ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

REPORT

ENG-466 DISTRIBUTED INTELLIGENT SYSTEMS

Fadel Mamar Seydou, Louis Jouret, Hao Zhao

7th June 2022

CHAPTER 1

PART 1

Question 1.1 In identify_object(), we detect three kinds of object according to distance sensors. If the

Algorithm 1 identify object

Input: distances, threshold_1, threshold_2
Output: obj_type

```
1:  function IDENTIFY_OBJECT(distances)
2:      num_detected = 0
3:      final_activated_sensor = 0
4:      for sensor_nb = 0 → NB_SENSORS do
5:          if distances[sensor_nb] > threshold_1 then
6:              num_detected ++
7:              final_activated_sensor = sensor_nb
8:              for j = 0 → sensor_nb do
9:                  if distances[j] > threshold_2 then
10:                     num_detected ++
11:                 end if
12:             end for
13:             for k = sensor_nb → NB_SENSORS do
14:                 if distances[j] > threshold_2 then
15:                     num_detected ++
16:                 end if
17:             end for
18:         end if
19:     end for
20:     if num_detected == 0 then
21:         obj_type = 0
22:     else if num_detected == 1 then
23:         obj_type = 1
24:         detected_ps = final_activated_sensor
25:     else
26:         obj_type = 2
27:     end if
28: end function
```

number of activated sensors is equal to 0, then no object has been detected. If the number of activated

sensors is equal to 1, then a free seed has been detected. If the number of activated sensors is larger than 1, then large objects (e.g., wall, robots, clusters) has been detected. To robustly determine whether one sensor has been activated or not, we set up two threshold. $threshold\_1 = 1000$ is used to detect a really close position with respect to objects. And $threshold\_2 = 150$ is used to detect a sub-activated sensor.

Question 1.2 In search(), we generate a list of random numbers, and use them to set up the velocity of

Algorithm 2 search

Input: robot_id, const_speed, counter
Output: msr, msl

```
1:  function SEARCH(robot_id)
2:      float left_speed
3:      int rand_num[5]
4:      num_detected = 800
5:      for i = 0 → NB_ROBOTS do
6:          rand_num[5] = rand()
7:      end for
8:      for j = 0 → NB_ROBOTS do
9:          if robot_id = j then
10:             msl = rand_num[j]
11:         end if
12:     end for
13:     msr = const_speed - msl
14:     counter ++
15: end function
```

robots. We keep the translational velocity constant by constraining the sum of velocity of two wheels the same.

Question 1.3 We implement a Braitenberg obstacle avoidance algorithm in this question. We used a group of weights different from the give weights. With this implemented Braitenberg obstacle avoidance algorithm. We can realize smooth obstacle avoidance. The pseudo code can be seen in Algorithm 3.

Question 1.4

In release_seed() and grip_seed(), we set up the query number and then send robot_id, detected_ps, and query information to the supervisor through emit_message_to_supervisor(). The pseudo code can be seen in Algorithm 4 and Algorithm 5.

Question 1.5 The robots can successfully detect free seeds and collect them to form clusters. And during the collection process, robots can smoothly implement state transitions. However, robots will destruct a big cluster.

Algorithm 3 perform_obstacle_avoidance

Input: l_weight, r_weight, $distances$, MAX_SPEED_WEB, counter
Output: msl_w, msr_w

```
1: function PERFORM_OBSTACLE_AVOIDANCE(l_weight, r_weights, distances)
2:     float left_speed = 0
3:     float right_speed = 0
4:     for i = 0 → NB_ROBOTS do
5:         left_speed += l_weight[i] * distances[i]
6:         right_speed += r_weight[i] * distances[i]
7:     end for
8:     double val = abs(distances[7] - distances[0)
9:     if val < 200 && distances[0] > 0 then
10:        left_speed = 100
11:        right_speed = -100
12:    end if
13:    msl_w = left_speed * MAX_SPEED_WEB / 1000
14:    msr_w = right_speed * MAX_SPEED_WEB / 1000
15:    counter ++
16: end function
```

Algorithm 4 release_seed

Input: $robot\_id$, $detected\_ps$
Output: message

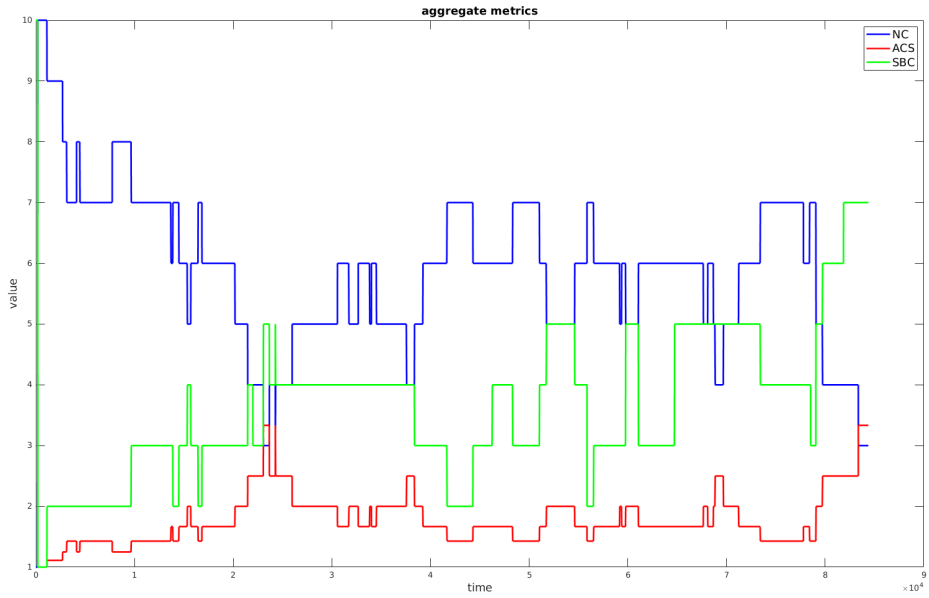
```
1: function RELEASE_SEED(robot_id, detected_ps)
2:     query = 0
3:     emit_message_to_supervisor(robot_id, detected_ps, query)
4: end function
```

Algorithm 5 grip_seed

Input: $robot\_id$, $detected\_ps$
Output: message

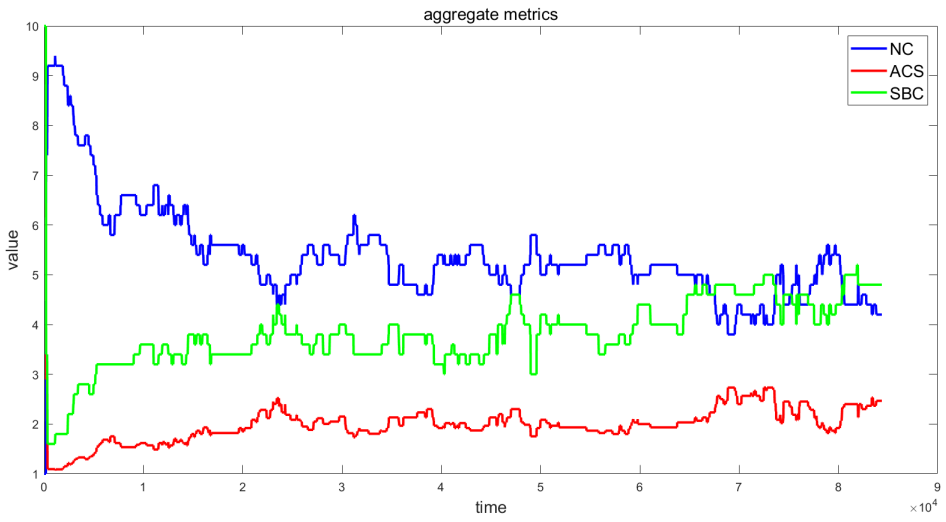
```
1: function GRIP_SEED(robot_id, detected_ps)
2:     query = 1
3:     emit_message_to_supervisor(robot_id, detected_ps, query)
4: end function
```

Question 1.6



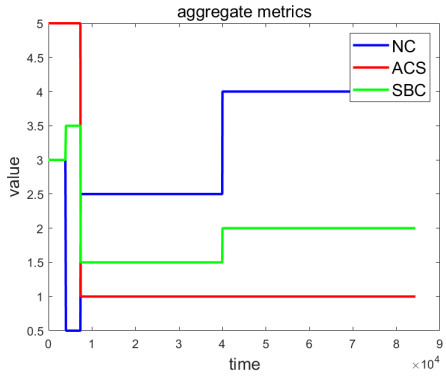
In this question, we implement calculate_metrics() function and obtain NC, ACS and SBC metrics with respect time. We can see that NC has a clear decreasing trend. ACS and SBC both have a increasing trend. In this simulation, maximum SBC can reach 7 and maximum ACS is higher than 3. And NC can be down to 3.

Question 1.7



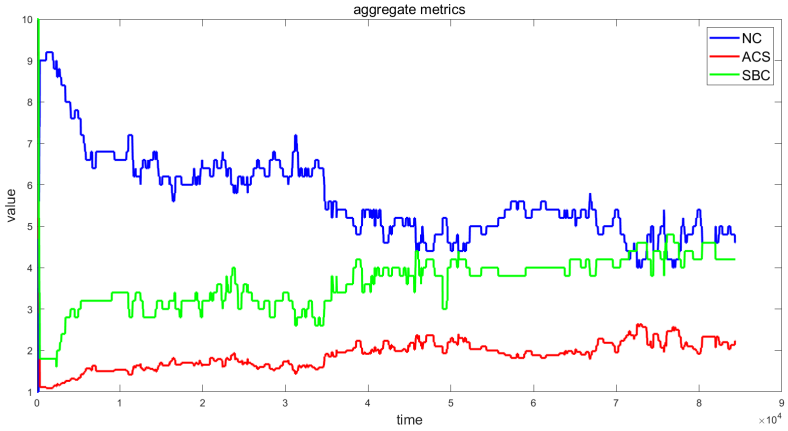
In this question, we repeat the procedure explained in Question 1.6 for 5 different runs (each 3h), calculate, plot and submit the mean of the three metrics over the experiments. The results is shown in the picture above. We can see that NC has a clear decreasing trend. ACS and SBC both have a increasing trend.

Question 1.8



With less seeds, the encountering probability greatly decreases, which means robots needs spend more time in searching. NC will be difficult to decrease. ACS and SBC metrics will also be much lower than the case with the configuration of Question 1.7.

Question 1.9



Compared with the result from Question 1.6, NC reduces slightly quicker and reaches a similar minimum value. ACS and SBC are slightly lower, I think the reason is that with more robots to work in the arena, a large cluster is more likely to be deconstructed by moving robots.

Question 1.10 We design the FSM of the system as follows,

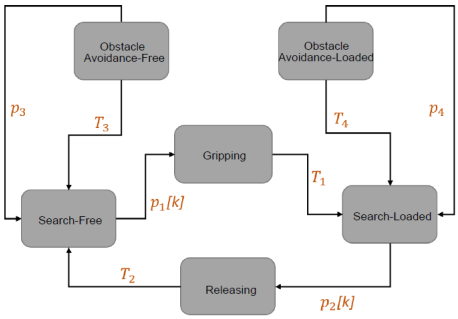


Figure 4. FSM of the system. p's represent transition probabilities and T's are transition delays.

For each robot at the state *Search-Free* and *Search-Loaded*, it has the probability of implementing *Obstacle Avoidance-Free* p_3 and probability of implementing *Obstacle Avoidance-Loaded* p_4, respectively. Also, between *Search-Free* and *Search-Loaded* state, a robot has a time-dependent probability of gripping $p_1[k]$ and another time-dependent probability of releasing $p_2[k]$.

Question 1.11

$$p_1[k] = \sum_{n=1}^{N_s} p_{c_n}^{dec}[k] N_{c_n}[k] \tag{1.1}$$

$$p_2[k] = \sum_{n=1}^{N_s} p_{c_n}^{inc}[k] N_{c_n}[k] \tag{1.2}$$

Where we iterate over all cluster sizes from 1 to N_s. $p_1[k]$ represents the time-dependent probability of gripping a seed. It can be calculated from adding the probability of gripping a seed from a cluster of size $N_{c_n}[k]$. And $p_2[k]$ represents the time-dependent probability of releasing a seed. It can be calculated from adding the probability of releasing a seed to a cluster of size $N_{c_n}[k]$. We use $p_{c_n}^{dec}[k]$ and $p_{c_n}^{inc}[k]$ to represent modified gripping and releasing probabilities.

Question 1.12

$$p_3 = \frac{v_r T}{S - 2r_s} + \frac{(N_r - 1)(2r_s + d_r)v_r}{S^2} + \frac{2N_s(2r_s + d_s)v_r}{3S^2} \tag{1.3}$$

$$p_4 = \frac{v_r T}{S - 2r_s} + \frac{(N_r - 1)(2r_s + d_r)v_r}{S^2} + \frac{2N_s(2r_s + d_s)v_r}{3S^2} \tag{1.4}$$

For a robot at the state of *Search-Free*, it enters the *Obstacle Avoidance-Free* state instantly when it comes across the border of the arena and another robots. The probability of coming across the border can be represented as $\frac{v_r T}{S-2r_s}$. And the probability of coming across another robots can be represented as $\frac{(N_r-1)(2r_s+d_r)v_r}{S^2}$. Here, we consider the maximum detection area for a robot is $(2r_s + d_r)v_r$. Meanwhile, we should consider the probability of a robot failing to grip a seed and then entering the *Obstacle Avoidance-Free* state. We made a strong consumption that all seeds are placed freely inside the arena and can be seen as N_s clusters. In this way, we can calculate the probability of coming across a cluster as $\frac{N_s(2r_s+d_s)v_r}{S^2}$. For the seed inside clusters, we have a third probability to grip a seed and a two third probability to implement obstacle avoidance. The same analysis can be applied onto robots at the state *Search-Loaded*.

Question 1.13 The discrete difference equation for $N_{c_n}[k]$ can be written as follows,

$$\begin{aligned} N_{c_n}[k+1] = N_{c_n}[k] + [p_{c_{n+1}}^{dec}[k - T_{rg}] N_{c_{n+1}}[k - T_{rg}] - p_{c_n}^{dec}[k - T_{rg}] N_{c_n}[k - T_{rg}]] W_{sf}[k - T_{rg}] \\ + [p_{c_{n-1}}^{inc}[k - T_{rg}] N_{c_{n-1}}[k - T_{rg}] - p_{c_n}^{inc}[k - T_{rg}] N_{c_n}[k - T_{rg}]] W_{sl}[k - T_{rg}] \end{aligned} \tag{1.5}$$

$N_{c_n}[k+1]$ can be represented as $N_{c_n}[k]$ add the number of seeds delivered from the cluster of size $c_n + 1$ and the cluster of size $c_n - 1$. For example, the number of seeds delivered from the cluster of size $c_n + 1$ can be calculated as the difference rate from $c_n + 1$ to c_n at time $k - T_{rg}$ times the number of robots at the state *Search-Free*. The same calculation works for the number of seeds delivered from the cluster of size $c_n - 1$.

Question 1.14 The difference equations are the following :

- 1. $W_{sf}[k+1]$ = $W_{sf}[k]$ + $p_2[k-T_{rg}]$*$W_{sl}[k-T_{rg}]$ + $p_3 * W_{sf}[k-T_{oa}]$ - $p_1[k] * W_{sf}[k]$ - $p_3 * W_{sf}[k]$
- 2. $W_{sl}[k+1]$ = $W_{sl}[k]$ + $p_1[k-T_{rg}]$*$W_{sf}[k-T_{rg}]$ + $p_4 * W_{sl}[k-T_{oa}]$ - $p_2[k] * W_{sl}[k]$ - $p_4 * W_{sl}[k]$
- 3. $W_g[k+1]$ = $W_g[k]$ + $p_1[k]$*$W_{sf}[k]$ - $p_1[k-T_{rg}]$*$W_{sf}[k-T_{rg}]$
- 4. $W_r[k+1]$ = $W_r[k]$ + $p_2[k]$*$W_{sl}[k]$ - $p_2[k-T_{rg}]$*$W_{s2}[k-T_{rg}]$
- 5. $W_{ol}[k+1]$ = $W_{ol}[k]$ + $p_4 * W_{sl}[k]$ - $p_4 * W_{sl}[k-T_{oa}]$
- 6. $W_{of}[k+1]$ = $W_{of}[k]$ + $p_3 * W_{sf}[k]$ - $p_3 * W_{sf}[k-T_{oa}]$

In these equation a positive term represents and *inflow* and a negative term represents an *outflow*. The explanation of the **individual term** of each equation is below :

- p_4 is the probability of entering "Obstacle avoidance-loaded" mode.
- p_3 is the probability of entering "Obstacle avoidance-free" mode.
- $W_{sf}[k]$ is the number of robots in "search free" at time step k.
- $W_{sl}[k]$ is the number of robots in "search loaded" at time step k.
- $p_1[k]$ is the probability of entering "Gripping" mode at time step k.
- $p_2[k]$ is the probability of entering "Releasing" mode at time step k.
- T_{rg} is the average delay during "Releasing" or "Gripping".
- T_{oa} is the average delay during "Obstacle avoidance-free" or "Obstacle avoidance-loaded".
- $W_g[k]$ is the number of robots in "Gripping" at time step k.
- $W_r[k]$ is the number of robots in "Releasing" at time step k.
- $W_{of}[k]$ is the number of robots in "Obstacle avoidance-free" at time step k.
- $W_{ol}[k]$ is the number of robots in "Obstacle avoidance-loaded" at time step k.

Question 1.15 After neglecting the number of robots in obstacle avoidance, releasing and gripping states; and setting all the delay variables to zero, the discrete equation for the macroscopic model can be represented as follows,

$$W_f[k+1] = W_f[k] - p_c^{dec}[k]W_f[k] + p_c^{inc}[k]W_l[k] \tag{1.6}$$

Apply the steady state analysis $W_f[k+1] = W_f[k]$ to the difference equation and we obtain,

$$\frac{W_f^*}{W_l^*} = \frac{p_c^{inc}}{p_c^{dec}} \tag{1.7}$$

Where $W_f^* + W_l^* \approx W_0$ and

$$p_c^{inc}[k] = \sum_{n=1}^{N_s} p_{c_n}^{inc} N_{c_n}[k] \tag{1.8}$$

In the steady state, we can assume there is only one cluster of size L^*. With this assumption, we can obtain,

$$\frac{W_f^*}{W_l^*} = \frac{p_{L^*}^{inc} * L^*}{p_{L^*}^{dec} * L^*} = \frac{\alpha_{inc}}{\alpha_{dec}} = \rho \tag{1.9}$$

With manipulation, we can compute L^* as follows,

$$L* = N_s - W_l^* \approx N_s - \frac{N_r}{1+\rho} \tag{1.10}$$

where N_s is the totol number of seeds in the arena and N_r is the total number of robots works in the arena.

Question 1.16

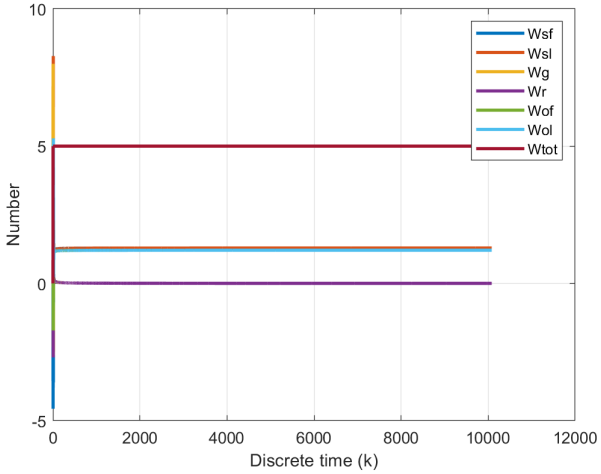


FIGURE 1.1
Time evolution of the number of robots in each state.

Question 1.17

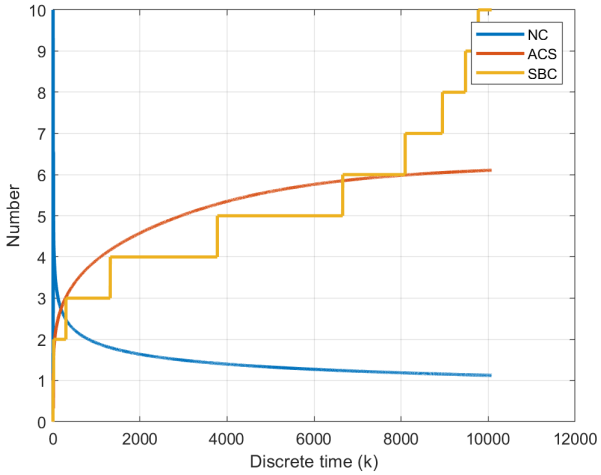


FIGURE 1.2
Time evolution of the NC, ACS and SBC metrics

Question 1.18 As derived in *Question 1.15*, $L^* = N_s - N_r/(1+\rho) = 7.5$ for $\rho = 1$, $N_s = 10$ and $N_r = 5$. This value (i.e. 7.5) is different from the one we get in figure 1.2 (i.e. 6). Nonetheless, we notice that we are getting closer to the steady state value derived earlier. We could increase the simulation duration to reach the steady state.

Question 1.19 There are some notable differences between the microscopic and macroscopic models. The convergence is to terminal values of NC, ACS, SBC doesn't follow the same trend. The microscopic model include webots implementation characteristics that heavily influences the results. In the macroscopic model, we use the mean field approach which allows us to make simplifications and deal with average and not discrete values.

CHAPTER 2

PART 2

Question 2.1 : In PSO, the main variables that are responsible for the trade-off between exploration and exploitation are the weights for the velocity inertia, the personal best and the neighborhood's best. The higher coefficients for the personal best and the neighborhood's best, the more we exploit previously found results. On the other hand, the higher the velocity inertia, the higher the exploration. It enables the particles to get out from a local minima and thus evaluate still unexplored regions.

Question 2.2 : The OCBA method is used for highly noisy problems, in which we then try to evaluate particles that render fitness's with high variances and that have a high mean. If we assume that the noise is negligible, the fitness function should be the same if we evaluate the function twice for the same particle. Thus we won't observe any variance in the fitness results for the same particle. So, if the noise is negligible, there is no use for OCBA.
For a heterogeneous problems we can use multiple robots to evaluate different particles in parallel. However, this comes with a downside. The more robots we have, the more noise we create because they can interfere between each other and they cannot have the same initial positions. Thus the same particle evaluated on two different particles won't render the same fitness value.

Question 2.3 : The implemented solution-sharing strategy is *public-group-heterogeneous*. First of all, it is public because we have one supervisor *pso_sup.c* for all robots. We can see that the fitness is evaluated in *group* because both robots have the same fitness value. Finally, it is heterogeneous because robots optimize different sets of weights of size $DATASIZE/2$. the advantage of having this solution-sharing strategy is that robots can have different behaviors while sharing solutions. Still,it has a major disadvantage which is its non-scalability. Each particle encodes a different behavior for the a given robot so if we increase the number of robots, we will have more parameters to optimize. The dimension of the search space will increase.

Question 2.4 : The best solution-sharing strategy is *public-individual-heterogeneous* because the task requires for a robot to get out of the maze. The robots optimize the same behavior but do not require collaboration. In this experiment, we are using multiple robots to accelerate the simulation.

Question 2.5 : The changes needed are the following : 1. we need to send different weights to the robots, 2. we need to decrease the search space dimension by 50% (i.e. we divide by 2), in *pso_sup.c*.

Question 2.7 : The modifications in are the following :

- The proposed version is in lines 211-218 in *pso_sup.c*.
- We set the $ROBOTS = 2$ in line 10 in *pso_sup.c*. This allows to use the two robots for fitness evaluation. This is an individual fitness evaluation.

- We divide $DATASIZE$ by 2 in line 12 in *pso.h*.This is done because the previous solution-sharing strategy increased the dimension by 2.

The Formulation of the individual fitness function is

$$fit = (1 + dist(robot, maze\_exit)^{-1} * f_{obs}$$

where $dist(robot, maze\_exit)$ computes the euclidean distance between the position of the robot and the exit of the maze and f_{obs} is the obstacle avoidance fitness proposed by *Floreano and Mondada* and implemented in *obs_con.c*. This fitness is inspired from the one proposed in the *Homework 1 part 3* and lab 10. It allows us to reward a solution that helps the robot to reach the end of the maze and also perform obstacle avoidance.

Question 2.8 : The best fitness values obtained are 0.668757 from iteration 0 to iteration 6, and 0.723276 from iteration 7 to iteration 9.

- iterations 0 and 1 : 0.240681
- iterations 2 and 3 : 0.266478
- iterations 4 to 6 : 0.272657
- iterations 7 to 9 : 0.367664

Question 2.9 : The sources of the noise are the sensors, the actuators, the search space and the randomnly initalised robot position. To maintain the same number of evaluations for both strategies, we set $ITS\_COEFF = 2$ when $Noisy = 0$. Additionally, we add a supplementary performance evaluation in *pso.c* in line 136.We re-evaluate the performance of previous bests and average them to get the noise resistant version of pso.

Question 2.10 : TODO

Question 2.11 : To test a flocking algorithm in real life, we would first of all need a flock of robots and a supervisor under the form of a computer. The supervisor can have the precise location of each robot by using a camera which films the field. Additionally, we would need an emitter on the computer and a receiver on each robot, to broadcast the location and particles from the computer to the robots.

Question 2.12 : Since the point of flocking is to get a multitude of robots to create a flock, the fitness function has to be evaluated by looking at the quality of the flock. Since every particle will be evaluated by measuring the performance of the whole group, we can only measure one particle at a time. Thus, the whole flock is the entity which evaluates a particle and having a private solution sharing doesn't make sense.

Question 2.13 : In our case the supervisor should broadcast the same weights to all the robots. Thus we only need one emitter on the supervisor and a receiver on each e-puck which are all tuned on the same channel. For technical reasons we did not manage to only use one emitter with one channel. The solution we came up with is to use 5 emitters on the supervisor which are all just sending particles to one single e-puck receiver. Since they send the same weights, this has the same effect as a single broadcast. Even though it is a less elegant solution, it is the only one we got to work properly.

Question 2.14 : For the flocking all the robots should have the same primitive behaviors and thus the problem is homogeneous. The best way to measure the quality of the flock is by looking at the whole group and thus the supervisor should compute the fitness.

Question 2.15 : To evaluate the fitness of a flock we use the same fitness function already used in the first homework. We used this function because first of all we have already tested it and know that it computes

high values for good looking flocks and secondly because we tried also other fitness function during the process that did not perform as well as this one. This fitness function can actually be broken down into 3 sub-fitness functions. The first sub-function evaluates the orientation alignment of the robots:

$$o[t] = \frac{1}{N}\left|\sum_{k=1}^{N} e^{i\psi_k[t]}\right|$$

where ψ_k represents the absolute heading of robot k.

The second sub-metric measures how well the distance between the robots is respected:

$$d[t] = \left(1 + \frac{1}{N}\sum_{k=1}^{N} |dist(x_k[t], \bar{x}[t]) - rule1_{thres}|\right)^{-1}$$

The last sub-function measures how well the flock moves towards the migration goal. It computes the projection of the velocity vector onto the vector linking the flock's center of mass to the migration goal:

$$v[t] = \frac{1}{v_{max}} max(proj_m\left(\frac{\bar{x}[t] - \bar{x}[t-1]}{\Delta T}\right), 0)$$

The complete fitness of the flock can thus be expressed as follows:

$$Fit[t] = o[t] \cdot d[t] \cdot v[t]$$

Question 2.17 : The main difference in the `flock_super_pso.c` file was made to the `calc_fitness` function. We only compute the fitness function for the second part of the simulation to not take into account the beginning where they are aggregated by default. You can find the pseudo-code on the last page of the report.

For the `collective_pso.c` file the main changes come in the function. We keep the main structure as for the `collective.c` controller of the first homework, but we and a loop which checks on the receiver queue. If new weights are in the queue, the new Reynolds controller changes the old weights for the new ones.

Question 2.18 : Unfortunately, training the flock on the same terrain and with the same initial conditions tends to result in overfitting that specific environment. To counteract that we should include some noise in the evaluation of a particle. The simplest manner we came up with is to initialize the robots with a random orientation. We could also have randomly added obstacles that change place for every iteration.

Question 2.19 : In the function that repositions the e-puck onto their initial position, we simple add the line `initial_rot[rob_id][3] += rnd()`. This will add some noise to the initial heading. The video submitted shows how the e-pucks recover from an noisy initial heading.

Algorithm 6 calc fitness

Input: weights[ROBOTS][DATASIZE],fit[ROBOTS], its, numRobs
Output:

```
1:  function CALC_FITNESS(robot_id)
2:      for i = 0 → numRobs do
3:          reposition(rob_id)
4:          initialize_migration()
5:          for j = 0 → DATASIZE do
6:              buffer[j] = weights[j];
7:          end for
8:          emit()
9:      end forreset_physics()
10:     for k = 0 → its do
11:         update_migration()
12:         send_poses()
13:         if k ≥ its/2 then
14:             flock_fit = compute_flocking_fitness()
15:             avg_fit += flock_fit
16:         end if
17:     end for
18:     avg_fit /= its
19: end function
```