Simulation of Autonomous Robots

2008-01-29

Lecture 3

Why use simulations?

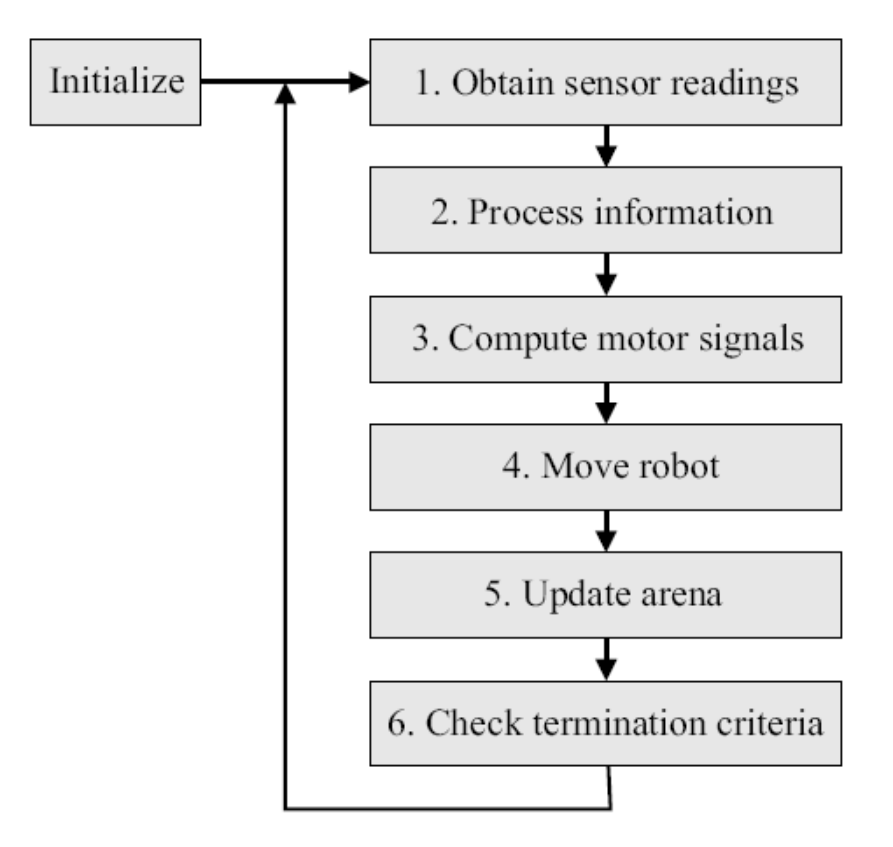
- Test the robot's behavior:
  - Catastrophic failure can be detected.
- Rapid prototyping:
  - Building robots can be very costly
  - Iterate the design using a simulator
- Designing robots in connection with EAs:
  - Huge number of evaluations is required!

Can I get my free lunch, please?

- *"the average performance of any pair of algorithms across all possible problems is identical"*

- Sorry, but there is **no free lunch** for you!
- Introducing simulators to robot development *might* introduce new problems as well!

Simulation flow

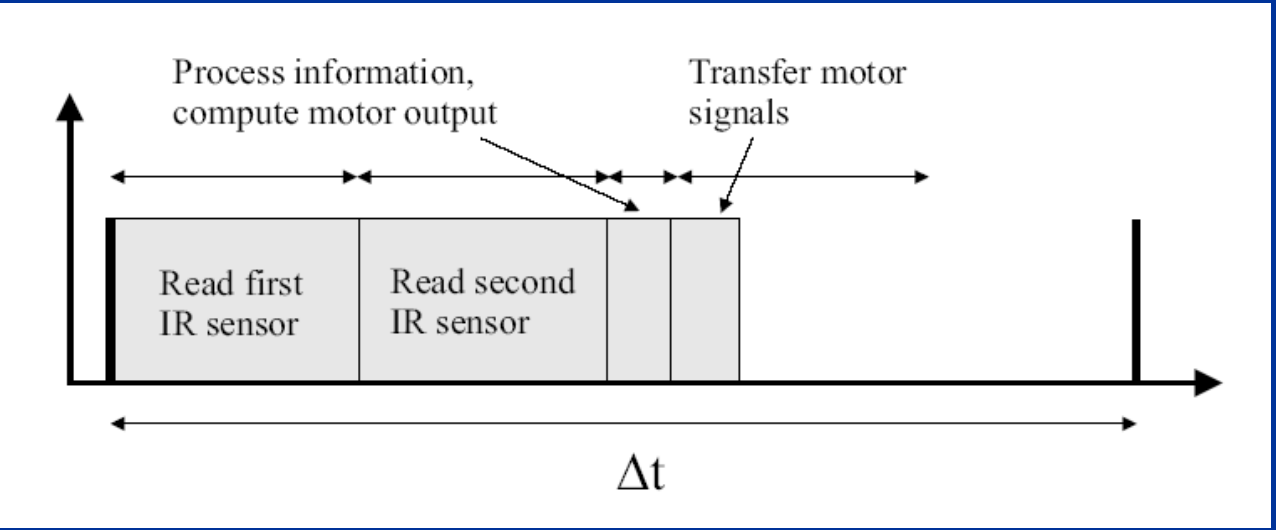


- Steps 1 - 6 is carried out in every timestep of the simulation

Issues: Timing

- **Timing of events:**
  - Time is **discretized** (Δt) in simulation, but a **continuous** variable (t) in real life.
  - Events can take *long* time to complete in simulation, but *short* time in a real robot, and vice versa.
    - **Collision-checking** is *slow* in simulation, but *fast* in a real robot.
    - **Reading of sensors** is *fast* in simulation, but *slow* in a real robot.

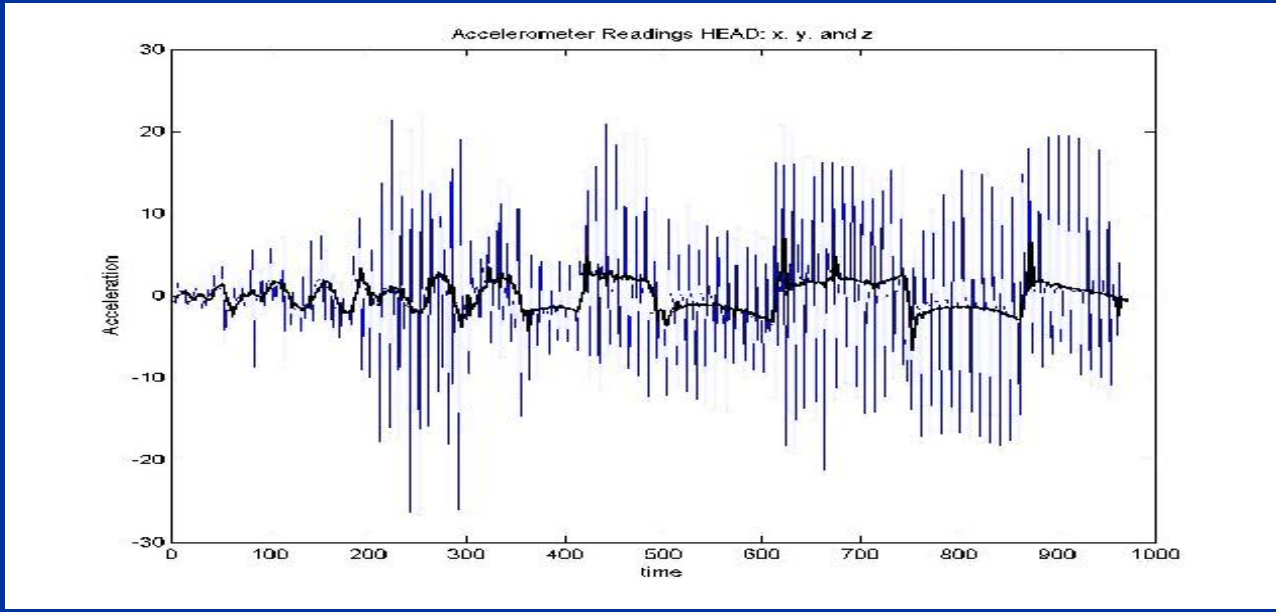
Timing diagram



- Time step Δt > time required in hardware
- Simulation too fast:
  => introduce **readability state** for each sensor
- Dynamics of motors etc. *must* be considered.

Issues: Noise

- Unwanted by-product of other activities:



- **Noise:** Real sensors (and actuators) are noisy => uncertainty in the signals (or actions)!

Issues: Noise

- Simulation *should* include a *realistic* level of noise
- Sensors (or motors) supposed to be identical often show different characteristics in practice.
- Note: EAs are good at exploiting noise and other "loopholes" in the simulation!

Issues: Sensing

- The simulated robot (brain) should *only* be provided with the *realistic* sensor information!
- For example, using wheel encoders:
  - *only* odometry data, *with errors*, is allowed!

Issues: Conclusions

- Simulations are *always* more or less *simplified* models of reality!
- Use sufficiently large timestep (Δt), compared with the hardware
  - However, reducing the timestep, Δt => better accuracy of the simulation
- Transfer of simulation results to real, physical robots is far from trivial => Overcome the **Reality gap!**

Robot simulators

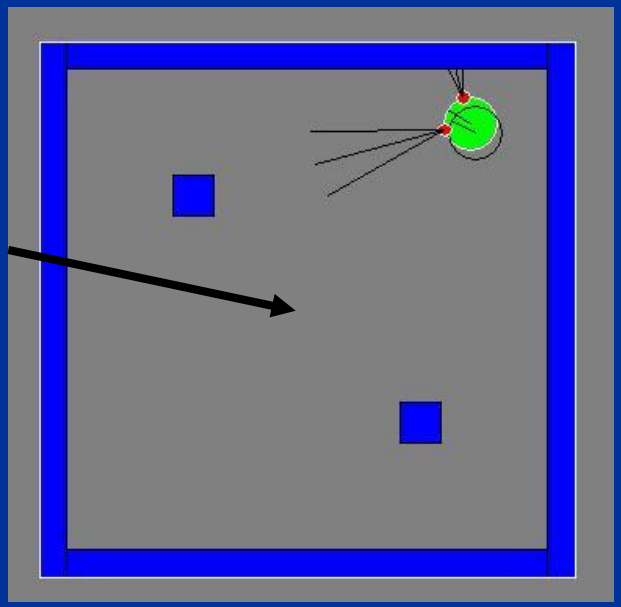
- ARSim, Matlab-based, 2D, "simple".
  - Will be used in this course
- ERSim, based on ARSim
  - Will be used in this course
- UFLib
- EvoDyn
- ODE
- + many more, availible on the web

ARSim

- Based on Matlab (tested for versions 6.5.3 and 7.0.14)
- Two-dimensional simulator
- Simulation of two-wheeled, differentially steered robots with circular bodies
- Current version supports IR sensors and odometry only

Components of ARSim

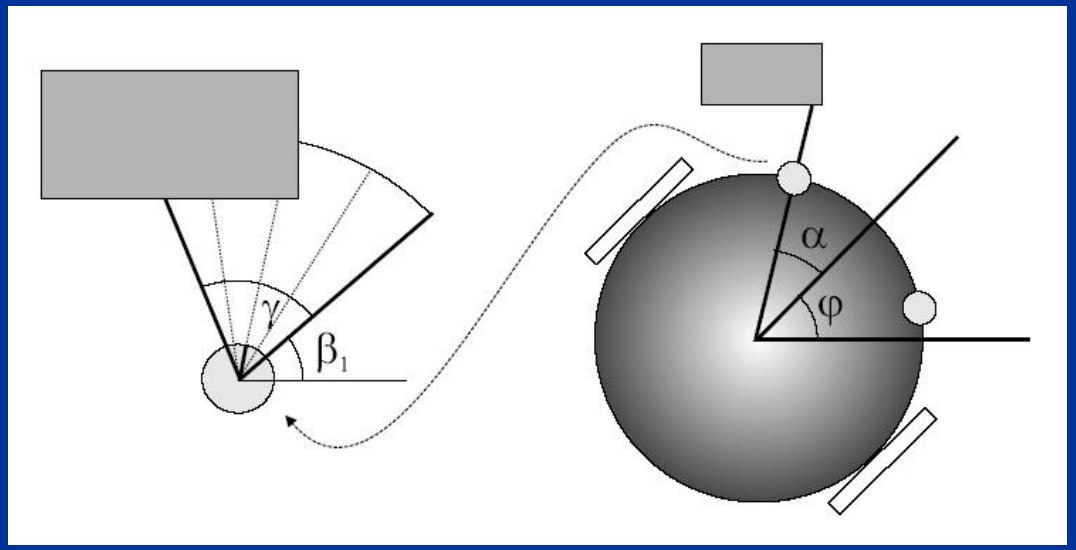
- Arena:



  - All objects represented as polygons => user specifies the coordinates: [-1 1]; [-1 0.7]; [-0.7 0.7]; [-0.7 1];
  - Polygons of arbitrary shape are allowed!

Components of ARSim

- IR sensors:
  - **Simplified** model, based on ray-traycing



φ is current heading of the robot, α is relative direction of the IR sensor, and γ is opening angle of sensor

Components of ARSim

- IR sensors:
  - The i:th ray is emitted in the direction β_i:

$$\beta_i = \varphi + \alpha - \frac{\gamma}{2} + (i-1)\delta\gamma,$$

  - where $\delta\gamma$ is given by:

$$\delta\gamma = \frac{\gamma}{N-1},$$

  - and N is the number of rays

Components of ARSim

- IR sensors:
  - reading of the sensor for ray *i,* provided that $d_i < r$:

$$\rho_i = \min\left(\cos\kappa_i\left(\frac{a}{d_i^2}+b\right),1\right),$$

  - where r is sensor range, d_i distance to nearest object, and:

$$\kappa_i = -\frac{\gamma}{2} + (i-1)\delta\gamma.$$

Components of ARSim

- IR sensors:
  - Total reading of the IR sensor is given by:

$$s = \frac{1}{N} \sum_{i=1}^{N} \rho_i .$$

  - NOTE: In the standard version of ARSim, *no* sensory noise is added!

Components of ARSim

- Odometry:
  - displacement of the robot in one timestep:

$$\Delta x_L = v_L \Delta t + \epsilon N(0, \sigma_o),$$

$$\Delta x_R = v_R \Delta t + \epsilon N(0, \sigma_o),$$

where $N(0, \sigma_0)$ is normally distributed random numbers

  - The change in position and direction of the robot is then computed according to the **kinematic eqs** (1.44-1.47).

- NOTE: Actual *wheel encoders* are **not** simulated!

Components of ARSim

- DC motors:
  - *Simplified model* : electrical and mechanical dynamics are neglected.
  - Torque acting on the motor shaft:

$$\tau = \tau_g - c_C \mathrm{sgn}(\omega) - c_v \omega, \; \tau_g = \frac{c_t}{R} V - \frac{c_e c_t}{R} \omega,$$

  - where c_t , c_e , c_v , and c_C are constants.

  - Furthermore **gears** are also implemented in ARSim =>

$$\tau_{\mathrm{out}} = G\tau,$$

Components of ARSim

- Generating the motion:
  - By numerical **integration** of the *equations of motion:*

$$M\dot{V} + \alpha V = A\left(\tau_{\mathrm{L}} + \tau_{\mathrm{R}}\right),$$

$$\overline{I}\ddot{\varphi} + \beta\dot{\varphi} = B\left(-\tau_{\mathrm{L}} + \tau_{\mathrm{R}}\right).$$

  - For each timestep $\dot{V}$ and $\ddot{\varphi}$ are computed.
  - Compute new values of V and $\dot{\varphi}$ a

$$V' = V + \dot{V}\Delta t,$$

$$\dot{\varphi}' = \dot{\varphi} + \ddot{\varphi}\Delta t,$$

Components of ARSim

- Generating the motion:
  - Update the value of φ using: $\varphi' = \varphi + \dot{\varphi}'\Delta t,$
  - The components of the velocity are then obtained:

$$V'_x = V' \cos\varphi, \quad V'_y = V' \sin\varphi.$$

  - The new position is then computed as:

$$X' = X + V'_x \Delta t, \quad Y' = Y + V'_y \Delta t.$$

Components of ARSim

- Robotic brain (controller):
  - Processing of information => generates the **behavior** of the robot!
  - Information processing takes place in the `BrainStep` function
  - Implementation: FSM, ANN, RNN, UF method, if-then-else rule, etc. etc.

Using ARSim

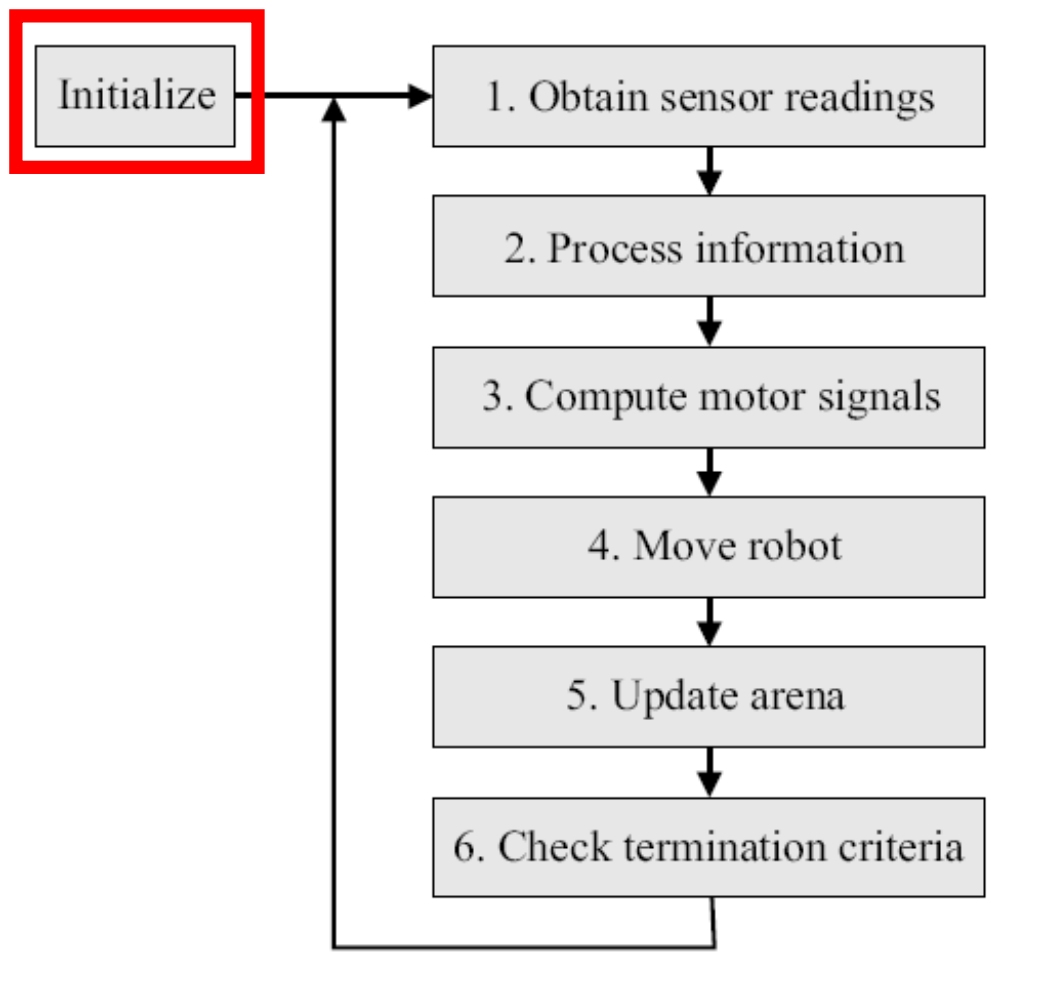
- Download the program:

  `http://www.am.chalmers.se/~wolff/AA/Software/ARSim_v1.1.8.zip`

- Unzip the fil

- Start Matlab, move to the `ARSim_v1.1.8` directory, and write:
  `>>TestRunRobot`
  in the command window and press [enter]

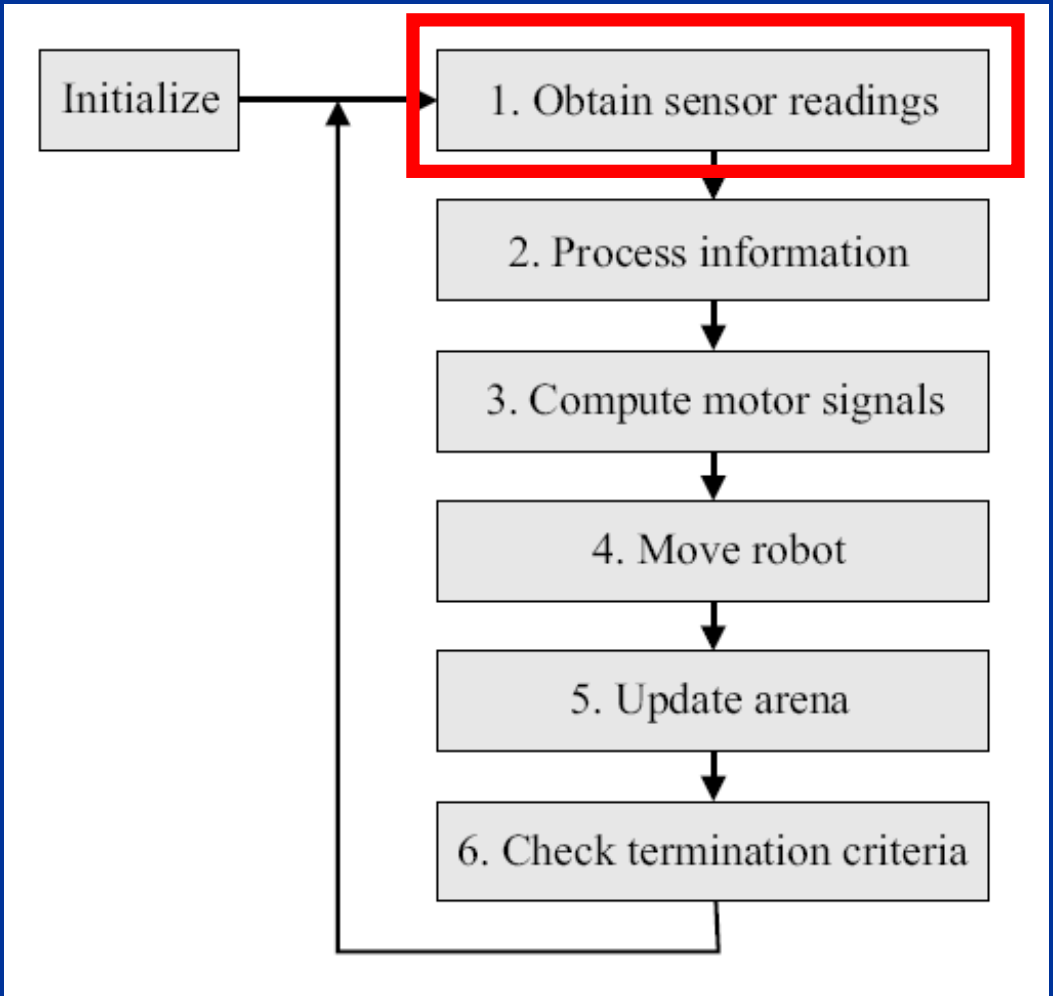
Using ARSim: The ARSim API



Using ARSim: The ARSim API

- `CreateArena`
  - Generates an arean
- `CreateBrain`
  - Generates the brain of a robot
- `CreateIRSensor`
  - Creates an IRSensor
- `CreateMotor`
  - Generates a DC motor
- `CreateRobot`
  - Generates a DC motor

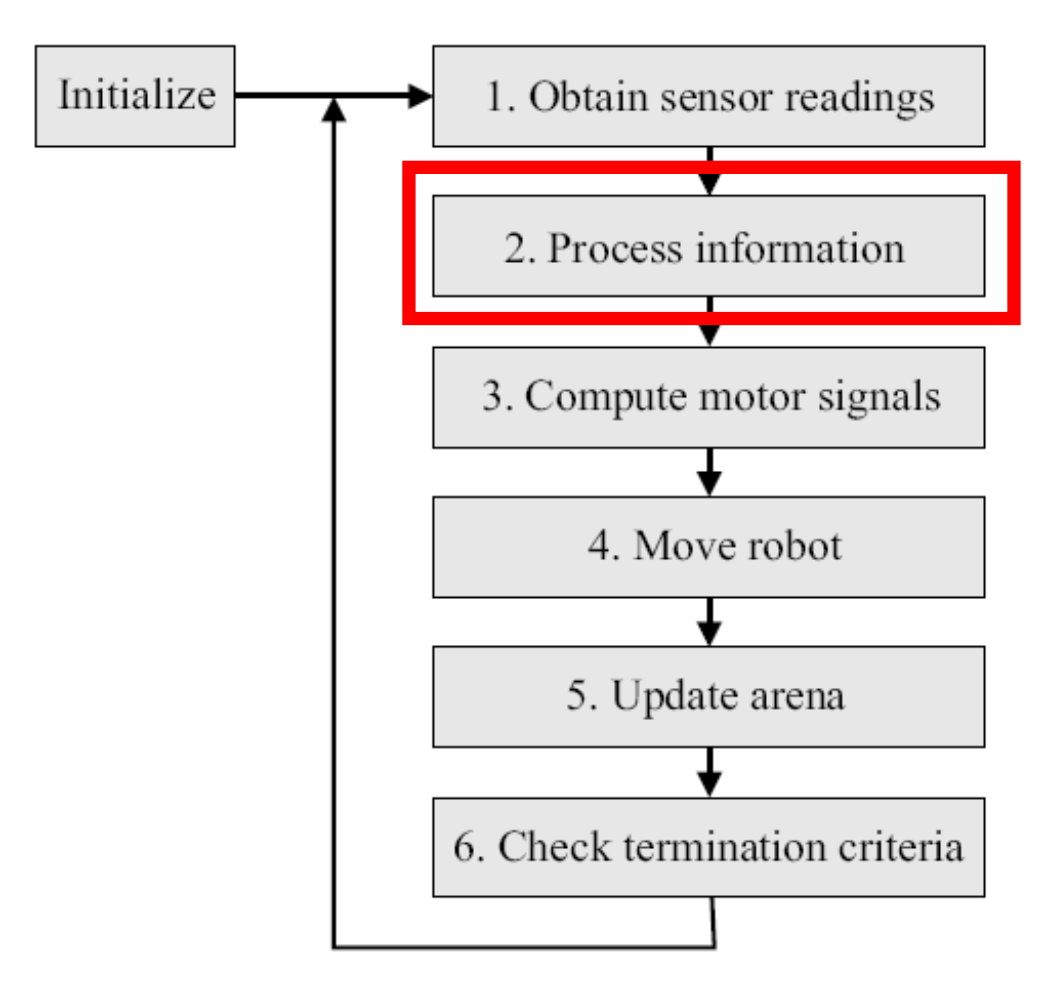
Using ARSim: The ARSim API



Using ARSim: The ARSim API

- `GetSensorReadings`
  - Obtains the reading of all (IR) sensors of the robot
- `GetIRSensorReading`
  - Determines the reading of an IRsensor of the robot
- `GetOdometerReading`
  - Updates the odometer readings

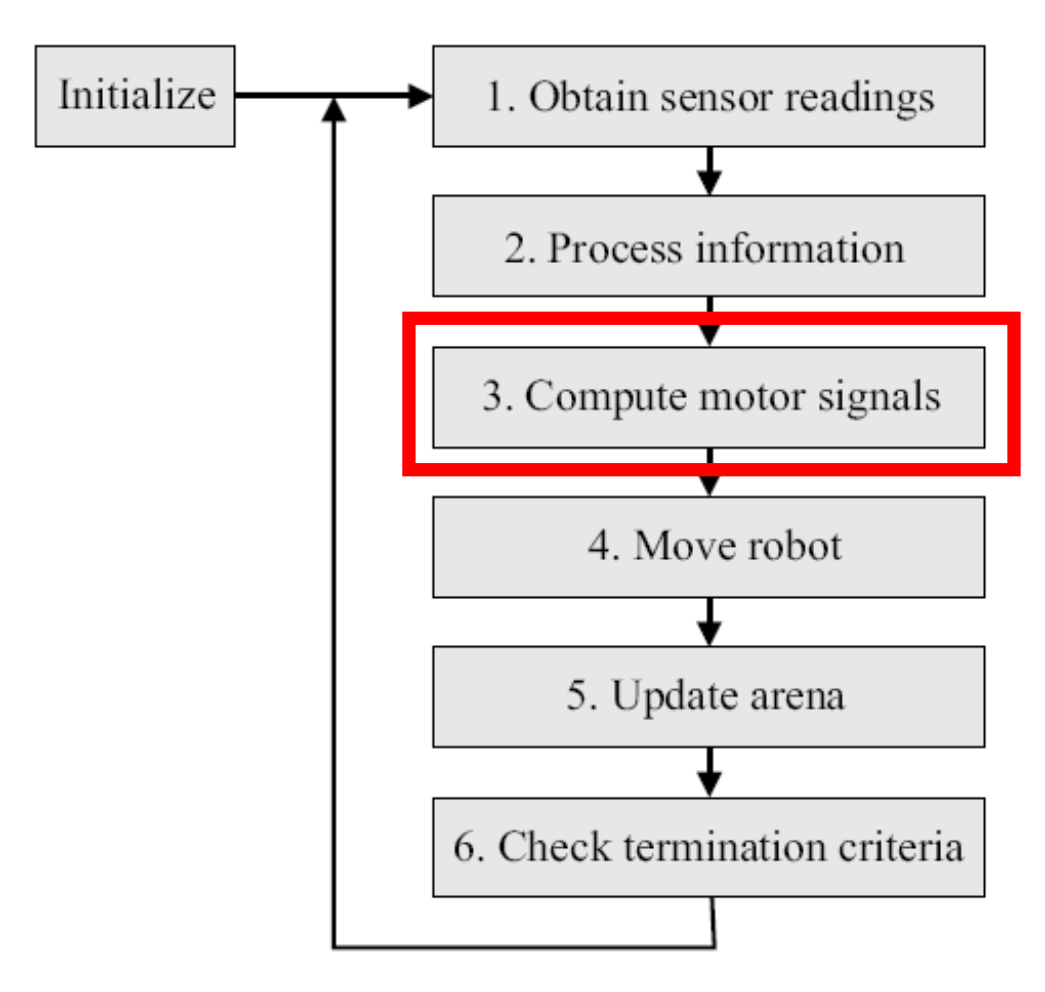
Using ARSim: The ARSim API



Using ARSim: The ARSim API

- `Brainstep`
  - Function that *implements* the decision-making system (brain) of the robot

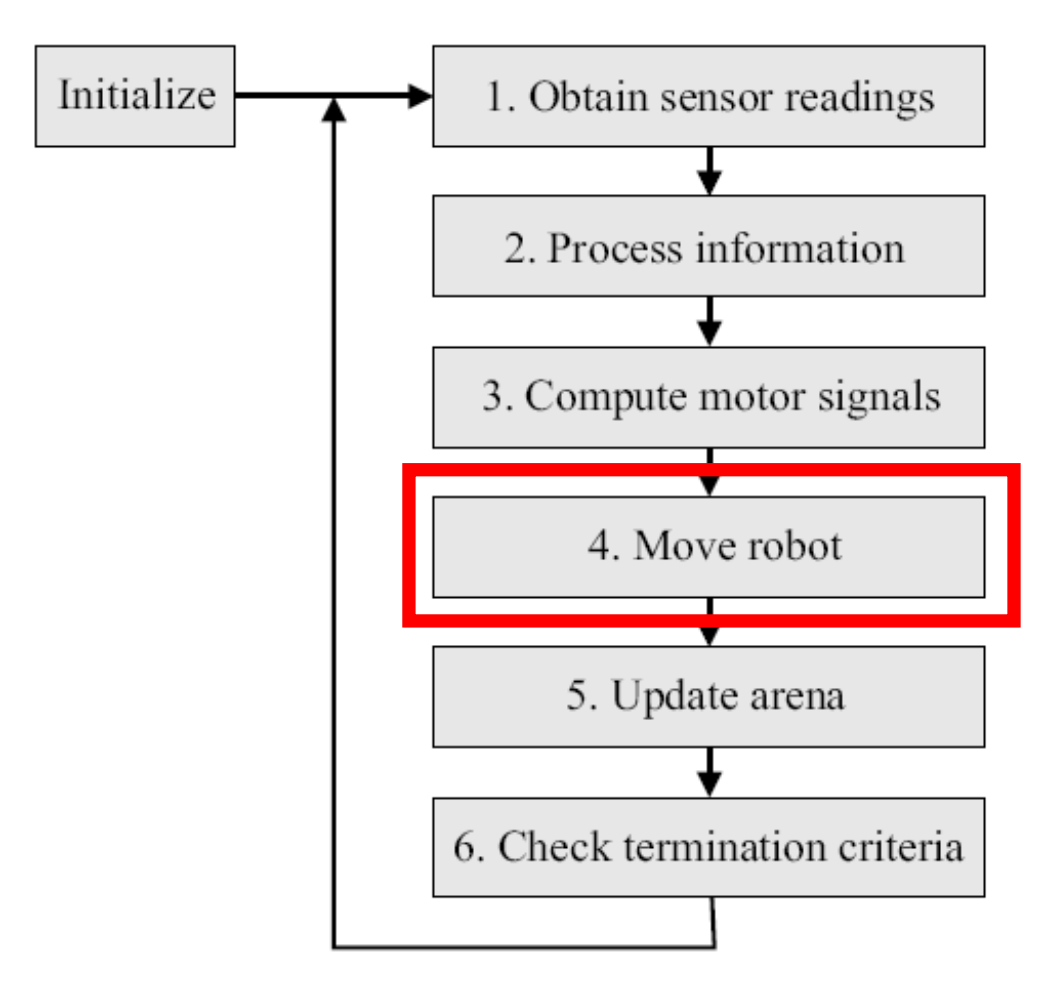
Using ARSim: The ARSim API



Using ARSim: The ARSim API

- `GetMotorSignalsFromBrain`
  - Extracts the motor signals from the brain of the robot

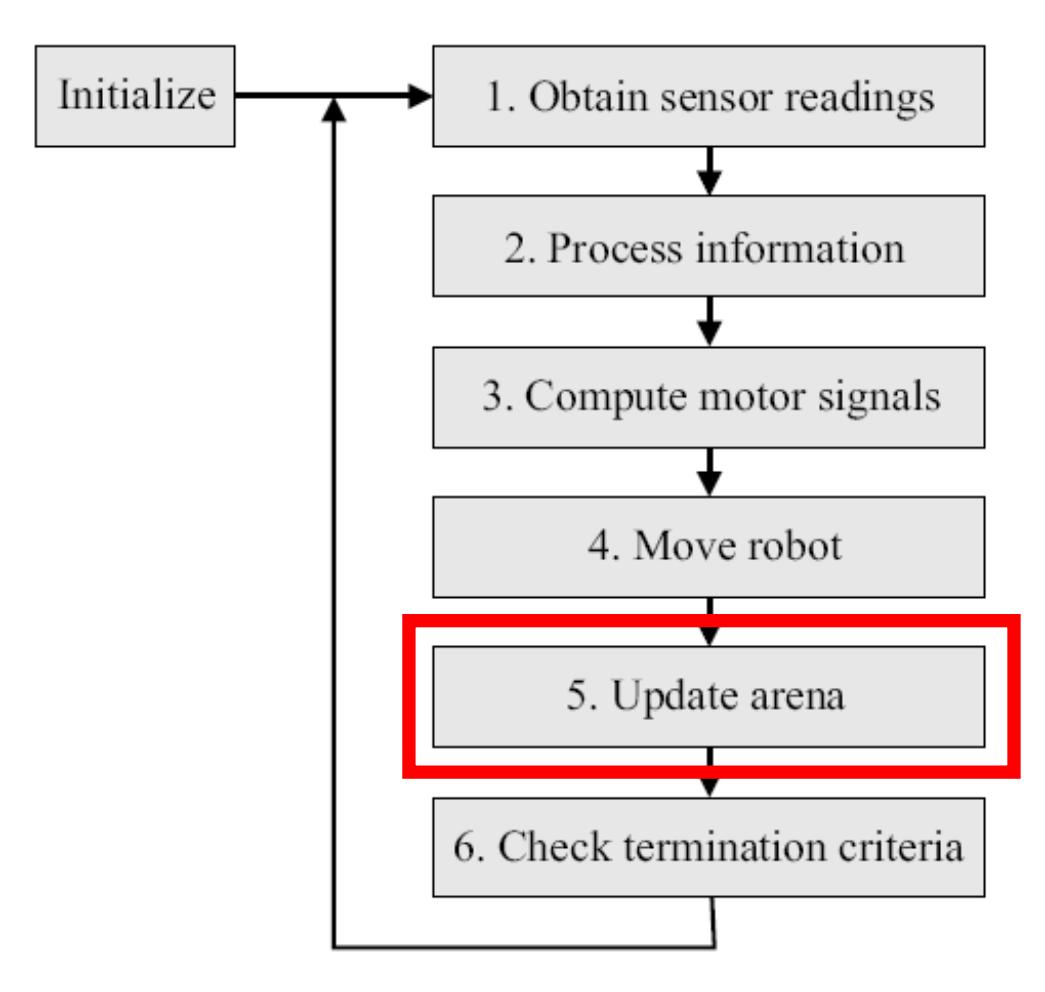
Using ARSim: The ARSim API



Using ARSim: The ARSim API

- `MoveRobot`
  - Moves the robot according to the equations of motion for a differentially steered two-wheeled robot

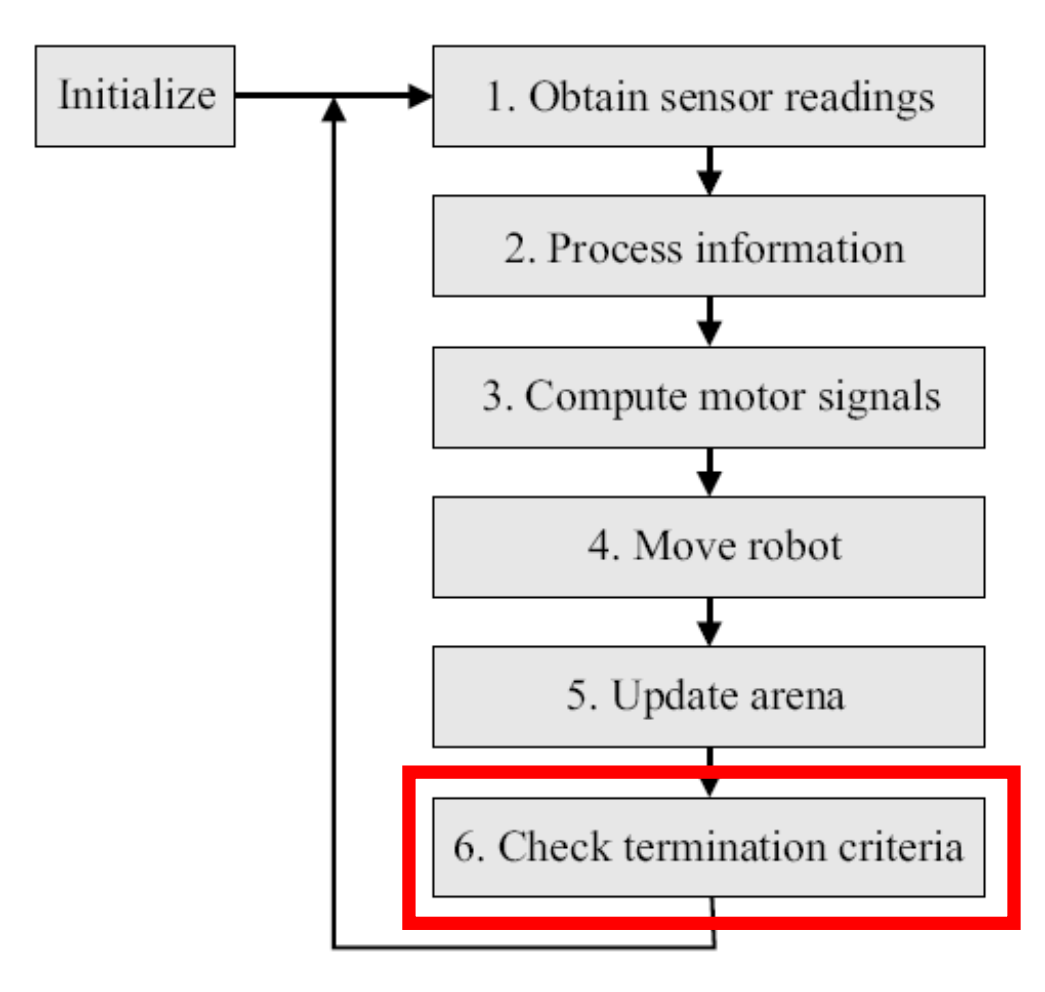
Using ARSim: The ARSim API



Using ARSim: The ARSim API

- `AddMotionResults`
  - Updates the motion results by adding the current position, velocity, heading, and sensor readings of the robot
- `CheckForCollisions`
  - Makes a collision check by running through all arena objects line by line, and checking for intersections with the robot.
- `ShowRobot`
  - Updates the Matlab plot of the robot (and arena)

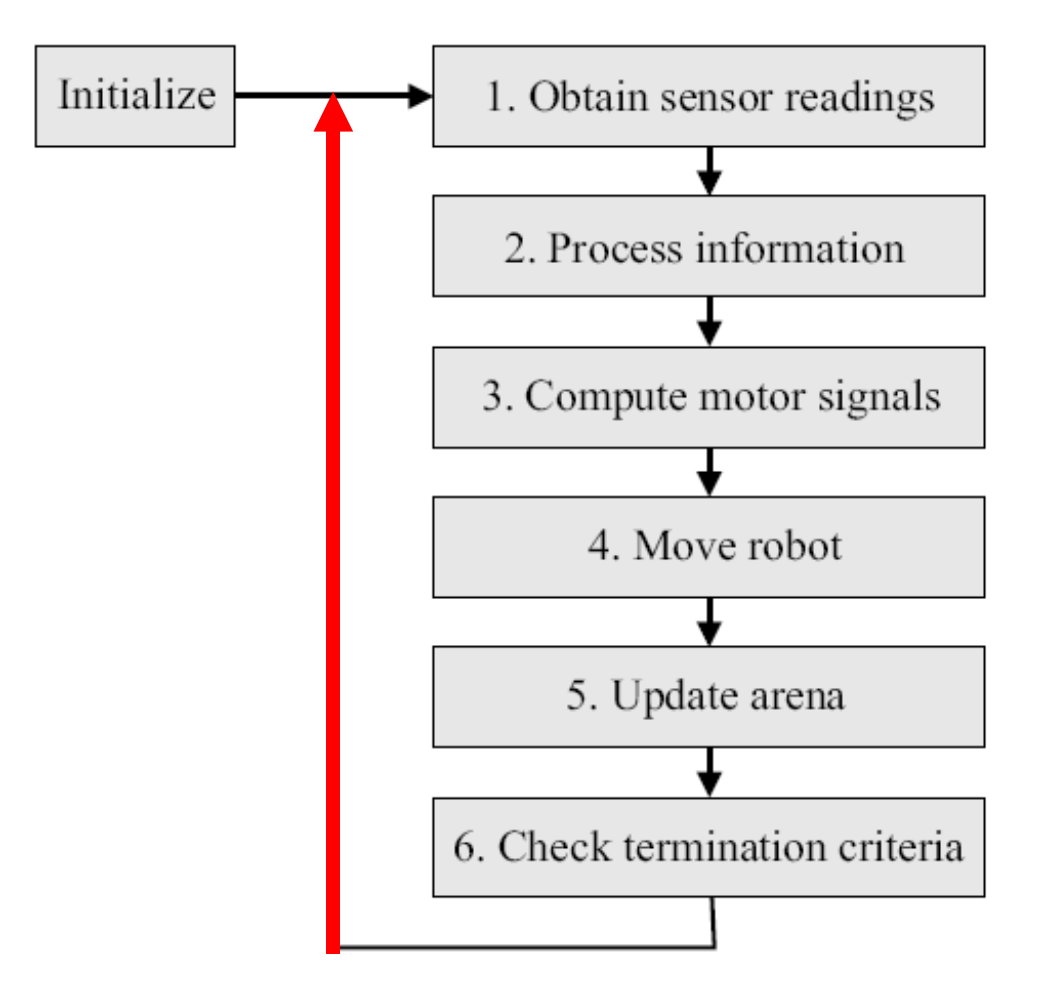
Using ARSim: The ARSim API



Using ARSim: The ARSim API

- Check the termination criteria:
  - Number of timesteps
  - Collision
  - etc.

Using ARSim: The ARSim API



Using ARSim

- User-defined robots and brains can be implemented in the following files:
  - `TestRunRobot.m`
  - `CreateBrain.m`
  - `BrainStep.m`
- **Do NOT edit any other files of ARSim!**

Using ARSim

- Have fun!

:-)

Next...

- Handout of Home problems, set 1
- Deadline: 2008-02-15, at 17.00!

- Remember, there is ***no lecture*** on Friday, February 1st!
  - Work on home problems instead...