ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

Realistic Simulation of Real-Time Embedded Systems

Grégory Mermoud

School of Architecture, Civil and Environmental Engineering

EPFL, SS 2008-2009

http://disal.epfl.ch/teaching/embedded_systems/

Outline

- A few words about simulation!
- Webots
  - Generalities
  - Sensors/actuators
  - Webots API
- Programming embedded systems
  - Buffers
  - Timing
  - An example using the microphone in Webots and on the e-puck robot

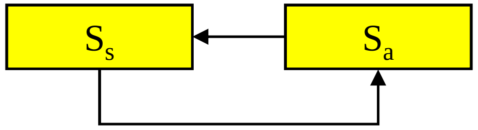
ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

Simulation: why?

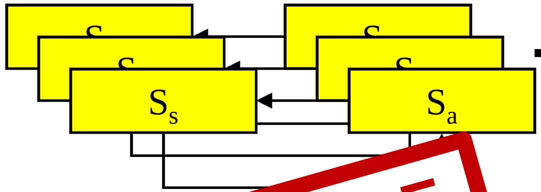
- Hardware prototyping is time-consuming and expensive
- Real commercial robots are expensive
- Quickly change the experimental setup
- Often easier for monitoring experiments (and evaluating specific metrics)
- Sometimes faster than real-time
  - Numerical optimization methods (genetic algorithms, particle swarm optimizations, etc.)
  - Enable systematic search of the parameter space

Simulation: one piece of the puzzle

$$\frac{dN_n(t)}{dt} = \sum_{n'} W(n|n',t)N'_n(t) - \sum_{n'} W(n'|n,t)N_n(t)$$



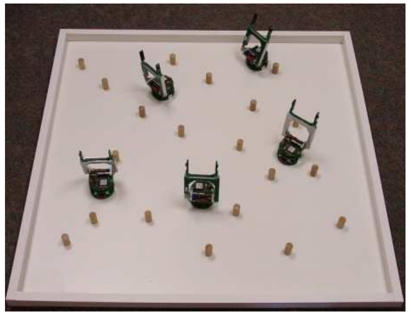
Macroscopic: rate equations, mean field approach, whole swarm



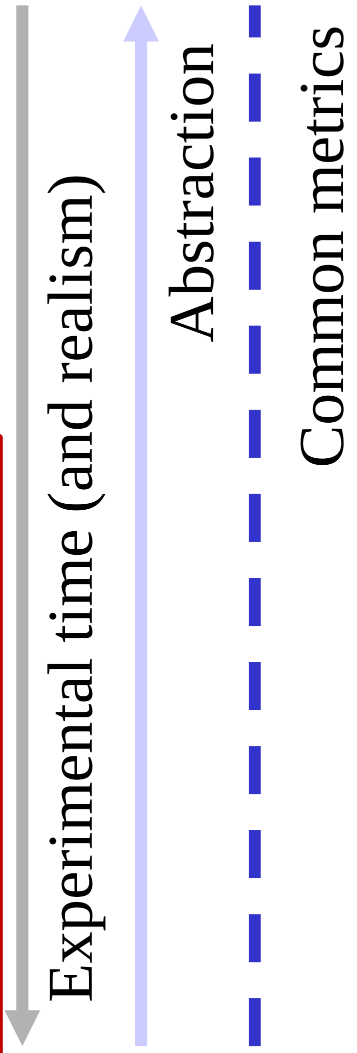
Microscopic – Agent-based: multi-agent models, only relevant robot feature captured, 1 agent = 1 robot

THIS COURSE

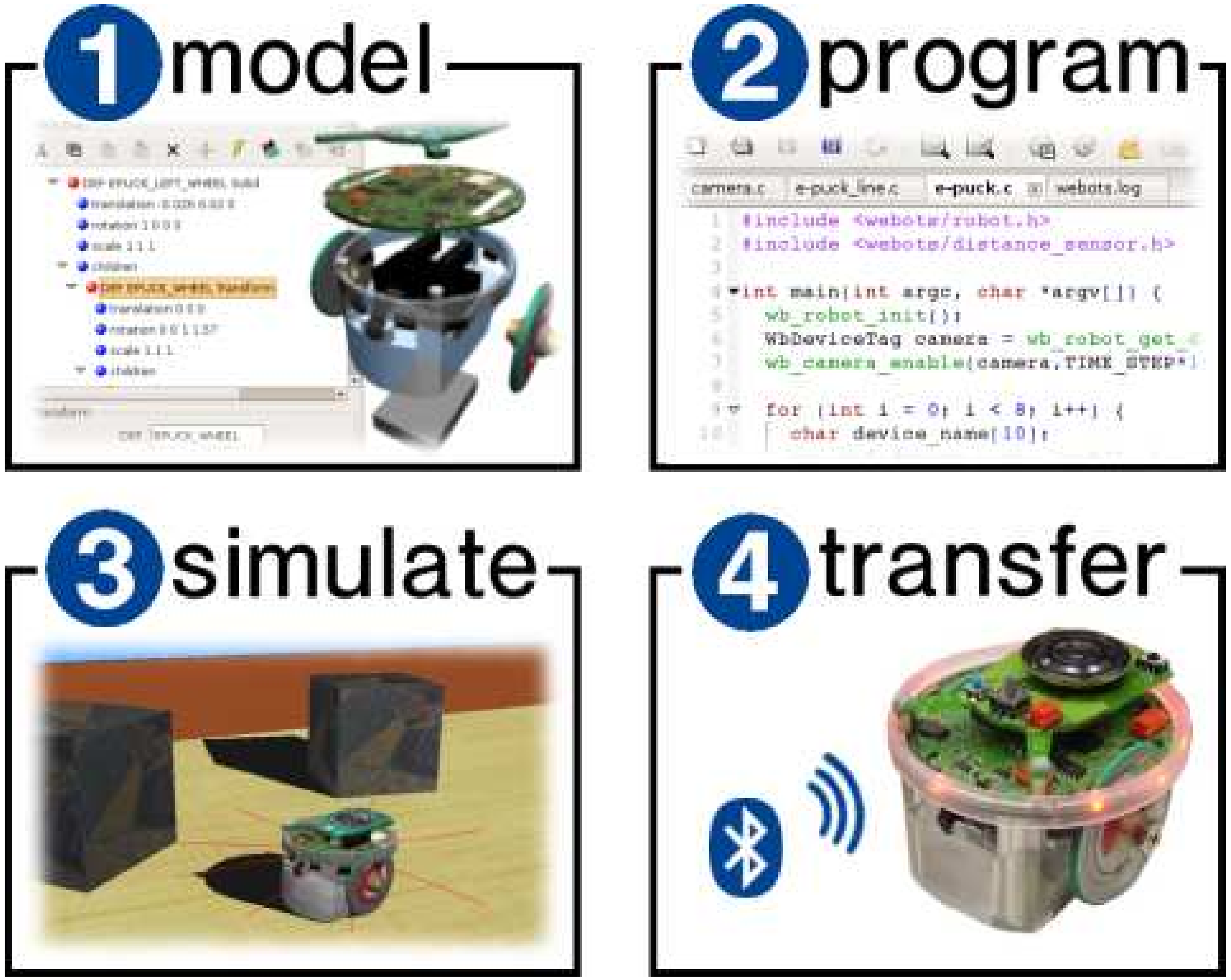
Microscopic – Module-based: intra-robot (e.g., S&A, transceiver) and environment (e.g., physics) details reproduced faithfully



Target system (physical reality): info on controller, S&A, comunication, morphology and environmental features



Webots realistic simulator



```
#include <webots/robot.h>
#include <webots/distance_sensor.h>

int main(int argc, char *argv[]) {
  wb_robot_init();
  WbDeviceTag camera = wb_robot_get_
  wb_camera_enable(camera,TIME_STEP*)

  for (int i = 0; i < 8; i++) {
    char device_name[10];
```

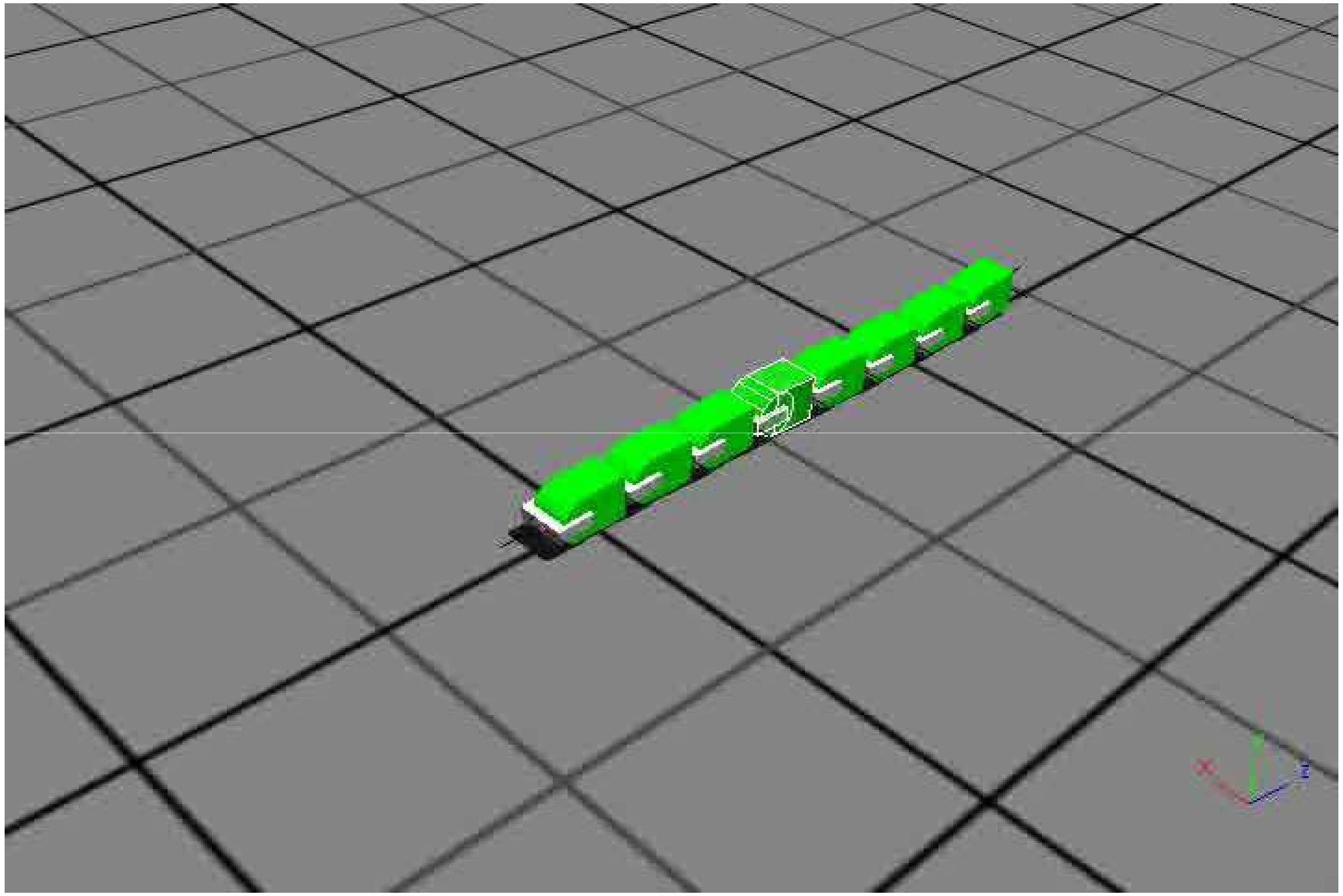
In this course, we will focus on steps **2** and **3** only.

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

Webots features

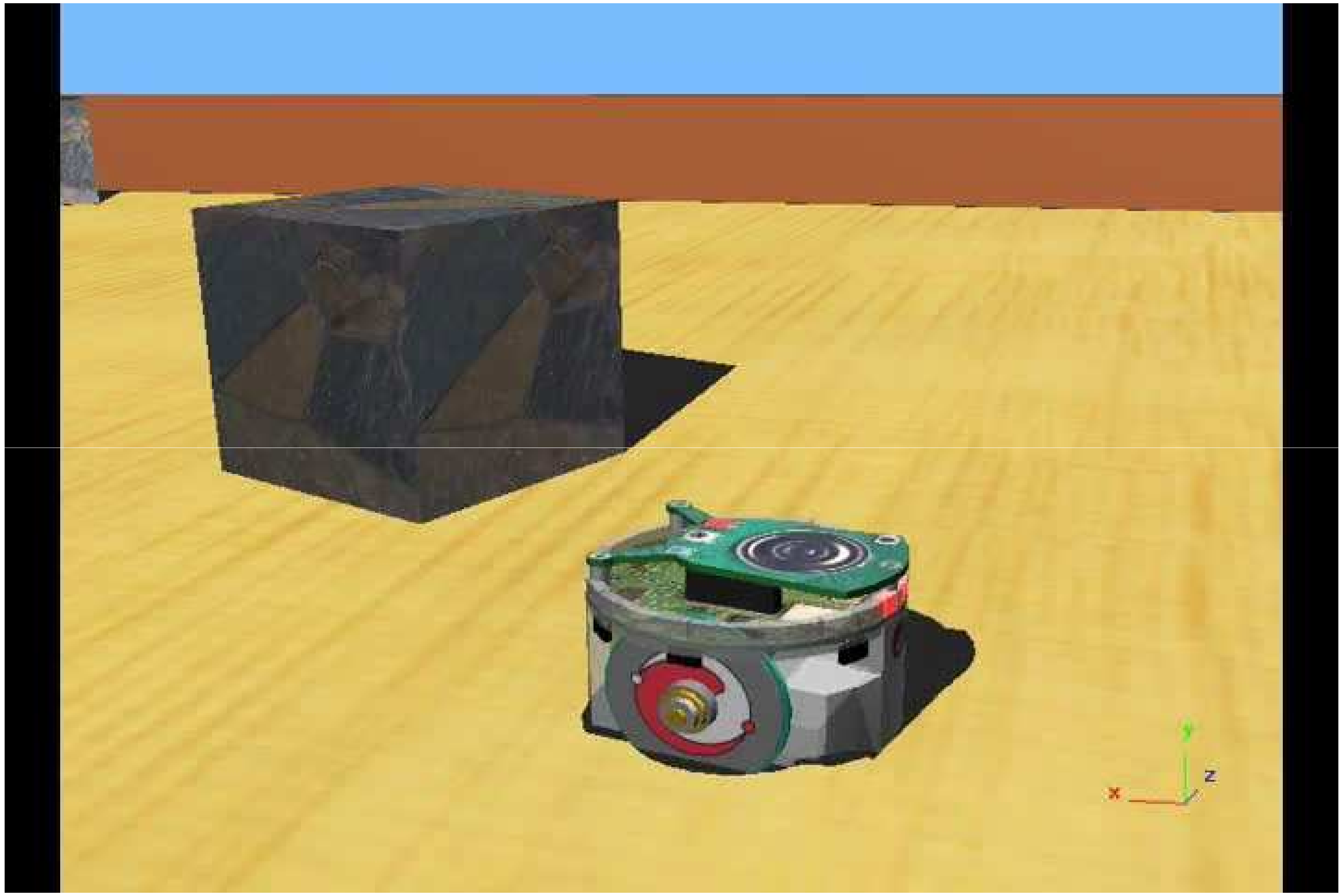
- Realistic (physics-based) robotics simulator
- Fast prototyping
- Experiment without real hardware
- Can *usually* run faster than real-time
- Available sensors: distance sensors, light sensors, cameras, accelerometers, touch sensors, position sensors, GPSs, receivers, force sensors, etc.
- Available actuators: servo-motors, grippers, LEDs, emitters, etc.

How does it look like?



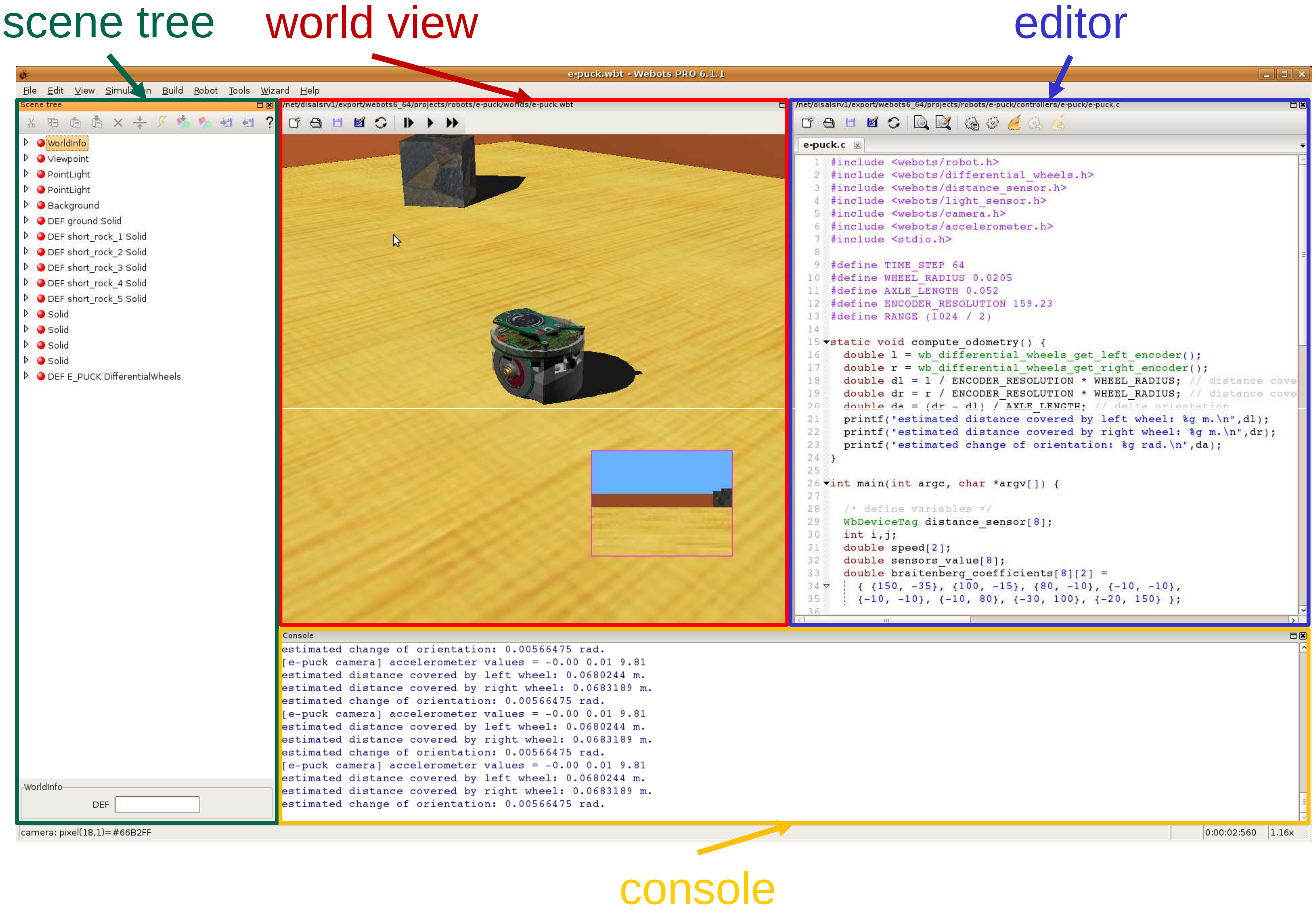
YaMoR modular robot performing different gaits and achieving a reconfiguration

How does it look like?



e-puck mobile robot performing obstacle avoidance and line following

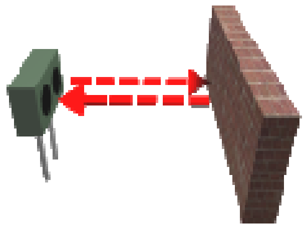
Webots GUI

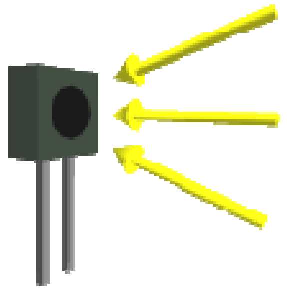

scene tree
world view
editor
console
e-puck.wbt - Webots PRO 6.1.1
File Edit View Simulation Build Robot Tools Wizard Help
Scene tree
WorldInfo
Viewpoint
PointLight
PointLight
Background
DEF ground Solid
DEF short_rock_1 Solid
DEF short_rock_2 Solid
DEF short_rock_3 Solid
DEF short_rock_4 Solid
DEF short_rock_5 Solid
Solid
Solid
Solid
Solid
DEF E_PUCK DifferentialWheels
WorldInfo
DEF
/net/disalsrv1/export/webots6_64/projects/robots/e-puck/worlds/e-puck.wbt
/net/disalsrv1/export/webots6_64/projects/robots/e-puck/controllers/e-puck/e-puck.c
e-puck.c
Console
camera: pixel(18,1)=#66B2FF
0:00:02:560
1.16x

```c
#include <webots/robot.h>
#include <webots/differential_wheels.h>
#include <webots/distance_sensor.h>
#include <webots/light_sensor.h>
#include <webots/camera.h>
#include <webots/accelerometer.h>
#include <stdio.h>

#define TIME_STEP 64
#define WHEEL_RADIUS 0.0205
#define AXLE_LENGTH 0.052
#define ENCODER_RESOLUTION 159.23
#define RANGE (1024 / 2)

static void compute_odometry() {
  double l = wb_differential_wheels_get_left_encoder();
  double r = wb_differential_wheels_get_right_encoder();
  double dl = l / ENCODER_RESOLUTION * WHEEL_RADIUS; // distance cove
  double dr = r / ENCODER_RESOLUTION * WHEEL_RADIUS; // distance cove
  double da = (dr - dl) / AXLE_LENGTH; // delta orientation
  printf("estimated distance covered by left wheel: %g m.\n",dl);
  printf("estimated distance covered by right wheel: %g m.\n",dr);
  printf("estimated change of orientation: %g rad.\n",da);
}

int main(int argc, char *argv[]) {

  /* define variables */
  WbDeviceTag distance_sensor[8];
  int i,j;
  double speed[2];
  double sensors_value[8];
  double braitenberg_coefficients[8][2] =
    { {150, -35}, {100, -15}, {80, -10}, {-10, -10},
    {-10, -10}, {-10, 80}, {-30, 100}, {-20, 150} };
```

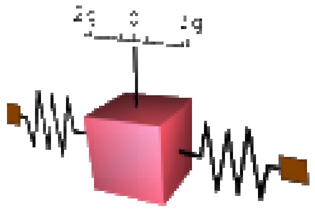
```
estimated change of orientation: 0.00566475 rad.
[e-puck camera] accelerometer values = -0.00 0.01 9.81
estimated distance covered by left wheel: 0.0680244 m.
estimated distance covered by right wheel: 0.0683189 m.
estimated change of orientation: 0.00566475 rad.
[e-puck camera] accelerometer values = -0.00 0.01 9.81
estimated distance covered by left wheel: 0.0680244 m.
estimated distance covered by right wheel: 0.0683189 m.
estimated change of orientation: 0.00566475 rad.
[e-puck camera] accelerometer values = -0.00 0.01 9.81
estimated distance covered by left wheel: 0.0680244 m.
estimated distance covered by right wheel: 0.0683189 m.
estimated change of orientation: 0.00566475 rad.
```

Modeled sensors

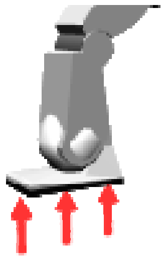

distance sensor

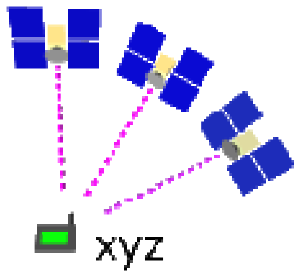

light sensor

camera



accelerometer


touch/pressure sensor



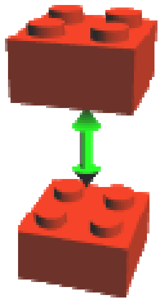
GPS

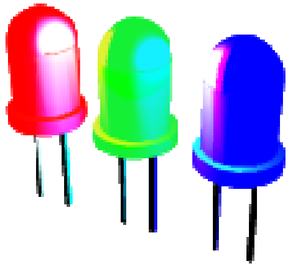
...and battery sensor, torque sensor, etc.

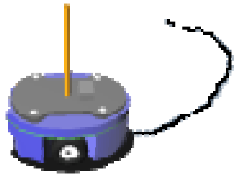
Modeled actuators

servo (rotational / linear)

gripper

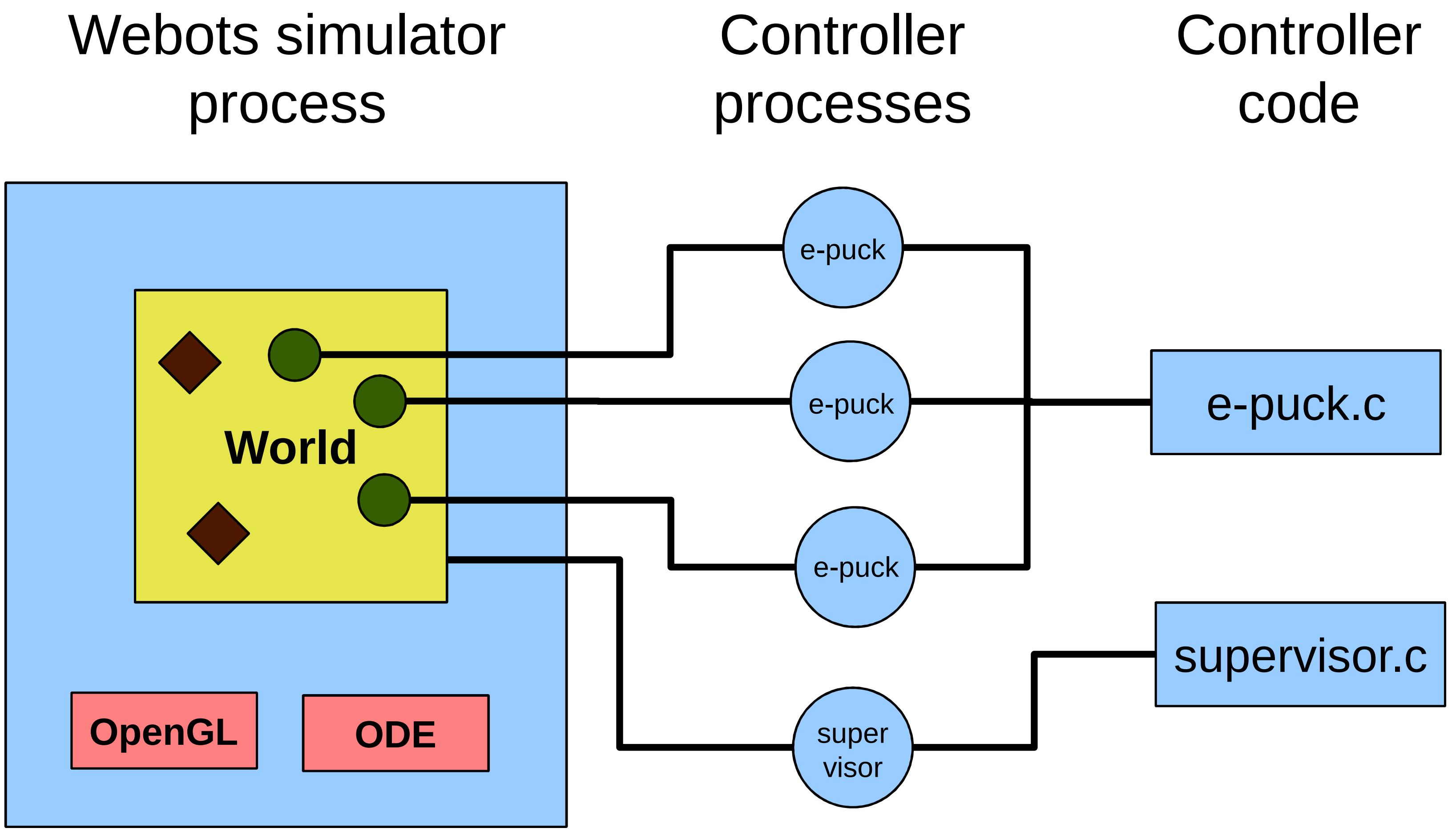

connector
(docking systems)


LED


pen

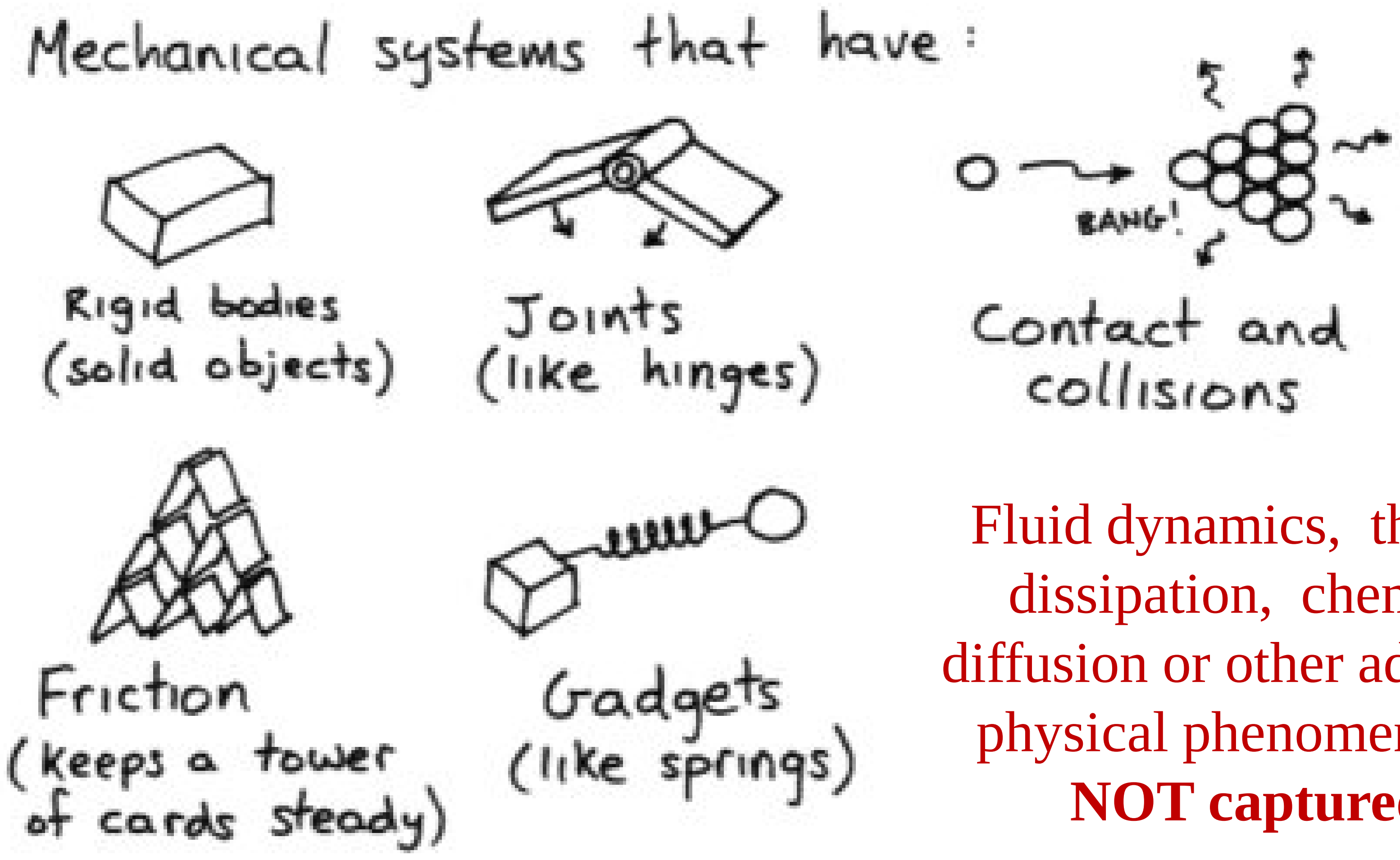
... and emitter & receiver, etc.

Webots principles



The more robots, the slower the simulation!

(Newtonian) physics-based simulation (ODE)


Mechanical systems that have:
Rigid bodies (solid objects)
Joints (like hinges)
BANG!
Contact and collisions
Friction (keeps a tower of cards steady)
Gadgets (like springs)

Fluid dynamics, thermal dissipation, chemical diffusion or other advanced physical phenomena **are NOT captured**!

http://www.ode.org/

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

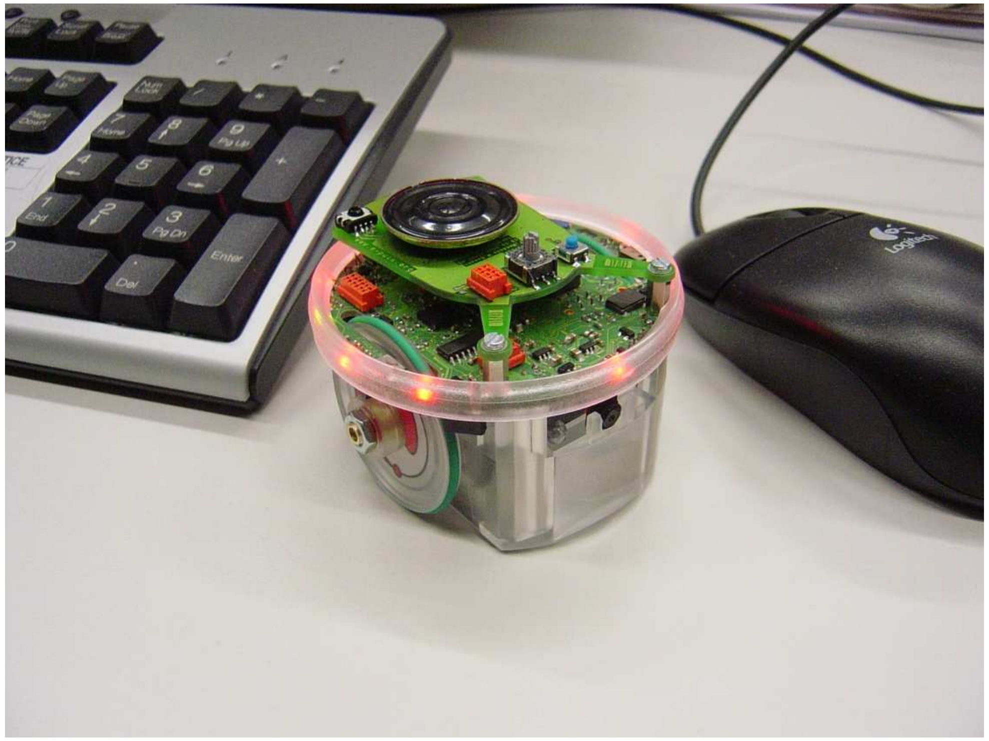
Supervisor

- A **supervisor** is a program that controls a world and its robots.
- For convenience, it is represented as a robot without any wheels, driven by a controller with **extended capabilities** that supervises the whole world.
- The supervisor can read/write any field of any node in the scene tree in order to:
  - move or rotate any object in the scene
  - simulate changing environmental conditions
  - track the position of a robot
- The supervisor can also take a snapshot of the scene or create movies.

The e-puck robot

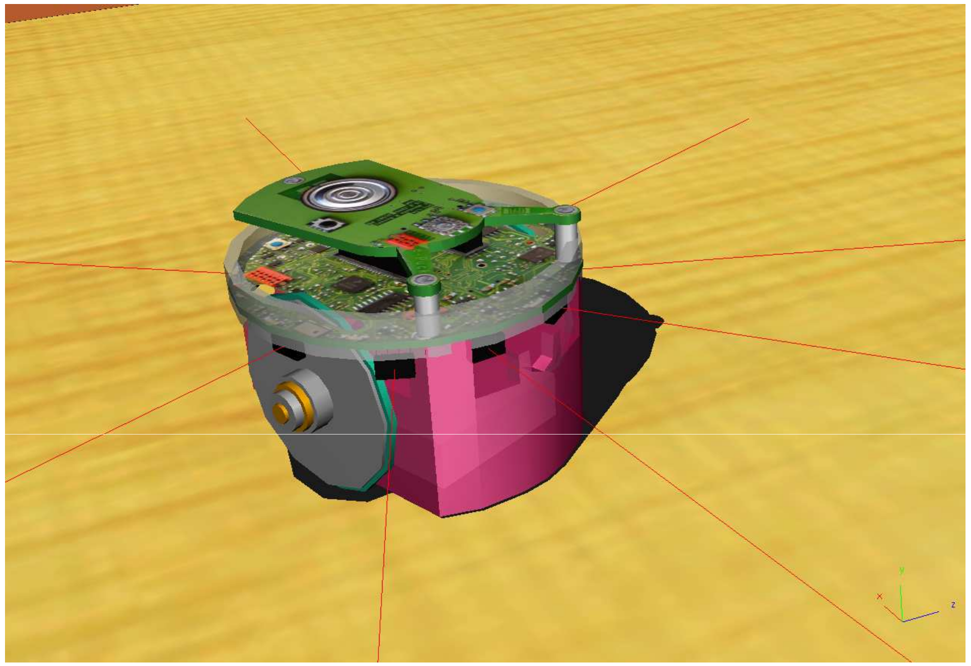
Main features:

- Cylindrical, Ø 70mm
- dsPIC processor
- Two stepper motors
- Ring of LEDs
- Many sensors:
  - ✓ Camera
  - ✓ 3 microphones
  - ✓ 1 loudspeaker
  - ✓ IR proximity
  - ✓ 3D accelerometer
- Li-ion accumulator
- Bluetooth wireless communication
- Open hardware (and software)



Real and simulated e-puck

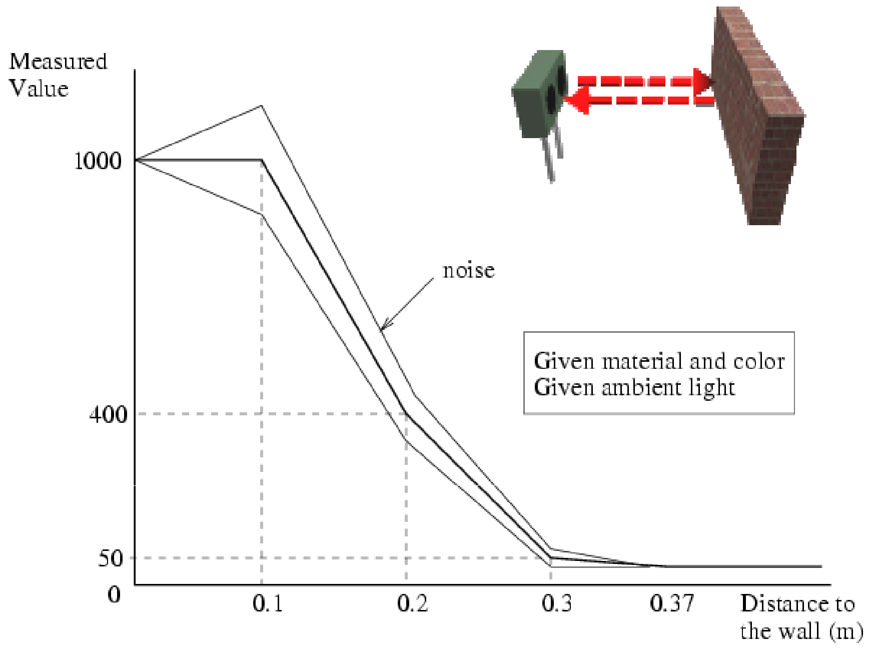
Real e-Puck


Simulated e-puck (Webots)

- sensor- and actuator-based
- noise, nonlinearities of S&A reproduced
- kinematic (e.g., speed, position) and dynamic (e.g., mass, forces, friction)

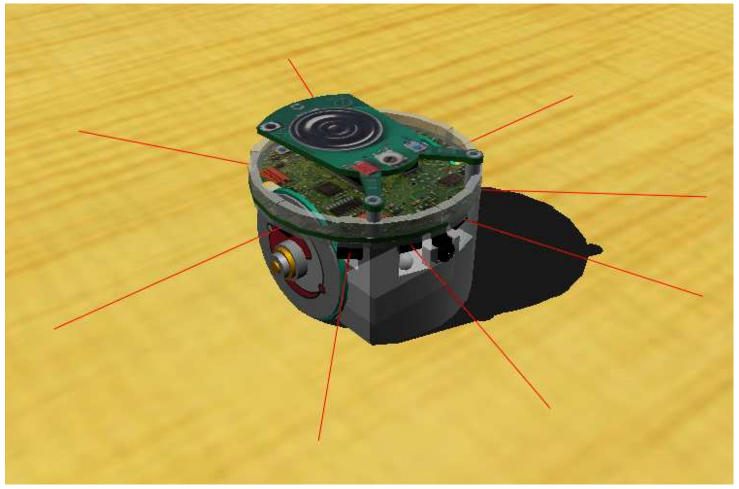
Modeling sensors

- Capture **non-linearities** and **noise** of sensors.
- However, **calibration** is often approximative.
- Most often, sensor response is defined by a lookup table (here a proximity sensor):



```
lookupTable [ 0     1000  0,
              0.1   1000  0.1,
              0.2    400  0.1,
              0.3     50  0.1,
              0.37    30  0    ]
```

distance   value   noise



A typical Webots controller

```c
#include <webots/robot.h>
#include <webots/differential_wheels.h>
#include <webots/distance_sensor.h>

#define TIME_STEP 32

int main(int argc, const char *argv[]) {
  double speed[2] = {200.0,200.0};
  double sensor_value;
  WbDeviceTag sensor;

  // initialize webots
  wb_robot_init();

  // find distance sensors
  sensor = wb_robot_get_device("ps0");
  wb_distance_sensor_enable(sensor, TIME_STEP);

  // main loop
  for (;;) {
    sensor_value = wb_distance_sensor_get_value(sensor);
    if (sensor_value > 500.0) {
      speed[0] = 0.0; speed[1] = 0.0;
    }

    // set the motors speed
    wb_differential_wheels_set_speed(speed[0], speed[1]);

    // simulation step
    wb_robot_step(TIME_STEP);
  }
}
```

Includes for accessing Webots API

Define simulation step in milliseconds

Define data structures

Initialize Webots

Get and enable different devices (sensors and actuators)

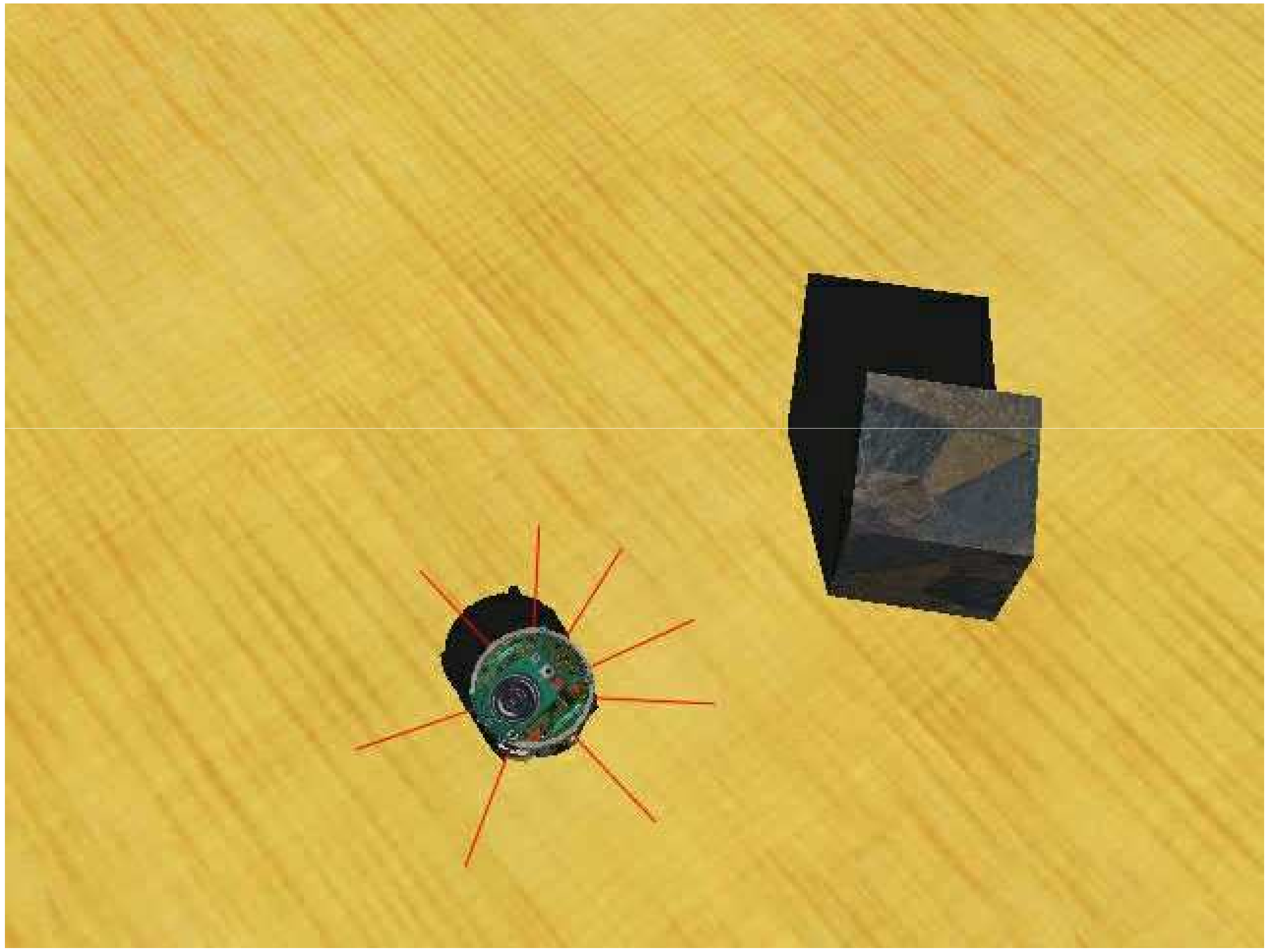
Read sensor values

Infinite loop (robot behavior must be coded here)

Update actuators

Perform one simulation step

How does it look like?



Webots controllers

- A Webots controller is a **C program** (Webots also supports C++, Java, Python, and even Matlab).
- Therefore, everything you can do in conventional C programs, you can do in a Webots controller.
- You must just keep in mind that your controller **will eventually run on a robot**.
- Webots **does not** simulate the microcontroller of your robot:
  - Your controller will run **much slower** on the real robot than it does in Webots .
  - Your memory will be **much more limited** on the real robot than in Webots.
- In the first case, the behavior of the real robot will be very different from that of the simulated robot. In the second case, you will not be able to compile your controller!

Webots API

- **Definition:** an application programming interface (API) is a set of functions, procedures, methods or classes that an operating system, library or service provides to support requests made by computer programs.
- Webots provides a lot of such functions that allow you to interact with the different devices of your robot:

```
#include <webots/distance_sensor.h>

void wb_distance_sensor_enable(WbDeviceTag tag, int ms);
void wb_distance_sensor_disable(WbDeviceTag tag);
double wb_distance_sensor_get_value(WbDeviceTag tag);
```

- You can find all of them in the Webots Reference Manual available at http://www.cyberbotics.com/cdrom/common/doc/webots/reference!
- The principle of the API is that you must always enable a sensor before using them (pretty much like on a real robot)!
- **The Webots API is NOT available on the real robot.** This means that you will need to modify your controller before transferring it to the real robot. In the **specific** case of the e-puck, cross-compilation is available, tough.

Webots API: camera

- If you want to use a camera, you need to include the following header file:

```
#include <webots/camera.h>
```

- The first thing to do next is to get the device by using the standard function:

```
WbDeviceTag camera = wb_robot_get_device("cam");
```

device tag

name of the device (see scene tree)

- Then, you need to enable it before using it (and possibly disable it if you no longer need it):

device tag

1/refresh rate in kHz

```
void wb_camera_enable(WbDeviceTag tag, int ms);
void wb_camera_disable(WbDeviceTag tag);
```

- You also have a bunch of auxiliary functions that might come in handy at some point:

```
int wb_camera_get_width(WbDeviceTag tag);
int wb_camera_get_height(WbDeviceTag tag);
```

Webots API: camera

- Then, you can of course get an image using the following functions:

```
unsigned char *wb_camera_get_image(WbDeviceTag tag);
unsigned char wb_camera_image_get_red(const unsigned char *image,
int width, int x, int y);
unsigned char wb_camera_image_get_green(const unsigned char *image,
int width, int x, int y);
unsigned char wb_camera_image_get_blue(const unsigned char *image,
int width, int x, int y);
unsigned char wb_camera_image_get_grey(const unsigned char *image,
int width, int x, int y);
```

- Here is an example of usage of these functions:

```
const unsigned char *image = wb_camera_get_image(camera);
for (int x = 0; x < image_width; x++) {
  for (int y = 0; y < image_height; y++) {
    int r = wb_camera_image_get_red(image, image_width, x, y);
    int g = wb_camera_image_get_green(image, image_width, x, y);
    int b = wb_camera_image_get_blue(image, image_width, x, y);
    printf("red=%d, green=%d, blue=%d", r, g, b);
  }
}
```

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

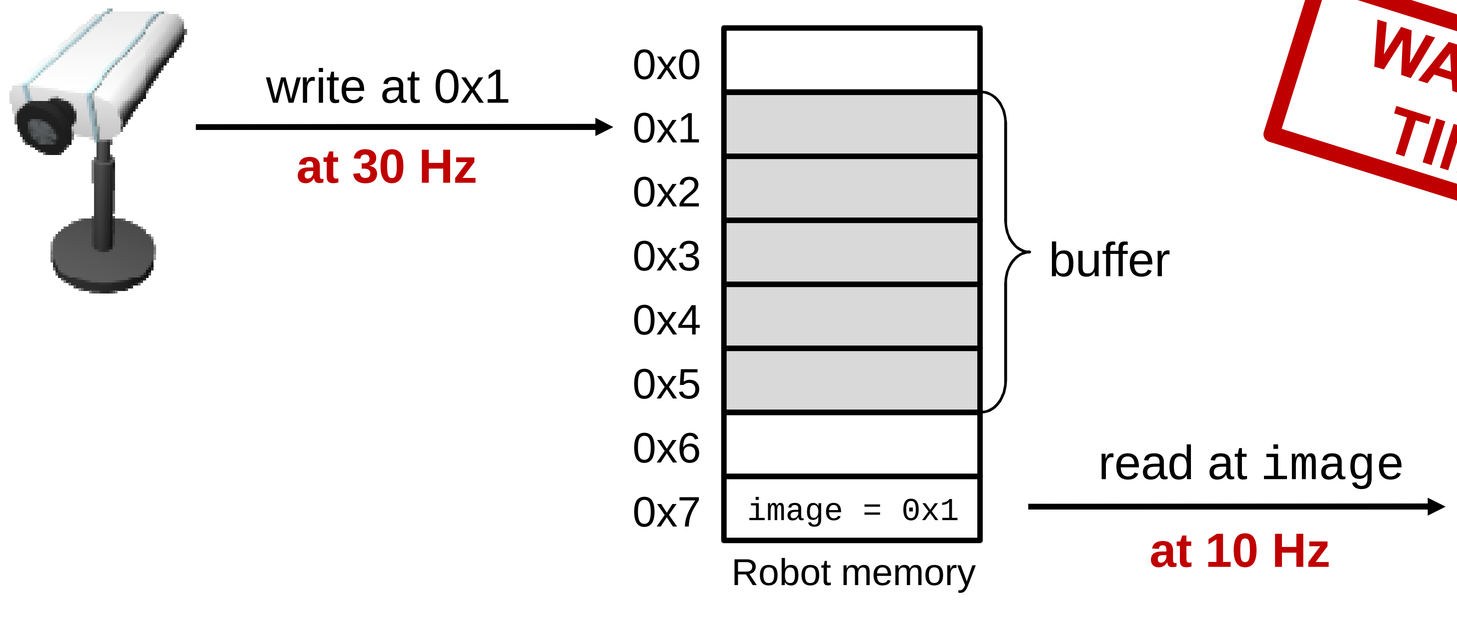
The concept of buffer

- A **buffer** is a region of memory used to temporarily hold data while it is being moved from one place to another.
- Typically, the data is stored in a buffer as it is retrieved from an input device (such as a sensor) or just before it is sent to an output device (such as an actuator).
- The device writes/reads in the buffer independently of the controller. Therefore, to read the device, you just need to read the buffer, using **a pointer**:

```
const unsigned char *image = wb_camera_get_image(camera);
for (int x = 0; x < image_width; x++) {
  for (int y = 0; y < image_height; y++) {
    int r = wb_camera_image_get_red(image, image_width, x, y);
    int g = wb_camera_image_get_green(image, image_width, x, y);
    int b = wb_camera_image_get_blue(image, image_width, x, y);
    printf("red=%d, green=%d, blue=%d", r, g, b);
  }
}
```

image buffer

Buffer mechanism



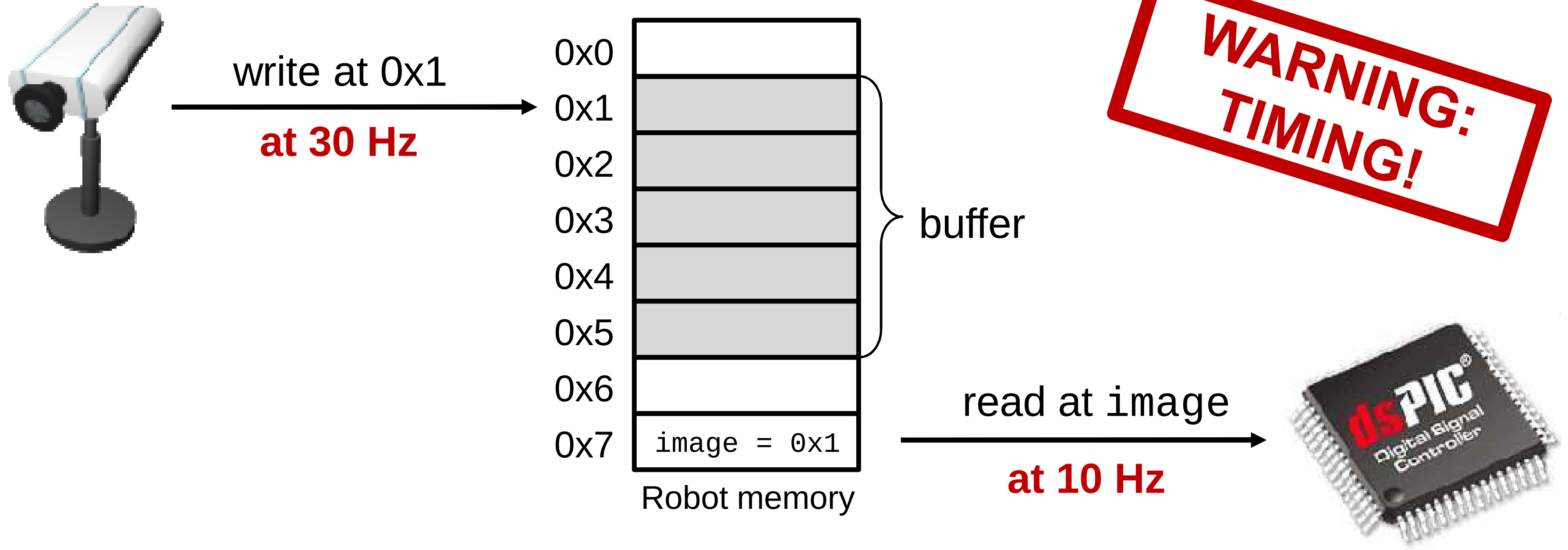
WARNING: TIMING!

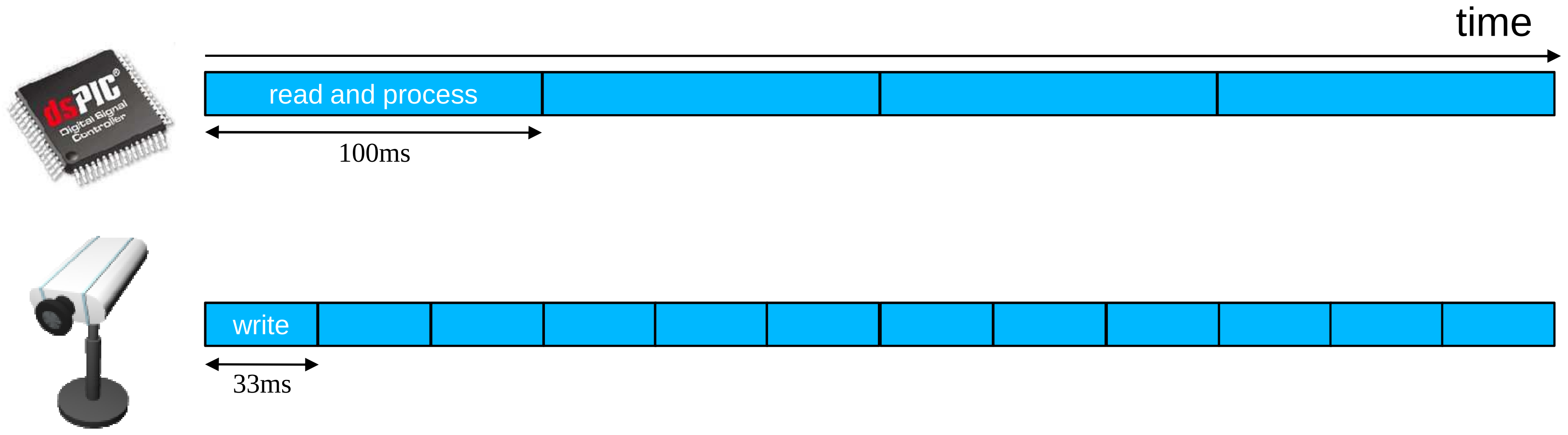
```
const unsigned char *image = wb_camera_get_image(camera);
for (int x = 0; x < image_width; x++) {
  for (int y = 0; y < image_height; y++) {
    int r = wb_camera_image_get_red(image, image_width, x, y);
    int g = wb_camera_image_get_green(image, image_width, x, y);
    int b = wb_camera_image_get_blue(image, image_width, x, y);
    printf("red=%d, green=%d, blue=%d", r, g, b);
  }
}
```

Data can be lost! Only one frame out of three is actually processed here!

Buffer mechanism

WARNING: TIMING!



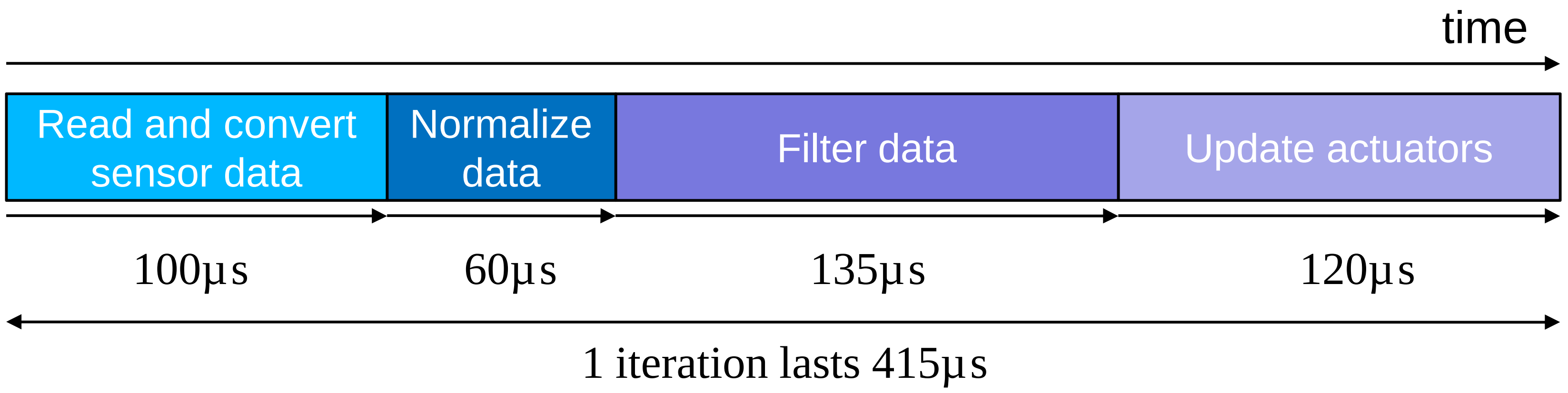


Asynchronous vs synchronous

- Each type of robot (DifferentialWheels, Robot or Supervisor) may be **synchronous** or **asynchronous**.
- Webots waits for the requests of **synchronous** robots before performing the next simulation step.
- It does not wait for **asynchronous** robots.
- Hence, an **asynchronous** robot may be **late** (if the controller is computationally expensive, or runs on a remote computer with a slow network connection).
- Obviously, in reality, all robots are **asynchronous** (with respect to real time).
- In practice, we use synchronous robots in simulation because Webots (like most simulation packages) **does not** simulate the microcontroller anyway.

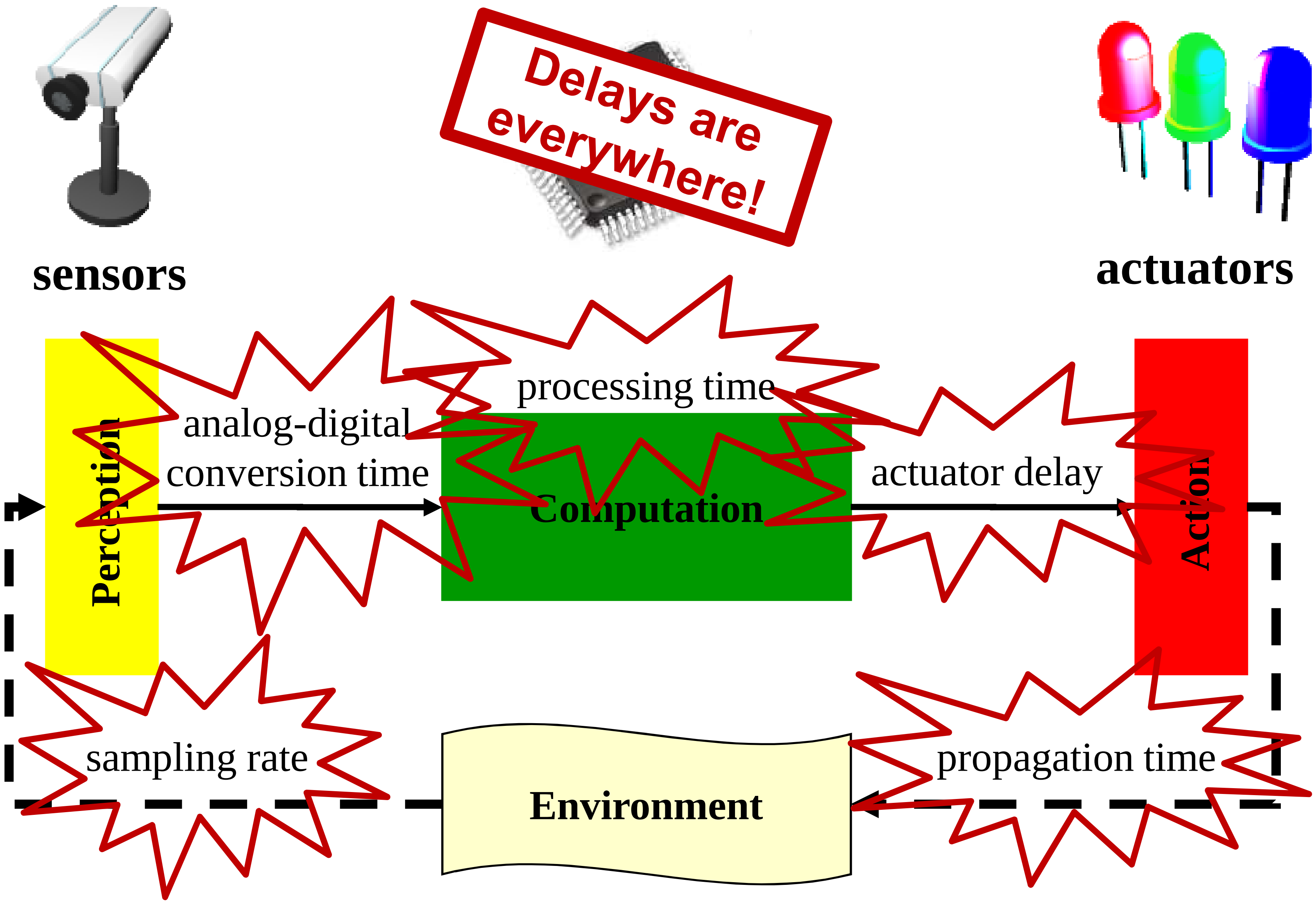
Real time?

- A robot (like any computing entity) has **limited computational ressources**. Therefore, any controller has a **computational cost**, which can be expressed as the time required to perform one iteration of the main loop.



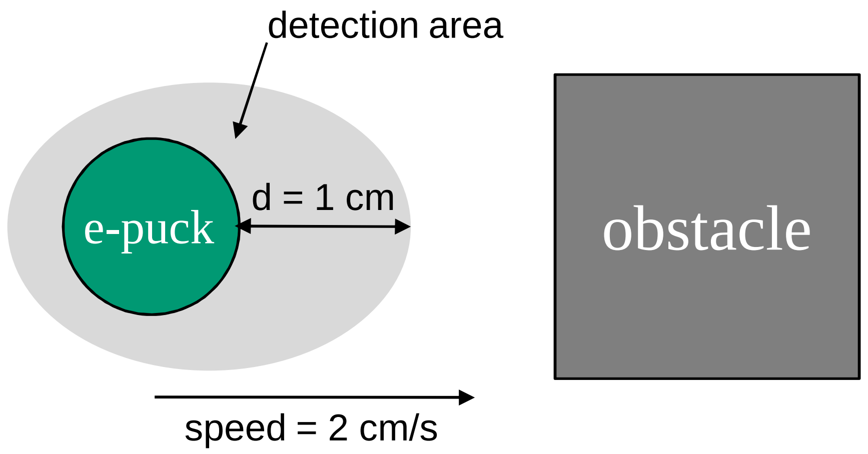
- For instance, the total duration of one single iteration of the controller depicted above is 415µs (~0.5 ms).
- Therefore, the maximal execution speed of the loop (which is also the update rate of the actuators) is 2 kHz.

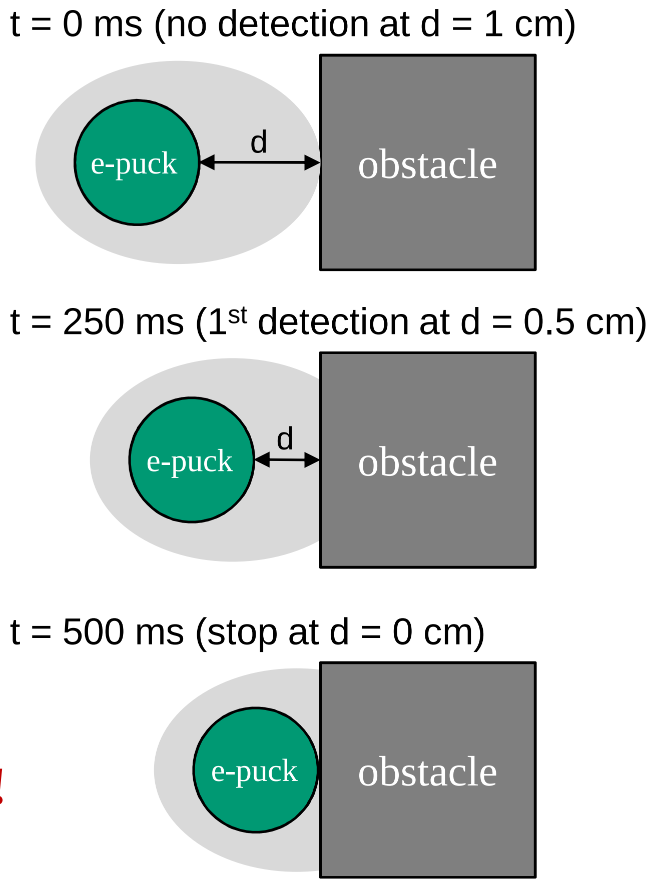
Perception-to-Action Loop



Real time!

- The perception-to-action delay defines the responsiveness of the robot.
- If the perception-to-action loop is too slow, the robot (and, in general, the embedded system) might miss important events!
- **Obstacle avoidance example:**
  - What is the maximal perception-to-action loop delay (in ms) required to prevent collisions?



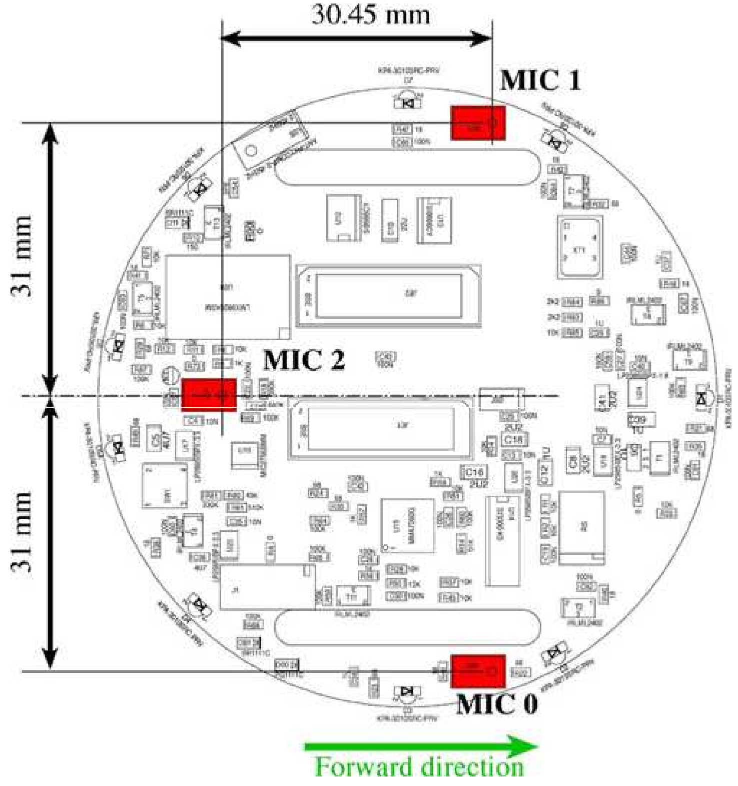


t = 0 ms (no detection at d = 1 cm)
e-puck
d
obstacle
t = 250 ms (1st detection at d = 0.5 cm)
e-puck
d
obstacle
t = 500 ms (stop at d = 0 cm)
e-puck
obstacle

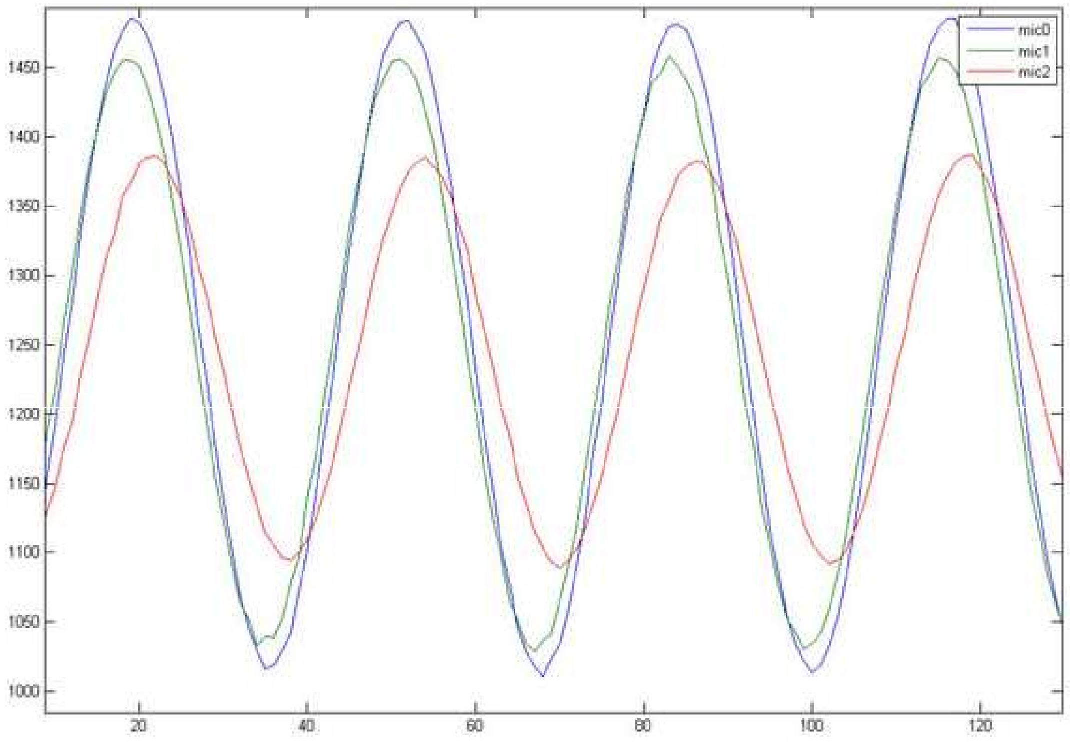
Theoretical answer: at most 250ms (at least 4 Hz)!
In practice, we need much faster responses!

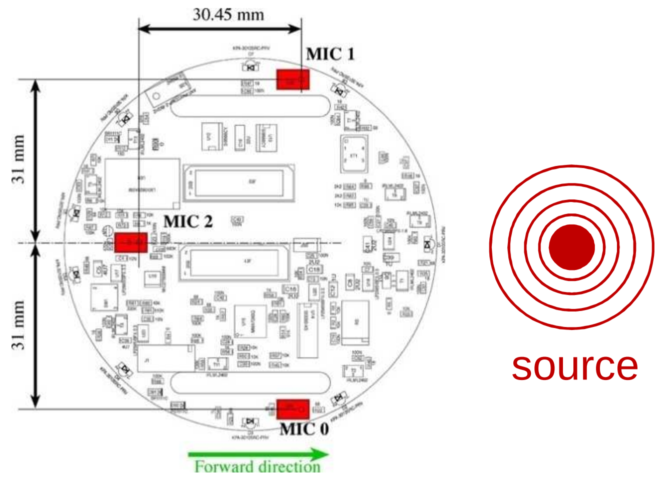
Microphone

- The e-puck robot is equipped with **3 microphones**.
- Acquiring the 3 microphones, the maximal acquisition speed is 33kHz (ADC maximal frequency of 100kHz divided by 3).
- You can use these 3 microphones to determine **the orientation** of the sound source!
- How can it be done? **By using signal processing!**



Source **in front** of the robot

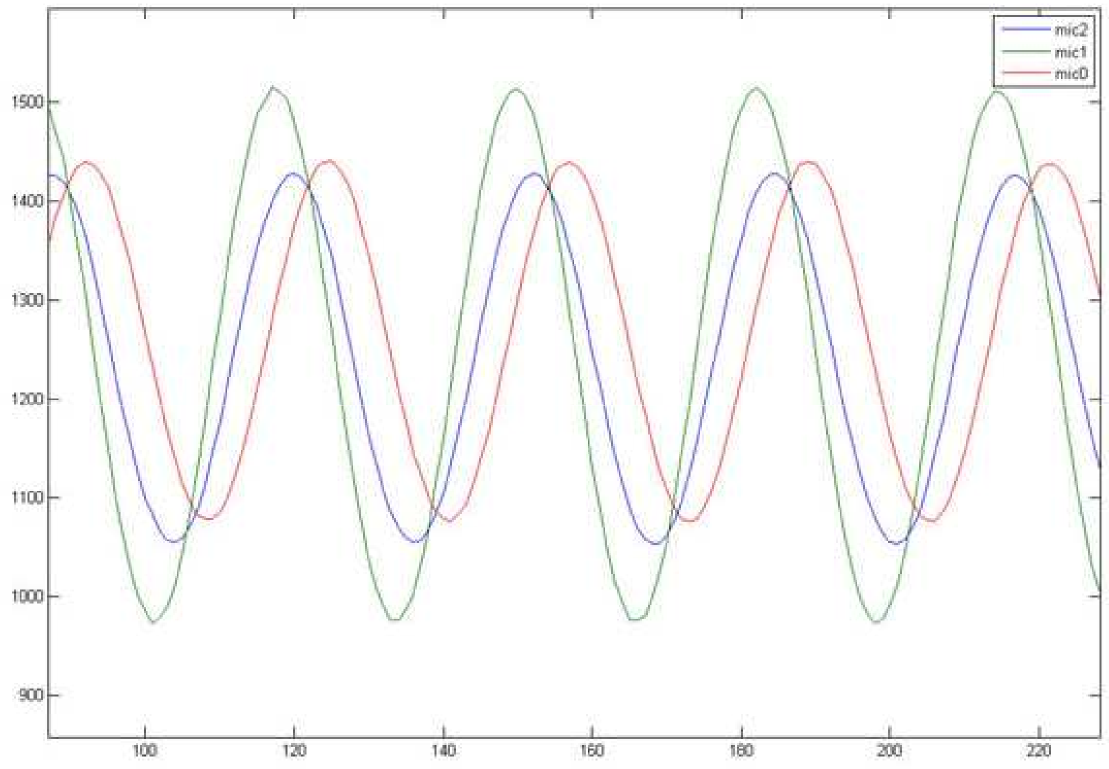


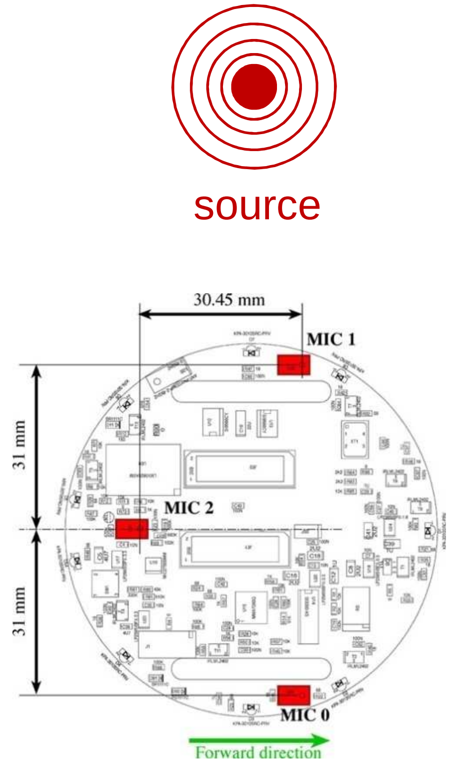


The signal from **mic2** is **time-shifted** (due to the delay of sound traveling) with respect to the other signals! The signals from **mic0** and **mic1** are in phase (same distance to the source)

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

Source **on the left** of the robot

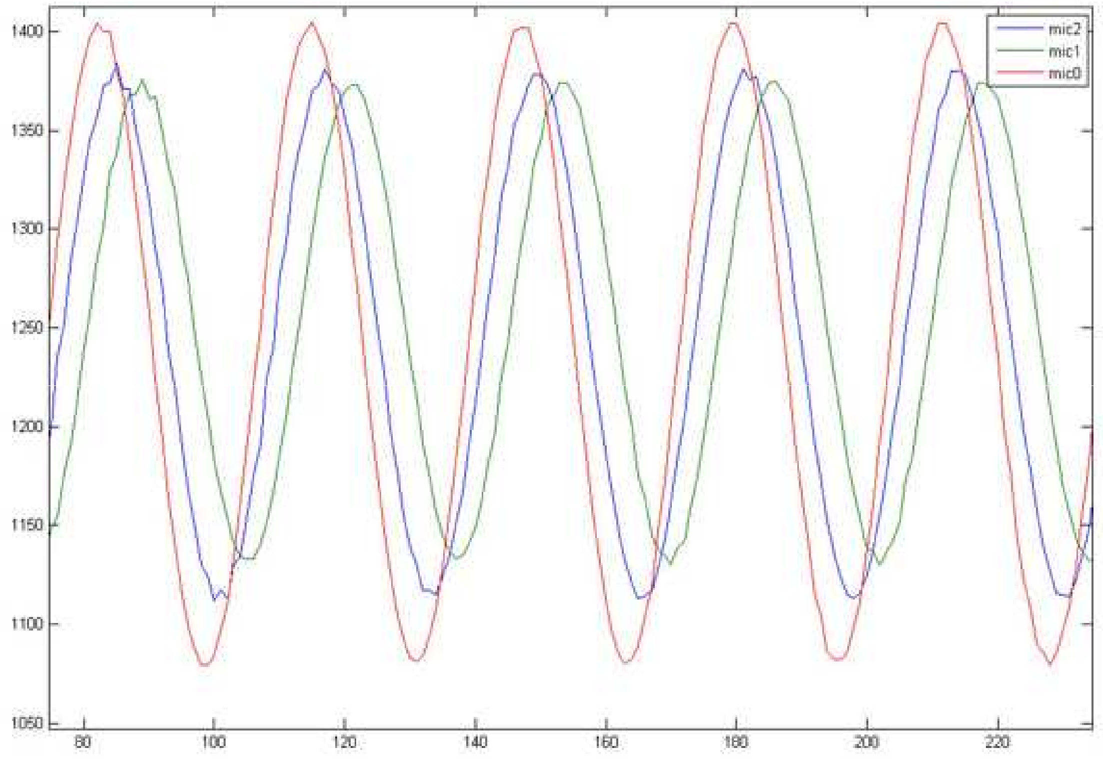


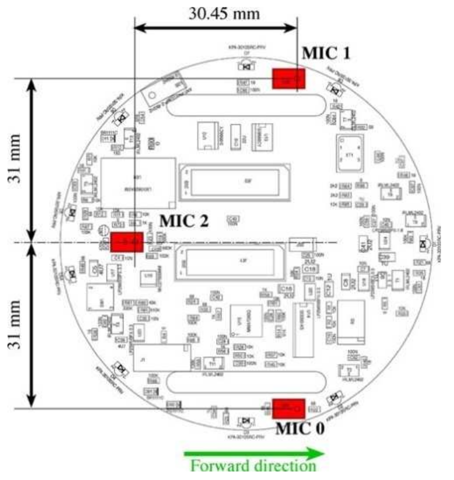


This time, the signal from **mic1** (left) is **in advance**! The signal from **mic0** (right) has the **largest time delay** with respect to mic1!

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

Source **on the right** of the robot





This time, the signal from **mic0** (right) is **in advance**! The signal from **mic1** (left) has the **largest time delay** with respect to mic1!

Microphone in Webots

- If you want to use the microphone and the speaker, you need to include the following header file:

```
#include <device/microphone.h>
```

- Then, you need to get the device by using the standard function `robot_get_device` and enable the microphone:

```
static DeviceTag microphone = robot_get_device("mic0");
microphone_enable(microphone, TIME_STEP / 2);
```

- Finally, in the main loop, you can read the data captured by the microphone using a buffer of signed short integers:

```
const signed short int *rec_buffer = (const signed short int*)
microphone_get_sample_data(microphone);
int size = microphone_get_sample_size(microphone)/sizeof(short);

for (int i = 0; i<size; i++) {
    printf("%d ",rec_buffer[i]);
}
```

Microphone on the real e-puck

- Like in Webots, a buffer is used to read the data from the microphone:

```
#define SAMPLELEN 1840         // for buffer allocation
static int values[SAMPLELEN]; // buffer and index for storing
static int valuesw;           // index
```

- However, the Webots API is no longer available on the real e-puck. Thus, you need to enable explicitly the microphone, using an interrupt.
- An interrupt is a special signal that triggers a change in execution. In this case, this interrupt triggers the execution of the function `_T1Interrupt` each 55µs (18 kHz). This function performs an A/D conversion and fills the buffer:

```
void _ISRFAST _T1Interrupt(void) {
  IFS0bits.T1IF = 0;           // clear interrupt flag
  timecounter++;               // tick the clock

  if (valuesw>(SAMPLELEN-1))   // stop writing when buffer is full
  {return;}                    //  (will be reset in main loop)

  values[valuesw++] = e_read_ad(MIC3);
}
```

Microphone on the real e-puck

- Finally, in the main loop, the code looks pretty similar to Webots:

```
valuesw=0;                     // reset index so that _T1Interrupt
while (valuesw<SAMPLELEN) {    // start to fill in the buffer
  __asm__ volatile("nop");     // and wait until it is full
}

for (int i = 0; i<size; i++) {
  printf("%d ",values[i]);
}
```

- Note that the following piece of code allows one to include assembler instructions in C code. In this case, the instruction is `nop`, which means "no operation". It is useful to make the microcontroller "wait" for a while:

```
__asm__ volatile("nop");
```

- Next week, further details about programming of real e-puck robots, interrupts, memory organization of dsPIC architectures, etc.

Using Webots at Home

- Available for Windows, Linux and Mac OS X
  - Support for the lab is only guaranteed for the DISAL virtual machine in GR B0 01
- Download Webots installation package from
  - http://www.cyberbotics.com
- Request a free trial license there:
  - www.cyberbotics.com/registration/webots/trial
- Computers connected to the EPFL intranet (or VPN) do not need a licence

Reading and acknowledgements

- Have a look at the Webots User Guide: http://www.cyberbotics.com/cdrom/common/doc/webots/guide/guide.html
- If you need help about a specific function and/or device, refer to the Webots Reference Manual: http://www.cyberbotics.com/cdrom/common/doc/webots/reference/reference.html
- In case of problems with the e-puck robot, refer to the official website www.e-puck.org.
- Thanks to Yvan Bourquin, CTO of Cyberbotics.com, for his slides about Webots!