Lab Verification Test

This lab verification test requires the following equipment:

a) Matlab

b) Webots

The duration of the test is 3 hours. It consists of four parts: (i) Single and Multi-Robot Navigation, (ii) Task Allocation and Division of Labor, (iii) Particle Swarm Optimization and (iv) Platforms, Localization and Distributed Sensing. Each part consists of 30 points. The maximum score you can get is 120 points; you will be awarded the maximum grade if you obtain 100 or more points; your potential bonus of 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_19-20_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.
- *part3:* Folder with files needed to answer Part III.
- *part4:* Folder with files needed to answer Part IV.

Submitting your answers

These are the **8** files that you have to submit on Moodle:

- *DIS_19-20_test_answer_sheet.odt:*
- *pose_estimation.m*
- *trajectory.png*
- *epuck_crown_type.c*
- *epuck_crown_priority.c*
- *pso_sup.c*
- *pso_sup.tar.gz*
- *guided_con.c*

Important Notes:

1. Do not answer on this sheet!

2. Please double-check your submitted solution files.

3. Only submit the above files. Additional files, or compressed files that are not asked for, will not be considered for evaluation.

4. Make sure that your codes compile before submitting them, otherwise they cannot be graded.

Part I: Single and Multi-Robot Navigation (30 points)

1.1. Localization (18 pts)

1. **(Q, 6pts):** Consider the following sensors that are on a mobile robot. For each of them, answer if they can be used for self-localization, and explain how or why not in two sentences.
   - Stepper motor encoder
   - Camera
   - One proximity sensor in the front, which gives a binary true/false output
   - Accelerometer

2. **(Q, 2pts):** What is a covariance matrix and a Jacobian matrix in the context of position uncertainty estimation? What is their significance?

3. **(I, 10pts):** Open the MATLAB file `pose_estimation.m`. Consider an e-puck travelling on a plane. The wheel velocities (rotational) over time are given in the code. All other parameters are also given.

   Hints:
   Look for comment "`Set up constant parameters`*" at the top of the file.*
   Places in the file where you need to add code are marked with a comment "`TODO`*".*

   (a) **(2 pts)** Write code to compute the linear velocity of each wheel (in cm/s).

   (b) **(3 pts)** Write code to compute the pose of the robot (x,y,theta) for the next time step. Use the e-puck's differential drive motion model seen in the lecture. Use cm as a unit of length.

   (c) **(3 pts)** Consider that there is wheel slip. Then, the actual position of the robot will deviate from the prediction computed in (b). Complete the part of the code that updates the uncertainty of the robot's pose at each time step. The covariance and the Jacobian matrices are given or already computed. You can find the variable names in the comment.

   (d) **(2 pts)** Observe the evolution of the error ellipse (which represents covariance in position) in the plot generated at the end of the MATLAB code. Explain why the size increases as the robot moves forward, and what its shape and orientation represent.

   Submit the code as well as the automatically generated `trajectory.png` image **on Moodle**.

1.2. Formations and flocking (12 pts)

Consider a group of robots with the following characteristics:

- The estimate their positioning using dead reckoning.
- They can broadcast their estimated position using a radio link with limited range.
- They are equipped with a range and bearing sensor to detect relative position of other robots. Note that this sensor requires line of sight.

4. **(Q, 2pts):** Imagine you are implementing formation control on a group of robots. You could either use
   - relative position information obtained from range and bearing sensor, or
   - each robot's estimated position that it broadcasts over radio.

   What are the advantages and drawbacks of each approach?

5. **(Q, 2pts):** Consider the following incidence matrix used for graph-based formation control. How many robots and connection edges are in the formation?

$$\begin{bmatrix} -1 & 0 & 0 & 1 & -1 & 0 \\ 1 & -1 & 0 & 0 & 0 & 1 \\ 0 & 1 & -1 & 0 & 1 & 0 \\ 0 & 0 & 1 & -1 & 0 & -1 \end{bmatrix}$$

6. **(I, 8pts)** Open and run laplacian_flocking2019.m in MATLAB. Note that the flock center converges to the origin (0,0).

   Note: you need to paste code here and also submit the file.
   Places for addition of code for each part are marked with comment "`TODO`*" in the file.*

   (a) (2 pts) Implement code to make the formation converge to a square centered at (0,0) of side length 4. Paste code that you added here (should be around 2-10 lines).

   (b) (6 pts) You will see that the robots slow down as they get closer to their goal, which makes convergence slow. Implement code to make it faster. Paste code that you added here (should be around 2-10 lines).
   Hint: Remember from the lecture that this is similar to a P controller. Add an I component. Tune the constant weight for the I component carefully, it should not be too big.

Part II: Task Allocation and Division of Labor (30 points)

2.1. Threshold-Based Task Allocation (5 pts)

7. **(Q, 5)** In a threshold-based task allocation scenario we have two types of tasks and two types of robots. Both types of robots are capable of handling both types of tasks but they have a better performance with one of them. We want the robots to specialize in the type of task that they do best, using adaptive thresholds. Suggest a solution for adapting the threshold during the experiment using equations and conditions in which they have to be applied. Explain briefly how the adaptation takes place.

2.2. Market-Based Task Allocation (25 pts)

This part is based on the scenario of the market-based task allocation exercise you have seen in Lab 8. The world contains 5 robots that are supposed to achieve tasks that randomly appear in the environment. In contrast to Lab 8, there are two types of tasks (green and red) and two types of robots (green and red). When a robot tries to address a task of its own type, the task is done instantaneously. But if the task and the robot are from two different types, the task takes about 6 seconds to be handled. During this time the robot needs to stay on the task until it is done. Since the goal is to achieve as many tasks as possible in a certain amount of time, the robots bid the time that they need to reach the task and finish it (as opposed to travelled distance in Lab 8).

8. **(Q, 8 pts)** Formulate a local and a global objective function that minimizes the time taken to achieve a fixed number of tasks. Briefly explain your answers.

   Open the world file located at `test/part2/world/market.wbt` in Webots where this scenario is implemented. Then compile the robots and supervisor controllers and run the simulation to see how it works. Note that the simulation runs until 50 tasks are handled.

9. **(Q, 4 pts)** The bidding strategy that is implemented in the robot's controller is the solution of Lab 8 (called Best Tactic). Briefly explain how it works and why it is not fully adapted to the current problem.

10. **(I, 8 pts)** In the robot's controller `epuck_crown.c`, in function `receive_updates()`, make the necessary changes to modify the bidding strategy to the one you formulated in question 8. Briefly explain your implementation in your answer sheet and mention the line numbers where you applied modifications in the code. Save your new controller and **submit it on Moodle** under the name `epuck_crown_type.c`.

    Notes: The robots are assumed to have a constant speed of 0.062 m/s. The type of a task is given to the robot in the message sent by the supervisor. Check the message structure in `message.h` to know which parameter to use. The type of a robot is given by a global variable called `robot_color`.

11. **(I, 5 pts)** Now, let's imagine that the red events have priority over the green ones for all the robots. It means that when they appear, we want to address them as soon as possible. Make the necessary changes in the bidding strategy in the code and briefly explain in your answer sheet along with the line numbers where you applied the modifications. Save your new code and **submit it on Moodle** under the name `epuck_crown_priority.c`.

Part III: Particle Swarm Optimization (30 points)

12. (Q, 4pts): What is the credit assignment problem in collective systems? Illustrate your answer with an example.

Open the world `/webots/worlds/labyrinth.wbt` in Webots, in which a maze is designed. The goal of this exercise is for a robot to learn a strategy to escape the maze as fast as possible. For this purpose, two robots are placed in the middle of the maze and will run PSO for this optimization.

Open the supervisor code `/webots/controllers/pso_sup/pso_sup.c` in which you will recognize the PSO structure from Lab 7. Compile both the supervisor and the robots' controller code (`controllers/obs_con`).

Hint: *Remember that most of the PSO parameters are defined in pso_sup.c and a few in pso.h*

13. (Q, 3pts): What solution sharing strategy is implemented here? Give some advantages and/or disadvantages of the implemented solution. Justify your answer.

14. (Q, 3pts): What solution-sharing strategy would be best for this scenario? Justify your answer.

15. (Q, 4pts): Compare the solution strategy you identified in Question 13 and the one you chose in Question 14. What needs to be changed in order to transform the implemented algorithm to your chosen one? Depending on your choice of algorithm, you should find 1- 2 main changes. Justify your answer.

16. (I, 5pts): Chose and implement an adapted fitness function for the solution you have chosen. Your fitness function should allow for each robot to learn to escape the maze. The function you have to modify is in the file `pso_sup.c` and is marked with TODO.
Is your implementation a group or individual fitness evaluation? Briefly report and explain your chosen fitness function in the answer sheet.
Submit the file `pso_sup.c` **on Moodle**.

17. (I, 11pts): Based on your answer in Question 15, implement the solution sharing strategy you have chosen in Question 14. Briefly mention and explain in your answer sheet all the modifications you make in the code, while mentioning the line number. Compress the folder `pso_sup` to **`pso_sup.tar.gz`** and **submit it on Moodle**.

Part IV: Platforms, Localization and Distributed Sensing (30 points)

18. (Q, 6pts): Load the world `space_division_net.wbt` . In this simulation, there are 16 robots forming a sensor network that aims at sensing the light field.

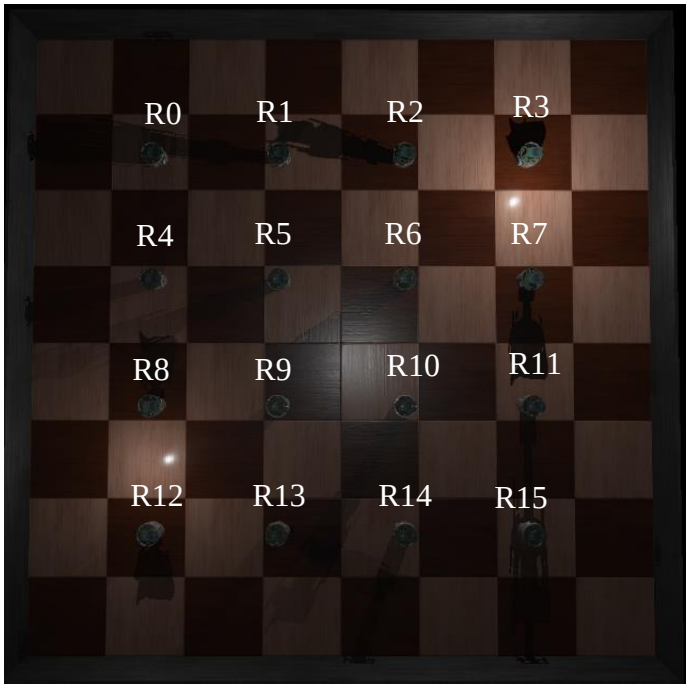

Figure 1: Sensor network to sense a light field

a) (2 pts) The robots are positioned on a grid and are 0.4 m apart from each other. Assume that the maximum radius of communication for the robots is 0.6 m. What is a network topology that allows the network to communicate and send data to R0 (please use the naming convention in Figure 1)? *(Hint: To answer this question without drawing the graph of the topology you can write the connections from each leaf (or terminal) node to R0. E.g. if R7->R6->R5->R0, no need to write R6->R5->R0)*

b) (4 pts) Compile the controller and run the simulation. You will see the values sensed from the light sensors of the robots displayed on the console. Explain how you would perform spatial suppression in the network topology that you described in the question above. What is the resulting network after pruning?

Load the world `controlled_mobile_net.wbt`. The robots are performing a random walk in a field with two light sources, one dynamic and one static. The robots are using a light sensor to acquire light information. After compiling the controllers, run the simulation and wait for it to end. Open Matlab, import the data by running the output file, run the `evaluate_mobile.m` script and wait for it finish.

```
>> output
>> evaluate_mobile
```

Note down the performance result, you will need it for later questions.

19. (I, 10pts): To optimize the controller, we want the robots to move if they are in an area where they are not acquiring new information (i.e. new light values are similar to previous light values), while we want them to stop if they are in an area where new information is acquired (i.e. new light values

are different from previous light values). In the code `guided_con.c`, implement this behavior by stopping the robot if it is acquiring new information. To answer this question, **only write code in the section marked as TODO-1, describe what you are doing in the comments and paste your code additions in the answer sheet.** *(Hint: to check if the robot is acquiring new information, you can use the gradient information that is printed on the console).*

After completing your implementation, run the simulation, then run the `evaluate_mobile.m` Matlab script on the new output. **Note down the performance result on your answer sheet**.

20. (Q, 2 pts): What are the advantages of this implementation over a solution where all the robots move? What are the advantages over a solution where all the robots are static?

21. (I, 8 pts): To further optimize the behavior of this controller, we want to replace the random walk with a more advanced navigation strategy. Here, we will compute the gradient of the light field based on the spatial measurements we get from other robots and use that to compute the direction of motion of the robot.

Let s_j be the light sensor measurement of robot j.

For each robot i, we will compute the gradient with respect to measurements of other robots as

$$\nabla f_j = \left(\frac{s_j - s_i}{x_j - x_i}, \frac{s_j - s_i}{y_j - y_i}\right), j \in \{0,1,2,\ldots,15\}, j \neq i$$

Additionally, we will compute the gradient with respect to the personal best position as

$$\nabla f_{sb} = \left(\frac{s_{selfbest} - s_i}{x_{selfbest} - x_i}, \frac{s_{selfbest} - s_i}{y_{selfbest} - y_i}\right)$$

Note that ∇f_j and ∇f_{sb} are 2D vectors.

Then, we will compute our desired direction of motion $\hat{d}$ (a normalized vector) as

$$\vec{d} = \nabla f_{sb} + \sum_{j \neq i} \nabla f_j, \hat{d} = \frac{\vec{d}}{\|\vec{d}\|}$$

You need to implement the following in the code. The part of the code to modify is marked with TODO-3 and TODO-4, **please modify code only in these sections**. Additionally, you will have to comment two lines of code, marked as TODO-2. Here are the steps you should follow:

- **Comment** the two following lines of code (around line 362): (see TODO-2 in the code)
  ```
  speed[0] = BASELINE1 + rand()%500;
  speed[1] = BASELINE2 + rand()%500;
  ```
- **Implement** the computation of the quantity ∇f_j (see TODO-3 in code)
- **Implement** the computation of the quantity ∇f_{sb} (see TODO-4 in code)
- **Paste** your code additions to the answer sheet
- **Save and submit your solution code** `guided_con.c` **on Moodle**

After completing your implementation, run the simulation, then run the `evaluate_mobile.m` Matlab script on the new output. **Note down the performance result on your answer sheet**.

22. (Q, 4pts): Among the three implementations above (random walk, random walk with informationtaxis, gradient-based movement), which one were you expecting to give the best performance? Which one gave the best performance in the end? What are the advantages and drawbacks of a random walk controller versus a gradient-based implementation?