1 Lab 10: Distributed Sensing

This laboratory requires the following:

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

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.

The combined total number of points for this laboratory exercise is 100. The laboratory is not graded.

2 Energy Efficiency in Distributed Sensing

Efficient use of limited energy resources is a central theme of the field of distributed sensing. In many applications, sensor networks need to be able to operate over long periods of time with limited or no external intervention after the initial deployment. For this reason much effort has been placed in developing intelligent algorithms capable of maximizing the operation time of distributed sensing systems through node activity management.

In order to evaluate the performance of a distributed monitoring system an adequate metric is needed that can integrate the tradeoff between field estimation quality and energy cost. A general performance metric for monitoring phenomena without prior assumptions about the form of the signal or type of application can be expressed by:

$$M_C(\alpha,\beta,\gamma,\delta) = \alpha \cdot \left(1 - \frac{1}{\varphi_{max} - \varphi_{min}} \cdot \sqrt{\frac{\sum_{n=1}^{N}\left(\hat{\varphi}_n(x,y,t) - \varphi_n(x,y,t)\right)^2}{N}}\right) + \beta \cdot \left(1 - \frac{\sum_{k=1}^{K} S_k}{K \cdot T \cdot F_s/L_s}\right) + \gamma \cdot \left(1 - \frac{\sum_{k=1}^{K} P_k}{K \cdot T \cdot F_m}\right) + \delta \cdot \left(1 - \frac{\sum_{k=1}^{K} V_k}{K \cdot T \cdot v_{max}}\right)$$

where:

φ_n: the fully-sampled dataset
$\hat{\varphi}_n$: the estimated dataset (based on the subsampled measurement set)
φ_{min}: the minimum observed value
φ_{max}: the maximum observed value
N : the number of samples in the fully-sampled dataset
K : the number of nodes in the network
S_k : the number of measurements taken by node n
T : length of experiment (time)
F_s : sampling frequency
L_s : samples per measurement
P_k : the number of messages sent by node n
F_m : maximum message transmission rate
V_k : length of agent n's trajectory
v_{max} : maximum agent velocity

The weights $(\alpha, \beta, \gamma, \delta)$ may be balanced according to the severity one wishes to associate with each of them, as long as they sum to one for normalization.

In this lab we will focus on the specific task of environment monitoring, by considering a group of e-pucks which are sensing a light field. Start by downloading lab10.tar.gz from Moodle and decompressing it:

```
$ tar xvfz Lab10.tar.gz
```

This will create a *Lab10* folder containing the worlds and controllers for this lab. After loading the Webots worlds, make sure you compile all the relevant controllers.

3 The Static Sensor Network

Load the *static_net* world. It contains a network of 16 static e-puck robots distributed on a regular grid. Each robot is equipped with an upward facing light sensor sampling a static light field. For this scenario we consider a quad-tree network topology (see Fig. 1), with robots using short-range radio messages to communicate between each other and the highest hierarchical cluster-head providing the only long-range up-link.

Q1 (2): Compile the two controllers (the supervisor: *static_sup.c*, and the robots' controller: *static_con.c*). Note that all the robots have the same controller. Have a look into the *static_con.c* code. How is the quad-tree network topology implemented? How does a robot find its leaves in the network?

Q2 (2): Run the simulation and wait until finished. Read the output messages that is printed on the console. Note that all the messages sent to the supervisor have

come from robot0. Why is this? Which robots forward some messages from other robots? Why?

S3: By running the simulation, the supervisor produces a file containing all the relevant data named *output.m*, in the *Lab10/matlab/* folder. Open Matlab, import the data by running the output file and then run the *evaluate.m* script.

```
>> output
>> evaluate
```

Q4 (2): Have a look at the loaded variables. Which variable contains the sampled set? Which variable contains the reference set? What is the size of each one?

Q5 (3): Have a look at the code of *evaluate.m*. How is the performance of the sensing system computed? Are all the terms of the general metric computed? Why?

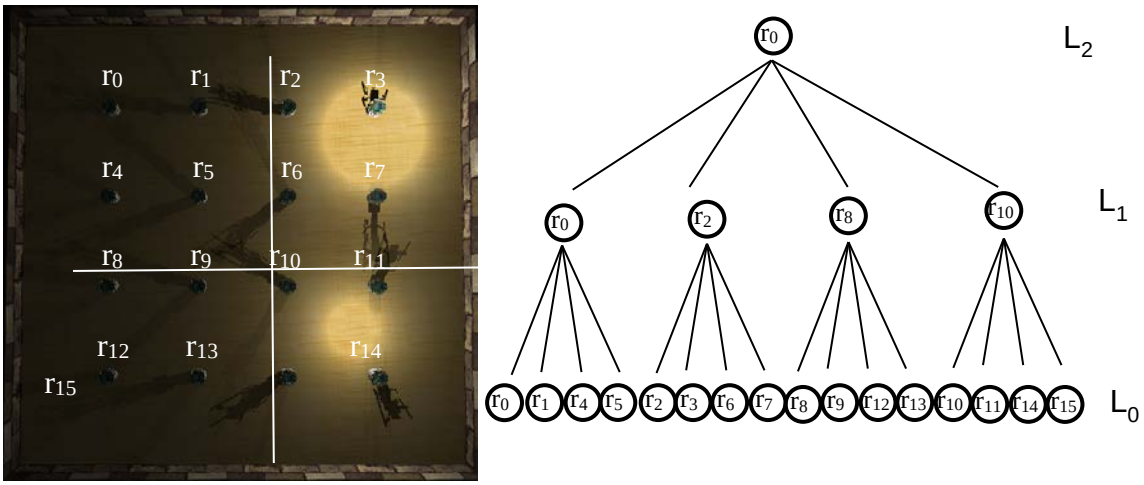

Figure 1: Quad-tree topology of static network

3.1 Spatial Suppression

Since the performance metric that we use balances field estimation quality with communication cost, one way to improve it is by reducing the number of sent messages by suppressing the transmission of measurements from homogeneous areas (with low information value), and maintaining higher resolution in areas of contrast in the underlying target field.

This can be achieved by exploiting the quad-tree network topology. The cluster-heads on each hierarchical level, which in the default controller acted solely as message relays towards the higher levels, can be modified to decide whether the information received from the other cell members contains enough information to be forwarded or should just be dropped.

I6 (11): Load the *space_division_net* world. Edit the incomplete *prun_stat_con* robot controller and implement a threshold-based management for message forwarding through cluster-heads. Compile and run it. (*Hint*: the robot_id variable is used to determine whether the robot is a cluster-head. The part of the code you need to implement is marked with TODO. You should set correct values to three variables: *sw*, *isCellApprox*, and *prunLevel*, so first read the descriptions of these variables in the code. For more information take a look at the function 'send_data' and see how the structure of a message is.)

Q7 (5): What performance do you obtain in this case? How does it compare with uniform spatial sampling method? (*Hint*: use the same Matlab scripts, *output.m* and *evaluate.m*, for evaluating the performance).

Note: The algorithm you have just implemented is a basic variant of backcasting. For a detailed description of how to optimally select the "pruning" thresholds refer to [1].

3.2 Temporal Suppression

Now we consider a direct link between each node and the base station (no tree structure), while the field (light) is dynamically changing.

S8 : Load the *uniform_net world*. Check the codes and see how the network is built.

S9 : Compile the controllers and run the simulation and evaluate the performance of the network in Matlab using the *output.m* and *evaluate_dynamic.m* scripts.

Q10 (5): How is the field changing? Are all points of the field equally affected?

Another axis for adaptive control of a sensing process is the temporal one. Consider a time-division measurement scheduler as the one presented in Fig. 2. Assuming a model of the underlying process, the controller switches to a low sampling frequency when the acquired data matches its model and increases the sampling frequency when the measurements deviate from this model.

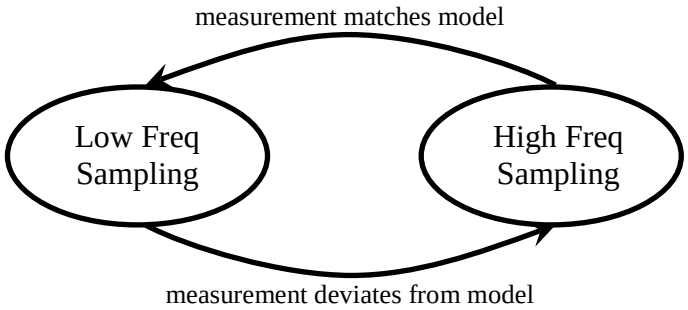

Figure 2: Basic time-division scheduler

I11 (20): Load the *time_division_net* world. Edit the *time_division_con* robot controller and implement a threshold-based algorithm for switching between high frequency and low frequency sampling, based on a linear process model. Compile and run it. (*Hint*: the part of the code you need to implement is marked with TODO. Here is one way to do it: always keep the last two sampled values (e.g., you can name them prevValues[1] and prevValues[0]) and their time stamps (e.g., named prevTstamps[1] and prevTstamps[0]). Note that there is a counter in the loop (named 'count'). With these variables (prevValues, prevTstamps, and count), using a linear model, calculate the expected value in each iteration. If the difference between the sensor reading and the expected value is less than a threshold (let's say 15), then change the Ts to have a lower sampling frequency.)

Q12 (5): What performance do you obtain in this case? How does it compare with uniform temporal sampling method? Run the world and evaluate the performance of the network in Matlab using the *output.m* and *evaluate_dynamic.m* scripts.

4 Mobile Sensor Network

Adding mobility to a sensor network presents the obvious benefits of being able to sample 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.

One broad classification of the different types of mobility relates to whether the measurements acquired by a sensor node play any role in shaping its trajectory. By this criterion we have either controlled or uncontrolled mobility. In the last part of the lab you will learn the impact of mobility of a sensor network on the performance of the monitoring system.

4.1 Uncontrolled Mobility

Load the *mobile_net* world, compile its default controllers and run it. Evaluate the performance of the network in Matlab using the *output.m* and *evaluate_dynamic.m* scripts.

Q13 (5): How does the performance compare with the performance of the temporally-uniform sampling method?

Q14 (5): Edit the dynamic_field_sup controller and increase the dynamics of the field by modifying the FIELD_DYNAMICS (by setting it to 1). How does this affect the performance of the sensing system?

Until this point we did not consider the cost of mobility in the performance metric. In some scenarios this might be a realistic assumption, as the sensor node might be attached to an otherwise independent source of mobility (e.g., a sensor node attached to a city bus, a smartphone attached to a pedestrian, etc.). This particular type of mobility is called parasitic.

S15: Re-evaluate the performance obtained with the randomly moving e-pucks to take into account the cost of mobility by using the *output.m* and *evaluate_mobile.m* scripts.

4.2 Controlled Mobility

The purpose of controlled mobility is to adjust behavior of the robot based on the environmental situation, i.e. the measurements acquired by the sensors node play a role in shaping the trajectory of the robot.

Q16 (5): How would you design a controller capable of guiding the e-puck robots in order to maximize their performance of gathering the information from the field?

In this part you will implement a basic robot controller capable of guiding the e-pucks in order to maximize the information gathered from the field. The behavior of the controller should be the following:

- using its short-range radio link each robot broadcasts locally the location where it has observed the largest measurement gradient
- 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 best location and a vector pointing towards the neighborhood best and a random walk vector

I17 (20): Load the *controlled_mobile_net* world and open in the editor the *guided_con.c* controller. Your task is to implement the strategy explained above in the code. (*Hint*: the part of the code you need to implement is marked with TODO. We have already provided a function named '*compute_wheel_speeds'* which calculates the wheel speeds given a *pose*, *heading* and *intended target point*.) Take a look at this function ('*compute_wheel_speeds'*) and understand what it does. Compile and run your code. Note: tuning the weights of this controller to get a high performance is not in the scope of this lab, so don't waist much time on that.

Q18 (5): What does this algorithm remind you of?

Q19 (5): Compare the performance of the controlled mobility network with the performance of the uncontrolled mobility. Which one is better? Justify your results. Use the *output.m* and *evaluate_mobile.m* scripts.

5 References

[1] A. Prorok, C. M. Cianci and A. Martinoli. Towards Optimally Efficient Field Estimation with Threshold-Based Pruning in Real Robotic Sensor Networks. 2010 IEEE International Conference on Robotics and Automation, Anchorage, Alaska, USA, IEEE International Conference on Robotics and Automation ICRA, 2010.