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 two parts: (i) Single and Multi-Robot Navigation, (ii) Task Allocation and Division of Labor. 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_18-19_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

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

- *DIS_18-19_test_answer_sheet.odt:*
- *pose_estimation.m*
- *reynolds.c*
- *graph_based.m*
- *threshold_super.c*
- *lab05_epuck_crown_green.c*
- *I16.tar.gz (folder with lab05_epuck_crown_green.c, lab05_epuck_crown_red.c and lab05_auct_super.c)*
- *I18.tar.gz (folder with lab05_epuck_crown_green.c and lab05_auct_super.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.

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 Robot localization and navigation (30 pts)

Q1(6 pts): Consider the following sensors that the e-puck is equipped with:

(a) Stepper motor counter (b) Camera (c) Proximity sensor
(d) Microphone (e) Accelerometer

Pick three of them from the list above and for each of them label their main feature (i.e., proprioceptive/exteroceptive; passive/active) and explain in two sentences how they can be used for self-localization, in what type of environments, or under what assumptions. Write your answers in the answer sheet.

Q2(6 pts): The diameter of an e-puck wheel is 4.1 cm, the distance between the wheels is 5.3 cm and a wheel command of 1000 corresponds to 60 rpm (revolutions per minute) of the stepper motor. What should be the wheel command for left and right wheel for:

(a) Forward motion at 5.55 cm/s.

(b) In place rotation at $2\pi/3$ rad/s.

I3 (12 pts): Consider an e-puck traveling on a plane, in a scenario similar to the previous question. However, in this scenario, the e-puck wheels can slip occasionally. This means that the e-puck progress at each time step will deviate from its prediction, according to some uncertainty related to the frequency of wheel slip. In the lecture you saw that it is possible to predict the pose of the e-puck (2D position and heading) and its uncertainty over time using dead-reckoning. Open the MATLAB file "*test/part1/matlab/pose_estimation.m*", where you will have to complete the code at the places marked with the comment `TODO` according to the following questions. When you are finished save your code in "*pose_estimation.m*" file and submit the file. **Make sure the script runs without errors and generates a plot in the end.**

(a) Complete the part of the code that predicts the pose of the e-puck at the next time step using the e-puck's differential drive motion model seen in the lecture.
(b) In the code, have a look at the terms `motion_jacobian` and `motion_covariance`. What do those terms represent? Write your answer in the answer sheet.
(c) Complete the part of the code that updates the uncertainty of the robot's pose at each time step. *Hint: The Jacobian matrices for pose and motion are already computed.*

Q4 (6 pts): Consider an e-puck traveling on a plane and using dead-reckoning to estimate its own position. Observe the evolution of the error ellipse in Figure 1, where a robot is moving along the x-axis from left to right. You should see something similar in the plot generated by the MATLAB script of the previous question.

(a) Why does the error grow faster in the direction perpendicular to the movement? In particular, what is the physical meaning of its dimension and orientation? Write your answer in the answer sheet.

(b) Give one example of a sensor that you can add to reduce the uncertainty in this direction. Write your answer in the answer sheet.

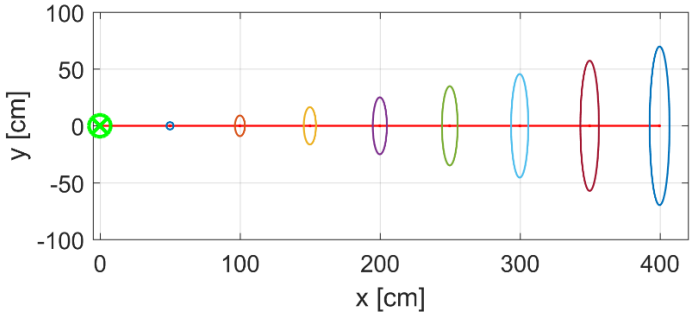

Figure 1: Evolution of the error in the position estimate during robot motion.

1.2 Flocking and formation control (30 pts)

Recall what you have learned in Lab 4 about coordinated movements between robots. One type of coordination consists of flocking behavior, based primarily on three Reynolds' rules: *Cohesion, Separation* and *Alignment*. In the exam materials "*test/part1/webots*" you can find a Webots environment and code similar to what you saw in Lab 4. Compile the controllers for the robots and the supervisor.

I_5 (8 pts): The robot controllers do not have the Reynolds' rules implemented yet, and the robots just go forward in their own directions. Implement the three Reynolds' rules so that each robot is able to be 40cm apart from all its neighbors. Save your code in "*reynolds.c*" and submit the file.

Q_6 (2 pts): In this Webots world the position between all the robots was acquired by the Supervisor. However, you also saw that this information could be extracted by relative sensors on board each robot (in this case simulated in Webots). Name one advantage and one disadvantage in changing to a solution with relative localization. Write your answer in the answer sheet. **Justify your answer.**

Another type of coordination between robots that you saw consists of graph-based formation control methods. In these methods, each robot has a set of connections to its neighbors. Each connection between two robots indicates that there is sensing information which can be leveraged for controlling their inter-robot positions. The robots and their connections can be elegantly described by an incidence matrix together with a weight matrix. The code on "*test/part1/matlab/graph_based.m*" implements a version of the graph-based formation control on a group of robots, using the following incidence matrix:

$$I = \begin{bmatrix} 1 & 0 & 0 & -1 & +1 \\ -1 & 1 & 0 & 0 & 0 \\ 0 & -1 & 1 & 0 & -1 \\ 0 & 0 & -1 & 1 & 0 \end{bmatrix}$$

Notice that the robots in this simulation are considered to be holonomic and mass-free.

Q_7 (2 pts): According to the incidence matrix shown above, how many robots and connection edges are in this formation? Write your answer in the answer sheet. **Justify your answer.**

Q_8 (6 pts): If obstacles were included in the environment, discuss how you would change the code in order to adapt the graph-based formation control algorithm so that the robots would be able to avoid the obstacles. Write your answer in the answer sheet. **Justify your answer.**

Q_9 (4 pts): Discuss what you could change in the code in order to increase the convergence speed of the group of robots into formation. What can be a possible drawback of doing so? Write your answer in the answer sheet. **Justify your answer.**

I_{10} (8 pts): Open and run graph_based.m in MATLAB. Note that the formation center converges into a random point in the environment. By adding a new robot into the formation acting as a leader, implement a leader-follower strategy to move the formation center to the origin (0,0). Implement your solution such that the formation center will always move towards the origin, regardless of the initial positions of the robots (including the leader). When you are finished, save your code in "*graph_based.m*" and submit the file. *Hint: the leader might need an additional control input leading it towards a goal position.*

Part II: Task Allocation and Division of Labor (60 points)

2.1 Threshold-Based Task Allocation (30 pts)

In the context of threshold-based task allocation similar to Lab 5, 5 e-puck robots are trying to accomplish the tasks that appear randomly in the arena.

I_{11} (13 pts): Unlike the code in Lab 5, each task has a random duration which corresponds to the number of iterations that a robot needs to spend on the task before it disappears. We would like to adapt the stimulus based on the duration of the tasks, instead of their number. Open the world file "*test/part2/webots/worlds/threshold.wbt*" in Webots, as well as the controller *threshold_super.c*. Modify the implementation of the stimulus both for local and global configurations and then set the threshold according to the new stimulus. Submit your file *threshold_super.c* and briefly mention the changes that you made, with the line number, in the answer sheet.

Q_{12} (9 pts): Now imagine that the robots would have limited battery autonomy. This implies that we want to decrease the energy consumption as much as possible, while keeping the number of unfinished tasks as low as possible the entire time. Moreover, the robots need some time to recharge after spending BATTERY_LIFE iterations active. In order to recharge their batteries, the robots need to stand still for CHARGING_TIME iterations. During the recharging state, the robots will not react to any stimulus and as soon as the battery is recharged, they will reengage in the task accomplishment according to the present stimulus and threshold. Based on the strategies studied in Lab 5, which configuration would be optimal for this situation: fixed or adaptive threshold, homogeneous or heterogeneous threshold, global or local stimulus? **Justify your answer** for each of the three components (no implementation details are necessary).

Q_{13} (8 pts): Imagine that there are two types of tasks A and B. Let's assume that the robots can become specialized in one specific type of task. In the case where the thresholds are probabilistic, adaptive and heterogeneous, how would you define the threshold, so the specialization takes place? Please **justify your answer** and include equations in your answer sheet.

2.2 **Market-Based Task Allocation (30 pts)**

Go to folder "*test/part2/webots*" and open the world file "*market.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 green e-pucks (e-puck 1 to 5) and the red ones (e-puck 6 to 10) for handling two types of tasks colored gray (e0 to e4) and pink (e5 to e9). The two types of robots are initially similar running the same controllers. Go to folder "*test/part2/webots*" and build the controller of the robots ("*lab05_epuck_crown_green.c*" and "*lab05_epuck_crown_red.c*") and the supervisor ("*lab05_auct_super.c*"). Run the simulation and see how it works. Note that the simulation runs until 50 tasks are handled or 180 seconds passed.

Q_{14} (4 pts): Explain briefly the bidding strategy that is implemented in the code.

I_{15} (5 pts): Change the bidding strategy of the **green robots** in a way that the green robots with no task assigned would have higher chance of winning the next events. **Explain your strategy (even if you fail to implement it)** in a few sentences in the answer sheet. When you are done with the code, save the controller and submit the "*lab05_epuck_crown_green.c*" file.

I_{16} (10 pts): Start over with the initial codes. In the current code there is no difference between the red and green robots in the task allocation. Implement a heterogeneous bidding strategy by changing the bidding method of the robots such that the red robots have more tendency (i.e. bidding lower) to handle the pink tasks and the green ones handle more the gray tasks. **Explain your solution** in a few sentences in the answer sheet. When you are done with the code, save the controllers and the supervisor ("*lab05_epuck_crown_green.c*", "*lab05_epuck_crown_red.c*" and "*lab05_auct_super.c*") in a "*I16*" folder. Compress the folder and submit the compressed file.

Hint 1: In the supervisor code you can distinguish between the gray and the pink task by reading their "DEF" property into an integer ID, using:

```
int event_type = wb_supervisor_node_get_def(node_)[1]-'0';
```

The `event_type` *for the gray tasks will be between 0 and 4 and for the pink will be between 5 and 9.*

Hint 2: You can use the above line in the part of the supervisor code where the events are created.

Hint 3: If you want to add an item to the messages sent to the robots, you would need to modify the structure of the message in `message.h` *file.*

Start over with the initial codes. Imagine the pink tasks are the urgent tasks that have to be handled as quickly as possible.

Q_{17} (3 pts): What would you change in the codes in order to send a robot (irrespective of the robot type) to the pink tasks as soon as possible?

I_{18} (8 pts): Implement your strategy only for the green robots (in order to save time in implementation, don't change the controller of the red robots). When you are finished with the code, save the controllers and the supervisor ("*lab05_epuck_crown_green.c*" and "*lab05_auct_super.c*") in a "*I18*" folder. Compress the folder and submit the compressed file. *Hint: you can re-use the supervisor code you modified in question* ***I_{16}***.