 1.00, d_{ref} = .34$	0.100 ± 0.020
$\beta = .38, \gamma = .29, \delta = .87, K_p = .94, d_{ref} = .36$	0.111 ± 0.020

Table II
PERF COMPARISON BETWEEN BASELINE AND PARAMETERS FROM PSO

We can observe that our baseline has the best performance in term of mean and also in term of standard deviation. The results found by PSO are not as good in term of mean and also have a greater standard deviation. However, we strongly think that we more iteration and more times, we could find a solution which should be better than our baseline.

VIII. RESULTS

The three flocking controllers, i.e. the unlimited range Reynolds, the limited range Reynolds, and the Kikker controller, will now be compared in performance. For each algorithm, we have made 20 runs with our flock on the *obstacles.wbt* world which contains 5 robots and some blocks to test the flocking behaviour. A run consist of the time the whole flock takes to get to the end of the world. First will be compared the metrics against each algorithm. Indeed, the metrics result is good a global indicator of the flock performance.

Algorithm	mean perf	standard deviation
Unlimited Reynolds	0.217	0.013
Limited Reynolds	0.210	0.012
Kikker controller	0.131	0.010

Table III
PERFORMANCE OF OUR FLOCKING ALGORITHMS

It can first be noticed that the limited and unlimited Reynolds algorithm have a similar performance. This result is very important as it indicates that the limited implementation approach would allow some real robots to drastically improve their energy efficiency. Therefore, it would be possible to consider large-scale flocks for real experiments with that kind of scalable controller.

The Kikker controller is shown to be less efficient in terms of performances than the other classical algorithms. However, when its behaviour is observed in simulation it presents no apparent problems compared to the Reynolds algorithm. It even seems to work its way better around the obstacles because the flock mates seem to be able to wait one another. In fact, the performance is lowered because the robots are much more slower with that kind of controller.

The metrics takes into account the speed of the robots to their migration point and the compactness (entropy) of the flock. These two metrics are well achieved by the Reynolds algorithm, since it creates a really compact flock going fast. However the flock can have some problems with the obstacles due to the Braitenberg algorithm built into Reynolds. This problem is also shown in the standard deviation of our results which show that the Reynolds algorithm produces results that are less consistent than the kikker controller, a robot can lag behind because of an obstacle which can lower entropy.

One can argue that the fact that an algorithm is more robust to all type of inputs would make it better than another one which doesn't, but the fact that the Reynolds flock is compact allow us to use the limited range algorithm. Whereas the kikker controller has a more loosely compacted flock which does a lot of exploration which means that the limited range should be much greater than the Reynolds flock, limiting the advantages we listed and made us implement this method. This means that the Reynolds algorithm is more suited to a flock with a great number of simple robots, with little computing power and battery, whereas the Kikker controller is more suited to a more complex and natural flock with robots of higher complexity.

Algorithm	time
Unlimited Reynolds	4min
Limited Reynolds	4min
Kikker controller	5min 30sec

Table IV
TIME TO GET THROUGH *obstacles.wbt*

The time taken is, as expected, again in favour of the Reynolds algorithm which takes the shortest path to get to the migration point. The Kikker controller, on the other hand, have the added advantage that the robots cover more terrain to avoid the obstacles and can navigate through a more complex world.

A. Flock collision

The controller has also been tested with two flocks having to cross each other. The challenge here is that the obstacles are the robots of the other flocks, which are now moving obstacles and should introduce additional noise in the simulation. This experiment is also a good way to see what are the problems of our implementations and to compare the diverse algorithms we have.

Algorithm	time
Limited Reynolds	1min 40sec
Kikker controller	3min to 5min

Table V
TIME TO GET THROUGH A COLLISION OF 2 FLOCKS OF 5 ROBOTS EACH

The Kikker controller is slower than the Reynolds algorithm. But on the *obstacles.wbt* world, the fact that the Kikker controller is slower present one advantage. It allows indeed the flock to find a better solution to avoid obstacles.

The Reynolds controller is faster the Kikker controller as it will take a quick decision on the path to go through the other flock. The Kikker controller, more hesitant, will have some troubles to get through the other flock as much hesitant.

IX. CONCLUSION

In a nutshell, two different kind of controllers have been implemented and tested. Their parameters have also been optimized using Particle Swarm Optimization with mixed results. Each of them present advantages and drawbacks.

First, the Reynold controller was implemented with unlimited range and yields good flocking results. However, its non scalable character make it difficult to justify a real experiments with a large amount of robots. This problem was tackled by implementing a limited range controller where there is only communication within a certain range. It has been shown that this local communication approach yields the same results as the unlimited approach. It presents also the significant advantage of being very scalable for real applications as well as it could significantly improve the energy efficiency of the e-pucks, without requiring a supervisor.

Second, the Kikker controller might be more expensive in terms of computation time because it is a bit more complex. Also, the results have shown that it does not yield better performances that the first controllers. The main raison for this result is that the Kikker controller makes the robots slightly more slower. However, it has been observed during simulation that the flock can wait for flock-mates who could eventually have difficulties to avoid certain obstacles, the fact that it is more computationally intensive that the Reynolds controller allows the flock to take more intelligent decisions, while being free of any supervisor.

Third, performing Particle Swarm Optimization on the controllers does not yield the expected results even if the solution is still good. We believe indeed that running PSO for much more iterations could potentially lead us to better solutions. The Reynolds controller as it is, already creates a flocking behaviour that is compact and fast and is able to navigate through obstacles. The Braitenberg routine of the Reynolds algorithm would take advantage of having its parameters tuned by PSO, since its high number and the high variance of the results makes it hard to tune by hand, but the randomness of our runs, the high number of parameters and the fact that each runs take a long time means that the PSO algorithm takes a really long time to converge to a solution which could improve on our current one.
The Kikker controller is more complex in terms of computation but have an enormous advantage in terms of optimization over the Braitenberg/Reynolds combination: it has less parameters defining its behaviour. Indeed, despite the apparent complexity of the Kikker controller, it was easier to find a satisfactory solution with PSO since it has few parameters to optimize. Where it would take hundreds or thousands of hours to find a Braitenberg solution. We are able to find a Kikker solution in hours.

In the end, we have two controllers, representing two opposite spectrums of the flock algorithms. On one side, we have the Reynolds controller which is a simple algorithm that we can describe in a few words and can work on any microcontroller. Its compactness allowed us to tinker and be able to create a limited communication scheme on it. But can be hard to fine-tune given the large amount of parameters.
On the other side we have the Kikker algorithm, more complex but more resistant to the noisy input of a world filled with obstacles. The small amount of parameters allowed us to easly optimize its behaviour to finally have a flock that can navigate through obstacles in a natural way.
This difference is reflected on the flock collision world, where the more forward Reynolds controller can find its way faster than the more hesitant and careful Kikker controller.

REFERENCES

[1] Ali E. Turgut. Hande Çelikkanat. Fatih Gokçe and Eraol Sahin, *Self-organized flocking in mobile robot swarms*, Swarm Intell, 2008.