Robotics and Autonomous Systems

Lecture 2: Mobile Robotics

Richard Williams

Department of Computer Science
University of Liverpool

Today

- Today we'll start to look at the main problems faced by mobile robots.
  - This sets up the issues we'll consider for the first half of the course.
- We'll also consider how these issues relate to the idea of agency.

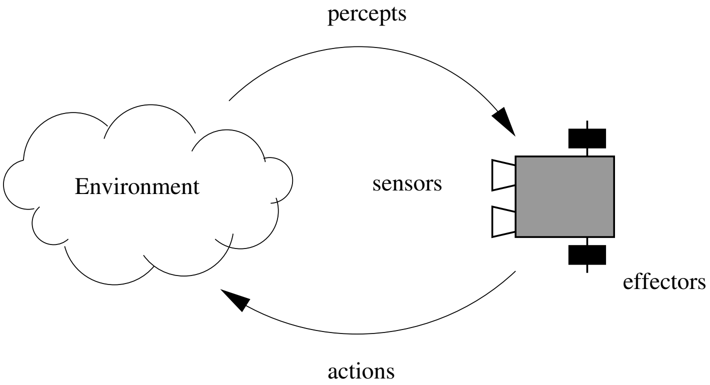
Autonomy

- Many autonomous vehicles are not really autonomous

- They are teleoperated.

Autonomy

- A really autonomous vehicle makes its own decisions about what to do.



- The notion of an agent can help us understand what this requires.

What is an agent?

- As we said before:

  An agent is a computer system that is situated in some environment, and that is capable of autonomous action in that environment in order to meet its delegated objectives.

- It is all about decisions
  - An agent has to choose what action to perform.
  - An agent has to decide when to perform an action.

What is an agent?

- Trivial (non-interesting) agents:
  - thermostat;
  - light switch;
  - UNIX daemon (e.g., biff).

- More interesting agents are intelligent.

Intelligent Agents

- An intelligent agent is a computer system capable of flexible autonomous action in some environment.
  By flexible, we mean:
  - reactive;
  - pro-active;
  - social.
- All these properties make it able to respond to what is around it.

Abstract Architectures for Agents

- Assume the world may be in any of a finite set E of discrete, instantaneous states:

$$E = \{e, e', \ldots\}.$$

- Agents are assumed to have a repertoire of possible actions available to them, which transform the state of the world.

$$Ac = \{\alpha, \alpha', \ldots\}$$

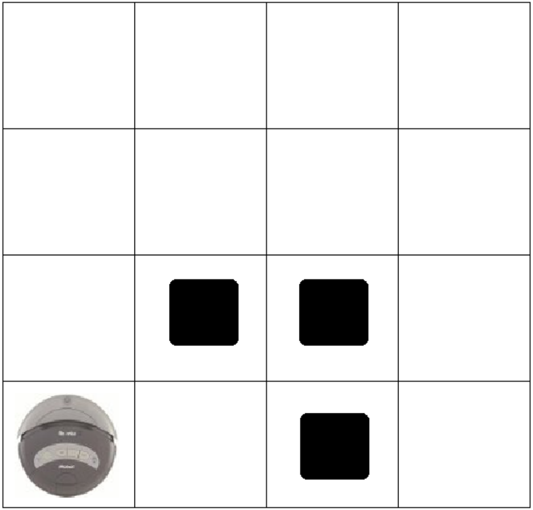
- Actions can be non-deterministic, but only one state ever results from an action.
- A run, r, of an agent in an environment is a sequence of interleaved world states and actions:

$$r : e_0 \xrightarrow{\alpha_0} e_1 \xrightarrow{\alpha_1} e_2 \xrightarrow{\alpha_2} e_3 \xrightarrow{\alpha_3} \cdots \xrightarrow{\alpha_{u-1}} e_u$$

Runs of agents

- When actions are deterministic each state has only one possible successor.
- A run would look something like the following:

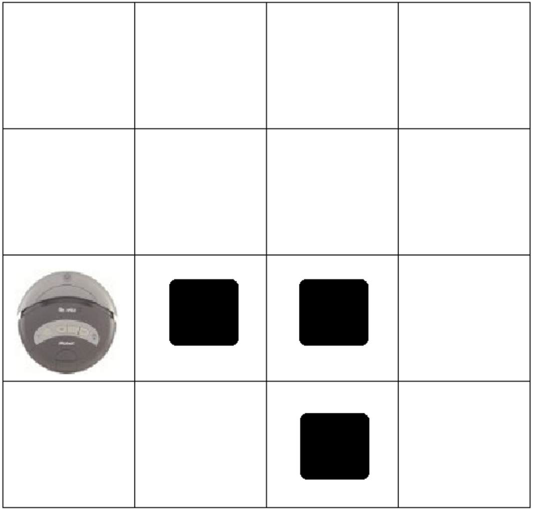
Runs of agents



Runs of agents

North!

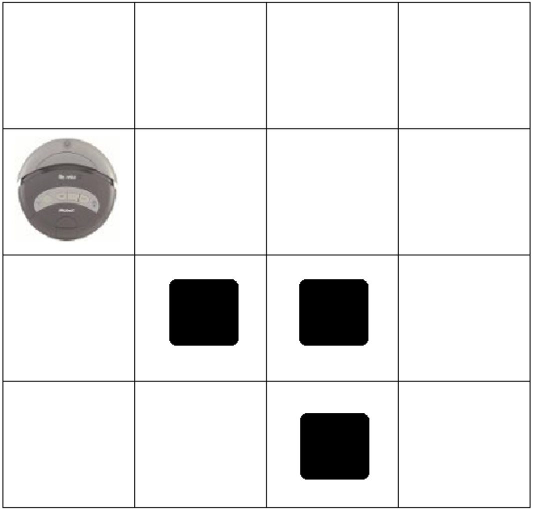
Runs of agents



Runs of agents

North!

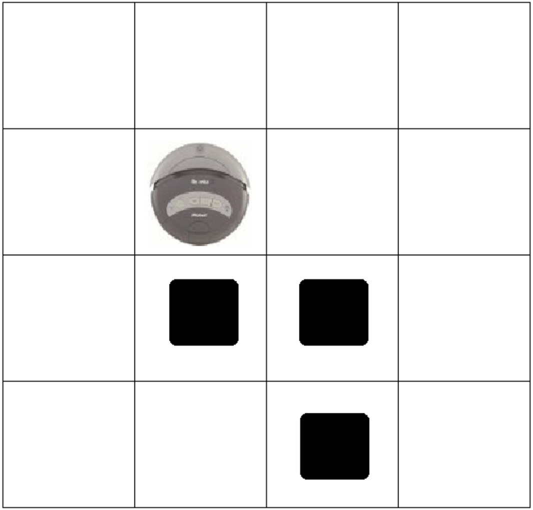
Runs of agents



Runs of agents

East!

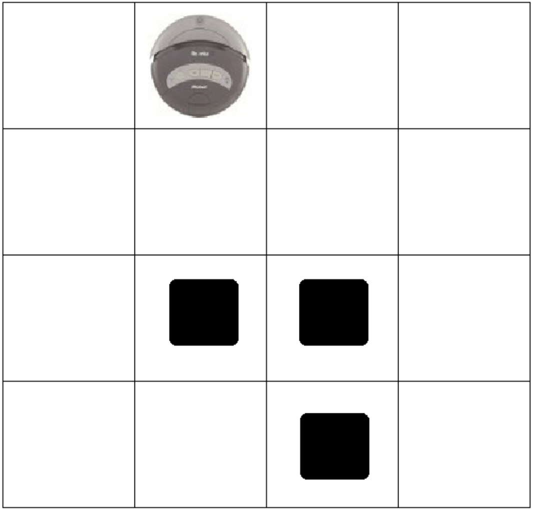
Runs of agents



Runs of agents

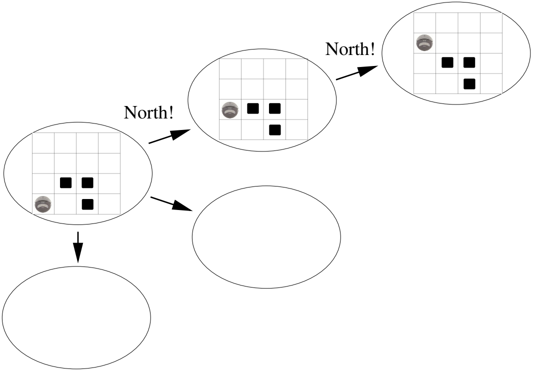
North!

Runs of agents



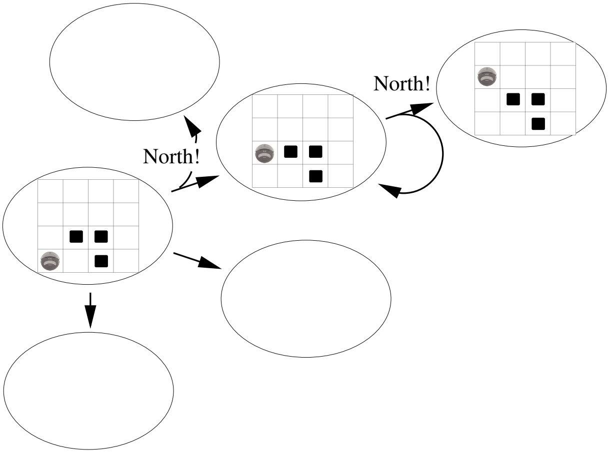
Runs of agents

- Which we might picture as so:



Runs of agents

- When actions are non-deterministic, a run (or trajectory) is the same, but the set of possible runs is more complex.



Runs of agents

- In fact it is more complex still, because all of the runs we pictured start from the same state.
- Let $\mathcal{R}$ be the set of all such possible finite sequences (over E and Ac) This is the set of all runs from all starting states.
- $\mathcal{R}^E$ is the subset of $\mathcal{R}$ that end with a state.
- All the ones where the agent needs to make a decision.

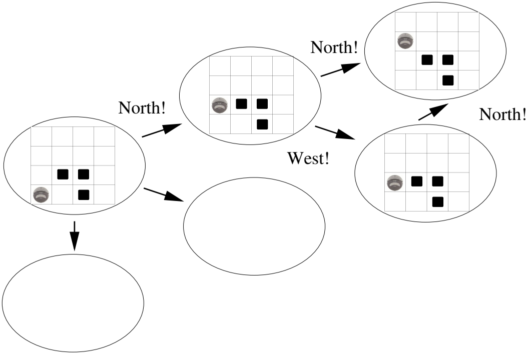
Agents

- We can think of an agent as being a function which maps runs to actions:

$$Ag : \mathcal{R}^E \rightarrow Ac$$

- Thus an agent makes a decision about what action to perform based on the history of the system that it has witnessed to date.

Agents



- Potentially the agent will reach a different decision when it reaches the same state by different routes.

Purely Reactive Agents

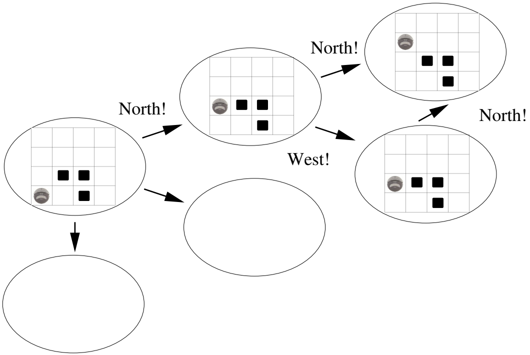
- Some agents decide what to do without reference to their history — they base their decision making entirely on the present, with no reference at all to the past.
- We call such agents purely reactive:

$$action : E \rightarrow Ac$$

- A thermostat is a purely reactive agent.

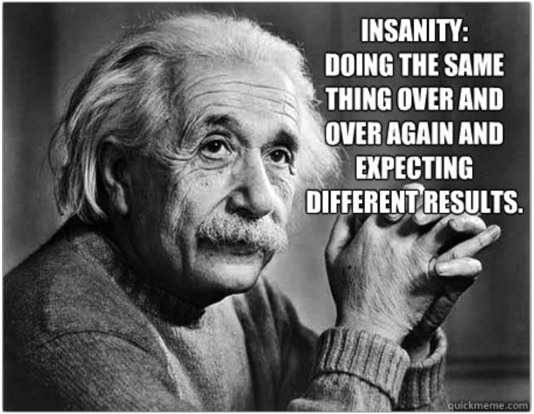
$$action(e) = \begin{cases} \text{off} & \text{if } e = \text{temperature OK} \\ \text{on} & \text{otherwise.} \end{cases}$$

Purely Reactive Agents



- A reactive agent will always do the same thing in the same state.

Purely Reactive Agents

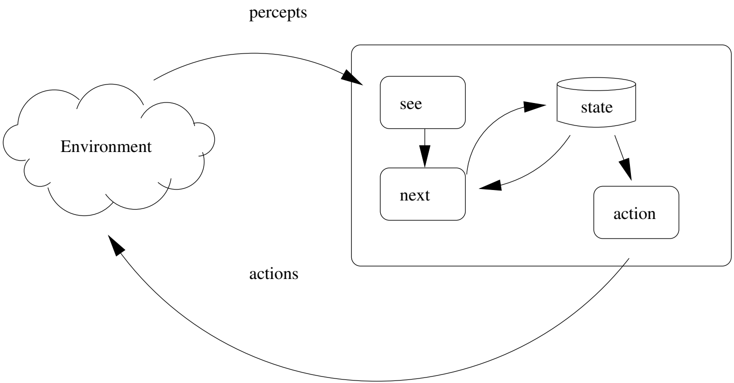


A reactive robot

- A simple reactive program for a robot might be:

  Drive forward until you bump into something. Then, turn to the right. Repeat.

Agents with state



Agents with state

- The *see* function is the agent's ability to observe its environment, whereas the *action* function represents the agent's decision making process.
- Output of the *see* function is a percept:

$$see : E \rightarrow Per$$

- The agent has some internal data structure, which is typically used to record information about the environment state and history.
- Let I be the set of all internal states of the agent.

Agents with state

- The action-selection function *action* is now defined as a mapping

$$action : I \to Ac$$

  from internal states to actions.
- An additional function *next* is introduced, which maps an internal state and percept to an internal state:

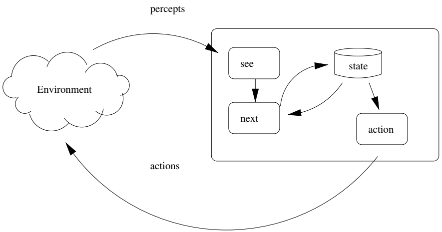
$$next : I \times Per \to I$$

- This says how the agent updates its view of the world when it gets a new percept.

Agents with state

1. Agent starts in some initial internal state i_0.
2. Observes its environment state e, and generates a percept $see(e)$.
3. Internal state of the agent is then updated via *next* function, becoming $next(i_0, see(e))$.
4. The action selected by the agent is $action(next(i_0, see(e)))$. This action is then performed.
5. Goto (2).

A robot with state



- *per* is a bool that indicates “against an object”.
- *i* is an integer, “against object for *n* steps”.
- *see* updates *per* each step, indicating if the robot is against an object.
- *next* is as follows:

$$next(i) = \begin{cases} \text{i+1} & \text{if } per = \text{true} \\ 0 & \text{otherwise.} \end{cases}$$

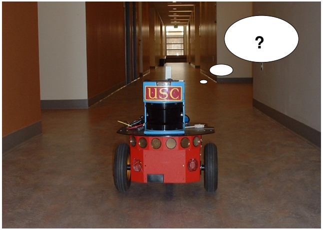
A robot with state

- Now the robot can take more sophisticated action.
- For example, backing up if it cannot turn away from the wall immediately.

- This is an example of a common situation in robotics.
- Trading memory and computation for sensing.

What is mobile robotics?

- Last time we boiled the challenges of mobile robotics down to:



  - Where am I ?
  - Where am I going ?
  - How do I get there ?
- Now we'll start talking about how to answer these questions.

The pieces we need

- Locomotion and Kinematics

  How to make the robot move, tradeoff between manoeverability and ease of control.

- Perception

  How to make the robot “see”. Dealing with uncertainty in sensor input and changing environment. Tradeoff between cost, data and computation.

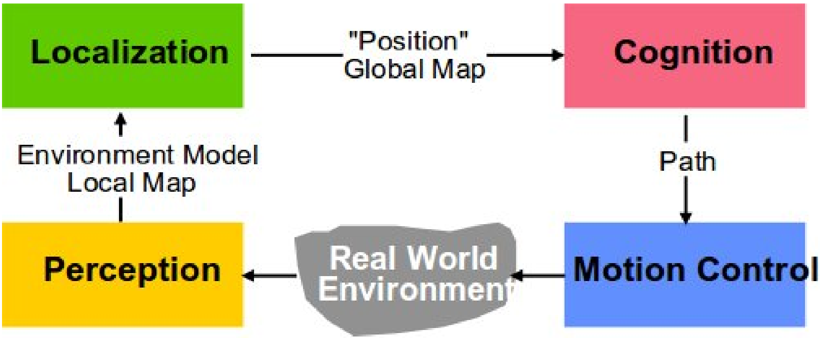
- Localization and Mapping

  Establish the robot’s position, and an idea of what it’s environment looks like.

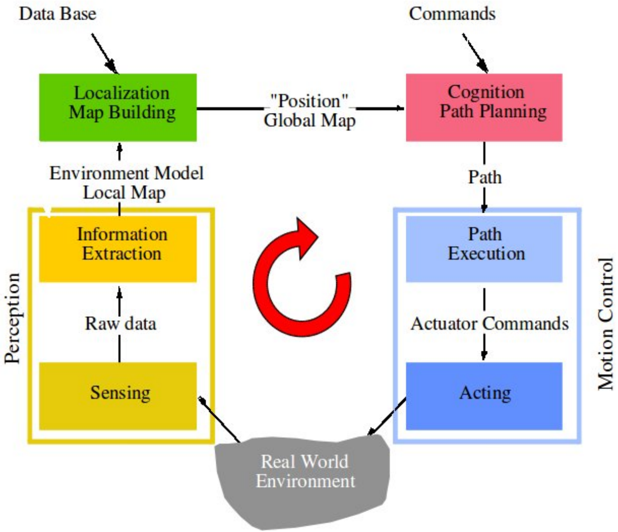
- Planning and navigation

  How the robot can find a route to its goal, how it can follow the route.

General control architecture



General control architecture



What makes it (particularly) hard

- Changing environment.
  - Things change.
  - Things get in the way.
- No compact model available.
  - How do you represent this all?
- Many sources of uncertainty.
  - All information comes from sensors which have errors.
  - The process of extracting useful information from sensors has errors

The basic operations

- We start with what the robot can “see”.

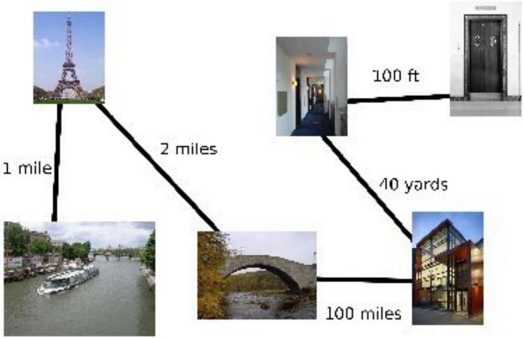
There are several forms this might take, but it will depend on:

What sensors the robot has

What features can be extracted.

- (These are not a particularly likely set of features.)

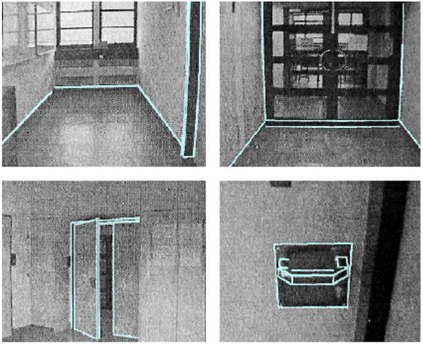
Map



- A map then says, for example, how these features sit relative to one another.

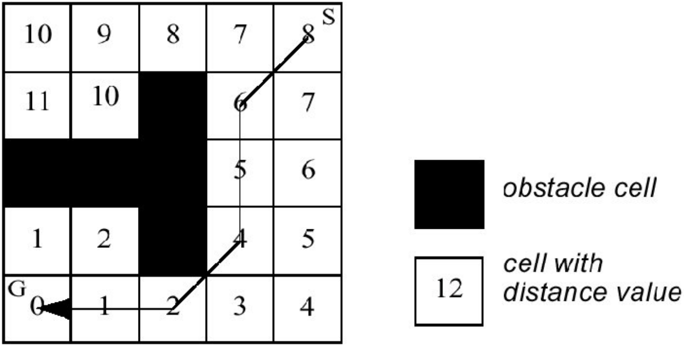
Localization

- A robot localizes by identifying features and the position in the map from which it could see them.



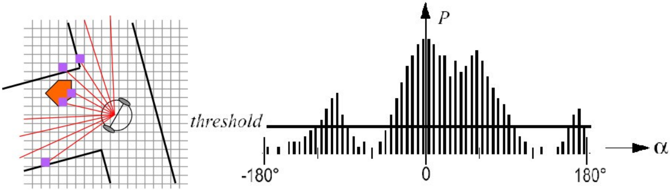
- Lanser et al (1996)

Navigation



- Navigation is then a combination of finding a path through the map. . . .

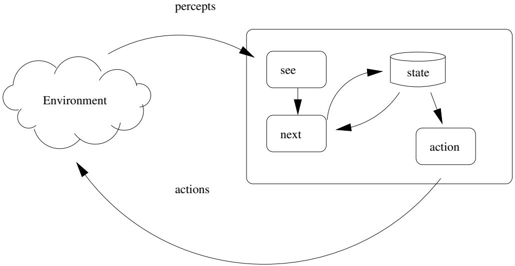
Navigation



- . . . and avoiding things that get in the things that get in the way.

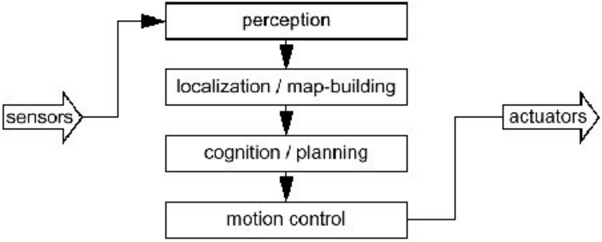
How do we put these pieces together?

- A system architecture specifies how these pieces fit together.
- Consider these to be refinements of the “agent with state” from above.



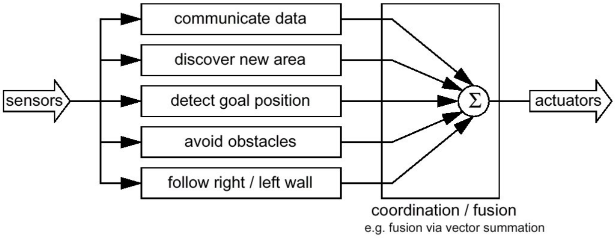
- Breaking down *next* and *action* into additional pieces.
- Adding in new aspects of state I.

Approach: Classical/Deliberative



- Complete modeling
- Function based
- Horizontal decomposition

Approach: Behaviour-based



- Sparse or no modeling
- Behavior based
- Vertical decomposition
- Bottom up

Approach: Hybrid

- A combination of the above.
- Exactly the best way to combine them nobody knows.
- Typical approach is:
  - Let “lower level” pieces be behavior based

    Localization
    Obstacle avoidance
    Data collection
  - Let more “cognitive” pieces be deliberative

    Planning
    Map building

Summary

- Last time we talked about what the main challenges of mobile robotics are.
- This lecture started to describe how we can meet these challenges.
- We covered the main things we need to be able to autonomously control a robot.
- Along the way we looked at how notions of agency — and what this means for autonomy — can help.