Final Project of INFO-H-414
Swarm Intelligence

Second Session

June 27, 2020

1 Modality

The exam is divided in two parts:

Project + presentation You have to provide the required deliverables for the project described in Section 2.3 of this document. In addition, you will present your project in a **7-minute talk**, followed by a question-answer session. We strongly encourage you to focus **only** on presenting your ideas (how you solved the problem), your results and findings. The project and the presentation will account for up to 10 points of your final grade.

Oral examination You will be asked a number of questions concerning *the entire course material*. This will account for up to 10 points of your final grade.

To pass the exam, both parts must be above the threshold.

1.1 Timeline

- The project submission deadline is **August 9** at 23:59.
- Delay on the submission will entail a penalty of 1 point every 12 hours of delay on the final evaluation of the project. Maximum delay is **August 12**, at 23:59. After this deadline you fail the exam.

1.2 Project Deliverables Guidelines

For the project, you will have to submit via the Assignments of TEAMS[1] three elements:

[1]Please, contact the assistants if you encounter problems with TEAMS.

- Your code in digital format, so we can test it (more details on the format are given in the description of the project).
- A short document (**max 7 pages** not counting the references[2]) written in English that describes your work. You have to explain both the idea and the implementation of your solution.

  The document must be typeset using the template of Lecture Notes in Computer Sciences[3] (LNCS – Springer), available at the TEAMS assignement in the file: Templates_report.zip

  The format of your report will be that of a scientific article, including abstract, introduction, short description of the problem, description of the solution, results, conclusions and references (a few examples on how to structure your project document are included in the file: Example_scientific_articles.zip, also attached to the assignment).
- Two videos of your swarms while performing the mission (one for the manual solution, one for the automatically optimized solution). The speed of the video must be adjusted so that the length of each video does not exceed one minute.

On the day of the oral examination, you are expected to present your project using slides that you will be shown via screen sharing, followed by a question-answer session. In your presentation, focus on your solution and your results, you do not need to describe the project itself. Because of the screen sharing, your slides might not be very readable, therefore, we advise to use as much graphs and plots as possible to illustrate your points. The rest of the exam will consist of a question answering session on all the subjects covered by the course.

1.3 General Remarks

Apply what you learned — It is very important that you stick to the principles of swarm intelligence: simple, local interactions, no global communication, no global information.

Avoid complexity — Good solutions are elegantly simple in most cases. So, if your solution is becoming too complex you are very likely in the wrong direction.

Honesty pays off — If you took inspiration from scientific papers or websites, cite the source and say why and how you used the idea.

[2] You can add as many pages of bibliographical references as you need to support your work.

[3] `https://www.springer.com/gp/computer-science/lncs/conference-proceedings-guidelines`

Cooperation is forbidden — Always remember that this is an individual work.

The project counts for 50% of your final grade. The basic precondition for you to succeed in the project is that it must work. If it does not, the project will not be considered sufficient. In addition, code must be understandable — **comments embedded in the software are mandatory**.

The document is very important too. We will evaluate the quality of your text, its clarity, its completeness, and its soundness. References to existing methods are considered a plus —honesty *does* pay off! More specifically, the document is good if (i) it contains all the information needed to reproduce your work without having seen the code, and (ii) a good and complete analysis of the results.

The oral presentation is also very important. As opposed to the report, a good talk deals with *ideas* and leaves the technical details out. Be sure that it fits in the 7-minute slot. Because you only have 7 minutes for your presentation, you should focus on presenting what you have done and the results you obtained, rather than on the presentation of the problem.

2 Project

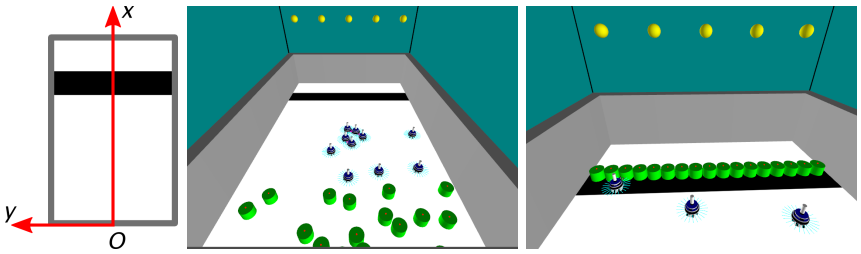
The project is composed of two parts (two subproblems if you like): one deals with a problem arising in swarm robotics and the other with one arising in optimization. However, both parts are equally important and comprise one single project. Therefore, in order to deliver a complete project, you have to provide a solution that tackles the two parts.

2.1 Swarm robotics: *Collective Building*

The goal of the project is to develop a *collective building* behavior. Collective building is a common activity in nature. Many social insects (e.g., ants, bees, termites) are capable of building amazing structures in a completely distributed, robust and flexible way. Such collective building behaviors are very interesting for swarm robotics, as they can be used as an inspiration for simple distributed behaviors.

2.1.1 Problem definition

The environment is an area in which several objects scattered. Blocks can be seen by the robot using their camera. A robot can physically pick up a block using its gripper. The objects are located in the bottom part of the environment; robots are located north of the initial location of the objects; the exact positions of the robots and of the blocks are chosen at random each time you start an experiment. The area of the environment, characterized by a black ground color, is where the wall must be built.



2.1.2 Goal

Your goal is to develop a behavior that lets the robots build a wall. Ideally this wall should divide the corridor in two areas, that is, once the wall is built, no robot can move from one area to the other.

Different performance metrics can be used to evaluate the quality of the wall. In addition to the number N of objects that are part of the wall (that is, that are in the black area), ARGoS automatically computes three quantitative numerical measures: the *thinness* of the wall, the *compactness* of the wall, and the *coverage*. An experimental run lasts for 10 000 time steps. After the experiment, ARGoS outputs the values of these 4 metrics (in the simulator or in the console, depending on whether the visualization is activated or not—see Section 2.1.4).

Thinness is defined as

$$\begin{cases} 0 & \text{if } N = 0, \\ 1 - \dfrac{A}{B} & \text{otherwise,} \end{cases}$$

where A is the standard deviation of the x coordinates of the objects within the construction area, B is the size along the x axis of the construction area ($B = 0.7\,\text{m}$). Thinness takes values in $[0, 1]$. You must maximize this measure.

Compactness is defined as

$$\begin{cases} 0 & \text{if } N < 2, \\ \dfrac{G_y}{D} & \text{otherwise,} \end{cases}$$

where G_y is the maximum distance along the y axis among the objects within the construction area, and D is the diameter of a construction object ($D = 0.2\,\text{m}$). Compactness is best when it is close to 1. A value lower than 1 means that the maximum distance (projected onto the y axis) between the center of the objects is smaller than D. A value larger than 1 means that the objects are too spread out. For the sake of this project, values slightly lower than 1 are preferable to values slightly larger than 1.

Coverage is defined as

$$\begin{cases} 0 & \text{if } N = 0, \\ \dfrac{C}{S} & \text{otherwise,} \end{cases}$$

where C is the projection of all the objects in the construction area on the Y axis, and S is the size of the construction area along the y axis ($S = 3.6\,\text{m}$). Coverage takes values in $[0, 1]$. You must maximize this measure.

Swarm composition The swarm comprises 10 identical robots. The robots are equipped with the following sensors and actuators:

- **wheels**: to move in the environment;
- **LEDs**: to communicate with the neighbour robots;
- **omnidirectional camera**: to detect LEDs, that may be useful to (i) identify the rooms, (ii) count the objects in a room, and (iii) read the LED color of the neighbour robots;
- **ground sensor**: to read the color of the floor;
- **range-and-bearing**: to locate nearby robots, and send and receive messages;
- **gripper**: to carry the objects.

2.1.3 Solution

Multiple solutions are possible. However, solutions that are not swarm-based will be considered as poor. An non swarm-based solution is, for example, a solution in which the robots pick an object up and then perform random walk until they luckily find the construction area. HINT: have a look at social odometry[4] or, even better, implement a solution based on DOL.

2.1.4 Setting up ARGoS

- Download the experiment files: SR_Project_H414.tar
- Unpack the archive into your $HOME directory and compile the code

```
$ tar -xzf SR_Project_H414.tar.gz   # Unpacking
$ cd SR_Project_H414                # Enter the directory
$ mkdir build                       # Creating build dir
```

[4] http://www.robolabo.etsit.upm.es/aguti/publications/GutCamMon-etal12IntModRob.pdf

```
$ cd build                              # Entering build dir
$ cmake ../src                          # Configuring the build dir
$ make                                  # Compiling the code
```

- Set the environment variable ARGOS_PLUGIN_PATH to the full path in which the build/ directory is located:

  ```
  $ export ARGOS_PLUGIN_PATH=$HOME/SR_Project_H414/build/
  ```

  You can also put this line into your $HOME/.bashrc file, so it will be automatically executed every time you open a terminal.

- Run the experiment to check that everything is OK:

  ```
  $ cd $HOME/SR_Project_H414       # Enter the directory
  $ argos3 -c col-building.argos   # Run the experiment
  ```

If the usual ARGoS GUI appears, you're ready to go.

Switching the visualization on and off. The experiment configuration file allows you to launch ARGoS both with and without visualization. When you launch ARGoS with the visualization, you can program the robots interactively exactly like you did during the course. Launching ARGoS without the visualization allows you to run multiple repetitions of an experiment automatically, e.g., through a script. By default, the script launches ARGoS in interactive mode. To switch the visualization off, just substitute the visualization section with: <visualization />, or, equivalently, comment out the entire qt-opengl section.

Loading a script at init time. When you launch ARGoS without visualization, you cannot use the GUI to set the running script. However, you can modify the XML configuration file to load automatically a script for you. At line 49 of col-building.argos you'll see that the Lua controller has an empty section <params />. An example of how to set the script is at line 52 of the same file. Just comment line 49, uncomment line 52 and set the script attribute to the file name of your script.

Changing the random seed. When you want to run multiple repetitions of an experiment, it is necessary to change the random seed every time. To change the random seed, set the value at line 12 of col-building.argos, attribute random_seed.

Changing the duration of the experiment. You can reduce the duration of the experiment if you observe that the robot swarm requires less than 10 000 time steps to accomplish the task. You can set the value at line 12 of col-building.argos, attribute length, to the desired value (by default is set to 1000 seconds, that translates to 10 000 time steps). Changes in the duration of the experiment must be justified.

Making ARGoS run faster. Sometimes ARGoS is a little slow, especially when many robots and many sensors are being simulated. You can make ARGoS go faster by setting the attribute threads at line 9 of col-building.argos. Experiment with the values, because the best setting depends on your computer.

2.2 Optimization: *Fine-tuning a Robot Swarm Controller*

Deciding the best value for the parameters of an algorithm is a difficult task. Manual approaches based on trial-and-error take a lot of time and often deliver poor results. Luckily, this problem can be treated as an optimization problem where the parameters are the variables of the solution and the objective function value of each solution is evaluated by running the algorithm.

In this part of the project, you will use a variant of particle swarm optimization (PSO) to optimize the parameters of the robot controller created in the swarm robotics part of the project. To do this, you first need to identify all parameters in the controller and then select those that could improve the behavior of the swarm once they are correctly fine-tuned. An example of such parameter could be a real variable $p \in [0, 1]$ that activates a behavior of the robots with probability p. In your controller, you may have set $p = 0.5$, but PSO might determine that 0.34 is a better value.

2.2.1 Goals

The goals are the following:

1. Problem definition. Many industrial problems can be cast as optimization problems. The definition of the optimization problem is normally a non-trivial task. In this part of the project, you will cast the problem of optimizing the performance of a robot swarm as a continuous optimization problem suitable to be solved with a PSO algorithm.

2. Implementation of an efficient optimization algorithm. Once the optimization problem is defined, there is still the problem of choosing an optimization algorithm capable of solving it. In this part of the project, you will implement a PSO variant called Frankenstein's PSO that is made up by different algorithmic component proposed for PSO.

3. Integration of systems. In many real scenarios, an optimization algorithm resides in a separate piece of software that needs to be integrated with the application software. It is not rare that this application software is a simulator of some sort. In this part of the project, you will integrate your PSO code with the ARGoS simulator.

2.2.2 Requirements

You will start from the controller that you manually designed for the previous part of the project. We recommend that you duplicate your solution: one could be called *manual_solution.lua* and consist in your solution for the swarm robotics part of the project; and the other could be called *pso_solution.lua* and consist in your solution modified slightly in order to work with PSO. Using Frankenstein's PSO algorithm, you have to optimize the parameters of your robot controller considering the experimental setup described in Sect. 2.1 (that is, the same evaluation function and stopping criterion.)

The optimization algorithm needs an objective function that evaluates how well the swarm performs the mission at hand. You are free to choose the objective function that you deem relevant. We recommend that you use one of the metrics described Section 2.1.2, or a (weighted) average of multiple ones. This objective function is extremely important for this part of the project, and should therefore be justified and not taken lightly. Also, you should use the same objective function for the evaluation of the manual and optimized solution (more on that in Section 2.3).

Another important step in this part of the project is to identify all parameters in your controller. In some cases, this task can be difficult, especially for the person who wrote the code. A good place to start is to consider all constants defined in your code: there is a good chance that they are just hard coded parameters. Among all parameters that you have identified in your controller, you will choose **at least five** and **up to ten** of them to optimize. Remember that your choice should be meaningful and that it will have to be justified in your project report. Once you have chosen the parameters you want to optimize, replace every occurrence of this parameter in you controller (*pso_solution.lua*) by a global variable (initialized at the begining of the Lua file).

You have to create the interface between the PSO code and the ARGoS simulator. This interface should take the parameter configuration from PSO; set the parameter values in the controller Lua file; launch the simulation with ARGoS and, at the end of the simulation, return the robot swarm evaluation to PSO.

The variant of PSO that you will use for this part of the project is Frankenstein's PSO [1], which combines algorithmic components that have showed to be advantageous in terms of speed and reliability for tackling a

number of complex problems. The two main components of Frankenstein's PSO are: (i) a fully informed model-of-influence and (ii) a time-varying population topology.

Regarding component (i), you will use the velocity update rule of the fully informed model-of-influence, which is defines as follows:

$$\vec{v}_{t+1}^{\,i} = \chi\left(\vec{v}_t^{\,i} + \sum_{k=1}^{n} \varphi_k U_{kt}^i \left(\vec{p}_t^{\,k} - \vec{x}_t^{\,i}\right)\right) \tag{1}$$

where t is the number of the current iteration of the algorithm, $\chi = 0.7298$ is a constant value called constriction coefficient, n is the number of particles in the swarm, and $\varphi_k = 4/n$, for $k = 1, 2, \ldots, n$. The goal of the velocity update rule shown in Eq. 1 is to let all neighbors of a particle i to contribute to update its velocity instead of using only the best particle.

Regarding component (ii), you will implement a time-varying topology that starts as a fully connected one and ends as a ring. To do so, particles' connectivity should decrease as the optimization process evolves until they form a ring. There are two constraints that you need to consider when implementing the time-varying topology of Frankestein's PSO. Firstly, particles' connectivity should decrease over a number of iterations and never abruptly from one iteration to another. Secondly, while reducing particles' connectivity, all particles in the swarm should have, on average, the same number of connections.

You are free to implement the time-varying topology as you decide, but you must keep in mind that the main idea is to let particles exploit the advantages of different topologies in a single run of the algorithm. Extra points will be awarded for clever implementations of the time-varying topology in which is possible, for example, to control the speed at which the topology changes.

Setting the parameters of PSO. While PSO implementations typically have a number of parameters to set before they can be applied to a problem, the version of Frankenstein's PSO you are ask to implement only has one: the number of particles. You are free to set this value either manually, between 5 and 20, or to use an experimental methodology to do so similar to one we saw in class for ant colony optimization. Additionally, you can include the use of automatic configuration tools. Some freely available configuration tools are [3] and [2].

Running experiments with PSO and ARGoS In order to decide which parameter settings to use for your robot controllers, you have to conduct experiments using an appropriate number of repetitions. We recommend that you perform at least 10 runs of PSO using different random

seeds. It is very important that you remember to control the random seed. That is, use the same seed for both ARGoS and PSO in each run.

2.3 Deliverables and Evaluation

The deliverable will be a zip file containing the following items (please create a folder for each one):

- Report
- Code + README
- Results
- 2 videos (one for your manual solution, one for the optimized solution)

Report The final part of this project consist in writing a report of up to 7 pages. In this report, you should describe both the controller you manually designed to solve the swarm robotics mission (functioning, finite-state machine representation, evidence of swarm robotics principles) and the optimization process setup (the objective function, the PSO algorithm, parameter settings, representation of solutions, perturbation mechanism, topology, etc.). To better manage your space, you should keep the description of the problem as small as possible and focus on describing what you did.

You should also test and analyze the performance of your work. That is, results must be contrasted with the design choices made in the development of the control software for the robots and with the setup of the optimization process. The experimental methodology is the following:

1. Execute 10 runs of PSO. For each run, collect the best solution found by PSO.
2. Evaluate the 10 solutions produced by PSO 10 times, and compare the performance of the solutions using the Wilcoxon rank-sum test. Display these results via boxplot.
3. Based on the statistical tests, determine the best solution among the 10 produced by PSO. If the statistical tests are not conclusive, either perform more evaluations until you can observe statistical differences, or select the solution with the best average performance.
4. Evaluate the best solution found by PSO, and your initial solution 20 times. Display these results via boxplot.

Remember to use the same initial conditions when evaluating the solutions, otherwise they will not be comparable. In ARGoS, the initial configuration of an experiment is controlled by the seed of the configuration file (.argos). Therefore, if you want to evaluate your solutions ten times, you will have

to use ten different random seeds and replace them after each run in the configuration file.

OPTIONAL: In addition to the analysis of the performance of your solutions in the original conditions given to you, you can also study their performance under different conditions you deem interesting. To do so, you can modify the attributes of the .argos file. For example, you can change the number of robots, or the number of objects.

Code You have to submit your code following the following guidelines:

- The Lua script that you developed, commented and well-structured.
- The source code of your PSO implementation, well-commented and properly indented. (You can implement PSO using the programming language you like as long as it runs on Linux.)
- You must include a README file with the instructions and specifications of how to compile, install and execute your PSO implementation to configure the controller.
- You must implement your own code. **Plagiarism will be strongly penalized**.

Results

- Create a *csv* file called *psoResults.cvs* with the results from the runs of PSO, and another one called *psoVsManual.cvs* with the results from the 20 evaluations of the best configuration found by PSO vs your initial solution. Also, include a copy of all your box plots and statistical tests carried out to analyze the performance of your implementations. Use the structure {pso|psoVsManual}-{bxp|wt}.* to name the files according to the case, where bxp stands for boxplot and wt for Wilcoxon test.

3 Contacts

Marco Dorigo	mdorigo@ulb.ac.be	for general questions
Mauro Birattari	mbiro@ulb.ac.be	for general questions
Christian Camacho Villalón	ccamacho@ulb.ac.be	for PSO
Federico Pagnozzi	federico.pagnozzi@ulb.ac.be	for PSO
Antoine Ligot	aligot@ulb.ac.be	for swarm robotics
David Garzon Ramos	dgarzonr@ulb.ac.be	for swarm robotics

References

[1] Marco A Montes De Oca, Thomas Stutzle, Mauro Birattari, and Marco Dorigo. Frankenstein's pso: a composite particle swarm optimization al-

gorithm. *IEEE Transactions on Evolutionary Computation*, 13(5):1120–1132, 2009.

[2] Frank Hutter, Holger Hoos, Kevin Leyton-Brown, and Thomas Stützle. Paramils: An automatic algorithm configurationframework. *Journal of Artificial Intelligence Research*, 36:267–306, October 2009.

[3] Manuel López-Ibáñez, Jérémie Dubois-Lacoste, Leslie Pérez Cáceres, Thomas Stützle, and Mauro Birattari. The irace package: Iterated racing for automatic algorithm configuration. *Operations Research Perspectives*, 3:43–58, 2016.