Lab Verification Test 2

This lab verification test requires the following equipment:

a) C compiler

b) Matlab

c) Webots

The duration of the test is 2 hours and 30 minutes. It consists of two parts: (i) Sensor Networks, (ii) Particle Swarm Optimization. The maximum score you can get is 120 points; you will obtain the full grade if you earn 100 or more points; your potential extra points above 100 will be integrated in the overall weighted sum for the course grade.

The test is open book, i.e., all printed/written material is allowed (e.g., books, lecture notes, exercises, solutions, personal notes). In addition, you will use a computer from the computer room running Ubuntu, as you did in the labs. You can use internet to look for the necessary information**, but never to communicate with other students or anyone outside the computer room. Note that we will log all the communications and connections that your computer will create. Before starting, please log out and close all e-mail and messaging services (Gmail, Google Docs, Yahoo, EPFL mail, Facebook, etc.). You are NOT allowed to use your personal computer, tablet, phone or any other digital device.**

Getting Started

To start with this test, you will need to download the material available on Moodle. Download test.tar.gz and extract it in your home directory (you can type tar xvzf test.tar.gz). The uncompressed folder *test* has the following contents:

- *DIS_15-16_test_answer_sheet.odt:* LibreOffice file to be filled with most of your answers.
- *part1:* Folder with files needed to answer Part I.
- *part2:* Folder with files needed to answer Part II.

Submitting your answers

Answers must be written on the provided answer-sheet file (*DIS_15-16_test_answer_sheet.odt*), and in a couple of different source files that you will need to complete along the test. In total, **four files** must be uploaded to Moodle by the end of the test. You can upload each source file as soon as you are finished with it, but you should wait until you finish the whole test to upload *DIS_15-16_test_answer_sheet.odt.*

You will have to upload to Moodle the following 4 files:

1. *DIS_15-16_test_answer_sheet.odt*: with answers for all the parts.
2. guided_con.c: C file for Part 1.1.
3. *pso.h:* C file for Part 2.1.
4. *waypoint_controller.c:* C file for Part 2.1.

Important Note:

1. Do not answer on this sheet!

2. Please double-check your submitted solution files.

Part 1: Sensor Networks (60 points)

As you saw in the part 4 of Lab 10, adding mobility to a sensor network offers the advantage of being able to sample at multiple locations with the same sensor node, increasing the coverage and spatial resolution of the sensing system. Its main drawback is the reduction in temporal resolution at any given location. Here you will investigate the case of an improved controlled mobility strategy.

1.1. Controlling a Mobile Sensor Network (35 pts)

We would like to implement an improved robot controller capable of guiding e-puck robots to maximize the information gathered from the field. The behavior of the controller has the following features:

- Feature 1: Using its short-range radio link each robot broadcasts locally the location where it has observed the largest measurement gradient.
- Feature 2: The velocity of the robot at each time step is updated as a weighted sum between the previous velocity, a vector pointing towards its historical location of highest observed gradient, and a vector pointing towards the neighborhood highest observed gradient, and a random walk vector.

S_1 (10 pts): Open the world file *controlled_mobile_net.wbt* (located in *code/part1/webots/worlds/*). Open the controller *guided_con.c* (located in *code/part1/webots/controllers/guided_con/*).

- a (5 pts): Check the implementation of the function *compute_wheel_speeds*. Explain the purpose of this function in no more than 5 sentences. (**Hint:** you can also see lines 250, 251, and 256)
- b (5 pts): Compile the controller code and also the supervisor code. Run the simulation and evaluate the performance of this controller using the provided Matlab script evaluate_mobile.m (located in *code/part1/matlab/*) in the same way you did in Lab 10. Report the performance values for three different field dynamics values of 4, 16, and 64. To set the field dynamics, check the supervisor code located in *code/part1/webots/controllers/dynamic_field_sup/*.

I_2 (15 pts): Using the controller *guided_con.c* (located in *code/part1/webots/controllers/guided_con/*), implement the improved robot controller as described above. The part of the code to modify is marked with TODO. The first feature of the behavior is already implemented in the code.

Note: Save and **submit** your solution code **guided_con.c**.

S_3 (10 pts): Compile your controller. Run the simulation and evaluate the performance of this controller using the provided Matlab script *evaluate_mobile.m* (located in *code/part1/matlab/*) in the same way you did in Lab 10.

- a (5 pts): Report the performance values for three different field dynamics values of 4, 16, and 64. Explain how the performance of your controller corresponds to the field dynamics by comparing the performance values to those you obtained for the original controller in question **S_1**, in no more than 5 sentences.
- b (5 pts): Explain the idea behind the design of this improved controller. How does it provide the improvement? Give your answer in no more than 5 sentences.

1.2. Modeling a Mobile Sensor Network (25 pts)

Now assume that in addition to having controlled mobility using the controller described in part 1.1., the robots vary their sampling frequency as well. By default, the controller samples at the low frequency of f_l. Under the condition that the robot observes a gradient of G larger than a threshold value of G_{th} between its two consecutive measurements ($G > G_{th}$), while having a number of neighbors N smaller than a threshold value of N_{th} ($N < N_{th}$), the controller switches to the high sampling frequency of f_h. The sampling will be done at f_h until the robot gains at least N_{th} neighbors or until the expiration of a timer counting down from T_{sw}.

We intend to study high level models of the system, i.e. microscopic and macroscopic level models. The structure for a probabilistic finite state machine (PFSM) describing an individual robot's behavior is provided below. Assume that p_i represents the probability for transition TR_i. The condition which triggers the transition TR_i is represented by C_i. For further simplification, we assume that all of the transition probabilities, i.e. all values of p_i, are constants regardless of the characteristics of the field or the number of the robots.

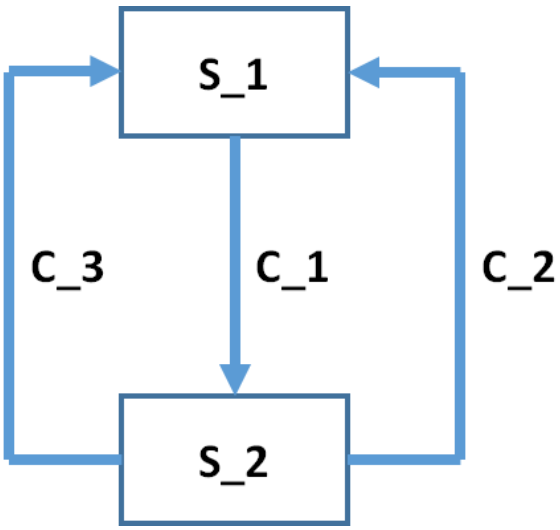

Figure 1: The PFSM for one robot.

Q4 (5 pts): Label the states and transition conditions of the provided PFSM describing the behavior of an individual robot.

(**Hint:** you may copy/paste the simplified forms f_l, f_h, G>/<G_th, N>/<N_th, and T_sw in your expressions.)

- a (1 pts): S_1 = …
- b (1 pts): S_2 = …
- c (1 pts): C_1 = …
- e (1 pts): C_2 = …
- f (1 pts): C_3 = …

Q_5 (10 pts): Formulate a difference equation that describes the dynamics of the population of the robots sampling at f_l, the low sampling frequency. Use the terms described below.

(**Hint:** you may copy/paste the simplified forms on the right-hand-side in your answer sheet)

- $N_l(k)$ = N_l(k) the average number of robots sampling at f_l at time step k.
- $\Delta_h(k)$ = Delta_h(k) the average number of robots which switch their sampling frequency from f_l to f_h at time step k.
- $\Delta_s(k)$ = Delta_s(k) the average number of robots which switch their sampling frequency back to f_l from f_h upon finding enough neighbors at time step k.
- $\Gamma(k - T_{sw}; k)$ = Gamma(k-T_sw; k) the fraction of the robots which start sampling at the high frequency starting at time-step $k - T_{sw}$ and do not get to find enough neighbors during T_{sw}, thus switching back to the low frequency only upon the expiration of their timer at time-step k.

N_l(k+1) = …

Q_6 (10 pts): With the simplifying assumption of the p_i values being constant, write the proper equations for the terms below.

(**Hint:** you may use $N_l(k)$ = N_l(k) and $N_h(k)$ = N_h(k) in your expressions.)

- a (3 pts): Delta_h(k) = …
- b (3 pts): Delta_s(k) = …
- c (4 pts): Gamma(k-T_sw; k) = $\Pi_{j=k-T_{sw}}^{k}$(...)

Part 2: Particle Swarm Optimization (60 points)

As previously seen in the labs, robotic controllers can benefit from machine learning techniques to find the optimal set of parameters with respect to a fitness function. Depending on the task at hand, the architecture of the controller and the appropriate fitness function should be designed after which the learning method can be applied. In this section we will revisit the exercise on application of PSO for learning the parameters of a robotic controller for a navigation task.

2.1. Learning a Robotic Controller (45 pts)

Consider the scenario in Figure 2 (defined for you in *waypoint_track.wbt* in code/part2/webots/worlds). The e-puck in the lower right corner of the arena needs to escape it by navigating through the curve and out the exit in the upper left corner. Your task is to use PSO to find the optimum location of a set of navigational waypoints so that the e-puck exits the arena in the shortest time possible Each particle contains a set of waypoints that will be used as input to the robot controller and evaluated to assess its fitness. The e-puck is driven by a waypoint controller (i.e. it takes a set of waypoint coordinates as input and drives the e-puck to each of them one by one).

Your goal is to decide on the number of waypoints for the robot to use *(Hint: you should choose a fixed number)* and to design and implement a fitness function. *(Note: robot uses dead reckoning)*

Note: Save and **submit** your solution codes for both **I_7** and **I_8** in the same files. Submit the code for **pso.h** and **waypoint_controller.c**.

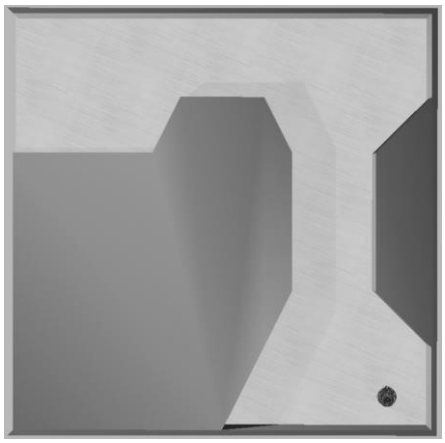

Figure 2: The robotic arena

I_7 (10 pts)**:** Choose the appropriate number of waypoints considering you want to minimize the optimization and navigation time. Make the appropriate changes in **pso.h** and **waypoint_controller.c,** and justify your choice. *(Note: Use the Webots PSO code from lab 8 that is already provided. It already adapts its candidate solution encoding and parameters to the current task.).*

I_8 (30 pts)**:** For learning the desired behavior using PSO, we need to design and implement an appropriate fitness function. To do this, you need to modify the code inside *fitfunc()* (this function is located in code/part2/webots/controllers/waypoint_controller.c). This fitness function should assign higher fitness values to sets of waypoints that achieve the desired behavior faster. Look inside the *fitfunc(),* you will need to complete the section marked with TODO. Briefly describe your

implementation and report the performance of your final controller on the answer sheet. Provide the coordinates of the best set of waypoints you found in your answer sheet.

(Hint: look at the conditions used in this function and the information they can provide when thinking about the fitness function. Additionally, you can use the exponential function to amplify small differences between values. These hints are only optional).

Q_9 (5 pts): Explain how the *maximum_velocity parameter* of *PSO* affects its performance and how it should be chosen in general. Provide examples based on the benchmark function that you tested in SwarmViz.

2.2. Budget Allocation (15 pts)

As you saw in lab09 budget allocation is done differently in PSO standard, PSO-pbest and PSO-OCBA learning methods. We learned how each of these methods work and how faithful they are to reality in the exercises of the lab. The next two questions are a follow up of the lab.

You can find the same data sets you used in the lab in *BudgetAllocation/results*. Figure 3 which is the same as Fig 6 of (Di Mario et al. 2015) is included in the test as the summary of the results you achieved in lab09.

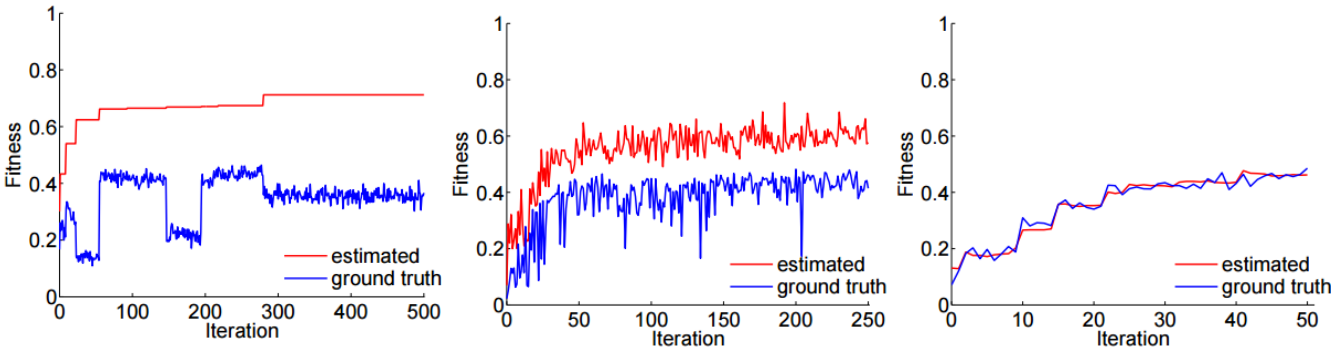

Figure 3 - Progress during a single run for each of the algorithms. The red curve represents the performance of the best solution as estimated by the algorithm, and the blue curve represents the ground truth performance obtained as the average of 100 a-posteriori evaluations. Left: PSO standard, middle: PSO-pbest and right: PSO-OCBA.

Q_{10} (8 pts): 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?

Q_{11} (7 pts): If the noise is negligible in an optimization problem would you recommend using OCBA? What are the problems and difficulties of having multiple robots in terms of learning? Explain your answers.

References

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