Signals, Instruments, and Systems – W8

Introduction to Mobile Robotics – High-Fidelity Simulation and Simple Control Architectures

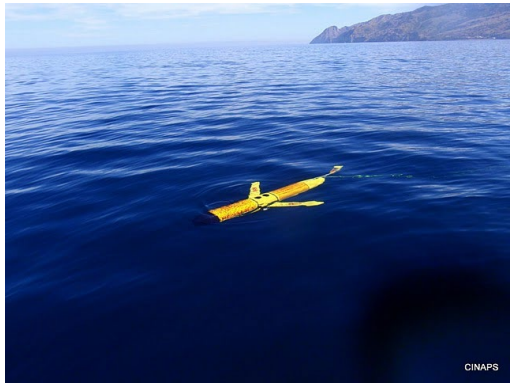
Outline

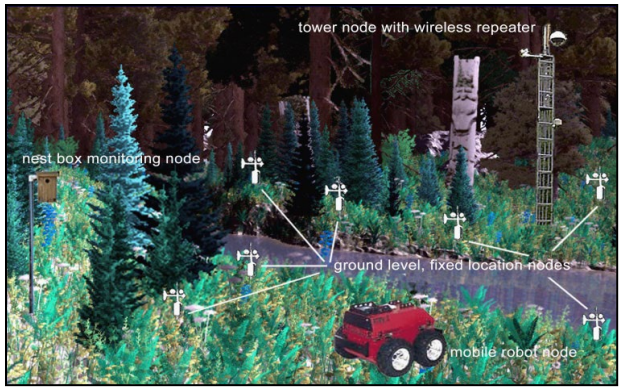
- Mobile robotics
  - Motivation for environmental applications
  - High-fidelity simulation: Webots
  - Simulation vs. reality
- Control architectures
  - Controller programming in Webots
  - The importance of real-time aspects
  - Taxonomy
  - Examples applied to obstacle avoidance

Mobile Robotics

Motivation for Environmental Applications

- Automation is progressing also in environmental engineering
- Because of the size of the area in which mission must be accomplished (e.g., sensing, monitoring, acting) mobility is key
- Being mobile add complexity and cost at the node level but extend coverage of potentially expensive assets
- Mobile robots become progressively essential tools for automating environmental engineering missions





The e-puck Mobile Robot

Main features

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

e-puck

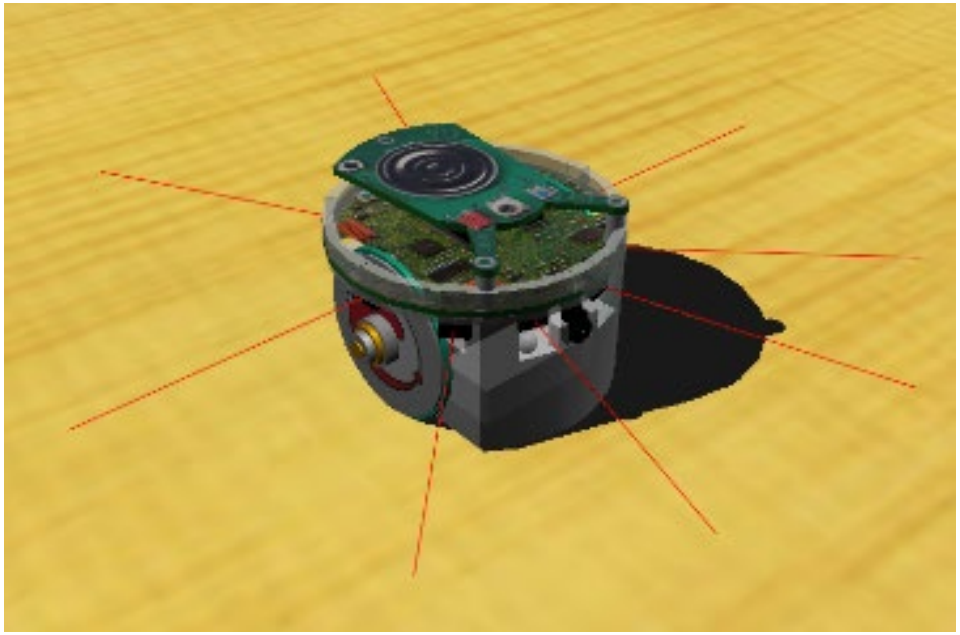
High-Fidelity Simulation

Simulation: Why?

- Hardware prototyping is time-consuming and expensive
- Real hardware platforms (e.g., robots) might be expensive
- Quickly change the experimental setup
- Often easier for monitoring experiments (and evaluating specific metrics)
- Can simulate experimental conditions difficult or impossible to reproduce in reality (e.g., noise-free environment) for investigating specific questions (e.g., impact of design choices on localization algorithms in absence of noise or with only specific ones)
- Sometimes faster than real-time
  - Useful for using numerical optimization schemes or machine-learning approaches
  - Enable systematic search of the parameter space

Real and Simulated e-puck

Real e-puck

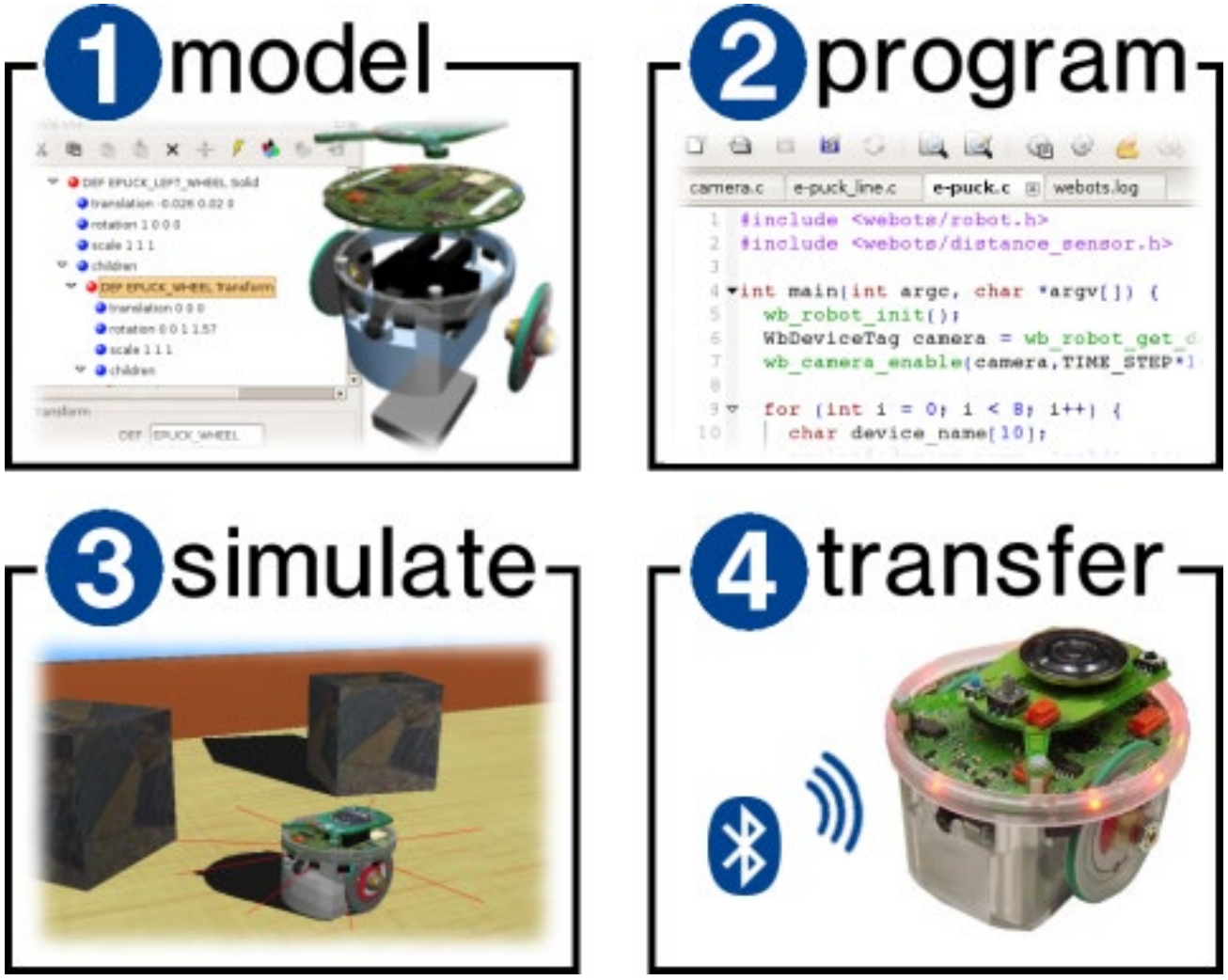


Simulated e-puck

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

The High-Fidelity Simulator Webots

Webots Robotic Simulator



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

Webots Features

- Open-source simulator supported commercially
- Realistic (physics-based) robotics simulator
- Flexible in the robot/environment configuration
- Allows for dedicated plug-in (e.g., radio network, gas plume, etc.)
- Speed-up indicator for various operational modes (can easily achieve up to two orders magnitude speed-up)
- Available sensors: distance sensors, light sensors, cameras, accelerometers, touch sensors, position sensors, GNSS, receivers, force sensors, etc.
- Available actuators (including com devices): linear and rotational motors, grippers, LEDs, emitters, etc.

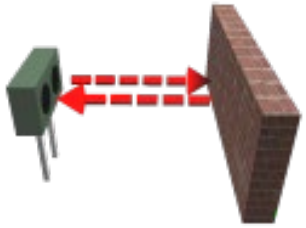
How Does it Look Like?

e-puck mobile robot performing obstacle avoidance and line following

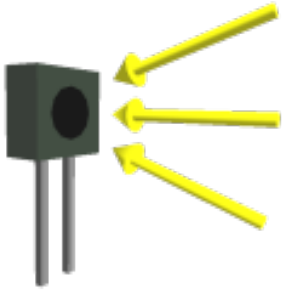
Webots GUI


scene tree
world view
editor
simulation speed-up
(w.r.t. real-time)
console

Modeled Sensors

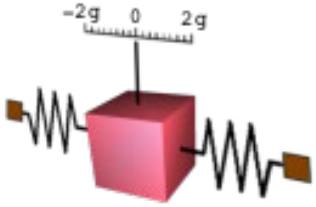


distance sensor

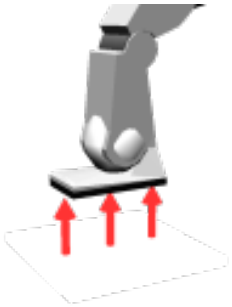


light sensor

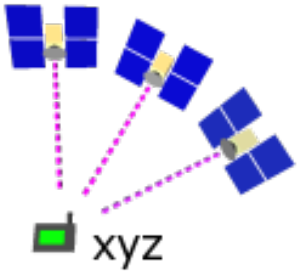
camera



accelerometer



touch/pressure sensor



GNSS

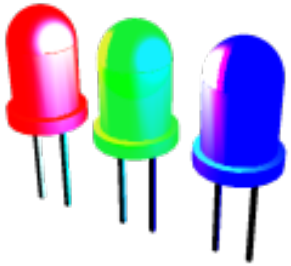
...and battery sensor, compass, gyroscope, torque sensor, radio receiver, etc.

Modeled Actuators

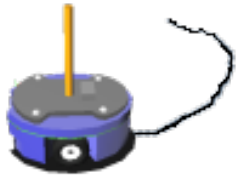
motor (rotational / linear)

gripper

connector
(docking systems)



LED



pen

... and propeller, radio transmitter, etc.

Simulation vs. Reality

Simulation vs. Reality - Differences

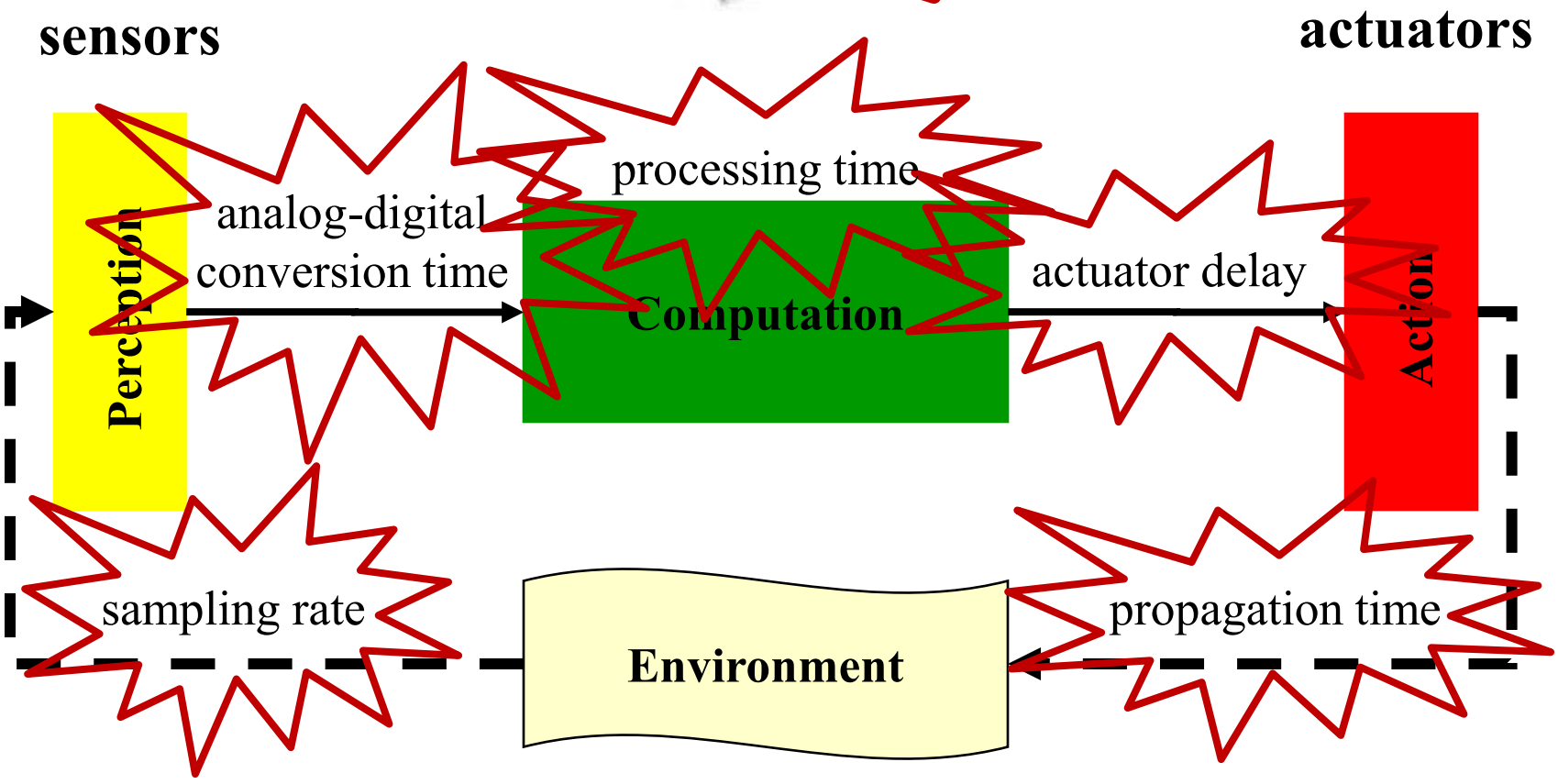
High-fidelity simulator Webots looks very powerful and realistic but (for instance):

- Manufacturing heterogeneities not reproduced (e.g., all sensor of a certain type are the same, all the robots as well).
- Noise distributions are typically uniform or Gaussian
- Sensor field of view simplified (e.g., ray instead of cone)
- Limitation in computational resources and internal electrical/computational architecture not reproduced
- World physics approximated (e.g., geometry, communication channel, fluid dynamics) or not reproduced (e.g., chemical dispersion, thermal dissipation, etc.)
- Real-time emulation very crudely approximated

Example

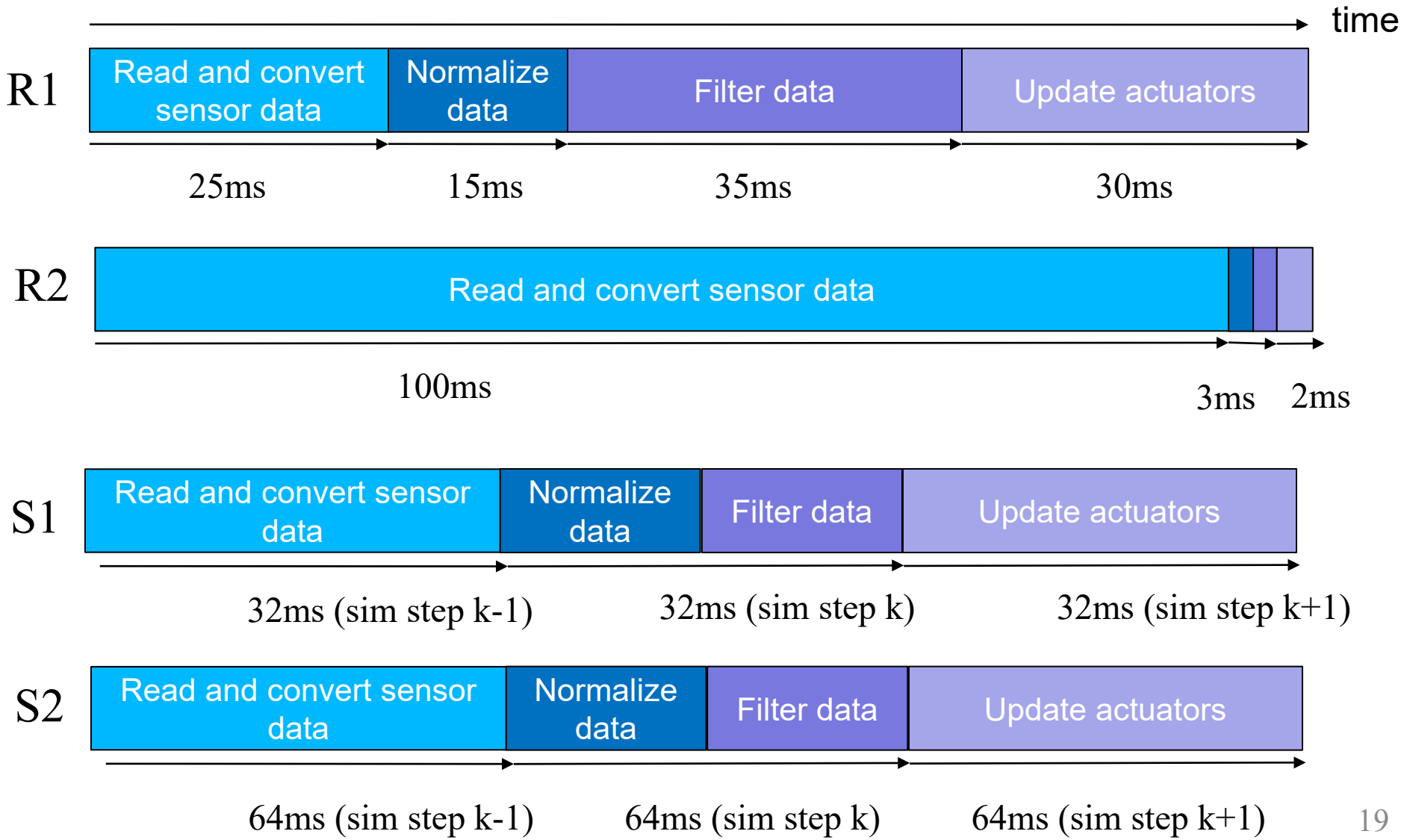
Example: Real-Time Emulation

See also W7, s. 51



Real-Time Emulation

R = possible reality situation; S: possible simulation parametrization



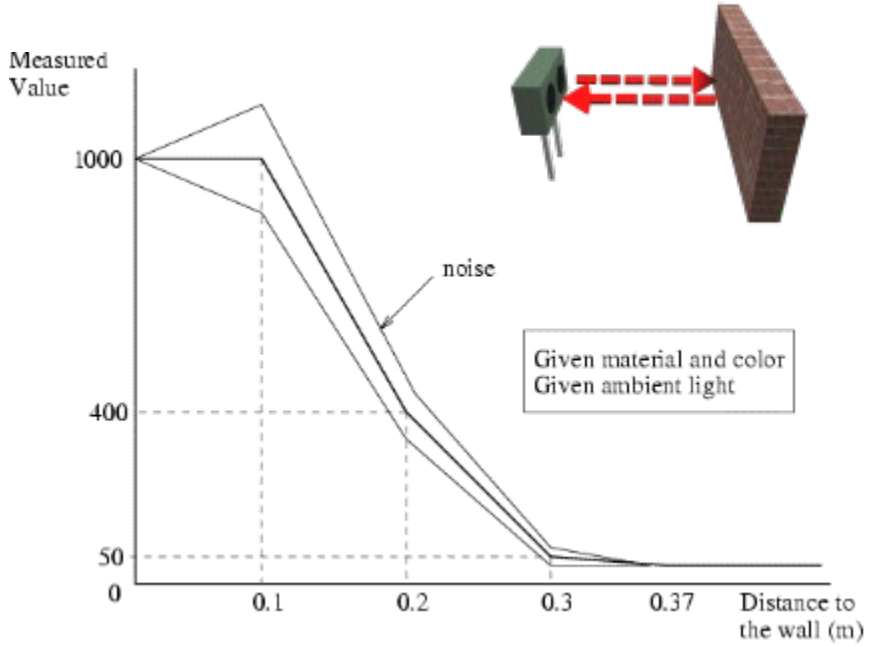
Asynchronous vs. Synchronous

- Each type of robot (DifferentialWheels, Robot or Supervisor) controller 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 robotic simulation packages) **does not** simulate microcontroller details anyway.
- Details: https://cyberbotics.com/doc/reference/robot#synchronous-versus-asynchronous-controllers

Simulated Sensors and Actuators

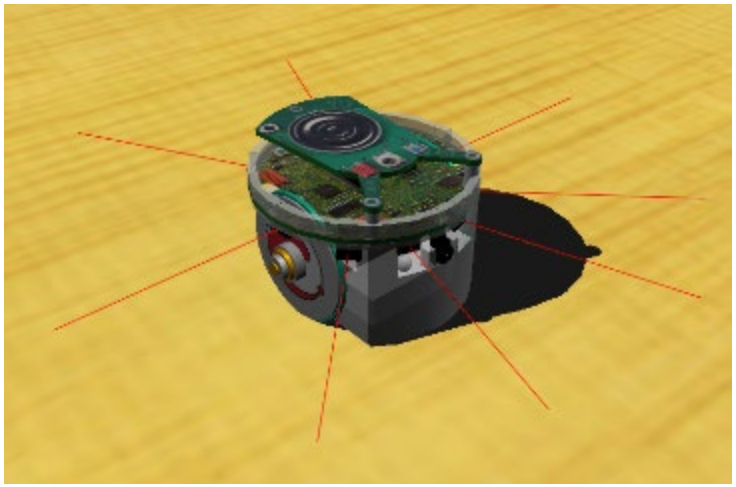
Modeling Sensors

- 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):



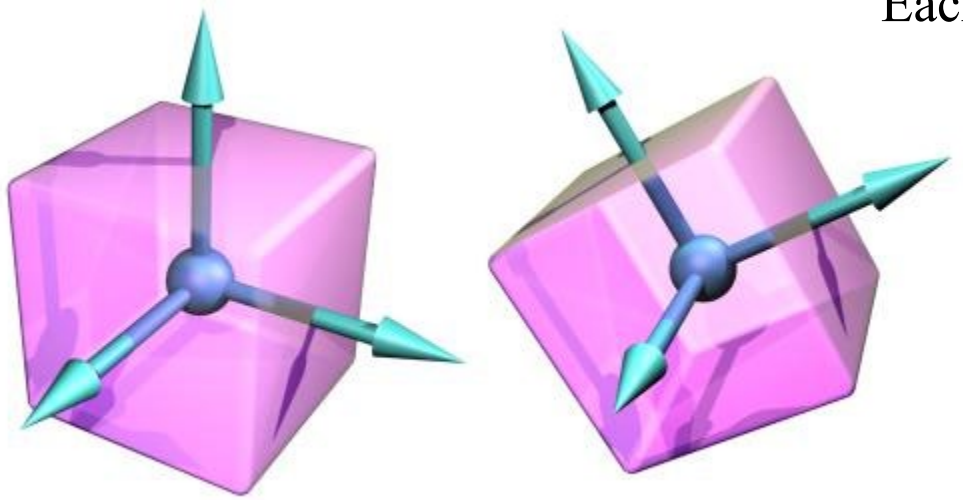

```
lookupTable [ 0       1000   0,
              0.1     1000   0.1,
              0.2      400   0.1,
              0.3       50   0.1,
              0.37      30   0     ]
```

distance value noise



(Newtonian) Physics-Based Simulation (ODE)

Rigid bodies simulation:



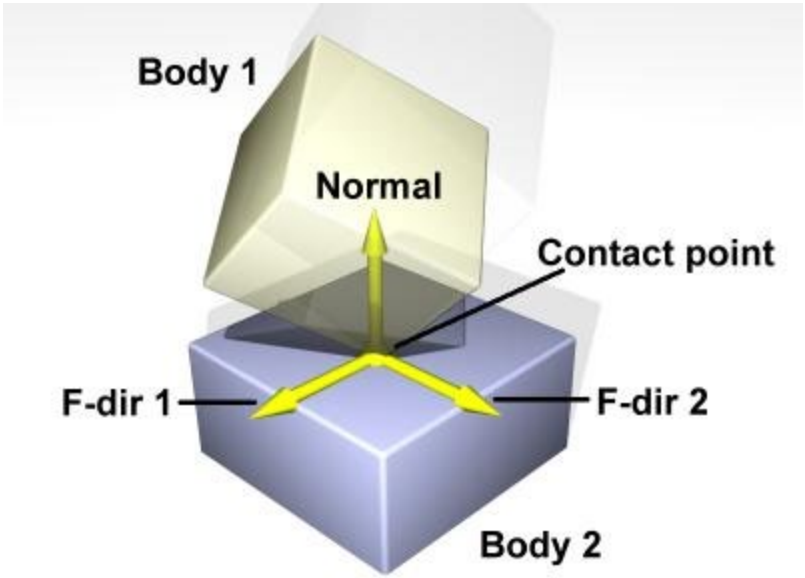
Each body is defined by its:

- Shape
- Mass (or density)
- Center of mass and Inertia Matrix
- Velocity (angular and linear)

Open Dynamic Engine: https://www.ode.org/

(Newtonian) Physics-Based Simulation (ODE)

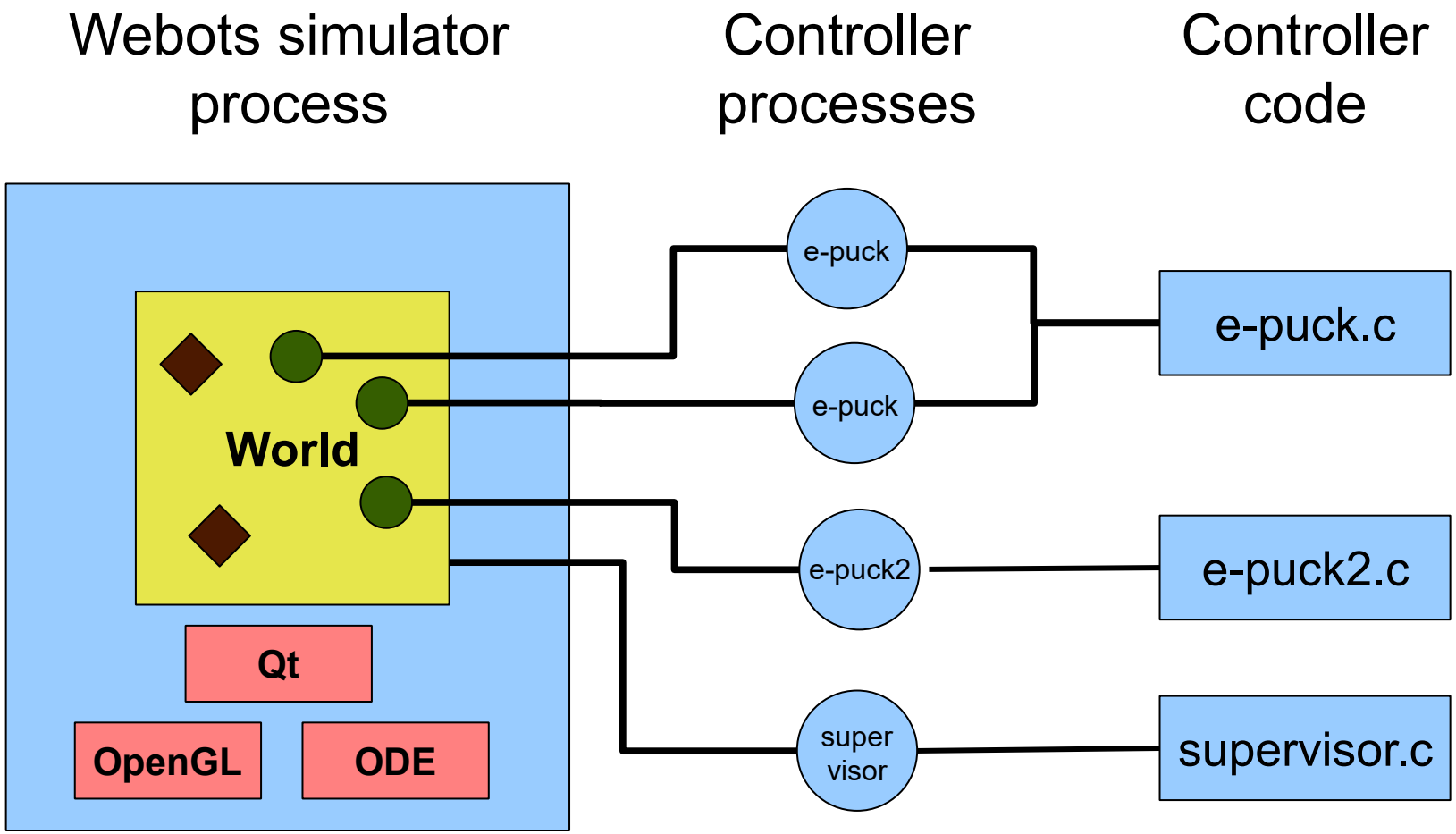
Collision detection:



Configurable friction:

Controllers in Webots

Webots Principles



Supervisor

- A **supervisor** is a program that controls the world and its robots.
- Similarly as a robot, supervisor is a node 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.

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** (although we stay in simulation in this course).
- Webots **does not** simulate the microcontroller of your robot:
  - Your controller typically will run **much slower** on the real robot than it does in Webots .
  - Your memory typically 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!

A Typical Webots Controller

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

Includes for accessing Webots API

```
#define TIME_STEP 32
```

Define simulation step in milliseconds

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

Define data structures

```
  // initialize webots
  wb_robot_init();
```

Initialize Webots

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

Get and enable different devices (sensors and actuators; ps0 = IR0 on s. 44)

```
  // main loop
  while (wb_robot_step(TIME_STEP) != -1) {
    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]);

  }
```

Perform one simulation step

Read sensor values

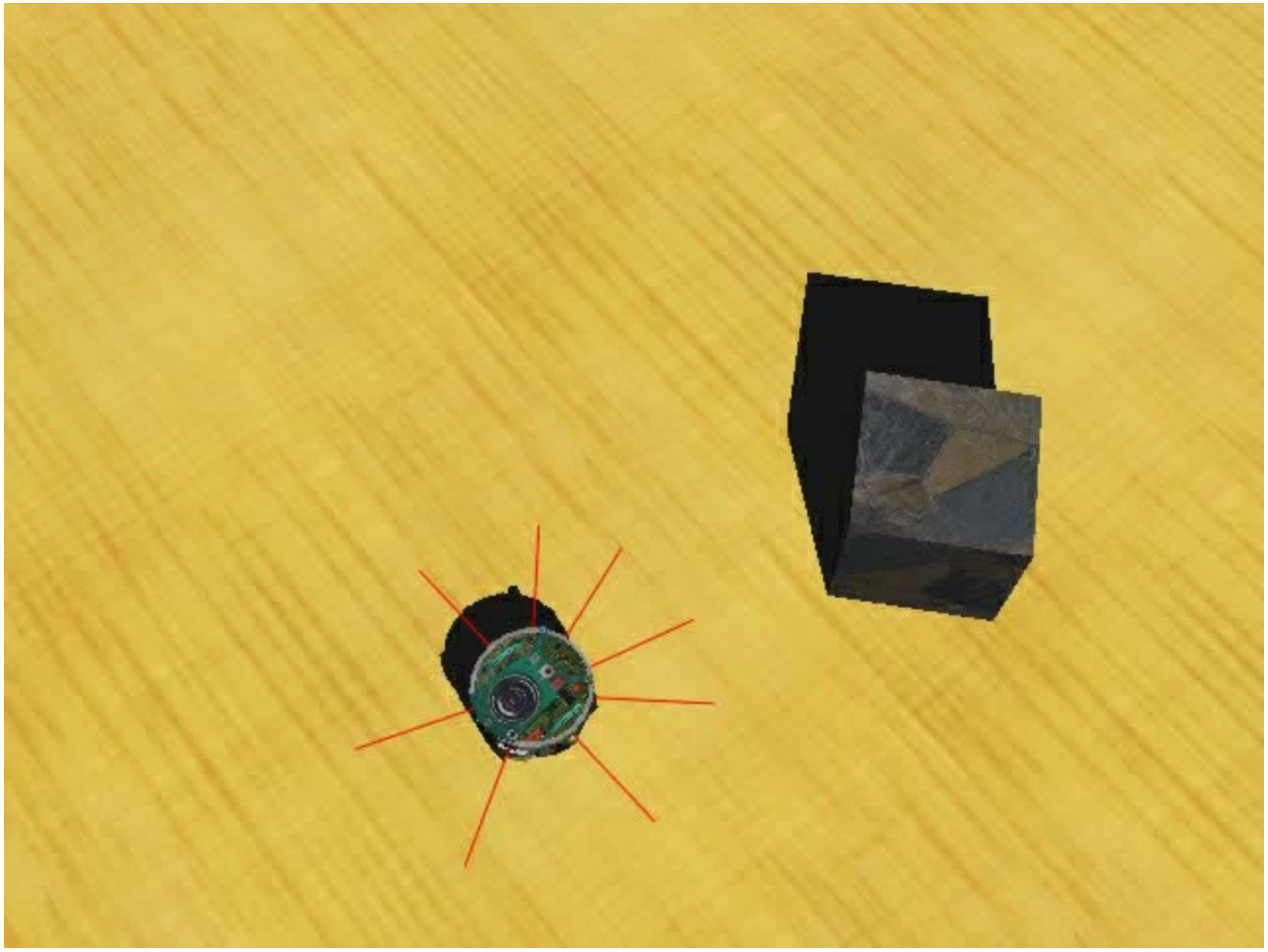
Controller lifetime loop (robot behavior must be coded here)

Update actuators

```
  // cleanup webots resources
  wb_robot_cleanup();
}
```

Cleanup Webots resources

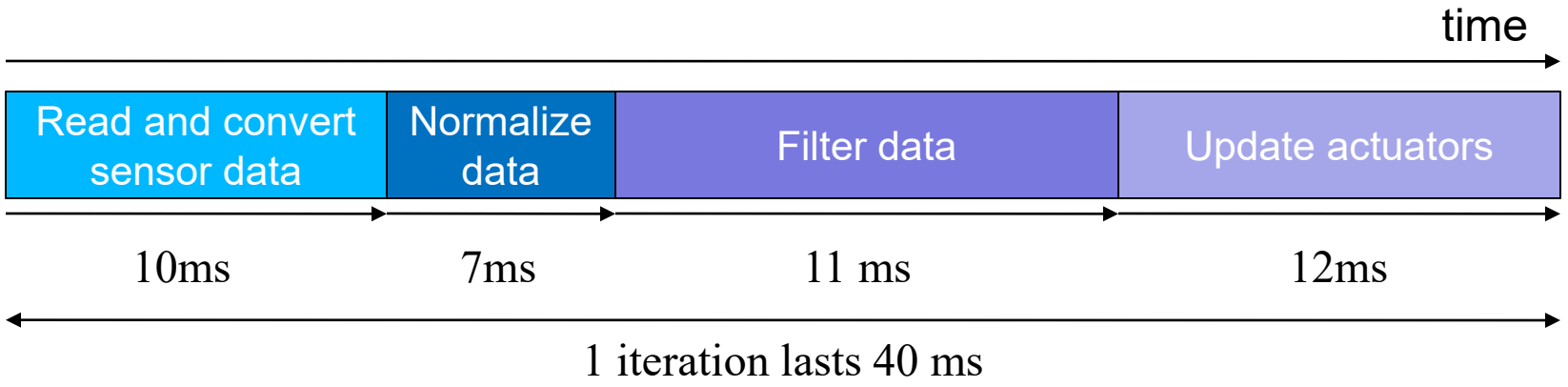
How Does it Look Like?



Real-Time Matters in Perception-to-Action Loops

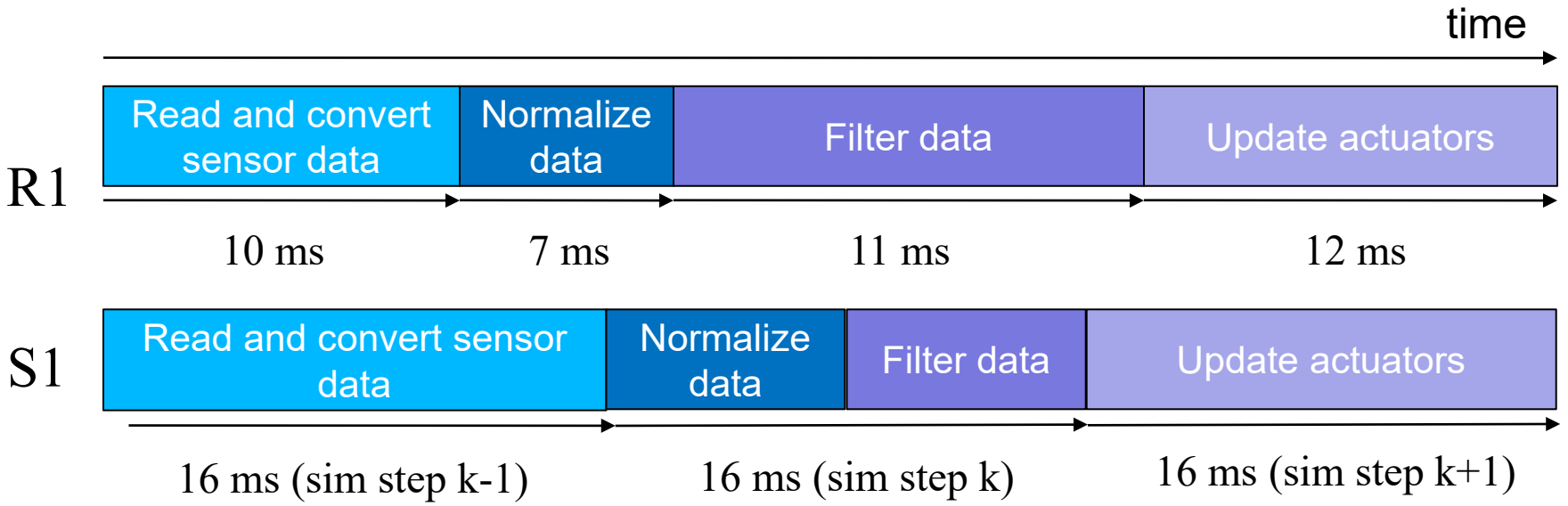
Real-Time Control in Reality

- A robot (like any computing entity) has **limited computational resources**. 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 40 ms.
- Therefore, the maximal execution speed of the loop (which is also the update rate of the actuators) is 25 Hz.

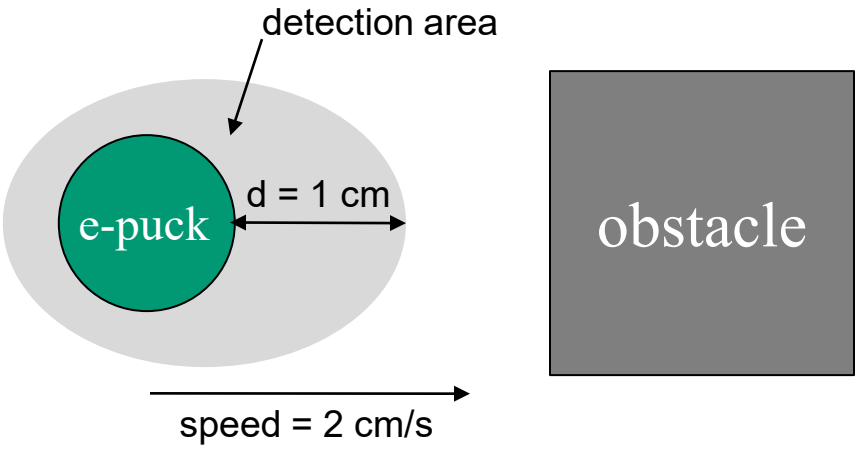
Real-Time Control in Simulation

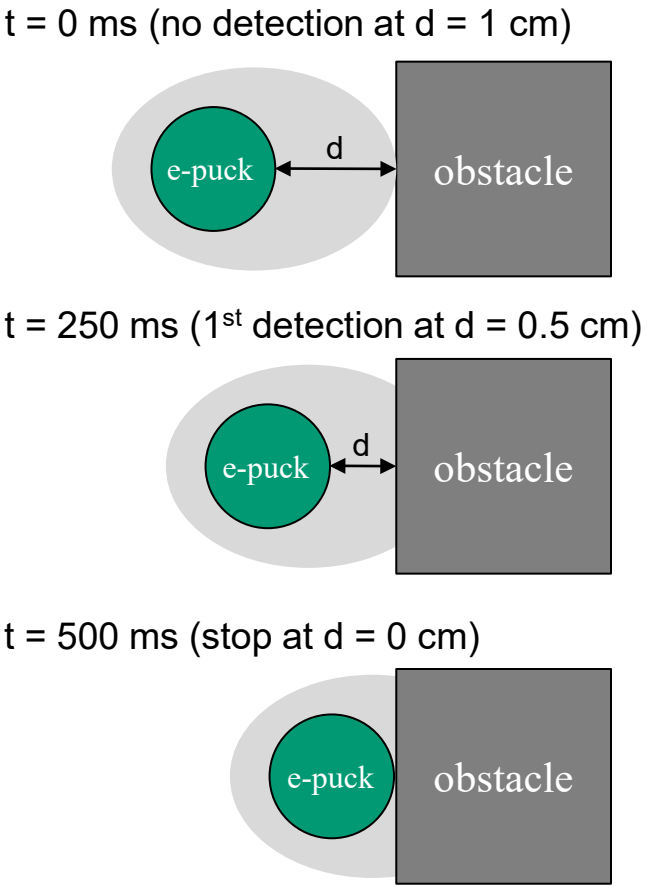


- Even if real-time is only emulated and approximated with the chosen simulation step in a high-fidelity simulator, closing the loop regularly within a certain duration is also relevant for simulation
- Hardware platforms used for simulation are typically way more powerful than embedded computers on robots: the emulation of the real-time cannot be too different if the code must be ported to reality

Ensuring Real-Time Control

- 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 (an hard real-time constraint)?





Theoretical answer: at most 250 ms (at least 4 Hz)!

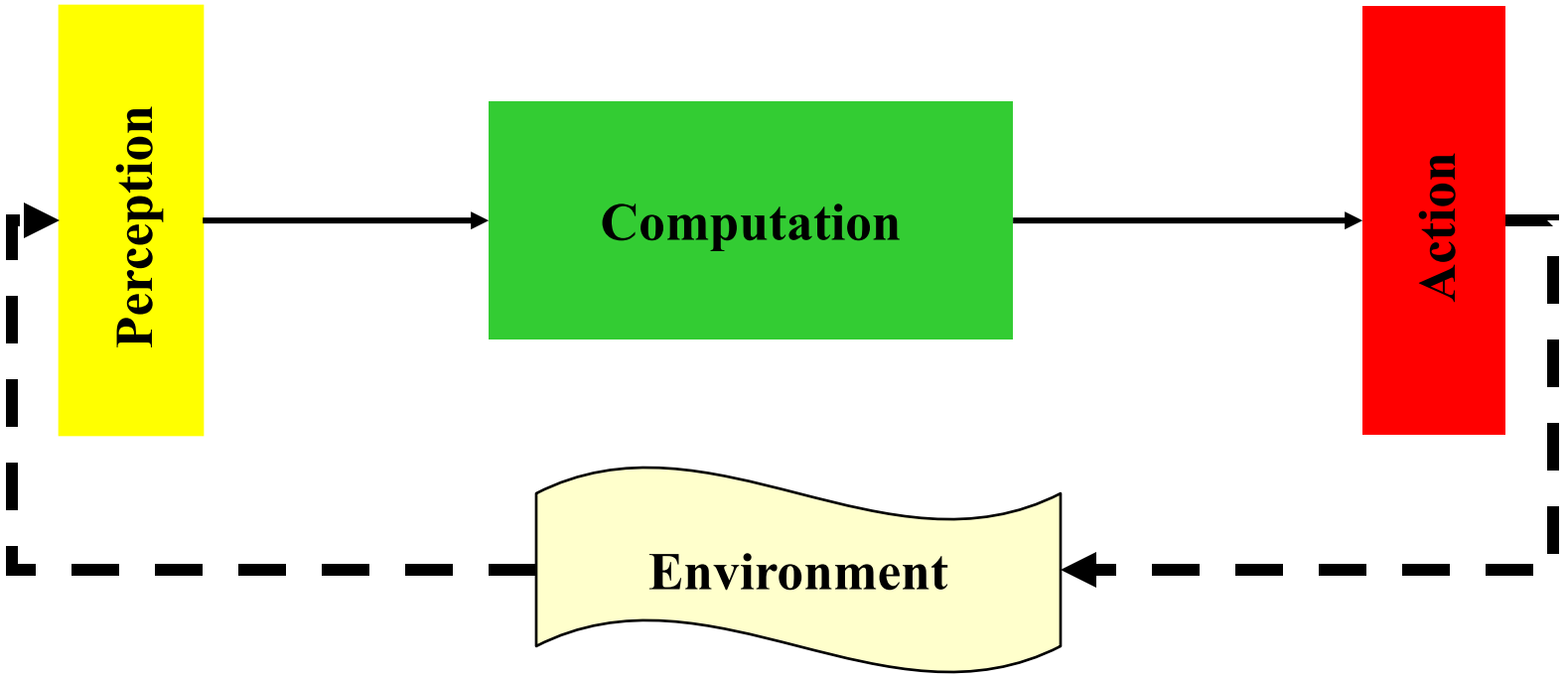
(potential implementation code: see s. 29)

In practice, we need much faster responses!

A Simple Taxonomy for Control Architecture in Mobile Robotics

Perception-to-Action Loop for a Mobile Robot

- sensors
- controller
- actuators



Reactive vs. Deliberative Architectures

- Reactive controller:
  - 1 perception-to-action loop horizon
  - No planning, no history stored

- Deliberative controller
  - Multiple perception-to-action loop horizon
  - Planning and history exploitation

- Reactive-deliberative boundary zone:
  - Short history, short look-ahead horizon
  - A few state variables and little memory

Sensory-Motor Control Architectures

- linear or nonlinear control laws
- no control hierarchy or layering
- high flexibility in shaping the behavior by changing parameters and keeping the structure fixed
- light architecture, fast execution time
- works well when few resources available
- can be described from a proximal perspective
- difficult to engineer in a "human-guided" way

Modular Architectures

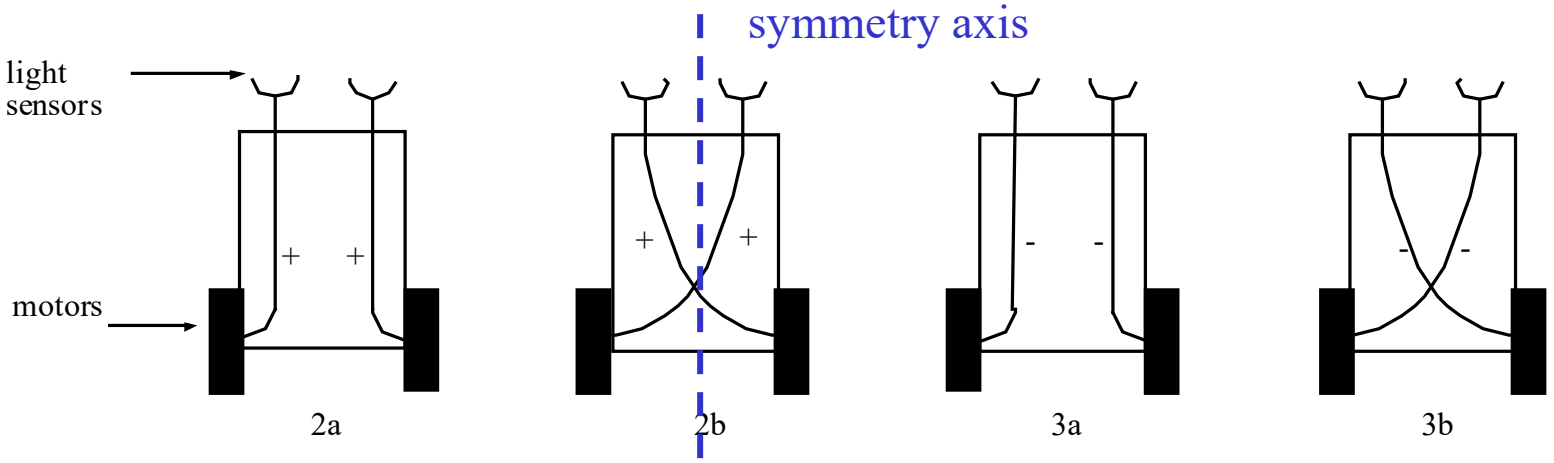
- Examples: behavior-based, layered, and hierarchical architectures
- less flexibility in shaping the behavior (behavioral module or basic behavior hardwired definition; flexibility only in the module "wiring" typically)
- not always computationally light architecture
- can be described from a distal perspective
- easier to engineer in a "human-guided" way because of the existence of modules (often hand coded)

Selected Reactive Architectures for Mobile Robots and their Application to Obstacle Avoidance

Overview

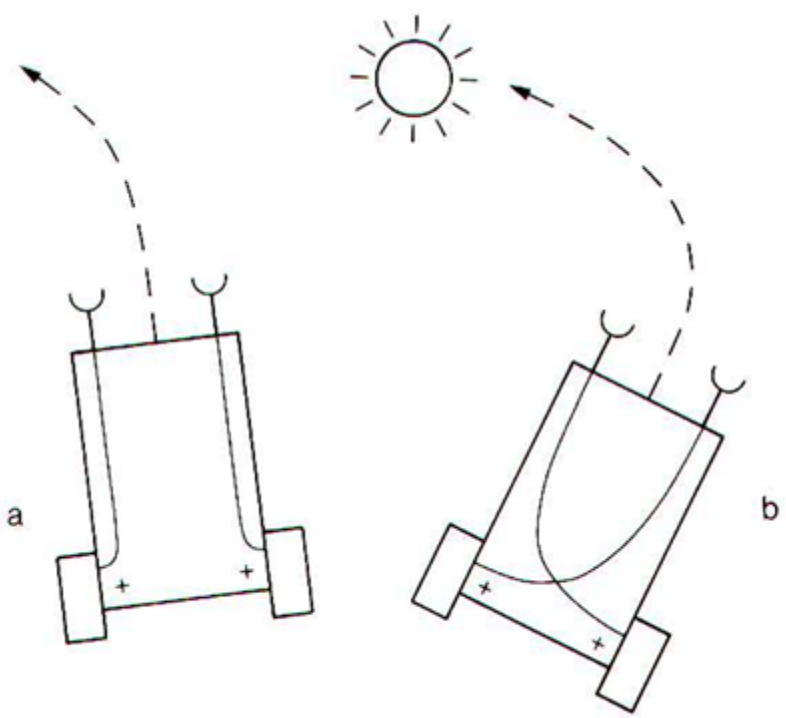
- Five “classical” examples of reactive control architecture for solving the same problem: obstacle avoidance.
- Two sensory-motor control architectures: Braitenberg and Artificial Neural Network
- Three modular architectures: Rule-based, Subsumption and Motor Schema, both behavior-based

Ex. 1: Braitenberg Vehicles

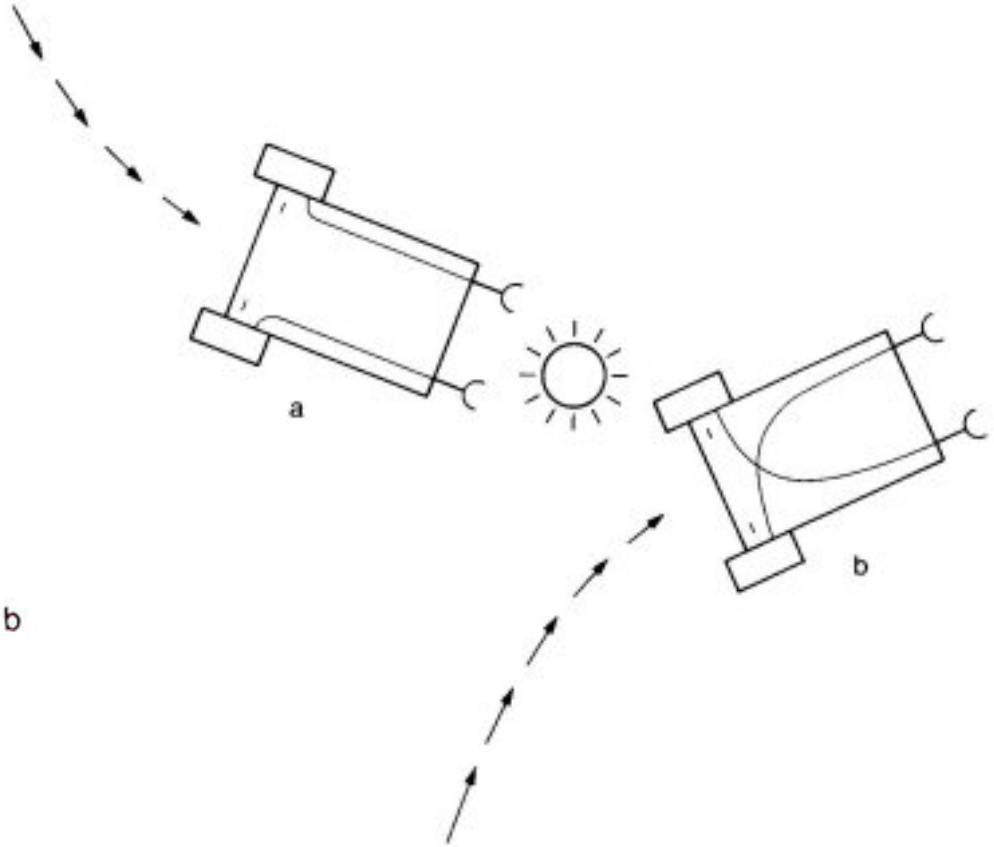


- Work on the difference (gradient) between sensors
- Originally omni-directional sensors; works also with directional sensors (sharper, potentially discontinuous differences at the sensory level -> more jerky movement)
- Originally: light sensors
- + excitation, - inibition; linear controller (out = signed coefficient * in)
- Symmetry axis along main axis of the vehicle (----)

Braitenberg Vehicles - Behaviors

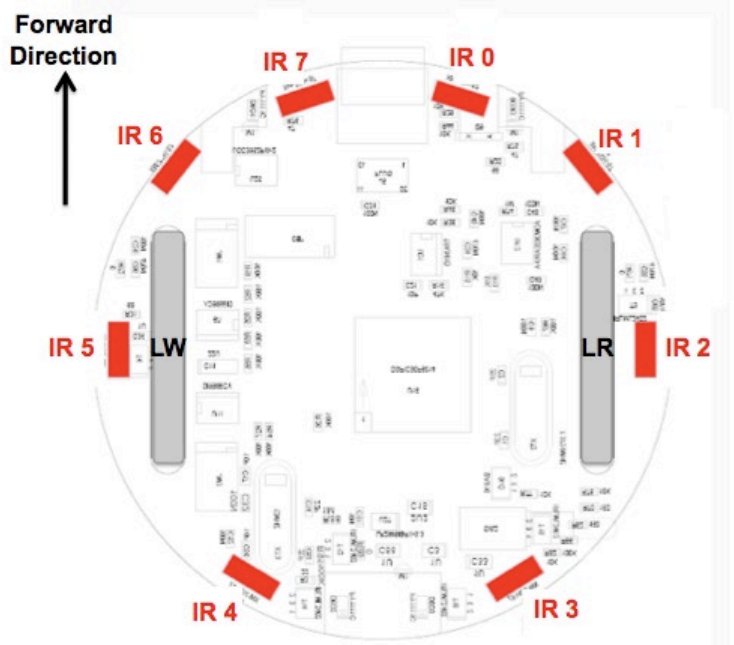


Excitatory connections



Inhibitory connections

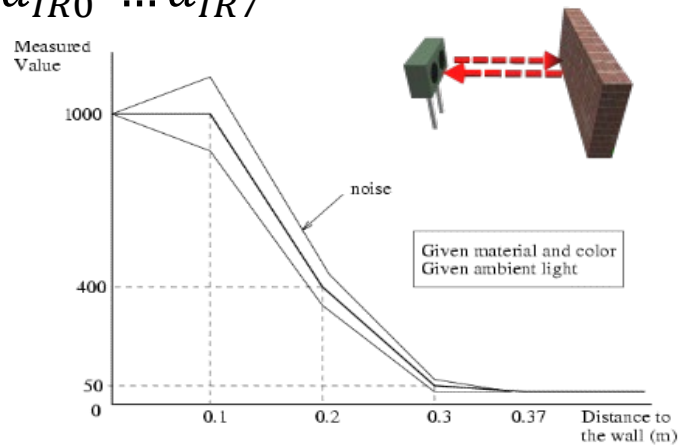
e-puck as a Braitenberg Vehicle



- 2 actuators
- 8 proximity sensors
- Motor speed is a linear combination:

$$\begin{bmatrix} v_L \\ v_R \end{bmatrix} = \begin{bmatrix} \alpha_{L0} & \alpha_{L1} & \cdots & \alpha_{L7} \\ \alpha_{R0} & \alpha_{R1} & \cdots & \alpha_{R7} \end{bmatrix} \cdot \begin{bmatrix} d_{IR0} \\ \vdots \\ d_{IR7} \end{bmatrix}$$

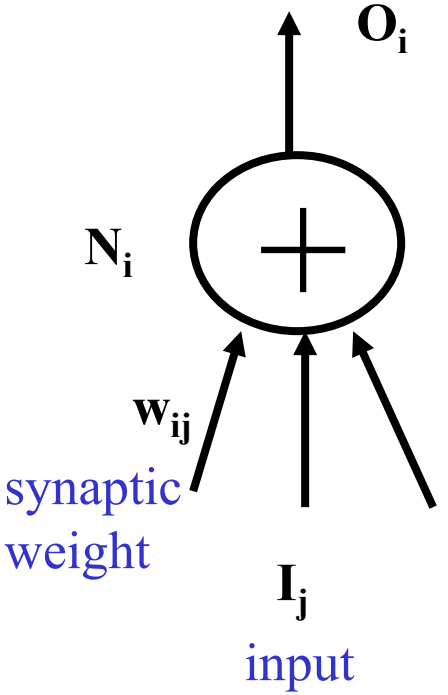
$d_{IR0} \ldots d_{IR7}$



- Bias otherwise no stimulus if no obstacle

$$\begin{bmatrix} v_L \\ v_R \end{bmatrix} = \begin{bmatrix} \alpha_{L0} & \alpha_{L1} & \cdots & \alpha_{L7} \\ \alpha_{R0} & \alpha_{R1} & \cdots & \alpha_{R7} \end{bmatrix} \cdot \begin{bmatrix} d_{IR0} \\ \vdots \\ d_{IR7} \end{bmatrix} + \begin{bmatrix} v_{L0} \\ v_{R0} \end{bmatrix}$$

Ex. 2: Artificial Neural Network

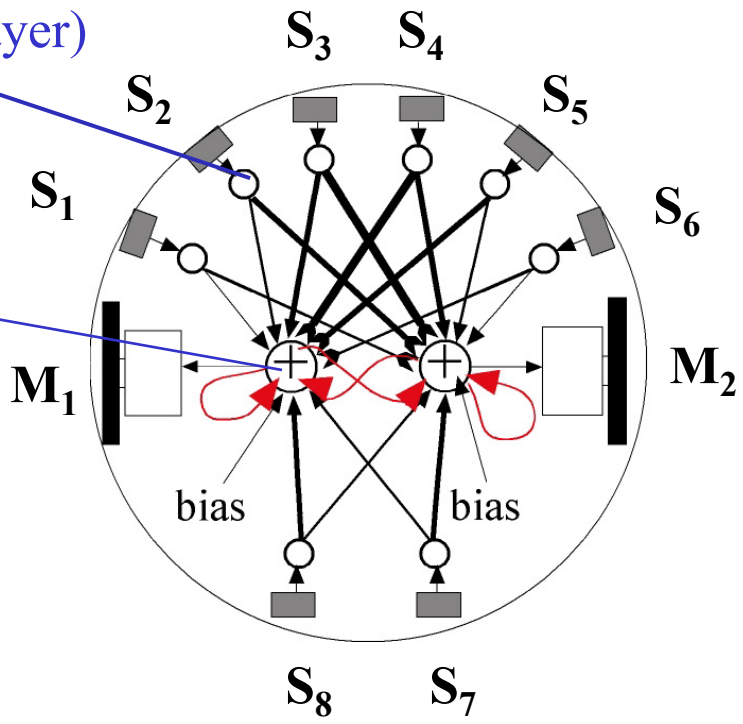


neuron (output layer) with sigmoid transfer function f(x)

$$O_i = Kf(x_i)$$

$$f(x) = \frac{2}{1+e^{-x}} - 1$$

$$x_i = \sum_{j=1}^{m} w_{ij} I_j + I_0$$



K: normalization constant

Ex. 3: Rule-Based

Rule 1:
if (proximity sensors on the left active) **then**
turn right

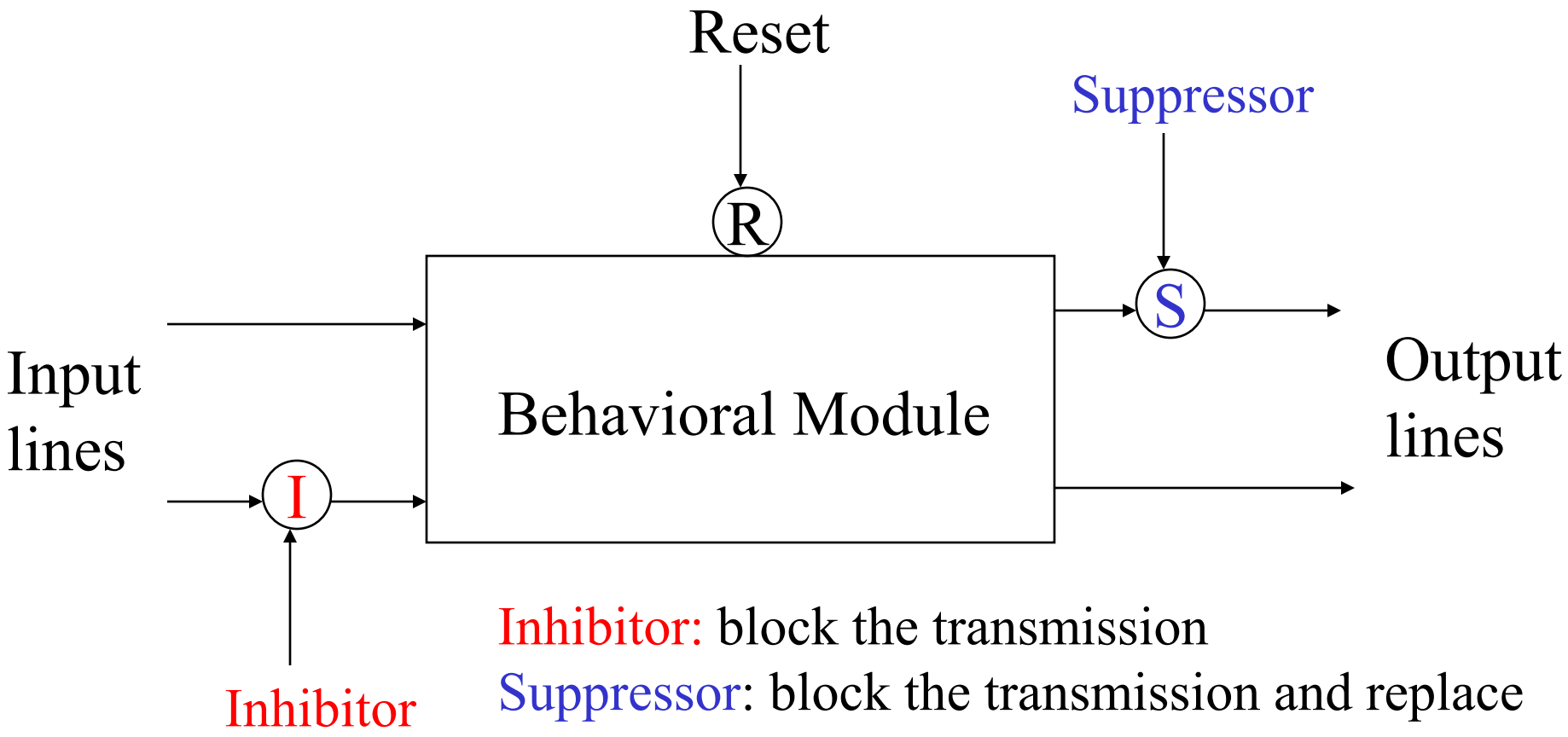
Rule 2:
if (proximity sensors on the right active) **then**
turn left

Rule 3:
if (no proximity sensors active) **then**
move forwards

Subsumption Architecture

- Rodney Brooks (1986)
- Precursors: Braitenberg (1984), Walter (1953)
- Behavioral modules (basic behaviors) represented by Augmented Finite State Machines
- Response encoding: predominantly discrete (rule based)
- Behavioral coordination method: competitive (priority-based arbitration via inhibition and suppression)

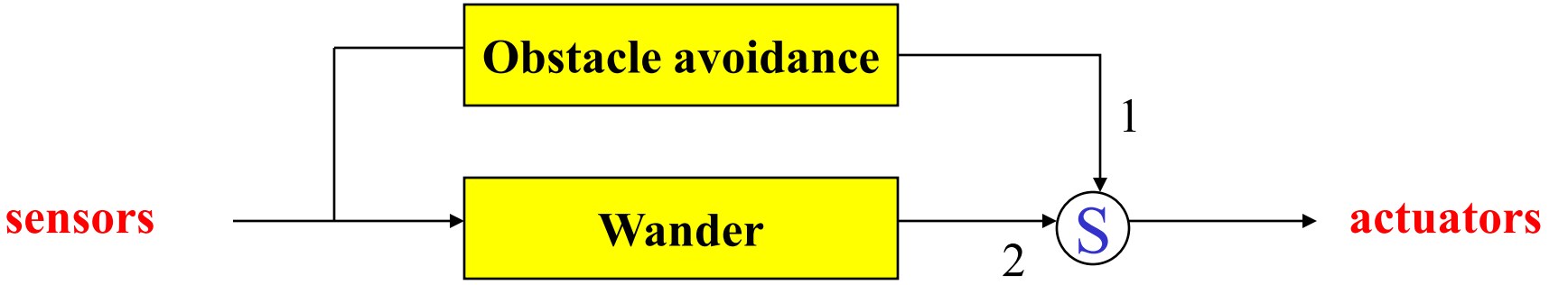
Augmented Finite State Machine



Inhibitor: block the transmission

Suppressor: block the transmission and replace the signal with the suppressing message

Ex. 4: Behavior-Based with Subsumption



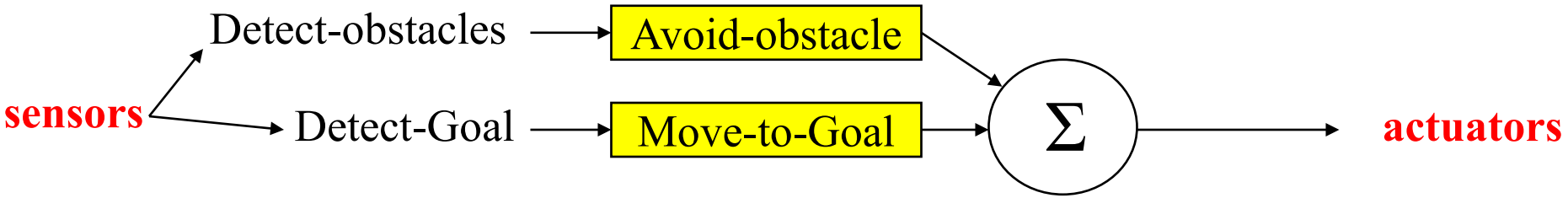
(1 suppresses and replaces 2)

Concrete implementation within basic behaviors:
Obstacle avoidance: Braitenberg without bias, rule-based, etc.
Wander: bias on motors (straightforward motion), random walk, etc.

Motor Schemas

- Ronald Arkin 1987, Georgia Tech
- Precursors: Arbib (1981), Khatib (1985)
- Parametrized behavioral libraries (schemas)
- Response encoding: continuous using potential field analog
- Behavioral coordination method: cooperative via vector summation and normalization

Ex. 5: Behavior-Based with Motor Schemas



Note: each motor schema generate an output vector that then gets summed up and normalized for controlling the actuators

Visualization of Vector Field for Ex. 5

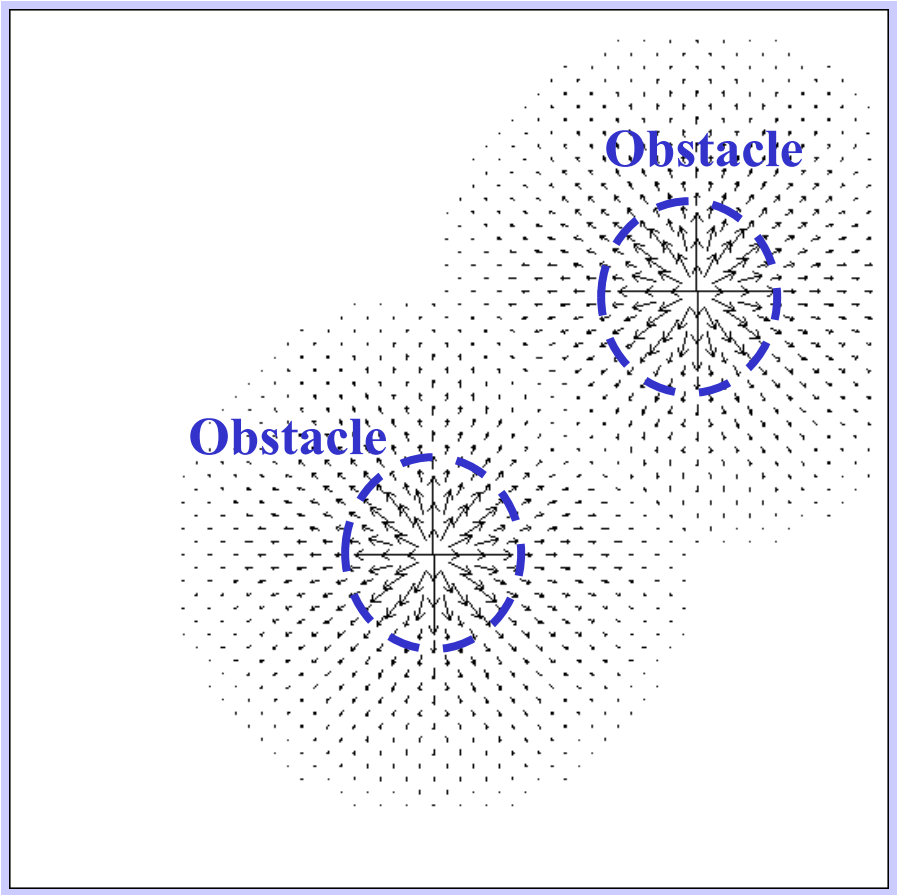
Avoid-obstacle

Vector = [magnitude, direction]

$$V_{magnitude} = \begin{cases} 0 & for \quad d > S \\ \frac{S-d}{S-R} G & for \quad R < d \le S \\ \infty & for \quad d \le R \end{cases}$$

S = obstacle's sphere of influence
R = radius of the obstacle
G = gain
D = distance robot to obstacle's center

$V_{direction}$ = radially along a line between robot and obst. center, directed away from the obstacle



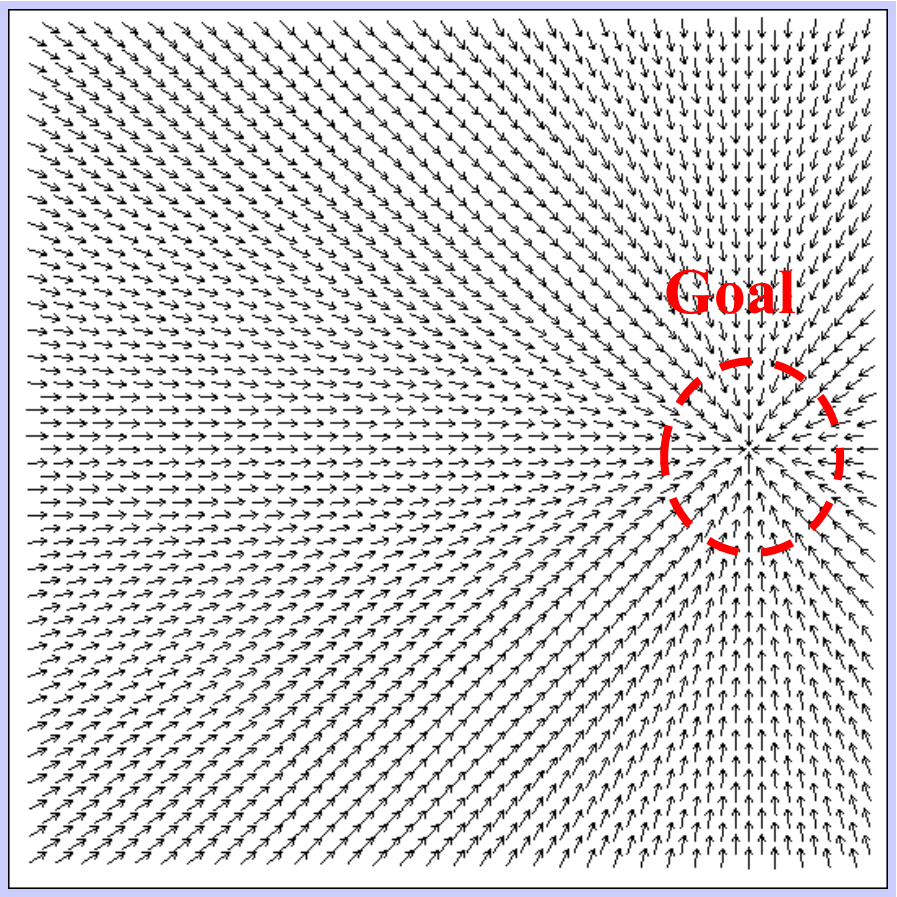
Visualization of Vector Field for Ex. 5

Move-to-goal

Vector = [magnitude, direction]

$V_{magnitude}$ = fixed gain value

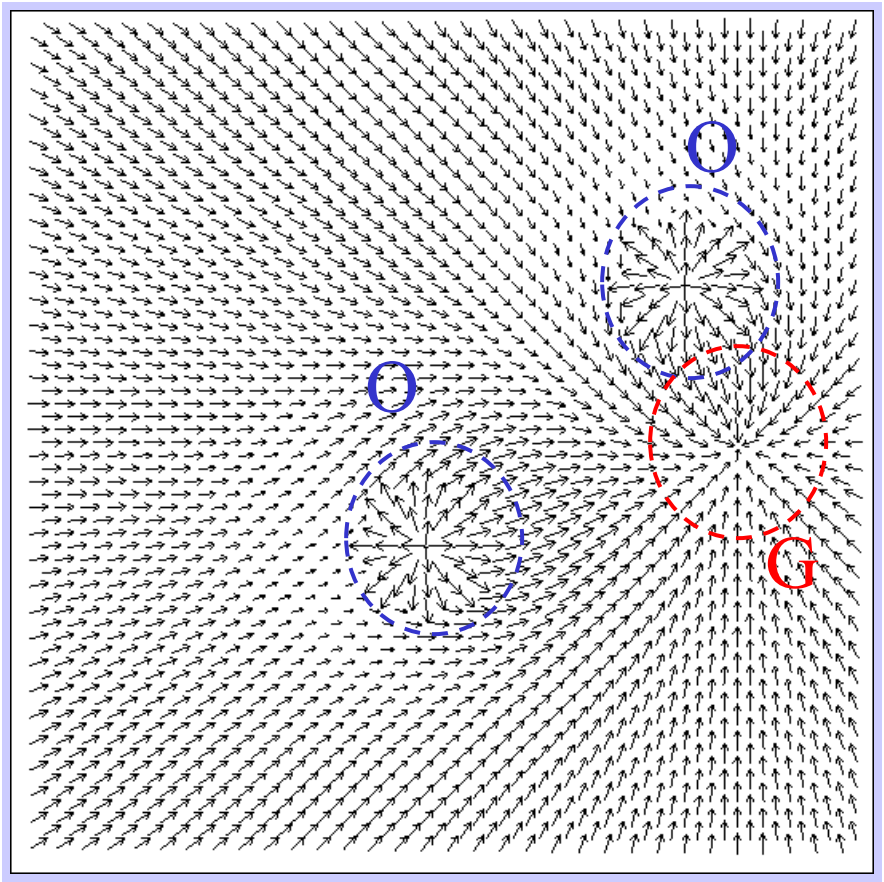
$V_{direction}$ = towards perceived goal



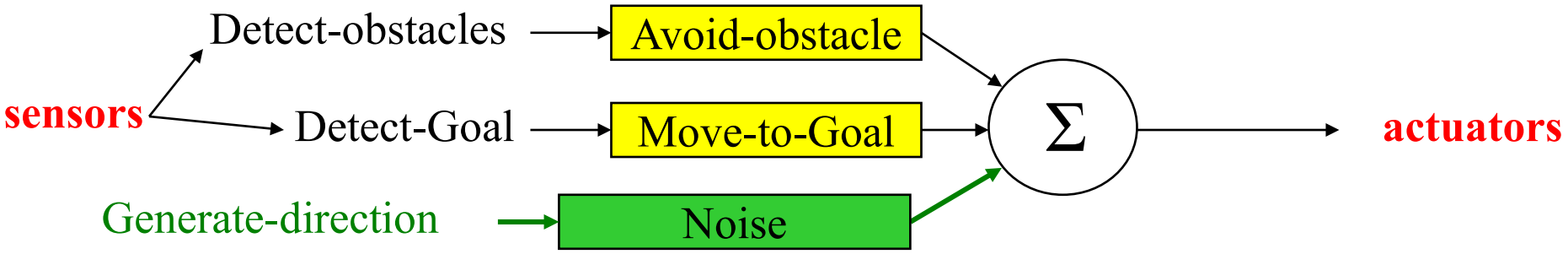
Visualization of Vector field for Ex. 5

Avoid-obstacle + **move-to-goal**

Linear superposition
(vectorial weighted sum)



Ex. 5: Issue with Motor Schemas



To avoid getting stuck in local minima of the vector field
(typical problem of vector field approaches)

Alternative more complex approach: use harmonic potential functions (explicitly designed for not generating local minima)

Conclusion

Take Home Messages

- High-fidelity simulation allows one to achieve rapid prototyping and experimentation **without using actual hardware**
- However, simulation is **an abstraction** of reality and does not account for all the details of the target system
- For instance, reproduction of **real time** is often crudely approximated even in the most sophisticated high-fidelity simulators
- A given **behavior** can be obtained with **different control architectures**
- Control architectures can be classified in **sensory-motor control** vs. **modular** and **reactive** vs. **deliberative**
- **Braitenberg** vehicles and **artificial neural networks** are two typical examples of reactive sensory-motor control architectures, **motor schemas** and **subsumption** are typical examples of reactive modular architectures

Additional Literature – Week 8

Papers

- Brooks R., "A robust layered control system for a mobile robot", *IEEE J. of Robotics and Automation*, 2(1): 14 – 23.
- Arkin R. C., "Motor Schema Based Mobile Robot Navigation". Int. *J. of Robotics Research*, 8(4): 92-112, 1989.

Books

- Braitenberg V., "Vehicles: Experiments in Synthetic Psychology", MIT Press, 1986.
- Siegwart R., Nourbakhsh I. R., and Scaramuzza D., "Introduction to Autonomous Mobile Robots", 2nd edition, MIT Press, 2011.
- Arkin R. C., "Behavior-Based Robotics". MIT Press, 1998.
- Everett, H. R., "Sensors for Mobile Robots, Theory and Application", A. K. Peters, Ltd., 1995.

Pointers

- https://edu.epfl.ch/coursebook/en/embedded-systems-and-robotics-MICRO-315
- Real e-puck website: https://e-puck.gctronic.com/
- Webots User Guide: https://www.cyberbotics.com/doc/guide/index
- Webots Reference Manual: https://www.cyberbotics.com/doc/reference/index