G54DIA:
Designing Intelligent Agents

Lecture 7: Hybrid Architectures I

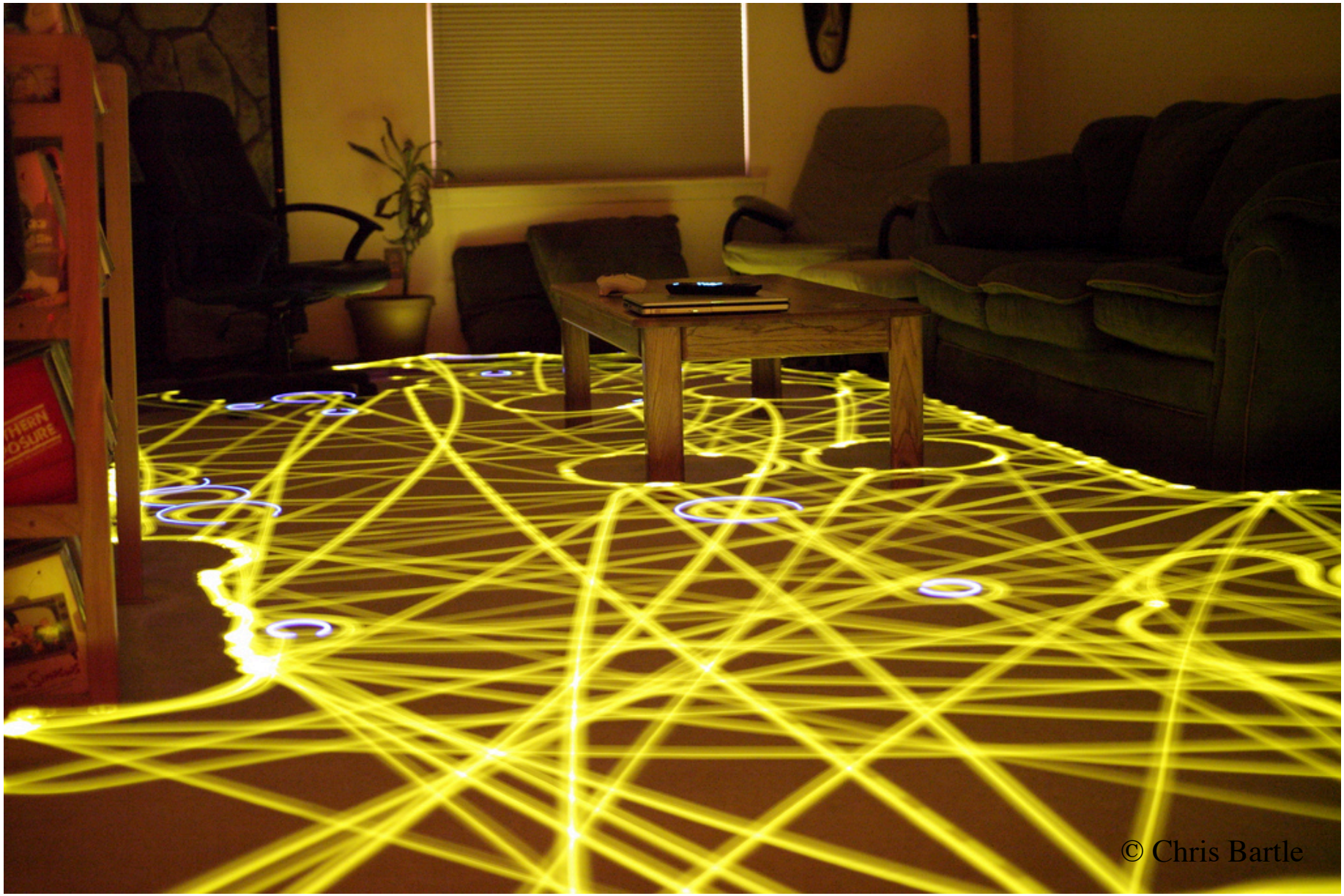
Natasha Alechina
School of Computer Science
nza@cs.nott.ac.uk

Outline of this lecture

- comparison of reactive & deliberative architectures
- hybrid architectures
- problems of integrating representations and timescales
- problems of controlling interactions between components
- example:
  - TouringMachines

Example: Roomba

- simple robot vacuum cleaner
- infra-red & bump sensors for collision avoidance, stair sensor to detect drops and a dirt (dust) sensor
- executes a modified random walk with simple behaviours to avoid obstacles, circle in dirty areas and untangle from cables
- essentially a *reactive architecture*


© Chris Bartle

Advantages of reactive architectures

- a reactive architecture with state can produce *any* kind of behaviour
- requires no, or very simple, representations of the environment
- fast (often real-time) response to changes in the environment
- easy to produce agents to solve simple problems

Disadvantages of reactive architectures

- can't form complex representations, or consider alternative plans/ solutions to a problem
- every solution to every problem must be coded in advance, either by the designer of the system, or by evolution
  - each new behaviour added may interfere with existing behaviours
  - possible interactions between behaviours must be anticipated when designing or extending the system
- agent programs for complex problems can be *very* large

Reactive task environments

Reactive architectures are a good choice for task environments that are:

- **goals**: small number of simple (ascribed) achievement and maintenance goals, typically no constraints on how goals are achieved
- **percepts**: observable, dynamic, nondeterministic and continuous
- **actions**: may be fallible, may have differing utilities and costs, agent may be mobile but typically doesn't communicate with other agents

Example: Trilobite

- more complex robot vacuum cleaner
- sonar, infra-red & bump sensors for collision avoidance, stair sensor to avoid drops and a dirt sensor
- initially cleans along the edges of the room, building a map of the room and obstacles
- then plans and executes an 'optimum' path to clean the rest of the room
- essentially a *deliberative architecture*

Advantages of deliberative architectures

- allows us to code a *general procedure* for finding a solution to a *class of problems*
  - may be better than reactive systems at coping with novel problems
  - we may be able to get a correct or even an optimal answer
- useful when the penalty for incorrect actions is high, e.g., when the environment is hazardous

Disadvantages of deliberative architectures

- requires accurate models of the current state of the environment and how it will change
- hard to offer real-time guarantees on performance:
  - deliberation takes more time than simply reacting
  - deliberation takes an unpredictable amount of time

Deliberative task environments

Deliberative architectures are a good choice for task environments that are:

- **goals:** strongly committed to its top-level goals, goals are not time-dependent, may have constraints on how goals are achieved
- **percepts:** partially observable, static, deterministic, discrete
- **actions:** infallible, may have differing utilities and costs, agent is typically immobile, may communicate with other agents

Reactive *vs* Deliberative architectures

- reactive architectures have to code every solution to every problem in advance
- deliberative architectures allow us to code a general procedure for finding a solutions to a class of problems in advance
- a reactive architecture will typically require *less time and space* to solve any *single* problem instance than a deliberative architecture
- a deliberative architecture will typically be *more space efficient* than an equivalent reactive architecture since it can solve a class of problems in a fixed amount of space, whereas a reactive architecture requires space proportional to the number of problem instances

Hybrid architectures

- a *hybrid* architecture has both reactive and deliberative components
- hybrid architectures attempt to obtain the advantages of both reactive and deliberative architectures without their disadvantages
- the reactive and deliberative components are typically organised in *layers*:
  - the reactive components are responsible for relatively simple, low-level, robust behaviours
  - the deliberative components are responsible for organising and sequencing the reactive behaviours
- a key problem is *integration* of the reactive and deliberative layers

Differences in representations

- *reactive layer* typically uses very simple representations of the current or previous state of the environment, e.g., agent-centred, vector based representations

- *deliberative layer* uses complex counterfactual representations, for example representations of objects and their attributes in world coordinates

- how can the representations used by the deliberative layer be derived from the information at the reactive layer?

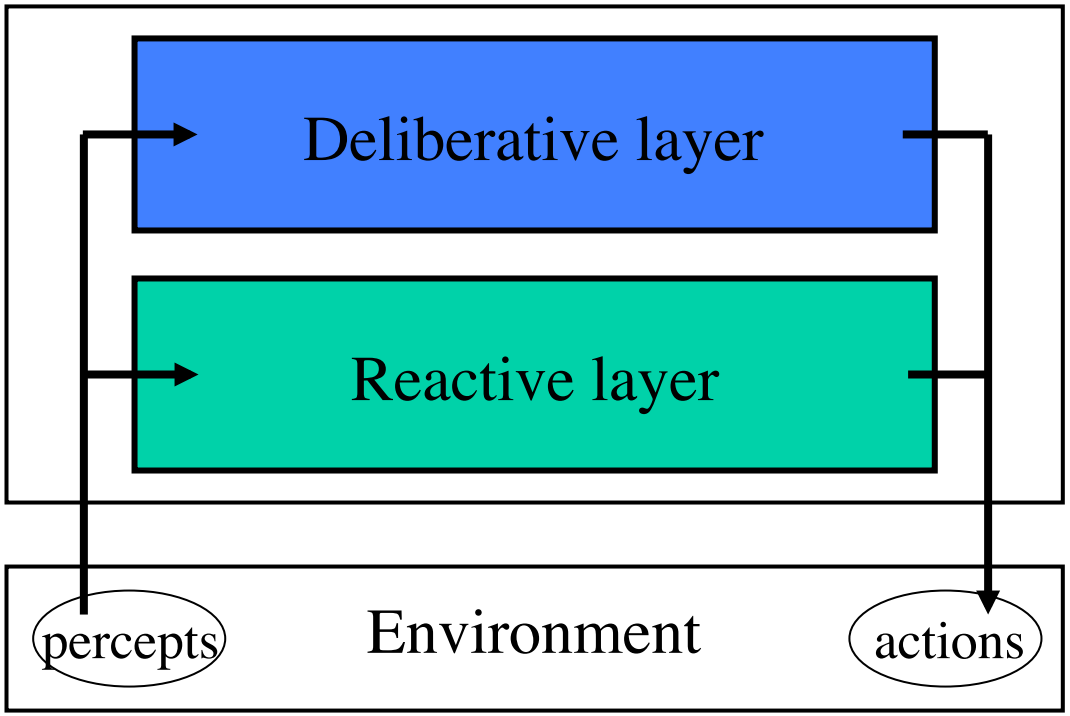
Differences in timescales

- *reactive layer* typically works over very short timescales in tight, sensor-motor feedback loops
- *deliberative layer* works on much longer timescales, from minutes to hours or even longer
- how can high-level actions at the deliberative layer be related to fine-grained actions at the reactive layer?

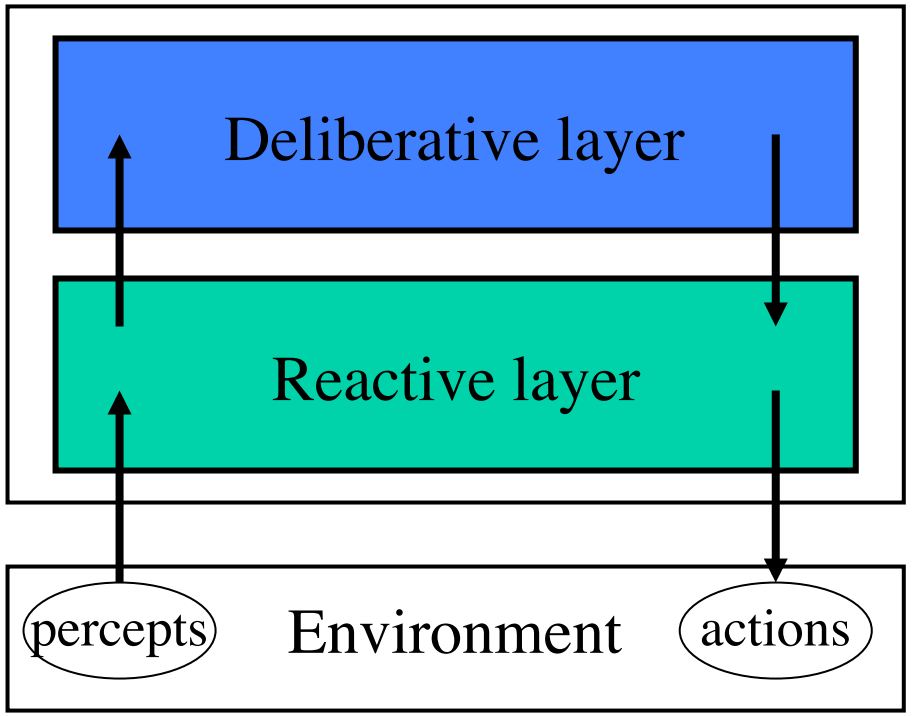
Control

- *decentralised control*: layers operate concurrently and independently, processing sensor data and generating actions
- *hierarchical control*: layers operate serially, with higher-level, deliberative layers controlling the execution of low-level reactive layers
- *concurrent control*: layers operate concurrently and can modify the behaviour of 'adjacent' layers

Decentralised control

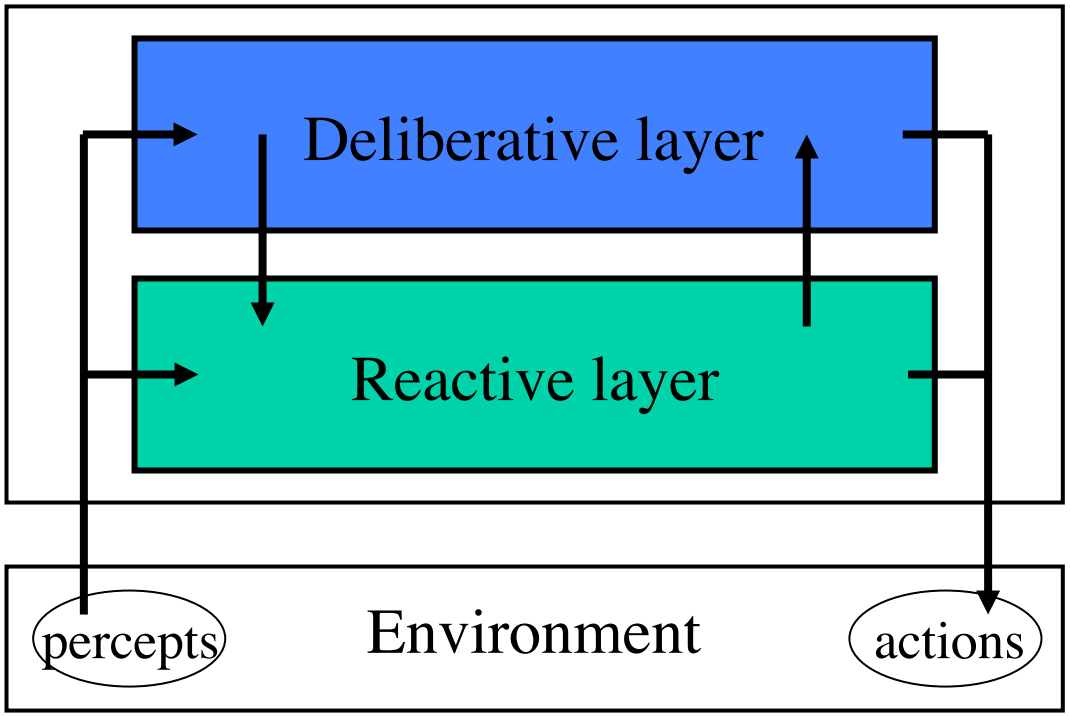


Hierarchical control



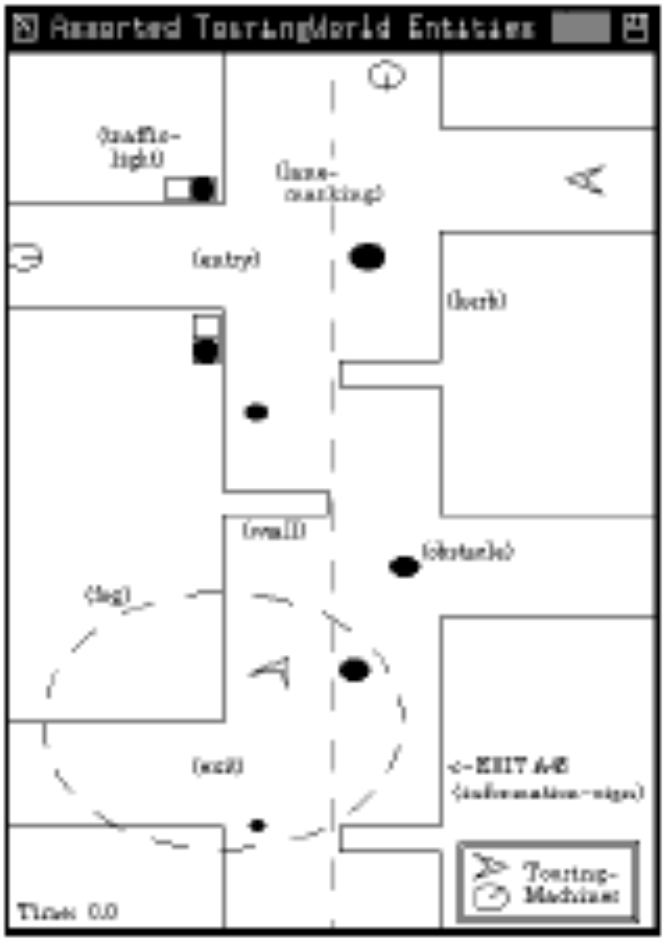
Example: Fred & Ginger

Concurrent control

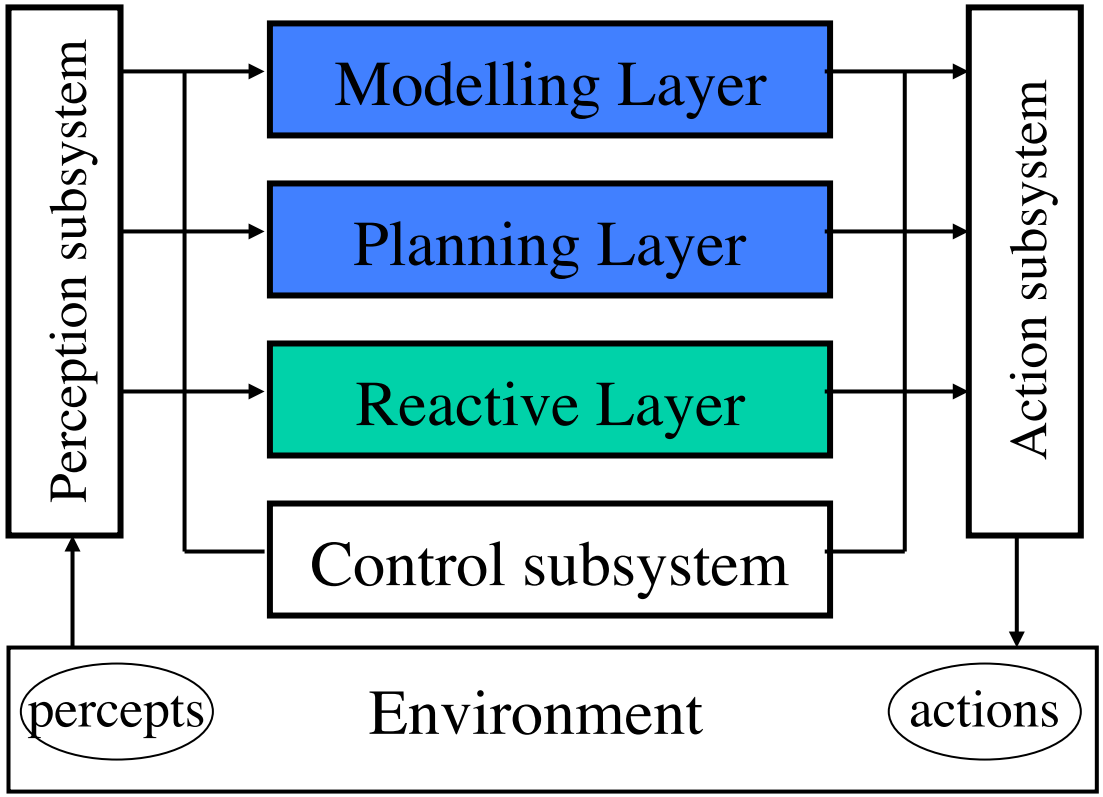


Example TouringMachines

- TouringMachines are autonomous (simulated) vehicles which drive along streets in the TouringWorld
  - the environment contains obstacles, traffic lights, rain (which affects braking distance) and fog (which changes the TouringMachines' visual field and range)
  - the TouringMachines goals are to reach a given location by a specified time, avoiding collisions with obstacles and other TouringMachines and obeying the traffic regulations.



TouringMachines architecture



Reactive layer

- responsible for producing an immediate response to changes in the environment
- e.g., obstacle avoidance
- implemented as a set of condition action rules which map percepts directly to actions.
- the rules can only refer to the agent's current state and they can't do any explicit reasoning about the world

Planning layer

- responsible for achieving simple goals, e.g., moving from place to place
  - implemented as library of predefined *plan schemas* which are elaborated at run time
  - to achieve a goal, the planning layer attempts to find a schema that matches that goal
  - if a schema contains subgoals, the planning layer attempts to find schemas in the plan library that match each sub-goal

Modelling layer

- responsible for representing other entities (agents) in the world, including the agent itself

  - predicts conflicts between agents and (autonomously) generates new goals to resolve these conflicts

  - these goals are passed to the planning layer which plans to achieve them in the normal way

Control subsystem

- responsible for deciding which of the reactive, planning and modelling layers should have control of the agent at any given time
- implemented as a set of *control rules* which can either
  - suppress percepts output by the perceptual subsystem; or
  - censor actions generated by the control layers
- e.g., a control rule may prevent the reactive layer from ever knowing that a particular obstacle has been perceived, if another layer is more appropriate for dealing with this type of obstacle

Integration

- the TouringMachines architecture can be viewed as hierarchical in the sense that there is a single subsystem (layer) which effectively makes all the control decisions

  - in this sort of architecture, the designer must potentially consider all possible interactions between the layers

  - if there are n layers and each layer can suggest k actions, this means that there are k^n interactions to be considered

The next lecture

- another example of hybrid architecture: Xavier
- early module feedback