1 Lab 9: PSO for Multi-robot Systems

This laboratory requires the following:

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

The laboratory duration is about five hours. Although this laboratory is not graded, we encourage you to take notes during the course of this laboratory to aid in preparing the corresponding lab verification test and 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 notation $\mathbf{S_x}$ means that the question can be solved using only additional simulation.
- The notation $\mathbf{Q_x}$ means that the question can be answered theoretically, without any simulation; if you decide to write a report, your answers to these questions should be submitted in your report. The length of answers should be approximately **two sentences** unless otherwise noted.
- The notation $\mathbf{I_x}$ means that the problem has to be solved by implementing a piece of code and performing a simulation.
- The notation $\mathbf{B_x}$ means that the question is optional and should be answered if you have enough time at your disposal.

To prepare yourself for the exam and to allow you for better time planning during the exercise session, we show an indicative number of points for each exercise between parentheses. The combined total number of points for the laboratory or homework exercises is 100.

2 Multi-robot PSO for obstacle avoidance

Using PSO on a single robot, it can take a long time to automatically design high-performing controllers. This is because the robot will have to test every member of the swarm (i.e., pool of candidate solutions) one after another for each iteration of the algorithm. For example, doing 100 iterations of PSO with 20 particles with each evaluation taking 1 minute means we need 2000 minutes or over 33 hours to complete!

Assuming that we want to automatically synthetize a specific individual behavior rather than a collaborative one involving multiple robots, we can leverage the parallelism of the distributed robotic system to speed up the learning process. In this

way, we can distribute the particles among the robots and evaluate their performance in parallel, thus saving a lot of time. Going back to our previous example, if we have 20 robots each evaluating a different particle, it will only take 1 minute to evaluate 20 particles, and the total adaptation time goes down to about 1.7 hours.

We will now test the shaping of an obstacle avoidance behavior using multi-robot PSO. In the following questions, you will be asked to run simulations that will take about 2-3 hours of simulated time, and several minutes of real time to complete. In order to save time, you may want to look ahead to the next questions to see if there is something which you can work on while you are waiting.

Start by downloading `lab09.tar.gz` from Moodle and decompressing it:

```
$ tar xvfz lab09.tar.gz
```

This will create a `lab09` folder containing the worlds and controllers for this lab. After loading the Webots worlds, make sure you compile all the relevant controllers.

S1: Load the `pso_obs` world. Run the world and have a look at the speed-up factor in at the top of the Webots window, while running in fast mode. As in Lab 8, the simulation will do 10 separate optimization runs and print the final fitness for each, as well as the average fitness at the end. When the runs are finished, observe the performance of the robots (they will be running the best found controller).

Q2 (5): In Lab 8 you were using a single robot to test all particles sequentially. This took ~20h of simulated time and the speed-up factor while running in fast mode was around 350. What is the improvement in simulation time? What is the improvement in wall-clock time speed-up, when moving from single- to multi-robot simulations? How do you explain these results?

Q3 (5): What kind of fitness values do you get?

I4 (10): Open the `pso.c` file in the `pso_obs_sup` directory and implement the noise-resistant version of the algorithm presented in lecture, which reevaluates the personal best and averages with the previous one. The part of the code you need to change is marked with TODO. Do not forget to set `NOISY=1`.

Q5 (5): What fitness values do you obtain in this case?

Q6 (5): What do you observe in terms of performance increase for the noise-resistant PSO when comparing to the single robot scenario in the previous lab? Explain why this happens.

Q7 (5): Have a closer look at the supervisor code, in particular the `calc_fitness()` function. How are the particles divided among the robots? Is this a homogeneous or heterogeneous team learning approach?

Q8 (5): Explain the differences between private and public policies in solution sharing.

Q9 (5): Why is a group performance evaluation, public solution sharing and heterogeneous learning approach not scalable with the increasing number of robots?

Q10 (5): For each co-adaptation strategy mentioned in the lecture, give an example of scenario in which that strategy would be the best. Justify your answers.

3 Budget allocation in PSO

Optimal computing budget allocation (OCBA) is a concept first introduced in the mid-1990s by Chun-Hung Chen. This approach intends to maximize the overall simulation efficiency for finding an optimal decision. Simply put, OCBA is a simulation approach that will help to determine the number of replication and/or the simulation time that is needed in order to receive acceptable/best results within a set of given parameters.

We will now see how OCBA can improve the performance of PSO in the presence of noise for a multi-robot obstacle avoidance benchmark task (Di Mario et al., 2015). In this section of the lab, you will be asked to compare the performances of PSO, PSO with re-evaluation of the personal best (PSO pbest) and PSO with OCBA for this given task.

For the learning, the controller used is a recurrent artificial neural network of two units with sigmoidal activation functions. The outputs of the units determine the wheel speeds. Each neuron has 12 input connections: the 9 infrared sensors, a connection to a constant bias speed, a recurrent connection from its own output, and a lateral connection from the other neuron's output, resulting in 24 weight parameters in total. The optimization algorithms try to learn the optimal set of weights for this controller. See (Di Mario et al., 2015) for more details.

The data of one run of PSO, PSO pbest and PSO OCBA can be found in the `Code/part3/results` directory. You will find two files in each of the sub-directories:

- `gbest_progress.txt`: the position of the best particle in the search space at each iteration, along with the fitness of that particle for 100 different evaluations. This file represents the ground truth of every particle and is independent of the PSO method that is being used. This means that regardless of how this particle was created, it is tested 100 times with the robots in simulation.
- `pso_fit.txt`: average and best fitness at each iteration. These fitness values are the results of weights obtained by a specific PSO method.

I11 (10): plot the performance of the best solution of each algorithm along with the ground truth of the best particle for each iteration in Matlab. Compare the results obtained by the three algorithms. Which method has the highest fitness? Which method is conforming the most to the ground truth? Which method is the best method for solving this problem in reality? *(Hint: You should get figures similar to Fig.6 of*

(Di Mario et al., 2015). Ground truth is the average of the 100 runs for a particle. You can load the data files using the `load` *command in Matlab: e.g.,* `load gbest_progress.txt`*).*

Q12 (5): In the results that you saw in the last question, the number of iterations per run for each method has been chosen is such a way that fairness is ensured among all methods. Explain how these numbers were chosen and why we claim that this comparison is fair?

Q13 (5): The implementation part of PSO OCBA can be found in `part3/PSO-OCBA` folder. Look at `static void pso_ocba_c_it(pso_t *psodata)`. Explain how the budget is calculated.

4 PSO for cooperative and collaborative tasks

We will now move on to a more complex task that requires collaboration or cooperation between robots. Collaborative and cooperative tasks are a greater challenge because it is hard for robots, in case of fully distributed control and especially with limited explicit information sharing about their mutual intentions, to determine whether an observed event is the result of their own actions or the actions of other individuals. This situation is characterized by a canonical credit assignment problem in multi-robot systems.

In this section, we will use PSO to learn the coordinated motion task, in which robots must cover as much distance as possible while remaining in each other's sensor range.

S14: Open the `pso_coop.wbt` world in Webots and run the simulation; it takes about 3 hours of simulated time. While the simulation runs, answer the following questions.

Q15 (5): Have a look at the `calc_fitness()` function in the supervisor code. What is the fitness function being optimized?

Q16 (5): Is this an individual or a group fitness evaluation? What hardware would be required to implement it with real robots?

Q17 (10): If your answer to the previous question was individual, can you suggest a group function for global evaluation? Or, if it was group, can you suggest an individual one? *(Hint: remember what you are trying to optimize. Any function should involve the distance traveled and the proximity)*

Q18(5): Have a close look at the supervisor code, in particular the `calc_fitness()` function. Which solution sharing strategy is chosen in this scenario? Is this strategy is the most appropriate one in this case? Why? What would you change in the strategy if there were 4 robots instead of 2?

Q19 (5): The optimization process should be done. What fitness values do you obtain?

Q20 (5): Observe the behavior of the final solution. Is the performance consistent across runs? Knowing that the sensor noise is the same as the one used for the obstacle avoidance task, what do you think is the main cause for the variations in the evaluations?

I21 (10): Copy over the noise-resistant code from I_4 and run the simulation with `NOISY=1`. What fitness values do you obtain in this case? What do you observe in terms of performance difference between noise-resistant and standard PSO?

Q22 (5): What causes noise in this scenario? If sensors and actuators were not noisy, would a noise-resistant PSO be relevant? Justify your answers.

5 References

J. Pugh, A. Martinoli. (2009). Distributed scalable multi-robot learning using particle swarm optimization, *Swarm Intelligence* 3(3), 203–222.

G. Baldassarre, V. Trianni, M. Bonani, F. Mondada, M. Dorigo, S. Nolfi. (2007). Self-organized coordinated motion in groups of physically connected robots. *IEEE Transactions on Systems, Man and Cybernetics—Part B: Cybernetics*, 37(1), 224–239.

E. Di Mario, I. Navarro, A. Martinoli. (2015). A distributed noise-resistant Particle Swarm Optimization algorithm for high-dimensional multi-robot learning. IEEE International Conference on Robotics and Automation (ICRA), 2015, vol., no., pp.5970-5976, 26-30.

Navarro I., Di Mario E., and Martinoli A., "Distributed Particle Swarm Optimization - Particle Allocation and Neighborhood Topologies for the Learning of Cooperative Robotic Behaviors", *Proc. of the 2015 IEEE/RSJ Int. Conf. on Intelligent Robots and Systems,* September-October 2015, Hamburg, Germany, pp. 2958-2965.