1 Lab 6: PSO for Benchmark Functions and Single Robot Systems

This laboratory requires the following (the development tools are installed in GR B0 01 and GR C0 02 already):

- C development tools (gcc, make, etc.)
- Webots simulation software
- Webots User Guide
- Webots Reference Manual
- SwarmViz

Depending on your programming skills, the laboratory duration might require an effort of up to four hours although assistance will be provided in the computer rooms during a time window of three hours. Although this laboratory is not graded, we encourage you to take notes during the course of this laboratory to aid in preparing the final exam. A solution to this lab will be posted after the lab session.

1.1 Office hours

Additional assistance outside the lab period (office hours) can be requested using the dis-ta@groupes.epfl.ch mailing list.

1.2 Information

In the following text you will find several exercises and questions. The type of question is indicated between parentheses:

- The notation S means that the question can be solved using only additional simulation.
- The notation Q means that the question can be answered theoretically, without any simulation.
- The notation I means that the problem has to be solved by implementing a piece of code and performing a simulation.
- The notation B means that the question is optional and should be answered if you have enough time at your disposal (the bonus questions can involve a maximal effort of 20 points)

To prepare you for the exams and to permit you better time planning during the exercise session, we also show an indicative number of points for each exercise between the very same parentheses. The combined total number of points for each laboratory exercise is 100, without bonus questions

1.3 Optimization

In many instances, we want to find either the minimum or maximum value of some function. If the function in question is complex (e.g., non-convex, discontinuous, multiple peaks) or even unknown, it is often impossible to accomplish

this task using exact methods. Furthermore, if the function parameter space is very large, a systematic search of this space is often too computationally expensive. Therefore, a number of metaheuristic techniques have been developed to find near-optimal solutions, such as Genetic Algorithms (GA) and Particle Swarm Optimization (PSO).

1.4 The Particle Swarm

To find minima/maxima, repeated evaluations of the function in question must be done at different points. A good optimization algorithm will use the information obtained from these evaluations to choose the locations of future evaluations, and eventually to decide where the minimum/maximum is.

In PSO, a set of "particles" is initialized with a random position for each particle. Each particle is also assigned a random "velocity" in the search space. Particles are then "flown" through the search space, moving at their respective velocities.

Optimization is achieved by each particle remembering at what position it achieved the best evaluation of the function, or the particle's "personal best". Particles also remember the best achieving position of their "neighborhood". The neighborhood for some particle A is a group of particles to which A belongs. This group can be, for instance, topological (whatever particles are closest) or index-based throughout the algorithm. It can consist of either a subset of the particle swarm (local neighborhood), or the entire population (global neighborhood). The "neighborhood best" of a neighborhood is the position that yielded the best evaluation by *any* particle in the neighborhood.

At each iteration of the algorithm, the velocity of each particle is updated, using a randomized attraction to both the particle best and the neighborhood best. This allows particles to move towards areas of the search space that have yielded good results, and in doing so, discover nearby better results. In this way, the particles will eventually converge on possibly global optima. The idea for this type of behavior took inspiration from the ways birds act while flying in a flock and searching for food.

2 Lab: Using PSO

2.1 Understanding the Main Loop

Download `lab06.tar.gz` from Moodle and unarchive it:

```
$ tar xvfz lab06.tar.gz
```

This will create one directory called `webots` which is used for part 2 of the lab. For this first part of the lab you will need to work with SwarmViz, a PSO visualization tool for educational purposes developed at DISAL (Jornod et al., 2015). To run this visualizer go through the following steps:

1- In your `lab06` folder clone the repository of SwarmViz located at https://github.com/epfl-disal/SwarmViz.git by typing the following command in the terminal (make sure you are in `lab06` directory):

```
$ git clone https://github.com/epfl-disal/SwarmViz.git
```

This will create a folder called `SwarmViz` in your current directory. You can find more information about this package in the `README.md` file or in the opening page of this repository on GitHub.

2- Go to the directory `SwarmViz` and run the configuration script using the following commands:

```
$ cd SwarmViz
$ ./config
$ make clean all
```

3- Run the program using this command :

```
$ ./bin/swarmviz &
```

4- Copy the `doc` folder in `lab06` into the `SwarmViz` folder. You can launch the documentation pages by typing (you should be in `SwarmViz` directory):

```
$ cd doc/html
$ firefox index.xhtml
```

You can refer to the `README.md` file for further documentation for this package. You are now able to run the program as well as having access to the source code of the software. Go to the `SwarmViz/src/pso` folder where you can find the relevant files for the PSO algorithm. This code provides a complete implementation of PSO.

1. **(Q, 5)** Browse through the `swarm` and `particle` source and header files to get an understanding of the structure of the code. Additionally, you can use the documentation to understand what different parts of the code are doing.

2.2 PSO on the Sphere and Rastrigin Functions

We will explore how PSO behaves when applied to optimizing two different functions: the Sphere and the Rastrigin function (see Figure 1).

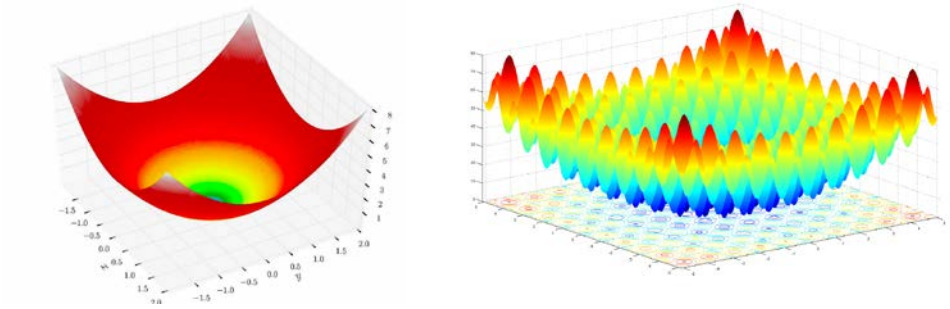


Figure 1: The Sphere function (left) and the Rastrigin function (right)

The Sphere function is defined as follows:

$$f(\bar{x}) = \sum_{i=1}^{N} x_i^2$$

and the Rastrigin function as:

$$f(\bar{x}) = 10N + \sum_{i=1}^{N} (x_i^2 - 10\cos(2\pi x_i))$$

We want to minimize the functions, therefore the optimal value of both functions is 0. In the optimization process, we want to find the set of x_i that give the value of f closest to 0 (given by $x_i = 0$ for all i). The values for x_i are initially randomly distributed between -10.24 and 10.24. Before starting the optimization, the parameters of PSO should be set according to the problem. The default values of the parameters are reported in Table 1.

Table 1- Default values for PSO parameters used in SwarmViz

Parameter Name	Default Value
Noise	0.00
Dimension of the search space	24
Number of particles	30
Minimum initial value	-5.12
Maximum initial value	5.11
Maximum velocity of particles	5.12
Inertia	0.60
Maximum number of iterations	1500
Local (personal best) weight	2.00
Neighbor (neighborhood best) weight	2.00
Total number of neighbors	2

Before starting the optimization, click on the *Plots* tab on the right hand side panel of *SwarmViz* graphical user interface (GUI). You can see a number of different plot types. Activate only *Fitness Landscape, Euclidian Distance* and *Fitness* plots. Take a look at the other tabs of the panel briefly. You will find where to change different parameters of PSO along with a number of different benchmark functions that can be optimized.

The three mentioned plots depict the 1) distribution of particles in the search space, 2) the mean interparticle distance at each iteration for all particles in the swarm (an indicator of compactness and convergence of the swarm), and 3) the average and the best fitness of all the particles in each iteration respectively. Note that only two dimensions of the fitness landscape are shown whereas our problem's search space could have many more dimensions. You can pause the simulation and zoom out on this plot to get a more global view of the landscape.

To start the optimization, choose the desired benchmark function and set it as the fitness function in the dropdown box in the *Swarm* tab. Go to the Simulation tab and click the *Simulate* button. Clicking this button again will cause the simulation to stop. Keep in mind all the parameters of the problem (PSO parameters, fitness function

type and dimension, simulation parameters) are initialized to a default value unless you modify them in the GUI before starting the optimization.

You can also see the best fitness value along with the coordinates of the particle with the best fitness in the search space printed to the terminal for each step. You can stop the simulation and zoom into each of the plots if needed. Additionally, you can speed up the visualization by reducing the delay in the Simulation tab.

2. **(S, 5)** Starting from the Sphere function, we will use PSO with 30 particles and index-based neighborhood which is given by the three nearest particles on each side (e.g., particle 8 has neighborhood {5, 6, 7, 8, 9, 10, 11}) with particle 29 being next to particle 0. We will optimize the function in a ten dimensional space; in other words, a particle is a set of 10 numbers, and the closer to 0 all the numbers become, the better the performance. The maximum velocity of the particles should be set to 0.5 and their inertia to 1.0. Set the simulation parameters accordingly and then run the program with 10 iterations, 100 iterations, 1000 iterations, and 10000 iterations. You can modify the parameters values in the Swarm tab. You can save the plots in a folder specified as the *Output directory* in the *Files* tab of the visualizer for comparing the results more easily.

3. **(Q, 5)** What do you observe in the final fitness? How many iterations do you think will be necessary to get the exact answer? Why?

4. **(S, 5)** There is a variable that controls the maximum velocity that particles can achieve in the simulation, which you can find under the Swarm tab. Using 1000 iterations, try varying maximum velocity from 0.5 to 4.0.

5. **(Q, 5)** How does the performance change?

It can be very inconvenient to have to tune maximum velocity to give the optimal performance for a simulation. A feature that was developed for PSO soon after its invention was an “inertia” coefficient. This was a coefficient typically less than 1 which is multiplied with the velocity at each iteration, in order to naturally slow particles down without imposing a hard velocity threshold (so far you were running the PSO with a value of 1 for inertia).

6. **(S, 5)** Run the program again, setting maximum velocity to 4.0 and the inertia coefficient to 0.6.

7. **(Q, 5)** How does this modification of inertia affect the fitness? Make sure your optimization runs for sufficient number of iterations (e.g., 1000).

Having found a good set of parameters that work on the Sphere function, let us now select the Rastrigin function by choosing this function as the benchmark fitness function in the Swarm tab. Keep the number of iterations set to 1000, maximum velocity 4.0. As a reminder, the number of particles used in questions above was 30.

8. **(S, 5)** How would you set the inertia and number of particles in order to find the optimal value for the Rastrigin function? Explain your choice and provide numerical results of your tests.
Hint: *since the function has many local minima, look not only at the performance of a run, but also the position of the best particle forming the solution (its coordinates). Note that PSO is a stochastic algorithm, so it is possible that not every run yields the optimal solution even when the algorithm parameters are correctly selected.*

9. (Q, 5) What type of neighborhood topology is used in the PSO implementation? You can find this information in the `swarm.cpp` file. How would you justify this choice? Can you think of other neighborhood types? What would be the best neighbor topology for each of the Sphere and the Rastrigin function?

10. (S, 5) Try changing the *neighborhood weight* and *neighborhood size* for the two functions, what value causes the optimization to perform better? How does this affect the optimization process and which function is more sensitive to this choice? Explain your choice and provide numerical results of your tests.

2.3 PSO for Evaluative Adaptation

We have established that PSO can do a good job of optimizing mathematical functions. The next step is to test it in a more demanding environment, in particular in the presence of noisy functions.

In "Evolution of Homing Navigation in a Real Mobile Robot" (Floreano and Mondada, 1996), a GA was used to train a robot to move through a maze and avoid obstacles. We will instead, use Webots to shape robotic controllers for the mobile robots, evaluating the performance of PSO as the method for learning the parameters of the controller. For the experiments in this lab, we use the e-puck robot instead of the Khepera that was used in the original paper. The behavior we wish to achieve on the robots is obstacle avoidance, the same as in Floreano and Mondada's paper.

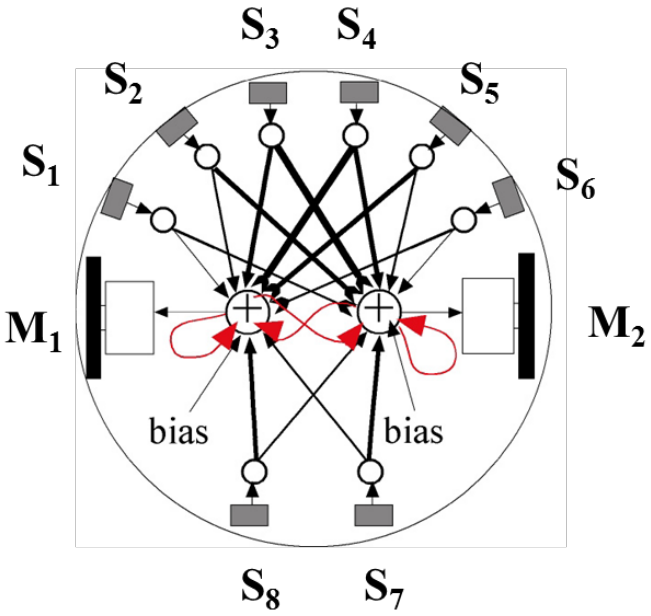
We have partially implemented the PSO algorithm in Webots for you. You can find the code in `/lab06/webots/controllers/pso_obs_sup/pso.c`.

11. (Q, 2) Open the world file `pso_obs.wbt` in webots and compile all the controllers for the robot and the supervisor. Browse through the files `pso.c` and `pso_obs_sup.c` and read the comments to get an understanding of the structure of the code and the way the candidate solutions are encoded. What are the values of PSO parameters in these codes (e.g., dimension of the search space, number of particles, etc.)?

12. (I, 8) The `pso.c` file is missing the speed and location update operations in the main loop. You can take a look at `SwarmViz/src/pso/swarm.cpp` file which uses the same concept to update speed and location. Implement this change and modify the code of this part accordingly in the marked sections.

(Note: the inertia factor is not being passed as a function parameter to `pso()`, just set it to a default value of 0.6, for now). Make sure you understand the code.

The worlds we are using here are simple walled arenas with no obstacles and one e-puck robot. We use a two neuron, single-layer neural network to control the robot, with proximity sensors as inputs and the motor speed as outputs. There are recursive and lateral connections from the outputs of both neurons back into each other. The controller we adapt is the set of weights for these neurons (8 proximity sensors + 2 recursive/lateral connections + 1 bias = 11 weights for each neuron => 22 weights total).



13. (I, 10) What would be an appropriate fitness function for evaluating obstacle avoidance behavior? Open the robot controller `your_obs_con.c` and implement your fitness function in the designated portions of the code. Then run the simulation until the optimization is over and observe the final behavior. The default time for a single evaluation of fitness is ~30 s. (Note: you should try to come up with a different fitness function from the one presented in class).

We will now test shaping of obstacle avoidance behavior using PSO. In the following questions, you will be asked to run simulations that will take about 20 hours of simulated time, and several minutes of real wall-clock time to complete. In order to save time, you may want to look ahead to the next question to see whether it is something which you can work on while you wait.

In the paper by Floreano and Mondada, the fitness of the controller was measured with the equation:

$$F = V(1-\sqrt{\Delta V})(1-i), \quad V,\ \Delta V,\ i \in [0,1]$$

Where V is a measure of the average rotation speed of the two wheels over the trial, ΔV is a measure of the average difference in wheel speed of the two wheels over the trial, and i is the average activation value of the proximity sensor with the highest activity. Each of these terms in the fitness function encourage the robot to go fast, go straight, and not stay close to walls, respectively.

14. **(S, 5)** Revert the world and select the `obs_con` as the robot controller. It implements the above fitness function. Run the optimization again. How does the behavior obtained now compare with the previous version using your own fitness function?

15. **(S, 5)** Try varying the maximum particle velocity parameter defined in `pso_obs_sup.c`. Choose values between 10 and 40. In the `pso.c` try to change the value of the inertia that you have defined as requested in the implementation question **I8**. Try varying this parameter between 0 and 1. How is the performance affected?

16. **(Q, 5)** How would you estimate the performance in terms of fitness and computation time for different number of particles, significantly lower (e.g., 7) and higher (e.g., 40) than the space dimension size? What value would be a good compromise?

17. **(Q, 5)** If you have a specific time budget, where do you put your evaluation effort: number of iterations or larger particle swarm? Justify your answers.

18. **(S, 5)** Look inside the code to find how the neighborhood is specified. There are four different types of neighborhood defined and implemented in the code. What does each of them do? Run the code for each type separately and compare the behavior. Explain how the choice of the neighborhood type and size should be for a given problem.

19. **(B, 20)** Now we want to optimize the weights of the neural network for wall following behavior, instead of obstacle avoidance. Design a new fitness function for this purpose. Open the world file `pso_wall_following.wbt` in Webots and implement your new fitness function in the file `your_wall_following_controller.c`. Note that the supervisor `pso_wall_following_sup.c` uses the files `pso.c` and `pso.h` of the previous questions, because the configuration of PSO remains the same. However, other parameters of the optimization (e.g., duration of optimization) can be modified in the supervisor.

20. **(Q, 5)** Why changing the optimization problem does not require any change in the PSO configuration?

3 References

Jornod, G.; Di Mario, E.; Navarro, I.; Martinoli, A., "SwarmViz: An open-source visualization tool for Particle Swarm Optimization," in 2015 IEEE Congress on Evolutionary Computation (CEC), pp.179-186, 25-28 May 2015

Floreano D. and Mondada F., "Evolution of Homing Navigation in a Real Mobile Robot", IEEE Trans. on System, Man, and Cybernetics: Part B, 26(3): 396-407, 1996.

Pugh, J., Zhang, Y., and Martinoli, A. “Particle Swarm Optimization for Unsupervised Robotic Learning”, In Proc. of the IEEE Swarm Intelligence Symposium 2005, Pasadena, CA, pp. 92-99.