Lab Verification Test 1

This lab verification test requires the following equipment:

a) Matlab

b) Webots

The duration of the test is 3 hours. It consists of two parts: (i) Single and Multi-Robot Navigation, (ii) Task Allocation and Collective Decisions. Each part consists of 60 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_17-18_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_17-18_test_answer_sheet.odt*), and in a couple of different source files that you will need to complete along the test. In total, **eight files** must be uploaded to Moodle by the end of the test. Make sure that your codes compile before submitting them, otherwise they cannot be graded. You can upload each source file as soon as you are finished with it. Don't forget to submit the answer sheet *DIS_17-18_test_answer_sheet.odt* on Moodle once you finished the whole test.

You will have to upload to Moodle the following 8 files:

1. *DIS_17-18_test_answer_sheet.odt*: with answers for all the parts.
2. *reynolds.c*: Webots controller for Part1.2.
3. *laplacian_formation1.m*: Matlab file for Part 1.3.
4. *laplacian_formation2.m*: Matlab file for Part 1.3.
5. *laplacian_formation3.m*: Matlab file for Part 1.3.
6. *lab05_epuck_crown_slow_I15.c*: Webots controller for Part 2.2.
7. *lab05_epuck_crown_I16.c*: Webots controller for Part 2.2.
8. *lab05_epuck_crown_slow_I16.c*: Webots controller for Part 2.2.

Important Notes:

1. Do not answer on this sheet!

2. Please double-check your submitted solution files.

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

Part I: Single and Multi-Robot Navigation (60 points)

1.1. Localization (12 pts)

Q1) An e-puck has to travel from the origin (x=0, y=0) to the destination (x=10, y=7, marked with a red X) (Figure 1). On the field there are red, green and blue cylinders which the e-puck can all see with its omni-directional camera. The destination is not marked on the field; it is only marked in Figure 1 for clarity. While the e-puck travels to a colored marker it does not accumulate any position error (feature-based navigation), but when not traveling towards a colored marker using odometry instead, it accumulates an average position error of 20% of the distance traveled. The e-puck can go to the final destination via the red markers, the green markers or the blue markers. On which route does it reach the destination with the smallest average position error? What is the average position error (in meters) at the destination when using that route? (12pts)

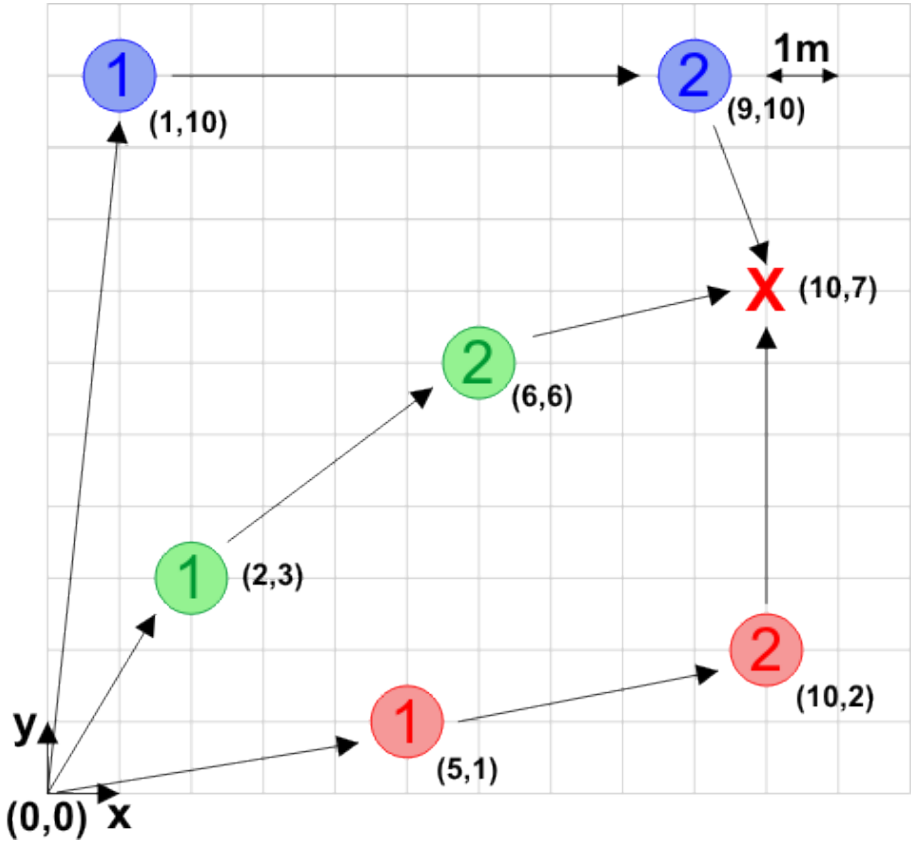

Figure 1: robot traveling from (0,0) to (10,7) via colored markers

1.2. Reynolds Flocking (24 pts)

In Lab 4 you learned how to implement a flocking behavior using the three Reynolds rules: *Cohesion, Separation* and *Alignment*. A complete code similar to the Lab 4 is provided to you, in the exam materials "test/part1/webots". The robots begin in random locations and start the flocking behavior. Similarly as in the lab, they are able to avoid obstacles using Braitenberg behavior. Compile the controllers for the robots and the supervisor.

Q2) Imagine that your flock consists of robots with a solar cell so they try to get closer to the beacon lights in order to recharge their battery. Assuming that the positions of the light sources are known to the robots, write down in the answer sheet how you would implement the additional *Charge* rule. You need to provide an equation, with a short explanation. (10pts)

Q3) Which of the three Reynolds rules is your *Charge* rule most similar to? (2pts)

I4) The supervisor code provides the robots with the position of the light sources (beacons) *Hint: the robots receive the positions of those beacons from the supervisor in lines 281-290 in reynolds.c*. Implement your *Charge* rule in the e-puck controller (*reynolds.c*). Your new controller should make the robots go towards the light when close to it, while migrating towards the target position in a flock. To do so, you need to carefully choose the weight of the *Charge* rule. (12pts)

Hint: the robot flock does not need to pass over the light, it just needs to go closer to it when passing it.

Note: Save your controller code *reynolds.c* and submit it on Moodle. In your answer sheet write the major changes you made in the code.

1.3. Graph-Based Formations (24pts)

Consider six agents, shown in Figure 2. The goal is to design a graph-based control algorithm which leads them to the desired topological formation (green circles).

I5) Write a controller in Matlab that brings the robots from their initial positions (the red circles in Figure 2) to the desired formation (the green circles). Assume that the robots are holonomic and are massless, i.e. they can instantly move in any direction. You can use the code provided to you "test/part1/matlab/laplacian_formation.m" (which is similar to the solution of Lab 4). Save and submit your solution script with the name "*laplacian_formation1.m*". *Hint:* It is not important to which absolute positions the desired formation converges to, only the relative topology of the agents should be the one shown in Figure 2. (10pts)

Q6) List the consequences of assuming holonomicity of the robots? (4pts)

I7) The simple graph-based controller you implemented in I5 makes robots converge into a formation, but not move afterwards. How can you control migration of the formation? Make the formation migrate at a constant velocity towards the y-axis. You can keep the previous changes or start from the original code. Save and submit your solution script with name "*laplacian_formation2.m*". (2pts)

I8) Another way to move the formation is to establish a leader-follower relationship between the robots. Modify your code so that one of the robots is a leader, which moves towards the y-axis. The rest of the robots (i.e. the followers) maintain the formation on the same manner as in part I5. You can keep the changes from part I5 or start from the original code. The number of robots does not matter in this question. Save and submit your solution script with name "*laplacian_formation3.m*". In your answer sheet write how the leader-follower formation controller affects performance of the formation, particularly its shape. How could you solve this problem? (8pts)

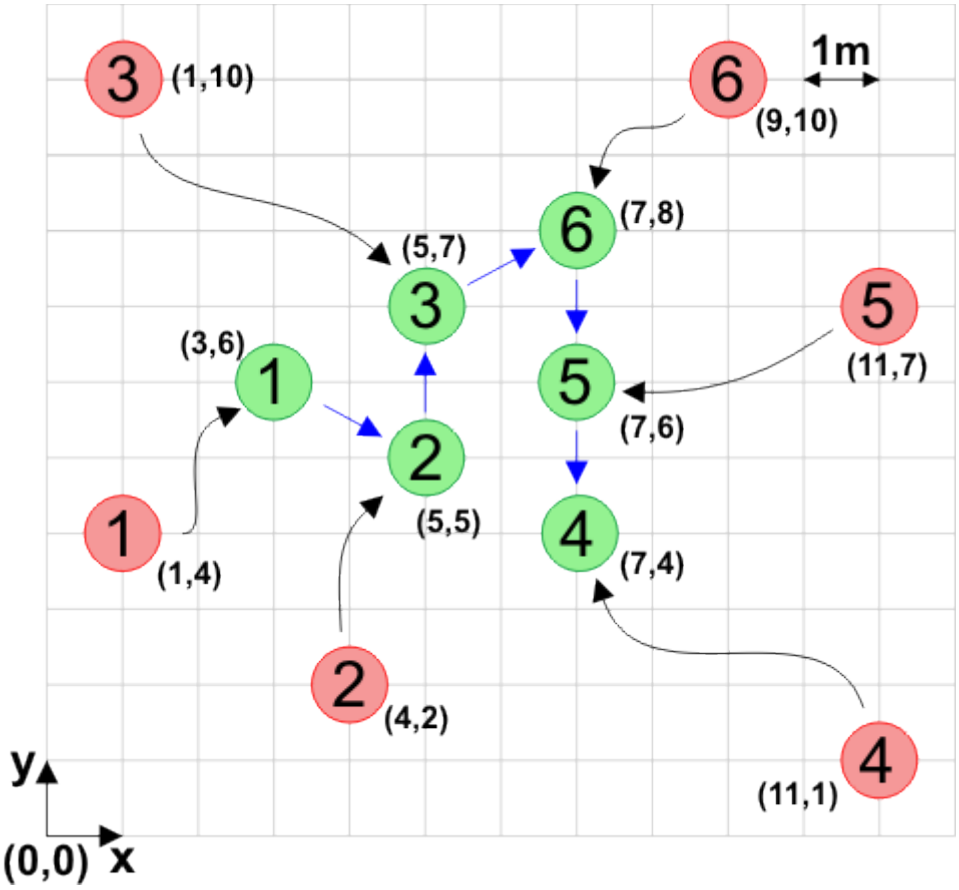

Figure 2: The initial (red) and the desired (green) formation of six robots.

Part II: Task Allocation and Collective Decisions (60 points)

2.1 Threshold-Based Task Allocation (20pts)

In the context of threshold-based task allocation problem that you have seen in Lab 5, imagine there are 4 robots (A, B, C, and D) working together in an environment. Each robot has a different probability of response to stimulus characteristics as shown in the graphs below (Figure 3):

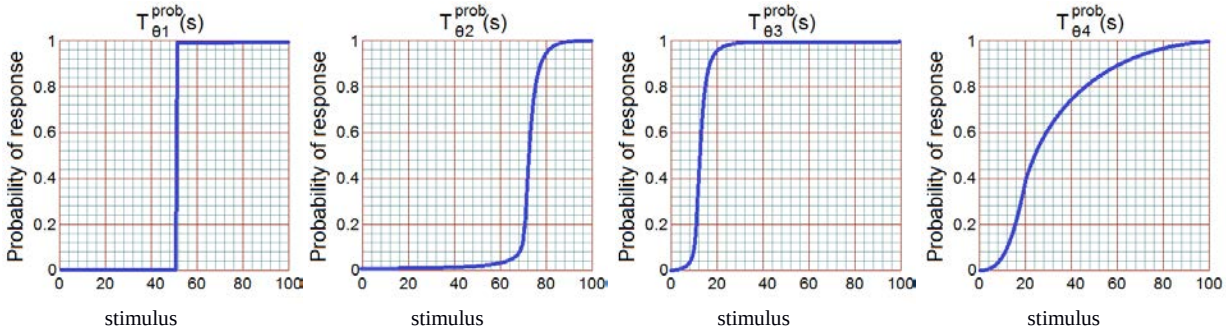

Figure 3: Probability of response characteristics

Assuming that the stimulus corresponds to the total number of tasks at every moment in the environment and that the robots keep their threshold-based response fixed,

Q9) Sort the robots in terms of how active they will be in handling the tasks, assuming that the number of tasks is constantly increasing over time if the robots are inactive (5pts)

Q10) Sort the agents by their nonlinearity parameter from smallest to largest. (5pts)

Q11) Is it possible that all the robots be active at some time? If yes, in what conditions? (5pts)

Q12) If there is a single task in the environment, is it going to be ever handled? If yes by which robot? (5pts)

2.2 Market-Based Task Allocation (40pts)

Open the world file "test/part2/Market/webots/worlds/part2.wbt" in Webots. This question is based on the market-based task allocation exercise you have seen in Lab 5. The world contains 10 robots of two types: the normal e-pucks (e-puck 1 to 5) and the slow ones (e-puck 6 to 10) for handling identical tasks. The two types of robots are represented as green (normal) and yellow (slow). Build the controller of the robots ('lab05_epuck_crown.c' and 'lab05_epuck_crown_slow.c') and the supervisor ('lab05_auct_super.cc'). Run the simulation and see how it works. Note that the simulation runs until 50 tasks are handled or 180 seconds passed.

Q13) Which lines in which file/s have implemented the bidding strategy of the robots? (4pts)

Q14) Explain briefly the bidding strategy that is implemented in the code. (6pts)

I15) In the current code there is no difference between the slow robots and the fast ones in the task allocation and this is the main reason why the number of handled tasks is small. Implement a heterogeneous bidding strategy by changing the bidding method of the slow robots in order to increase the number of handled tasks of the system by giving more chance to the fast robots to handle tasks. **Explain your solution** in a few sentences in the answer sheet, then save, **rename** and submit your controller code with name **"lab05_epuck_crown_slow_I15.c**". (10pts)

I16) Now we would like to have the slow robots to handle only the tasks that are in the left side of arena (i.e., x-axis task location < 0.0) and fast robots to handle the tasks that appear on the right side of arena. Do the necessary changes in the codes in order to perform this change. **Explain your strategy** in a few sentences in the answer sheet, then save, **rename** and submit your controller codes with names "**lab05_epuck_crown_I16.c**" and "**lab05_epuck_crown_slow_I16.c**". (14pts)

Q17) If you were asked to address the task sharing problem explained in the question I15 using threshold-based algorithms, how would you define the thresholds and the stimulus functions of the two types of robots in order to have a good performance? (6pts)