1 Lab 6: Collective Decision in Sensor/Actuator Networks

This laboratory requires the following equipment:

- Webots simulation software (Linux)
- One MICAz radio node module per group of 4 students
- One full e-puck kit per student: an e-puck robot with battery, a communication board, a speaker board and a Bluetooth dongle
- One white arena per group of two students

Office hours

Additional assistance outside the lab period (office hours) can be requested using the dis-ta@groupes.epfl.ch mailing list.

Information

In the following text you will find several exercises and questions.

- The notation $S_x$ means that the question can be solved using only additional simulation.
- The notation $Q_x$ 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 $I_x$ means that the problem has to be solved by implementing a piece of code and performing a simulation.
- The notation $B_x$ means that the question is optional and should be answered if you have enough time at your disposal.

Radio Communication in Webots

A plugin module has been developed for Webots which will allow the simulation of realistic radio communication within a Webots world, using libraries from the OmNET++ network simulation architecture (http://www.omnetpp.org/). Note that as these libraries only exist for Linux, it will not be possible to use the plugin in Windows.

1.1 MICAz Mote

The MICAz is an off-the-shelf platform commonly used for deploying a wireless sensor network (WSN). It operates in the 2.4GHz range, and is IEEE 802.15.4 compliant. TinyOS (http://www.tinyos.net/), while originally designed for the MICA family, has been adopted by several other devices as well, and is rapidly becoming an industry standard. Other features include:

- IEEE 802.15.4/ZigBee compliant RF transceiver
- 2.4 to 2.4835 GHz, a globally compatible ISM band
- Direct sequence spread spectrum radio which is resistant to RF interference and provides inherent data security
• 250 kbps data rate
• Runs TinyOS 2.1
• Has a wide range of available sensor boards, data acquisition boards, etc.

1.2 e-puck Communication Module

The e-puck robots in this lab will be capable of wireless communication via a radio board (see Figure 1-1). This board uses the same Zigbee protocol as the sensor motes, allowing e-pucks to communicate with motes, as well as with each other. The communication range of the board can be adjusted from approximately 10 centimeters to 5 meters. We will use only short range radio communication with the e-pucks in this lab.

![Figure 1-1: e-puck Robot with Radio Module](image)

1.3 Collective Decision

One difficulty with distributed intelligent systems is that there isn’t a central arbiter to make global decisions. For example, how can a hive of bees decide where to construct a new nest, or a cockroach society pick a shelter? To overcome this limitation, the multi-unit system must use an effective method for making a collective decision. In distributed robotics, this usually is accomplished by each robot forming an opinion on the issue, and varying that opinion over time based on the interactions it has with other members of the system until the entire swarm or network converges on a single opinion. Often the interactions within the distributed intelligent system are very limited, making rapid convergence difficult.

1.4 Hybrid Networks

When using simple hardware devices such as miniature robots and sensor nodes, units tend to be limited in their capabilities. Often tasks will require functionality which isn’t available to a single type of device. In these cases, it can be helpful to use a hybrid network, which is composed of two or more types of devices working in tandem. In this way, very simple units can be combined to accomplish more complex tasks.
2 Lab

2.1 Wireless Sensor Networks as Distributed Intelligent Systems

Another type of multi-unit system in which we can potentially apply the principles of distributed intelligence is a wireless sensor network. Note the similarities between a distributed, networked robotic system and a wireless sensor network: in fact, both systems can be seen as specific instances of a broader category of “sensor and actuator networks”, or according to a recent labeling adopted by the research community we could classify them as instances of “distributed cyber-physical systems”.

2.2 Collective Decision in Real World Hybrid Networks

We are going to try a collective decision with real hardware. To do this, you have to team up with another student. Your team should have 2 e-pucks, each equipped with a communication board.

Start your computer with Linux and plug in the Bluetooth dongle. You will need to download the e-puck development environment onto your computer. Copy these files in your home directory by doing the following:

a) Download (checkout) the necessary files from the e-puck subversion repository by typing the following (in one line):

```bash
$ git clone https://disalgit.epfl.ch/epuck/epuck.git
```

The download might take a few minutes.

b) Inside the `epuck/EpuckDevelopmentTree/` directory, you will find a `library/` and a `program/` directory. In order to use the e-puck library code that you just downloaded, it must be compiled. Do this by going to the `library/` directory and executing `make`:

```bash
$ cd epuck/EpuckDevelopmentTree/library
$ make
```

Download the lab code from Moodle: lab06.tar.gz

```bash
cd myfiles/My\Documents
tar xvf lab06.tar.gz
```

Open a new terminal, compile and program your e-pucks with the provided controller using the following commands (where `##` should be replaced with the numbers of the e-puck to program):

```bash
cd lab06/collectivedecision
make
```

Turn on the robot.

```bash
epuckupload -f collectivedecision.hex ##
```

If you see a box asking you for an ID enter 00## where `##` should be replaced with the ID of the epuck. As soon as dots appear on your screen (`$.....`) press the
blue reset button on the robot. This will start writing the program to the e-puck memory. Now stars will appear on your screen, indicating that the robot is being programmed ($\ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots 

When the robot is programmed, we need to calibrate its IR sensors. Make sure the robot is turned off. Then set the selector switch on top such that the arrow is pointing to the rear of the robot. This puts the robot into calibration mode. Put the robot in a spot where it is at least 30 cm away from any obstacle. Turn it on and make sure you remove your hands. The red LEDs will come on briefly while the IR sensors are calibrated. When the red LEDs turn off again, the IR sensors are calibrated and the calibration information is stored in non-volatile memory. This means that you only have to do this calibration once for each robot. Turn the robot off and change the selector switch so that the arrow points to the front of the e-puck.

When you switch on or reset your e-pucks, they will randomly choose either right wall following (state R) or left wall following (state L). The collective decision algorithm then works as follows: each e-puck listens for messages during time $t=1s$ and counts the number of left wall follow messages ($n_L$) and right wall follow messages ($n_R$) that it receives. If the majority of received messages coincides with its own state, it keeps that state—otherwise, it changes the state with probability $p=0.5$. Right afterwards, the e-puck locally broadcasts a message with its own state (independent of whether the state was changed or not). Note that such messages are received by e-pucks in the vicinity (~ 15 cm) only.

Note that the e-pucks are not synchronized with each other. (You can neglect the case where two e-pucks simultaneously transmit their state.)

**Q1(5):** Switch on your e-pucks and put them in different corners of the arena. Can you observe collective decision? About how much time does it take until both e-pucks are in the same state?

**Q2(15):** With just two e-pucks, the system can be in one of four possible states, as shown in the following graph. Each time one of the e-puck takes a decision after having received a message of the other e-puck the system state possibly changes. Complete the graph with all possible transitions. What are their probabilities?

![Graph](image)

**Q3(10):** Which system states are stable (absorbing states) according to the graph? (Stable states are states which the system doesn’t leave once it has reached them.)

Team up with another group now. You should have 4 e-pucks and 1 MICAz sensor node in your team. The ID number (labeled “M###”) of the MICAz that you have is the group ID that it is programmed with. In order to have the e-pucks
communicate though the MICAz node, you need to specify this group ID in the file “lab06/collectivedecision/collectivedecision.c”. Open the file and change the two hashes (“##”) in the line:

```c
#define GROUP_ID 0x##
```
to the MICAz’s group ID.

Compile and reprogram all four e-pucks with your modified code. Place the sensor node outside (between) your two arenas with e-pucks still inside them. The e-pucks can then occasionally communicate with the node, which can relay messages between the two arenas; hence forming a backbone network between the e-pucks. Note that while e-pucks are configured to send messages within a range of about 15 cm only, the motes have enough transmission power send messages in the whole room. (This is why we need the team number to separate between different teams: the motes simply discard all messages that don’t have the same team number.)

Q4(5): Switch on the MICAz and reset the e-pucks. Do you observe collective decision? How long does it take now?

Q5(5): Now you will intentionally try to influence the decision. Turn the dial of one e-puck to either 90° left or 90° right, for fixed left or right wall following, respectively. (If the dial is in an invalid state, all the LEDs will be turned on.) What happens now? Why?

2.3 Collective Decision in Webots

We’re now going to work with e-puck robots in Webots endowed with a simulated copy of the radio module, which leverages the OmNet++ radio communication plug-in. The collective decision algorithm is exactly the same as before. Don’t forget to compile the controllers before opening the world!

Q6(5): Open the world collective_decision.wbt. This world contains 10 robots with random initial wall following states. Observe the default settings for broadcast frequency, radio communication range, and opinion change probability. Does a collective decision happen now? If so, how long does it take? (You may want to run this in fast mode.)

Q7(5): Try changing the opinion change probability (OPCHANGE_PROB) of the robots. What happens for very low probabilities (e.g. 0.01)? What happens for very high probabilities (e.g. 0.99)? Why?

Q8(5): Increase the decision interval (DECISION_INTERVAL) to t=2s. How does this affect the time until a collective decision happens? What happens if you increase it to t=20s? Why?

Q9(5): Now change the radio range (COM_POWER) from 0.001 to 1 (a much larger value). What happens now? Why?

A scalable approach to solving this consensus problem might be for the robots to use a threshold-based algorithm, like the one previously deployed for task allocation purposes, for deciding whether to change their current state. Assume that all robots occasionally broadcast the state which they are in (either “L” or “R”) and that your
robot therefore always knows $n_L$, the number of robots in state “L” and $n_R$ the number of robots in state “R” within his communication range.

$Q_{10}(10)$: What would be a sensible value for the stimulus $s$? What would be a sensible value for the threshold $\theta$? Please indicate $s$ and $\theta$ in terms of $n_L$ and $n_R$.

$Q_{11}(15)$: Describe (don’t implement!) an algorithm with the following two stable states:
- 10 robots doing left wall follow, 5 robots doing right wall follow
- 10 robots doing right wall follow, 5 robots doing left wall follow

The system must converge towards these states. You can assume that each robot has a unique ID in the range 1 – 15 and that robots don’t fail/crash. (In addition, you can assume that each robot receives an infinite number of messages from all other robots in any infinitely long time interval. In other words: there is enough randomness in the system so that e-pucks meet with each other from time to time.)

*Hint*: This is equivalent to having all 15 robots doing left wall following, with exactly 5 of them also playing a sound on their speaker.

$Q_{12}(15)$: How would you have to extend your algorithm if the 15 robots had unique IDs in the range 1 – 200 (with some numbers left out)?