1 Lab 09: Collective Decision-Making and Distributed Sensing

This laboratory requires the following:

- C development tools (gcc, make, etc.)
- Matlab
- Webots simulation software
- Webots User Guide
- Webots Reference Manual
- Arduino IDE
- 2 DISAL Arduino Xbee kit
- 1 USB cable

Depending on your programming skills, the laboratory duration might require an effort of up to four hours although assistance will be provided in the computer rooms during a time window of three hours. Although this laboratory is not graded, we encourage you to take notes during the course of this laboratory to aid in preparing 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 **S** means that the question can be solved using only additional simulation.
- The notation **Q** means that the question can be answered theoretically, without any simulation.
- The notation **I** means that the problem has to be solved by implementing a piece of code and performing a simulation.
- The notation **B** means that the question is optional and should be answered if you have enough time at your disposal.

To prepare yourself for the exam and to allow you for better time planning during the exercise session, we show an indicative number of points for each exercise between parentheses. The combined total number of points for the laboratory or homework exercises is 100.

2 The DISAL Arduino Xbee kit

The DISAL Arduino Xbee kit was developed at DISAL in 2019. It is composed by an Arduino Mega 2560 board, a custom shield board designed at DISAL and an Xbee

module for communication. The custom shield board allows the Arduino Mega to be interfaced with several sensors and an easy-to-use communication module.

2.1 Arduino Mega board

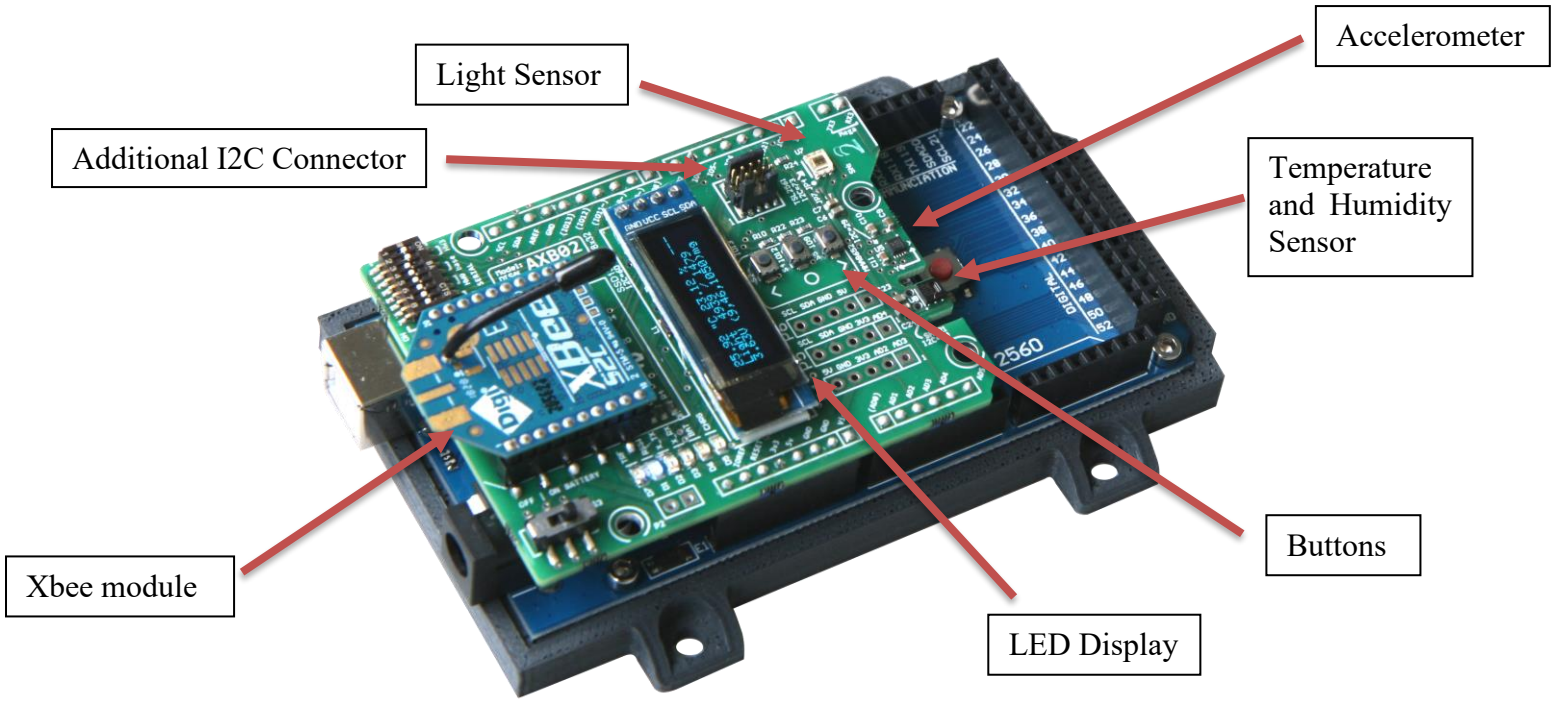
Arduino is an open source electronic platform that provides easy-to-use hardware and software. Arduino boards can be used for a wide variety of development projects, from using sensors, to controlling a motor, to creating your own greenhouse.

The Arduino Mega board was designed by the Arduino company. With its ATMega2560 microcontroller, it is a more powerful board than the company's most renowned Arduino UNO. The Mega board is also bigger with 54 digital I/O pins and 16 analog pins, making it ideal for more complex development projects.

2.2 DISAL shield

The DISAL shield was designed in 2019. This shield board can be plugged into an Arduino Mega or UNO and hosts several sensors and a communication module based on the Zigbee-compliant module Xbee. These modules provide a low-cost and easy-to-use solution for indoor short-range communication. They also consume little power, which makes them ideal for devices powered with a battery. They are compliant with the IEEE 802.15.4 and Zigbee standards for short range, low power networks. These standards allow to build reliable networks of different topology very easily.
Having a pluggable shield board, rather than wiring the sensors to the Arduino Mega by hand, makes it easier and more robust to use sensors. The shield board also provides additional pins to easily connect sensors that are not already present on the board. Moreover, the shield is powered with a battery, which is recharged when the board is connected to a computer.
The DISALshield hosts the following modules:

- Light sensor (TSL2561-T, ams)
- Humidity and Temperature sensor (SHT20, Sensirion)
- Digital Accelerometer (MMA8652, NXP)
- Xbee connector
- LED display
- Buttons
- Additional connector that supports the I2C protocol



2.3 Arduino IDE and code structure

The Arduino Integrated Development Environment (IDE) is an open source software that allows for easy code development for Arduino boards. The Arduino IDE's key functionalities allow a user to write code, compile it, upload it and communicate to the Arduino board. More information and resources about the IDE can be found on the Arduino website (www.arduino.cc/en/Guide/Environment).

Each Arduino program is constituted by two functions: setup() and loop(). The setup() function is called at each power-up or reset of the Arduino board and runs only once. It is used to initialize variables, configure pins, start using a library etc. The loop() function contains the main body of the program. As the name suggests, it keeps looping consecutively.

You can change the language of the IDE by going to *Fichier/Preferences/Choix de la langue.*

2.4 Testing the hardware/software setup

Start by verifying that all the tools we provide you with are working properly

1. Extract the provided tools:

```
tar -xzf lab9.tar.gz
cd Lab9
```

2. Start the Arduino IDE software on your computer (it is already installed in the computer room)
3. Connect one of the two Arduino Mega board to the computer using the provided USB cable

4. In the Arduino IDE, select the type of board you want to upload your code to by going to *Tools → Board (Outils → Type de carte)* and selecting the correct board (Arduino/Genuino Mega 2560 for this lab)
5. Select the port that the board is connected to by going to *Tools → Port* (e.g. dev/ttyUSB0). The name of the port to which the board is connected is important, as it identifies the serial line that will be used to communicate from the computer to the board. If you have multiple boards connected to the computer, you will see multiple port names, each one corresponding to one port.
6. In the Arduino IDE, open the file *helloworld.ino* which is in the folder *arduino/part1/helloworld.* If you click on the file containing the code and start the IDE this way, you will not be able to upload the code to your board due to some permission restrictions.
7. Look at the code. On the top you will see some libraries that need to be added to your Arduino environment. To do this go to *Sketch/Include Library/ Manage Libraries…(Croquis/Inclure une bibliothèque/Gérer les bibliothèques)* and use the search bar to look for the libraries you need to add. Add each of them by clicking on the *Install* button. You will need to add the libraries only once to your Arduino environment, since they are saved in your local configuration.
8. Verify your code by clicking on 
9. If there are any mistakes detected by the verification step, address them and verify again until no error is shown.
10. Upload the code to your board by clicking on 

1. **(S,2)**: Now look at the board, what is happening?
2. **(S,2)**: Try pressing the buttons (look at the figure on page 2 to see where the buttons are), what happens? Can you find the lines in the code that allow this behavior?
3. **(Q,2)**: Look at the code and in particular focus on the content of the functions setup() and loop(). Describe in detail what happens in each of them.

3 Sensing with the DISAL Arduino Xbee kit

3.1 Local sensing with one node

In this section of the lab, you will use the sensors embedded in the shield to monitor basic environmental parameters.

From the Arduino IDE, open the file *ambientlight.ino* which is in the folder *Arduino/part2/ambientlight.*

4. **(Q,2)**: Read the code and try to understand what is happening. What do you think will happen when you upload the code on the board?

5. **(S,2)**: Upload the code on the board (follow again the steps described in Part 1) and verify your assumptions.

Focus on the command "*Serial.print(format("Tsl2561 (full / ir) = %5u /%5u\n", full, ir));"* This command is used to print data to the serial line that is connected to the computer via USB cable. To visualize what is printed open the Arduino serial monitor (*Tools/Serial Monitor*). Make sure that the baud rate is set to 9600 (you can change it in the bottom right corner), which is the same as the one set on the board and used for communication. The baud rate represents the amount of data that is transmitted in a second. You can see that the baud rate is set in the first lines of the *setup()* function in the *ambientlight.ino* code. If more than one Arduino board is connected to your computer, remember to select the right serial port to upload the code to the correct board.

6. **(S,2)**: What gets printed on the serial monitor? How many samples do you get in 20 s? What is the frequency of data transmission?

Close the serial monitor and now let's get acquainted with another very useful tool of the Arduino IDE: the Serial Plotter. With the same code running on the board, open the serial plotter (*Tools/Serial Plotter, Outils/Traceur serie*).

7. **(S,2)**: What do you see? What are these values? Why do you think that there are variations in the signal even when the light conditions do not change?

8. **(S,2)**: Now orient the light sensor on the board towards a more intense light source and then cover it with your finger, repeat these actions a few times. How does the plot change? Write down some examples of values when the light is intense, low and medium range.

9. **(S,3)**: In the code, understand how the frequency of the data transmission in determined. Change the frequency to receive 1 sample every 5 seconds, upload the code to the board and validate your implementation by looking at the serial monitor. Now take a look at the serial plotter. How does the plot compare to the previous questions? What differences do you see in the plot when the light conditions change?

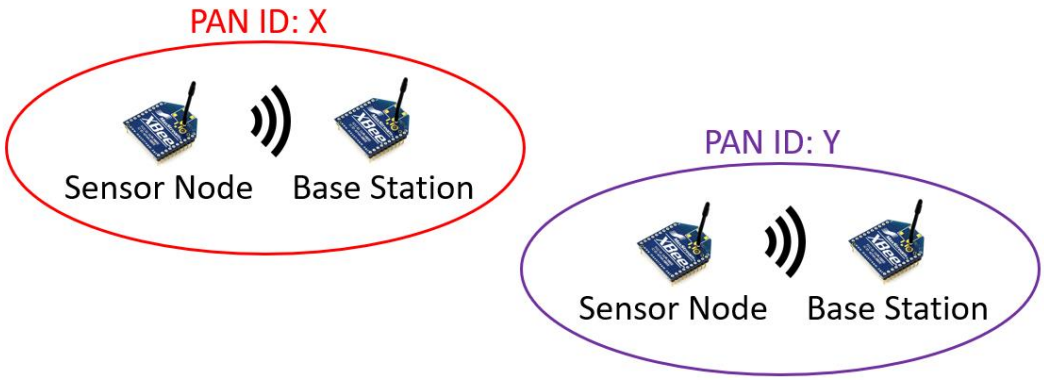
Set the frequency back to 1 Hz. We now want to monitor the ambient light and send a warning to the user of this sensor when the light is too low. Based on the previous question, pick a threshold for which you consider the light to be too low (for example, you should have a warning when you place your finger on the sensor).

10. **(I,5)**: Go to the loop() function and add a few lines of code to first check if the light value is lower than the threshold you selected and then send a warning message on the serial line when this happens. Upload your code to the board, open the serial monitor and check that, when your finger is placed on the sensor, the warning is triggered correctly.

3.2 Remote sensing with two nodes

You will now simulate a remote sensing station using two Arduino boards. In this scenario, one board, the remote sensor node, is powered with a battery and can be placed freely in the room, while the other board is connected to the computer and acts as a base station to receive data from the remote node.

In this laboratory, you will program the remote node to send data to the base station. To accomplish this and make sure that you are sending data to your base station and not to someone else's, you have to configure a parameter called PAN ID. Your Xbees will broadcast data to all Xbees with the same PAN ID, creating a network.

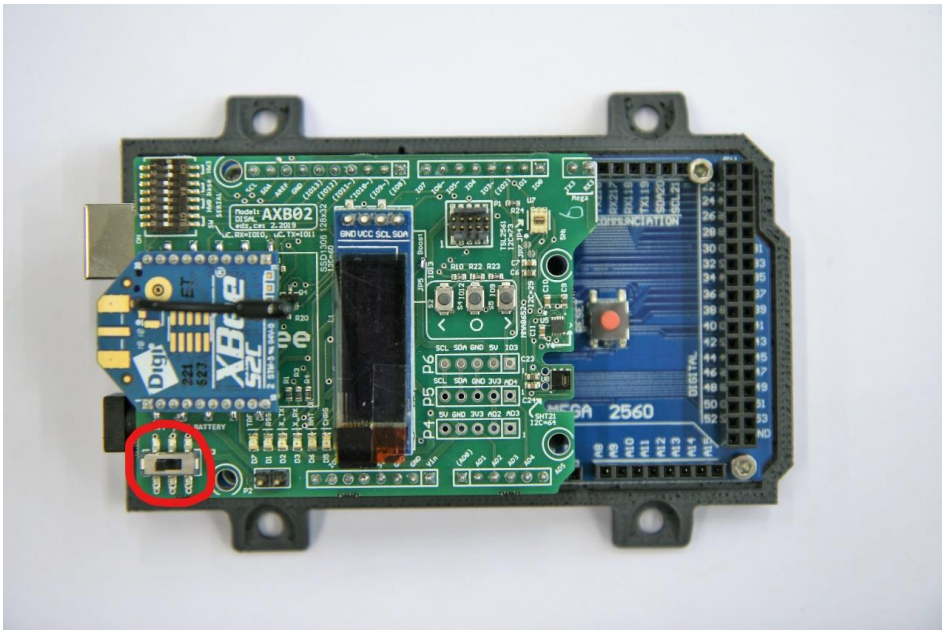


In the Part 3 folder, open the file receiver.ino. Take a look at the code and notice that another serial line is initialized in the setup (Serial3). This serial line is connected to the Xbee and is used to transmit and receive data through wireless communication. At line 33 **change the PAN ID by substituting the number 2019 with the number of your computer** in the computer room (eg., Serial3.print("ATID 3\r");). Upload the code to the board that you are going to use as base station.

11. **(S,2)**: Look at the loop() function. Can you explain what is happening?

Your base station is now ready to be used. Disconnect it from the computer and connect the board that you will use as a remote node.

Open the file transmitter.ino in the part3 folder. At line 48 **change the PAN ID by substituting the number 2019 with the number of your computer** in the computer room (eg., Serial3.print("ATID 3\r");). This number has to be the same as the one in the receiver.ino file. Upload the code to the board and disconnect it from the computer. Turn on the battery switch (circled in red in the figure below), the board should turn on.



12. **(S,2)**: Keep the transmitter.ino code open and connect the base station to the computer. Open the serial monitor (if the serial monitor does not open, try to select the correct Port of the receiver board from the Tools menu). It might take up to 30 seconds for the communication to start. What do you observe? Which sensor is used for data transmission? What is the frequency of transmission?

13. **(S,2)**: Open the serial plotter and observe the data. Let the board send data for about 30 seconds, then place your finger on the temperature sensor on transmitter board and wait for about 30 seconds. Remove your fingers and wait for 60 seconds. How does the data plot vary? Is the sensor responsive? Is it noisy? How does it compare to the response time of the light sensor used in Part 2?

14. **(S,3)**: In the transmitter.ino file, change the value of the SAMPLE_FREQ variable to allow faster sampling, upload the code to the remote node and repeat the experiment described in S2 (pay attention to upload the code to the remote node and not to the base station board!!). Do you observe any differences in the plot? What would happen if you decrease the sampling frequency?

4 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.

4.1 Spatial suppression with the DISAL Arduino Xbeekit

***NOTE**: **For this section of the lab you should team up with one colleague since you will need four DISALkits to solve the following exercise**.*

In Section 3, you experienced how a sensor node works and how you can get data from it into a base station. Now we will create a sensor network composed of 4 static sensor nodes, whose topology can be seen in Fig 1.

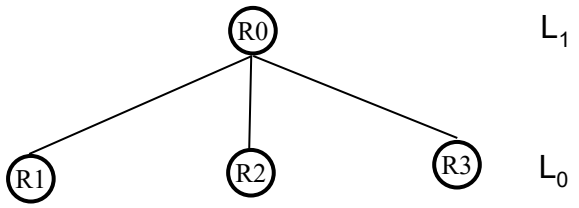


Figure 1: Topology of static network with four nodes

In this network topology, the leaf nodes (R1,R2 and R3) are sending data to the root node (R0), which will then relay the data to the computer. For this implementation, we will use data coming from the light sensors.

One way to improve the communication cost of the network, and consequently its energy efficiency, is to reduce 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 a quad-tree network topology. The root node (or cluster head node in more complex topologies) 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.

In this simplified network, the root node (R0) will decide whether to suppress sensing information coming from the light sensor of its leaf nodes. If the suppression, or pruning, happens, the root node will just send information about its own light sensor to the computer.

In the Part 4 folder, open the file sensornode.ino. **Change the PAN ID by substituting the number 2019 with the number of your computer** in the computer room (eg., Serial3.print("ATID 3\r");). Upload the code on each of your sensor nodes and **change the nodeID (line 27) for each of them** with numbers from 1 to 3.
Open the folder rootnode.ino, **change the PAN ID by substituting the number 2019 with the number of your computer** and upload the code to your root node, that you will keep attached to your computer.

15. **(Q,3)**: Once your network is set up, turn on all the sensor nodes. The value of the standard deviation of the light measured by the network is displayed on the screen of the root node. How does the standard deviation change when you cover one or more sensors with your fingers? Note down some sample values of the standard deviation when the light is homogeneous on all nodes and when it is not.

16. **(I,7)**: Look at the rootnode.ino code and try to understand what is happening. Change the value of the THRESHOLD variable (line 26) to implement the suppression of the values from the leaf nodes when the light field is homogeneous.

Test your implementation by looking at the screen of the base station, a message should appear on the screen of the base station if you pruned correctly.

17. **(I,7)**: Now look at the array called sensorValues. It is used to store the values of the light sensors of all the nodes of the network. The light value of R0 (the root node) is in sensorValues[0], R1 is in sensorValues[1] and so on. Complete the implementation in lines 238,244 and 252 by sending to the computer the light values of all sensor nodes or just of the root node, depending on whether pruning happened. Check your implementation by opening the serial monitor while the base station is attached to the computer.

18. **(Q,4)**: This implementation presents some limitations. What are ways in which you would improve it?

IMPORTANT: What to do before handing the board back to the TAs

Before handing the boards back to a TA, go to the folder *maintenance* and upload the code you find inside on both boards. This code reads the battery voltage and prints it on the OLED screen. This way, we can quickly turn the boards on and see if they need to be recharged. Thank you for your help!

5 Energy Efficiency in a Simulated Static Sensor Network

In the previous section, you experimented with a simplified pruning implementation for a small sensor network. In this section, we will study a larger simulated static network and we will talk about spatial and temporal suppression to increase the energy efficiency of the network.

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.

5.1 Simulated Static Sensor Network

Load the *static_net* world in the *webots/worlds* folder. 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. 2), 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.

19. **(Q,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?
20. **(Q,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?
21. **(S,2)**: By running the simulation, the supervisor produces a file containing all the relevant data named *output.m*, in the *matlab/* folder. Open Matlab, import the data by running the output file and then run the *evaluate.m* script.

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

22. **(Q,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?

23. **(Q,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 presented above computed? Why?

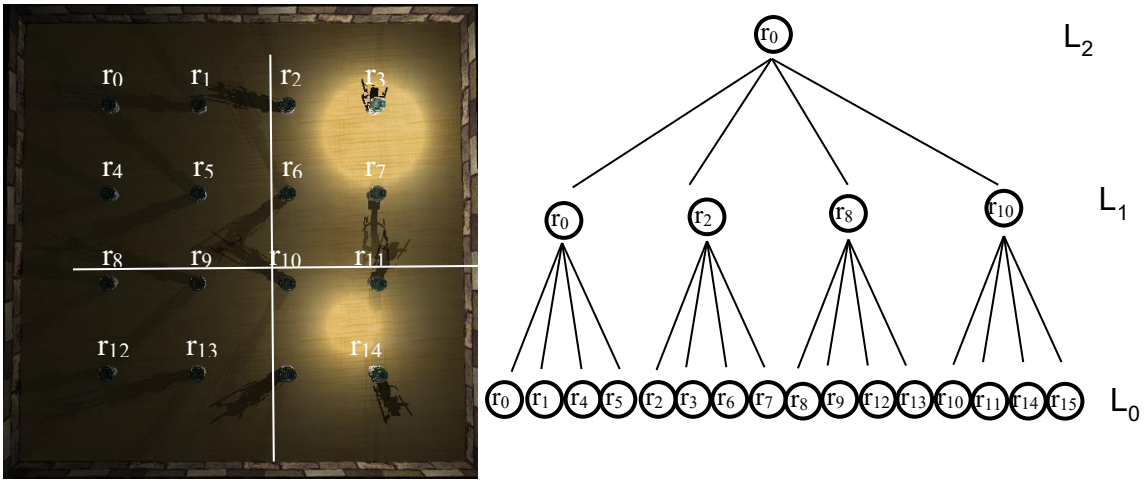


Figure 2: Quad-tree topology of static network

5.2 Spatial Suppression

Since the performance metric that we use balances field estimation quality with communication cost, we will now implement spatial suppression to reduce the number of messages sent. This can be achieved by exploiting the simulated quad-tree topology, with a similar implementation of the one in section 4.

24. **(I,10)**: 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.)

25. **(Q,3)**: 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].

5.3 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.

26. **(S,2)**: Load the *uniform_net world*. Check the codes (the controller `uniform_con.c` and the supervisor `dynamic_field_sup.c`) and see how the network is built.

27. **(S,2)**: 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.

28. **(Q,4)**: 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. 3. 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.

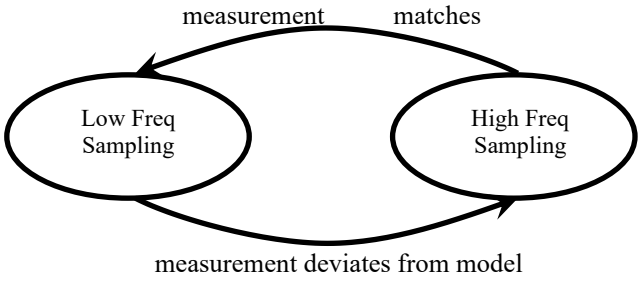


Figure 3: Basic time-division scheduler

29. **(I,12)**: 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.)

30. **(Q,2)**: 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.

6 References

[1] Prorok A., Cianci C. M., and Martinoli A., "Towards Optimally Efficient Field Estimation with Threshold-Based Pruning in Real Robotic Sensor Networks". Proc. of the 2010 IEEE Int. Conf. on Robotics and Automation, May 2010, Anchorage, AK, U.S.A, pp. 5453-5459.