Robots Collective Behavior Demos

using Embedded Programming with E-Pucks vers.1 (dsPIC) and Sound

BACHELOR THESIS

Author:
Jean-Roch LAUPER [1]

Supervisor:
Dr. Julien NEMBRINI [2]

Professor:
Dr. Denis LALANNE [3]

UNIVERSITY OF FRIBOURG
(SWITZERLAND)

Faculty of Science and Medicine
Department of Informatics

November 1, 2020

[1]jrlauper@yahoo.fr, University of Fribourg
[2]julien.nembrini@unifr.ch, University of Fribourg
[3]denis.lalanne@unifr.ch, University of Fribourg

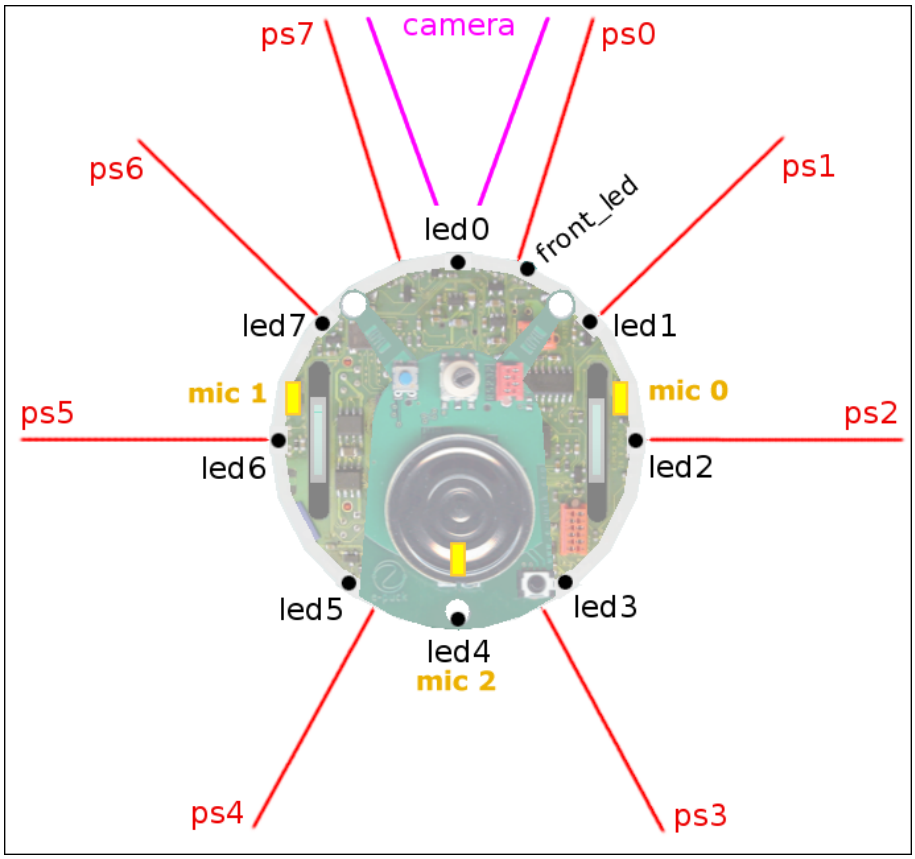


Figure 1: Schema of the e-puck with its platform from above.
"ps" are proximity sensors and "mic" are micros.
(Modified version of a figure from Cyberbotics Ltd (2020))

Arena	# robots (suggested)			File	Function	Demo link
			Initialization	main.c		https://youtu.be/fx46Hbjesd4
180x140cm	2 - 15+ (9 - 11)	0	Swarm Coherence Algo. Alpha threshold 6-1 (cycles of 50)	run_alpha_algo.c/h	run_alpha_algo_from_to(6,1,50)	https://youtu.be/PN_FrZWih0Q
"	"	1	Swarm Coherence Algo. Alpha threshold 5	"	run_alpha_algo(5)	https://youtu.be/n0Ww2lsuaYE
"	"	2	Swarm Coherence Algo. Alpha threshold 3	"	run_alpha_algo(3)	"
"	"	3	Swarm Coherence Algo. Alpha threshold 1	"	run_alpha_algo(1)	"
140x70cm	1 - 5+ (5)	4	Collective Heap Buliding."Swiss XP"	runbraitenberg_mod3.c/h	run_braitenberg_swiss_EL()	https://youtu.be/AevML_FSlEM
	2 - 15+	5	Synchronicity through Beep Sounds. "Chirping"	run_chirping.c/h	run_chirping()	https://youtu.be/I6GSagp46zs
To choose	1 - 15+	6	Braitenberg Explorer	runbraitenberg_mod3.c/h	run_braitenberg_explorer()	https://youtu.be/yz6ijhXEd9Y
"	"	7	Braitenberg Explorer/Lover	"	run_braitenberg_explorer_and_lover()	https://youtu.be/iLH2aBzI9MI
	1	8	IR Piano & Full Sound Demo (paper)	e_freq_sound.c/h	freq_ir_piano_C_to_C1()	https://youtu.be/VEd9eDacvss
	"			"	freq_video_full_demo()	https://youtu.be/udtJUNvuzgM
	1	9	Robot Guiding by Whistling (GCTronic)	runfftlistener.c/h	run_fft_listener()	https://youtu.be/uaQSAEbv7jI
	1 - 15+	10	Rotating toward Hand Clapping (GCTronic)	runlocatesound.c/h	run_locatesound()	https://youtu.be/zhrk4egCTJU
To choose	1 - 15+	11	Rotating toward Light	runbraitenberg_mod3.c/h	run_light_lover()	https://youtu.be/kJHteD2_1GQ
"	"	12	Explorer & Light Follower	"	run_moving_light_lover(true)	https://youtu.be/uTkaBSoXj2o
	1	13	Simon Game: via Proximity Sensors or Whistling	run_simon.c/h	run_simon()	https://youtu.be/mGj8a8Cdw6Y
		14	Selector corresponding to IR Remote Control use			
		15+	Battery Charge Display			

Table 1: Full list of programs included in the demonstration set
Full playlist: `https://www.youtube.com/playlist?list=PLrscHgSUZPdr38tirAsB4_4Q93khKP9Rv`
Github: `https://github.com/jrlauper/jrl_epuck/tree/master/jr_demo`
"(GCTronic)": the code comes from GCTronic demos with slights mods
(more precisely programs 2 and 9 in complete demo 2: see section D.2).
"15+": 15 and beyond.

Contents

Contents (detailed)

List of Figures

List of Tables

List of Abbreviations

DCI	Data Converter Interface
DFT	Discrete Fourier Transform
DSP	Digital Signal Processing
FFT	Fast Fourier Transform
HWRev	E-Puck Hardware Revision. There are three HWRev of e-pucks 1: HWRev 1.1. (2006), HWRev 1.2. (2008), and HWRev 1.3. (2014). For more details, see (GCtronic, 2019c)
MIPS	Million Instructions Per Second.
RC	Remote Control
IR	Infrared
UART	Universal Asynchronous Receiver/Transmitter

1 Introduction

In 2018, the computer science department of the University of Fribourg (Switzerland)[1] acquired new more powerful robots for its robotics lectures. This left the previous older ones free for other uses. With my supervisor, we decided to use them for a series of demonstrations aimed at schools and university visitors. The robots in question are first-generation e-pucks described in chapter 2.

Several key points guided the selection and the implementation of the demonstrations. First, the most suitable choice for such demonstrations sounded to be able to use the robots *directly*, "out of the box", i.e. without the need for any other external means such as extra computers. For this reason, we opted to program the robots in an *embedded* way. Second, after some repairs, we had quite a large number of fully functioning robots at our disposal: 16 in total. Wanting to take advantage of this situation, we decided to favor demonstrations focusing on *collective* behaviors. Finally, given the targeted audience, we looked for projects interesting from an educational point of view while representing classical robotics experiments.

In the main part of this report, I present the three main demonstrations I carried out:

I. Collective heap building process (chapter 3) (based on Maris and Boeckhorst (1996)):
Although not directly programmed to such an end, robots gather polystyrene cubes and create heaps thanks to their morphology and their interaction with their local environment.

II. Decentralized synchronization between robots using sound (chapter 4)
(based on Mirollo and Strogatz (1990))
Without any external help, robots achieve to beep together in a synchronized way after a series of inner adjustments.

III. Decentralized swarm coherence using sound (chapter 5)
(based on Støy (2001), Nembrini (2005) and Nembrini et al. (2002))
Robots remain together in some radius while "exploring around", thanks to the volume of the beeps they emit at a regular pace.

These three demonstrations are examples of *behaviour-based robotics*.[2] This approach emphasises the importance of exploiting the actual environment in which the robots are situated, and the possible interactions between them. It contrasts with traditional artificial intelligence where actions are planned relying on a internal representation of the environment. In behavior-based robotics, actions are taken directly based on sensor inputs, the robots gradually adapting their behavior through their interactions with their environment (cf. Støy 2001, p.2 and "Behavior-based robotics",Wikipedia 2020). It should be added that such an approach is particularly well suited to perform a task with multiple robots – one of our criteria. Moreover, in contrast to a deliberative approach, which uses planning and internal model, such an approach is not very resource demanding (cf. Michaud and Nicolescu, 2016, pp. 307–313). This makes it especially well adapted to e-pucks, which have rather limited material resources.

[1] https://www3.unifr.ch/inf/

[2] Another well-known example of this robotics approach are Braitenberg vehicles (Braitenberg, 1984).They will appear several times in this work and are described in the section 3.2.1.

In order to expand the range of available demonstrations while taking advantage of the manual selectors still available on the e-pucks (they are 16 in total), further demonstrations complete the three main projects. They are described in chapter 6. There you will also find a complete list of the programs included in the final demos package.

As the reader will notice, *sound* plays a prominent role in many of the demonstrations of the final set. And the three main projects are no exception to this rule: sound is used as a means of communication between the robots in the main projects **II.** and **III.**. The use of sound seemed especially suitable for demonstrations, as it is directly perceptible by the audience – unlike, for example, infra-red or radio waves. Personally, we also simply wanted to be able to use the e-puck's loudspeaker and microphones – something we hadn't been able to do before in other experiments with them.

A large part of the work involved in making the demonstrations was to familiarize ourselves with the *embedded* programming of the e-pucks, and to develop a range of general hardware and software solutions able to address specific challenges raised by them. For space reason, the description of these different elements couldn't be kept in the main part of this report, but it had to be relegated into the appendices. These elements, although less relevant to the general reader, should find all their interest for people working or wanting to work with e-pucks in an *embedded* way. They includes very probably the greatest original contribution of the current work.

- In Appendix A, we shortly present the main resources available to program the e-puck in an **embedded** way.
- In Appendix B, Iweshow how we can use e-pucks to work with **sound**:
  - sound production (includes original software libraries to easily produce sound with e-pucks),
  - volumes and frequencies recognition,
  - and inter-robot communication.

  Using sound to work with e-pucks presents, surprisingly, many major challenges – a fact reflected in the significant length of this part.
- In Appendix C, we present other general aspects we had to deal with in order to carry out the three main projects mentioned:
  - the problem raised by the **mutual non-detection** of robots through the infrared proximity sensors (C.1);
  - a **pseudo-random** number generator (C.2), and
  - a module suitable to use an infrared **remote control** (C.3).

As the appendices have not been fully reviewed by our supervisor, we are of course solely responsible for any errors that may be found in it.

Some last notes before starting:

- **[GitHub]** All the code of the demos as well as the files mentioned can be found on the GitHub of this work : `https://github.com/jrlauper/jrl_epuck`

  If some original part of this code is used or mentioned, its reference should mention the author as well as its github location. A possible way to do it would be, for instance:

  > LAUPER, Jean-Roch (2020). "E-Puck demos". `https://github.com/jrlauper/jrl_epuck/tree/master/jr_demo`

- **[Software Module]** In Appendix B and Appendix C, when some section title starts with "[Software Module]", this means that the section describes some piece of software that can be reused as such by other people working with e-pucks in an embedded way.
- **[Pictures and Figures Used]** When no source is specified for some picture, figure or graph, we are the author of it.
- **[Videos]** All the videos connected to this report are gathered in the following youtube playlists:
  - demos: `https://www.youtube.com/playlist?list=PLrscHgSUZPdr38tirAsB4_4Q93khKP9Rv`
  - documentation: `https://www.youtube.com/playlist?list=PLrscHgSUZPdrjsOhXDUBphgNyq3NGuZ5R`

2 The e-puck and its main features

Contents

2.1 General overview

The e-puck is a small robot developed by EPFL[1] for engineering education. Its design was completed in 2005. Figure 2.1 show the robot with its dimensions and main features. We can notice the small dimensions of the robot: 7 cm in diameter without bumper. It was one of the key motivations behind its design. Indeed, it allows students to perform experiments with the robots without the need for large spaces – contrary to other educational robots existing before the e-puck (cf. Mondada et al., 2009).

[Sensors] In its standard version, the e-puck is equipped with the following sensors:

- 8 infrared proximity sensors,
- 3 microphones,
- 1 front camera,
- 1 3D accelerometer, and
- 1 infrared receiver (to be used e.g. with a remote control).

[Actuators] As for its actuators, they are the following ones:

- 2 independent wheels capable of stepped movements,
- 1 speaker placed on a small upper platform,
- a circle of 8 red leds placed under its transparent bumper,
- 1 directional front led just beside its camera, and finally
- 1 green led able to illuminate its transparent body.

Figure 2.2 shows the position of some key sensors and actuators of e-puck's body (top view).

[1] École Polytechnique Fédérale de Lausanne – Swiss Federal Institute of Technology in Lausanne (Switzerland).

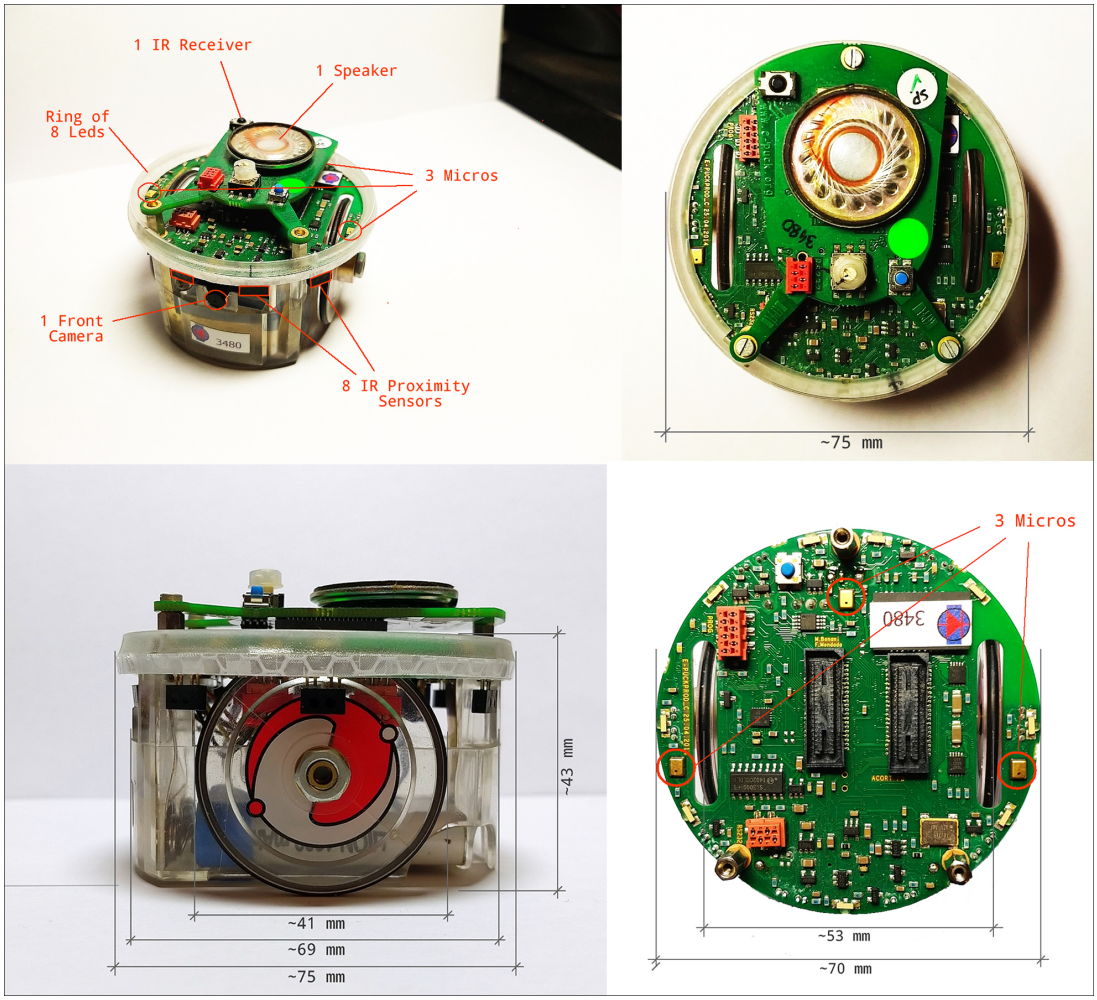


Figure 2.1: E-puck main features and dimensions

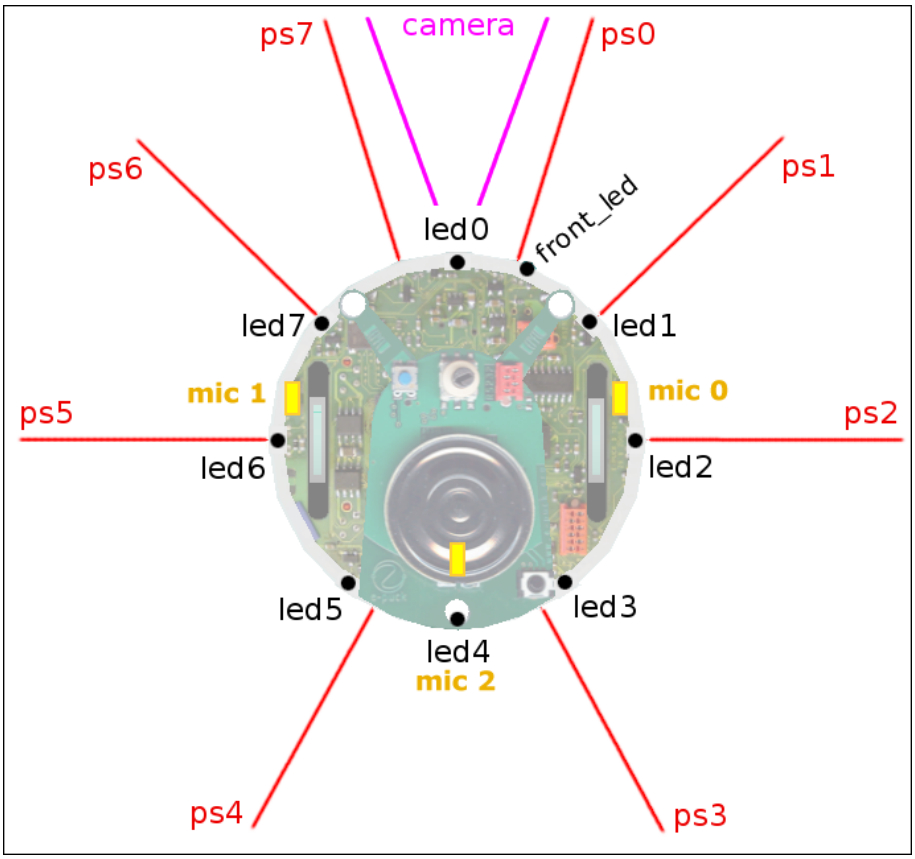


Figure 2.2: Schema of the e-puck with its platform seen from above.
"ps" are proximity sensors and "mic" are micros.
(Modified version of a figure from Cyberbotics Ltd (2020))

The actuators and the sensors of the e-puck can both be extended through extra hardware modules that can be added to the robot. To know more about these possible extensions, see *E-Puck Main Wiki* (GCtronic, 2019c) and, above all, the shop part of the GCtronic site dedicated to the e-puck extensions: this one contains a multitude of links toward descriptions of these extensions (*GCtronic Shop*, GCtronic 2019a)

The **microcontroller** at the heart of the e-puck is a dsPIC 30F6014A made by Microchip Technology. Its main features are the following ones:

- 16-bit with a DSP Core
  (modified Harvard architecture – 24-bit wide instructions, 16-bit wide data path);
- 60Mhz, 15 MIPS;
- 8 KB RAM;
- 144 KB flash storage.

More info about the dsPIC 30F6014A can be found on its dedicated webpage (Microchip Technology Inc., 2019) :
`https://www.microchip.com/wwwproducts/en/dsPIC30F6014A`
and in its complete data sheet (Microchip Technology Inc., 2011):
`http://ww1.microchip.com/downloads/en/DeviceDoc/70143E.pdf`.

Three different versions of the e-puck appeared over the years. The differences between them involve numerous hardware components: the camera, the bluetooth module, the accelerometer, the addition of a gyroscope, the microphones and the speaker. The details of these differences can be found in the *E-Puck Main Wiki* (GCtronic, 2019c). Among these differences, the ones affecting sound have a direct impact on the projects presented this work. These differences are summarized in Table 2.1. More info about which sensors and actuators are used in each project is given in section 2.4.

Version	Production Year	Speaker	Microphone	Version
HWRev 1.1	2006	(opaque black) Produces **Louder** Sound	Less sensitive (about 15%)	HWRev 1.1
HWRev 1.2	2008	(transparent) Produces **Lower** Sound		HWRev 1.2
HWRev 1.3	2014		More sensitive	HWRev 1.3

Table 2.1: Sound hardware differences between e-puck versions
(*E-Puck Main Wiki*, GCtronic 2019c)

Among the 16 e-pucks at our disposal to make the demonstrations described in this work, 5 were HWRev 1.1 and 11 were HWRev 1.3.

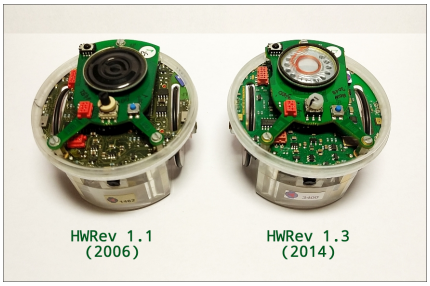


Figure 2.3: Appearance difference between e-puck HWRev 1.1. and HWRev 1.3

2.2 Further information

Here are the places to consult if you want more information about the e-puck:

- For further general information about the e-puck:
  - the three following brochures:
    - GCtronic (2006): *E-Puck. EPFL Educational and Research Mini Mobile Robot*: https://www.gctronic.com/files/flyere-puck.pdf,
    - GCtronic (2008): *E-Puck Mini Doc (Ver. 1.0). EPFL Educational and Research Mini Mobile Robot*: https://www.gctronic.com/files/miniDocWeb.pdf,
    - Cyberbotics Ltd (2015): *E-Puck. Mini Mobile Robot from EPFL*: https://www.cyberbotics.com/e-puck.pdf, archive https://web.archive.org/web/20170710203516/https://www.cyberbotics.com/e-puck.pdf";
  - the section of the Webots simulator user guide devoted to the e-puck : Cyberbotics Ltd (2020): https://cyberbotics.com/doc/guide/epuck ; and
  - the Wikipedia page dedicated to the e-puck: Wikipedia (2019a): https://en.wikipedia.org/w/index.php?title=E-puck_mobile_robot&oldid=910935510 .
- For a much more detailed description of the robot and the motivations behind its design, see the reference article: Mondada et al. (2009): "The e-puck, a Robot Designed for Education in Engineering".
- For precise schematics of the e-puck and its different parts, the "robot" section of the E-Puck.org (2018) website is the place to look: http://www.e-puck.org/index.php?option=com_contentview=articleid=2Itemid=8.
- To really work with the e-puck, the most complete and up-to-date source is definitely the *GCtronic E-Puck Main Wiki* (GCtronic, 2019c): https://www.gctronic.com/doc/index.php/E-Puck.
  The only exception is the **libIrcom** library – allowing communication via the infrared sensors – which can only be found on e-puck.org (Campo et al., 2008): http://www.e-puck.org/index.php?option=com_content&view=article&id=32&Itemid=28.

Except otherwise mentioned, all information contained in the first part of this chapter comes from the sources mentioned in this section.

2.3 The e-puck 2, since 2018

Till this point, we've been talking about the first version of the e-puck only – referred to by "e-puck1" henceforth in this subsection. However, since 2018, a completely new version of the e-puck exists: the e-puck2. Like its predecessor, it is the result of the collaboration of the EPFL and GCtronic. This new version is an evolution of the e-puck1. As a consequence, all the previous sensors and actuators are there as before or in an improved manner. This should guarantee full backwards compatibility with projects previously made for the e-puck1 with high level programming tools such as Webots[2]. New sensors and actuators such as some RGB

[2] For more info about Webots, see for instance: https://en.wikipedia.org/wiki/Webots

leds have also been added. All the hardware differences between the e-pucks 1 and 2 can be found on the main wiki of the e-puck2 (GCtronic, 2020b, "specifications" section):
`https://www.gctronic.com/doc/index.php/e-puck2`.
More details about the e-puck 2 can also be found on its dedicated webpage (GCtronic, 2020a):
`https://www.gctronic.com/e-puck2.php`
or in its official flyer (GCtronic, 2018).

One major change between the e-puck 1 and 2 is their processor and the memory coming with it: the processor of the e-puck2 is ∼10 times faster, its RAM is ∼24x larger, while its flash storage is ∼7x larger – improvements that give us a much more powerful robot overall. More precisely, at the hardware level, we passed from:

- a 16-bit dsPIC from Microchip to
- a 32-bit ARM-cortex M4 from STMicroelectronics: the STM23F407 (STMicroelectronics, 2016).

This change of processor has a crucial consequence for the current work which used e-puck1s only. Indeed, all the demonstrations and projects described in it are programmed in an *embedded* way, making low-level hardware calls and using some libraries specifically programmed in assembler for the dsPIC. Therefore, although the e-puck2 provides backwards compatibility when it comes to Webots or other high level programming tools, anything that was programmed in an embedded way for the e-puck1 should normally be incompatible with the e-puck2.

From now on and in the rest of this document, when we'll talk of the "e-puck", it will be assumed that I am talking of the e-puck1 and not its newer version.

2.4 Sensors and actuators used in the main projects

In the main projects of this work – projects described in chapter 3, 4 and 5 – we use a great part of the sensors and actuators available on the standard version of e-puck, but not all: e.g. none of the projects make use of the front camera or the accelerometer. The sensors and actuators used in each of them are summarized in Table 2.2.

	ch.3 Heap Formation	ch.4 Synchronization	ch.5 Swarm Coherence	Note
A. Sensors				
IR Receiver	(✓)	(✓)	(✓)	Program Selection via Remote Control
IR Proximity Sensors	✓	(✓) collision avoidance during the movement accompanying the beeps	✓	
Micros		✓	✓	
B. Actuators				
Wheels	✓	(✓) movement accompanying the beeps	✓	
Leds (all)	✓	✓	✓	
Speaker		✓	✓	

(✓) Used but non-essential to the experiment

Table 2.2: Sensors and actuators used in the main projects

3 Collective heap building

Contents

Sensors: IR proximity sensors (IR receiver).
Actuators: motorized wheels (leds).

In this chapter, we describe how we adapted the experiment of Maris and Boeckhorst (1996)[1] to carry it out with e-pucks. In this experiment, robots perform an apparently complex coordinated task generating patterns: they form heaps by moving and gathering white cubes in an arena. However, the robots actually follow very simple rules of behavior, and the task is performed by exploiting physical constraints: the morphology of the robots, the arena and the cubes, and the possible interactions between them.

- Files location: https://github.com/jrlauper/jrl_epuck/tree/master/jr_demo
- Main files: `runbraitenberg_mod3.c/h`
- Function: `void run_braitenberg_swiss_EL()`

- Robots Selector: 4.
- Demo Video: `https://youtu.be/AevML_FSlEM`
- Arena Size: approx. 140x70[cm].
- Robots Number: 1 to 5; suggested 5.
- Cubes Number: 12 to 25; suggested 25.
- Recommended configuration to get a nice result: 5 robots / 25 cubes / 15-20′ duration.

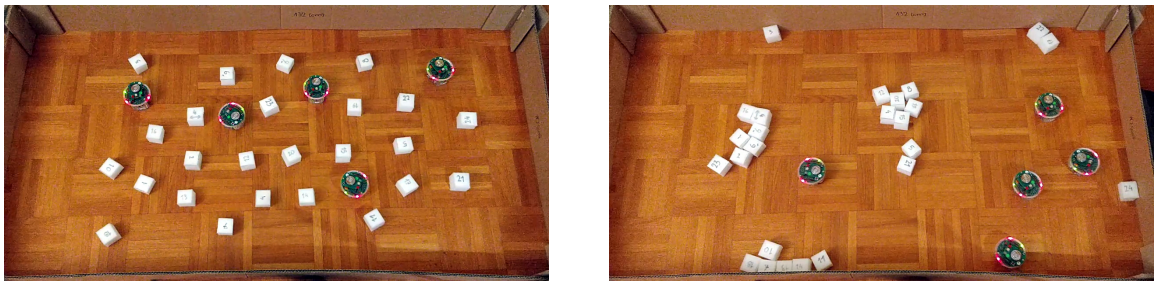
[1] "Exploiting Physical Constraints: Heap Formation Through Behavioral Error in a Group of Robots".

- Reflective tape: for the avoidance behavior, it is assumed that the robots are equipped with reflective tape (3 [mm] on the lower part of the bumper; see 3.4.1 Sensors and avoidance behavior below for more details). If this is not the case, the code has to be modified accordingly.

3.1 Task description

Let's start by describing how the experiment unfolds when an external observer watches it:

> At the beginning of the experiment, 12 to 25 white polystyrene cubes are randomly disposed in an arena. The operator randomly places 1 to 5 robots within it, switches them on, and the robots start moving. While moving around, the robots sometimes push polystyrene cubes, either to form piles or to "store" them against the arena walls. During the course of the experiment, the robots avoid each other and avoid the walls of the arena. After about 20 minutes and depending on the number of robots and cubes, 1 to 3 heaps of cubes are formed in the arena.



Start of the experiment After 30'

Figure 3.1: Collective heap building: start and end of an run of the experiment adapted to e-pucks (with 5 robots and 25 cubes)

3.2 Original implementation (Maris and Boeckhorst, 1996)

To achieve this heaps formation, Maris and Boeckhorst (1996) combines two things:
- some basic robot behavior (3.2.1) and
- some physical constraints: the morphology of the robots, the cubes and the arena (3.2.2).

It is the interaction between these two elements that will give the desired result.

3.2.1 Robots basic behavior

The basic behavior used in the experiment is a variant of Braitenberg vehicles (Braitenberg, 1984, p.6-14). In their most basic version, Braitenberg vehicles consist of two sensors and two motors connected together.

- The connection between sensors and motors can be
  - . direct or
  - . crossed;
- while the sensor influence on the corresponding motor can be
  - . an *excitation* effect: the more the sensor is stimulated, the more the motor is accelerated; or
  - . an *inhibiting* effect: the more the sensor is stimulated, the slower the motor is.

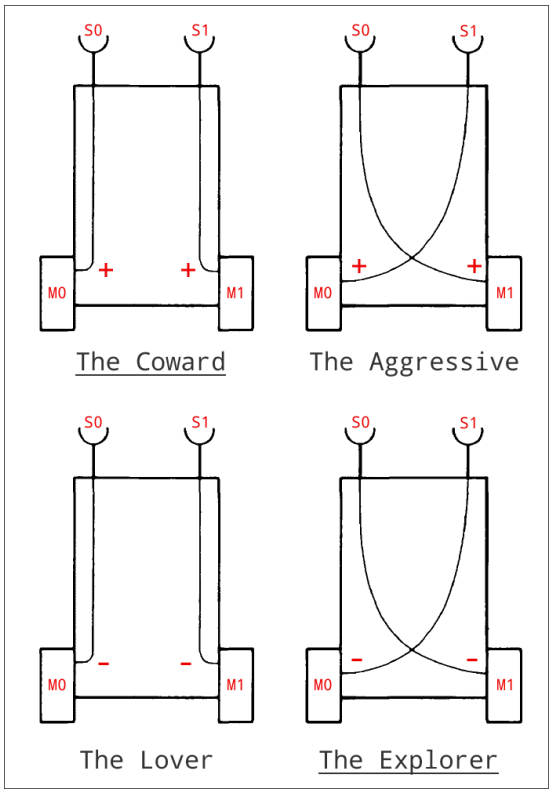


Figure 3.2: Braitenberg four basic vehicles.
(+) excitement, (-) inhibition / S0, S1: sensors / M0, M1: motors.
Modified version of Braitenberg (1984, p.7, fig2)

By combining these possibilities, we obtain the four possible vehicle types of Figure 3.2.

The first two vehicles based on excitement **(+)** – the coward and the aggressive – run into any object right in front of them. But as soon as some object is a little on their left or on their right:

- the **coward** quickly runs away from the object before slowing down when distant,
- while the **aggressive** turns towards the object before running right into it.

The **lover** is also attracted by any object it meets: it turns and then moves towards it. However, thanks to the inhibition **(-)**, it slows down more and more as it approaches it. Finally, the **explorer** goes around the objects by slowing down and accelerates in empty spaces.

In the initial experiment of Maris and Boeckhorst (1996), the sensors and the motors of the DidaBots – the robots used by the authors – are connected in the way described in Figure 3.3. The tangle of connections between the sensors and the motors on the schema can make it difficult to tell at first sight what is the robot behavior. But a second glance is enough to see that these connections are equivalent to a mix of two Braitenberg vehicles: the coward and the explorer. Therefore, the basic behavior of the robot is a simple **avoidance** one. And this is exactly what you can observe in the experiment: each robot normally avoids any large enough object such as another robot or the arena wall – "large enough" given that the front sensor is disabled; we'll come back to this point shortly.

> **Blocking Situations. First Look.** A small note about this avoidance behavior. In theory,
>
> - if two robots arrive perfectly in front of each other, or
> - if a robot approaches a wall perfectly orthogonally,

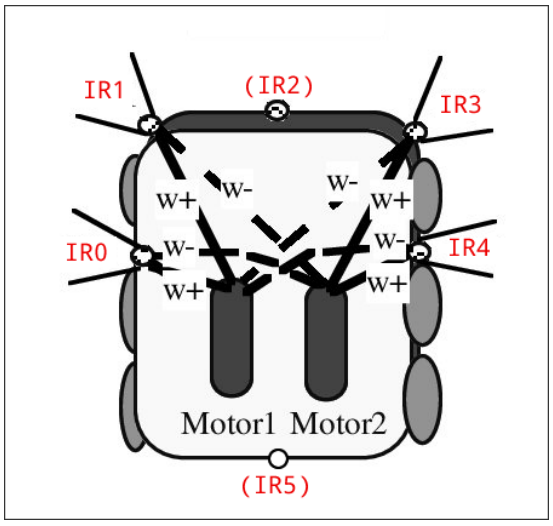


Figure 3.3: DidaBots: connections between sensors and motors.
(+) excitement, (-) inhibition. / IR0 to IR5 : infrared sensors. / (IR#) : disabled sensor.
(Slightly modified version of Figure 2 of Maris and Boeckhorst (1996, p.1656b))

the robot(s) should stop or collide and being blocked (depending on the dominance of the explorer or the coward behavior, or the balance between them). However, in reality, these blocking situations should happen quite rarely, and if they do happen, the situation should tend to unblock on its own after a reasonable amount of time (a couple of minutes at most). Indeed, first of all, these situations are simply not very likely in themselves. But, more importantly, proximity sensors always return a certain amount of noise with their measurement which induces small deviations, while wheels never rotate perfectly – two facts that further reduce the likelihood of these blockage situations to occur. Furthermore, if these situations happens anyway (i.e. if perfectly "in front" or "orthogonal" happens anyway), as soon as there are more than two robots, there is a strong chance that the passage of a third robot will unblock the situation. We'll speak of these situations again when we discuss the results obtained in the section 3.5 below.

3.2.2 Physical constraints: morphology and interactions

In order to obtain the desired result, in order for the robots to gather the polystyrene cubes together in heaps using a simple avoidance behavior, the experiment plays with physical constraints: the size and shape of the cubes, of the robots and of the arena, and the position of the sensors on the robots. More precisely, in order for the heaps to form:

1. the front sensor(s) of the robot must be disabled;
2. a cube must be *light enough* to be pushed by a robot colliding with it;
3. the size of a cube must be *small enough* not to be detected by sensors when it is exactly in front of the robot; and
4. the size of a cube must be *large enough* to be detected by a robot
   - when not exactly in front of it, or
   - when forming a group with other cubes

   thus triggering the avoidance behavior of the robot.

The main goal of these conditions is that an isolated cube, *and it alone*, is not detected when in front of a robot and is thus pushed by it.

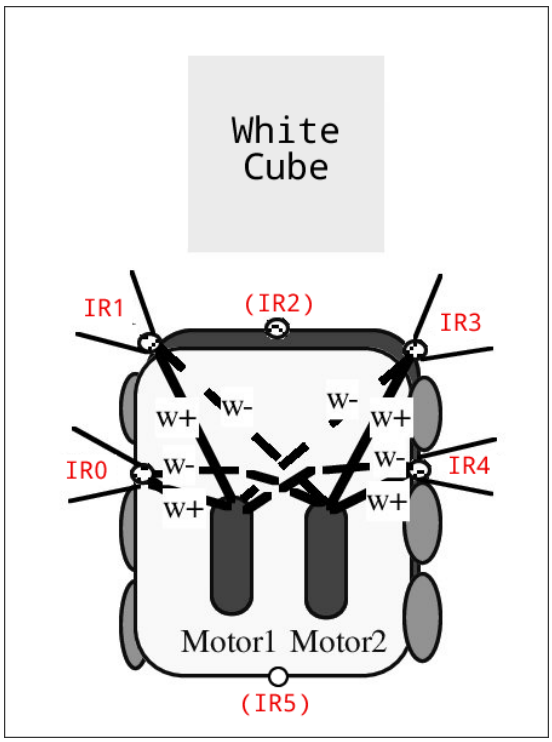


Figure 3.4: DidaBot in front of a cube – dimensions.
(+) excitement, (-) inhibition. / IR0 to IR5 : infrared sensors. / (IR#) : disabled sensor.
(Slightly modified version of Figure 2 of Maris and Boeckhorst (1996, p.1656b))

Maris and Boeckhorst (1996, p1656b) explains very well how heaps appear quite naturally if these conditions are met:

> For the experiments, the weights of the frontal IR-sensor are set to zero. This has the effect that the Didabot cannot "see" anything that is exactly in front of it and too small to be detected by any of the other IR-sensors. It implies that the Didabot collides with such an object and pushes it until the robot detects another obstacle (a wall, another object or another DidaBot). Then an avoidance movement occurs and the shifted object is left behind. If the shifted object is deposited close enough to another, they form a constellation that is easily detected and avoided by the robots. In this manner, clusters are created. Objects that are pushed against the wall will not be removed anymore and are "lost" for cluster building.

3.3 Experiment stakes and pedagogical interest

One of the main interests of this experiment is the unexpected and very simple way in which it performs a seemingly complex task involving patterns generation. To understand this, it is very interesting to ask an audience attending the experiment to guess how it is carried out and how the robots are programmed, or alternatively, how they would proceed to carry out such a task. Without additional information, somebody not already acquainted with the experiment or others of its kind will have a strong tendency to interpret the behavior of the robots roughly in the following way:

- the robots have been programmed to recognize the white cubes when they are in front of them and to push them to form heaps;
- the formation of the heaps seems to show some coordination between the robots as well as a global knowledge of their environment: perhaps there is some communication between them or a central control unit that coordinates them.

This way of thinking about robot behavior is centered on the robot and its "cognitive" capabilities: it is as if you put yourself at the place of the robot that has to perform the task and wonder what predefined rules it has to follow in order to do it. Following this approach, robots have to perform complex operations in order to accomplish the described task: recognition of a cube, detecting and/or storing the position of the other cubes, detecting the presence of other robots, coordination and so on (cf. Maris and Boeckhorst, 1996, p.1655).

As seen, the approach actually used in the experiment is quite different. Instead of focusing on the robot and its "point of view", it considers the *ensemble* formed by the robot, its local environment and the possible *interactions* between them: "The environmental structure, changed by the activities of the entities, affects the behavior of these agents which in turn influences the environment" (Maris and Boeckhorst, 1996, p.1655).T he predefined rules governing the experiment then concern not the robot alone, but the possible *interactions* between it and its local environment. Such an approach is inspired by what happens in the animal kingdom (especially insects), where simple animals are able to carry out together seemingly very complex achievements (hives, foraging, ...) (cf. Maris and Boeckhorst, 1996, p.1655).

3.4 Adaptation to e-pucks

At the heart of the experiment described by Maris and Boeckhorst (1996) is the *interactions* between the robot and its environment. As seen, the key elements governing these interactions are:

- the robot avoidance behavior;
- the robot sensors position and setting (front sensors disabled);
- the size and the weight of the cubes: each cube having to be
  - *small enough* not to be detected when it is alone in front of a robot, but
  - *large enough* to be detected as soon as it is not just in front of it or when it forms a group with other cubes; and
  - *light enough* to be easily pushed by a robot colliding with it; finally
- the size of the arena: its walls play a key role in the movement of the robots that avoid them.

3.4.1 Sensors and avoidance behavior

At sensors level, the e-puck has everything needed to get an avoidance behavior similar to the DidaBot, the robot used by Maris and Boeckhorst. To achieve it, we programmed the e-puck with an avoidance behavior while disabling its front sensors. Let's note that we used an *explorer* behavior only as it seems enough to get the desired result, and not a combination of an *explorer* and a *coward* like the initial experiment. (In another work it could be interesting to further explore the difference induced by the addition of the coward component, if any.)

However, in their standard version, the e-pucks struggle to detect each other using their IR sensors. This is due to their transparent body, which reflects IR light very poorly. This is problematic for our experiment where mutual avoidance of robots plays a key role. Indeed,

- on the one hand, if we adjust the sensors to detect walls and cubes in an optimal way, then the robots do not detect each other sufficiently and collisions between them are numerous. But
- on the other hand, if we adjust the sensitivity of the sensors to avoid robots collisions, then the robots become too sensitive in their avoidance behavior, and always stay far away from walls and cubes – which is very detrimental to the experiment course.

One way to solve this problem is to try to find a compromise in the settings of the IR sensors: but the result is not very satisfactory: collisions still often occur whereas the robots still pass far away from walls and cubes. The best solution is to modify the e-pucks in a general way in order to improve the reflection of their bodies and their mutual detection. We did this by sticking a 3 [mm] wide reflective tape on the lower part of their bumper. Reflection measurements of the e-pucks with and without reflective tape, and details about this modification are discussed in the appendix C.1 [Hardware] E-pucks mutual detection problem (with IR proximity sensors).

3.4.2 Dimensions

The e-puck is much smaller than the DidaBot and of a different shape. Therefore, the cubes and the arena must be adapted accordingly to perform the experiment in a manner close to the initial one.

- Since the e-puck's width is roughly half the DidaBot's, we used cubes with half the edge of the original experiment's cubes. This dimension fulfils its role very well once the front sensors IR7 and IR0 are disabled: a cube facing the e-puck alone is not detected by the semi-lateral sensors IR6 and IR1 placed at $\pm$ 45°, whereas it becomes detected by them as soon as it is slightly on the left or the right, or if it forms a cluster with other cubes. See Figure 3.5 just below for the position of the different sensors.

- In the original experiment, the arena's width corresponds to 10x the DidaBot's length, while the arena's length corresponds to 20x the robot's width. We used these ratios for our arena.

Concerning the weight of the cubes, we simply used some polystyrene: this makes the cubes light enough to be easily moved by an e-puck colliding with them.

Figure 3.5 gathers all the adaptations made to carry out the experiment of Maris and Boeckhorst (1996) with e-pucks. As a reminder, the sensors in brackets are the deactivated or unused sensors.

3.5 Results

If you adapt the dimensions of the cubes and the arena as described in Figure 3.5 and add reflective tape to the e-pucks to improve their mutual detection, you get an experiment that replicates very well the one of Maris and Boeckhorst (1996). Figure 3.6 below shows it well through photos of the arena at the beginning and end of the experiment for 5 runs (with 5 e-pucks and 25 cubes); while the demo mentioned at the beginning of the chapter gives a very good idea of the typical course of the experiment adaptation: https://youtu.be/AevML_FSlEM.

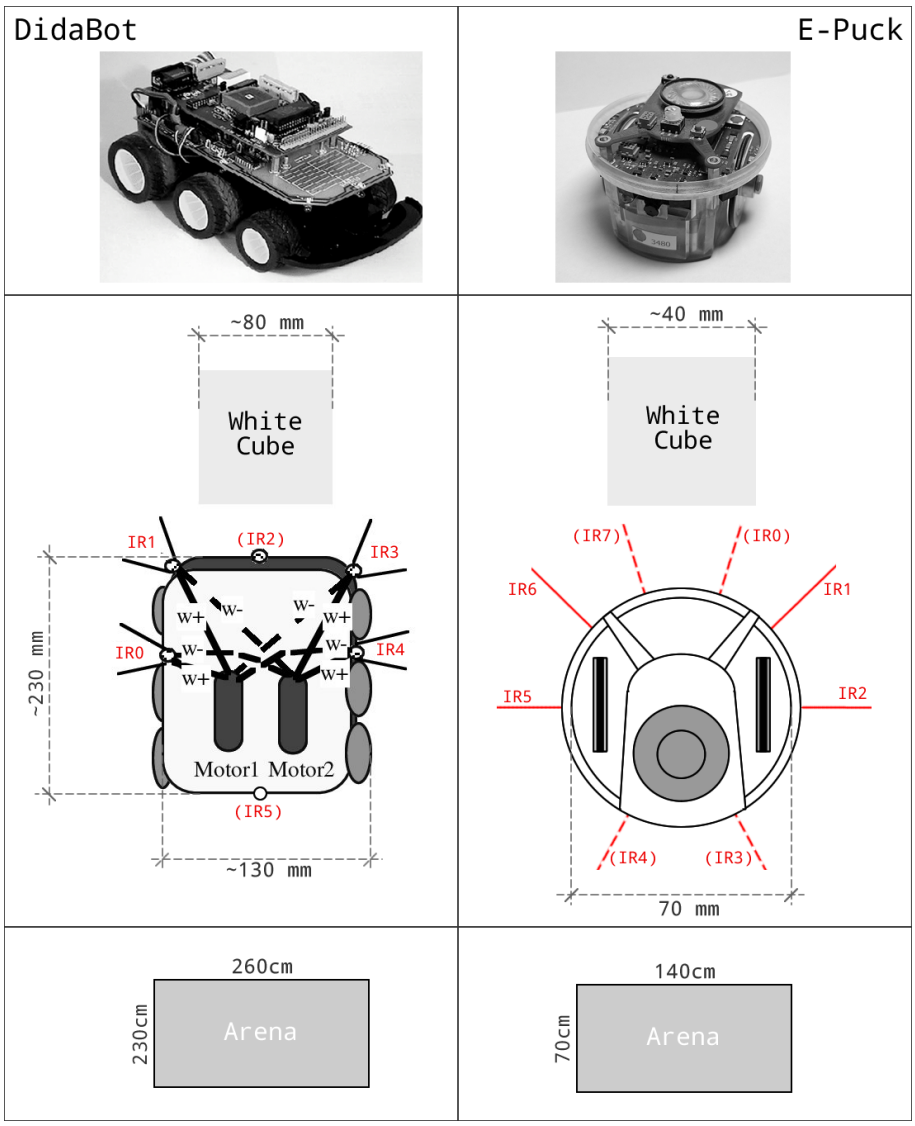


Figure 3.5: DidaBot vs e-puck: robots, cubes and arena dimensions.
Be aware that the drawings are not to scale.
(IR#) are unused or disabled sensors.

The photo of the DiBabot and the base of its schema come from Maris and Boeckhorst (1996)

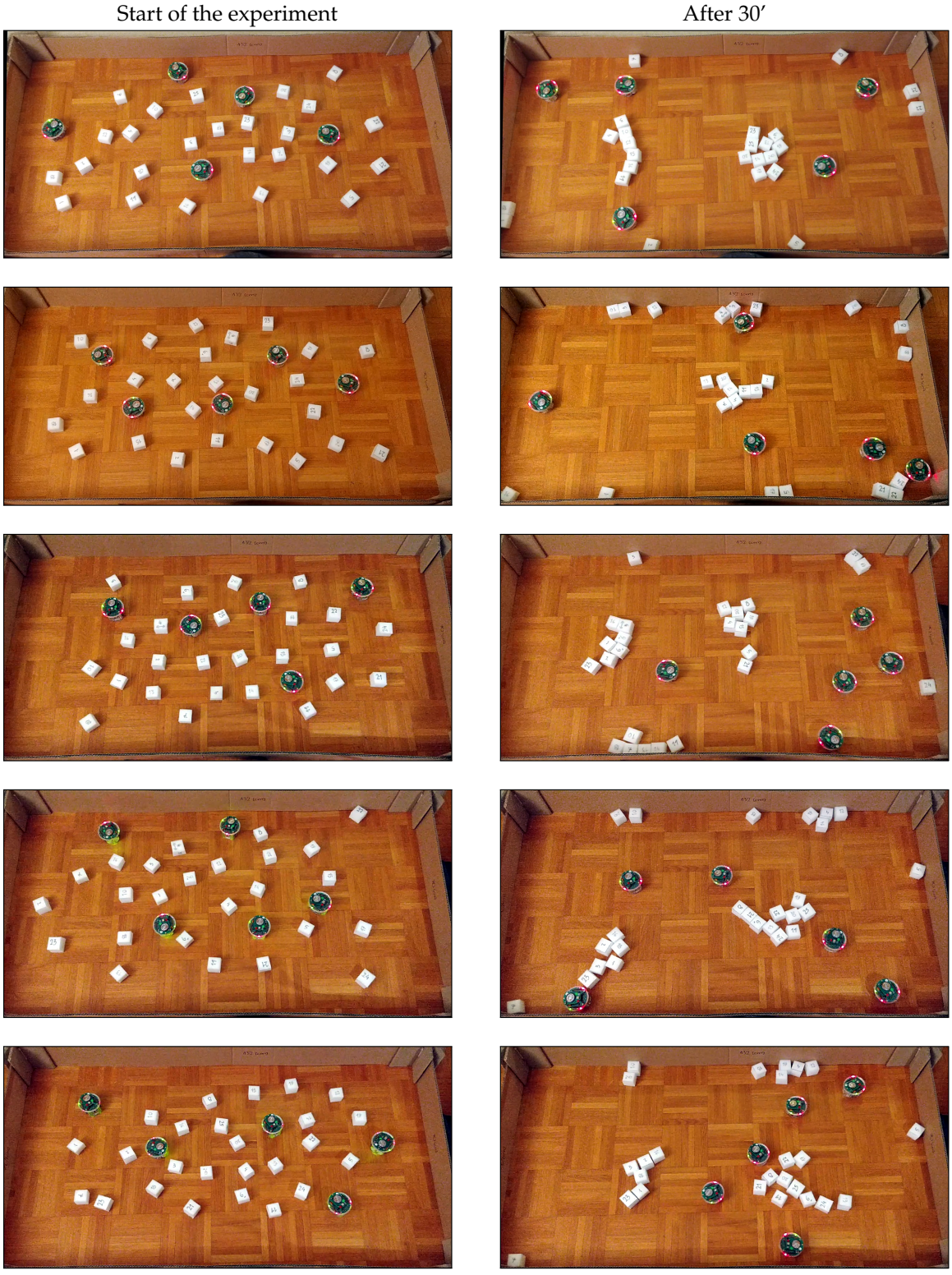


Figure 3.6: Collective heap building with e-pucks:
start and end of 5 experiment runs (with 5 robots and 25 cubes)

3.5.1 General trends

In their paper, Maris and Boeckhorst study systematically the *dynamics* of heaps formation: they explore and analyze what happens when you vary the number of cubes (12 or 25) and the number of robots (from 1 to more than 5), and how the number and size of emerging clusters varies accordingly. Some key points of their results are quite observable, with some modifications, in our adaptation. For this reason, we mention them here.

In the initial experiment of Maris and Boeckhorst (1996)

- The average number of clusters after 30' is:
  - 2.3 with 12 cubes (with an average size of 4 cubes) and
  - 3.4 with 25 cubes (with an average size of 8 cubes)
    (cf. Maris and Boeckhorst, 1996, p.1657b).
- The rate of the number of cubes pushed increases when the number of robots is increased up to 5. However beyond 5, the interaction between robots is such that they get in the way of each other and start to encounter fewer cubes (cf. Maris and Boeckhorst, 1996, p.1658b).
- The lowest number of final clusters is reached:
  - with 2 robots for 12 cubes,
  - with 3 robots for 25 cubes.
    (cf. Maris and Boeckhorst, 1996, p.1658b)

Unlike Maris and Boeckhorst (1996), given the pedagogical orientation of this work (making demonstrations for schools and university visitors), we renounced to conduct a systematic study of the dynamics of heap formation with our adaptation. Nevertheless, repeated observations of its course allowed us to identify several major trends, trends that should of course be confirmed in other works. The 5 runs represented on Figure 3.6 are quite representative of them.

General trends in our adaptation (with 5 robots and 25 cubes)

- Most of the time, the number of clusters is between 1 and 3, with a strong tendency towards 2.
- The number of cubes per cluster seems to vary roughly between 5 and 10.
- It is during the first 20' that most of the movements occur at the level of the cubes. In the last 10', less usually happens, although, if the arena contains two small clusters (of 3-4 cubes) close together, it is in these last 10' that they will merge into a larger one.
- We've got no real observation on the relation between robots' and clusters' number. We didn't do enough testing while varying the robots' number to say anything substantial about this point. We can nonetheless tell that, contrary to the initial experiment, 3 robots do not seem a good choice, because of the different blocking situations that can happen (more on this in 3.5.2). Indeed, a reduced number of robots tends to allow these situations to set up longer, which is counterproductive for a demonstration.
- Among the different blocking situations (will be discussed in 3.5.2), the one in which a robot is blocked against a cube that is itself stuck to a wall frequently occurs when the experiment is already well advanced, i.e. when several clusters are already present and especially when several cubes have already been " stored " against the walls.

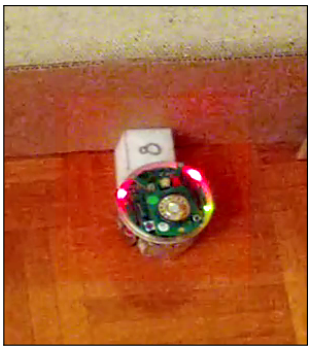


Figure 3.7: A robot stuck by a cube against a wall because it doesn't "see" the cube.

3.5.2 Blocking situations

Maris and Boeckhorst don't mention it in their articles, but it happens quite often that the robots get stuck during the experiment – at any rate in our adaptation. We already discussed the theoretical possibility of such blockages when we were speaking of the avoidance behavior (see the note of 3.2.1 above). At that time, we mentioned two situations:

1. A robot arrives perfectly in front of another robot. Given the sensors' settings (front sensor(s) disabled), they should both then try to keep moving forward while being blocked by each other. As a result, their wheels turn on themselves while the robots stays at the same place.

2. A robot arrives perfectly perpendicular to the wall, which should cause a stop or a collision.

We then said that these situations seem very unlikely to happen – especially because of the measurement noise inherent to the infrared sensors and the inaccuracy of the wheels movements. All this is confirmed in our adaptation. The first situation, where two robots arrive in front of each other and are blocked (Figure 3.8), happens very rarely : like 2-3x in 30' at most. And when this happens, it doesn't last more than 1 to 2' because of the aforementioned reasons (measurement noise and wheels inaccuracy) or thanks to the passage of another robot close to an active sensor (see for instance https://youtu.be/AevML_FSlEM?t=3 at the bottom left of the arena). As for the second situation, the one where a robot would arrive perfectly orthogonal to the wall and stop, we simply never actually observed it.

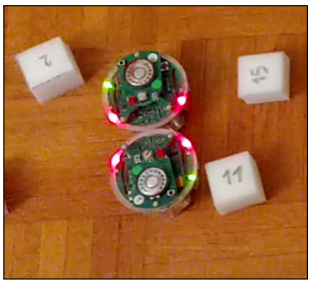


Figure 3.8: Two robots in front of each other get stuck, because they don't "see" each other.

Another situation we hadn't thought about then is happening on a regular basis, even if not very frequently (something like 0-2x/30'): a robot finds itself in a situation where two cubes are exactly at the level of its semi-lateral IR sensors 7 and 1, which makes its wheels stop: Figure 3.9. But here too, the situation is quickly unblocked by the actions of or the interactions with the other robots (see for instance https://youtu.be/AevML_FSlEM?t=54 at the bottom left of the arena).

Figure 3.9: Two cubes exactly in front of the semi-lateral IR sensors 7 and 1 makes the robot stop.

A situation that we hadn't thought about before observations appears much more frequently (about 10 times for 30') and is more annoying: a robot gets stuck because it is separated from a wall of the arena by a cube placed exactly in the zone where it cannot detect it (see for example https://youtu.be/AevML_FSlEM?t=39 in the upper right corner of the arena). This is the situation we mentioned in the last point of the general trends of the last subsection, situation we already depicted by the following figure:

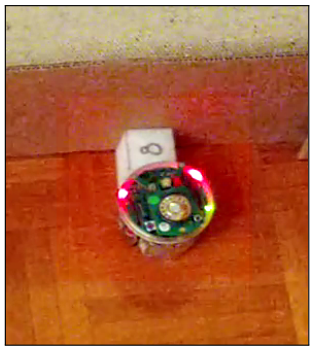


Figure 3.7: A robot stuck by a cube against a wall because it doesn't "see" the cube.

In this situation, as the robot does not detect anything, there is no inhibition: its wheels turn on themselves while it doesn't move forward at all.

At the beginning of the experiment this blocking situation rarely happens, but it regularly happens once the first clusters are formed and the first cubes have been pushed against the wall. When this blocking situation happens, it is worth waiting 1 to 2'. Indeed, most of the time, the situation unblocks itself: the passage of another robot, or more surprisingly, nothing special (probably some noise in sensors measurement), and the robot unblocks itself. Moreover, if the arena floor is smooth enough, wheels rotation on themselves while the robot doesn't move shouldn't damage the robot's motor or the rubber of its wheels too much. If the situation nevertheless persists, a small manual intervention (passing the hand in front of one active sensors) is enough to unblock it.

We tried to handle this problem elegantly by detecting this kind of situation through the robot's accelerometer, but unfortunately without success. You might be tempted to try to deal with this problem by using wider cubes, still narrow enough not to be detected when exactly in front of the robots. However, if you try, these ones are likely to be too easily detected by the robots, which would be detrimental to the experiment course. Finally, it should be noted that, since in these situations the robot is pushing the arena wall via the cube, it is important to fix its walls well if you want the experiment to run successfully.

4 Decentralized synchronization between robots through beep sounds

Contents

Sensors: micros (IR proximity sensors, IR Receiver).
Actuators: speaker, leds, motorized wheels.

In this chapter, we show how, by applying the model of Mirollo and Strogatz (1990)[1], e-pucks can synchronise with each other without any centralized device. At the beginning of the experiment, each robot emits a regular beep sound, but out of sync. Over time, thanks to small inner adjustments following the model, they all emit a synchronous beep together.

- Files location: https://github.com/jrlauper/jrl_epuck/tree/master/jr_demo
- Main files: `run_chirping.c/h`
- Function: `void run_chirping()`

- Robots selector: 5.
- Demo video : `https://youtu.be/I6GSagp46zs`
- Number of robot: 2 to no limit.

[1] "Synchronization of Pulse-Coupled Biological Oscillators".

4.1 Experiment and stakes

Let's start by describing the experiment from outside, without "worrying" about what's going on inside the robots. Here is what happens when we start a *single* robot:

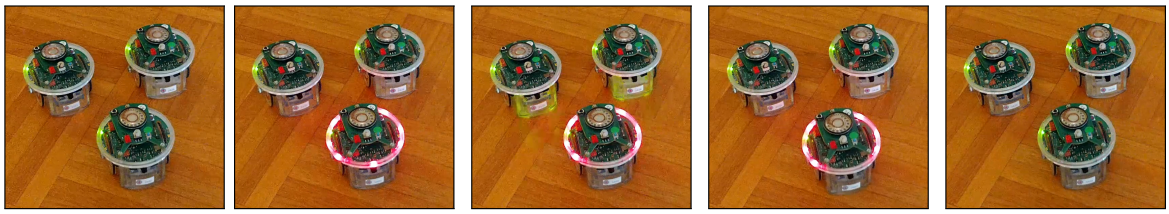
> **[1 single e-puck]** After a very short while, the robot starts beeping at a regular pace. Visually, each emitted beep is accompanied by the illumination of the whole red leds circle from its bumper, while the robot performs a small back and forth movement.

And, now here is what happens when we activates *two* robots at the same time:

> **[2 e-pucks]** Each robots start beeping regularly, but not exactly at the same time, out of sync. When one robot beeps (sound, red leds circle and slight movement), the body of the other robot lights up green. This means that it has recognized, that it has "perceived" the sound emitted by the first one. Over time, as the number of beeps emitted and recognized by each robot increases, the time lag separating the beeps of each robot decreases. After some time, they both beep synchronously at the same pace.

If we increase the number of robots, the same situation occurs in an extended way. Initially, all the robots beep regularly, but out of sync. Each time one of them emits a beep (sound, red leds circle, small movement), the others that have not emitted a beep at the same time "perceive" it, and their body lights up green. With each beep emitted and perceived, the lags between the different robots diminishes. After a certain time, all the robots beep synchronously at the same pace and keep on doing so.

The next figure shows three robots during one beep emission and recognition:



(a) initial "idle" state

(b) the 3rd robot beeps (red leds circle and small movement)

(c) the two other robots "perceive" the beep (body in green)

(d) the 3rd robot completes its small movement

(e) back to idle state

Figure 4.1: A beep is emitted by a robot and recognized by the other two.

This experiment reproduces in a simplified way a mechanism found at the heart of the living world: fireflies flickering in rhythm, pacemaker heart cells firing in sync, or crickets chirping in unison. In each of these cases, several living organisms synchronize through punctual communication and mutual adjustments without the presence of any central control. (cf. Mirollo and Strogatz, 1990, p.1645).[2]

Beyond its role in the living world, such a mechanism reveals a strong utility in the computer world. Indeed, the synchronization of multiple computers within a distributed system is an important IT issue and a real challenge. The principle underlying the current experiment

[2] For the reader interested in this animal side, Mirollo and Strogatz (1990, p,1645) gives a few more examples before mentioning one paper and two books from A.T. WINFREE containing many more examples: *Biological rhythms and the behavior of populations of coupled oscillators* (1967), *The Geometry of Biological Time* (1980), and *The Timing of Biological Clocks* (1987).

shows a way to achieve this in a decentralized way. This makes it possible to achieve a robust synchronization system that can be maintained despite local failures. In the specific context of e-pucks, this experiment allows us to have ready at our disposal a way to synchronize e-pucks for more complex experiments requiring it.

Now remains the question of the "how?": how to achieve such synchronization between entities that communicate only punctually? what principles, which "inner adjustments" can assure us that the different entities or devices will synchronize and continue to be so? This is the question Mirollo and Strogatz (1990) model answers.

4.2 The model: coupled-oscillators (Mirollo and Strogatz, 1990)

At the core of the model proposed by Mirollo and Strogatz (1990) are small oscillators. In our experiment, they are the e-pucks, and in the other examples mentioned, they are the fireflies, the pacemaker heart cells or the crickets. Each oscillator possesses an *internal state* that can be thought of as *a charge level* ranging from 0 (completely empty) to 1 (completely full). When the charge of the oscillator reaches or exceeds 1, it fires – this is e.g. the e-puck beep e-puck –, before dropping back to 0. Taken individually, each oscillator possesses a temporal cycle (described by a phase) during which it gradually charges up to 1 before firing and going back to 0.

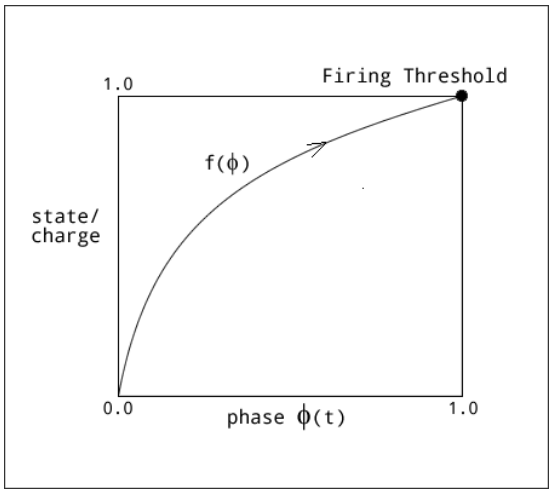


Figure 4.2: Timecourse of a single oscillator during one cycle
(modified version of Mirollo and Strogatz (1990, fig.1))

When several oscillators are put together, they interact in the following way: if an oscillator "perceives" the firing of another oscillator (in our experiment, when a robot "hears" the beep of another robot), then its internal charge is instantaneously increased by a certain amount ε (see Figure 4.3). As a consequence, when an oscillator "perceives" the firing of another, it is as if it temporally jumps forward within its cycle as a reaction; in other words, the perception of the firing of another oscillator generates a phase shift.

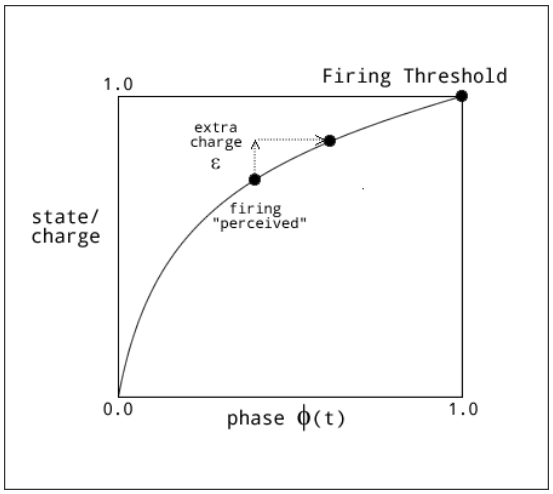


Figure 4.3: The charge increase ε and the corresponding phase shift of an oscillator caused by the perception of the firing of another oscillator (modified version of Mirollo and Strogatz (1990, fig.2))

The model assumed that the charge increase is each time the same for every robot. Furthermore, if the charge increase causes one oscillator's total charge to reach or exceed 1, then it in turn fires before its charge falls back to 0.

Mirollo and Strogatz (1990) proves that a group of oscillators will always synchronize if the following assumptions are met:

(1) each oscillator has *the same charge dynamics* and *the same basic cycle*;

(in other words, the Figures 4.2 and 4.3 are the same for every oscillator;
or, still in other words, the values that t can take, the phase $\phi(t)$, the charge function $f(\phi)$ and ε are the same for every oscillator.)

(2) each oscillator is *coupled to all the others* through firing perception;

(3) the function f describing the charging curve is *monotonic* and *concave down*.

A few additional notes about these assumptions.

About (1):
If the oscillators do not have the same cycle, they will still be synchronized after a certain time, the fastest setting the pace (cf. Mirollo and Strogatz, 1990, p.1660).

About (2):
If each oscillator does not hear *all* the other oscillators but only some of them, the synchronization will proceed differently. However, the authors suspect that, in the end, all the oscillators will be synchronized anyway (cf. Mirollo and Strogatz, 1990, p.1660). This would be an interesting hypothesis to explore and to test in other experiments with our e-pucks.

About (3):
If the function f is *linear*, the oscillators remain perpetually out of sync (cf. Mirollo and Strogatz, 1990, p.1658). Synchronization also fails if the function f is *concave up* (cf. Mirollo and Strogatz, 1990, p.1659).

4.3 Application and adaptation to e-pucks

The proof of the last section is the key part of Mirollo and Strogatz model to carry out our experiment with e-pucks. Indeed, thanks to it, if assumptions (1) to (3) are met, we are guaranteed to obtain the desired result, i.e. that initially out of sync e-pucks end up beeping in chorus.

4.3.1 Same cycle, same charge dynamics, but random start

Meeting assumption (1) – same basic cycle and same charge dynamics for every oscillator – is not a hard task to achieve. Indeed, it is only necessary to program the different e-pucks *with one and a same code* to ensure that this hypothesis is fulfilled. However, there is a "hidden" condition to be able to carry out our experiment in a satisfying way, a prerequisite so obvious that it can easily be forgotten. It is simply that the robots are *out of sync* at the beginning. This can be achieved in two ways:

- If the robots are switched on one after the other *manually*, one can expect that they will usually be out of sync without the need for anything else.
- If the robots are started *all at the same time*, which we decided to do with the help of a remote control, a random moment must be picked out in each robot's cycle for the start of its first cycle.[3]

To choose a random moment in the cycle of each robot, we used a function able to return a pseudo-random integer. This function, which takes advantage of the inherent noise present in the IR sensors of the e-puck, is the following one:

- Files location : https://github.com/jrlauper/jrl_epuck/tree/master/jr_demo
- Files: `utility_mod3.c/h`
- Function: `int jr_random(int max_number)`
  (with `max_number` = cycle length)

The details concerning this function and its working are given in Appendix Section C.2.2 [Software Module] Pseudo-random number using IR proximity sensors only.

4.3.2 Oscillators' coupling and firing through beep sounds

To satisfy assumption (2) – coupling between all oscillators –, we needed a firing behavior that could be perceived by all the oscillators. Our goal being demonstrations for schools, we decided to do it with a loud enough beep sound: something that is easily perceptible by an audience, in contrast to, for example, communication via IR light or radio waves. As simple as it may seem, managing to get an e-puck emitting a beep is unfortunately an adventure in itself. All the details about this challenge and how we met can be found in B.1 "How to produce sound with e-pucks". In order to emit a beep, we used one of the functions of the library we created to produce sound on the fly with e-pucks:

- Files location : https://github.com/jrlauper/jrl_epuck/tree/master/jr_demo
- Files: `e_freq_sound.c/h`
- Function: `void freq_beep1_3072(double std_percent)`

[3] More about the usage of an IR remote control can be found here : C.3 [Software module] Infrared remote-control module for selection.

More details specifically about the `e_freq_sound.c/h` library can be found in the part devoted to it: B.1.4.

Beep sound emission is just the first part of coupling, the second part being its recognition . To recognize the specific beep emitted by the robots, we used the following function:

- Files location : https://github.com/jrlauper/jrl_epuck/tree/master/jr_demo
- Files: `e_freq_recognition.c/h`
- Function: `int fft_get_frequency(void)`

This function is a generalization of the `runfftlistener.c/h` program (program 9 of the full demo 2 (EPFL, 2007), see Appendix D). More details about this specific function can be found in B.2.6 while more general information about sound frequencies recognition with e-pucks can be found here: B.2 How to recognize sounds with e-pucks.

The function just mentioned works pretty well to recognize some specific frequencies emitted for instance by a piano or while whistling. However, recognising a sound emitted by another e-puck is a completely different story and raises unexpected difficulties. In very short, to the ear, the e-puck speaker seems to emit the frequency we're asking it to emit by using e.g. functions from `e_freq_sound.c/h` library. However, in reality, the sound emitted by the speaker includes many harmonics (multiples of the target frequency) of a volume as high as the desired frequency. As a consequence, very often, the frequency recognized by another e-puck is not the one expected, not the one meant to be emitted at first, but one of these extra frequencies. Thus, to combine the emission and the recognition of a specific sound by e-pucks, you need to determine empirically a *group* of emitted frequencies and can't rely at all on the frequency supposed to be emitted. Typically for the beep at 3072 [Hz] used in this experiment, you have to compare the frequency captured and returned to 12 possible frequency values instead of 1! More about this specific difficulty can be found here: B.3 Sound frequency communication between e-pucks.

Once you are able to make an e-puck emit and recognize a beep, remains a difficulty specific to the adaptation of the model of Mirollo and Strogatz to e-pucks. Indeed, one of the key elements of the model is the increase in charge ε that occurs at each beep perceived. For the model's implementation to work well, each beep must be perceived, and only once. This requires finding an adequate correspondence between the duration of the emitted beep and the duration during which each robot checks if it perceives a beep (this last duration being the TIC variable in our code). To simply explain the respective impact of each of these durations:

- About TIC: the time during which the robots check if they perceives a beep:
  - if TIC is too short, there is a risk that the robots recognizes 2 beeps, when in fact they hear only 1;
  - if TIC is too long, there is a risk of recognizing 1 beep while the other robots have emitted more beeps; besides, there is an extra risk of making the robots too slow in their reactions.
- About the emitted beep:
  - if it is too short, it risks not being recognized;
  - if it is too long, it risk being recognized several times instead of once.

After many tests, we got a good beep recognition result for a TIC of 1 and a beep length of 0.4 (`freq_beep1_3072(0.4)`).

4.3.3 Charging function $f(\phi)$

At the heart of the Mirollo and Strogatz (1990) model is the charging function f. As shown in Figure 4.2, this function describes how the internal charge of the oscillator varies according to its phase. To meet assumption (3), this function can be any function as long as it is *monotonous* and *concave down*. To carry out our demonstration, we used the very function class that Mirollo and Strogatz use in their paper for convenience – this function class being particularly suitable to capture what happens in their model. With $b > 0$, this function class f and its inverse g are the following ones:

$$\begin{aligned} f: \quad phase \quad &\rightarrow \quad state/charge \\ \phi \quad &\mapsto \quad f(\phi) = \frac{1}{b} ln(1 + [e^b - 1]\phi). \end{aligned}$$

$$\begin{aligned} g: \quad state/charge \quad &\rightarrow \quad phase \\ u \quad &\mapsto \quad g(u) = \frac{e^{bu} - 1}{e^b - 1}. \end{aligned}$$

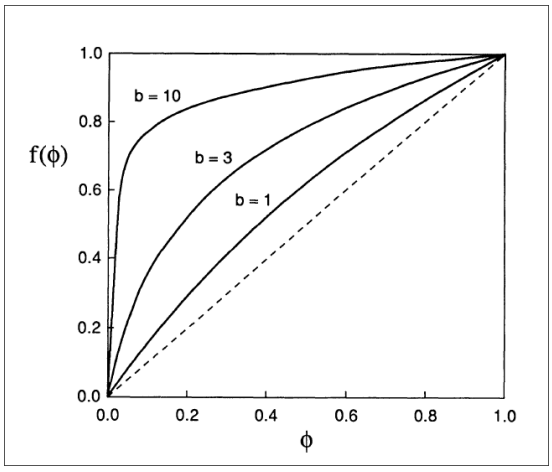


Figure 4.4: Graphs of the charging function f chosen for different values of the parameter b
(Modified version of Mirollo and Strogatz (1990, fig.3))

The parameter b is used to adjust the speed at which the synchronization emerges: the higher b is set, the faster the synchronization will take place. As a reminder, as indicated above about (3), if $b \leq 0$, the synchronization fails.

In practice, within our adaptation of the model to the e-pucks, you can modify the course of the experiment by playing with:

- the value of b,
- the length of the cycle, and
- the increase in charge ε that occurs when a beep is "perceived".

4.3.4 Pseudo-code

In the previous parts of this section, we've introduced the various pieces needed to adapt Mirollo and Strogatz's model to e-pucks:

- a random start for each robot (4.3.1);
- beep's emission for firing (accompanied by red leds lighting and a small back and forth movement) (4.3.2); and
- beep's recognition for firing perception (accompanied by green lighting of e-puck's body) (4.3.2).

We've also defined which function class f and inverse function class g we employed in our implementation (4.3.3). With these elements at hand, we used the following algorithm to apply and adapt Mirollo and Strogatz's model to e-pucks:

Algorithm 1: Decentralized synchronization using beep sounds

```
1  t: time;
2  T: cycle_length;
3  phi(t) : phase = t/T;
4  epsilon: the extra charge increase;
5  f(x): a monotone concave down function for x in [0, 1]  with  f(0) = 0  and  f(1) = 1;
6  g(x): the inverse function of f(x);
7  beep_check: returns true if some beep has been heard, false otherwise;

8  t := random(T);
9  while true do
10     t++;
11     if beep_check then
12         t := g{f[phi(t)] + epsilon} · T;
13     end
14     if t ≥ T then
15         beep_emit;
16         t := 0;
17     end
18 end
```

Figure 4.5: Decentralized synchronization using sounds (pseudo-code)

4.4 Results

In our implementation of the model, three parameters can be used to vary the course of the experiment and the time needed for all robots to be synchronized :

- the cycle length T:
- the parameter b, which determines the curvature of the load function f (see 4.2);
- and the load increase ε that occurs at each perception of a beep.

Our goal being a demonstration, it was important that the spectator could well observe the transition from a rather strong initial desynchronization to a final synchronization. At the beginning of the experiment, a random moment is picked out during the cycle of each robot A moment – "TIC" in the code – is a multiple of the time needed by a robot to check if it has perceived a beep. Thus, to obtain a good initial desynchronization, it is important that the number of moments composing the cycle is large enough. This number must be even bigger if you want to use a large number of robots and want them to have good chance to start out of sync. We performed the experiment with 3, 9 and 15 robots. Empirically, a cycle length of 300 ($T = 300$ which correspond roughly to 6 [s] in reality), gives us good results: most of the time, the robots start out of sync in a way easily perceptible by the audience.

Concerning the ε charge increase, rather than giving it a fixed value, we decided to make it depend on the length of the cycle in the following way: $\varepsilon = 1/T$. Here again, our motivation was the context of a demonstration. A sufficiently small ε allows progressive phase shifts without constantly reaching the end of the cycle. This allows a progressive synchronization that is well observable by the audience and that corresponds well to the type of biological phenomena represented by the model.

The value of b allows us to modify the curvature of the load function f we have chosen: for a same ε, a high b favors a faster synchronization. But a too high b, like a too high ε, has the negative effect that the oscillator constantly reaches the end of the cycle at the slightest perceived beep or couple of beeps. We have tried several values for b. The expected trend is clearly visible: higher b, faster synchronization (see Figure 4.6). Taking $b = 8.0$ allows us to have a demonstration where you have time to observe the synchronization without it being "interminable" to observe. With this value of b the synchronization takes roughly between 30 and 60 [s]' for 3 robots.

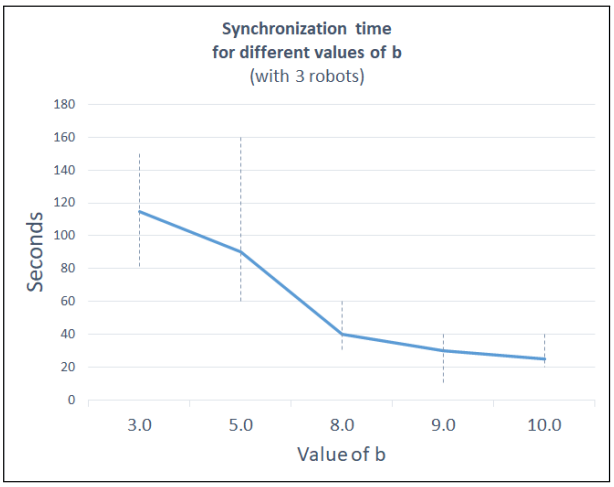


Figure 4.6: Time needed for synchronization according to the value of b.
5 measurements for each b.
(average, with min, max values represented)

In order for the values chosen for the three parameters T, ε and b to be adequate for a demonstration, the synchronization phenomenon should remain observable regardless of the number of robots: the synchronization should remain neither too fast nor too "interminable". After testing, the chosen values for the parameters ($T = 300$, $\varepsilon = 1/T$ and $b = 8.0$) remain adequate to this end for 9 and 15 robots as shown in the following figure:

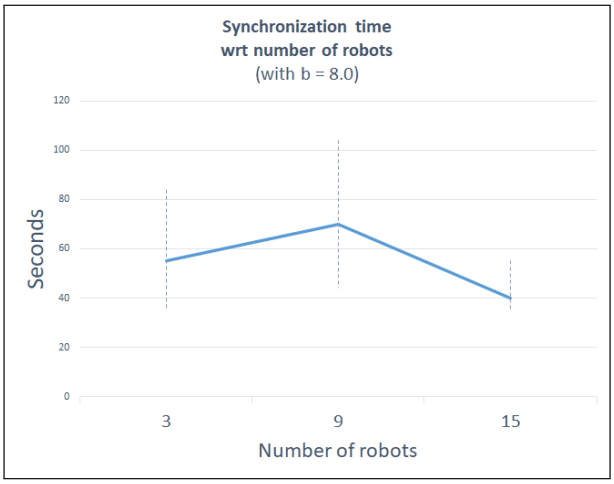


Figure 4.7: Time needed for synchronization depending on the number of robots.
b is fixed at 8.0. 5 measurements for each robots' number.
(average, with min, max values represented)

However, with its slight increase in duration for 9 robots, the trend observed in Figure 4.7 does not correspond to what we expected. Indeed, by observing many times the demonstration with different numbers of robots, the perceived trend seemed to be the following one: more robots allow a faster synchronization, especially during the first cycles. Our underlying hypothesis was: the higher the number of robots, the more likely each robot is to hear a high number of different beeps during its cycle (especially when the robots are still well desynchronized) and, thus, more chance each robot has to adjust its phase. However, the number of measurements performed to achieve this figure is very low with 5 measurements for each robots' number. Thus, the figure gives just a rather rough trend and is perhaps misleading in this respect. It would be interesting to see if this trend with the slight increase for 9 robots is verified by making more measurements, and if so, if some plausible explanation is conceivable.

We would like to add that we looked for good parameter values for a demonstration, i.e. values that allow good observation for the audience. But as said before, this model and its implementation could just as well be used as an auxiliary tool for a more complex project requiring prior synchronization from the e-pucks. In such a case, parameters that favor rapid synchronization will of course be favoured.

5 Decentralized swarm coherence using beep sounds

Contents

Sensors: micros, IR proximity sensors (IR Receiver).
Actuators: speaker, leds, motorized wheels.

In this chapter, we describe how we adapted a swarm coherence algorithm to e-pucks. In the resulting experiment, a group of e-pucks explores an arena while staying together. This is achieved by exploiting beep sounds emitted and captured by the robots. Indeed, each robot beeps regularly, and thanks to the volume of the perceived beeps, each robot can determine the density of the surrounding robots. If this density becomes too low, the robot turns around, trying to go back to a position where it was higher. The algorithm we adapted to e-pucks is presented in:

- Nembrini (2005, p.84 ff.), where it is called "alpha-algorithm", and in
- Nembrini et al. (2002, section 3.2 ff.), where it is called "basic algorithm".

In these two sources, this coherence algorithm is just a *first* step in the development of a more powerful one. This "first-step" is itself an improved version of the idea presented in Støy (2001): it adds to it the use of a threshold in order to avoid over-reacting swarms clumping together and exploring little space only.[1]

[1] Note that the goal of Nembrini (2005) and Nembrini et al. (2002) is to be able to keep the robots together in an *unbounded* environment, whereas in Støy (2001), the experiment takes place in an arena (a very large arena of 35x4 [m], but still an arena). In this regard, our own adaptation, which uses an arena and takes advantage of its boundaries, is closer to Støy (2001).

- Files location: https://github.com/jrlauper/jrl_epuck/tree/master/jr_demo
- Main files: `run_alpha_algo.c/h`
- Functions:
  - `void run_alpha_algo(int goal_beep_number);`
  - `void run_alpha_algo_from_to (int max_goal_beep_number, int min_goal_beep_number, int cycles_number);`

- Robots selectors: 0 to 3
  - [selector 0] threshold going from **6 to 1** by 50 cycles (then from 1 to 6, and so on);
  - [selector 1] threshold of **5**;
  - [selector 2] threshold of **3**;
  - [selector 3] threshold of **1**;
- Demo video:
  - [selector 0] `https://youtu.be/PN_FrZWih0Q`
  - [selectors 1,2 & 3] `https://youtu.be/PN_FrZWih0Q`
- Number of robots: 2 to no limit; successful tests made with 6, 9, 10 and 11 robots.
- Arena size: no specific requirement, but enough space to enable some exploration to the swarm; the size of the arena used in the video is 180x140 [cm].
- Robots to use: the code enables to use together e-pucks HWRev 1.1 and HWRev 1.3. Because of the difficulties met with volume detection (see 5.3.2 below) we recommend, if possible, to use e-pucks of the same version to optimize the results. While working on the code we used mostly HWRev 1.3. Thus, the code should work especially well with them.
- Reflective tape: for the avoidance behavior, it is assumed that the robots are equipped with reflective tape : 3 [mm] on the lower part of the bumper (see C.1 [Hardware] E-pucks mutual detection problem (with IR proximity sensors) for more details). If this is not the case, the IR proximity sensors threshold should be modified accordingly in the code.

5.1 Experiment description

Let's start by describing what a viewer of the experiment observes, and let's start with the simplest situation: two robots.

[2 e-pucks] After the robots are switched on, each of them starts beeping at a regular pace while moving. Visually, each emitted beep is accompanied by the robot's body lighting up in green. When one robot perceives the other robot's beep loud enough, one of its red leds lights up. As long as the two robots are close enough, each one perceives the beep of the other loud enough, and they move independently in "explorer" mode, exploring free spaces and avoiding obstacles. When they move too far away from each other (about 15 to 20 cm, see p. 40), they no longer perceive each other's beep loud enough. Consequently, no red led lights up anymore, and each robot turns around to try to come back to a position where it was still able

to perceive the beep of the other one loud enough. Seen from outside, the general behavior of the two robots gives the impression that they are kept together in a certain range through an elastic band (cf. Nembrini, 2005, pp.84-85 for this last comparison).

If we add more robots:

[> 2 e-pucks] Each robot can perceive *several* robots through their beeps, if they are perceived loud enough. Visually, each beep perceived loud enough during a cycle will be accompanied by a new led that lights up on the leds circle – a *cycle* corresponding to the time separating two emitted beeps. If a robot perceives fewer beeps from one cycle to another, it can react in two ways. If the total number of beeps heard during the last cycle is lower than a certain threshold, the robot performs a corrective half-turn movement. If the number is higher or equal to this threshold, its behavior remains unchanged.

5.2 Initial algorithm (Støy, 2001; Nembrini et al., 2002; Nembrini, 2005)

Before considering its adaptation to e-pucks, let's start by the initial version of the algorithm, which assumes neither e-pucks as robots nor sound as communication means. Besides of being capable of fairly precise movements, the robots must have two essential hardware characteristics:

(1) sensors to detect and avoid potential obstacles (and of course other robots); and most importantly,

(2) a means of communication of *limited* range.

(2), limited range communication, plays a crucial role in the algorithm. Indeed, thanks to it, it is possible to assess the distance separating two robots without them possessing or transmitting any information about their position: if there is communication, the two robots are close enough, i.e., close within a certain radius; while if the communication stops, they are too far away from each other. Such communication is called "*situated* communication". In such communication, the meaning of a received message comes not only from the content of the message itself, but also from the properties of the signal carrying the message (Støy, 2001, sect.1).

5.2.1 Basic algorithm (*without* threshold)

Thanks to the limited range communication, a first algorithm can be proposed to keep the robots together while exploring their environment. The basic principle of this algorithm is the following one:

Let a *cycle* be a time corresponding to the duration separating two messages sent by a same robot; and let assume that the cycles are roughly the same between robots, with some margin of error and drifts in time. During each cycle, each robot counts the messages received from the other robots which are in its range. Then:

- as long as this number remains unchanged between two cycles, the robot keeps exploring the environment: it goes straight ahead while avoiding obstacles;
- if this number decreases, the robot turns around and tries to return to a situation with as many messages as before; and, finally

- if this number increases, if new messages are received, the robot rotates by a random angle before returning to exploration – this random angle favoring the exploration behavior of the swarm as a whole.

The pseudo-code of this first algorithm can be written as follows:

Algorithm 2: Swarm coherence algorithm – basic form (without threshold)

```
1  m: number of messages received;
2  cycle_length (in iterations number);
3  i_message_to_send = random modulo cycle_length;
   /* to avoid all messages to be sent at the same time - not present in the
      original version of the algorithm */
4  i := 0;
5  m := 0;  last_m := 0;
6  explorer behavior;
7  while true do
8      if i == i_message_to_send then
9          sends message;
10     end
11     i++;
12     listens for messages from robots in range → increases m accordingly ;
13     if i == cycle_length then
14         if m < last_m then
15             half-turn;
16         else if m > last_m then
17             random turn;
18         end
19         last_m := m;
20         m := 0;
21         i := 0;
22     end
23 end
```

Figure 5.1: Swarm coherence algorithm – basic form (without threshold)

5.2.2 Improved algorithm (*with* a threshold α)

In its basic form, the coherence algorithm has a defect: it is too reactive to communication loss – a fact that becomes particularly apparent when the number of robots involved starts to be large. Indeed, if the robots turn around as soon as there is the slightest loss of message, you get a clump of robots that explore its environment very little. At the other extreme, reacting only when a robot loses its last connection would be not reactive enough: you would get a too loose swarm, lacking coherence. To allow to adjust the reactivity of the swarm between these two extremes, Nembrini (2005) and Nembrini et al. (2002) propose introducing a threshold α

such that robots react to communication loss *only* when the total number of messages received is *below this threshold*. With such an α threshold,

- hyper-reactivity corresponds to: α = the total number of robots (or, in any case, α = the maximum number of different messages a robot can perceive within the range of its limited communication);
- while under-reactivity corresponds to $\alpha = 1$.

Once introduced, the α threshold allows to adjust the coherence according to the desired goals. It remains to determine its suitable values empirically.

The previous algorithm (Figure 5.1) only needs to be modified very slightly in order to introduce the α threshold. Indeed, to do so, it is sufficient to replace the condition in line 14:

if $m < last\_m$ **then** ...

by

if $m < last\_m$ and $m < \alpha$ **then** ...

5.2.3 Algorithm limitations

A major limitation of the improved algorithm resides in treating any connection between robots on a par. However, not all connections have the same importance for coherence as this passage clearly explains:

> When a robot (or a group) is linked to the rest of the swarm by *a single communication* link, the danger lies in the possibility of a robot not reacting to the loss of such a connection essential to global connectivity, because the number of remaining connections is above the threshold. In graph theory an edge representing such an important connection is known as a bridge. In the case of a vertex that is essential for connectivity, it is called a cutvertex: in such a situation, a robot failure would lead to disconnection.
>
> (Nembrini 2005, p.91, our emphasis; see Figure 5.2 illustration)

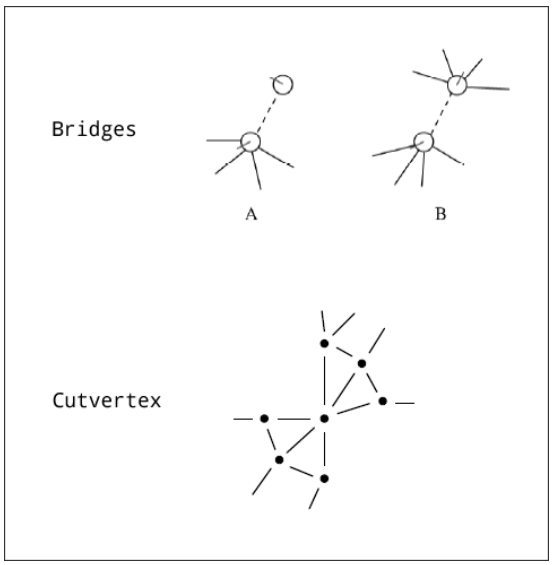


Figure 5.2: Extreme connection states in robots swarms favoring disconnection: bridges and cutvertices (Nembrini, 2005, p.91. Fig.4.10 and 4.11)

The loss of a bridge leads to an immediate disaggregation of the group. Thus, a loss of such a connection should be immediately corrected by a half turn of the two robots in question. However, if the total number of messages received by either robot during the last cycle is greater than or equal to α, they will not perform any corrective behavior. On the other hand, the loss of a connection with a cutvertex favor such a disaggregation in the next cycles and should be corrected too. The fact that the improved algorithm makes no difference regarding the more or less high significance of some connections for coherence is its greatest weakness and what tends to lead to coherence loss. For this reason, the improved algorithm we adapted to e-pucks is just a very first step in Nembrini (2005) and Nembrini et al. (2002) toward a more elaborate and more efficient algorithm that corrects this problem.

Let's note that the limitation favoring swarm disaggregation is, of course, found in any adaptation of the algorithm. As can be seen in experiments and on demos videos, our adaptation is no exception to it.

5.3 Adaptation to e-pucks

5.3.1 Avoidance and communication

In order to barry out the coherence algorithm with e-pucks we need two things :

(1) an avoidance behavior, and

(2) a communication means *with a limited range*.

(1) For the avoidance behavior, we use a Braitenberg vehicle behavior of explorer type. This behavior involves more or less inhibiting the wheels' normal rotation according to the values returned by the IR sensors of a robot perimeter (see 3.2.1 before, for more details). The IR sensors distributed all around the e-puck enable to easily achieve this aspect.

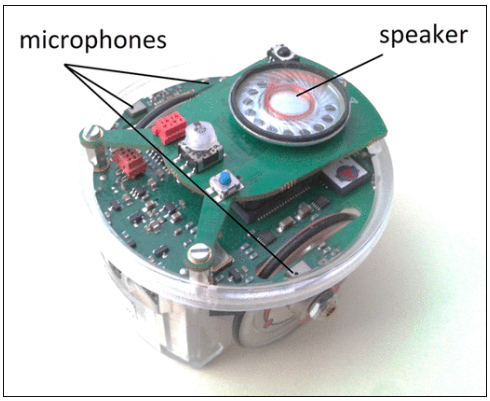


Figure 5.3: The speaker and the three micros of the e-puck (Nemec et al., 2017)

(2) We decided to implement the inter-robot communication by taking advantage of the e-puck's sound capabilities: its top speaker and its three microphones (see Figure 5.3). To carry out the coherence algorithm and its idea of *situated* communication, the meaning of each message sent has to be made up of two facets (cf. Støy, 2001, sect.1):

(a) the content of the message itself; and

(b) a physical property from the message carrier – in the case of our algorithm, this property being the *limited range* of the communication.

(a) In our adaptation of the algorithm, the content of the message is a beep sound: a brief tone emitted at a frequency of 3072 [Hz]. This beep can be identified by the robots and differentiated by them from any other sound. The beep is emitted "on the fly" using the module we developed (see B.1.4 for details), and is recognized using the robot's FFT module (see B.2.6). In order to facilitate the observation of the course of the experiment,

- each message sending, each beep emission, is accompanied by the illumination of the robot body in green, while
- each message received during a cycle lights an additional red led on the robot bumper.

Unfortunately, recognizing a beep emitted by another e-puck is more complicated than expected (a fact already mentioned in the previous experiment in 4.3.2). Indeed, while the FFT module plays its role very well, this is not the case of the e-puck speaker: in addition to the frequency of the beep it is meant to emit and which is clearly recognizable to the ear, the speaker emits a series of extra sound frequencies at roughly as high a volume. Result: when you try to recognize a beep produced by an e-puck, you can't just identify the frequency supposed to be emitted, but each time you have to recognize a whole *group* of possible frequencies. To identified the beep used here, for example, you have to compare the frequency returned by the FFT module not to 1, but to 12 values! See B.3 Sound frequency communication between e-pucks for a detailed discussion of this difficulty.

Same message but random delay. By using a same message for all the robots, a beep of a same specific frequency, the algorithm used is closer to that of Støy (2001) than that of Nembrini et al. (2002) and Nembrini (2005). Indeed, in the latter ones, the content of the messages is the identity of the sending robot. Since in our adaptation, the content of the messages is the same for all robots, and since the algorithm is essentially based on counting the number of messages received, it would not be desirable for all robots to send their messages at the same time. For this reason, each robot emits its message with a small randomly generated delay with respect to the start of its cycle (using the module described in C.2), while the robot's cycles, on the other hand, run more or less synchronously: that is, they start at the same time if a remote control is used to start the robot, and then, there are slight drifts over time. The use of a randomly generated delay does not completely prevent that some messages sent are sent almost exactly at the same time and that, as a consequence, are not counted because received simultaneously. This risk is even reinforced by the cycles drifts that appear as the experiment progresses. However, without having precise numerical data to rely on (we will come back to this point in the results sub-section), observations and experimentations seem to show that the use of this random delay works pretty well. Indeed, the number of received messages appearing on the robots through the leds that light up reflects fairly well the number of messages supposed to be received, whereas the general behavior of the swarm corresponds fairly well to the desired behavior.

(b) The *sound volume* of an emitted beep as it is perceived by the other robots is the physical property that will contribute to the complete meaning of the message. We don't have here an "all or nothing" property as in the experiment of Støy (2001) or in the simulations of Nembrini et al. (2002) and Nembrini (2005): communication vs. absence of communication. If we wanted to have this with a beep sound, we would need either a very large arena, big enough for the robots to be able to be at such a distance that they can't hear each other; or we would have to use a beep emitted at a very low volume, which would go against our wish for a demonstration whose attributes are easily grasped by the audience. The property chosen and used is a certain threshold of sound volume – the e-pucks' microphones being fully capable of returning this value. Below a certain perceived volume, we will consider that communication no longer takes place, while above it, we will consider that it does. Given the e-puck dimensions (7 [cm] in diameter) and the other constraints related to the perception of volume discussed just below, we have used a sound volume corresponding very approximately to a distance from one robot

center to another of 20 [cm] (13 [cm] from bumper to bumper).

HWRev differences between microphones and speakers. It should be remembered that there are important hardware differences regarding microphones and speakers between versions 1.1, 1.2, and 1.3 of the e-puck: the speaker in version 1.1. is louder than the other two versions, while the microphones of version 1.3 are about 15% more sensitive than those of the other two versions (Table 5.1). To have an optimal running of the experiment, we recommend using, if possible,

Version	Production Year	Speaker	Microphone	Version
HWRev 1.1	2006	(opaque black) Produces **Louder** Sound	Less sensitive (about 15%)	HWRev 1.1
HWRev 1.2	2008	(transparent) Produces **Lower** Sound		HWRev 1.2
HWRev 1.3	2014		More sensitive	HWRev 1.3

Table 5.1: Sound hardware differences between e-puck implementations (*E-Puck Main Wiki*, GCtronic 2019c)

robots of the same version only. However, the code has been written in such a way to enable mixing e-pucks 1.1 and 1.3 without any adaptation. With a slight modification, the same code could also allow mixing e-pucks 1.2 and 1.3. However, it is not possible to mix e-pucks 1.1 and 1.2 with one and the same code, as the standard library only allows detecting the presence of e-pucks 1.3 (it does it by detecting the presence of the gyroscope absent from the two previous versions). If someone really wants to mix 1.2 and 1.1. or all three versions of e-pucks, it is always possible to flash a slightly modified code in the e-puck 1.2 instances. The following videos show how we achieved good compatibility and quite similar volume behavior while mixing HWRev 1.1 and 1.3 e-pucks.

- Emitting robots : HWRev 1.1 vs. 1.3. / Listening robots: all HWRev 1.3.
  `https://youtu.be/CEjNUAbJp1g`
- Listening robots: HWRev 1.1 vs 1.3. / Emitting robots: all HWRev 1.3.
  `https://youtu.be/1sPNnn6-OlY`
- Emiting and listening robots: HWRev 1.1 vs 1.3:
  `https://youtu.be/Xp_bidK0A_4`

The following screenshot gives you an idea of what you see if you launch the videos:

Figure 5.4: Screenshot of one of the videos showing the sound range visualization program in action. At the very left, the robot surrounded by a paper is the robot emitting the beep. The robots that follow it to the right light up if the volume of the beep is perceived above the threshold.

The program used in the videos to visualize volume perception is the following one:

- files location: https://github.com/jrlauper/jrl_epuck/tree/master/jr_demo
- main files: `run_alpha_algo.c/h`
- function: `void run_scope_visualizer_demo (int min_start, int min_end, int increment_number);`

5.3.2 Volume issues

A priori, it seems easy to implement the communication limited scope by using the sound volume of the e-pucks. Indeed, we have a speaker to produce sound, and we have microphones to perceive it. Furthermore, the micros are able to return the perceived volume through functions of the standard library. However, as with the recognition of the frequency of a beep emitted by another e-puck, recognizing the volume of a beep emitted by another e-puck is unexpectedly problematic. The source of the problem is twofold. It comes from

(1) the non-center position of the speaker; and from

(2) the arrangement of the microphones on the e-puck body (see Figure 5.5).

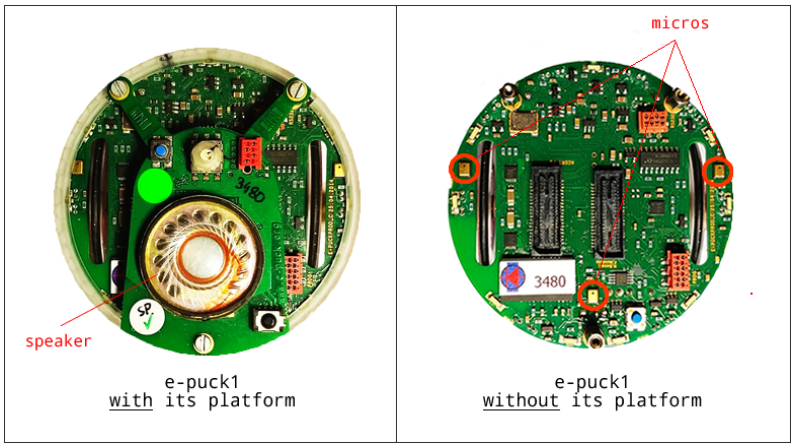


Figure 5.5: The position of the speaker and the three microphones on the body of the e-puck (seen from above)

B.4 "E-puck distance assessment through volume" discusses in detail the use of the microphones to evaluate the distance of another e-puck emitting a beep, as well as the difficulties this raises. We summarize here the conclusions and key points of this discussion.

(1) The speaker of the e-puck is placed on a small platform. The speaker is not placed at the center of the e-puck – its center being about 1.2 [cm] away from the center of the robot (see left part of Figure 5.5). However, the difference in perceived volume generated by this difference in position to the center is surprisingly high. To see it, let's take two e-pucks HWRev 1.3. One is used to emit a beep at volume 80, the other to record the average of the recorded volumes. We consider different distances between the two robots while varying the orientation of the emitting robot between the two extreme distances: "close" and "far" (see Figure 5.6). The Figure 5.7 shows the measurements we got for the perceived volume when we did it with two robots of our lab.

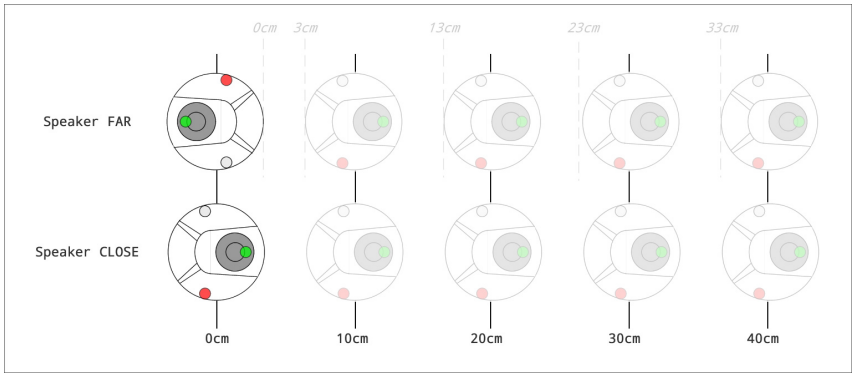


Figure 5.6: The different positions of two e-pucks used to measure the effect of the speaker orientations "close"/"far" on the perceived volume

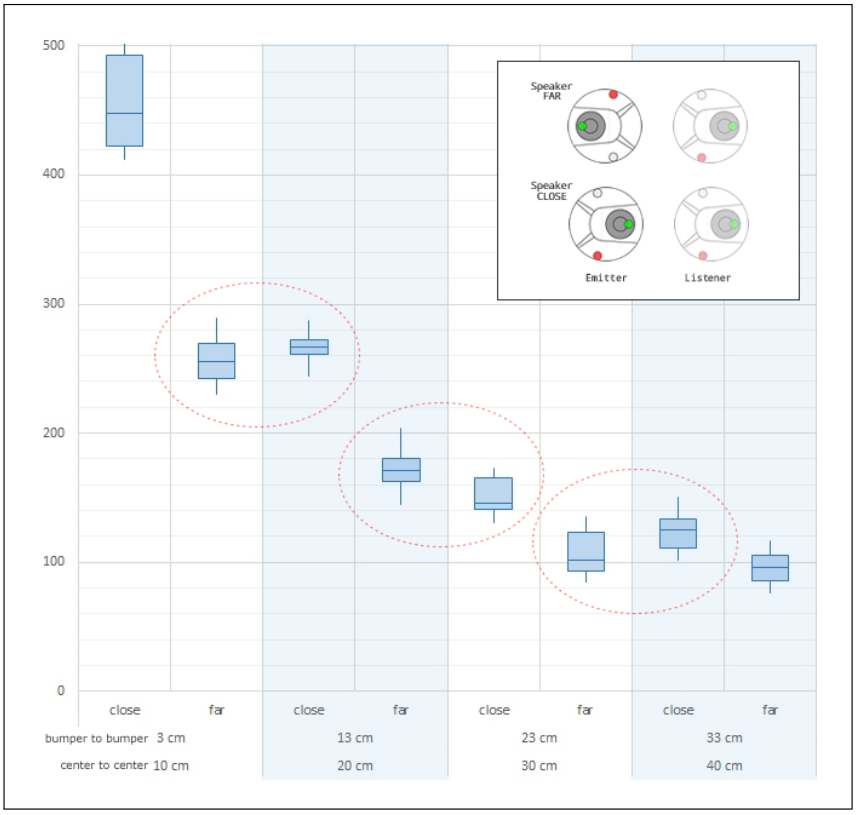


Figure 5.7: Measurements of the (average) perceived volume wrt the distance separating two robots of our lab (3435, 3480). The orientation of the emitting robot's speaker varies between the two extremes "close"/"far". The recording robot stays in the same orientation during all the measurements. Between 150 and 500 measurements were made for each boxplot. The red circles emphasize the problematic similarity of volumes measured between the "close" and "far" orientations for distances separated by 10 [cm].

Looking at Figure 5.7, we can observe two things:

- the change in the "close"/"far" orientation of the emitting speaker generates a large difference in the perceived volume;
- the volume measured in the "far" orientation for a certain distance is very close to the volume measured in the "close" orientation for 10 extra cms. This is especially troublesome if you want to use the volume to assess this distance.

(2) The arrangement of the microphones on the body of the e-puck (Figure 5.5, right part) generates a significant variation in the perceived volume too. To see it, we took two e-pucks as earlier: an emitter in "close" and "far" orientations alternatively, and a listener placed at different distances. But this time, the listener rotates on itself by 45° steps during the measurements. By proceeding in this way with two robots of our laboratory, we obtained the values reported in Figure 5.8.

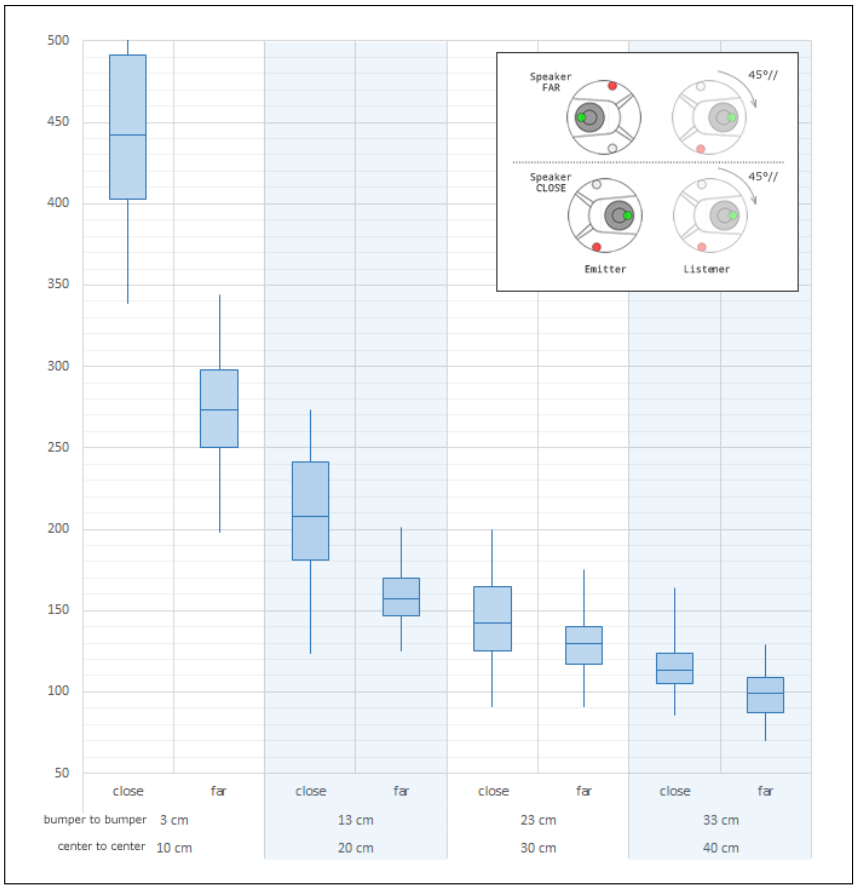


Figure 5.8: Measurements of the (average) perceived volume wrt the distance separating two robots of our lab (3435, 3480). The orientation of the emitting robot's speaker varies between the two extremes "close"/"far". The recording robot rotates on itself by 45° steps during the measurements. More than 800 measurements were made for each boxplot .

As can be seen in this figure, the intervals of volumes measured for a same distance and a same emitter orientation are considerable. More problematic for the distance assessment, there are large overlaps between them. However, when the robots explore an environment, the emitter's and the listener's orientations vary and cannot be predicted. This makes it difficult to use the volume of a beep emitted by one e-puck as measured by another e-puck to estimate the distance between two robots. In B.4, we discuss different options we consider to improve

this situation (including, among others, adding a small reflective cardboard above the emitting robot). We also show there why, unfortunately, they all fail to improve this situation.

From this situation, what can we do? Simply giving up trying to implement the algorithm with e-pucks using their sound? or trying to adapt it as well as possible given these large variations? We tried the second option by looking for a compromise.

First of all, if we look at the graph of Figure 5.8 , we can see that it is not possible to find a volume threshold corresponding to a single specific distance: a threshold such that, if the perceived volume is higher than it, the distance separating the two robots is smaller than some specific distance; whereas if the perceived volume is lower than this threshold, the distance separating the two robots is greater than this specific distance. This is made impossible, even approximately, by the too large overlaps between the measured volumes.

In order to perform the experiment properly, it would be good to be able to keep the robots within a radius of less than 20 [cm] center to center. Thus, ideally, we would like a threshold such that: if the perceived volume is below this threshold, the message is not considered as transmitted, and the robot is beyond 20 [cm]; while if the perceived volume is above this threshold, the message is considered as transmitted, and the robot is within a radius of 20cm. But as just seen, this is simply not possible. Therefore, let's consider other possible thresholds while keeping the graph of Figure 5.8 in front of us:

- If we take a volume threshold of 200 as a criterion, i.e. a message is considered as transmitted if the beep is perceived at a volume higher than 200, then no robot beyond 20 [cm] will perceive such a volume. However, many robots at a distance lower than 20 [cm] will not perceive this volume either.
- At the other extreme, if we take 130 as the threshold, then all robots within 20 [cm] will perceive the beep loud enough. However, a lot of robots located at 30 [cm] or 40 [cm] will also hear the beep loud enough.

To perform the experiment as well as possible, we sought a trade-off between these two extremes. After several empirical tests using the scope-visualizer function mentioned above (5.3.1, page 41), a threshold of 150 gave us good results: most of the time, robots at a distance of less than 20 [cm] perceive the beep loud enough in the different orientations, whereas, at 20 [cm], this is already no longer the case.

In the following video, we can see that 150 gives particularly good results: `https://youtu.be/oM8CY36nAgM?t=63`. Unfortunately, the results are not as good and consistent when robots are in motion during the experiment.

5.3.3 Corrective half-turns and increased complexity

The pseudo-code proposed to realize the algorithm (Figure 5.1) by adding the necessary modification in line 14 ($m < \alpha$) is very elegant. However, it represents an *idealized* version of the algorithm and cannot be used as such for a concrete implementation. The idealization concerns the robot's rotations: the corrective half-turn when a connection is lost below a certain threshold, and the random turn when a new connection is acquired. The idealization resides in the fact that the pseudo-code makes "as if" these rotations take place *instantaneously*. But in reality, each of these rotations takes time, and during this time, two things can happen:

(a) the moment of the cycle at which the robot must send its message can happen;

(b) the robot can receive messages.

Moreover, one thing necessarily happens during these rotations: the counter i must continue to increment if we do not want the duration of the cycles to vary too much from one cycle to another. Therefore, it is essential that each robot can send and receive messages during these

rotations while the i-counter continues to increment. The addition of these constraints makes the very elegant basic code of the algorithm considerably more complex, as the reader can see by looking at the code of `run_alpha_algo.c/h`: we must modify the main loop, but also and especially the rotation behavior which must allow message transmission and reception, as well as general counter incrementation. In the real code, this touches above all the function :

```
static void alpha_direction_turn_behavior(int turn_degree_angle)
```

Without going into the details of the final code, the main decision we took was to make these rotations always occur at the beginning of the cycles. It is also necessary to ensure that the rotation time is less than the duration of a cycle. When watching the demonstration videos, this is difficult to observe, but this last condition is well respected: a cycle lasts approximately 1.15 [s] (manual measurement over 20 cycles), while a half-turn lasts very approximately 0.6 [s].

5.3.4 Recovery mode

In the original context of Nembrini et al. (2002) and Nembrini (2005), the algorithm's environment is supposed to be without boundaries. However, in our experiment, we use an arena as in Støy (2001). This has an interesting consequence: there is good chance that a robot that has lost contact with other robots can re-establish a connection later with a robot or a group of them. Nevertheless, in the original version of the algorithm, there is a lot of chance that an encounter between an isolated robot and another robot (or a group of robots) does not end in a reconnection. Indeed, let's assume that a solitary robot A suddenly hears again another robot as it approaches a group (as illustrated in Figure 5.9):

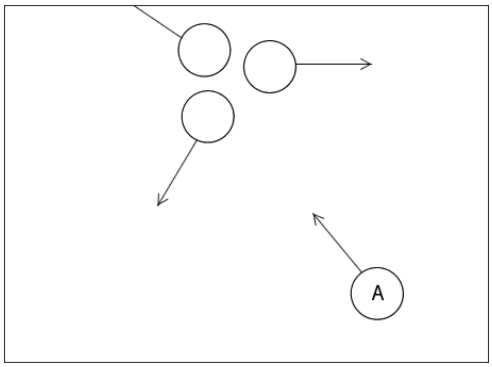


Figure 5.9: A "lost" robot A perceives other robots beeps again.

Having won a connection, the robot A performs a small random rotation before going into the explorer mode. However, during the next cycle, there is good chance that it will lose the unique connection just acquired: the movement of the other robots, a variation in volume perception, etc... The causes of such a possible loss of connection are multiple. In this case, the robot A, which has lost a connection (very likely below the α threshold), turns around. It will thus move away from a promising direction.

To promote reconnections, we modified the original algorithm as follows: when a robot that has been alone for some cycles hears another robot, it continues to behave as an explorer for some cycles even if it loses connections below α. For example, in the above situation, it will continue to head toward a promising place for reconnection instead of going back to where it came from. This technique greatly increases the chances of reconnection – a fact observed in the experiments.

5.3.5 Power supply issue

An unexpected problem occurred after our first successful implementations of the algorithm. The algorithm worked well within the limits and difficulties discussed above – especially the ones regarding volume perception (see 5.3.2). However, during the course of the experiment:

- some robots reset;
- some others became zombies: i.e., they moved and remained in explorer behavior, but otherwise, stopped to react at all: they did not send or receive messages and did not perform corrective behavior anymore; finally
- some others completely switched off without switching on again.

The identification of the problem was complicated by the fact that there was no systematicity, neither about which robot stopped, nor about the batteries used, nor about the moment in the algorithm when it happened. The general cause seems to be a peak in electrical demand as it is true that the robots are heavily solicited: IR detection, FFT recognition, "on the fly" sound generation, volume detection, and wheels motion.

In spite of testing and searching in many directions, we were unable to solve the problem completely. However, two measures were effective to some extent:

- decreasing the robot's rotation speed to avoid too sudden and too high a demand for electricity to move the wheels during the corrective U-turn;
- modifying the value of the agenda that manages and regularly restarts the explorer behavior in the background.

These two measures allowed us to eliminate the zombie and power off behaviors, but unfortunately, resets continued to happen despite all our attempts.

Thus, if nothing is done, over the course of an experiment, some robots can reset and become immobile and non-reactive. If we are lucky, their number is small; but they can easily be half of the robots out of 10 after just a few minutes in the less fortunate cases. How can we make the experiment usable in this state? We managed to do this in the following way. Our code allows you to choose the robot program and activate it using an IR remote control. This function is, for example, especially convenient to start all robots at the same time. In the case of this coherence algorithm, we made the following modification: once the program has been launched, the robot does not react to the remote control anymore; however, if the robot is reset, it becomes reactive to the remote control again. Thus, it is possible to quickly and remotely restart a reset robot by manually assisting the experiment.

However, this modification makes one of the planned demonstration modes a bit flawed. Indeed, before encountering this problem, we wanted the alpha threshold to vary automatically: that it regularly decreases by one unit every x cycles and then increases in a similar way. We kept this mode (selector 0), but it is obvious that it works only very roughly, given all the possible resets and restarts.

5.4 Results

5.4.1 Beacon attraction behavior

In this demonstration, we adapted the first step of the coherence algorithm of Nembrini et al. (2002) and Nembrini (2005), which, by a threshold α, allows to adjust the reactivity of the swarm in case of communication loss. However, in these papers, this coherence behavior is complemented by another behavior. In addition to staying together in a swarm, robots are attracted by a beacon. This additional behavior is very interesting because it allows seeing how a swarm and its coherence evolve in a directed exploration. You can then observe how the swarm changes its shape while moving towards the beacon and how it varies depending on

the α coherence threshold set. You can also observe how the swarm goes around or surrounds one or more obstacles to reach the beacon.

Our initial goal was to realize the coherence algorithm *plus* this attraction behavior by a beacon. In our real implementation, the role of the beacon was played by a flashlight with sufficient IR radiation, while we used the light sensitivity of the IR sensors of the e-pucks to realize the attraction. We have implemented this additional behavior, and it can be found in two programs of the final demonstration set: program 11. "Rotating toward light" and 12. "Exploring and light follower" (see chapter 6).

Unfortunately, the power supply problem encountered in the coherence algorithm made us give up on this project. The most likely hypothesis that could explain this problem and the unexpected stops generated is that the workload required is already too high for the e-puck resources in the coherence algorithm without adding the attraction to a beacon. If this is indeed the case, the addition of this attraction behavior should add more current jumps favoring even more unexpected stops. This is what we observed in tests when we tried to add this attraction behavior. The addition of this behavior in a version running without too many stops is clearly a goal to achieve in future works given the interest it would have at demonstration level. Optimization of the code and lower-level settings of the robots could perhaps make it possible.

5.4.2 A more powerful algorithm: not feasible?

As mentioned several times in this chapter, the algorithm we have adapted here is only a first step in Nembrini et al. (2002) and Nembrini (2005). After this first step, these works present another, more powerful algorithm. This one allows for better identification of the key connections of the swarm: those that should not be lost to avoid swarm dissociation. This more powerful algorithm tends to solve the problems raised by not sufficiently taking into account bridges and cutvertices (cf. 5.2.3).

At the beginning of our work, we thought we could try to carry out this more advanced algorithm too. However, the problems met for the first coherence algorithm absorbed much of the time available for this project. Hence the fact that we had to give it up.

However, the power supply problem encountered raises the question of the very feasibility of this more powerful algorithm with e-pucks. Indeed, as described in the papers, the algorithm in itself only slightly increases the computing load for the robots. Yet, its concrete realization with e-pucks would significantly increase their workload. Indeed, to realize this more powerful algorithm, the content of the messages sent and received must include each robot's identity. This should be possible by using a different beep for each robot. But recognizing many different beeps would be strenuous for the e-pucks because of the problem caused by their speakers. As a reminder, their speakers do not emit just the frequency they are meant to emit but also emit a large number of other frequencies at an equally loud volume. Therefore, to recognize a single beep emitted by an e-puck, the frequency returned by the FFT must be compared not to a single value but to many values. For example, for the single beep used in the implemented algorithm, the value returned by the FFT must be compared with no less than 12 values to be sure to be correctly identified! (B.1.1 and B.3 for more details about this). The fact that a beep corresponds to a group of values has a second negative effect: to have a beep that is distinguishable for each robot, it is necessary to have no overlap between the frequency groups corresponding to each beep. This is quite laborious to achieve, but in addition, this considerably decreases the number of distinguishable beeps available – which is again detrimental to the very feasibility of the more powerful algorithm with e-pucks. This last problem is further reinforced by the fact that the FFT recognition offered by the e-puck is not very fine and that the returnable values are limited – see B.2.3 [Software] Limited values returnable by FFT for more details on this topic.

Despite all these problems, we managed to get 12 well distinguishable beeps when we still thought we could achieve this more powerful algorithm and were working on it: see B.3.4 in B.3 "Sound frequency communication between e-pucks". This would allow us to implement the more powerful algorithm with already 12 different robots: this is a start. A reader wanting to try further the adventure of implementing the more advanced coherence algorithm with e-pucks could use these results as a basis to extend.

5.4.3 Coherence observed

After all these regrets, let's consider what we get in the end. First of all, we get a coherence algorithm which requires regular manual interventions to counteract the regular and unpredictable resets of the robots – what is clearly not ideal for a demonstration. Indeed, we would have liked to be able to launch the demo and let it run on its own. But at least, thanks to the remote control "trick" to restart the robots remotely, the demo can be run over a long period of time. These manual restarts also mean that you can't really use a version of the algorithm with a threshold changing on its own (see 5.3.5 about this). But apart from this problem, what about coherence?

First, the demonstration suffers from the limitations of the algorithm itself which makes no difference as to the importance of some connections or some robots for the loss of connection. This in itself already favors robot losses and swarm breaks (see 5.2.3). Then, concerning the concrete realization of the algorithm. The implementation of the limited range communication by taking advantage of the perceived volume raises big problems as seen in 5.3.2: the arrangement of the e-puck's speaker and microphones makes it difficult to estimate the inter-robot distance using the volume. It makes it very imprecise, to say the least. But here too, what do we get very concretely when we nonetheless run the algorithm?

First of all, let's note that we don't really have any concrete figures – this would have required a lot of experimentation and repetition of the demonstration, which was not possible for us. However, within the limit of the experiments actually carried out, we can make the following observation: despite all the implementation problems encountered with e-pucks, and especially the problems related to volume perception, the coherence algorithm using the α threshold seems to work, i.e., it definitely ensures some coherence among the robots, even if this is weakened by the problem of volume which generates additional losses. Moreover, the change of threshold actually changes the swarm's coherence, as can be seen in the synoptic view offered by the demo video: `https://youtu.be/n0Ww2lsuaYE`. Finally, we have been able to observe that the added recovery option seems to work well.

6 The full demonstration set

Contents

The three experiments described in the previous chapters – collective heap building, synchronization, and coherence algorithms – represent this work's core. However, they occupy only a part of the e-puck's manual selector's available positions: more precisely, 6 out of 16. We, therefore, decided to populate the remaining selections with other interesting programs suitable for demonstrations. At the end of this chapter, we give the complete list of the programs included in our demo set: the ones you get on your e-puck if you flash it with the corresponding .hex file. However, before that, we introduce other aspects necessary for its use: how to make good use of the demo's initialization phase and how to use the remote control available to facilitate its use.

- Files location: `https://github.com/jrlauper/jrl_epuck/tree/master/jr_demo/jr_demo.X/dist/default/production`
- Main file (compiled and built): `jr_demo.X.production.hex`

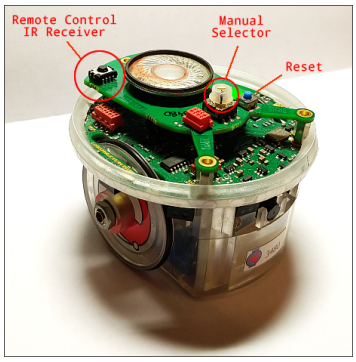
6.1 Remote control usage (selector position 14)

The demonstration set has been programmed to work very well without remote control. If you don't have a remote control, you can simply manually choose one of the 14 possible programs by selecting a position between 0 and 13 on the manual selector (Figure 6.1a). If, on the other hand, you want to use a remote control with the e-puck, set the manual selector to 14. The position 15 has been reserved for showing the battery charge status – we'll come back to this later when we'll describe the initialization phase in 6.2.

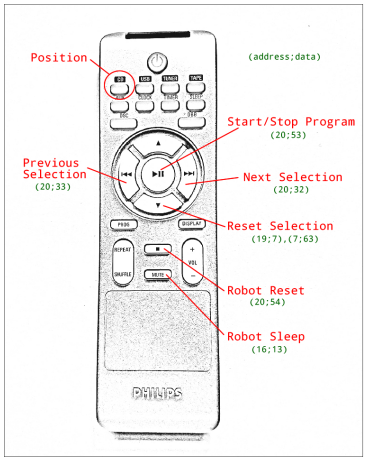
Although not necessary, using a remote control is recommended because it makes the use of the demo programs much more pleasant and easy. Indeed, once the e-pucks are switched on, the remote control allows you to select, start or stop a program. It also allows you to restart the robots, and most importantly, it allows you to do all this for all the robots at the same time. This becomes particularly convenient when the robots' number becomes high. Furthermore, as part of the swarm coherence project (chapter 5), a remote control enables you to remotely restart robots that unexpectedly reset because of power issues. This allows for an experience that continues smoothly.

In the course of the development of our projects, we created a small software module that makes it easy to use a remote control with the e-pucks. We also developed a small function that

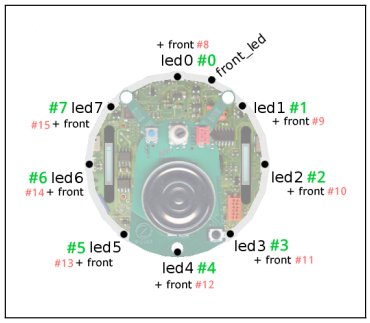
easily identifies the signals emitted by a remote control and modifies the code accordingly. All this is described in C.3 [Software module] Infrared remote-control module for selection.



(a) Position of the remote control IR receiver and the manual selector on the e-puck



(b) Sketch of the remote control we used



(c) Remote control leds-selectors correspondence

Figure 6.1: The IR remote control usage

6.2 Demo start and initialization phase

Initialization phase. When flashed with our demonstration, the e-puck start is accompanied by a series of calibrations necessary for a smooth running of the experiments. The whole process takes about 8 [s] and includes the following stages:

(1) IR proximity sensors calibration: all red leds light up (Figure 6.2c);

(2) microphones volume calibration: red leds are of while, body lights green (Figure 6.2f);

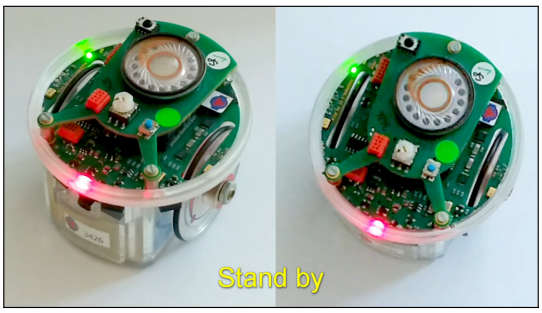
(3) battery charge display on the leds circle: preceded by a beep, then followed by a double beep (Figure 6.2e). The leds meaning for battery charge is explained in the next section 6.3.

(1) During the IR sensors' calibration, the e-pucks must be kept away from each other and from any objects. Otherwise, the sensors will return biased values, and the robots will behave erratically. Given the sensitivity of the infrared sensors (see Figure C.3 in C.1), the distance between an e-puck and another e-puck or other object does not need to be very large either: 4 [cm] is usually sufficient, while 5 [cm] is ideal.

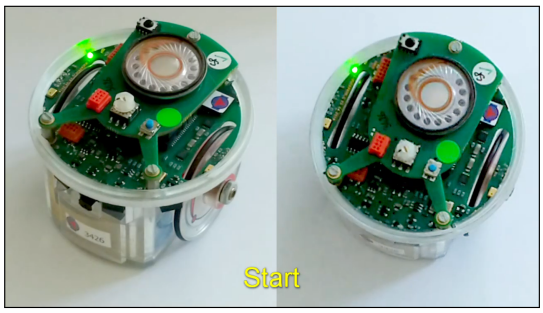
(2) During the microphones' calibration, there must be no noise in the room.

A small comment about the coherence algorithm: during this algorithm, the robots tend to reset themselves randomly and can be restarted manually with the remote control. These resets happening during the experiment, the robots are not necessarily far away from each other, while another robot may emit a beep during the volume calibration. These conditions are clearly not ideal for calibrations. However, experience showed us that this does not interfere too much with the course of this experiment.

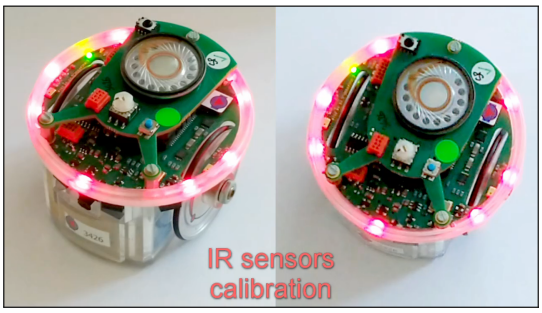
Given these different comments, starting an experiment with e-pucks using our demonstration will unfold as follows:



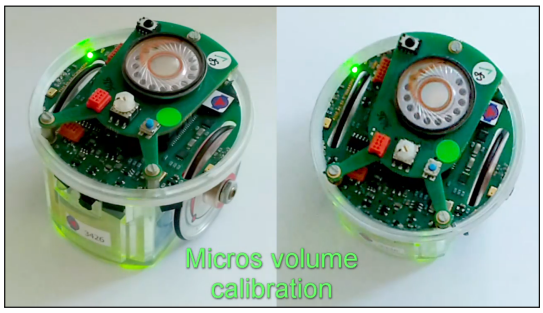
(a) Stand by



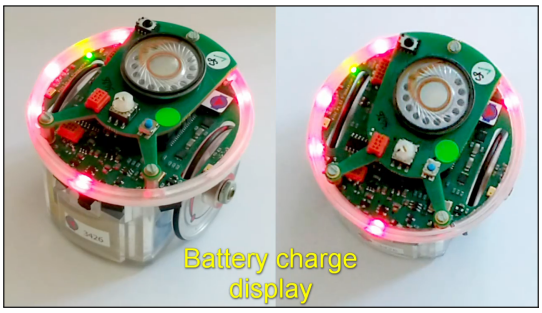
(b) Initialization start (e.g. after a reset)



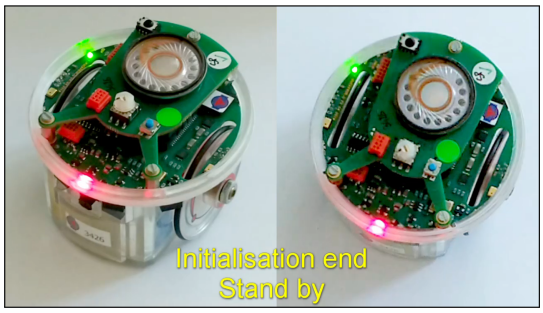
(c) IR proximity sensors calibration



(d) Micros volume calibration



(e) Battery charge display



(f) Initialization end. Back to stand by.

Figure 6.2: Demo initialization phase stages.
The photos are taken from the following video: `https://youtu.be/fx46Hbjesd4`

1. the e-pucks are placed in the arena with sufficient space between them: at least 4 to 5 [cm] apart from each other.
2. they are reset;
3. after the reset, they are finally placed in the desired location – ready to run the selected program.

Remote control. If a remote control is used, once the robots are placed correctly:
- you select the program to run with the appropriate buttons and then
- you launch it with the corresponding button.

The remote control works very well, but sometimes, depending on the angle, only some robots pick up its signal while others do not. We recommend proceeding to a small test for an optimal experience simply by changing the selection to see if all robots receive the signal properly. Should this not be the case, the button to set the selectors of all the robots back to 0 avoids having to proceed to a new reset.

Suppose you don't use a remote control. In that case, the passage from 2. to 3. is more delicate but remain feasible for experiments requiring a specific placement different from the one suitable for calibration. From experience, the simplest way to achieve it is then to have at disposal a series of small cubes of polystyrene (or other material) whose width is smaller than the e-pucks wheels spacing while remaining larger enough to offer some stability. In this way, the e-pucks can be slightly raised during the calibration phase – their wheels beginning to rotate mid-air thanks to the elevation. Then they can be placed on the floor of the arena one after the other. The manipulation is most of the time possible. However, the swarm coherence experiment is e.g., much more complicated to set up in this way without a remote control.

Let us add a last remark about program change. When changing program, it is imperative to reset the robot. Indeed, some values are initialized during the first program cycles. So non-resetting can lead to unwanted behaviors. This is particularly the case if, for example, you move the selector from position 1, which runs the coherence algorithm with a threshold of 5, to position 2, which runs it with a threshold of 3. It is also during the initialization phase that the mode of the analog-digital converter (e_init_ad_scan) is determined and set by default to ALL_ADC. Hence, the special importance of resetting if the previous selection temporarily used its alternative MICRO_ONLY mode.

6.3 Battery charge display

We have programmed the e-pucks so that, on HWRev 1.3 e-pucks, the battery charge is displayed as follows:

- led 0 on : $\geq 10\%$
- led 1 on : $\geq 20\%$
- led 2 on : $\geq 30\%$
- led 3 on : $\geq 40\%$
- led 4 on : $\geq 50\%$
- led 5 on : $\geq 60\%$
- led 6 on : $\geq 70\%$
- led 7 on : $\geq 80\%$
- green body led on : $\geq 90\%$
- frontal led on : $= 100\%$

On the picture of Figure 6.2e above, e.g., all leds are lit up to led 5 included. This means that the battery is more than 60% charged.

With the robots of our lab, even when the batteries are completely full, it is very rare to have more than 70% of charge (led(s) higher than 6 on). But the batteries are not young and have been pretty much used.

E-pucks HWRev 1.1. and 1.2 do not allow access to battery charge. Therefore, in their case, we decided that the e-puck simply displays a leds cross (leds 1, 3, 5 and 7 on).

6.4 Full list of programs

You will find a table with the full list of the programs included in the final demo set on the next page. For easy access to it, we added it at the very start of this report too. For each program, you will find information about the arena and the number of robots to use. The C files and the function corresponding to the programs are also given to allow easy access to the corresponding codes. The table also provides an illustrative video for each program. For programs other than the core projects described in previous chapters of this report, these videos should be sufficient to grasp what they do.

Arena	# robots (suggested)			File	Function	Demo link
			Initialization	main.c		https://youtu.be/fx46Hbjesd4
180x140cm	2 - 15+ (9 - 11)	0	Swarm Coherence Algo. Alpha threshold 6-1 (cycles of 50)	run_alpha_algo.c/h	run_alpha_algo_from_to(6,1,50)	https://youtu.be/PN_FrZWih0Q
"	"	1	Swarm Coherence Algo. Alpha threshold 5	"	run_alpha_algo(5)	https://youtu.be/n0Ww2lsuaYE
"	"	2	Swarm Coherence Algo. Alpha threshold 3	"	run_alpha_algo(3)	"
"	"	3	Swarm Coherence Algo. Alpha threshold 1	"	run_alpha_algo(1)	"
140x70cm	1 - 5+ (5)	4	Collective Heap Buliding."Swiss XP"	runbraitenberg_mod3.c/h	run_braitenberg_swiss_EL()	https://youtu.be/AevML_FSlEM
	2 - 15+	5	Synchronicity through Beep Sounds. "Chirping"	run_chirping.c/h	run_chirping()	https://youtu.be/I6GSagp46zs
To choose	1 - 15+	6	Braitenberg Explorer	runbraitenberg_mod3.c/h	run_braitenberg_explorer()	https://youtu.be/yz6ijhXEd9Y
"	"	7	Braitenberg Explorer/Lover	"	run_braitenberg_explorer_and_lover()	https://youtu.be/iLH2aBzI9MI
	1	8	IR Piano & Full Sound Demo (paper)	e_freq_sound.c/h	freq_ir_piano_C_to_C1()	https://youtu.be/VEd9eDacvss
	"			"	freq_video_full_demo()	https://youtu.be/udtJUNvuzgM
	1	9	Robot Guiding by Whistling (GCTronic)	runfftlistener.c/h	run_fft_listener()	https://youtu.be/uaQSAEbv7jI
	1 - 15+	10	Rotating toward Hand Clapping (GCTronic)	runlocatesound.c/h	run_locatesound()	https://youtu.be/zhrk4egCTJU
To choose	1 - 15+	11	Rotating toward Light	runbraitenberg_mod3.c/h	run_light_lover()	https://youtu.be/kJHteD2_1GQ
"	"	12	Explorer & Light Follower	"	run_moving_light_lover(true)	https://youtu.be/uTkaBSoXj2o
	1	13	Simon Game: via Proximity Sensors or Whistling	run_simon.c/h	run_simon()	https://youtu.be/mGj8a8Cdw6Y
		14	Selector corresponding to IR Remote Control use			
		15+	Battery Charge Display			

Table 6.1: Full list of programs included in the demonstration set
Full playlist: `https://www.youtube.com/playlist?list=PLrscHgSUZPdr38tirAsB4_4Q93khKP9Rv`
Github: `https://github.com/jrlauper/jrl_epuck/tree/master/jr_demo`
"(GCTronic)": the code comes from GCTronic demos with slights mods
(more precisely programs 2 and 9 in complete demo 2: see section D.2).
"15+": 15 and beyond.

7 Conclusion

Our goal in starting this work was to give a second life to our old lab's e-pucks. Instead of putting them aside after some years of loyal educational service, we wanted to "reuse" them for robotics demonstrations intended for schools and university visitors. To this end, we wanted robots that work "out of the box", which is why we opted for *embedded* programming. We also wanted to focus our work on interesting projects from an educational perspective and representing classical experiments. Finally, we wanted to take advantage of the large number of robots at our disposal.

We selected three classical experiments using a behavior-based approach. This approach, often inspired by the animal kingdom, insists on the interaction between the robots and their environment. It uses actions caused in a rather direct way by the stimuli of the environment, rather than actions based on complex plans and representations (cf. Michaud and Nicolescu 2016, pp. 307–313; and "Behavior-based robotics",Wikipedia 2020). This type of approach generally requires less memory and computing resources than traditional AI approaches. It seemed thus particularly suitable for our e-pucks and their limited resources.

Exp1 (chapter 3). The first experiment adapts the experience of Maris and Boeckhorst (1996) to e-pucks. In this experiment, robots gather polystyrene cubes without being programmed for this purpose. Indeed, the main part of the experiment relied on the fact that, thanks to the deactivation of their frontal sensors, the robots do not perceive the cubes which are exactly in front of them and push them, whereas as soon as an object is wider (a cluster of cubes, a robot, a wall), the robots avoid it. The main challenge encountered in our adaptation of this experiment was a problem of mutual non-detection between e-pucks. Indeed, the transparent body of the e-pucks means that they poorly reflect infrared rays and are therefore poorly detected. To solve this problem we had to modify the bumper of the e-pucks by sticking some extra reflective tape. Once this modification made, the adaptation was a real success and the initial experience can be reproduced very well.

Exp2 (chapter 4). The second experiment adapts the mutual synchronization model of Mirollo and Strogatz (1990) to e-pucks. At the beginning of the experiment, each e-puck beeps randomly; but after a while, through small adjustments, they all beep synchronously. The first difficulty encountered in this adaptation was to manage to emit beeps with e-pucks. A surprisingly difficult task that led us to create a general library allowing it. The second unexpected difficulty was the recognition of a beep emitted by an e-puck. Contrary to what our ear hears, the e-puck speaker does not emit only the desired frequency at a louder volume than the others, but many other frequencies just as loud. This makes it very difficult to identify a beep emitted by another robot. Once the beeps' emission and recognition were mastered, the adaptation was a real success: the robots synchronize well, and the audience can well grasp what is happening thanks to the leds and the beep sounds.

Exp3 (chapter 5). In the third experiment, we wanted to adapt to e-pucks a coherence algorithm described in Nembrini et al. (2002) and Nembrini (2005) and inspired by Støy (2001). This algorithm is based on limited range communication: during each cycle, which is very short, each robot sends a message, receives messages from other robots close enough to it, and counts them. If, between two cycles, this number decreases and does so below a certain threshold, the robot turns around and returns to their previous situation. For communication,

we wanted, here too, to take advantage of the e-puck's sound capabilities – capabilities particularly suited for demonstrations. For the limited range aspect of the communication, we decided to use the volume of the beeps emitted by the e-pucks: communication occurs only when the volume of the received beep is above a certain volume threshold. Here too, we came across some surprising difficulties: the non-central position of the e-puck speaker, as well as its three microphones arrangement, result in a perceived volume that can vary enormously depending on e-pucks' orientation for the same distance. This makes it very difficult to evaluate the distance separating two e-pucks thanks to the volume. We tried to find a trade-off allowing the algorithm to work despite this situation – which is the case. Another unexpected and delicate issue encountered in the realization of this algorithm is a power supply problem leading to regular the robots' resets – and this, even after improvements. Not being able to solve this problem completely, we opted for a little trick that allows the experiment to run nonetheless smoothly: the auto-reset robots can easily be restarted remotely through the use of a remote control.

Contrary to experiments 1 and 2, adapting the coherence algorithm to e-pucks is far from being a complete success. The result nevertheless enables to observe how the algorithm works and to see how the coherence threshold influences the swarm. However, the difficulties encountered prevented us from adding an attraction behavior to a beacon, with all the interest this would have brought, or from trying to develop the better coherence algorithm proposed by Nembrini et al. (2002) and Nembrini (2005).

Extra programs. Finally, in order to have a more complete and varied demonstration set, we have added to these three main experiments:

- some classical behaviors (Breintenberg's explorer and lover),
- two initial demonstrations of the e-puck (guiding through whistling, rotating towards hand clapping),
- some demonstrations playing with light detection, and
- a little game: Simon's adaptation where the player can either use his hands or whistle.

E-pucks, when programmed in an embedded way, seem well suited for demonstrations. Moreover, embedded programming makes it possible to do amazing things given the modest technical resources of its microcontroller – for example, FFT recognition.

While performing experiments 2 and 3, a large part of our work consisted in exploring the sound capabilities of the e-puck: its speaker and microphones. Each one works well in isolation: once the speaker is able to emit sound, the speaker acts well as an actuator, while the micros work very well to recognize frequencies for whistling or musical instruments. However, the e-puck designers don't seem to have conceived it at all with the idea of using both aspects together – be it for beep recognition or distance evaluation through volume. Indeed, as mentioned many times in this work, the speaker emits a lot of extra frequencies in addition to the desired one at a loud level. This makes the recognition of a sound emitted by an e-puck complicated and resource-intensive. Not to mention that wanting to have several clearly distinguishable beeps becomes very demanding because of the very likely overlaps between groups of frequencies corresponding to different beeps. The speaker creates problems because of its non-central position on the robot too: this position generates large differences in volume depending on the emitting robot's orientation. Furthermore, the micros' arrangement also generates large differences in perceived volume depending on the listening robot orientation. The combination of these large volume differences make it difficult to use volume measurement to estimate distances as soon as the robots are in motion.

We do not know if the speaker of the new e-puck, the e-puck2 is better concerning the frequencies emitted, i.e., if it clearly emits the frequency meant to be played louder (about the e-puck2, see 2.3). However, unfortunately, its new position is even more eccentric, which

will make the use of volume measurement for distance evaluation between robots even more problematic (see Figure 7.1). The microphones' situation is at least better: the e-puck2 has an extra fourth frontal micro; besides, and especially, the rear microphone is no longer placed under the small platform as in the e-puck1 – a platform which played the unfortunate role of a resonance box or interfered with the sound reception depending on the listening robot orientation.

The use of sound in experiments seems to us a great educational plus. We can only hope that this aspect will be more usable in future implementations of the e-puck. Since the e-puck2, as previously the-pucks1, can welcome additional extensions, a small platform with a central speaker of higher quality would greatly improve the sound usability. Such a small platform should not interfere with the microphones, what would be possible either by covering none or all of them - unless one or more microphones are added to this platform too.

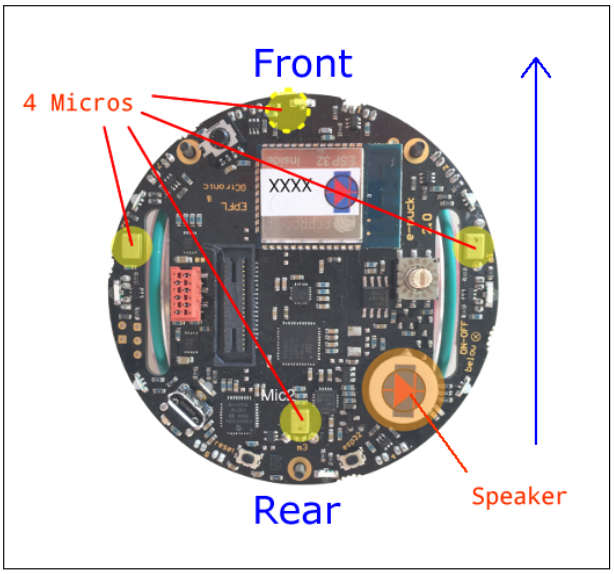


Figure 7.1: Speaker and micros positions on the e-puck2.
(Slightly modified version of one of the picture
of the "overview" section of the *E-Puck2 Main Wiki* (GCtronic, 2020b))

A Embedded programming with the e-puck: main resources

Contents

A.1 Standard library

The most important material to work with e-pucks in an embedded way is the standard library provided on the GCtronic github: GCtronic (2007):
`https://github.com/gctronic/e-puck-library/tree/master/library`.
It is written in C and Assembler. But in the most cases, knowing C is enough to program the e-puck in an embedded way. The Appendix E contains a tree-like version of the different files constituting the standard library.

The code of the standard library is extensively commented and often contains very helpful usage examples. A Doxygen documentation is also available here: GCtronic (2016). But it is of very limited use in our experience. Indeed, the vast majority of its content is already in the comments of the code, directly in their relevant context.

A.2 Other general materials

The material for using the e-puck in an embedded way is spread over two sites with a lot of overlaps:

- the official e-puck website: e-puck.org (E-Puck.org, 2018), and
- the site of GCtronic, the main distributor of the robot, especially its wiki dedicated to the e-puck (GCtronic, 2019c).

In general, it is better to privilege the GCtronic site:

- its wiki (GCtronic, 2019c):
  `https://www.gctronic.com/doc/index.php/E-Puck` and
- its GitHub (GCtronic, 2019b):
  `https://github.com/gctronic/e-puck-library`

It is the most up to date site and it contains definitely more explanations.

A.3 IDE and compiler

To work with the e-puck in an embedded way, we followed the advice of GCtronic wiki (GCtronic, 2019c) and e-puck.org website (E-Puck.org, 2018): we used

- the MPLab IDE v.5.40 (a modified version of Netbeans), and
- the XC16 compiler v.1.50,

both provided by Microchip Technology. Both can be found in the section "Development Environment" of the webpage devoted to the dsPIC30F6014A, the microcontroller at the heart of the e-puck (Microchip Technology Inc., 2019):
`https://www.microchip.com/wwwproducts/en/dsPIC30F6014A`

For the first steps with MPLab, see

- the GCtronic wiki (GCtronic, 2019c) (section 2.4.1 "MPLab X" and then, section 2.3.1 "Project Building"), and, optionally,
- the e-puck.org tutorial (Mondada and Bonani, 2006).

A.4 Bootloaders

In the GCtronic wiki (GCtronic, 2019c), the Section 2.7 "Bootloaders" lists the different softwares that should allow you to transfer the compiled code from your PC to the e-puck. In addition to the direct links of the wiki, these bootloaders can be found on the GCtronic GitHub under "e-puck-library/tool/bootloader/computer_side/" :
`https://github.com/gctronic/e-puck-library/tree/master/tool/bootloader/computer_side`
We tried the different bootloaders proposed for Linux and Windows. The only one that really worked for us is the Linux bootloader programmed in C++ based on libbluetooth-dev (epuck-uploadbt.cpp) (Michel and Magnenat, 2007):
`https://github.com/gctronic/e-puck-library/tree/master/tool/bootloader/computer_side/linux-libbluetooth`.

A.5 First steps

For the very first steps with e-pucks embedded programming, the GCtronic wiki offers a series of very useful little demos: (GCtronic, 2009).

In addition to the examples provided in the comments of the standard library, the two complete demos on the GCtronic GitHub are extremely helpful (GCtronic, 2017; EPFL, 2007). A description of the contents of both demos can be found in Appendix D.

B Using e-pucks with sound

Contents

In this work, we created school demos using several e-pucks programmed in an embedded way. Two of our main projects use communication between robots. We decided to achieve these communications using sound -– taking advantage of the speaker and the microphones of the e-puck.

- In section B.1, we explain how to produce sound with the e-pucks;
- in section B.2, how to use FFT recognition with e-puck; and
- in section B.3, the specific challenges that arise when you want to combine the use of both to allow the e-pucks to communicate together.
- Finally, in section B.4, we explore how the sound volume can be used to estimate the distance between two e-pucks.

The substantial length of this section reflects the many challenges that the use of sound with e-pucks raises. These challenges arose while we were working on our main projects. But their general nature justifies to present them in a separate way.

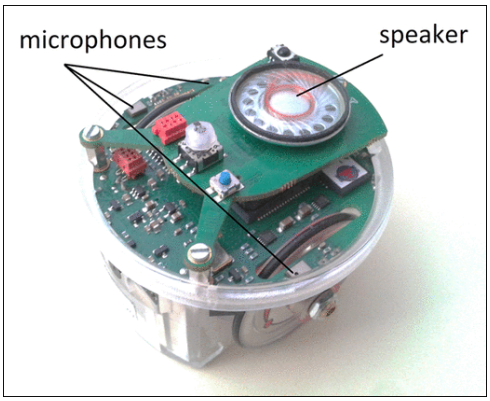


Figure B.1: The speaker and the three micros of the e-puck (Nemec et al., 2017)

B.1 How to produce sound with e-pucks

In this section we first discuss the hardware specifics of the e-puck speaker and the challenges they pose. Then we present two software modules that we have created in order to easily produce sound with e-pucks.

Table B.1 reminds the hardware differences concerning the sound between the different versions of the e-puck.

Version	Production Year	Speaker	Microphone	Version
HWRev 1.1	2006	(opaque black) Produces Louder Sound	Less sensitive (about 15%)	HWRev 1.1
HWRev 1.2	2008	(transparent) Produces Lower Sound		HWRev 1.2
HWRev 1.3	2014		More sensitive	HWRev 1.3

Table B.1: Sound hardware differences between e-puck versions (*E-Puck Main Wiki*, GCtronic 2019c)

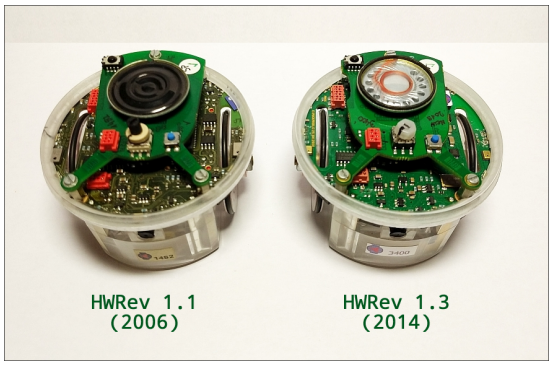


Figure B.2: Appearance difference between e-puck HWRev 1.1. and HWRev 1.3

B.1.1 [Hardware] The speaker: sound range and volume-frequency relation

The little speaker on the top of the e-puck responds surprisingly well to a wide range of frequencies. However, below 400 [Hz] the output volume starts to decrease strongly, while above 2000 [Hz] it increases strongly. This observation fits well the frequency-response graph of Figure B.3 drawn from Nemec et al. (2017), and made for the speaker of the e-puck HWRev 1.2. Such a graph should be the same for an e-puck HWRev 1.3 which apparently owns the same speaker as the HWRev 1.2. Another similar graph should be established for an e-puck HWRev 1.1 with its different speaker emitting sound louder. But we can assume that the general shape of the graph should be, *mutatis mutandis*, close.

Given the volume-frequency response of the speaker, if you want to work with the volume of the sound emitted by the robot, you really need to experiment to determine precise volume values. You can't simply assume, for instance, that the volume is constant or evolves in a linear fashion. The plateau that appears between 900 and 2000 [Hz] does however give us a whole zone where we can work comfortably with a volume that depends only slightly on the frequency emitted. A similar plateau happens between 3000 and 4000 [Hz].

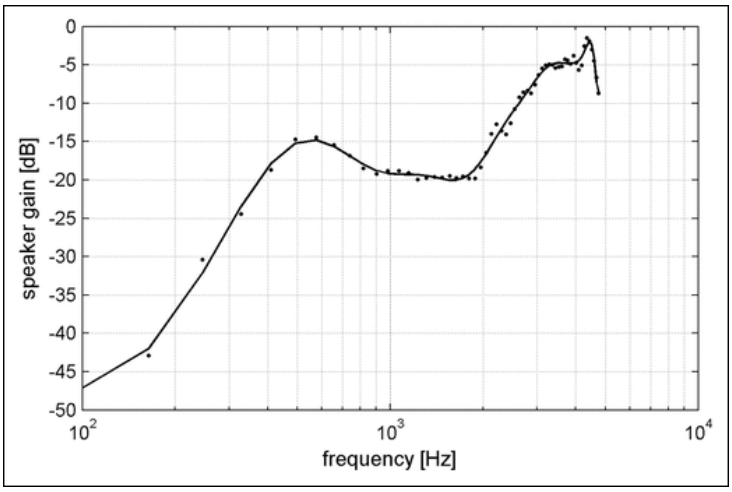


Figure B.3: Frequency response of the speaker mounted on the e-puck. (Nemec et al., 2017, p.4a)

B.1.2 [Hardware] Frequencies emitted: discrepancy and harmonics

B.1.2.1 Frequencies discrepancy: extra louder frequencies

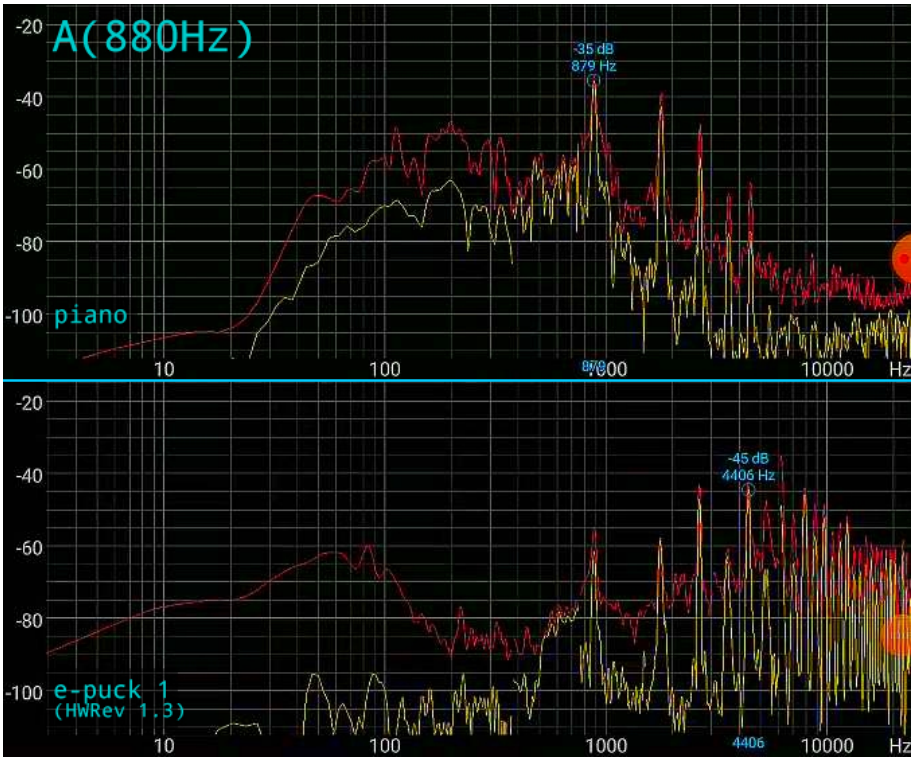
A distinctive feature of the e-puck speaker, for all its versions, is the way it emits sound at frequencies level. Let me explain. Suppose you emit a certain tone at a specific frequency with an e-puck: for instance, you emit some sinusoidal wave at 880 [Hz] thanks to one of the modules we'll introduce below. The tone will sound right to your ear: it will sound like an A 880 [Hz]. However, if you look at what really happens with a spectrometer, you see that, instead of having a higher peak for the frequency supposed to be emitted, you have a multitude of other peaks at volumes as high as or higher than the one of the desired tone. To illustrate this, let's look at the difference between a sound emitted by a piano and the one emitted by an e-puck HWRev 1.3. for an A 880 [Hz] and an A 1760: Figure B.4.

This feature of the e-pucks speaker is problematic if you want to perform experiments using frequency recognition with sound emitted by the robot (typically for inter-robot communication). Indeed, the recognized frequency will most of the time not be the frequency meant to be emitted but one of these other higher peaks. Moreover, the volume levels of these different additional peaks are often close to each other, without one clearly higher than the others. When this is the case, the recognized frequency returned, in addition, will *vary* instead of being unique.

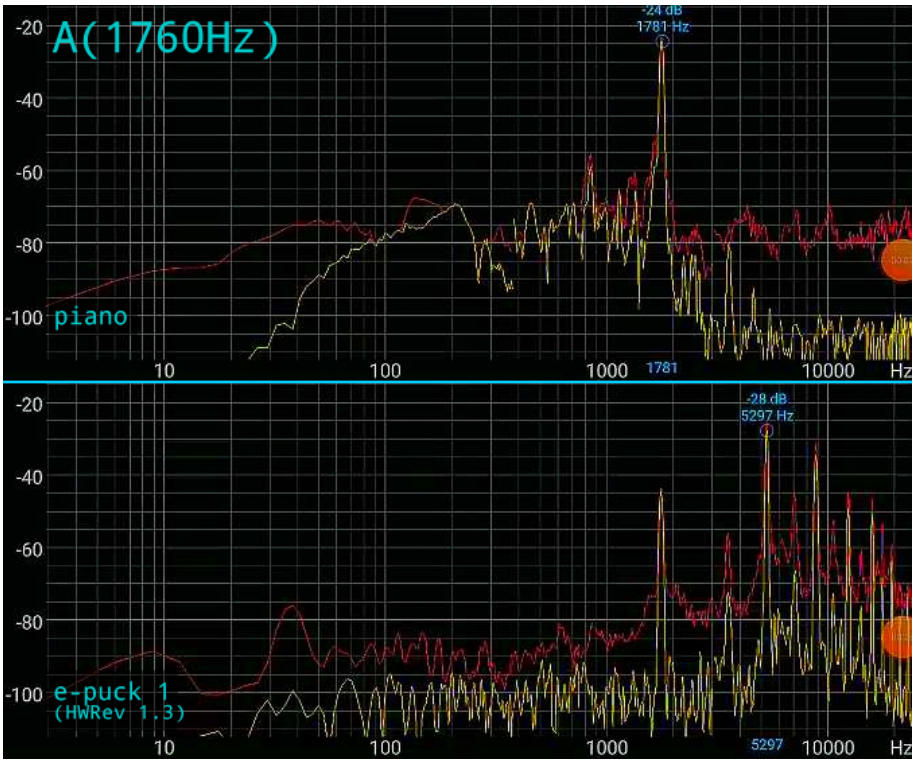
In the case of inter-robot communication between two e-pucks, this problematic aspect will still be amplified by the fact that the sensitivity of the e-pucks microphones increases strongly for frequencies beyond 4000 [Hz] (for more details about this last point, see B.2.1.2 "Volume sensitivity variation relative to frequency; its consequences for frequency recognition: pure vs dirtier sounds")

B.1.2.2 Harmonics (frequencies multiples)

The e-puck's speaker emits other frequencies as loud as or louder than the base frequency, i.e. the frequency meant to be emitted. However, these extra frequencies are neither mere noise nor random additional frequencies. They are in fact harmonics of the base frequency, i.e. multiples of it. The fact that the speaker tends to emit sound louder from 2000 [Hz] ahead seems to explain why the harmonics, that are normally emitted at a lower volume than the base frequency, tend to be emitted at volumes as high or higher than the base frequency. Fortunately, the harmonics thus amplified are not amplified in an exaggerated way. Thus, their volume does



(a) A 880 [Hz]



(b) A 1760 [Hz]

Figure B.4: Audio spectrum comparison of a same tone emitted by a piano and an e-puck (the yellow line shows the current measurement, while the red line describes the maximum values)

not become disproportionately higher than that of the base frequency. Without this last fact, the volume emitted by the robots would become simply unusable for experiments making use of it (This would be especially problematic for the last one of our main projects which uses volume to assess inter-robots distances. see chapter 5)

The fact that the extra frequencies "contained" in a sound emitted by an e-puck are not mere noise but harmonics can be shown with A 880 [Hz] and A 1760 [Hz] using the values of the table Table B.2.

Mult.	Actual Frequencies	Closest Returnable by e-puck	Observed (MICRO_ONLY)
1	**880**	902	902
2	1760	1804	
3	2640	2688	
4	3520	3609	
5	4400	4382	4382
6	5280	5285	
7	6160	6187	6187
8	7040	7089	
9	7920	7992	7992
10	8800	8894	
11	9680	9796	
12	10560	10570	
	"	10699	
13	11440	11472	
14	12320	12375	
15	13200	13277	
16	14080	14179	15082
17	14960	15082	15984
18	15840	15984	
19	16720		
...	...		

Mult.	Actual Frequencies	Closest Returnable by e-puck	Observed (MICRO_ONLY)
1	**1760**	1804	1804
2	3520	3609	
3	5280	5285	5285
4	7040	7089	
5	8800	8894	8894
6	10560	10570	
7	"	10699	10699
8	12320	12375	12375
9	14080	14179	14179
10	15840	15984	15984
11	17600		
12	19360		
	21120		
13	...		
14			
15			
16			
17			
18			
19			
...			

Table B.2: Frequencies returned by an e-puck through FFT
when listening another e-puck emitting an A 880 [Hz] and an A 1760 [Hz]

About Table B.2:

- In column 1, are the real frequencies of the harmonics, i.e. the multiples of the frequency really meant to be emitted by the e-puck speaker.
- In column 2, are the closest frequency values that *can* be returned by an e-puck using the FFT module of the standard library (the FFT module is used with 256 samples in the MICRO-ONLY mode; the usage of the FFT module of the standard library and these "returnable frequencies" will be discussed in section B.2 "How to recognize sounds with e-pucks").
- In the column 3, are the frequencies we observed to be really returned by an e-puck listening to another e-puck emitting an A 880 [Hz] and an A 1760 [Hz]. The listening e-puck uses here too the FFT module of the standard library.

All the values actually returned by the listening e-puck correspond to some harmonic: there are no extra values in addition to the ones written in the third column which are without harmonic correspondence. The fact that the table doesn't include the occurrence rate of the observed values can be misleading. Indeed, 902 [Hz] is observed for an 880 [Hz] emitted, but unfortunately not very frequently or dominantly (*ditto*, *mutatis mutandis*, for 1760 and 1804).

Therefore, we can't just use 902 or 1804 as this would be the case if the other harmonics observed appeared just rarely.

B.1.2.2.1 Sampling rate and the highest emittable frequency. An attentive reader may be surprised to see in Table B.2 some observed frequencies well beyond 4000 [Hz], whereas in the graph of Figure B.3 seen earlier, the sound drops sharply in volume shortly after 4000 [Hz] before disappearing completely shortly before 5000 [Hz].

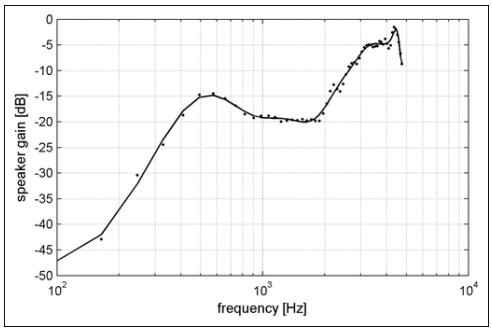


Figure B.3 Frequency response of the speaker mounted on the e-puck.
(Nemec et al., 2017, p.4a)

However, the frequency of Figure B.3 graph is not the frequency *measured*, but the frequency meant to be *emitted* by e-puck speaker. Besides, these phenomena of sound drop and disappearance are not caused by the speaker, but by the sampling frequency chosen to emit the sound. Let me explain. In the case of the graph, the sampling frequency used by Nemec et al. (2017) is 9600 [Hz]. In consequence, the highest frequency that can be emitted is theoretically 4800 [Hz]. It is therefore not surprising that the line of the graph does not go beyond this frequency value. We have to admit that we don't know why the sound volume drops as we approach this limit. In any case, we observed the same phenomenon with another sampling frequency: with the sampling frequency of 7200 [Hz] we use, the theoretical maximum frequency is 3600 [Hz], and the sound emitted by the e-puck drops regularly from 3400 [Hz] onward before disappearing completely just before 3600 [Hz].

B.1.2.3 Practical consequences for frequency recognition: groups of comparison, need for tests and harmonics avoidance

The behavior of the e-puck's speaker has several important practical consequences if it is to be used in experiments involving the frequency of the sound emitted by the robot (typically inter-robot communication):

- In order to identify the frequency of a sound emitted by an e-puck, the frequency measured should not be compared to a single value, but to a *group* of values.
- In order to determine which values should be included in a group for comparison, you are almost forced to test it by observing the values that can be measured since not all harmonics systematically appear.
- If you want clearly identifiable and distinguishable frequencies, you must definitely avoid to choose frequencies whose harmonics can coincide with other frequencies or harmonics of them. (Section "Selection of ID Frequencies" of Nemec et al. (2017) contains a very useful discussion of this last point).

These practical consequences and how they should be taken into account are illustrated in the following video: https://youtu.be/bFydJq7alVA. Let me explain. Following a C scale, a piano and then an e-puck emits the following tones:

Tones	F0	G0	A0	B0	C	D	E	F	G	A	B	C1	D1	E1
Emitted [Hz]	698.46	783.99	880	987.77	1046.5	1174.66	1318.51	1396.91	1567.98	1760	1975.53	2093	2349.32	2637.02
Closest Returnable [Hz]	644	773	902	**1031**	**1031**	1160	1289	1417	1546	1804	1933	2062	2320	2707

Table B.3: Tones emitted by a piano and an e-puck in video https://youtu.be/bFydJq7alVA

"closest returnable" means "closest values returnable by the FFT module of the e-puck. This point is explained in more details below in B.2.3 "[Software] Limited values returnable by FFT". Let's note that the same considerations and practical consequences remain valid, *mutatis mutandis*, if you use a device other than an e-puck to recognize the frequency of a sound emitted by another e-puck

I. https://www.youtube.com/watch?v=bFydJq7alVA&t=3s

In the first part of the video, the listening e-pucks try to recognize the emitted sounds by comparing the value returned to unique frequencies:
the frequency values returnable that are the closest to the one meant to be emitted (2nd row of the table).

Result: while almost all tones are recognized when emitted by the piano, almost no ones are recognized when emitted by an e-puck (HWRev 1.1 and then HWRev 1.3).

II. https://www.youtube.com/watch?v=bFydJq7alVA&t=100s

In the second part of the video, the listening e-pucks compare the frequency returned to groups of values and the results obtained are already clearly better.

III. https://www.youtube.com/watch?v=bFydJq7alVA&t=190s

In the third part, the listening e-pucks compare 5 picked up samples before determining to which frequency group these samples correspond the most. The results become to be really good.

B.1.3 [Software] Producing sound from pre-recorded .wav files

B.1.3.1 The codec module and its default sounds

The standard library of e-puck contains a module dedicated to sound production: the `codec` module which gathers C and Assembler files.
https://github.com/gctronic/e-puck-library/tree/master/library/codec
The core of this module consists of:

- the `e_const_sound.s` file: which contains a series of pre-recorded sounds, and
- the `e_play_sound` function (contained in the `e_sound.c/h` file): which allows to read a part of `e_const_sound.s` by indicating a start line and a duration (specified in number of lines).

(A list of all the files included in the module can be found in Appendix E – an appendix that contains a tree representation of the files of the standard library)

Thanks to this module, it is very easy to produce sound with the e-puck once you've got the right/relevant `e_const_sound.s` file at hand. Let's note that the total duration of the series of sounds contained in `e_const_sound.s` must remain very short: by experience, not more than 2.8 [s]. Beyond this limit, no sound is produced at all.

The default `e_const_sound.s` file supplied with the `codec` module contains a series of five quite specific sounds:

- "haa": somebody hurt by a blow, like a French "ouille";
- "spaah": a banger explosion;
- "ouah": variation of somebody hurt by a blow;
- "yaouh": positive exclamation;
- "wouaaaaaaaah" : a baby crying.

(in quotation marks, the official description of the sound in the comment of the file; after the colon, what they make me think of.)

This series of sounds can be listened to here: https://youtu.be/0a9UhIze3Ko.

The sound samples contained in the default `e_const_sound.s` file have been recorded with a very specific application in mind: a shock and free fall detection program that can be found in the complete demo 1 (selector 0) and in the complete demo 2 (selector 1) (GCtronic (2017), EPFL (2007), list of programs in Appendix D). When the robot's accelerometer detects a shock or a free fall, it plays one of these samples, excepted the "yaouh".

B.1.3.2 Matlab script to produce other sounds (new e_const_sound.s files creation)

The `codec` module from the e-puck standard library enables to produce very easily some sound read from the `e_const_sound.s` file. However, the comments included in the module do not sufficiently explain how to create `e_const_sound.s` files other than the one provided by default. Indeed, the `README.txt` file included in the module evokes a simple Matlab script that could do it, but without supplying this script or giving any further explanation or details about it. This lack of sufficient information can easily leave you in a dead end: you have a module that allows you to play sounds very simply, but you are not able to create these sounds.

A Matlab script found on the net solves this problem and allows to create `e_const_sound.s` files with new sounds (Caprari, 2009). Among others, this script provides some crucial information: the sampling frequency at which the sounds must be resampled in order to be read by the `codec` module: 7200 [Hz]. (If already known, this information can actually be found in the `codec` module. But it is definitely too discreetly mentioned in the `e_common.inc` and `e_init_codec_slave.s` files.)

As this script is hard to find, we added it to the GitHub of this project:
https://github.com/jrlauper/jrl_epuck/blob/master/Matlab_Sound_Script/Sound_Transfo_Caprari.m

A slightly modified version of the script can be found in the GitHub too. It has been modified to make it work with Matlab R2018b, and some comments have been added to make it even easier to use:
https://github.com/jrlauper/jrl_epuck/blob/master/Matlab_Sound_Script/Sound_Transfo_Caprari_JR_mod.m

B.1.3.3 [Software module] Tones library from pre-recorded wav (e_wav_music2.c/h)

- Files location: https://github.com/jrlauper/jrl_epuck/tree/master/jr_wav_music2
- Main files:
  - `e_wav_music2.c/h`
  - modified `e_const_sound.s`

The `codec` module of the standard library of the e-puck enables to easily play some sound from the `e_const_sound.s` file and, in the last section, we have seen how you can create new `e_const_sound.s` file thanks to some Matlab script. In order to be able to easily emit musical notes and small melodies without having to re-record sounds each time, we created a small general library. Its `e_const_sound.s` file is composed of the following 14 musical notes of the C scale (in order of increasing frequency) plus two beeps:

F, G, A(880 [Hz]), B / C, D, E, F, G, A(1760 [Hz]), B / C, D, E

(the two extra beeps have been chosen with frequencies easily recognizable by other e-pucks)

This small library, called `jr_wav_music2`, can be found in the github of this work (link in the frame just above) while the detailed content of its `e_const_sound.s` file can be found here: https://github.com/jrlauper/jrl_epuck/blob/master/jr_wav_music2/Beeps_and_A(880).pdf

This library contains a series of small functions that facilitate the production of the mentioned tones and spare you from having to specify the corresponding lines in `e_const_sound.s` each time you want to play them. Furthermore, the tone emission can be accompanied with the lighting of a corresponding led according to the schema of Figure B.5.

Since the total duration of the sound samples included in `e_const_sound.s` can hardly exceed 2.8 [s], the duration of each recorded sound becomes already very short: 170 [ms] for the 14 notes and 200 [ms] for the 2 beeps. This short total duration makes it difficult to have much more than about 16 small sounds available.

You can easily reuse this small library and its functions by adapting it for other purposes: this is typically what we did on many occasions for inter-robot communication using other

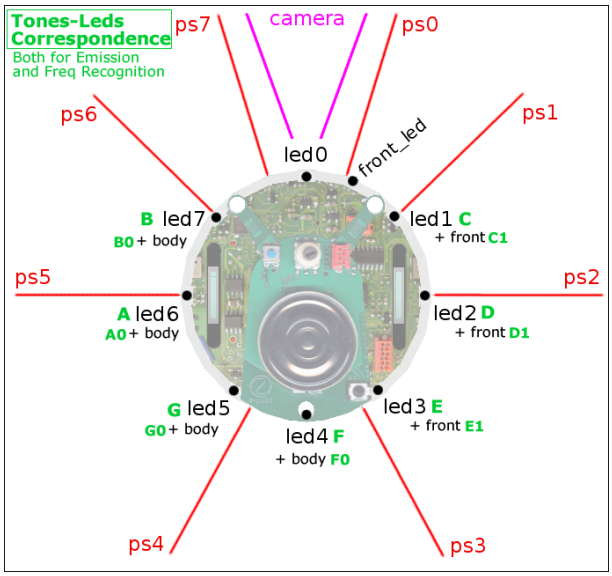


Figure B.5: Tones-leds correspondence - tones emission.
Modified version of a figure from (Cyberbotics Ltd, 2020).

series of sounds. A demonstration video showing an e-puck using this library to play some tunes can be watched here: `https://youtu.be/MA7AYAWtUhg`

B.1.4 [Software module] Producing sound "on the fly" from frequencies (e_freq_sound.c/h)

- Files location: `https://github.com/jrlauper/jrl_epuck/tree/master/jr_freq_sound`
- Main files: `e_freq_sound.c/h`
- Demo video: `https://youtu.be/udtJUNvuzgM`

We created a second library that makes possible to produce sound without having to record any sound sample beforehand. This library allows you to emit a sound of any frequency "on the fly". In addition to the frequency of the sound, you can choose the shape of the wave (sinus, square, triangle, or sawtooth) (Figure B.6) as well as its volume.

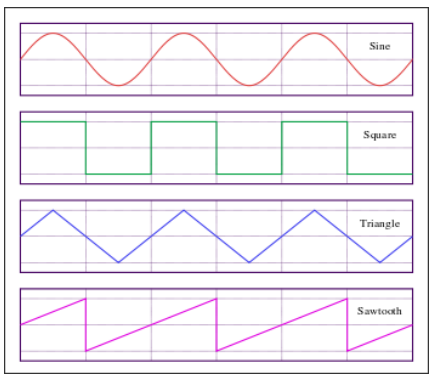


Figure B.6: Wave shapes ("Waveform", Wikipedia 2019c)

The library contains a series of auxiliary functions to facilitate its use. Among these functions, some allow to emit musical notes over many octaves (with half-tones possible), some allow to emit beeps, while still others allow to easily perform some glissendi effects.

This library uses a significantly modified version of the standard `codec` module. In particular the two assembler files `e_isr_dci.s` and `e_sub-dci_kickoffs.s` have been substantially modified and enriched. Consequently, if you use the standard version of the `codec` module, the library stops working. This modified version of the `codec` module comes from the demo *Audio Recording* (GCtronic, 2011), which is mentioned in section 2.7.3. of *E-Puck Main Wiki* (GCtronic, 2019c).

B.2 How to recognize sounds with e-pucks

In this section, we start by presenting the hardware specificities of the microphones equipping the e-puck. We then discuss in detail the working of the FFT module supplied in the standard library – the module enabling to recognize sound frequency –, and the difficulties it raises. Finally, we introduce two little software modules that we created in order to facilitate the use of FFT recognition in experiments.

B.2.1 [Hardware] Micros and sensitivity-frequency relation

The e-puck own three microphones allowing it to record sounds. Their disposition on the robot body is shown on Figure B.7 and Figure B.8.

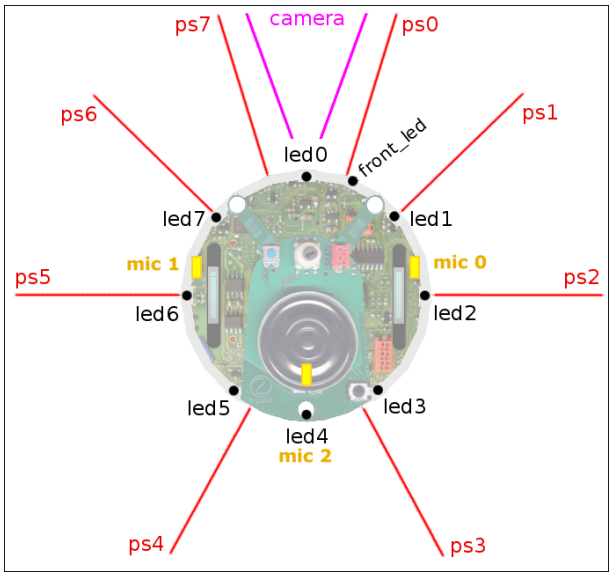


Figure B.7: Micros disposition on the e-puck *with* its platform.
Modified version of a figure from Cyberbotics Ltd (2020)

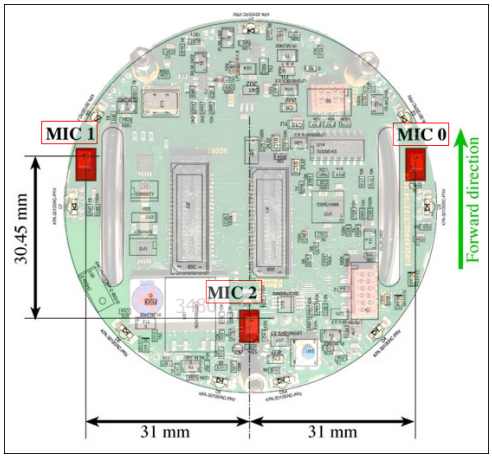


Figure B.8: Micros disposition on the e-puck *without* its platform
. Modified version of http://www.e-puck.org/images/stories/sound/sound.jpg

The location of the rear microphone Mic2 poses a problem for volume perception. It is placed under the small platform supporting the speaker, platform which thus forms a small

resonance box (see Figure B.9). Furthermore, the two large black rectangular connectors linking the body of the e-puck to its platform can, depending on the sound direction, be an obstacle to it or form a small corridor for it. Both phenomena generate errors in the volume measured by Mic2. (Nemec et al. (2017) puts numbers on this effect in the part of the article called "Experiments and results").

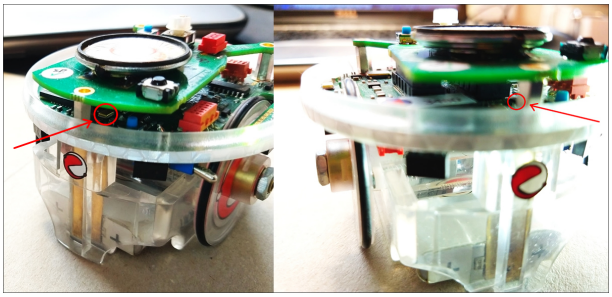


Figure B.9: Position of the rear micro under the platform

B.2.1.1 Microphone difference between e-puck versions

The microphones used in each version of the e-puck are different and their sensitivity vary accordingly (Table B.4). This variation has to be taken into account when using *volume* measurements, especially when using e-pucks of different versions together. Luckily, in spite of different hardware, the microphones of HWRev 1.1 and HWRev 1.2 have very close sensitivity according to the documentation (GCtronic, 2019c).

Version	Production Year	Speaker	Microphone	Version
HWRev 1.1	2006	(opaque black) Produces Louder Sound	Less sensitive (about 15%)	HWRev 1.1
HWRev 1.2	2008	(transparent) Produces Lower Sound		HWRev 1.2
HWRev 1.3	2014		More sensitive	HWRev 1.3

Table B.4: Sound hardware differences between e-puck versions
(*E-Puck Main Wiki*, GCtronic 2019c)

Our experience shows that the different implementations of the microphones can generate variations in the recognized sound frequencies, especially when the sound recorded contains many near-volume frequency peaks. What we observed when testing inter-robot communication using the HWRev 1.1 and HWRev 1.3 e-pucks at our disposal: the majority of the frequencies returned coincide, but some of them only appear with one of the two micros implementations. So, on a practical level, if you want to use both types of robots together, it is wise to test the returned frequencies to see if there are any differences.

B.2.1.2 Volume sensitivity variation relative to frequency; its consequences for frequency recognition: pure vs dirtier sounds

As shown on Figure B.10, e-puck microphones sensitivity varies with the frequency of the sound captured. On it, it can be observed that the volume measured by the microphone starts to take off slowly from 2000 [Hz] before flying away after 4000 [Hz]. If the sound is quite *pure*, i.e. with a single louder peak for the emitted *frequency*, this increase in sensitivity doesn't raise any particular issue for frequency recognition. This is for instance typically the case for a pure sine wave. Figure B.11 shows the spectrograph of a sine wave at 1760 [Hz] generated

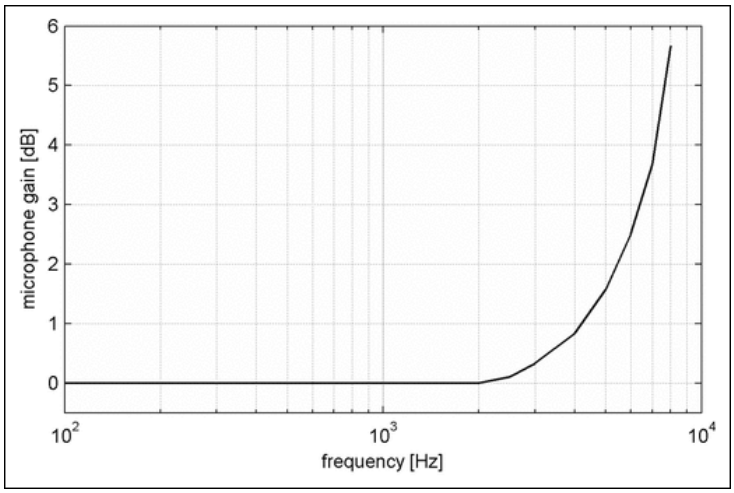


Figure B.10: Frequency response of the micros mounted on the e-puck (HWRev 1.2., micro SPM0208) – (Nemec et al., 2017, p.4a)
Let's note that the Y scale varying from 0 to 6 [dB], and the [B] being a logarithmic scale in base 10, the increase is still in "reasonable" proportions: it is not for instance as strong as the possible variations of the volume emitted by the speakers which went up to 50 [dB] (see B.1.1 [Hardware] The speaker: sound range and volume-frequency relation).

by the software Audacity and emitted through the speakers of a computer. The peak at 1760 [Hz] is clearly louder without any "rivals" and it will be the one grasped as the loudest by the e-puck. Let's add that in the "pure" sound context, *volume measurement* doesn't raise any

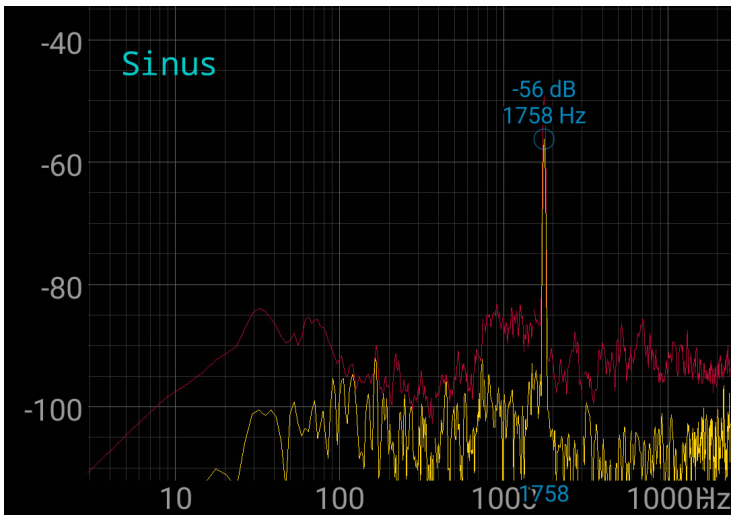


Figure B.11: Spectrograph of a 1760 [Hz] sine wave generated with the software Audacity and emitted through PC speakers (in yellow the current values, in red the max reached values)

particular issue too, as long the frequency of the sound emitted is below 3000 [Hz]. Beyond this limit, volumes measurements comparison becomes problematic if the frequencies of the sounds whose volume is compared are too far apart.

Unfortunately, the sound captured by the microphones can be dirtier: instead of having a single peak of clearly higher volume, the peak corresponding to the base frequency is accompanied by many other peaks with high volumes. This is for instance the case for a square wave, or a sawtooth wave (see Figure B.12). In this kind of situation, the fact that the sensitivity of the e-puck microphone increases much from 4000 [Hz] on can become very problematic for frequency recognition as well as for volume measurement. Indeed, the volume of other

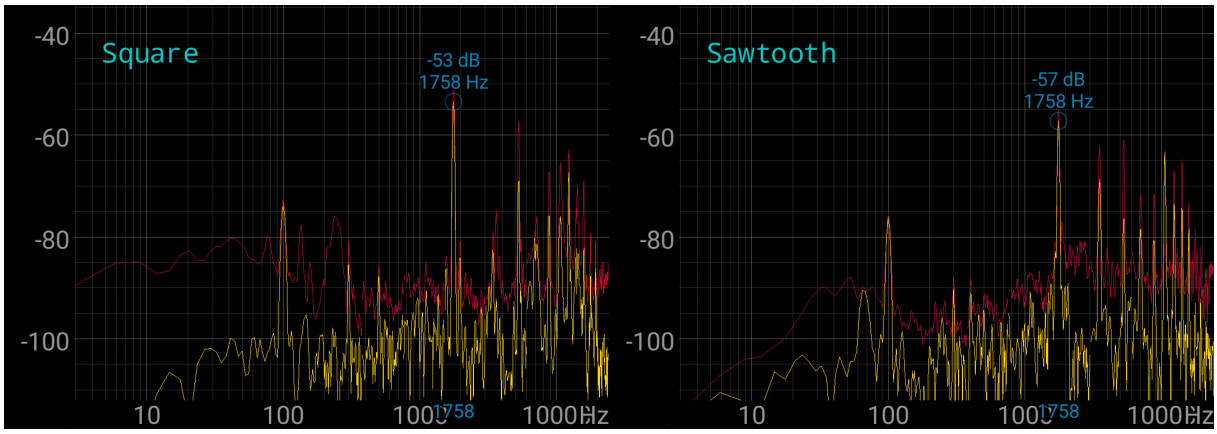


Figure B.12: Spectrograph of 1760 [Hz] square and sawtooth waves generated with the software Audacity and emitted through PC speakers (in yellow the current values, in red the max reached values)

peaks beyond the frequency emitted (in our example, the one at the right of 1760 [Hz]) will be multiplied by the increased sensitivity of the microphones. As a result, it is very likely that the frequency recognized by the e-puck as having the highest volume will not be the base frequency emitted while its measured volume will be very high. Furthermore, this increase in sensitivity can also result in situations within which several frequencies compete for the place of the loudest ones. This generates situations where the frequency recognized as the loudest *varies*.

The same issue can happen with sounds rich in harmonics, i.e. with frequencies multiples of the base frequency, as are the sounds emitted by music instruments. Figure B.13 shows a nice illustration of harmonics with a piano playing an A 880 [Hz].

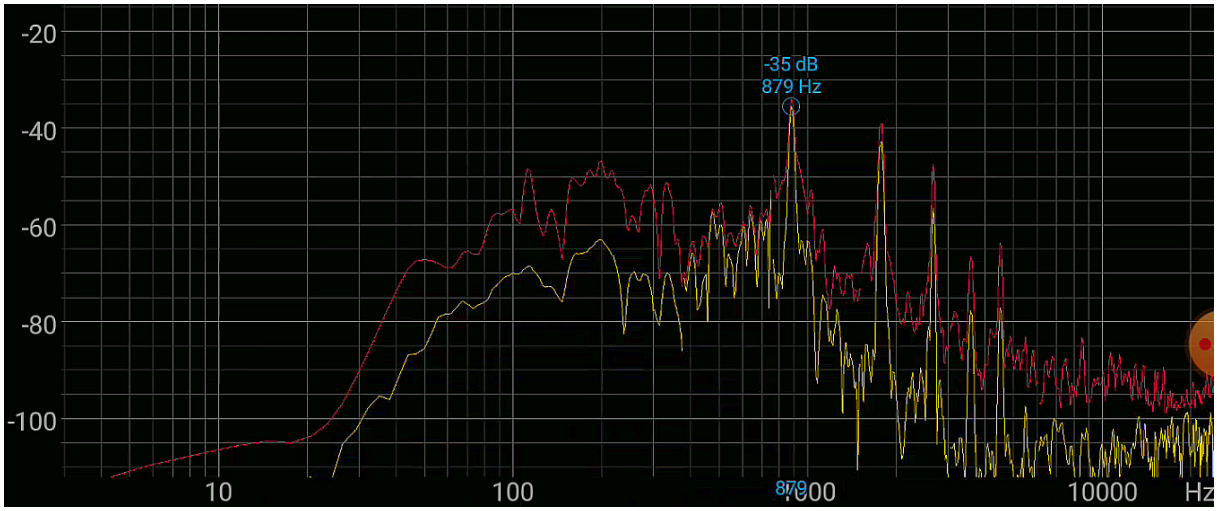


Figure B.13: Spectrograph of an A 880 [Hz] played on a piano showing harmonics.

B.2.1.3 Practical consequences: groups of comparison and need for tests

Although different in origins, the practical consequences of the strong sensitivity increase of the e-puck microphones beyond 4000 [Hz] and the practical consequences generated by e-puck speaker specificity are the same as:

1. When you want to use frequency recognition in experiments using e-puck, very often you need to compare the frequency returned to a *group* of frequencies and not only to the frequency meant to be emitted or captured.

2. To determine which frequencies must be included in comparison groups, you need to make tests.

If you want to use volume in some experiments implying e-pucks, you will have definitely to test and "play" with the fact that the volume can start to be very high for the extra frequencies "accompanying" the base frequency.

B.2.2 [Software] Standard library modules for micros: volume and FFT

The standard library of the e-puck contains two modules giving access to the microphones (for more details on the files included in each module, see the tree representation of the standard library in Appendix E):

- The module `a_d` handles the analog-to-digital conversion.
  It includes the `e_micro.c/h` file which allows access to the *volume* measured by each of the three microphones.

- The FFT module allows to estimate the *frequency* of the sound captured by Mic 0, the right microphone. As its name suggests, this module does this using an FFT (a Fast Fourier Transform algorithm), process rendered possible by the dsp core of the dsPIC microcontroller. Although forming an independent module, the FFT module presupposes the advanced version of the `a_d` module to work.

For the concrete use of the FFT module, the program `runFFTlistener.c/h` written by the very author of the FFT module is especially helpful (demo 2, selector 9, EPFL (2007), see Appendix D). This program allows you to guide the e-puck's movements according to sounds played or whistled. This program served as the basis for the two modules we developed ourselves.

Two important things should be noted about the FFT recognition performed by the e-puck using the FFT module:

1. In our experience, the FFT recognition can only be performed on one of the three microphones at a time, and not on two or all three of them. Furthermore, it is not easy to change the microphone used to perform the FFT recognition (i.e. to use another one than Mic 0).

2. The `a_d` module manages the analog-to-digital conversion in a general way and not only for microphones. This module can be initialized in two ways: `MICRO_ONLY` or `ALL_ADC`. The frequency values returned are not the same depending on the chosen mode. More details on this point is given below in B.2.4 "[Software] FFT and the two analog-to-digital modes: MICRO_ONLY and ALL_ADC".

B.2.3 [Software] Limited values returnable by FFT

The FFT module of the e-puck standard library uses dsPIC-specific operations to perform an FFT (Fast Fourier Transform) on the data recorded by the right microphone (Mic 0) . An FFT is an algorithm for calculating a DFT, a Discrete Fourier Transform[1]. Due to the *discrete*

[1]For more more info about Discrete Fourier Transform, see for example https://math.stackexchange.com/questions/30464/what-is-the-difference-between-the-discrete-fourier-transform-and-the-fast-fouri and the wikipedia pages cited by it.

nature of the DFT, certain frequencies only can be returned by an FFT. The frequencies that can be returned by it depend on F_{mic}, the microphone sample rate, as well as N_{mic}, the number of recorded samples processed by the FFT (cf. Nemec et al., 2017). Their values are given by the following formula:

$$F_{DFT}(k) = \left\{ k \cdot \frac{F_{mic}}{N_{mic}} \right\}_{k=0}^{\frac{N_{mic}}{2}} \tag{B.1}$$

Adapted with small modifications from Nemec et al. (2017).

B.2.3.1 N_{mic}: Recorded samples number

The value of N_{mic} is the number of samples captured and stored in the RAM of the e-puck for FFT processing. In the code, N_{mic} corresponds to the value of MIC_SAMP_NB defined in `e_ad_conv.h` (a file of the `a_d` module, `/a_d/advance_ad_scan/`, see Appendix E). The recommended value that we used is: $N_{mic} = 256$. From experience, using higher values for N_{mic} has two disadvantages:

1. This generates some latency to store the sound within the RAM of the e-puck;
2. This uses a lot of the RAM of the robot. For example
   - For MIC_SAMP_NB = 512: this uses almost 50% of the RAM.
   - For MIC_SAMP_NB = 1024: this uses more than 80% of RAM.

B.2.3.2 F_{mic}: Micro sampling frequency

If we look into the code of `e_ad_conv.c/h` (a file of the `a_d` module, `/a_d/advance_ad_scan/`, see Appendix E), two values could be good candidates for F_{mic}:

- 16'384 [Hz], the value of MIC_SAMP_FREQ in `e_ad_conv.h`;
- 33'000 [Hz], a value mentioned in the comment on `MICRO_ONLY` in `e_ad_conv.c`.

The values returned by the e-puck using the FFT module in the `MICRO_ONLY` mode show that we must take: $F_{mic} = 33'000$ [Hz] (see B.2.4 below for more details about the analog-digital modes). Taking $F_{mic} = 16'384$ [Hz] in the equation above gives values that do not correspond in any way to the measured values, whether in `MICRO_ONLY` or in `ALL_ADC` mode.

B.2.3.3 FFT returnable and observable values

If you take $N_{mic} = 256$ and $F_{mic} = 33'000$ [Hz] in Equation B.1 above, and take the *truncated* result of it, you get exactly the values returnable by an e-puck when carrying out a FFT using the FFT module. The vast majority of these values have been directly checked and confirmed by our own observations. For the first $120k$'s, the values returnable by the e-puck are the ones given in Table B.5. In spite of its dimensions, we give this table here as it is very practical to have it under your eyes when you work with e-pucks and FFT. Besides, we'll use this table repeatedly in the sections to come.

k	Freq	k	Freq	k	Freq	k	Freq
1	128	31	3996	61	7863	91	11730
2	257	32	4125	62	7992	92	11859
3	386	33	4253	63	8121	93	11988
4	515	34	4382	64	8250	94	12117
5	644	35	4511	65	8378	95	12246
6	773	36	4640	66	8507	96	12375
7	902	37	4769	67	8636	97	12503
8	1031	38	4898	68	8765	98	12632
9	1160	39	5027	69	8894	99	12761
10	1289	40	5156	70	9023	100	12890
11	1417	41	5285	71	9152	101	13019
12	1546	42	5414	72	9281	102	13148
13	1675	43	5542	73	9410	103	13277
14	1804	44	5671	74	9539	104	13406
15	1933	45	5800	75	9667	105	13535
16	2062	46	5929	76	9796	106	13664
17	2191	47	6058	77	9925	107	13792
18	2320	48	6187	78	10054	108	13921
19	2449	49	6316	79	10183	109	14050
20	2578	50	6445	80	10312	110	14179
21	2707	51	6574	81	10441	111	14308
22	2835	52	6703	82	10570	112	14437
23	2964	53	6832	83	10699	113	14566
24	3093	54	6960	84	10828	114	14695
25	3222	55	7089	85	10957	115	14824
26	3351	56	7218	86	11085	116	14953
27	3480	57	7347	87	11214	117	15082
28	3609	58	7476	88	11343	118	15210
29	3738	59	7605	89	11472	119	15339
30	3867	60	7734	90	11601	120	15468

Table B.5: Values returnable by the FFT module of the e-puck with N_{mic} = 256 and F_{mic} = 33'000 [Hz].

Apparently, only the *result* of Equation B.1 is truncated, while the intermediary operations are neither truncated nor rounded to the unity. This would explain that the difference between two consecutive values is most of the time 129 and sometimes 128 (roughly every 11 lines), since the exact value of this difference, being equal to F_{mic}/N_{mic}, is exactly 128.90625.

B.2.3.4 Practical consequences: "rounding" issues

The discrete nature of the frequencies recognized by the FFT and the distance separating two measurable values (129, and sometimes 128) have a double impact on the frequency measurements made by the e-puck.

1. The e-puck can return the same frequency via FFT for sounds that are clearly different (e.g. to the ear). This can be seen for example by playing the notes of the C scale below on a piano: the e-puck won't be able to differentiate B0 and C:

	F0	G0	A0	B0	C	D	E	F	G	A	B	C1	D1	E1
Emitted by the piano	698.46	783.99	880	987.77	1046.5	1174.66	1318.51	1396.91	1567.98	1760	1975.53	2093	2349.32	2637.02
Closest returnable by FFT	644	773	902	1031	1031	1160	1289	1417	1546	1804	1933	2062	2320	2707

Table B.6: Closest returnable values by FFT for C scale tones, and how B0 and C are indistinguishable.

2. If the frequency "perceived" by the e-puck is not clearly closer to one of two values that can be recognized, the returned value may vary between both.

In other words, the discrete nature of the FFT means that we are faced with situations similar to rounding problems.

B.2.4 [Software] FFT and the two analog-to-digital modes: MICRO_ONLY and ALL_ADC

The use of the FFT module of the standard library requires the use of the analog-to-digital conversion module `a_d`, more precisely its advanced version: the one you can find in the folder `a_d/advance_ad_scan`. To initialize this module, in the file `e_ad_scan.c` you must use the function `e_init_ad_scan(var)`. This function can be called with two different variables: `MICRO_ONLY` or `ALL_ADC`:

- In the `MICRO_ONLY` mode, as its name indicates, the acquisition of sample values and the analog-to-digital conversion is entirely used by the microphones (with a sampling rate of 33Khz according to the file comment). Within this mode, the other features of the e-puck using the analog-to-digital conversion are unusable. This concerns the infrared proximity sensors and the accelerometer.
- In the `ALL_ADC` mode, samples acquisition and then analog-to-digital conversion are used at the same time for:
  - the microphones,
  - the infrared proximity sensors and
  - the accelerometer.

The fact that the `MICRO_ONLY` mode disables the use of the infrared proximity sensors makes it of rather limited interest. One way to get around this limitation is to alternate the two kinds of initializations – `MICRO_ONLY` and `ALL_ADC` – within the code. This is fully possible and it is working. However, in our experience this has two major drawbacks:

1. this generates some latency in the behavior of the e-puck; and

2. this uses additional power resources – which can generate unwanted resets when e-puck is already quite intensively used.

The code comments and documentation remain silent about what happens to the microphones when they are used in the `ALL_ADC` mode. Our experience shows that they remain fully operational and that the FFT module itself continues to work. However, the values returned by the FFT module no longer seem to correspond to anything real.

Table B.7 illustrates this point for a series of musical notes of the C scale ranging from F 698.46 [Hz] to E 2637.02 [Hz]. The sound is emitted by an electronic piano, and is recorded and processed by an e-puck HWRev 1.3 using the FFT module. The 2nd column contains the values returned in the `MICRO_ONLY` mode, while the 3rd column contains the ones returned in the `ALL_ADC` mode by the same robot.

Actual Frequ (Hz)	Tone	FFT MICRO_ONLY	FFT ALL_ADC		Ratio ALL_ADC/Actual	Ratio ALL_ADC/MICRO_ONLY
698.46	F0	644	1546		2.21	2.40
783.99	G0	773	1804		2.30	2.33
880	A0	902	2062		2.34	2.29
987.77	B0	1031	2320/4511		2.35/4.56	2.25/4.38
1046.5	C	1031	2449		2.34	2.38
1174.66	D	1160	2707		2.30	2.33
1318.51	E	1289	2964/3093		2.25/2.35	2.29/2.39
1396.91	F	1417	3222		2.31	2.27
1567.98	G	1546	3609		2.30	2.33
1760	A	1804	3996/4125		2.27/2.35	2.22/2.29
1975.53	B	1933	4511		2.28	2.33
2093	C1	2062/2191	4769/644/515/386/2062		---	—
2349.32	D1	2320/2449	5414		2.30	—
2637.02	E1	2707	6058		2.30	2.24

Table B.7: Comparison between the FFT values returned in the MICRO_ONLY and ALL_ADC ad modes for C scale tones.

(It is worth noting that the difference separating the discrete values returnable by the FFT keep to be the same in both modes: 129 or 128 (for the reason behind these "returnable values" and this difference, see B.2.3.3 "FFT returnable and observable values"). There is no change between the two modes in this regard.)

As told, the value returned by the FFT with the `ALL_ADC` mode no longer corresponds at all to the actual frequency value of the sound recorded. But nor is it a simple multiple of the actual value: which would be the case if, for instance, instead of the emitted value the microphone grasped some harmonic of it. However, the returned value doesn't seem to be just arbitrary either: for instance, its increase follows the increase of the emitted frequency.

In the last two columns of the table, the value returned by the FFT `ALL_ADC` is compared with the actual value emitted, then with the value returned by the FFT `MICRO_ONLY`. We can see that a fairly constant ratio around 2.3 appears. The fact that we don't have a single value, but several values "around" 2.3 is very likely explained by the discrete and spaced character of the values that can be returned. As we have seen in the previous section, this generates some kind of rounding error.

We haven't found any explanation for this precise figure of 2.3: for example, 2.3 does not correspond to 33'000 [Hz] divided by 16'384 [Hz] – another value given for F_{mic}, the microphone sampling rate, in the code. On the other hand, the fact that the FFT `ALL_ADC` value is

higher than the one returned by FFT `MICRO_ONLY` seems to be explainable in the following way: the sampling of values for the microphones is alternated with the sampling for the other modules, while the calculation for the FFT remains unchanged; this gives the illusion of a wave with a faster than real oscillation.

B.2.4.1 Practical consequences: need for tests again

The FFT module remains fully usable if you use the `ALL_ADC` mode instead of the `MICRO_ONLY` mode for analog-to-digital conversion. This allows us to use analog-to-digital conversion for all features that require it (infrared proximity sensors and accelerometer) and not only for microphones. However, the value returned by the FFT module in the `ALL_ADC` mode no longer corresponds to the actual value that is supposed to be measured. To determine the returned value, you can multiply the actual value by 2.3 and then look at the closest value that can be returned by the FFT (see Table B.5 in previous subsection B.2.3.3).

Because of the discrete values that can be returned, one of the two problems connected to rounding mentioned in subsection B.2.3.3 is again present: the fact that a value may not be clearly closer to one returnable value than to another, resulting in a returned value that oscillates between two values. We see this in Table B.7 above for example: for E 1289 [Hz] and A 1760 [Hz], using `ALL_ADC` returns *two* values instead of only *one* for `MICRO_ONLY`.

The fact that using `ALL_ADC` multiplies the measured value combined with the fact that the returnable values have a constant distance of 129/128 implies that sounds or musical notes that were not distinguishable with `MICRO_ONLY` can become distinguishable with `ALL_ADC` – which seems an advantage. However, this can also be a good reason to have more harmonics measured. Let's finally add that if you want to isolate a certain emitted frequency by comparing it to a group of frequencies, using `ALL_ADC` can be prone to generate more overlaps between comparison groups.

In practice, if you want to use e-puck to recognize a frequency with the `ALL_ADC` mode, the wisest thing to do is to test the frequencies that can be recognized by e-puck in order to determine a comparison group. Given the other problems mentioned for the microphones and speaker of the e-puck, performing such tests remains a general advice for using frequency recognition with the e-puck, and is not specific to the use of if with the `ALL_ADC` mode.

B.2.5 [Software module] FFT basic recognition with leds – runFFTlistener_mod.c/h: run_FFT_listener()

- Files location: https://github.com/jrlauper/jrl_epuck/tree/master/jr_sound_listening
- Main files: `runFFTlistener_mod.c/h`.
- Function: `run_FFT_listener()`.
- Demo videos: `https://youtu.be/516BiB3IN2Q`

This is a first basic module we developed around the FFT module of the standard library. Using an FFT (Fast Fourier Transform), it allows, when recording of a sound,

- to light leds according to the frequency recognized; and/or
- to return the value of the frequency recognized via bluetooth/UART in verbose mode.

The module was created from `runFFTlistener.c/h`, the program 9 of the full demo 2 (EPFL (2007), see Appendix D). The lighting of the leds wrt the tone recognized follows the same

tones-leds correspondence we used for tones emission in B.1.3.3 "[Software module] Tones library from pre-recorded wav (e_wav_music2.c/h)":

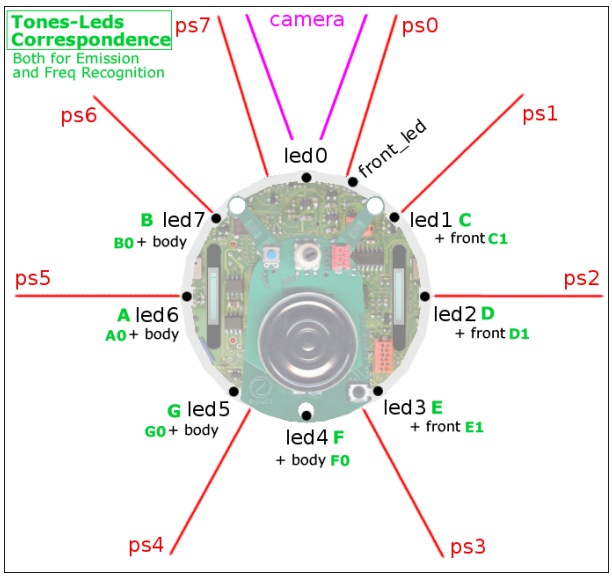


Figure B.14: Tones-leds correspondence - FFT recognition.
Modified version of a figure from (Cyberbotics Ltd, 2020).
Identical to Figure B.5.

The values used for the recognition are the ones of the 2nd row of this table:

	F0	G0	A0	B0	C	D	E	F	G	A	B	C1	D1	E1
Emitted by the piano	698.46	783.99	880	987.77	1046.5	1174.66	1318.51	1396.91	1567.98	1760	1975.53	2093	2349.32	2637.02
Closest returnable by FFT	644	773	902	**1031**	**1031**	1160	1289	1417	1546	1804	1933	2062	2320	2707

Table B.8: Frequency values used for FFT tones recognition with piano and whistling (about the 2nd row and the closest returnable values by FFT, see above, B.2.3 "[Software] Limited values returnable by FFT")

The following video illustrates how the module recognizes different music notes emitted by a piano or whistled: https://youtu.be/516BiB3IN2Q (Let's note that the robots cannot distinguish between B0 and C given the limited number of values that can be returned by the FFT).

In its current version, the module uses the analog-to-digital mode `MICRO_ONLY`. But it can easily be adapted to use `ALL_ADC` (see B.2.4 on `ALL_ADC` and `MICRO_ONLY`). The module can also be easily modified to recognize other tones and frequencies – what we did for robots communication (see below B.3 Sound frequency communication between e-pucks) .

B.2.6 [Software module] General purpose FFT library (e_freq_recognition.c/h)

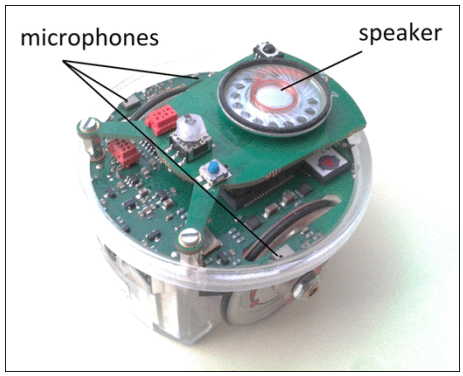
The second module we developed around the FFT module of the standard library is a small software brick that can be reused as such in other programs. Again, this is primarily a generalization of the `runFFTlistener.c/h` program (program 9 of the full demo 2 (EPFL, 2007), Appendix D).

The module can be found here:

- Files location: https://github.com/jrlauper/jrl_epuck/tree/master/jr_demo
- Main file: `e_freq_recognition.c/h`

The default analog-to-digital mode used is `MICRO_ONLY`, but it can easily be changed. Once a minimum and maximum volume threshold set, the module returns the frequency of any sound captured by the right microphone (Mic 0), as long as its volume is within their limits. Moreover, regarding the volume, the module takes into account the difference in microphone sensitivity between the e-puck HWRev 1.1 and HWRev 1.3 versions.

B.3 Sound frequency communication between e-pucks



(Nemec et al., 2017)

In section B.1, we describe how the e-puck and its speaker can be used to produce sounds of specific frequencies. In part B.2, we describe how the e-puck and its microphones can be used to recognize the frequency of a sound. In each of these parts, we discuss the particular challenges and difficulties that each of these operations raises for the e-puck. In this section, we describe how two e-pucks can communicate by combining these two operations.

When we think of two robots using sound frequencies to communicate, we imagine the following situation: a first robot emits a sound of a certain frequency; a second robot picks up this sound wave, processes it, and recognizes its frequency. This ideal situation is unfortunately not possible with e-pucks. Instead, the e-pucks can only transmit and pick up *groups* of frequencies in a particular way. This makes communication between e-pucks via frequency transmission much more complicated than in the ideal situation, although it is still possible. In this section, we briefly discuss the causes of this situation before presenting a software implementation that makes such frequencies group transmission possible in a satisfactory manner.

B.3.1 E-puck sound emission in brief

section B.1 describes in detail the particularities of the e-puck speaker in terms of the emission of a certain sound frequency. The special features to be retained in the context of robots communication are the following ones:

(E1) Range of sounds emitted. The e-puck can emit sounds ranging from 300 [Hz] to 3600 [Hz]. Below 300 [Hz] the volume drops sharply. Above 3600 [Hz] no sound is emitted for "theoretical" reasons, as the sampling rate is set at 7200 [Hz]. On a hardware level, however, nothing prevents the speaker from emitting sounds with much higher frequencies.

(E2) Quality of transmitted frequencies – non-pure sound. When the e-puck is asked to emit a certain sound frequency, the speaker does not emit "pure" sound with a clearly louder peak corresponding to the frequency that is supposed to be emitted. Instead, the peak corresponding to the intended frequency is accompanied by multiple other high volume peaks (sometimes as much or more high). These peaks apparently correspond to multiples of the basic sound (harmonics). For the sake of clarity and simplicity, we will call in the rest of this section:

- *base frequency*: the frequency supposed to be emitted by the e-puck; and
- *accompanying frequencies*: the frequencies that are emitted at the same time and at close volumes by the speaker.

(E2) is particularly problematic for transmitting information via the transmission of a sound wave of a specific frequency. This is simply not possible with the e-puck. Instead, although the e-puck is asked to emit a specific frequency, its speaker emits a group of frequencies, with the result that, on the listener's side, you have to recognise not a specific frequency, but a group of frequencies.

B.3.2 E-puck sound recognition in brief

All the parts of the system that allow e-puck to return the frequency of a sound are described in detail in section B.2. These parts are the three microphones of the e-puck and the dsp core of its microcontroller. The particularities of this system relevant for robots communication are the following:

(L1) The microphones. At hardware level, the microphones become very sensitive for frequencies above 4000 [Hz]. This reinforces the impact of the characteristic (E2) above: the fact that the frequency supposed to be emitted is accompanied by other frequencies. Indeed, among these, the frequencies above 4000 [Hz] will be perceived much more strongly by the microphones and generally recognized in priority over the basic frequency.

(L2) Rounding issue. The e-puck returns the frequency of a sound wave by performing a FFT (Fast Fourier Transform). The frequency values that can be returned by the FFT module are *discrete*: they depend on the number of samples recorded and the sampling frequency of the microphones. In the most common setup, the distance between two values that can be returned by the FFT module is 129 or 128 [Hz]. This results in a high rounding error and the possibility of not being able to distinguish at measurement level waves that are yet different.

(L3) Analog-digital conversion mode (`MICRO_ONLY` vs `ALL_ADC`). Depending on the analog-digital mode selected, the returned/measured values no longer correspond at all to the actual values emitted by the speaker. However, they can still be used, e.g. they continue to be higher or lower depending on how high or low the recorded frequency is.

B.3.3 Example combining emission and recognition by e-pucks

When you want to make two e-pucks communicate using a sound frequency, the characteristics and problems of both aspects – emitting and listening – will combine. The most problematic aspect is (E2), the fact that the speaker emits accompanying frequencies in addition to the base frequency. The simplest way to illustrate the type of problem encountered is to consider the following concrete example:

> We make an e-puck emit a sinusoidal sound with a frequency of 3072 [Hz]: this is the beep we use in many experiments. In the analog-digital `ALL_ADC` mode, the frequencies recognized by another e-puck listening to this beep are:
>
> 2191, 2320, 2449, 7089, 7218, 7347, 8378, 10441, 10570, 11730, 11859, 15339.
>
> Moreover, among these frequencies, there is not one that is clearly always there in every case. Thus, to be able recognize a beep emitted by an e-puck at 3072 [Hz], you have to compare the captured sound to these 12 values!

For one frequency to recognize it's a bit cumbersome, but it's still manageable. However, things get quite tough if you want to transmit not just *one* but *several* frequencies. Indeed, (E2), the fact that each base frequency comes out "accompanied" by other frequencies, added

to (L2), the limited accuracy of the values that can be returned by the FFT module, makes overlaps between the sets of frequencies recognized for the different transmitted frequencies very likely.[2]

B.3.4 Managing e-puck sound communication issues: a promising start with 12 distinguishable beeps

- Emitter:
  - Files location: `https://github.com/jrlauper/jrl_epuck/tree/master/jr_music_comm`
  - Files:
    - `e_music_comm.c/h` and
    - `HighFrequ7_mod.s` to use, renamed, instead of `e_const_sound.s` (can found in the `e_const_sound.s alternatives` folder)-
- Listener:
  - Files location: `https://github.com/jrlauper/jrl_epuck/tree/master/jr_sound_listening`
  - Files: `runFFTlistener_mod.c/h`
  - Function: `run_FFT_listener_comm_stat()` with `init_comm_freq_HF7_stat()`
- Frequencies table used (for emission and FFT recognition): `https://github.com/jrlauper/jrl_epuck/blob/master/jr_music_comm/e_const_sound.s%20alternatives/High%20Frequencies%20Attempt%207.pdf`
- Demo video: `https://youtu.be/w7GG8Ibt3go`.

On a practical level, however, the problem posed by the speaker sound combined with the limited accuracy of the FFT module can be fairly effectively circumvented by combining two approaches:

1. By numerous tests, you look for base frequencies that produce groups of frequencies to be recognized with as few overlaps as possible.
2. On the listener robot side, you seek to determine the group to which the captured frequency belongs by comparing not one but several successive samples.

By adopting such an approach, we have achieved good results: we can transmit and identify with a low error rate more than 12 sounds while being able to mix e-puck HWRev 1.1 and 1.3 (in fact, 14 sounds in total: but the two with the highest frequencies are starting to have a volume that is too low because their frequency is too close to 3600 [Hz] – the theoretical upper limit of emission.) The video mentioned in the frame above shows these results. It can be observed that transmission can be made over long distances without any problem (35 [cm], 100 [cm] and 150 [cm]), although some transmission errors appear with increasing distance.

The system should be further improved, but it is clearly promising. The total number of base frequencies, however, will still be quite limited due to the combined effects of :

[2] Nemec et al. (2017) proposes a way to calculate the frequencies to be emitted to avoid confusion. However, the paper does not address or take into account the specific problem raised by the e-puck speaker that emits a lot of extra frequencies.

- the limited range of base frequencies an e-puck can emit: (300-3600 [Hz]);
- the possible overlaps between the *groups* of frequencies emitted by the e-puck (the frequency meant to be emitted being always "accompanied" by other frequencies); and
- the *rounding* issue affecting the frequency values returnable by the e-puck's FFT recognition module.

B.4 E-puck distance assessment through volume

In the other sections of this chapter, we explored the sound capabilities of the e-puck with a strong emphasis on frequency recognition. But a sound wave contains another piece of information: its amplitude; in other words, its volume. The volume can be used in two different ways: to estimate the distance between an emitter and a listener, and/or to estimate the direction in which an emitter is located. In this section, we are only interested in distance estimation. Furthermore, we do not do this in general, but only in the special case in which emitter and receiver are both e-pucks lying on a horizontal plane: the speakers and microphones are thus of a special type with special relative positions to each other.

Other previous sections of this chapter contain elements relevant to experiments with e-pucks using volume:

- B.1.1 explains how the volume emitted by the e-puck's speaker can vary according to the sound frequency emitted;
- B.1.4 introduces a software module that allows you to choose the volume at which a certain sound wave is emitted; and finally
- B.2.2 presents the main software modules of the standard e-puck library for microphones: the module for recognizing the frequency of a wave (by performing a FFT) and the module for measuring volume using the three microphones.

In this section, we explore other aspects relative to sound volume:

- B.4.1 shows how the volume of the sound emitted by the e-puck's speaker varies according to the robot's orientation;
- B.4.2 explores how the volume of a sound captured by the e-puck's three microphones varies with its orientation; Finally,
- In B.4.4, combining the observations from B.4.1 and B.4.2, we examine the limits within which the sound volume can be used to estimate the distance between two e- pucks.

Like with sound frequency recognition, there is a fairly large discrepancy between the "ideal situation" and the e-puck reality when observing how the volume varies with distance. In the ideal situation, due to the attenuation phenomenon and quite trivially: the lower the recorded volume, the farther the emitter is from the listener; and conversely, the higher the volume, the closer the emitter. In the e-pucks case, by contrast, the recorded volume can vary greatly depending on the orientation of the emitter and the listener, while the distance between them remains unchanged. Due to variations in the orientation of the robots, the volume can even decrease slightly, while the distance between robots diminishes; or increase slightly, while the distance between robots grows. This makes the distance evaluation through volume possible, but rather imprecise - to say the least. The description of these phenomena is the main purpose of this section.

We use distance estimation via volume in the third of our key projects: 5 Decentralized swarm coherence using beep sounds. This is what motivated us to observe e-pucks behavior in this respect. In the coherence algorithm used, a robot's behavior depends on the number of other robots found within a given distance-radius around it. Hence, the necessity of being able to estimate the distance of the other robots, e.g. by the volume of a sound emitted by them.

We remind that there are important differences between the three hardware versions of the e-puck in terms of microphones and speakers. Table B.9 reminds them. In the experiments carried for this section, we only used one version of e-pucks – HWRev 1.3 – to avoid additional variations besides those we want to observe. With the other e-puck versions, the data obtained

would probably be a bit different, but our concrete experience with the robots allows us to confidently assume that the observed phenomena would remain – this should, however, be tested and confirmed in another work.

Version	Production Year	Speaker	Microphone	Version
HWRev 1.1	2006	(opaque black) Produces **Louder** Sound	Less sensitive (about 15%)	HWRev 1.1
HWRev 1.2	2008	(transparent) Produces **Lower** Sound		HWRev 1.2
HWRev 1.3	2014		More sensitive	HWRev 1.3

Table B.9: Sound hardware differences between e-puck versions
(*E-Puck Main Wiki*, GCtronic 2019c)

Experiments and measurements presented in this section and their limitations

In this section, we illustrate several phenomena by presenting measurements made with only some of our laboratory robots. These measurements have been made and are proposed only for observational purposes and to allow a general grasp of the discussed phenomena – phenomena we observed and confirmed many times in practice with other robots. If our goal was to obtain precise values about these phenomena, it is obvious that the measurements discussed would be insufficient and would have to be supplemented. In particular, by many other measurements carried out while varying the robots used and error calculations.
A large part of our measurements was performed with the same pair of robots (emitter: 3435, listener: 3480) to focus on the variations generated by the differences in situations without adding the variations due to the differences in robots. We performed all measurements by initializing the analog-digital module in the ALL_ADC mode (see B.2.4 for more details on this subject). When a sound volume is measured, the sound is first recognized via FFT through the right microphone: this can lead to small differences compared to a situation where only the volume is measured without prior FFT recognition. In all the cases mentioned, the emitting robot emits a continuous beep at 3072 [Hz] using the software module `e_freq_sound.c/h` (cf. B.1.4) with a volume of 80. About this last figure, let's note that the value of the volume emitted does not use the same scale as the volume perceived.

B.4.1 Speaker orientation and volume variation – the emitter

B.4.1.1 Observation

The volume perceived by the microphones of a fixed e-puck listening to the sound emitted by another e-puck varies greatly depending on the emitting robot's orientation. Admittedly, the speaker is not placed in the center of the e-puck – the speaker and robot centers being about 1.2 [cm] apart. However, the measured volume variations are surprisingly strong. We observed this by conducting the following experiment:

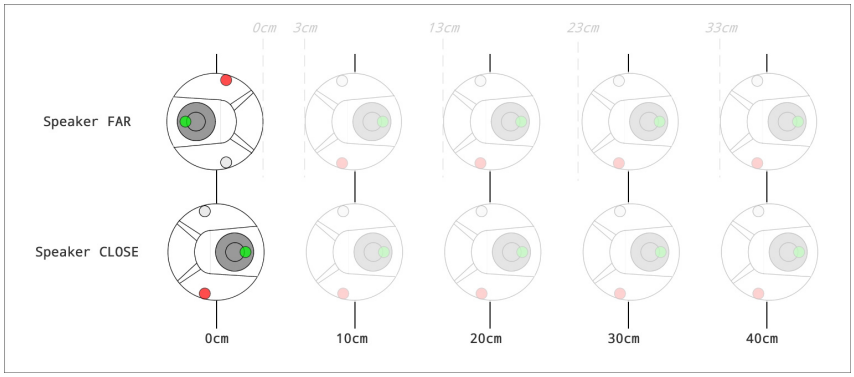


Figure B.15: The different positions of two e-pucks used to measure the effect of the speaker orientation ("close" vs. "far") on the perceived volume.

Close/far speaker orientation experiment (Figure B.15):

- The e-puck emitting the sound, on the far left of the diagram ("0cm" mark), is kept at the same place during all the observations. (As reminder, the emitted sound is a continuous beep of 3072 [Hz] with a volume of 80 on a e-puck HWRev 1.3)
- The e-puck listening to the sound, transparent on the diagram, is placed successively at 10, 20, 30 and 40 [cm] center-to-center distances. The bumper-to-bumper distance is indicated in transparent grey at the top of the diagram: 0, 3, 13, 23 and 33 [cm].
- The emitting e-puck is successively placed in two different orientations:
  - once its speaker is away from the listening robot: henceforth, the "far" orientation;
  - once it is close: the "close" orientation.

  These two orientations correspond to the two extremes in terms of distance.
- The robot listening to the sound, as for it, always keeps the same orientation during all the experiment, the one indicated by the diagram.

We conducted this experiment with two robots from our lab, averaging the values measured by the three microphones of the listening robot. The graph of Figure B.16 shows the results we obtained. These results are surprising:

(1) For a given distance between two robots, there is usually a big difference between the average volume recorded in the "close" orientation and the one recorded in the "far" orientation;

(2) the average volume recorded in the "far" orientation for a given distance d can be very close to the average volume recorded in the "close" orientation for the distance $d + 10$ [cm]. Consequently, and to make things a bit simpler: a measurement of the average volume can only indicate the distance between two robots up to 10 [cm]. Should

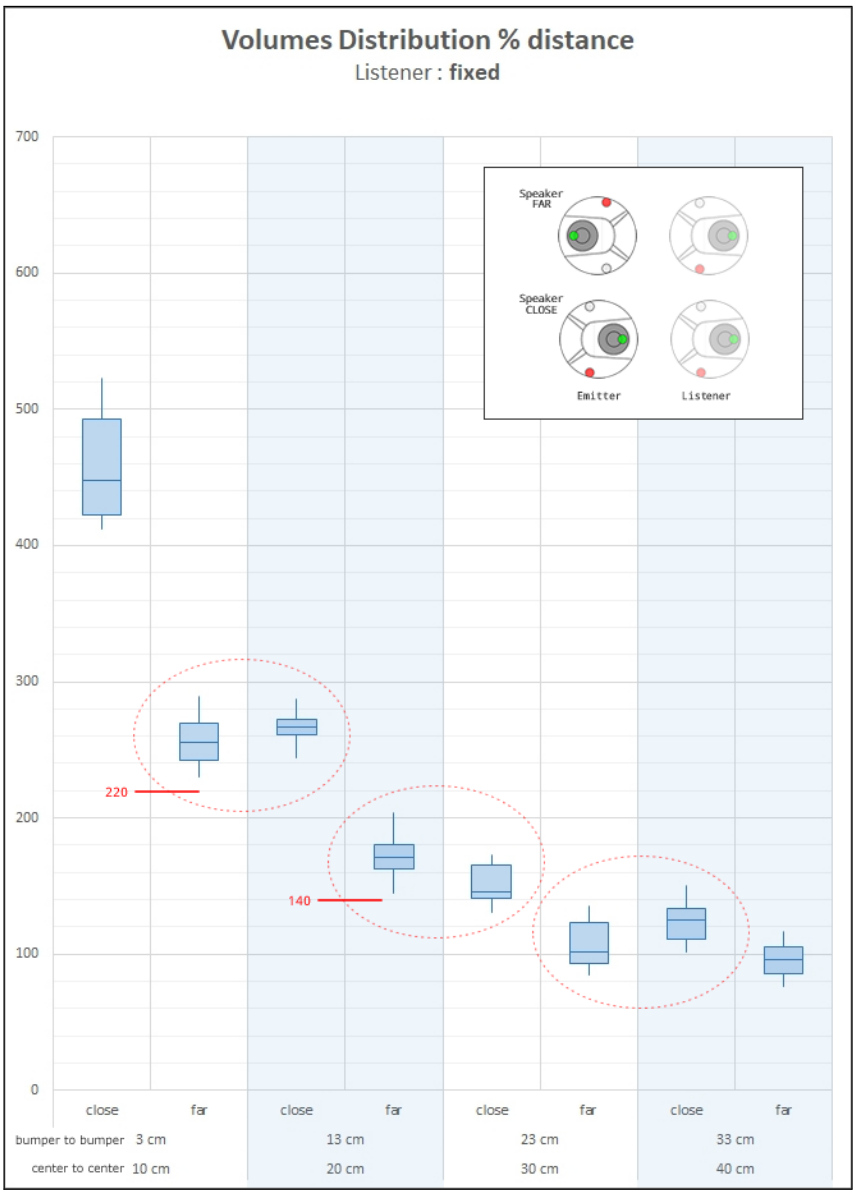


Figure B.16: Measurements of the (average) perceived volume wrt the distance separating two e-pucks of our lab (3435, 3480). The orientation of the emitting robot's speaker varies between the two extremes "close"/"far". The recording robot stays in the same orientation during all the measurements (indicated by "listener: fixed" at the top). Between 150 and 500 measurements were taken for each box plot. The red circles emphasize the problematic similarity of volumes measured between the "close" and "far" orientations for distances however separated by 10 [cm].

The two red lines, at 220 and 140, are the two thresholds used in the illustration of Figure B.17.

we make more measurements with more different distances, this would probably be more than 10 [cm].

B.4.1.2 A too imprecise estimate

If you want to estimate the distance between two robots using the sound volume, the point (2) just above is as problematic as surprising. In order to have a good grasp of the problem raised, we can imagine the following situation:

- We have two e-pucks: an emitter and a listener;
- We'd like the listener to turn on all its leds when it's within 20 [cm] (center-to-center) of the emitter and turn them off when it's out of this radius.

The situation described in the measurements graph of Figure B.16 makes this task unworkable. Indeed, let's first suppose that we set a volume threshold at 220: that is to say that if the measured volume is higher than 220, the leds light up, whereas if it is lower, they light off.Let's also assume that the listener is actually between 10 and 20 [cm] (robots center to center). If the orientation of the emitter is "close", then the listener will light up. But if the emitter is simply rotated 180°, its orientation being "far", the listener will be off (the left part of Figure B.17 illustrates this situation). If, instead of 220, the volume threshold is set at 140, then the listener will always light up between 10 and 20 [cm], but it will also sometimes turn on between 20 and 30 [cm], contrary to what is wanted (right part of Figure B.17).

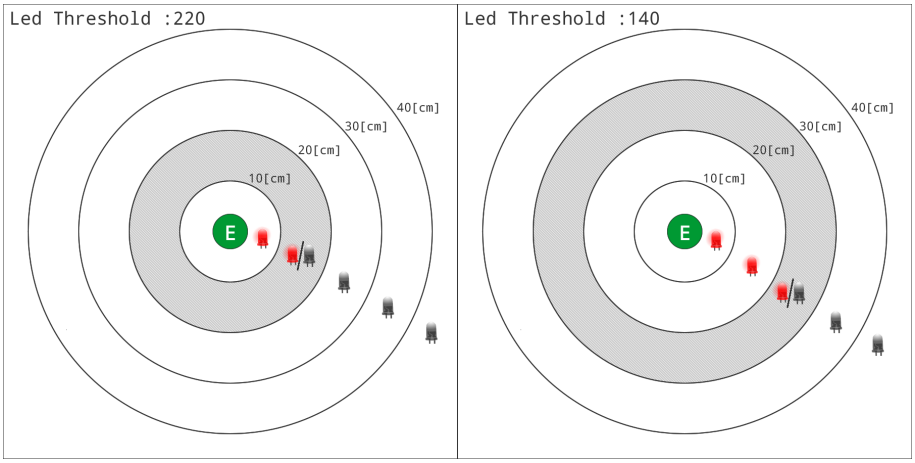


Figure B.17: Large volume variations due to speaker orientation make distance evaluation problematic. Illustration with leds lighting (for two volume thresholds)

The green "E" represents the emitting robot. The leds light up if the perceived volume is above a certain threshold. The substantial variations in volume that appear depending on the orientation of the e-puck's speaker (see Figure B.16 graph) mean that, regardless of the chosen volume threshold, there are "gray" zones: zones within which, although the distance doesn't change, leds may or may not light up depending on the speaker orientation only.

Given the swarm coherence experiment we wanted to develop in chapter 5, the experimental situations that interested us the most were situations where e-pucks are in motion. In these cases, it cannot be assumed that the emitter is always in the same orientation with respect to the listening robot. Thus, the observed phenomenon, the fact that the measured volume may correspond to a certain distance only to the nearest 10 [cm] due to the emitter, is especially problematic.

B.4.1.3 Extra sound reflective equipment

The difference in speaker position between the two "near" and "far" orientations is about 2.4 [cm] – with the speaker's center being about 1.2 [cm] away from the robot's center. This difference is quite large in a relative way when the center-to-center distance between two e-pucks is 10 [cm]. But how can this still make such a big difference when the distance between the robots is 20 or 30 [cm]?

One hypothesis is that the problem comes from the speaker directed upwards. Indeed, because of this, a large part of the sound emitted goes "in the air" instead of going horizontally towards the second e-puck that measures it. In fact, as soon as you place an object or just your hand a few centimeters above the emitting speaker, the sound is partially reflected downwards and the volume measured by the second e-puck is immediately much higher.

In order to test this hypothesis, we slightly modified the emitting e-puck as follows: at about 2.5 [cm] above its speaker, we mounted a small cardboard disc with a diameter of 6.2 [cm]. 6.2 [cm] corresponds approximately to the diameter of the inside of the e-puck bumper. The disc is arranged as follows: its center is placed above the center of the speaker, which results in a small offset to the back of the robot body. Figure B.18 shows what the resulting small setup looks like.

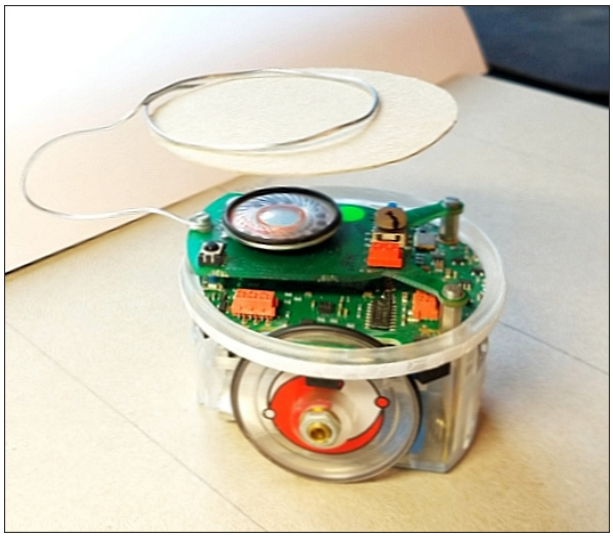


Figure B.18: A small extra cardboard disc is attached above the e-puck to improve sound reflection.

If we repeat the volume measurements made before for Figure B.16 by adding this little cardboard disc, we get Figure B.19. We have then what we wished: the "far"and "close" speaker's orientation no longer significantly impact the volume measured for the same distance. Consequently, we can for instance set a volume threshold around 180 to make sure that the robots are within 20 [cm] center-center, or a threshold around 400, to keep them even closer. We don't have any gray area anymore, where the volume perceived significantly varies depending on the emitter orientation. In a general way, the measured volume is higher – which is clearly what was expected as the cardboard disk partly reflects the sound. The use of this small supplement seems therefore very promising.

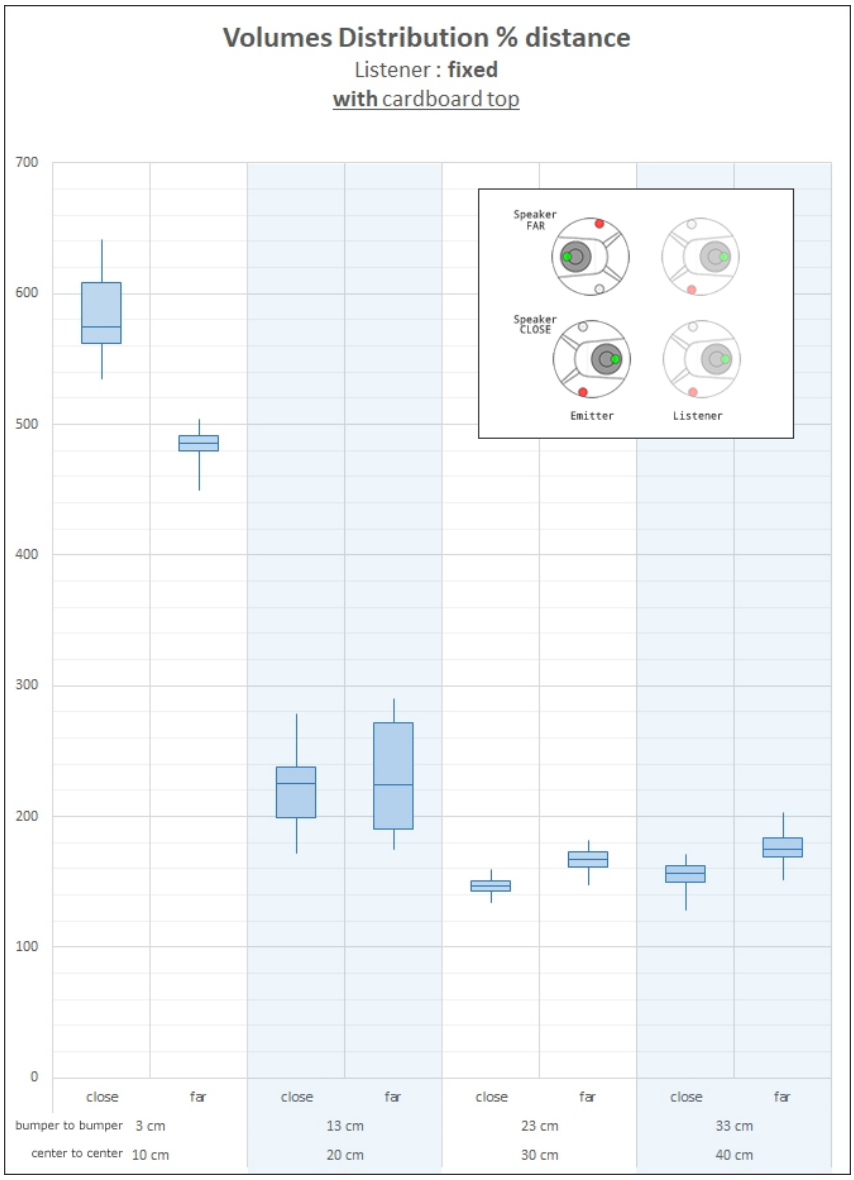


Figure B.19: Measurements of the (average) perceived volume wrt the distance separating two e-pucks of our lab (3435, 3480) if we add a reflexive cardboard disc above the emitting speaker. The orientation of the emitting robot's speaker varies between the two extremes "close"/"far". The recording robot stays in the same orientation during all the measurements (indicated by "listener: fixed" at the top). Between 150 and 500 measurements were taken for each box plot.

In contrast to what we had before without the addition of cardboard (Figure B.16), the volume measurements for the same distance are close and there are no more strong overlaps in the first 30cm.

B.4.2 Microphones orientation and volume variation – the listener

Our goal was to use the sound capabilities of the e-puck to estimate the distance between two robots. In section B.4.1, "Speaker orientation and volume variation – the emitter", we saw how the speaker's orientation of the *emitting* robot can cause strong variations in the measured volume. In this section, we explore how the measured volume varies with the orientation of the *listening* robot. As we will see, the result is similar: we measure large volume variations depending on the listening robot's orientation – at the level of each of the three individual microphones as well as at the level of their average.

Figure B.20 reminds how the three microphones are arranged on the e-puck body. On it, we introduce the color code we will use henceforth in the measurement graphs of the section: [left/red], [right/gray] and [rear/green]. Let's recall that, as shown on Figure B.21, the rear

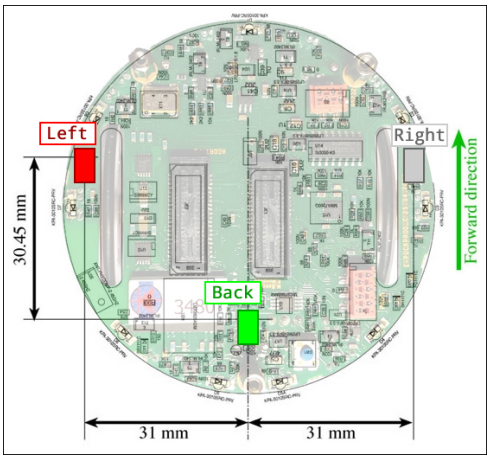


Figure B.20: Micros disposition on the e-puck *without* platform and their color code for the graphs to come: [left–red], [right–gray] and [rear–green]
. Modified version of http://www.e-puck.org/images/stories/sound/sound.jpg

microphone is located

- under the small platform that supports the speaker, which makes it a resonance chamber, and
- behind the two large black rectangular connectors that can constitute, depending on the case, an obstacle or a corridor for the sound.

Two facts that affect the way the sound reaches the microphones and the perceived volume in a way that is difficult to predict.

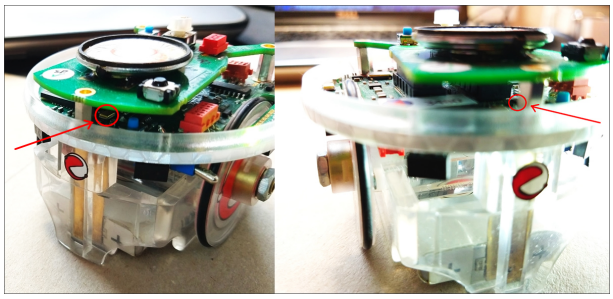


Figure B.21: The rear microphone placed under the speaker platform and behind the two large rectangular connectors.

B.4.2.1 Microphones calibration

Before measuring anything with the microphones of the e-puck, you need to calibrate them. To see it, you just need to have look at Figure B.22. For four different e-pucks placed in a situation of silence, its graphs represents the volume values returned for each of the three microphones.

Figure B.22: Volume returned by each of the 3 micros for 4 different e-pucks in a situation of silence. The dotted lines are averages.

On the four graphs, significant differences in each microphone's average can be observed: both for the same robot and from one robot to another. We can also observe that, for each of the four robots, the average value of the right microphone is higher, while its individual values vary much more than those of the other two microphones. We do not know the reason of this behavior. However, for these calibration measurements, the FFT module that employs the right microphone is not used. Therefore, it can't be the reason of this difference in behavior.

Due to the large differences in the observed inter- and intra-robot averages, any work with the volume of the e-puck microphones should begin with a calibration. By experience, once calibrated, the volume measured in silence remains quite stable. All the measurements made for this section B.4 about volume and distance were made with calibrated microphones. In

particular, this was already the case for the measurements mentioned previously in B.4.1 about the speaker volume.

B.4.2.2 Each micro values

Contrary to the position of the speaker which is close to the centre of the e-puck, the microphones are much more widely distributed on the robot body (cf. Figure B.20 above). The distances between the microphones are not huge: 6.2 [cm] between the left and right microphones, and about 4.4 [cm] diagonally between the rear and each of the other two microphones. However, these distances result in very large differences in perceived volume here too. This makes individual use of the microphones inadequate for estimating the distance between two robots. To observe these variations, we used a variant of the experiment presented in section B.4.1.1 above : the "close/far speaker orientation experiment" represented by Figure B.15.

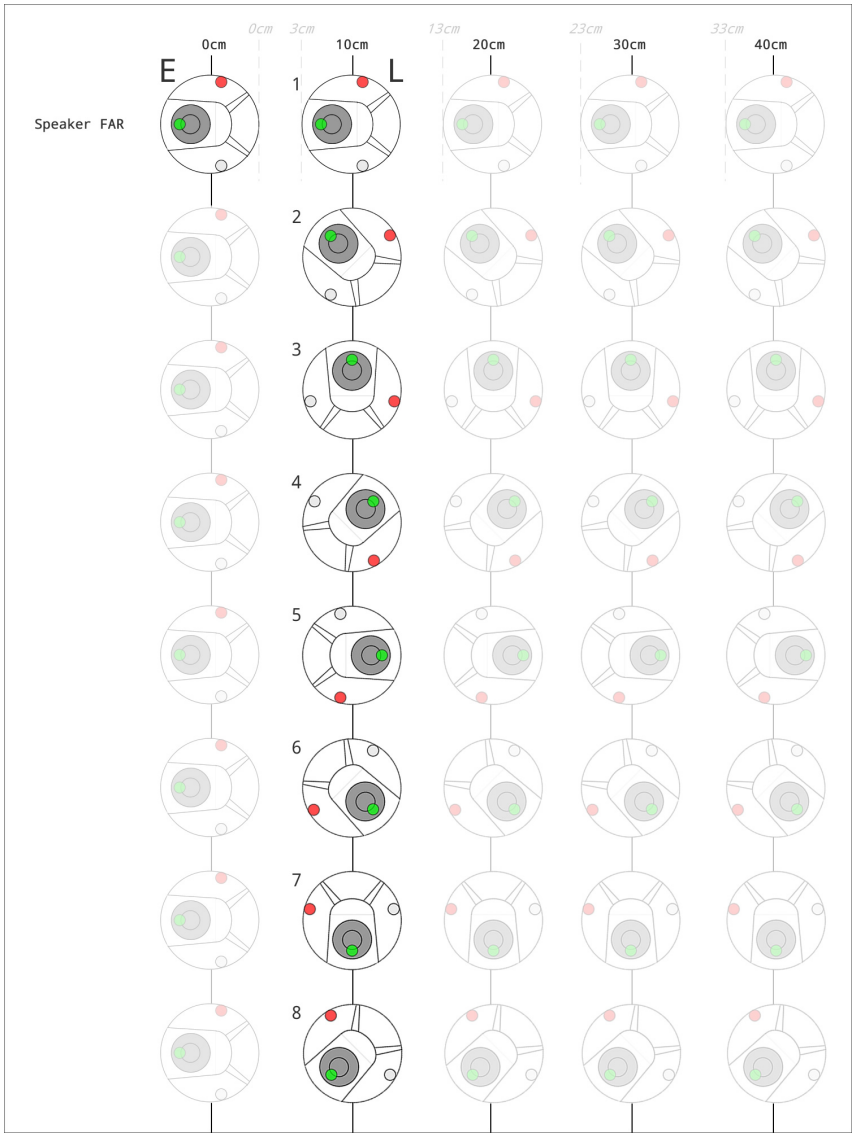
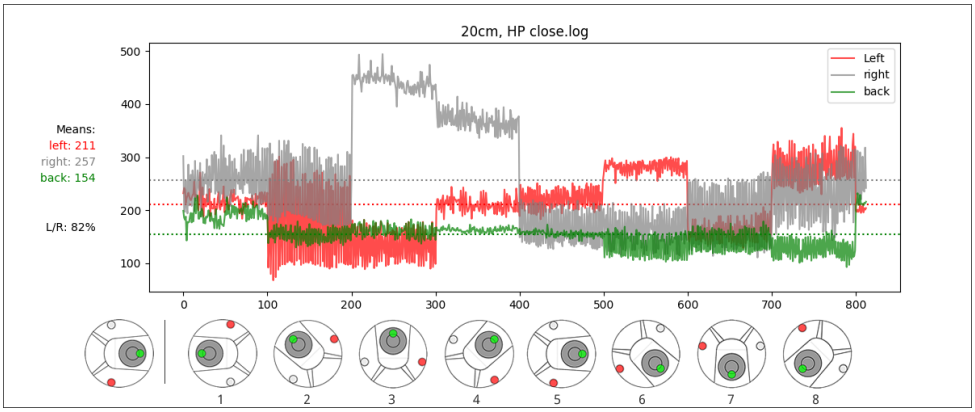


Figure B.23: The different positions of the two e-pucks used to measure the effect of the listener orientation (by 45° steps) on the perceived volume. (E: the emitter, L: the listener.) This diagram represents the first half of the experiment with the emitter in the "far" orientation. Another similar diagram with the emitter in the "close" orientation would describe the second half of it.

Rotating listener volume experiment (Figure B.23):

- For each of four distances (10, 20, 30, 40 [cm] center to center), we considered 8 different orientations for the listening robot: the ones obtained by repeatedly rotating it by 45° steps.
- As in the experiment of B.4.1.1 before, all the measurements were performed with the speaker of the emitting robot in two different orientations: once in the "close" orientation and once in the "far" one. The emitted sound is a continuous beep of 3072 [Hz] with a volume of 80 on an e-puck HWRev 1.3.
- We measured the values for each of the three microphones separately. On the diagram, each coloured dot symbolizes a microphone: the left one in red, the right one in grey and the rear one in green.

For each distance and each speaker orientation far/close, we obtain a graph such as Figure B.24 representing the variations of the perceived volume for each micro depending on the orientation of the listening robot.



.

Figure B.24: Example of a graph showing the volume perceived by each of the 3 micros of the listening e-puck while it rotates on itself by 45° steps.

Under the X axis, the robot on the far left represents the emitter and its orientation: far or close. The next drawings represent the 8 successive orientations the listening robot can adopt during the measurement.

The top of the graph mentions the distance center-to-center and the speaker orientation. The Y-axis represents the value of the volume perceived. Dotted lines are the means for each micro. Finally "L/R" return the value of the mean of the left micro divided by the mean of the right one. This ratio is used in the text.

The measurement were carried out with slightly modified version the scope visualizer program developed in the context of our third main project – the one connected to swarm coherence (chapter 5). Among others, we turned off the leds display and recorded the measurement values through bluetooth.

- files location: https://github.com/jrlauper/jrl_epuck/tree/master/jr_demo
- main files: `run_alpha_algo.c/h`
- function: `void scope_visualizer(...)`

B.4.2.2.1 Without reflective cardboard The first measurements were made using the e-puck in its standard version, i.e. without the add of the reflective cardboard mentioned in B.4.1.3. They are represented in Figure B.25.

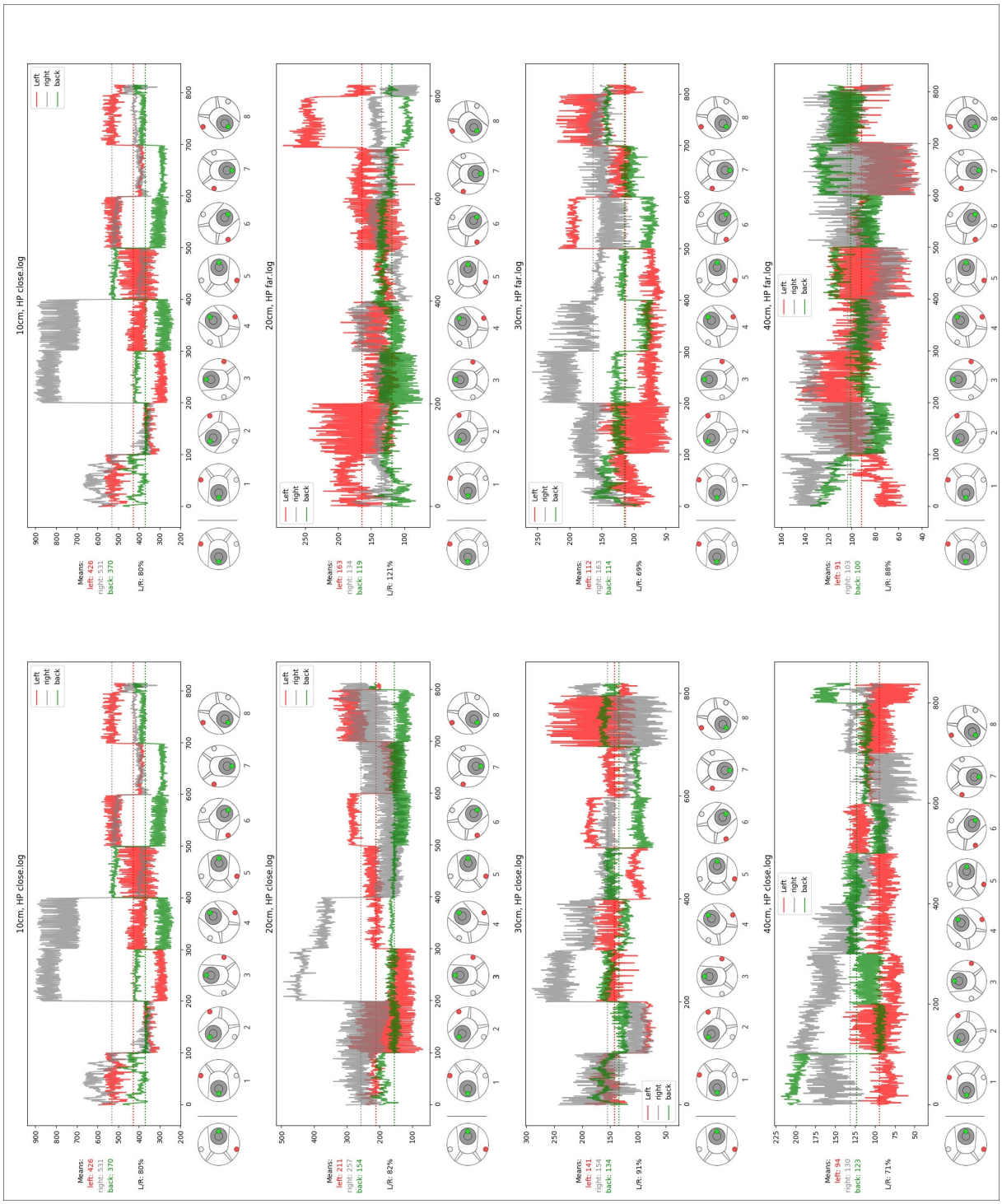


Figure B.25: Perceived volume measurement for each of the three microphones when the listening robot rotates on itself by 45° steps – individual values.
A graph for each distance (10,20,30,40 [cm] center to center)
and for each orientation of the emitting speaker: close vs far.

Let's pause and take some time to have a look at the 8 graphs obtained. Many things are as surprising as unexpected on them . As an example, let's consider the second row corresponding

to 20 [cm], and let's compare the graph for [20cm, close] with the graph for [20cm, far] where the listener is simply 2.4 [cm] further away from the emitting e-puck. While in the "close" graph, the right microphone measures high volumes, it measures very low volumes in the far one. Why does the right microphone suddenly behave like this and not the others?

Among all the possible observations concerning these 8 diagrams, the most general and the most relevant ones given our goal (to estimate the distance given the perceived volume) are the following ones:

(1) In many cases, it is difficult to understand the relationship between the difference in volume measured and the orientation of the listening robot. It is as if there is no systematicity - or at least we do not manage to perceive any. In addition, the left and right microphones do not behave in a similar way in similar positions: there is not the symmetry of behavior that one could expect between them given their symmetrical disposition on the e-puck's body. This can be illustrated, for example, the graph [20cm, close]:

> In orientation 3, the right microphone (in gray) is in the position closest to the emitter and, not surprisingly, the volume measured for the right microphone is the highest. However, if we look at the left microphone (in red), we can see that the highest volume measured is not in orientation 7, as one would expect, especially by symmetry, but in orientation 6. Furthermore, in 7, the volume measured by the left microphone is not just a little bit lower, but really much lower.

And this is just one of many examples - the 8 graphs being full of hard to explain surprises like this one.

(2) For the same microphone at the same distance, the volume variations measured for different orientations are very large, while the overlaps between the possible volume variations for different distances are considerable. See the boxplots of Figure B.26 .

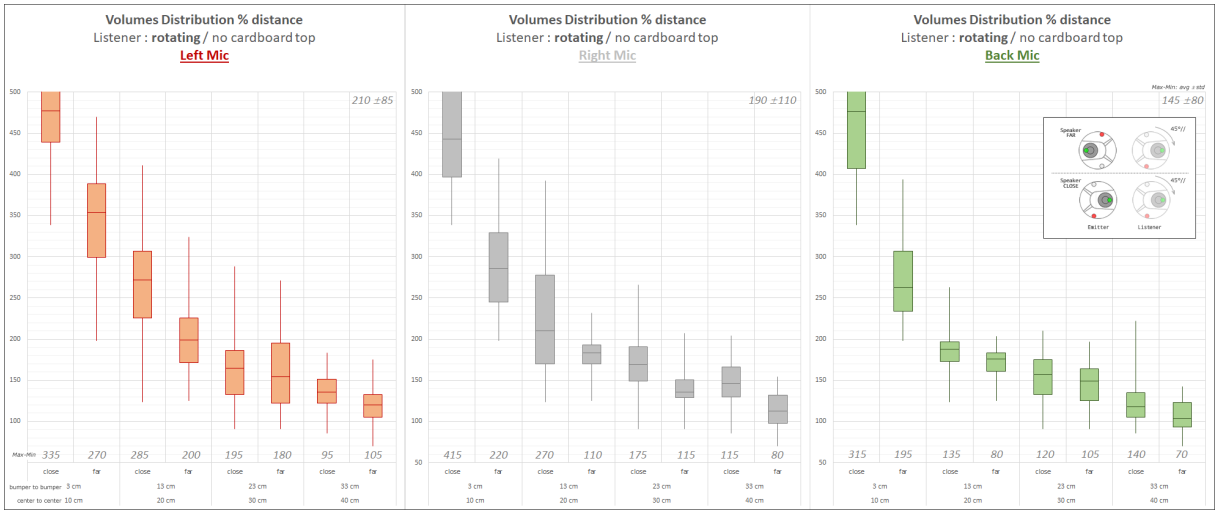


Figure B.26: Boxplots summarizing the distribution of perceived volume values for each of the three microphones when the listening robot rotates on itself by 45° steps.
The values at the bottom are the max-min distances rounded up to 5.
The value in the upper right corner is the mean of these distances ± standard deviation.
(more than 800 measurements for each boxplot)

Let's consider the left microphone, for example. For the same measured volume of 220, a robot can be 10, 20 or 30 [cm] away from the emitting robot. Since part of the recorded variations come from the orientation of the emitting speaker

in far or close orientation (see B.4.1), we can consider measurements for one of the two orientations only. However, although it diminished, the same problem then remains – without mentioning the fact that we would like to be able to use distance assessment while robots are in motion, in consequence, without being able to fix in such a way the orientation of the emitter.

B.4.2.2.2 With reflective cardboard Adding a reflective cardboard (see B.4.1.3 for more details) improves the far/close differences for the right and back microphones, but unfortunately worsens the general extent of variations and overlaps – which is detrimental to distance estimation. See Figure B.27 .

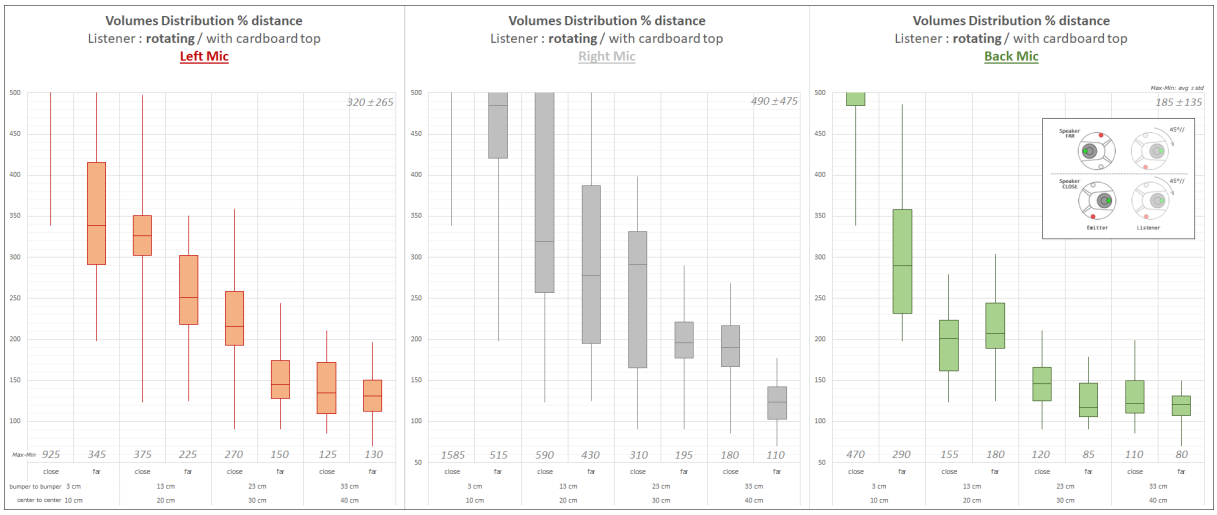


Figure B.27: Boxplots summarizing the distribution of perceived volume values for each of the three microphones when the listening robot rotates on itself by 45° steps, if a reflective cardboard is added.
The values at the bottom are the max-min distances rounded up to 5.
The value in the upper right corner is the mean of these distances ± standard deviation.
(more than 800 measurements for each boxplot)

B.4.2.3 Average value on the three microphones

An important part of the variations in the measured values for each microphone (for a distance and a far/close orientation fixed) comes from the greater or lesser distance between the microphone and the emitting speaker caused by the listening robot's orientation. By using the average on the three microphones, the possible volume differences (for a distance and a far/-close orientation of the emitter given) become much less important: see Figure B.29. However, not surprisingly, these differences are still quite large. Indeed, we have seen that there is not the expected symmetry of behavior between the left and right microphones, while the position of the rear microphone, hidden in its "resonance chamber", considerably alters its perception of the volume.

Figure B.28 gathers the 8 graphs obtained for the *average* on the three microphones, for the different distances and the two far/close emission orientations. As announced, there are still strong variations depending on the orientation of the listening robot. However, the ranges within which the average measured volumes are found are much smaller than the ones we got when using the microphones alone and become more usable for distance estimation: compare the boxplots and the values of Figure B.29 for the average with the boxplots and the values of Figure B.26 for each individual micro.

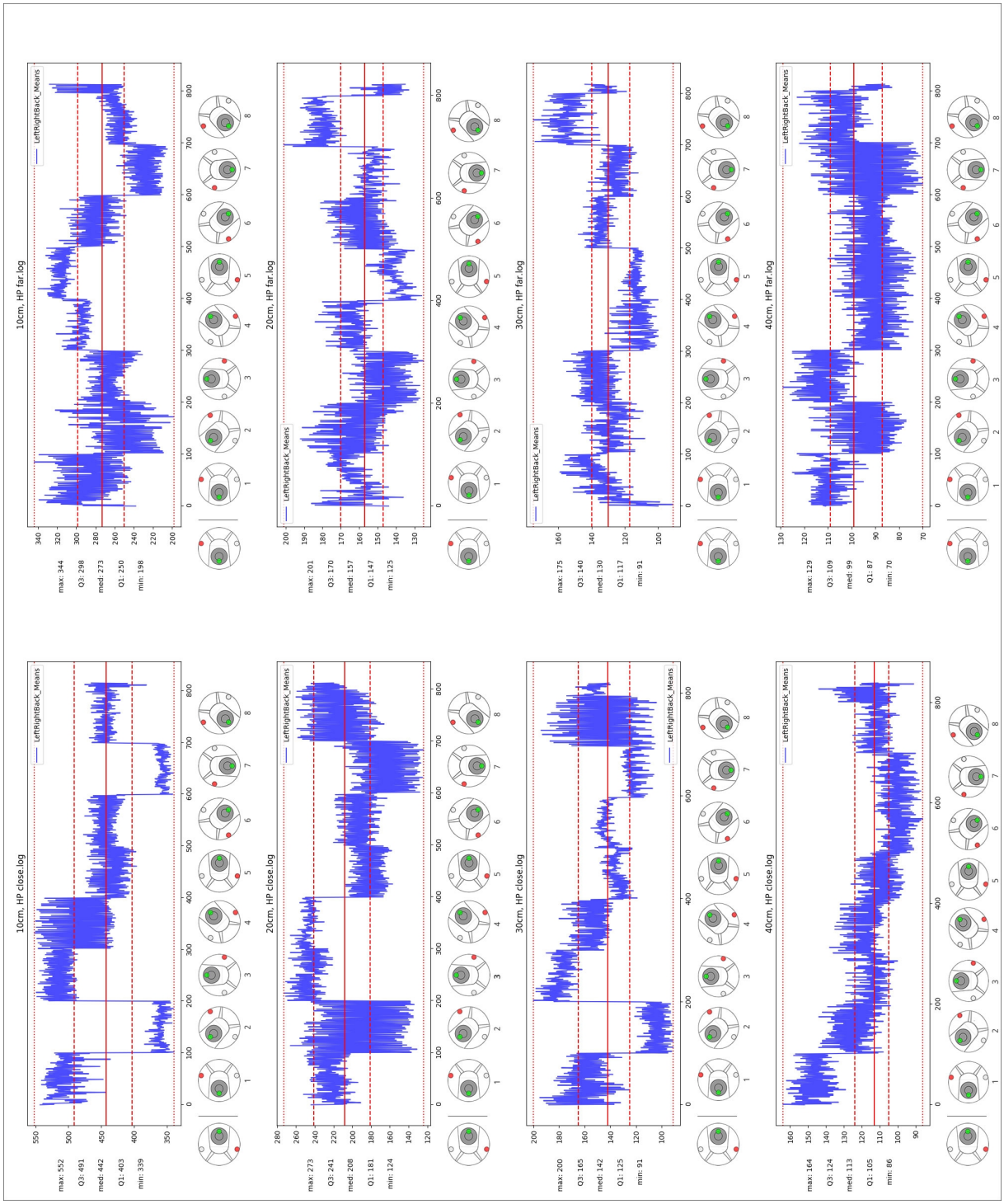


Figure B.28: Average perceived volume measurement
when the listening robot rotates on itself in 45° steps – individual values.
A graph for each distance (10,20,30,40 [cm] center to center)
and for each orientation of the emitting speaker: close vs far.

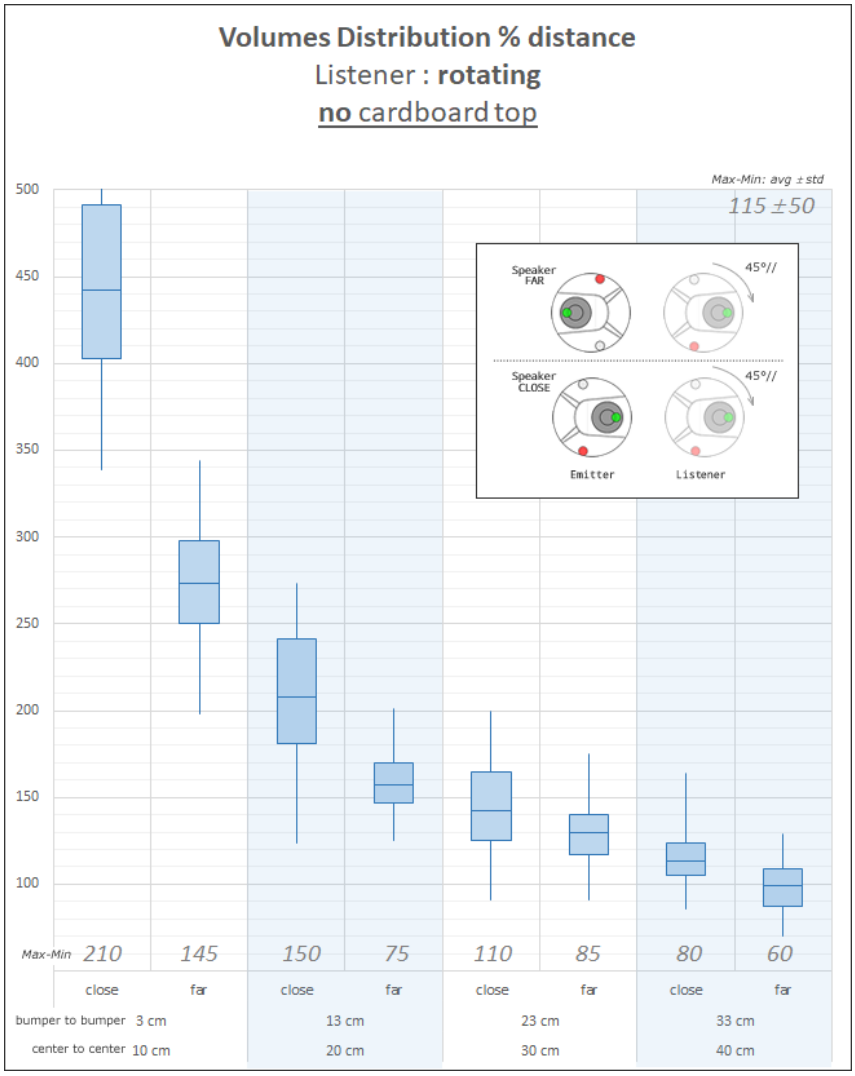


Figure B.29: Boxplots summarizing the distribution of average perceived volume values when the listening robot rotates on itself by 45° steps.
The values at the bottom are the max-min distances rounded up to 5.
The value in the upper right corner is the mean of these distances ± standard deviation.
(more than 800 measurements for each boxplot)

Although the possible variations for volume averages are reduced, the situation is still far from ideal if we want to use these values to estimate the distance between two robots. Indeed, there are still many overlaps among the volume ranges for different distances, and the same measured volume can correspond to different distances within 10 [cm]. For example, what volume threshold should we take if we want to keep the robots at a 20 [cm] distance? Above 200, we lose the [20cm, far], whereas below 200, we are likely to accept robots that are at 30 [cm].

B.4.3 Possible improvements?

Is it possible to do better? i.e. to obtain a better distance estimation by using the volume perceived. When you look at the 8 diagrams with the volumes for each micro measured separately (Figure B.25) several ideas can emerge. Here are some of them and what you get if you try to apply them.

1. *Weighing the average of the volumes* by giving a lower coefficient for the right micro. Indeed, the average for the right micro almost always seems to be higher than the average for the left micro. On the 8 diagrams, the L/R value on the left of the graphs expresses this ratio which varies between 70 and 90% depending on the case. However, if one tries to weight the average by giving a lower coefficient to the right micro (for example 0.8), the average of the right micro and the left micro calculated for all the samples certainly becomes much closer, but unfortunately, the total average of the three microphones calculated for each sample continues to fluctuate a lot. Moreover, in some situations, such as [20cm, far], the average of the left micro is the highest.

2. *Using the rear microphone only.* A quick glance at the 8 graphs with the separate volumes shows that the volume of the rear micro very often varies much less than that of the right and left micro across the samples. And this is true. However, the observed differences are still higher than those obtained with the average of the three microphones. Thus, there is no gain when using the volume of the rear micro only.

3. *Using the cardboard add-on above the emitting speaker* (as seen in subsection B.4.1.3). This should already reduce the differences between the volumes measured in the "close" and the "far" orientations of the speaker as observed on Figure B.19. And indeed, as shown in Figure B.30 below, the use of a cardboard add-on does quite good job at it. Unfortunately, in situations where the listening robot moves and changes orientation (simulated by the 45° rotation steps), the use of a cardboard add-on also increases the ranges of possible average volumes measured for a certain distance. This increases the overlaps between measured volumes for different distances. As a consequence, there is no total gain in using a cardboard add-on if you want to use volume to estimate distance. A comparison of Figure B.29 and Figure B.30 should quickly convince you.

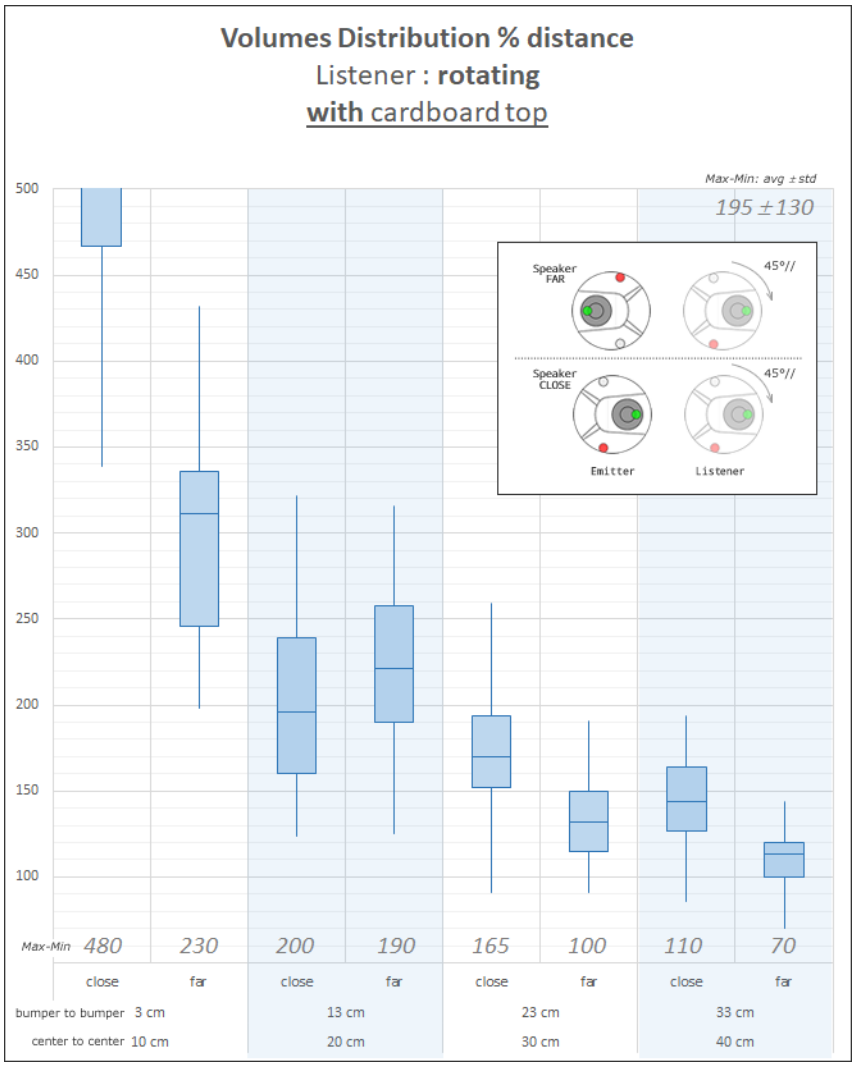


Figure B.30: Boxplots summarizing the distribution of average perceived volume values when the listening robot rotates on itself by 45° steps, if a reflective cardboard is added. The values at the bottom are the max-min distances rounded up to 5. The value in the upper right corner is the mean of these distances ± standard deviation. (more than 800 measurements for each boxplot)

B.4.4 Best option at our disposal

Given the observations made in B.4.2 and B.29 , the best option we have to use the recorded volume to estimate the distance between two robots is to consider the *average* over the three microphones *without* using a cardboard add-on. Indeed, it is in this situation that the measured volume ranges for each distance are the smallest. The ranges we then have are the ones of Figure B.29 previously seen and reproduced here:

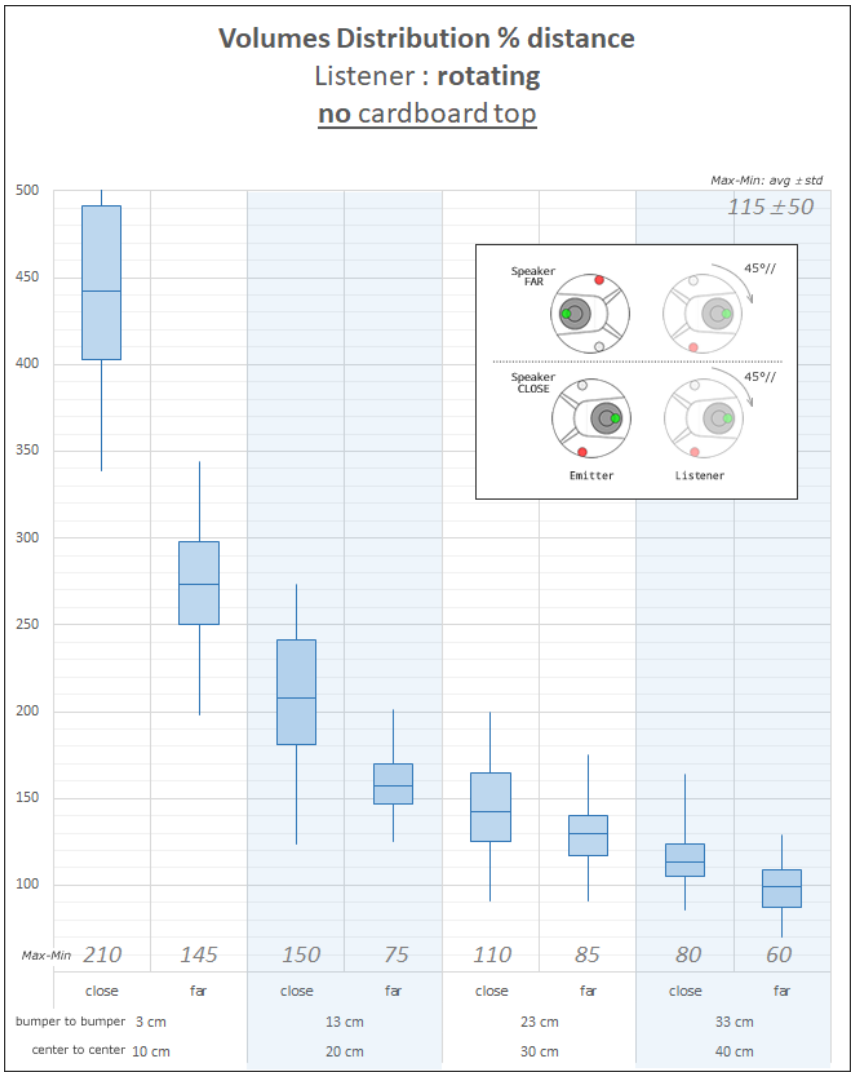


Figure B.29 Boxplots summarizing the distribution of average perceived volume values when the listening robot rotates on itself by 45° steps.
The values at the bottom are the max-min distances rounded up to 5.
The value in the upper right corner is the mean of these distances ± standard deviation.
(more than 800 measurements for each boxplot)

The best option at our disposal is far from ideal. Indeed, there are still many overlaps between the volume ranges corresponding to different distances. Let's say it is the "least bad" option available to us. From this point, if we want to work with the perceived volume to determine the distance between two robots, it is clear that we have to play the compromise and do some tests to see what behaves best. However, no matter what we decide, there will be mistakes, and the best compromise will be just to aim for as few mistakes as possible.

To illustrate this last point, let us take the example already taken several times: Let's suppose that we want the leds to light up when the robots are less than 20 [cm] away from each

other.

- If you take a threshold higher than 200, no robot above 20 [cm] should turn on. However, some robots below this threshold will not turn on either.
- If you set the threshold to 130, then all robots within a distance of less than 20 [cm] will turn on, but many that are more than 20 [cm] apart will also turn on.
- A compromise would be to use a threshold around 145 (first interquartile of [20cm, far]). In this way, you can hope to include a large part of the robots that are at a distance of less than 20 [cm] while not including too many of those at 30 [cm], since the median for [30cm, closed] is lower than 145.

Finally, let's recall that we used the same specific pair of e-pucks for all the measurements of this section – the idea was to observe the variations of the perceived and emitted volume without adding to them inter-robots variations. However, given the existing variations between e-pucks, the possible variations in volume for each distance are even larger. This further reinforces the need for testing and compromise.

C Other general software bricks and issues solved

Contents

In this chapter, we introduce three aspects we explored while developing our three main projects. They are general enough to be of interest to anyone working with e-pucks in an embedded way. These aspects are the following ones:

1. the e-pucks difficulty in detecting the presence of the other e-pucks via infrared proximity sensors and how it can be remedied;
2. the generation of random numbers with the e-puck whereas it does not include a real-time clock;
3. the usage of an infrared remote control with e-pucks to choose a program and launch it (especially useful when working with many e-pucks).

C.1 [Hardware] E-pucks mutual detection problem (with IR proximity sensors)

The e-puck has infrared proximity sensors placed all around its body under its bumper. These sensors enable it to efficiently detect objects that are in close proximity to it thanks to infrared light reflection. Unfortunately, in its original version, the e-puck detects other e-pucks very poorly because of their transparent bodies that reflect little infrared light. In this section we briefly introduce the infrared sensors of e-puck (C.1.1), we document the poor mutual detection of e-pucks in their original version (C.1.2), and then we show how adding a small reflective tape on the bumper of e-puck solves this mutual detection problem satisfactorily (C.1.3).

C.1.1 E-puck IR proximity sensors

The e-puck has 8 infrared proximity sensors arranged on its circumference as shown on Figure C.1.

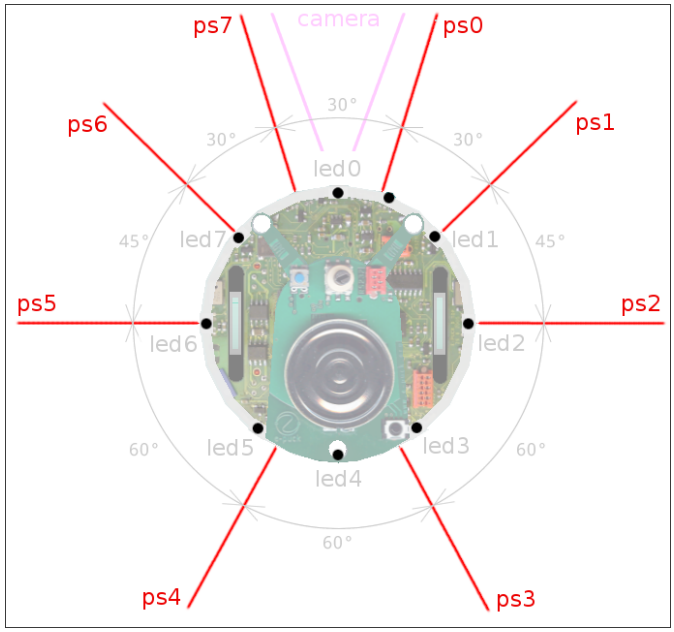


Figure C.1: E-puck IR proximity sensors: disposition with angles. Modified version of a figure from Cyberbotics Ltd (2020).

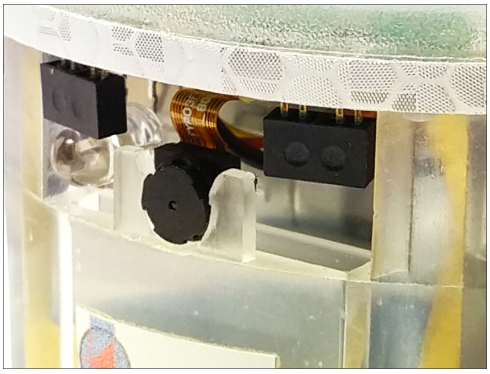


Figure C.2: Rectangular front IR proximity sensors ps0 and ps7 surrounding the camera.

Infrared proximity detectors have a small diode capable of emitting infrared light and a sensor capable of measuring ambient infrared light (Figure C.2). To determine if an object is nearby, the detector :

1. measures the ambient infrared light a first time;
2. emits infrared light; and
3. measures the new ambient infrared light almost instantaneously.

The difference in ambient light measured before and after the emission of infrared light allows the robot to detect an object and estimate its distance:

- if an object is close to the sensor, a lot of infrared light will be reflected by the object and thus the difference in measured ambient infrared light will be high; conversely
- if an object is far away, little infrared light will be reflected and the measured difference in ambient light will be low.

The reflective properties of the object to be detected thus play a key role in its detection. There are large differences between the measurements returned by the different sensors and they must therefore always be properly calibrated before use. Once calibration done, there are still differences between the measurements returned by the sensors in similar situations; besides the returned measurement contains a certain amount of noise. But the ensemble is perfectly usable.

The value measured in relation to the distance drops sharply in the first few centimeters and then decreases more gradually. The Figure C.3 from Cyberbotics Ltd (2020) represents very well the typical behavior of a sensor (the values themselves are of course to be relativized, each sensor behaving a little differently):

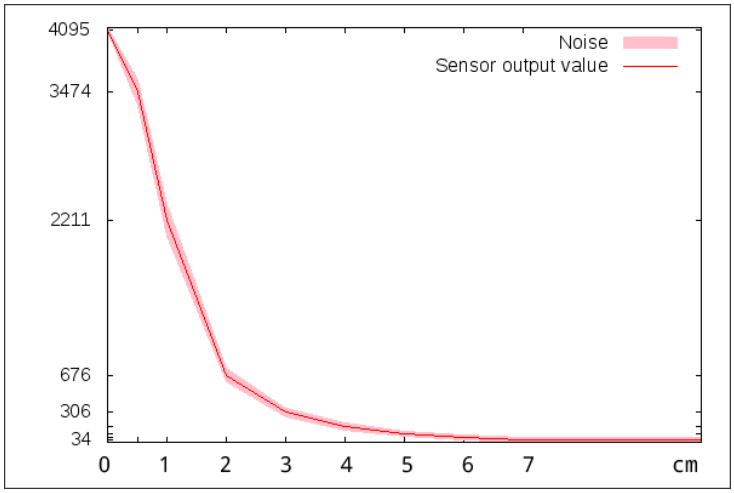


Figure C.3: IR proximity sensor values wrt to the distance.
Very slightly modified version of a figure from Cyberbotics Ltd (2020).

This behavior is particularly suitable for detecting close objects and avoiding obstacles. Indeed, given the sudden drop, one does not need to be very precise in setting a threshold to avoid an obstacle. This behavior is much less suitable for distances greater than 4 [cm], considering the noise surrounding the measurement and the differences between sensors.

C.1.2 E-pucks bad mutual detection

The main body of the e-puck is completely transparent (see Figure C.4). In terms of design, this allows its entire body to be illuminated with a green led, providing it with an additional actuator.

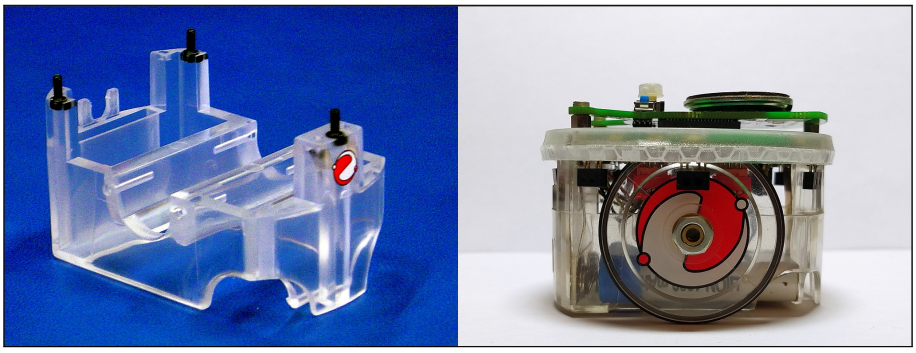


Figure C.4: The transparent body of the e-puck.
The left picture comes from E-Puck.org (2018).

Seen from another e-puck on a horizontal plane, the only non-transparent elements likely to reflect infrared light are mechanical or electronic elements or a few stickers. The result is problematic: e-pucks detect each other very poorly via infrared sensors. To see this, consider Figure C.5. What the graph of the figure shows is striking: when the object to be detected is an e-puck, the values returned by the sensors are extremely low and do not correspond at all to what can be expected from a normal object such as a white chipboard, a piece of cardboard, or a polystyrene (EPS) cube. Depending on the orientation of the e-puck to be detected, or whether some of its lights are on or off, there are certainly small differences. But the measured values all remain very low. Let's compare:

- at 1 [cm],
  - for an e-puck to detect: in general, all values are < 300, whereas
  - for a usual object: all values are > 800 (they can even exceed 2000!)
- at [2 cm]:
  - for an e-puck to detect: in general, all values are < 150, whereas
  - for a usual object: all values are > 250.

The consequence of this situation is the following one. Suppose that our goal in using proximity sensors is to avoid obstacles (which is in fact their most frequent use):

- if we choose a detection threshold suitable for usual objects, then the e-pucks will not detect each other and collisions between them will be frequent; on the contrary,

- if a suitable threshold is chosen for the mutual detection of e-pucks, then there are no more collisions between them, but usual objects are "over-avoided" with obstacles that are avoided with far too great a distance.

If we don't make any hardware changes to the basic e-puck, then we have to try to find a compromise. However, experience shows that the detection difference between e-pucks and usual objects is so great that most of the time, the result of such a compromise tend to be unsatisfactory: collisions between e-pucks remain, while normal objects are still avoided at far too great a distance (which is particularly problematic in small arenas).

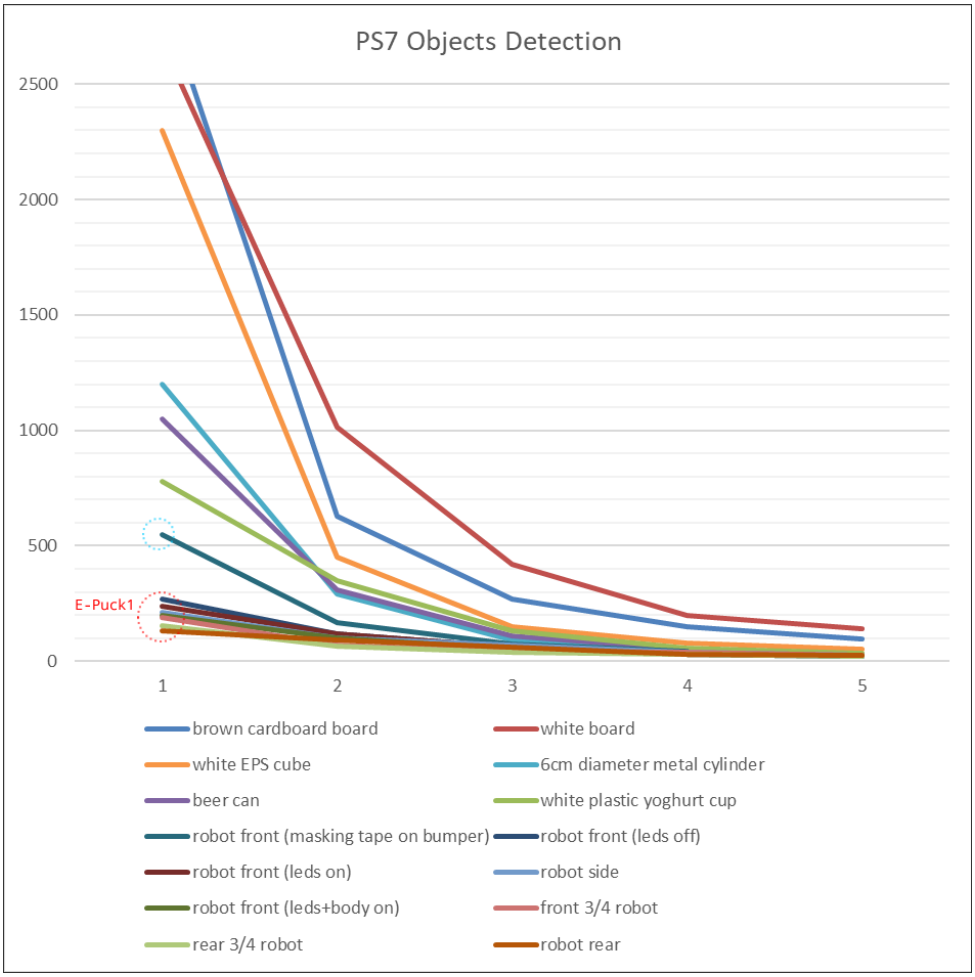


Figure C.5: How a standard e-puck is detected by an IR proximity sensor in comparison to other objects.

The front of the measuring e-puck is directed towards the object: it is oriented along the led4-led0 axis, with the sensors ps7 and ps0 in the direction of the object. For clarity, the graph shows only the *average* measured values without margins of errors and this for one specific robot. However, the graph is well representative of what happens with other sensors and other robots. For clarity and brevity, we focus on ps7 leaving out ps0. Let's observe that the measurements for "normal" cylindrical objects (metal cylinder, can, yoghurt pot) are lower than for "normal" objects with a straight face. This is quite normal given their cylindrical shape joined to the arrangement of the ps7 and ps0 sensors.

However, on Figure C.5, the blue curve that starts a little bit above 500 (surrounded by a small blue dotted circle) gives the track to follow if we want to improve things. It describes the way an e-puck with painter's tape stuck on its bumper is detected. We can observe that this simple change, already allows it to be detected much better.

C.1.3 Highly reflective tape

The mutual detection problem between e-pucks must be an unexpected but recognized effect of its design. In any case, the manufacturers of e-puck must be aware of it since, on the gctronic shop (GCtronic, 2019a), you can buy reflective tapes to stick on e-puck's bumper (∼ CHF 15.-/robot, pose included, if I'm not wrong): see Figure C.6.

Figure C.6: Reflective strips for e-puck mutual detection proposed on GCTronic shop: `https://www.gctronic.com/shop.php` (GCtronic, 2019a)

Inspired by these tapes, we ordered a roll of reflective tape used, for example, for the visibility of bicycles at night. These tapes, containing some kind of small crystals, have a very strong reflective power. Their precise reference is the following one, but other reflective tapes of the same kind should give similar results:

> VICTGOAL Bike Stickers, Basic Silver, 1 [cm] x 8 [m] (about $ 2.-/8 [m])
> https://www.aliexpress.com/item/32913143781.html
> (accessed 19.02.2020) [Figure C.7]

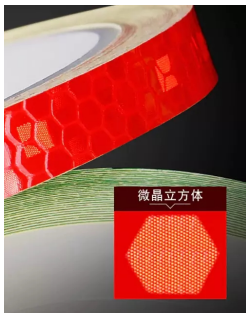


Figure C.7: The reflective tapes we used for e-pucks mutual detection. (Illustration of the red version).

These tapes have such a reflective power that you have to be careful not to put too much of them, because the e-pucks can then be over-detected! After several tests, the use that gives us the most adequate results is the following:

Apply a 3 [mm] wide strip to the lower part of the bumper.
(in a manner similar to what is offered by the GCTronic shop)
In terms of length: 25 [cm] works especially well.

Figure C.8 shows the measurements you get if you modify the e-puck in this way

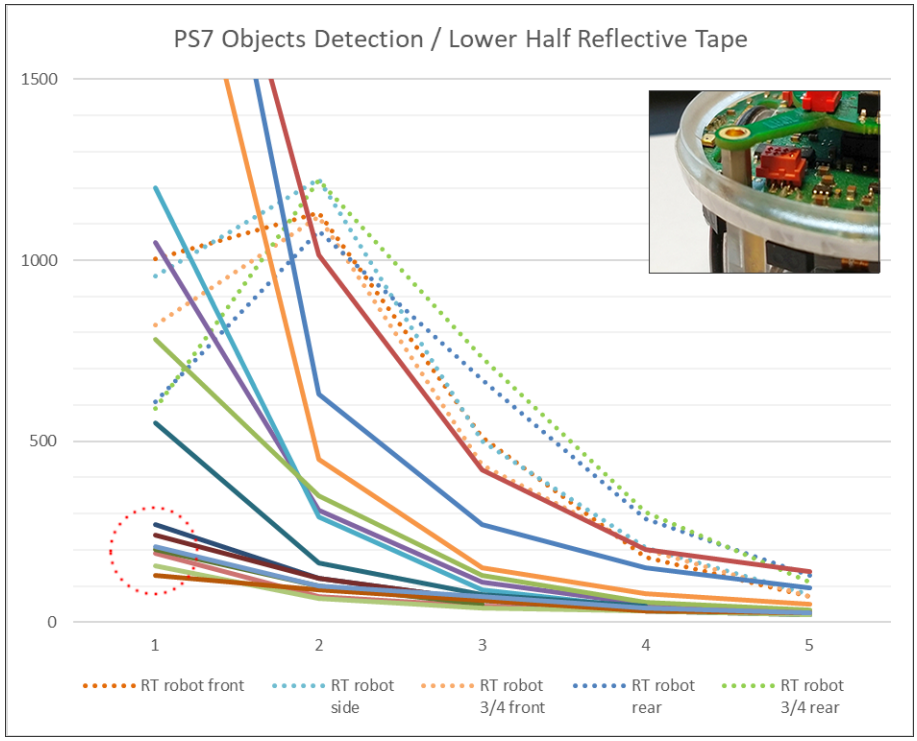


Figure C.8: How IR proximity sensors detects an e-puck equipped with a 3 [mm] reflective strip stuck on the lower part of its bumper.

At 1 [cm], some values, especially for the rear of the robot, are lower than those of a "normal" object. But at 2 [cm] the robot is detected as a normal object with a strong reflectivity similar to a white chipboard. The lower values at 1 [cm] are most likely due to the fact that the bumper is higher than the proximity sensors: about 4 to 4.5 [cm] for the bumper and about 3 to 3.5 [cm] for the sensors. This means that when two e-pucks are very close to each other, the light emitted by the sensors probably cannot reflect sufficiently into the reflective tape.

It is also important that the height of the reflective tape is as close as possible to the height of the sensors. Figure C.9 show us what we get if we attach a 3 [mm] reflective tape not to the lower part of the bumper, but to the upper part of the bumper. In this case, the improvement in reflection compared to the normal tapeless situation is much smaller – the reflective tape is not sufficiently in front of the infrared sensors to reflect the infrared light adequately.

Let us add that the edge of the bumper is not completely vertical like a $|$, but is slightly oblique like a $\langle$: the upper part of the edge of the bumper is slightly inclined upwards while its lower part is slightly inclined downwards. Therefore, if a IR beam hits the upper part, there is good chances that it will be reflected slightly upwards, out of the range of the infrared sensors. Conversely, if a beam hits the lower part of the bumper, it will be reflected slightly downwards, making it easier to detect.

In our first tests, we tried to place a 7 [mm] reflective tape across the entire width of the e-puck's bumper. It can be observed that the reflection then becomes too strong: see Figure C.10. The little extra given by the upper part of the covered bumper then makes the curves obtained for the detection of e-pucks deviate too much from those of "normal" objects.

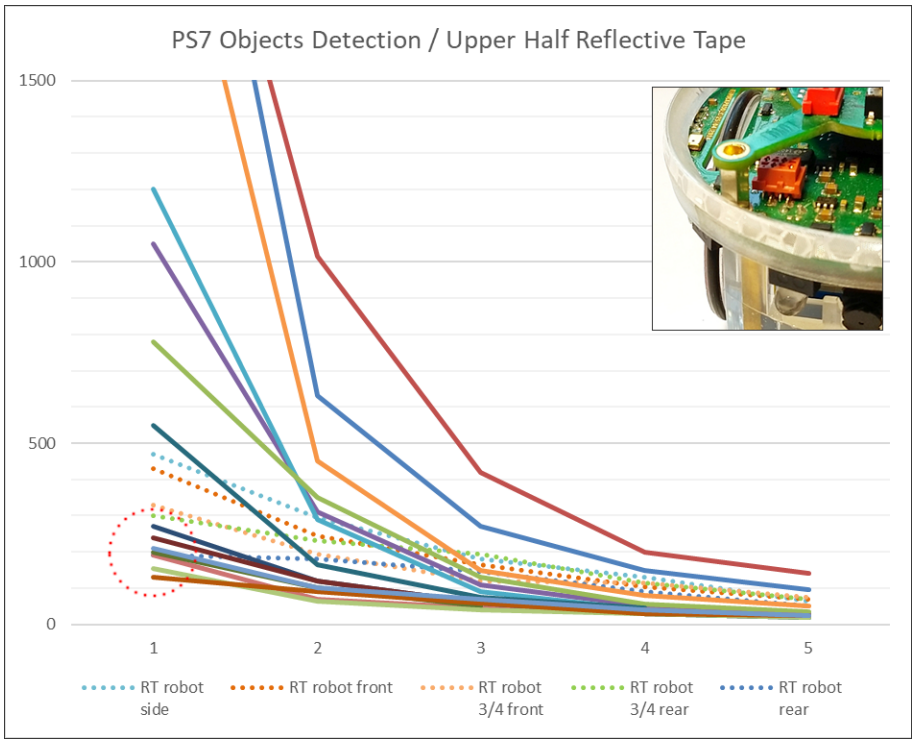


Figure C.9: How IR proximity sensors detects an e-puck equipped with a 3 [mm] reflective strip stuck on the upper part of its bumper.

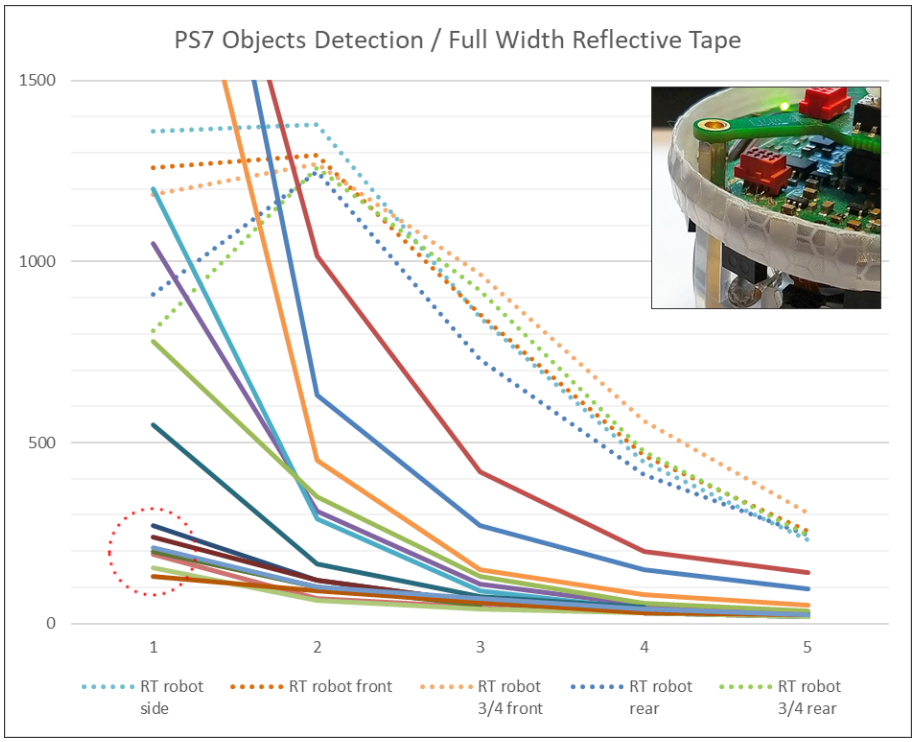


Figure C.10: How IR proximity sensors detects an e-puck equipped with a reflective strip covering the full width of its bumper.

C.2 [Software] Pseudo-random number generators with e-pucks (using IR proximity sensors)

Many programming situations require the use of pseudo-random numbers and experiments with e-pucks are no exception.

For example, for our second main project "chirping", we want the e-pucks to synchronize with each other. But in order for the experiment to be possible they must be out of sync at the beginning. This "desynchronization" can be generated manually by launching the program on each e-puck one after the other with a certain delay. But if you run them all at the same time, for example using a remote control (see 5.3.), they will already be almost all synchronized from the start and there is no experiment. Hence the interest of a random number generator which will allow us to have different offsets for each robot.

In another small side-project, we made a little game where the player has to repeat a series of notes played by the e-puck. How can there be a game if the series of tones played is always the same?

In this section we explain the e-puck's specific issues with getting random numbers and describe two ways that can be used to solve them.

C.2.1 Pseudo-random numbers with C rand() and a suitable seed

When programmed in an embedded way, the e-puck is programmed mainly in C while its libraries are programmed in C and Assembly. To generate a random number in C, two functions of the `stdlib.h` library are generally used. The library and these two functions are also available for e-puck:

- `int rand(void)`: which generates a random number between 0 and RAND_MAX
- `void srand(unsigned n)`: which initializes the random number generator with the seed n. (Prinz and Kirch-Prinz, 2002, p.91)

The sequence of numbers returned by `rand()` is always the same for the same seed. And, if no seed value is provided, the rand() function is automatically seeded with a value of 1. (Linux Man-Pages). In order not to have exactly the same sequence of numbers, the main challenge concerns the value used as seed. In a personal computer, the most commonly used solution is to use the real time of the internal clock as seed. To this end, you use the function

```
time_t time(time_t *pSec);
```

which returns the number of seconds that have elapsed since a certain time (usually January 1, 1970, 00:00:00 o'clock).

Since the e-puck has no internal clock returning the real time, you must find another source that can give you a seed that varies enough. To do this, you can simply use the value returned by one of the infrared proximity s. Indeed, these s always have a certain noise and therefore seem well suited for this purpose. Once the seed is entered, you can use rand().

C.2.2 [Software Module] Pseudo-random number using IR proximity sensors only

- Files location : https://github.com/jrlauper/jrl_epuck/tree/master/jr_demo
- Files: `utility_mod3.c/h`
- Function: `int jr_random(int max_number)`

- Demo video: https://youtu.be/MvjC3RLYyS8

We have created an alternative to obtain a random number with an e-puck that uses neither `rand()` nor `srand()`, but relies only on variations in measurements inherent to proximity s. The basis of the corresponding files are the `utility.c/h` files of the e-puck standard library to which we added other additional functions of general use. The function enabling the generation of a random number is :

> `int jr_random(int max_number)` : returns any number between 0 and max_number inclusive.

The basic principle of this small function is the following one:
Suppose that max_number is 3456:

- For each of the 4 digits forming max_number, you randomly draw a number between 0 and 9 using the `get_basic_random` function described below.
- you then get a certain number, let's call it result.
- the function return result modulo (max_number+1).

The key function at work is thus function:

> `int get_basic_random(int ir_)`.

It works like this:

- You measure the value Val1 from a 1st infrared designated by ir_ ($0 \leq$ ir_ ≤ 7).
- The measured value Val1 modulo 8 indicates a 2nd whose value Val2 is measured.
- The measured value Val2 modulo 8 indicates a 3rd whose value Val3 is measured. At last,
- The function returns Val3 modulo 10.

The use of three different measures is dictated by experience. This is the way to get the best results in a reasonable amount of time.

A "false good idea". Since infrared s can return values going from 0 to 4000, you may be tempted to use, instead of get_basic_random, the following basic principle:

> for a random number between 0 and Max with Max = 4000,
> you take directly the value returned by one of the s, modulo Max+1

However, this is not a good strategy because if the value that can be returned by a can range from 0 to 4000, it is not in any condition! To illustrate this point, let's imagine the following example:

- you've got 10 robots;
- for an experiment, you need each one to return a random number between 0 and 4000;
- in the initial situation, the robots are in a calibration situation, so there is nothing in front of their s for a long enough distance (at least 4 to 5 [cm]);
- just after calibration, the value of one of their s is taken as a random number.

What do we get? Certainly, the returned values will be between 0 and 4000. But above all, they will all be lower than 150, given the context! This is certainly not what you want when you want a random number between 0 and 4000. Using `get_basic_random` avoids this problem by using the s for only one digit at a time – without mentioning that you are not limited by the 4000, the max value returnable by the IR-sensors.

C.3 [Software module] Infrared remote-control module for selection

- Files location: https://github.com/jrlauper/jrl_epuck/tree/master/jr_demo
- Files: `remote_control_utility.c/h`

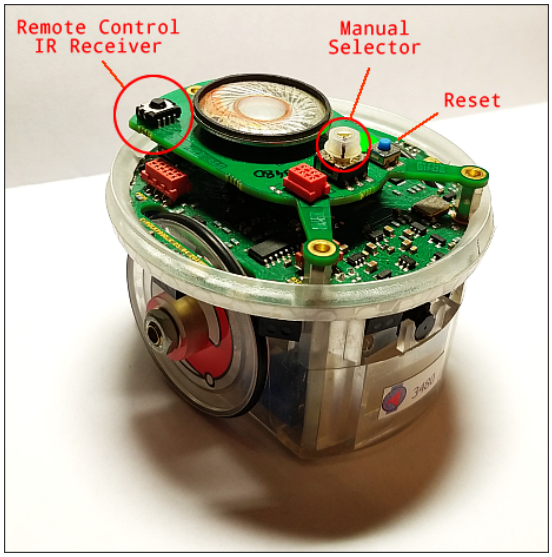


Figure C.11: Position of the remote control IR receiver on the e-puck

The e-puck has an infrared remote control receiver. It is located on the rear of the platform fixed on the top of the robot's body (Figure C.11). It is used to capture signals sent by a remote control emitting a signal according to the RC5 protocol (E-Puck.org (2018) section Accessories; for more information on the RC5 protocol, see Wikipedia (2019b)).

We created a software module that makes it easy to use a remote control to transmit commands to e-puck which are normally transmitted manually. This module allows you to use a remote control:

1. to select a program – instead of using the manual selector;
2. to start or stop a program – instead of using the power switch;
3. to reset the e-puck – instead of pressing the reset buttons.

This module becomes especially interesting when working with several robots at the same time. Indeed, the remote control allows you to change programs or launch a program on all robots at the same time without having to do it manually on each of them one after the other.

Our module is based on the following module from the standard library (see Appendix A):

- folder: `\motor_led\advance_one_timer`
- Files: `e_remote_control.c/h`

The module of the standard library makes it possible to read the commands received by the infrared remote control receiver of the e-puck. Moreover, once initialized, it regularly controls "in the background" what is received by the remote control receiver.

The module we have developed from this base contains two main functions:

1. the first one allows to identify which information is sent by which button of some remote control you want to use;
2. once some remote control has been suitably set up, the second function manages the transmission of the commands mentioned earlier: program selection, launching and resetting.

C.3.1 Identification of the commands sent by some remote control

- Files location : https://github.com/jrlauper/jrl_epuck/tree/master/jr_demo
- Files: `remote_control_utlity.c/h`
- Function: `void ir_remote_code_key_identification();`

This small function makes it easy to capture the information sent by some RC5 remote control when a button is pressed, without having to look into the base module. The information captured is transmitted to the computer via bluetooth so that it can be read. The relevant piece of information sent by the remote control when a button is pressed consists of the pair: (address;data). It is this pair that can be used in the program to identify the button.

Not all remote controls using the RC5 protocol work with the e-puck and you have to test to see if this is the case or not. Experience has shown that when this is not the case, the (address;data) received always contains the same series of values regardless of the button pressed – making the buttons indistinguishable and the remote control unusable.

C.3.2 Choosing and launching programs using a remote control

- Files location : https://github.com/jrlauper/jrl_epuck/tree/master/jr_demo
- Files: `remote_control_utlity.c/h`
- Functions:
  - `void init_run_remote();`
  - `int run_remote();`

These two functions allow the use of a remote control :

- to choose a program from the 16 programs of the manual selector;
- to start or stop one of these programs;
- to restart the robot;
- to put the robot in sleep mode in order to consume less electricity during waiting or setting up.

In order to use these two functions, you must first, in
`remote_conrol_utility.h`
enter the pairs `(address;data)` corresponding to the buttons you want to use to perform the different commands (to determine the values of these pairs see C.3.1 just above). Figure C.12 shows all the commands and the buttons to which they have been assigned on the remote control we used in our experiments.

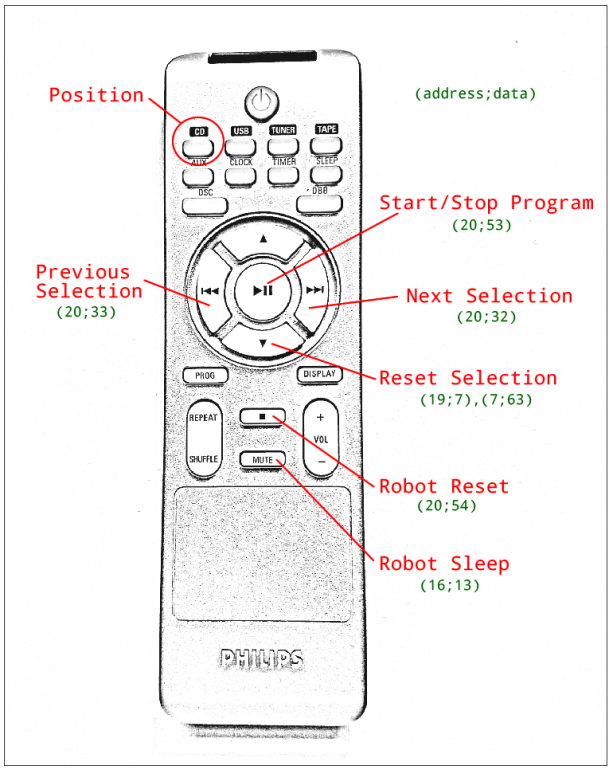


Figure C.12: Sketch of the remote control we used in our experiments with the couple (address;data) assigned to the buttons used. The "CD" position must be choosen because, on this remote control, the same button can send different signals in other modes such as "tuner" or "tape".

Once you have assigned the pairs (address;data) corresponding to the buttons you want to use, simply run

```
void init_run_remote()
```

in the initialization phase of the program – typically in `main.c`. From then on, the call of the function

```
int run_remote()
```

in the main loop manages everything and returns the number of the currently selected program.

The program selected among the 16 possible programs is displayed on the leds using the convention depicted on Figure C.13. Programs can be changed and selected only when a program is not already running. This means that one program must be stopped before another can be selected. The reset command as for it can be triggered any time.

In order to be able to interrupt a program that is being executed at any time, it is important that the main loop of the executed program regularly returns to the main loop of `main.c`. This may seem to add unnecessary programming difficulties. But only in appearance. Indeed, regardless the use of the remote control, our experience and that of other dsPIC users tends to show that you should avoid using too many loops inside other loops. Indeed, this can lead to unexpected resets after 5 or 6 cycles.

`The main_c.c` that can be found here

https://github.com/jrlauper/jrl_epuck/tree/master/jr_demo

shows how to use the `run_remote` function in a concrete way.

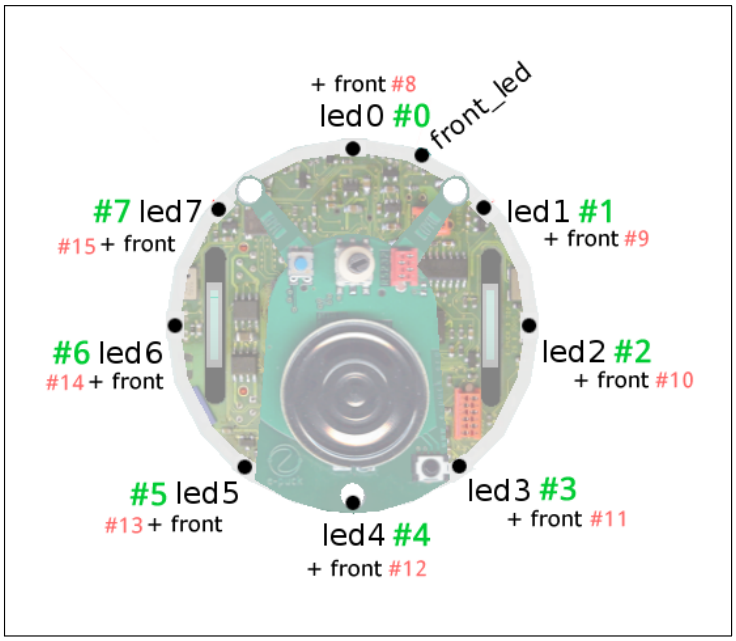


Figure C.13: Remote control leds-selectors correspondence.
#5 = manual selector position number 5.

In the case of our demo, we have made sure that the user can always choose a program, either with the remote control or with the manual selector so that the robots do not become inoperative should the remote control have a problem. See the `main_c.c` file above for more details on how we did it.

D The two complete demos of GCtronic and E-Puck.org

Both demos can be found in *E-Puck Github* (GCtronic, 2019b). There are important overlaps between both, but also specific programs that can be found only in demo 2. Especially important for the current work, a program showing how to use FFT recognition. But this is not the only one! See below.

The fact that there are two different but close demos can easily be overlooked. Indeed, the fact is just alluded in *E-Puck Main Wiki* (GCtronic, 2019c)[1] and not really mentioned in *e-puck official website* (E-Puck.org, 2018). As for *E-Puck Github* (GCtronic, 2019b), it includes two folders: you have two folders:

- one called "DemoGCtronic-complete", which corresponds to the standard firmware and seems to be "The emo" of the e-puck1 (called "Complete Demo 1" below), and
- another quite discrete one just called "demo" (the one called "Complete Demo 2" below).

Note 1: the youtube videos mentioned below are videos posted on youtube by Jonathan Besuchet – responsible of the documentation for the complete demo 2 as I understand it.

Note 2: I specify the name of the developers responsible for the program when I found it mentioned in the code.

D.1 Complete demo 1: GCTronic standard firmware

Code location: GCtronic (2017)
Direct link: DemoGCtronic-complete

Source for the text: *E-Puck Main Wiki* GCtronic (2019c)
Direct link: 2.3 Standard firmware.
(The text below is modified and completed in comparison with the wiki.)

/e-puck-library/program/DemoGCtronic-complete
The number corresponds to selector positions:

0. Shock detection using the accelerometer.
1. Sound source detection I: “simply” tells which microphone records the highest volume by lighting the corresponding led.
2. Wall follower. (uses the IR sensors).
3. Advanced sercom protocol.
4. The robot move in a square path (using either odometry or gyroscope).
5. Sensor “feedback display”. (feedback on IR-sensors, Micros and Camera)

[1] section 2. Software > 2.2. Library. Direct link: Library

6. Camera points to light.
7. Act like the ASL RS232 - I2C translator.
8. Show Ground direction show using accelerometer(Lucas Meier)
   https://youtu.be/a_4eymEv4bs
9. Show the rotation rates of the gyroscope axes.
10. This position is used to work with the gumstix extension.
11. Bluetooth configuration (serial communication).
12. Global test (serial communication).
13. Uart1 to Uart2 transponder.
14. Braitenberg "Lover"/"Follower" vehicle using IR sensors. ((Braitenberg, 1984)).
    https://youtu.be/6DYp5O1cnew
15. Simple dust cleaner behavior.

D.2 Complete demo 2: EPFL e-pucks capabilities demonstration

Code Location: EPFL (2007)
Direct link: demo

Source for the text: main.c comments in demo with arrangements.

Except otherwise mentioned, the developers of program are Michael Bonani and Jonathan Besuchet.

/e-puck-library/program/demo
The number corresponds to selector positions:

0. Show Ground direction show using accelerometer. (Lucas Meier) [= demo1, selector 8.]
   https://youtu.be/a_4eymEv4bs
1. Shock detection using the accelerometer. [= demo1, selector 0.]
2. Sound source detection II. More elaborate than sound source detection I. Plays with sound speed and the fact that the sound wave reaches the three micros at different time depending on the source. The robot turns toward the source.
3. Wall Follower [= demo1, selection 2, with some modifications]
4. Braitenberg "Lover"/"Follower" vehicle using IR sensors. ((Braitenberg, 1984)). [= demo1, selector 14.] https://youtu.be/6DYp5O1cnew
5. Braitenberg "Explorer"/"Avoider" vehicle using the IR sensors. ((Braitenberg, 1984)).
   https://youtu.be/Y-RsvDyUfUE
6. Follow all balls using camera. Example with a black ball : (Jonathan Besuchet, Alain Balleret) https://youtu.be/Pga71leqf1A

7. Follow *red* ball *only* using camera. (Jonathan Besuchet, Alain Balleret)
8. Follows *green* ball *only* using camera. (Jonathan Besuchet, Alain Balleret) `https://youtu.be/FGXBy9FOnmw`
9. Using fft, commands the movement of the e-puck with sound (Michael Bonani, Jonathan Besuchet). `https://youtu.be/bfHFo79uZGY`

E The standard e-puck library and its authors

Epuck Standard Library

Source : https://github.com/gctronic/e-puck-library/tree/master/library

- acc_gyro
  - e_lsm330.c
  - e_lsm330.h
- a_d
  - a_d.mcp
  - advance_ad_scan
    - a_d_adv.mcp
    - e_acc.c
    - e_acc.h
    - e_ad_conv.c
    - e_ad_conv.h
    - e_micro.c
    - e_micro.h
    - e_prox.c
    - e_prox.h
  - e_accelerometer.c
  - e_accelerometer.h
  - e_ad_conv.c
  - e_ad_conv.h
  - e_micro.c
  - e_micro.h
  - e_prox.c
  - e_prox.h
  - e_prox_timer2.c
  - Makefile
- bluetooth
  - bluetooth.mcp
  - e_bluetooth.c
  - e_bluetooth.h
- camera
  - fast_2_timer
    - cam_fast.mcp
    - Doxyfile
    - e_calc_po3030k.c
    - e_calc_po6030k.c
    - e_calc_po8030d.c
    - e_common.c
    - e_interrupt.s
    - e_po3030k.h
    - e_po3030k_registers.c
    - e_po6030k.h
    - e_po6030k_registers.c
    - e_po8030d.h
    - e_po8030d_registers.c
    - e_poxxxx.h
    - e_timers.c
  - slow_3_timer
    - cam_slow.mcp
    - Doxyfile
    - e_calc.c
    - e_po3030k.h
    - e_registers.c
    - e_timers.c
- codec
  - e_common.inc
  - e_const_sound.s
  - e_init_codec_slave.s
  - e_init_dci_master.s
  - e_isr_dci.s
  - e_sound.c
  - e_sound.h
  - e_sub_dci_kickoff.s
  - Makefile
  - README.txt
  - sound.mcp
- contrib
  - LIS_sensors_turret
    - e_devantech.c
    - e_devantech.h
    - e_sensext.c
    - e_sensext.h
    - e_sharp.c
    - e_sharp.h
  - SWIS_com_module
    - ComModule.c
    - ComModule.h
    - Makefile
- Doxyfile
- fft
  - e_fast_copy.s
  - e_fft.c
  - e_fft.h
  - e_fft_utilities.h
  - e_input_signal.c
  - e_subtract_mean.s
  - e_twiddle_factors.c
  - libdsp-coff.a
  - libdsp-elf.a
- folder_tree
- I2C
  - e_I2C_master_module.c
  - e_I2C_master_module.h
  - e_I2C_protocol.c
  - e_I2C_protocol.h
  - I2C.mcp
  - Makefile
- logo.gif
- matlab
  - matlab.c
  - matlab files
    - CloseEpuck.m
    - EpuckFlush.m
    - EpuckGetData.m
    - EpuckSendData.m
    - OpenEpuck.m
  - matlab.h
- motor_led
  - advance_one_timer
    - e_agenda.c
    - e_agenda.h
    - e_led.c
    - e_led.h
    - e_motors.c
    - e_motors.h
    - e_remote_control.c
    - e_remote_control.h
    - fast_agenda
      - e_agenda_fast.c
      - e_agenda_fast.h
      - e_led.c
      - e_led.h
      - e_motors.c
      - e_motors.h
    - Makefile
    - motor_LED_adv.mcp
  - e_epuck_ports.h
  - e_init_port.c
  - e_init_port.h
  - e_led.c
  - e_led.h
  - e_motors.c
  - e_motors.h
  - e_motors_timer3.c
  - Makefile
  - motor_LED.mcp
- uart
  - e_epuck_ports.inc
  - e_init_uart1.s
  - e_init_uart2.s
  - e_uart1_rx_char_aseba.s
  - e_uart1_rx_char.s
  - e_uart1_tx_char.s
  - e_uart2_rx_char.s
  - e_uart2_tx_char.s
  - e_uart_char.h
  - e_uart_char.mcp
  - Makefile
- utility
  - utility.c
  - utility.h

Epuck Standard Library **Authors**

Source : Comments from the code of: https://github.com/gctronic/e-puck-library/tree/master/library

- acc_gyro
  - e_lsm330.c
  - e_lsm330.h
  - *Stefano Morgani*
- a_d
  - a_d.mcp
  - advance_ad_scan
    - a_d_adv.mcp
    - e_acc.c
    - e_acc.h
    - *Jean-Joel Borter*
    - e_ad_conv.c
    - e_ad_conv.h
    - *Francesco Mondada*
    - *Michael Bonani*
    - *Jean-Joel Borter*
    - e_micro.c
    - e_micro.h
    - *Jean-Joel Borter*
    - e_prox.c
    - e_prox.h
    - *Lucas Meier*
  - e_accelerometer.c
  - e_accelerometer.h
  - *Michael Bonani*
  - e_ad_conv.c
  - e_ad_conv.h
  - *Lucas Meier*
  - *Francesco Mondada*
  - *Michael Bonani*
  - e_micro.c
  - e_micro.h
  - *Michael Bonani*
  - e_prox.c
  - e_prox.h
  - *Lucas Meier*
  - *Francesco Mondada*
  - *Michael Bonani*
  - e_prox_timer2.c
  - Makefile
- bluetooth
  - bluetooth.mcp
  - e_bluetooth.c
  - e_bluetooth.h
  - *Michael Bonani*
- camera
  - fast_2_timer
    - cam_fast.mcp
    - Doxyfile
    - e_calc_po3030k.c
    - e_calc_po6030k.c
    - e_calc_po8030d.c
    - e_common.c
    - e_interrupt.s
    - e_po3030k.h
    - *Philippe Rétornaz*
    - e_po3030k_registers.c
    - e_po6030k.h
    - *Philippe Rétornaz*
    - e_po6030k_registers.c
    - e_po8030d.h
    - *Stefano Morgani*
    - e_po8030d_registers.c
    - e_poxxxx.h
    - *Philippe Rétornaz*
    - e_timers.c
  - slow_3_timer
    - cam_slow.mcp
    - Doxyfile
    - e_calc.c
    - e_po3030k.h
    - *Philippe Rétornaz*
    - e_registers.c
    - e_timers.c
- codec
  - e_common.inc
  - e_const_sound.s
  - e_init_codec_slave.s
  - e_init_dci_master.s
  - e_isr_dci.s
  - e_sound.c
  - e_sound.h
  - *Michael Bonani*
  - e_sub_dci_kickoff.s
  - Makefile
  - README.txt
  - sound.mcp
- contrib
  - LIS_sensors_turret
    - e_devantech.c
    - e_devantech.h
    - *Alessandro Ambuehl*
    - e_sensext.c
    - e_sensext.h
    - *Walter Karlen*
    - *Alessandro Ambuehl*
    - e_sharp.c
    - e_sharp.h
    - *Alessandro Ambuehl*
    - *Walter Karlen*
  - SWIS_com_module
    - ComModule.c
    - ComModule.h
    - Makefile
- Doxyfile
- fft
  - e_fast_copy.s
  - *Florian Vaussard*
  - e_fft.c
  - e_fft.h
  - *Jonathan Besuchet*
  - e_fft_utilities.h
  - *Jonathan Besuchet*
  - e_input_signal.c
  - e_subtract_mean.s
  - *Florian Vaussard*
  - e_twiddle_factors.c
  - libdsp-coff.a
  - libdsp-elf.a
- folder_tree
- I2C
  - e_I2C_master_module.c
  - e_I2C_master_module.h
  - *Davis Daidie*
  - e_I2C_protocol.c
  - e_I2C_protocol.h
  - *Davis Daidie*
  - I2C.mcp
  - Makefile
- logo.gif
- matlab
  - matlab.c
  - matlab files
    - CloseEpuck.m
    - EpuckFlush.m
    - EpuckGetData.m
    - EpuckSendData.m
    - OpenEpuck.m
  - matlab.h
  - *Michael Bonani*
  - *Jonathan Besuchet*
- motor_led
  - advance_one_timer
    - e_agenda.c
    - e_agenda.h
    - *Lucas Meier*
    - *Francesco Mondada*
    - e_led.c
    - e_led.h
    - *Francesco Mondada*
    - *Lucas Meier*
    - *Jonathan Besuchet*
    - e_motors.c
    - e_motors.h
    - *Lucas Meier*
    - *Francesco Mondada*
    - e_remote_control.c
    - e_remote_control.h
    - *Valentin Longchamp*
- motor_led (continuation)
  - advance_one_timer (contin.)
    - fast_agenda
      - e_agenda_fast.c
      - e_agenda_fast.h
      - *Julien Hubert*
      - e_led.c
      - e_led.h
      - *Francesco Mondada*
      - e_motors.c
      - e_motors.h
      - *Francesco Mondada*
      - *Lucas Meier*
    - Makefile
    - motor_LED_adv.mcp
  - e_epuck_ports.h
  - *Michael Bonani*
  - *Francesco Mondada*
  - *Davis Daidie*
  - e_init_port.c
  - e_init_port.h
  - *Michael Bonani*
  - *Francesco Mondada*
  - *Davis Daidie*
  - e_led.c
  - e_led.h
  - *Michael Bonani*
  - *Francesco Mondada*
  - *Davis Daidie*
  - e_motors.c
  - e_motors.h
  - *Lucas Meier*
  - *Francesco Mondada*
  - *Michael Bonani*
  - e_motors_timer3.c
  - Makefile
  - motor_LED.mcp
- uart
  - e_epuck_ports.inc
  - *Michael Bonani*
  - e_init_uart1.s
  - *Michael Bonani*
  - *Stephane Magnenat*
  - e_init_uart2.s
  - *Michael Bonani*
  - e_uart1_rx_char_aseba.s
  - *Michael Bonani*
  - *Stephane Magnenat*
  - *Davis Daidie*
  - e_uart1_rx_char.s
  - *Michael Bonani*
  - *Stephane Magnenat*
  - *Davis Daidie*
  - e_uart1_tx_char.s
  - *Michael Bonani*
  - *Davis Daidie*
  - e_uart2_rx_char.s
  - *Michael Bonani*
  - *Stephane Magnenat*
  - *Davis Daidie*
  - e_uart2_tx_char.s
  - *Michael Bonani*
  - *Davis Daidie*
  - e_uart_char.h
  - *Michael Bonani*
  - e_uart_char.mcp
  - Makefile
- utility
  - utility.c
  - utility.h

References

Theoretical Papers

BRAITENBERG, Valentino (1984). *Vehicles. Experiments in Synthetic Psychology*. Cambridge, Mass.: MIT Press.

MARIS, Marinus and René te BOECKHORST (1996). "Exploiting Physical Constraints: Heap Formation Through Behavioral Error in a Group of Robots". In: *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems. IROS '96*. Vol. 3, 1655–1660 vol.3. DOI: 10.1109/IROS.1996.569034.

MICHAUD, François and Monica NICOLESCU (2016). "Behavior-based systems". In: *Springer Handbook of Robotics*. Springer, pp. 307–328.

MIROLLO, Renato E. and Steven H. STROGATZ (1990). "Synchronization of Pulse-Coupled Biological Oscillators". In: *SIAM Journal on Applied Mathematics* 50.6, pp. 1645–1662. ISSN: 00361399. URL: http://www.jstor.org/stable/2101911.

NEMBRINI, Julien (2005). "Minimalist Coherent Swarming of Wireless Networked Autonomous Mobile Robots". PhD thesis. Bristol. URL: http://infoscience.epfl.ch/record/50997 (visited on 12/18/2019).

NEMBRINI, Julien, Alan WINFIELD, and Chris MELHUISH (2002). "Minimalist Coherent Swarming of Wireless Networked Autonomous Mobile Robots". In: *Proceedings of the Seventh International Conference on Simulation of Adaptive Behavior on From Animals to Animats*. ICSAB. Cambridge, MA, USA: MIT Press, pp. 373–382. ISBN: 0-262-58217-1. URL: http://dl.acm.org/citation.cfm?id=773380.773437.

NEMEC, Dušan et al. (2017). "Mutual acoustic identification in the swarm of e-puck robots". In: *International Journal of Advanced Robotic Systems* 14.3. URL: https://doi.org/10.1177/1729881417710794.

STØY, Kasper (2001). "Using Situated Communication in Distributed Autonomous Mobile Robotics". In: *Proceedings of the Seventh Scandinavian Conference on Artificial Intelligence*. SCAI '01. IOS Press, pp. 44–52.

WIKIPEDIA, Contributors (2019b). "RC-5". In: *Wikipedia, The Free Encyclopedia*. URL: https://en.wikipedia.org/w/index.php?title=RC-5&oldid=916587046 (visited on 02/21/2020).

— (2019c). "Waveform". In: *Wikipedia, The Free Encyclopedia*. URL: https://en.wikipedia.org/w/index.php?title=Waveform&oldid=895659604 (visited on 02/20/2020).

— (2020). "Behavior-based robotics". In: *Wikipedia, The Free Encyclopedia*. URL: https://en.wikipedia.org/w/index.php?title=Behavior-based_robotics&oldid=965044481 (visited on 09/24/2020).

E-Puck Information and Description

CYBERBOTICS LTD (2015). *E-Puck. Mini Mobile Robot from EPFL*. Brochure.
The original link https://www.cyberbotics.com/e-puck.pdf doesn't work anymore; an archive version can be found here: https://web.archive.org/web/20170710203516/https://www.cyberbotics.com/e-puck.pdf.

CYBERBOTICS LTD (2020). *Webots User Guide R2020a revision 2 > Robots > e-puck*. URL: https://cyberbotics.com/doc/guide/epuck (visited on 02/15/2020).

E-PUCK.ORG (2018). *E-Puck Website. EPFL Education Robot (e-puck official site)*. Last Page Update 28 February 2018. URL: http://www.e-puck.org/index.php (visited on 03/07/2019).

GCTRONIC (2006). *E-Puck. EPFL Educational and Research Mini Mobile Robot*. Brochure. URL: https://www.gctronic.com/files/flyere-puck.pdf (visited on 01/01/2020).

— (2008). *E-Puck Mini Doc (Ver. 1.0). EPFL Educational and Research Mini Mobile Robot*. Brochure. URL: https://www.gctronic.com/files/miniDocWeb.pdf (visited on 01/01/2020).

— (2019a). *GCtronic Shop*. URL: https://www.gctronic.com/shop.php (visited on 01/01/2020).

— (2019c). *E-Puck Main Wiki*. URL: https://www.gctronic.com/doc/index.php/E-Puck (visited on 03/07/2019).

MICROCHIP TECHNOLOGY INC. (2011). *dsPIC30F6011A/6012A/6013A/6014A Data Sheet. High-Performance, 16-bit Digital Signal Controllers*. URL: http://ww1.microchip.com/downloads/en/DeviceDoc/70143E.pdf (visited on 04/01/2019).

— (2019). *dsPIC30F6014A. Official Web Page*. URL: https://www.microchip.com/wwwproducts/en/dsPIC30F6014A (visited on 04/01/2019).

MONDADA, Francesco et al. (2009). "The e-puck, a Robot Designed for Education in Engineering". In: *Proceedings of the 9th Conference on Autonomous Robot Systems and Competitions* 1.1, pp. 59–65. URL: http://infoscience.epfl.ch/record/135236.

WIKIPEDIA, Contributors (2019a). "E-puck mobile robot". In: *Wikipedia, The Free Encyclopedia*. URL: https://en.wikipedia.org/w/index.php?title=E-puck_mobile_robot&oldid=910935510 (visited on 12/20/2019).

E-Puck Embedded Programming

CAMPO, Alexandre, Alvaro GUTIERREZ, and Valentin LONGCHAMP (2008). *libIrcom 1.0.7. IR Communication Library for E-Pucks*. (absent from GCtronic Github). Last Page Update 03 December 2010. URL: http://www.e-puck.org/index.php?option=com_content&view=article&id=32&Itemid=28 (visited on 03/07/2019).

CAPRARI, Gilles (2009). *E-Puck Sound Tranformation Script (in MATLAB)*.
The original mail within which the script is mentioned: https://www.mail-archive.com/e-puck-user@gna.org/msg00107.html.
The original url of the script (not working anymore): http://gctronic.com/files/e-puck_wav2s_matlabScript.txt.
Its archive version on Wayback Machine: https://web.archive.org/web/2019*/http://gctronic.com/files/e-puck_wav2s_matlabScript.txt
A copy of the script can be found in the github of this work here: https://github.com/jrlauper/jrl_epuck/tree/master/Matlab_Sound_Script
. (Visited on 05/13/2019).

EPFL (2007). *Program to Demonstrate All The E-Puck's Capabilities. Version 2.0. "E-Puck Complete Demo 2"*.
The content the demo can be found in Appendix D. URL: https://github.com/gctronic/e-puck-library/tree/master/program/demo (visited on 05/21/2019).

GCTRONIC (2007). *E-Puck (Standard) Library*.
Several versions of the e-puck standard library are available between e-puck.org and gctronic.com with different dates. Here, is the most recent version I found on the Github of GCtronic. It is dated of 2017 on github, but most of the modules were created between 2004 and 2007 according to the comments.
The content of library displayed as a tree is available in Appendix E.
Another tree with the name of all the contributors to the different modules can be found in

the same index. URL: https://github.com/gctronic/e-puck-library/tree/master/library (visited on 03/11/2019).
— (2009). *Basic Demos*. (absent from GCtronic Github)
For GCtronic: Stefano Morgani and Gilles Caprari;
Description page https://www.gctronic.com/doc/index.php/E-Puck#Basic_demos;
Code download page: http://www.gctronic.com/doc/images/BasicDemos.zip". (Visited on 03/24/2019).
— (2011). *Audio Recording*. (absent from GCtronic Github)
For GCtronic: Stefano Morgani;
Description page https://www.gctronic.com/doc/index.php/E-Puck#Audio_recording;
Code download page: http://projects.gctronic.com/E-Puck/DemoGCtronic-recording/DemoGCtronic-recording.zip; (visited on 04/11/2019).
— (2016). *E-Puck Library Documentation (Doxygen)*. Version 1.0. URL: http://projects.gctronic.com/E-Puck/e-puck-library.pdf (visited on 01/01/2020).
— (2017). *Standard Firmware. "E-Puck Complete Demo 1"*.
The content the demo can be found in Appendix Appendix D.
Several versions of this demo are available between e-puck.org and gctronic.com with different dates. Here, this is the most recent version I found on GCTRONIC (2019b).
Description page: https://www.gctronic.com/doc/index.php/E-Puck#Standard_firmware;
Code download page: https://github.com/gctronic/e-puck-library/tree/master/program/DemoGCtronic-complete.
— (2019b). *E-Puck Github*.
Includes the standard library, 'complete demo 1', 'complete demo 2', and various tools (e-puck monitor and several bootloaders). (The content of the standard library as its authors is given in Appendix E.
A description of the programs included in the two demos as well as their authors name can be found in Appendix D includes .) URL: https://github.com/gctronic/e-puck-library (visited on 04/10/2019).
MICHEL, Olivier and Stephane MAGNENAT (2007). *Epuck Bluetooth Uploader (for Linux)*. Uses libbluetooth-dev. URL: https://github.com/gctronic/e-puck-library/tree/master/tool/bootloader/computer_side/linux-libbluetooth (visited on 03/19/2019).
MICROCHIP TECHNOLOGY INC. (2019). *dsPIC30F6014A. Official Web Page*. URL: https://www.microchip.com/wwwproducts/en/dsPIC30F6014A (visited on 04/01/2019).
MONDADA, Francesco and Michael BONANI (2006). *Tutorial for Programming the E-Puck Robot Using the Bootloader via Bluetooth*. URL: http://www.e-puck.org/index.php?option=com_phocadownload&view=category&id=5:tutorials&Itemid=38 (visited on 03/06/2019).
PRINZ, Peter and Ulla KIRCH-PRINZ (2002). *C Pocket Reference*. O'Reilly Media, Inc.

E-Puck2

GCTRONIC (2018). *E-Puck2. EPFL Educational and Research Mini Mobile Robot*. Brochure. URL: http://projects.gctronic.com/epuck2/e-puck2-flyer.pdf (visited on 07/15/2020).
— (2020a). *E-Puck2. EPFL educational and research mini mobile robot VERSION 2*. WebPage. URL: https://www.gctronic.com/e-puck2.php (visited on 07/15/2020).

GCTRONIC (2020b). *E-Puck2 Main Wiki*. URL: https://www.gctronic.com/doc/index.php/e-puck2 (visited on 07/15/2020).

STMICROELECTRONICS (2016). *STM32F405xx, STM32F407xx Datasheet*. URL: http://projects.gctronic.com/epuck2/doc/STM32F407xx_datasheet.pdf (visited on 07/16/2020).