2

Intelligent Control for Robots
Robots Architectures

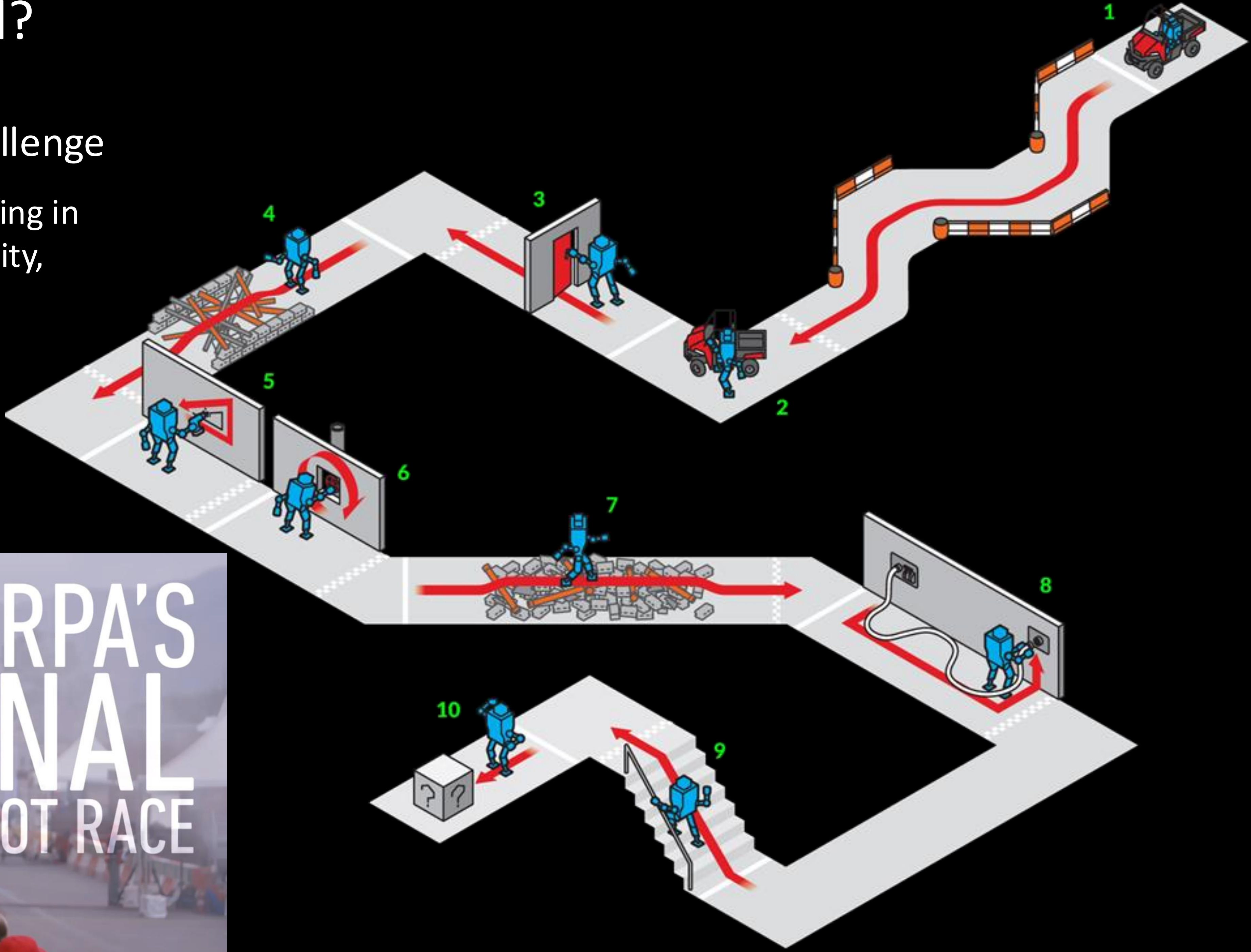
Agenda

1. Robot Architectures
2. Architecture Structures
   3.1 Deliberative Structure
   3.2 Reactive Structure
   3.3 Hybrid Structure
   3.4 Behavior Based Architecture
3. Architecture Styles
4. Exercise

Is robot control hard?

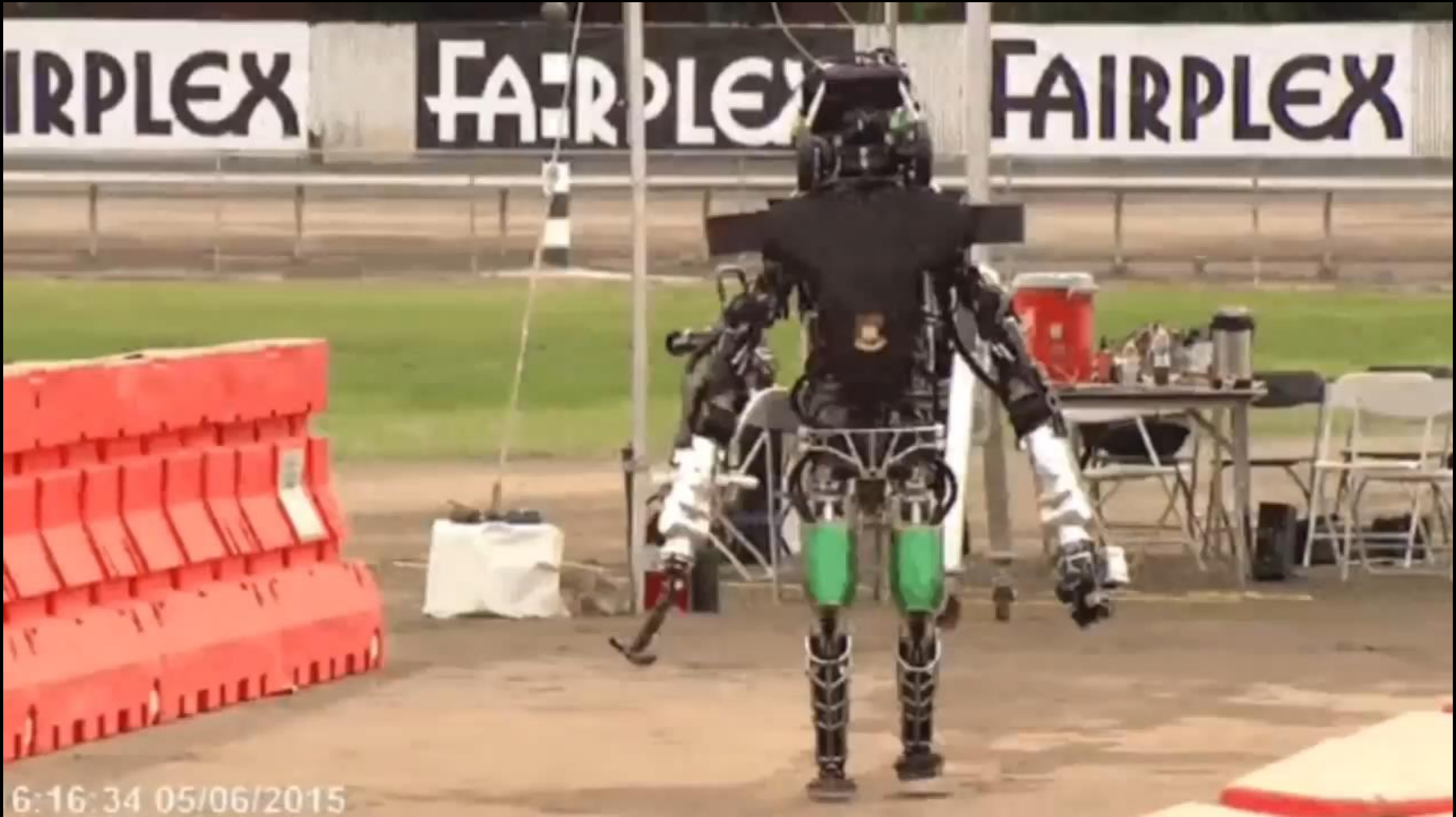
DARPA Grand Robotics Challenge

- 8 typical real-world tasks varying in degree of autonomy, complexity, difficulty etc.
- 60 minutes to solve all tasks

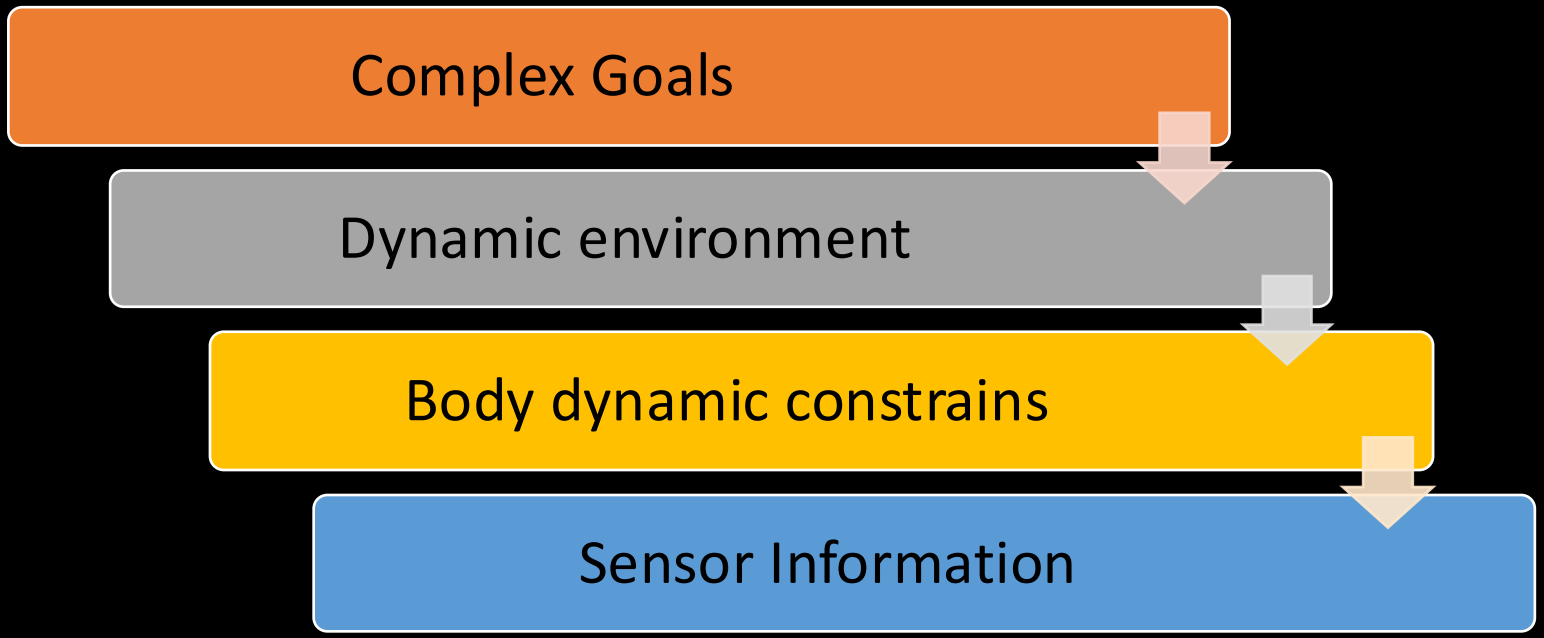


DARPA GRC 2015 participants

Answer: Robot control is hard…



Why is robot control hard?



Complex Goals

Dynamic environment

Body dynamic constrains

Sensor Information

Robot Architecture

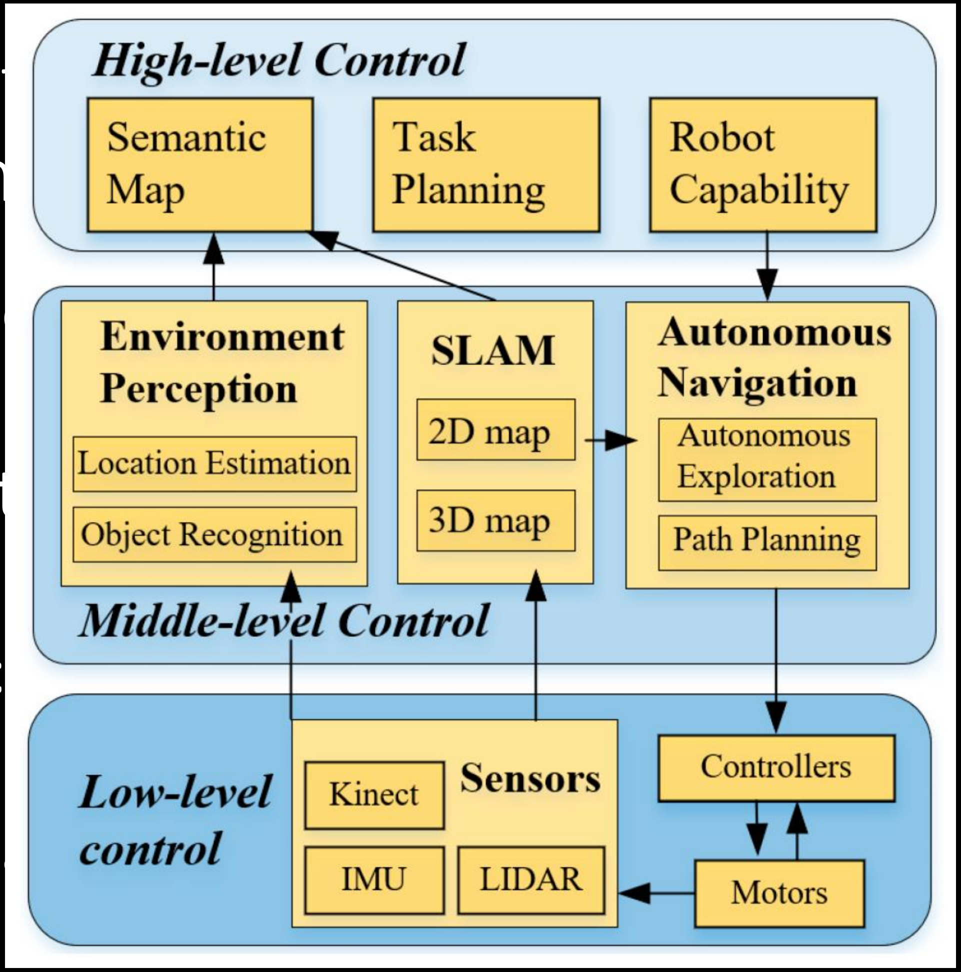
Structured framework used to conceptualize robot systems' software and hardware arrangements and interactions.

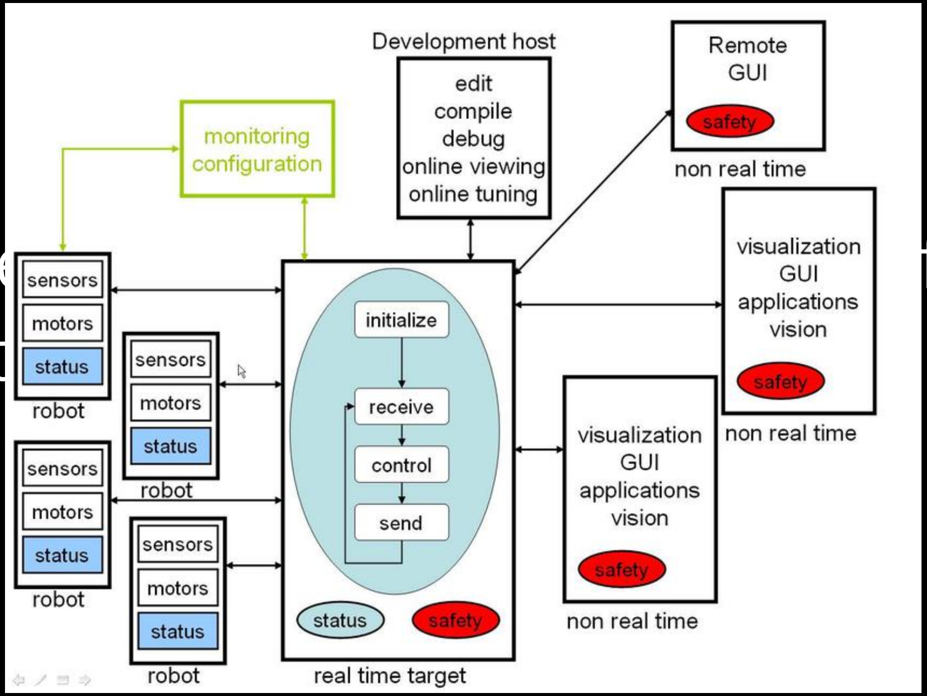
- Modularity
- Interoperability
- Scalability
- Maintainability
- Performance

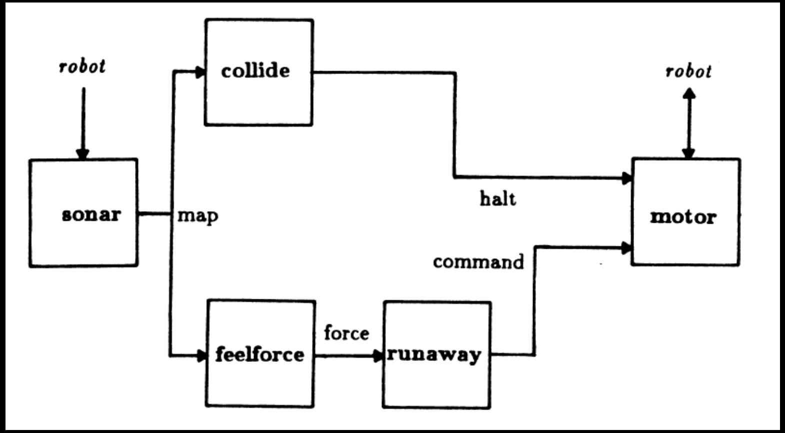
Robot Architecture

Struc nce ftware and h int

- M
- Int
- Sc
- M
- Performance







Paradigms for Robot Structural Architectures

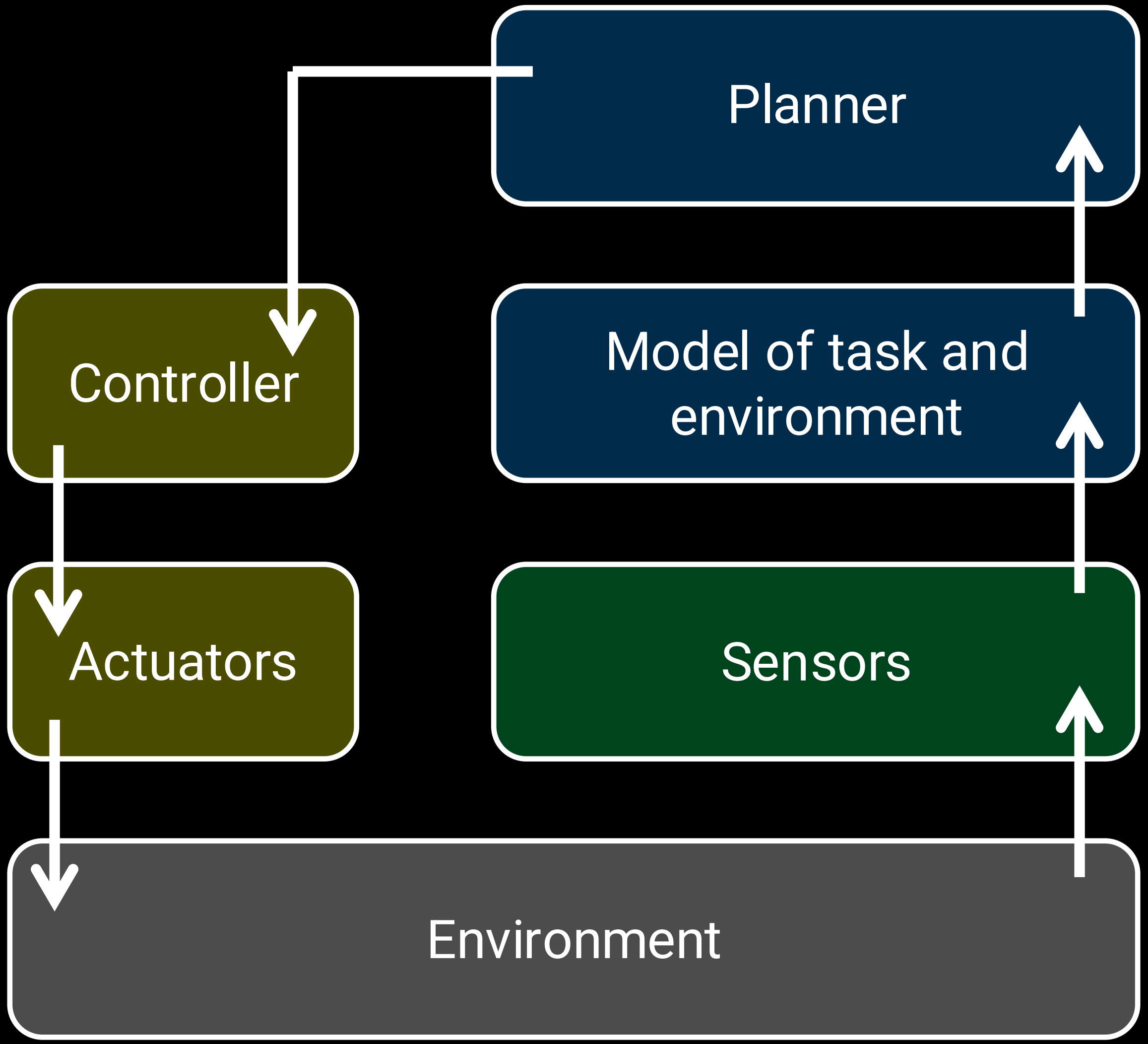
Deliberative – “Think (hard), then act”

Reactive – “Don’t think, just (re)act”

Hybrid – “Think and act concurrently (and separately)”

Deliberative (Sense Plan Act SPA) – "Think (hard), then act"

- Rely on sensor data and models.
- "Reasoning" is usually planning (searching possible state-action sequences and their outcomes).
- Top-down design.



Deliberative – “Think (hard), then act”

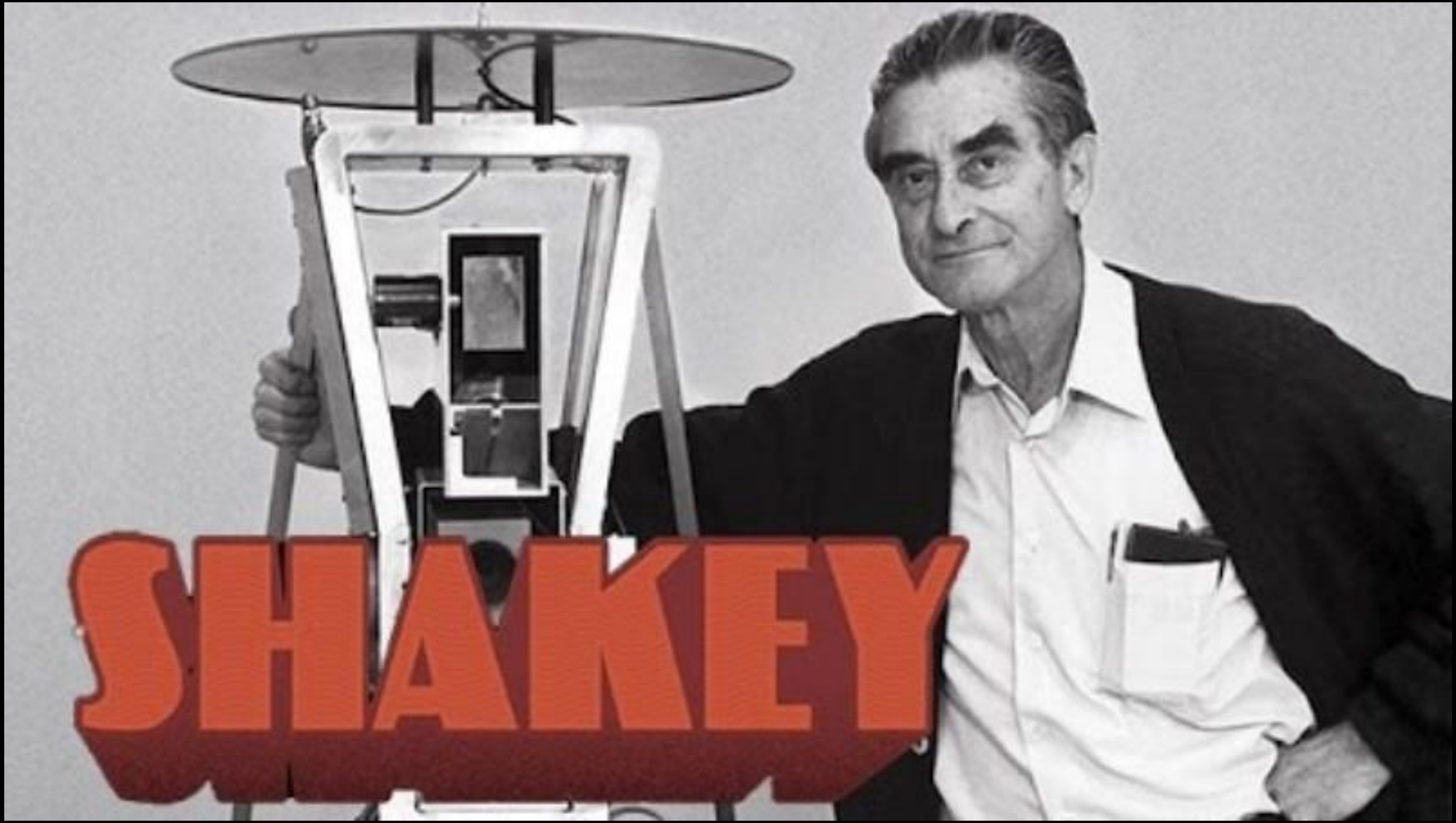
Advantages

- Strategically select the best course of action for a given situation
  - IF there is sufficient time to generate a plan AND the world model is accurate
- Capable of prediction and learning

Issues

- Frequent replanning
- Internal model accuracy
- Scalability problem

Deliberative – “Think (hard), then act”



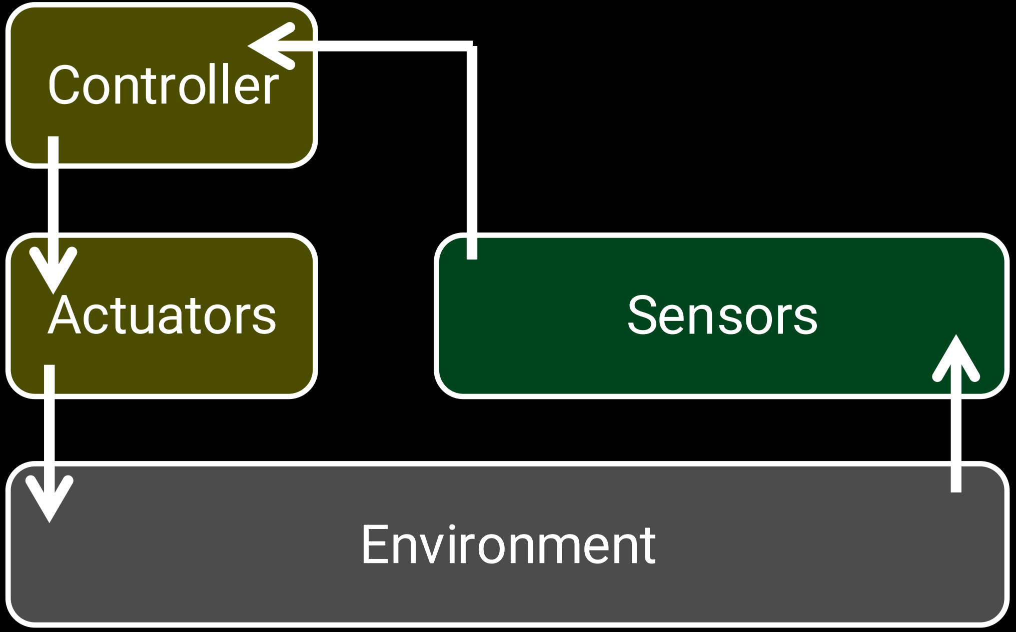
Reactive – “Don’t think, just (re)act”

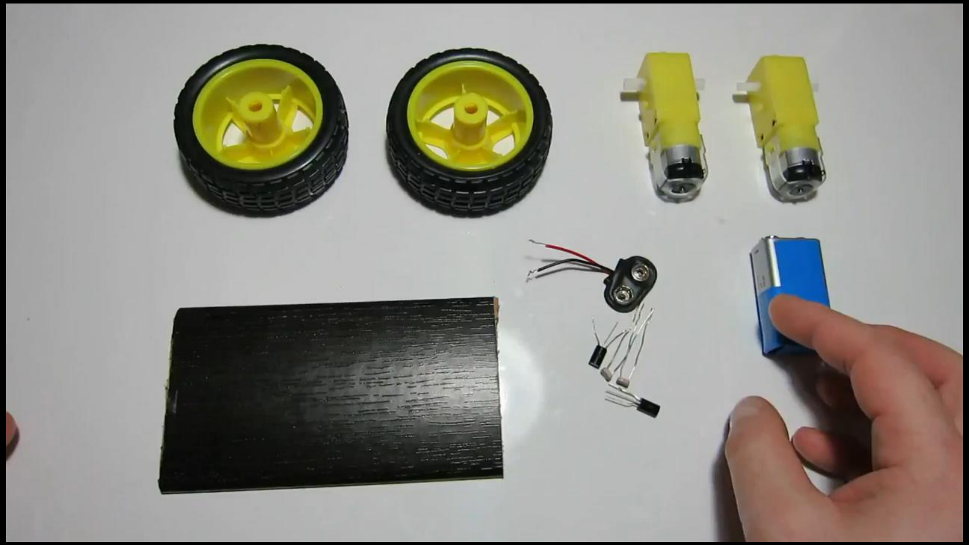
- Tight coupling between sensory inputs and actuator outputs

- Pre-programmed and concurrent condition-action rules with minimal amount of computation
  - For e.g., IF bumped THEN stop; IF stopped THEN back up

- None or minimal internal model of the world or memory so no planning possible

- Optimal performance in environments and tasks that are fully known in advance





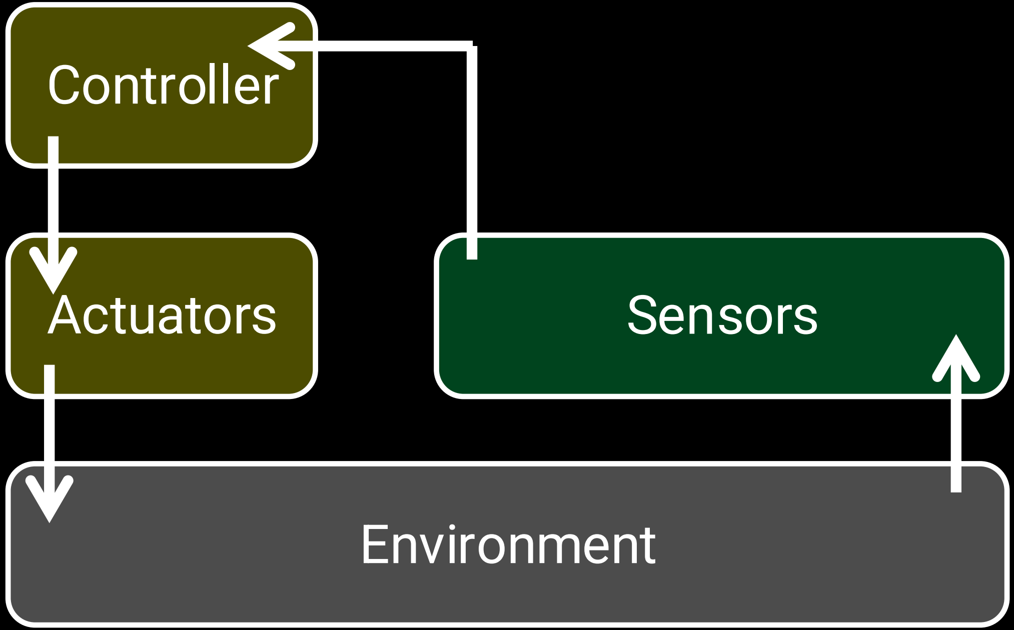
Reactive – “Don’t think, just (re)act”

- Tight coupling between sensory inputs and actuator outputs

- Pre-programmed and concurrent condition-action rules with minimal amount of computation
  - For e.g., IF bumped THEN stop; IF stopped THEN back up

- None or minimal internal model of the world or memory so no planning possible

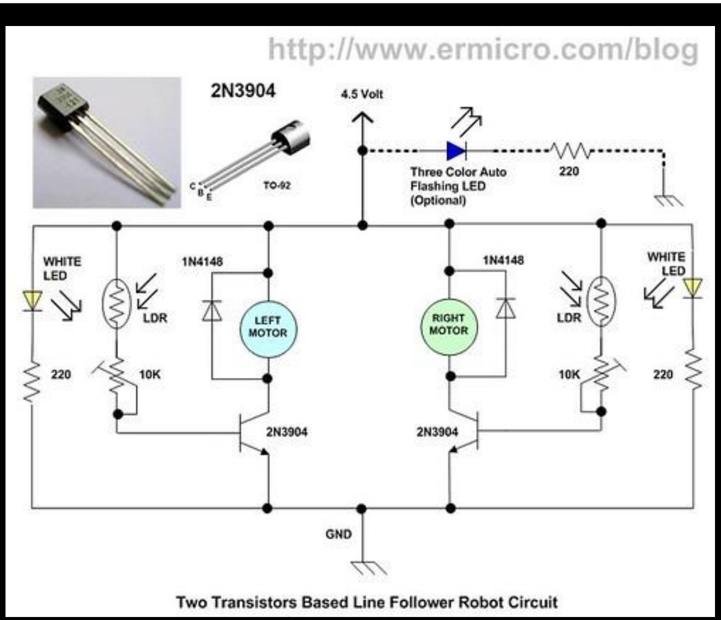
- Optimal performance in environments and tasks that are fully known in advance


Controller

Actuators

Sensors

Environment


http://www.ermicro.com/blog

Two Transistors Based Line Follower Robot Circuit

Reactive – “Don’t think, just (re)act”

Advantages

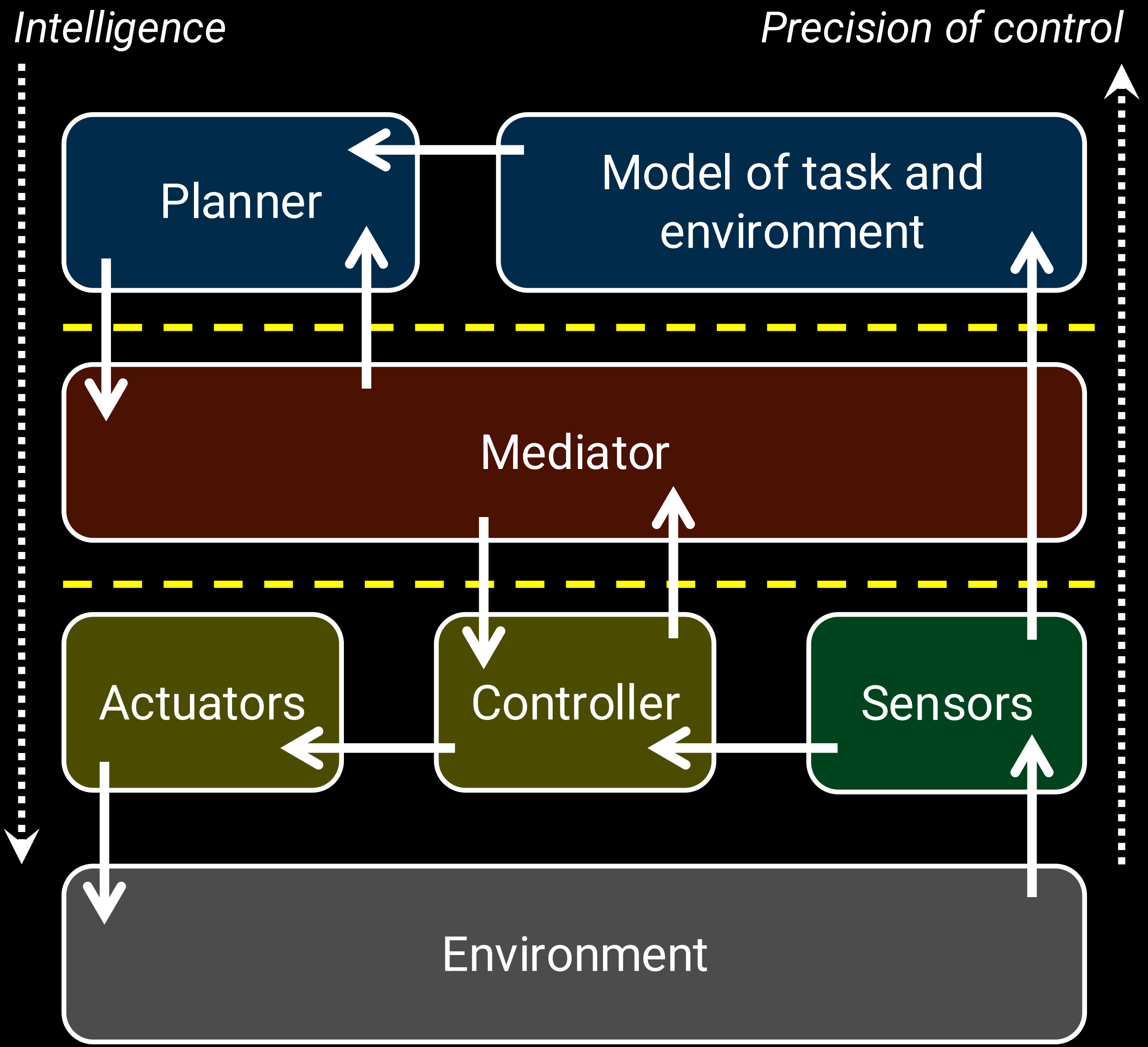
- Minimal amount of computation.
- Well-suited to dynamic environments

Issues

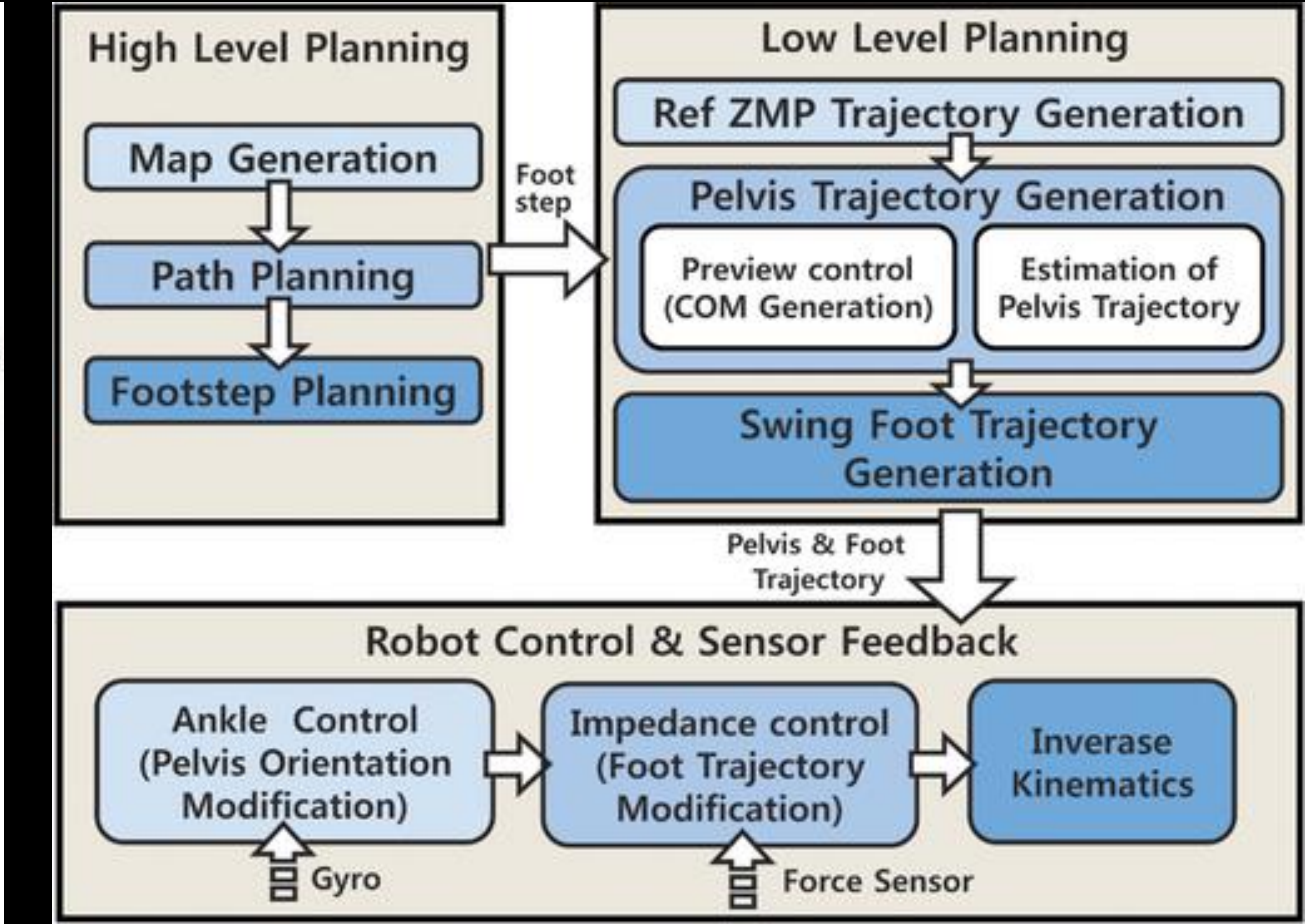
- Low complexity of reasoning
- No memory or information storage, no/minimal internal representation of the world (**Cannot predict and learn**)
- Unsuitable for tasks or environments that are not fully known in advance

Hybrid – “Think and act concurrently (and separately)”

- Compromise between deliberative and reactive approaches
  - Reactive part must override deliberative part when unexpected changes occur in the world
  - Deliberative part must supervise the reactive part to generate efficient and optimal solutions
- A mediator co-ordinates the interaction between deliberative and reactive parts
  - Resolves differences between representations and conflicts between their outputs



Hybrid(ish) example of locomotion control: Thormang the Clever


High Level Planning
Map Generation
Path Planning
Footstep Planning
Foot step
Low Level Planning
Ref ZMP Trajectory Generation
Pelvis Trajectory Generation
Preview control (COM Generation)
Estimation of Pelvis Trajectory
Swing Foot Trajectory Generation
Pelvis & Foot Trajectory
Robot Control & Sensor Feedback
Ankle Control (Pelvis Orientation Modification)
Impedance control (Foot Trajectory Modification)
Inverase Kinematics
Gyro
Force Sensor

Paradigms for Robot Structural Architectures

Deliberative – “Think (hard), then act”

Reactive – “Don’t think, just (re)act”

Hybrid – “Think and act concurrently (and separately)”

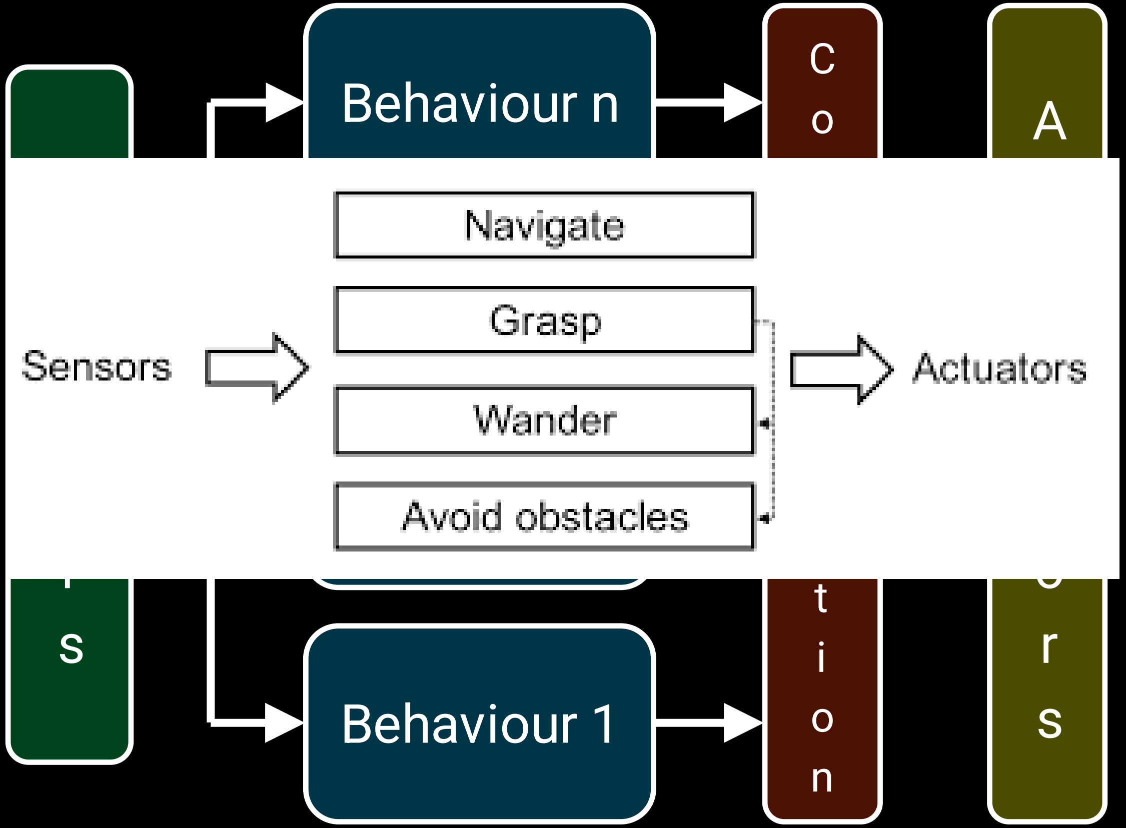
Behaviour-based – “Think the way you act”

Behaviour-Based – “Think the way you act”

A layered network of set of distributed, concurrent, interacting modules called behaviours, that collectively achieves desired system-level behaviour.

A behavior is a specific set of actions or responses by a robot to environmental stimuli.

- Layered Network
- Emergent Functionality
- Bottom-Up Design
- Individual behaviors and networks can store world representations

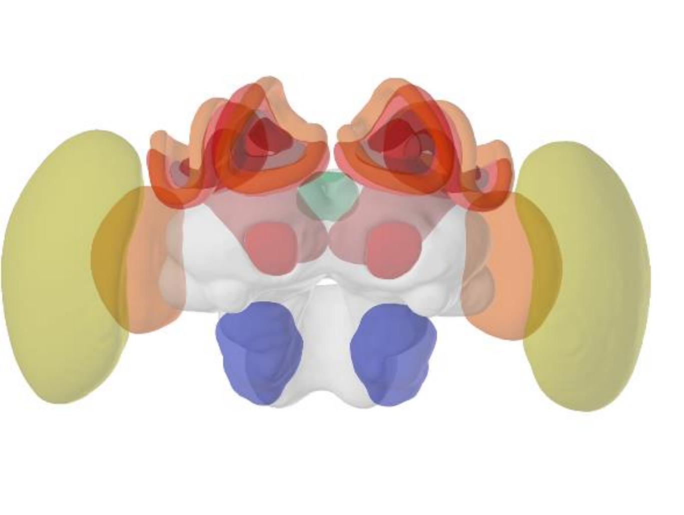


Behaviour-Based – “Think the way you act”

A layered network of set of distributed, concurrent, interacting modules called behaviours, that collectively achieves desired system-level behaviour.

A behavior is a specific set of actions or responses by a robot to environmental stimuli.

- Layered Network
- Emergent Functionality
- Bottom-Up Design
- Individual behaviors and networks can store world representations



Distinct brain regions specialize in different behaviors, such as: perception, olfaction, multimodal memory centre.

Hybrid – “Think and act concurrently (and separately)”

- Advantages
  - Fast, Dynamic Response: Quickly adapts to changes in the environment.
  - Near-Optimal Solutions: Maintains overall efficiency even if short-term actions diverge from the optimal path.

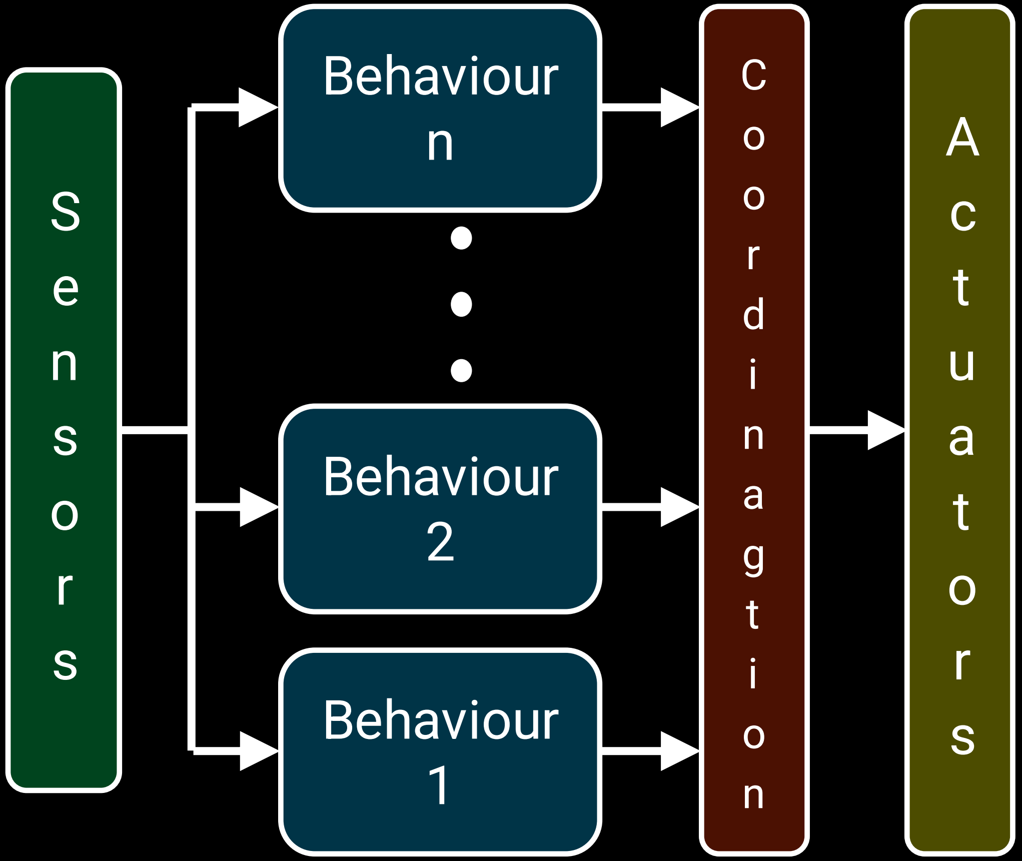
- Issues
  - Coordination Challenges: Difficult to synchronize both.
  - Time Scale Discrepancies
  - Different World Representations
  - Complex Mediator Design

Characteristics of behaviour-based systems as an AI

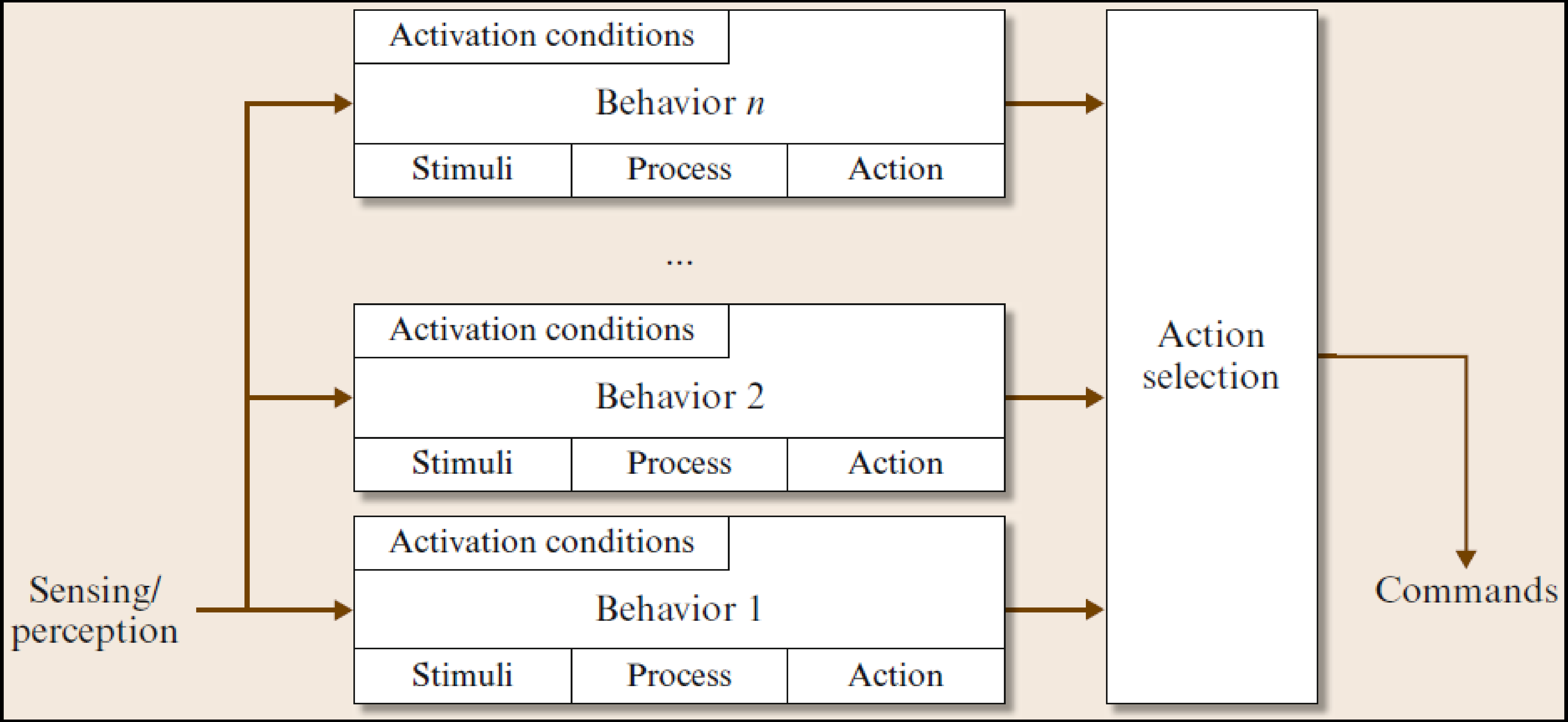
- Intelligence and emergence
- Situatedness (being embedded in an environment)
- Embodiment (directly couples sensed data to meaningful actions)

Behaviours

- **Behaviors as Control Laws:** Implemented in software or hardware.
- **Inputs and Outputs:** Behaviors interact with sensors, actuators, and other behaviors.
- **Shared Resources:** Multiple behaviors can use the same sensors and actuators.
- **Simplicity:** Behaviors are simple and task-specific.
- **Time-Extended Processes:** Behaviors handle ongoing processes, not just single actions.
- **Concurrent Execution:** Behaviors run in parallel for efficiency and dynamic interaction.

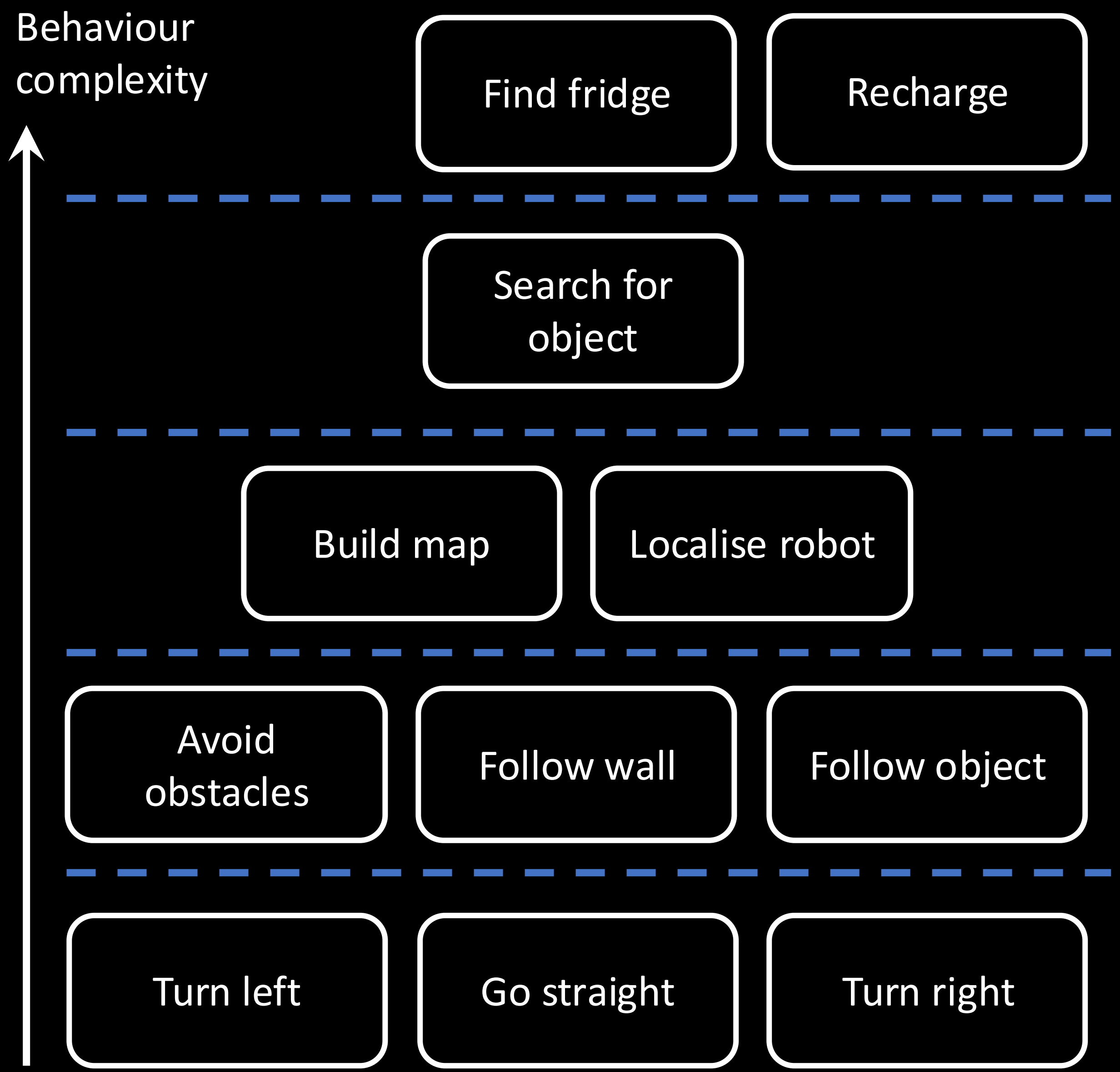


A general template for behaviour-based control



Bottom-up design and implementation methodology

- Design behaviour architecture with a variety of abstraction levels or layers, from simple to complex
- Add new behaviours incrementally in the robot, from simple to complex
- Test robot in real environment after adding each behaviour(!)
- Stop adding behaviours when their interaction results in the desired overall capabilities of the robot
- KISS – Keep It Simple Stupid



Behaviour Coordination

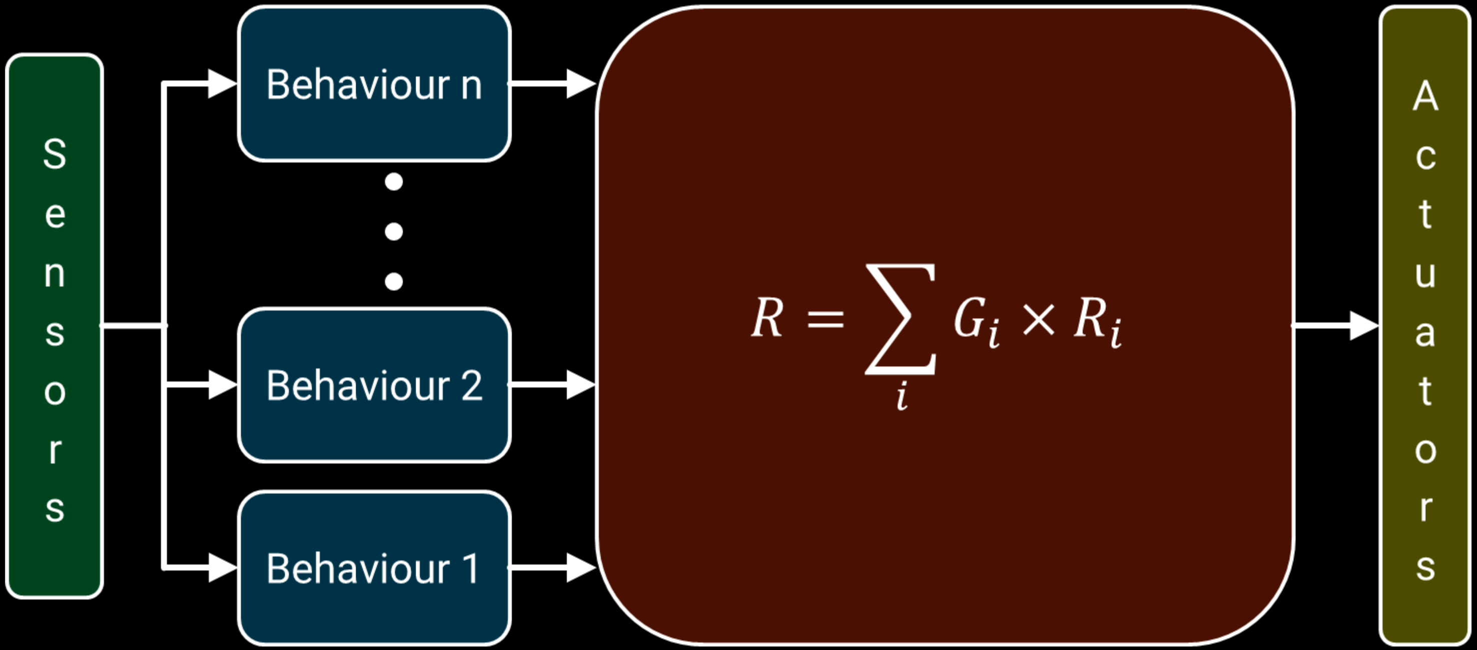
- Typically there are many behaviours but only few actuators, so behaviour coordination is needed
  - Competitive – select action from one of the behaviours and send it to actuators (winner-take-all)
  - Cooperative – combine actions from several behaviors to produce a new action to send to actuators

Behaviour coordination is typically a major design challenge (so test your robot thoroughly!)

Cooperative behaviour coordination example

- Behaviours get sensory inputs and generate a multi-dimensional vector R_i that defines how should the robot move
- Each behaviour's contribution to the global response is scaled by a gain factor G_i.
- Vector summation leads to cooperative coordination as all behaviours contribute to final response

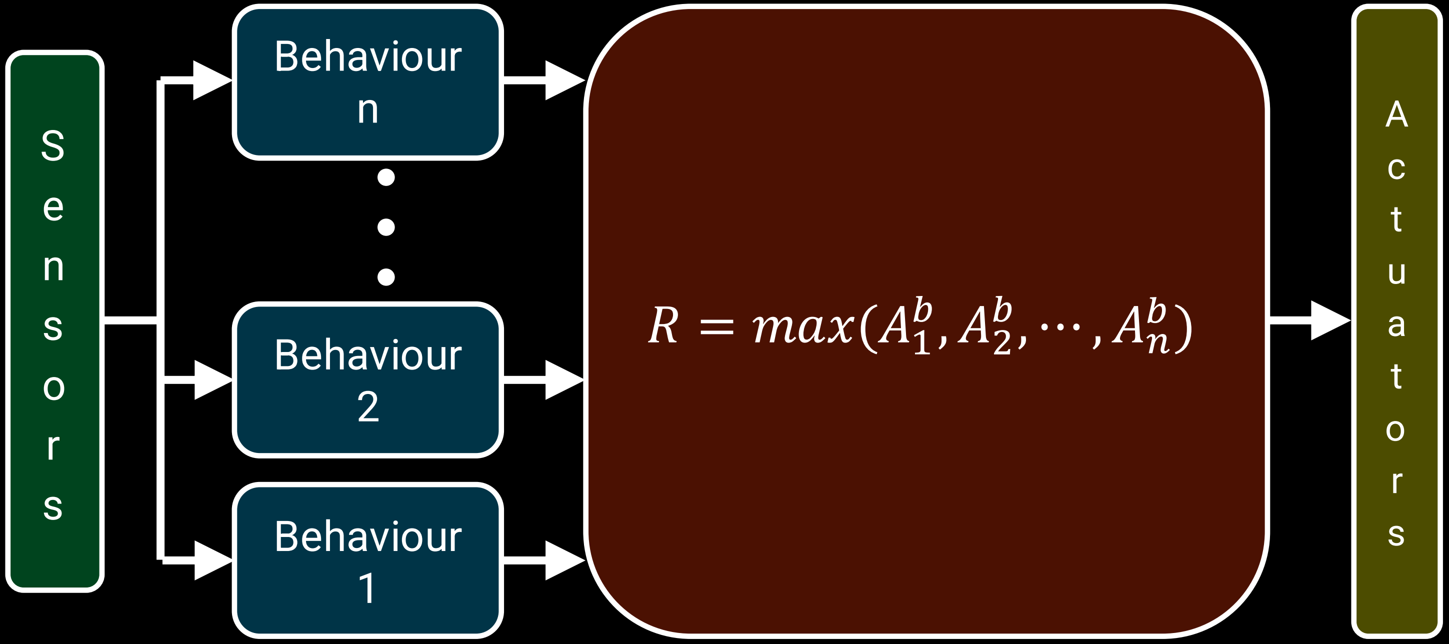
Ronald Arkin



Competitive behaviour coordination example

A dynamic priorities definition based on activations levels by sensor information, goals, and other behaviors.

- **Dynamic Priorities Definition**
- **Preconditions for Activation**
- **Control of Actuators by the highest activation level behavior**



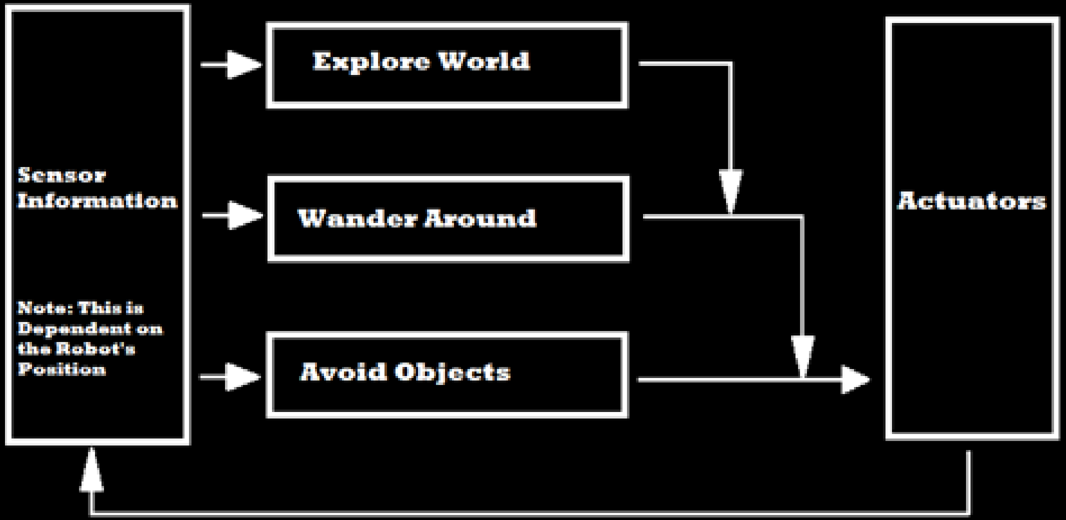
Patti Maes

Subsumption architecture

- Behaviors are organized into a hierarchy of layers
- Higher levels subsume lower levels, i.e. combine lower levels into a more comprehensive whole.
- Higher layers utilize the competencies of the lower layers
- All layers receive sensor information, work in parallel and generate outputs
  - Outputs can be commands to actuators, or signals that suppress or inhibit other layers

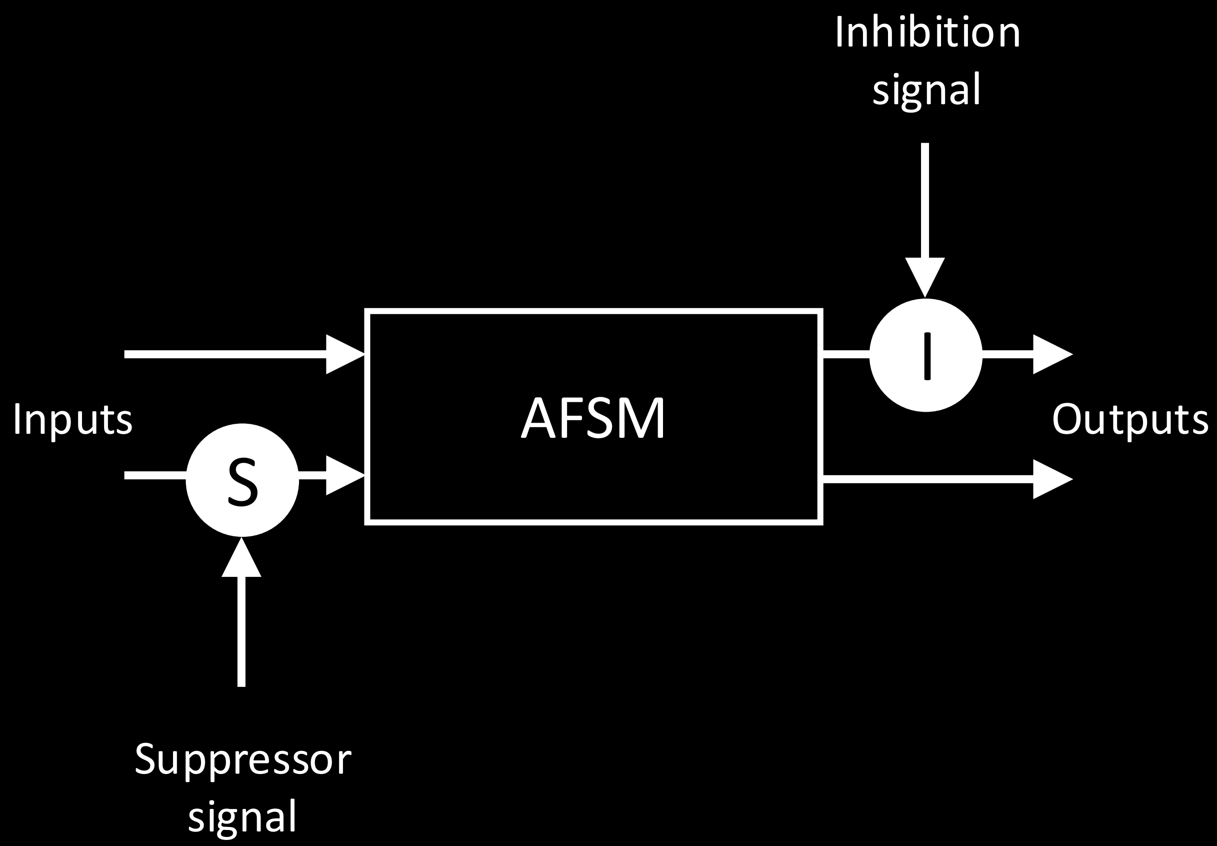
Rodney Brooks

Genghis hexapod

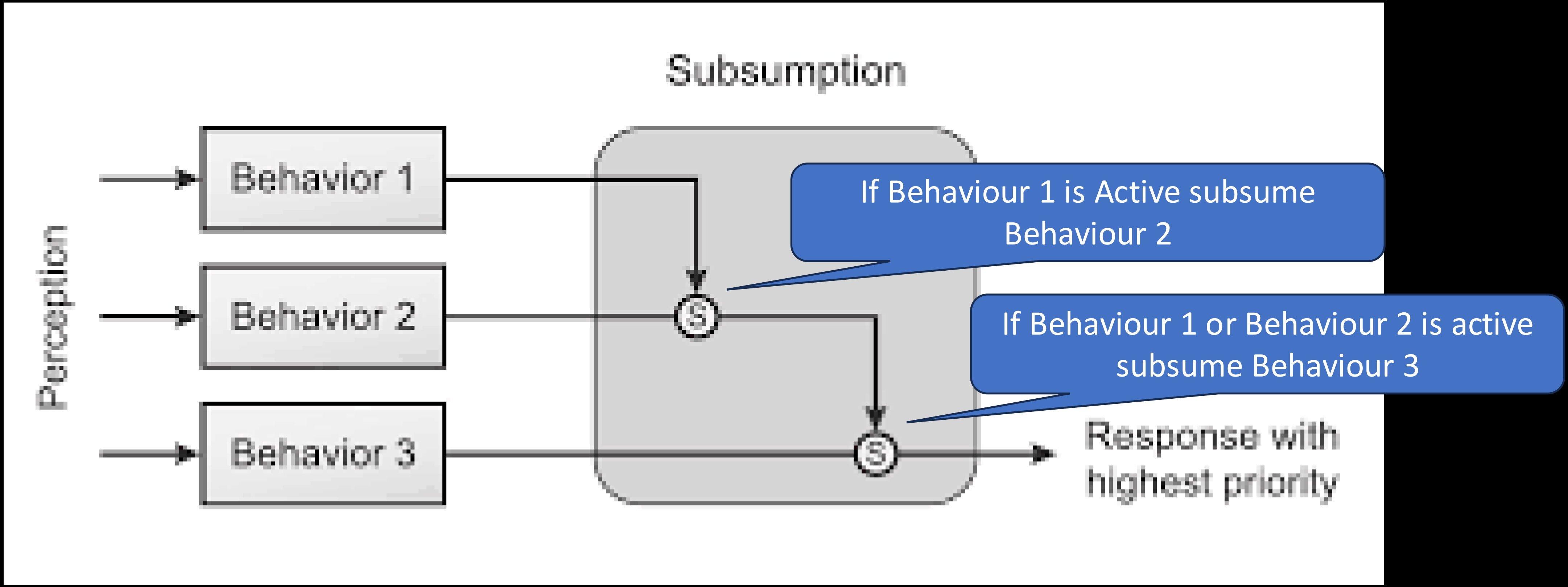


Layer implementation in the subsumption architecture

- Each layer is made up of a set of augmented finite-state machines (AFSM)
- All AFSMs continuously and asynchronously receive input from relevant sensors and send output to actuators (or other AFSMs)
- AFSMs communicate with each other via inhibition and suppression signals
    - Inhibition signals block outputs from reaching actuators or other AFSMs
    - Suppression signals block and replace inputs to layers or their AFSMs



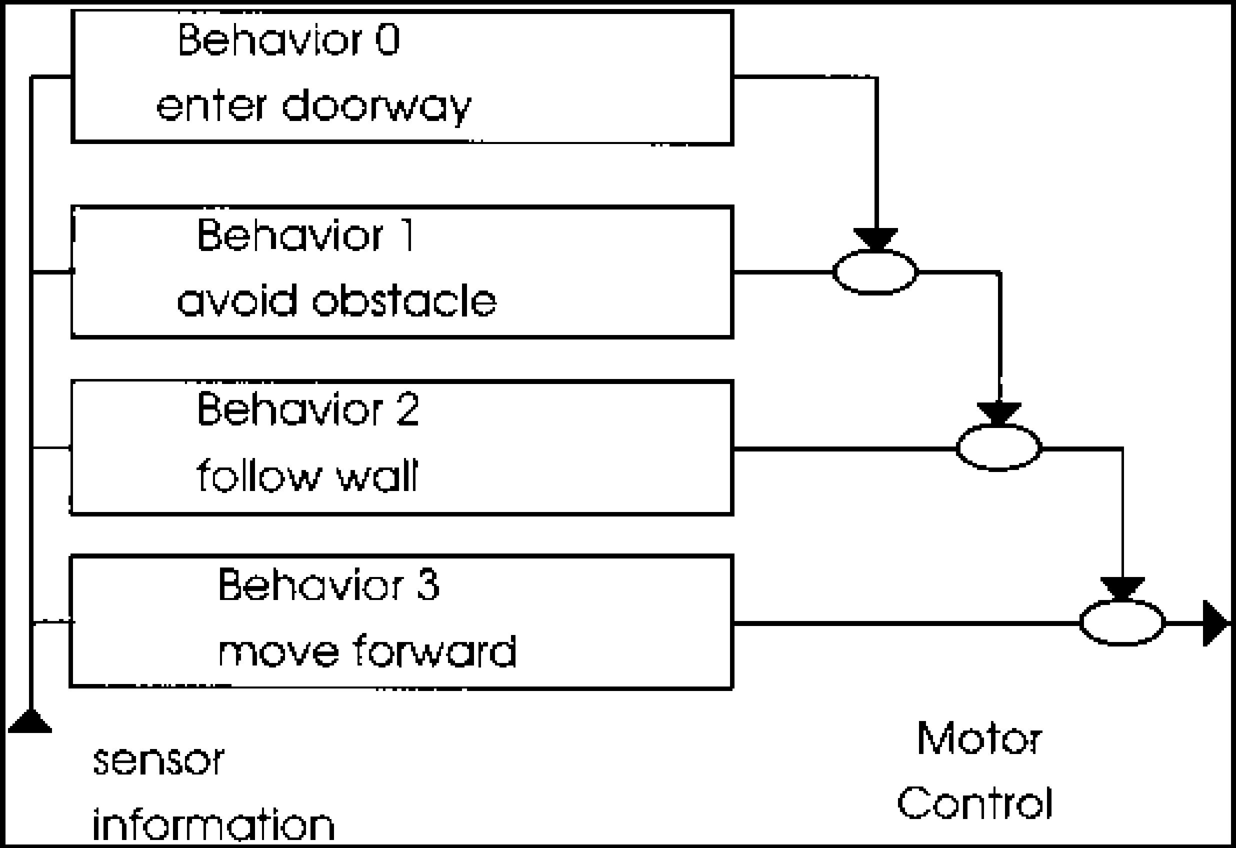
- Subsumption architecture
  - Built-in control hierarchy by assigning fixed priorities to behaviours



If Behaviour 1 is Active subsume Behaviour 2

If Behaviour 1 or Behaviour 2 is active subsume Behaviour 3

- Subsumption architecture
  - Built-in control hierarchy by assigning fixed priorities to behaviours



Advantages and disadvantages of the subsumption architecture

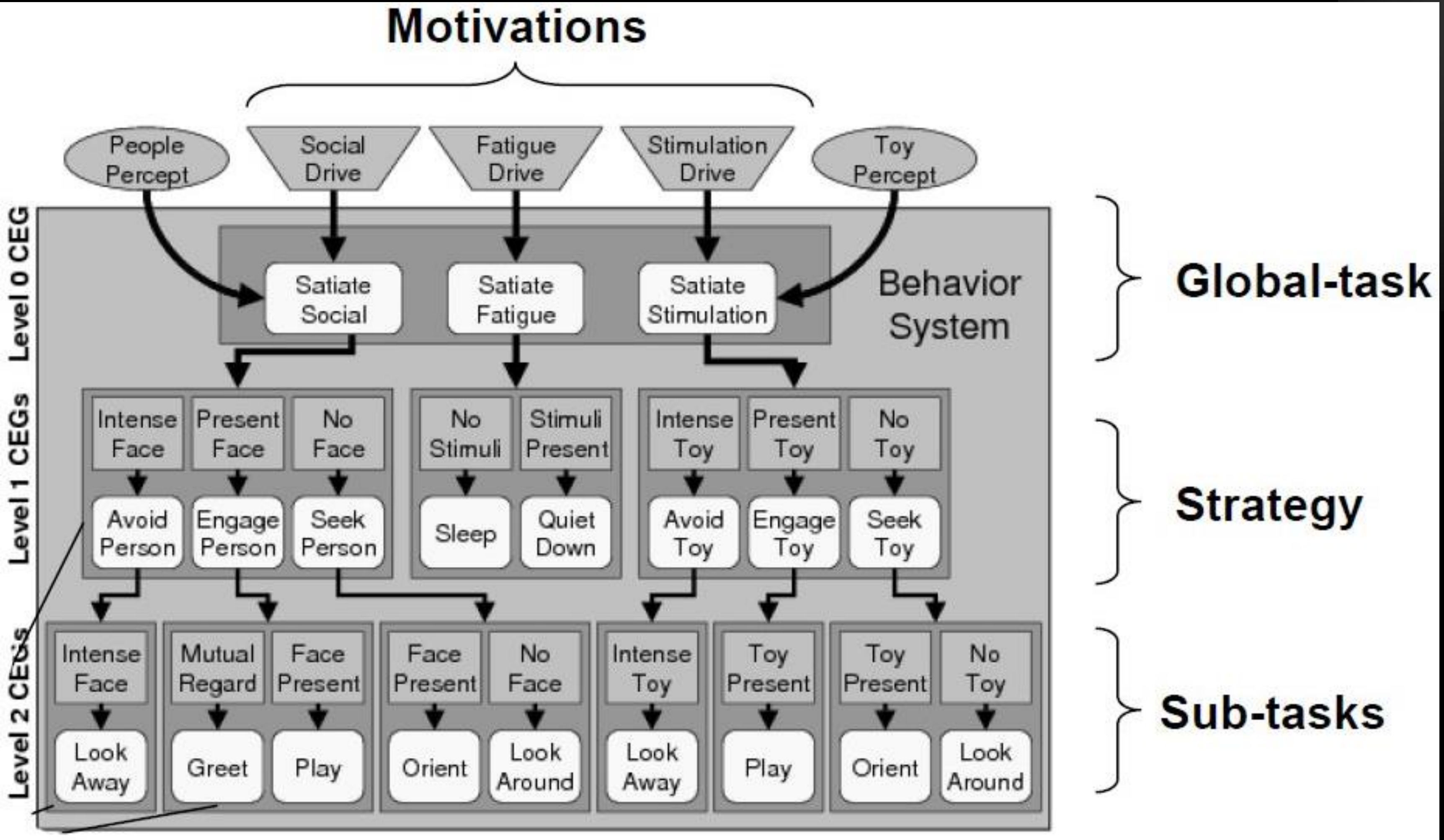
- Advantages
  - Iterative development and testing of real-time systems in their target domain
  - Connecting limited, task-specific perception directly to the actions that require it
  - Distributive and parallel control

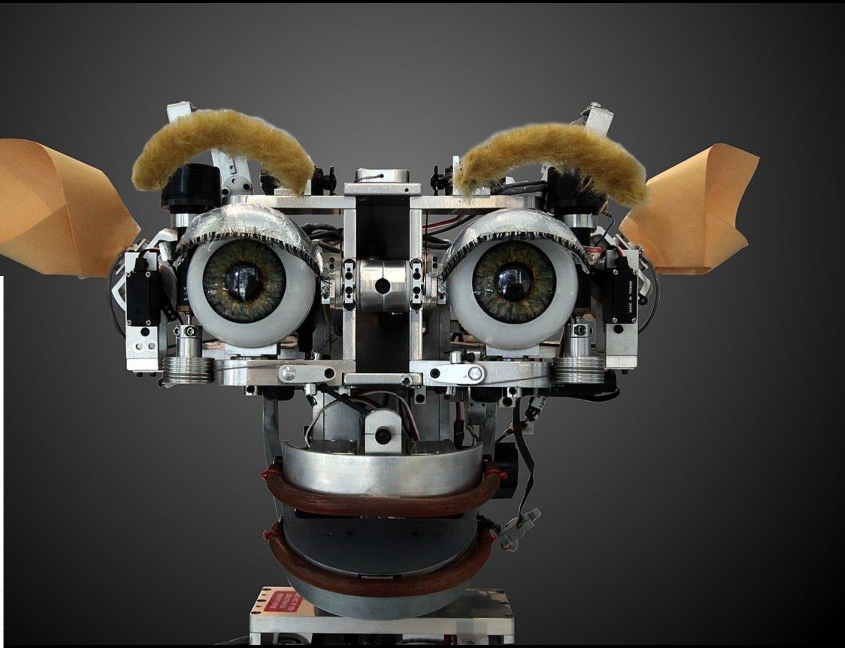
- Disadvantages
  - Difficulty of designing adaptable action selection through highly distributed system of inhibition and suppression
  - Lack of large memory and symbolic representation makes learning complex actions, in-depth mapping, and understanding language difficult
  - Forces the designer to prioritise behaviours, so behaviours with equal priority cannot be represented

FAST, CHEAP AND OUT OF CONTROL: A ROBOT INVASION OF THE SOLAR SYSTEM

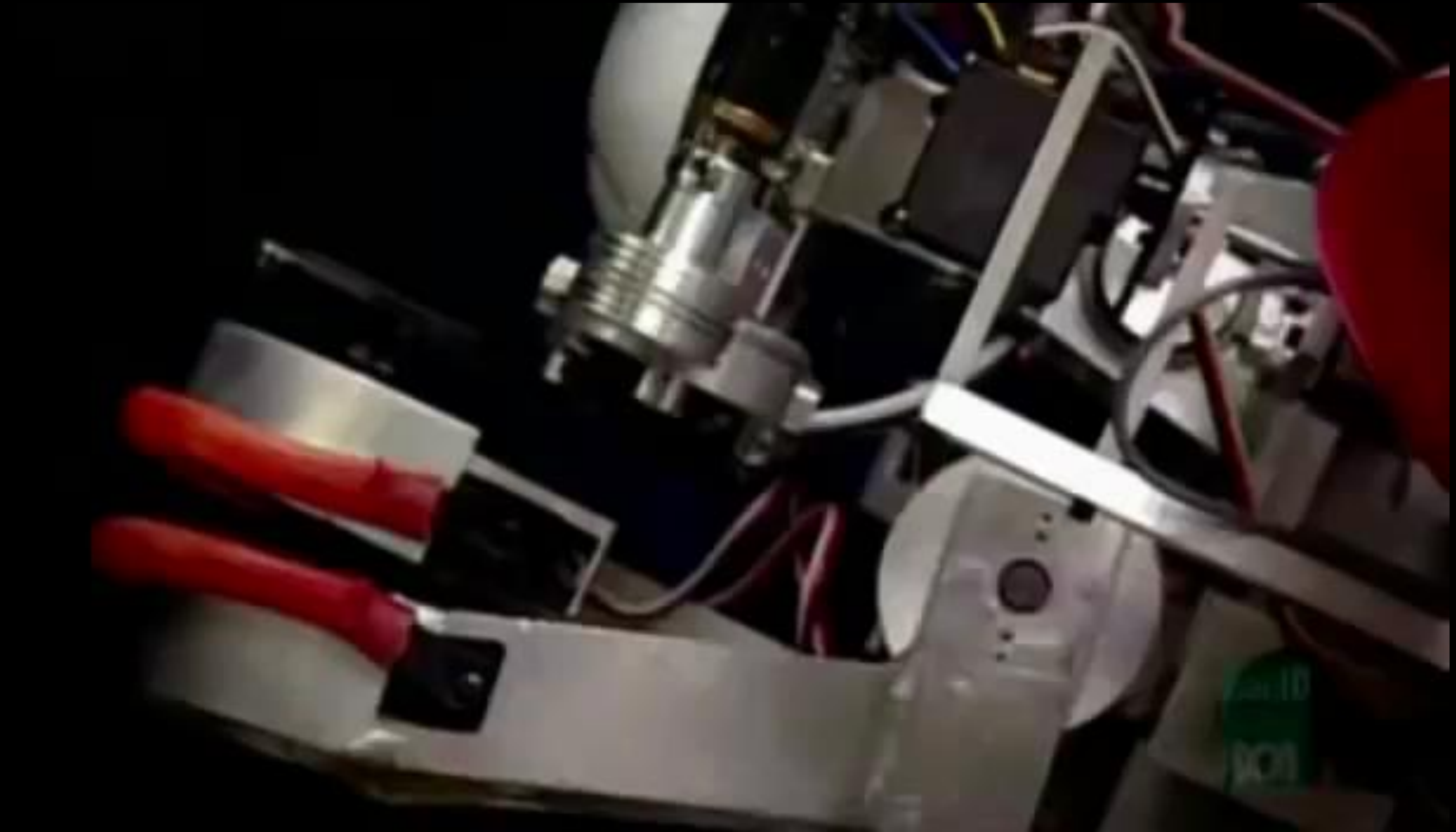
Rodney Brooks

Example: Kismet the robot





Example: Kismet the robot



How to create your own Behaviour Based Architecture ?

1. Define the Task/Environment

- What should the robot do? (*e.g., explore a room, deliver mail, patrol an area*).
- What constraints exist? (*obstacles, limited energy, goals*).

2. Identify Primitive Behaviors

- Break the task into **small, modular behaviors**, each mapping sensory input → motor output.
- Examples:
    - **Avoid obstacle**
    - **Follow wall**
    - **Wander**
    - **Seek goal**
    - **Recharge when battery low**

3. Specify Inputs and Outputs

- **Inputs** = what sensors trigger this behavior (e.g., sonar, IR, vision, battery level).
- **Outputs** = what motor commands it produces (e.g., turn, go forward, stop).

How to create your own Behaviour Based Architecture ?

4. Design Behavior Representation

•Each behavior is independent and reactive.

•Implementation options:

- Simple *if–then rules*.
- Finite State Machine.
- Etc.

5. Decide on Arbitration / Coordination

•Since multiple behaviors may activate simultaneously, decide **how to combine them**:

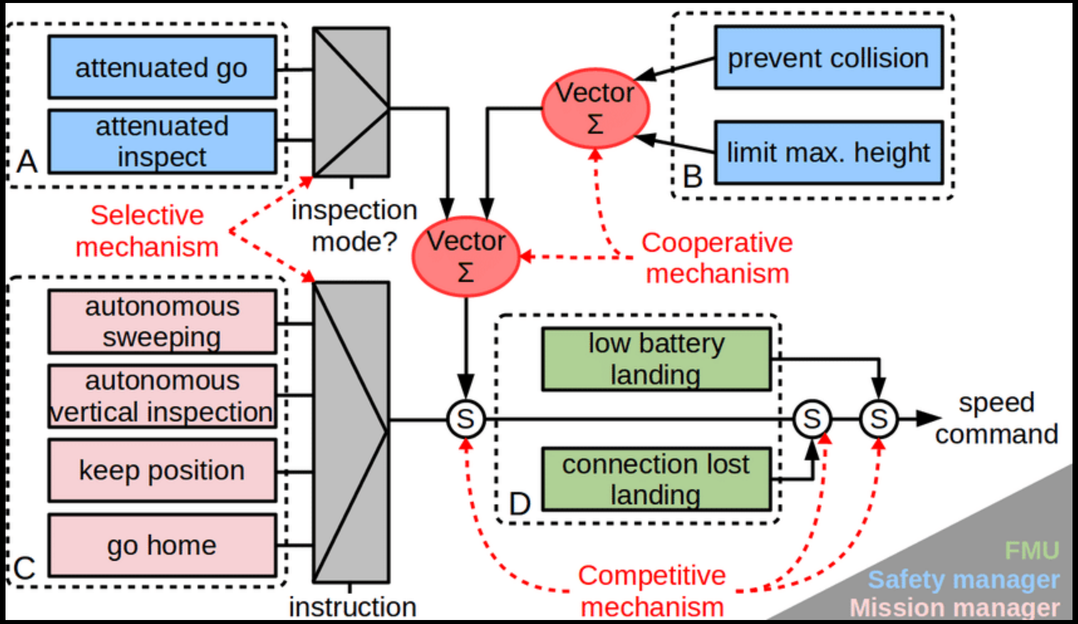
- **Competitive (Winner-takes-all)** – only one behavior executes (e.g., strongest activation).
  - **Subsumption (priority)** – higher priority behaviors override lower ones.
- **Collaborative** – behaviors output vectors, final action is a weighted combination.

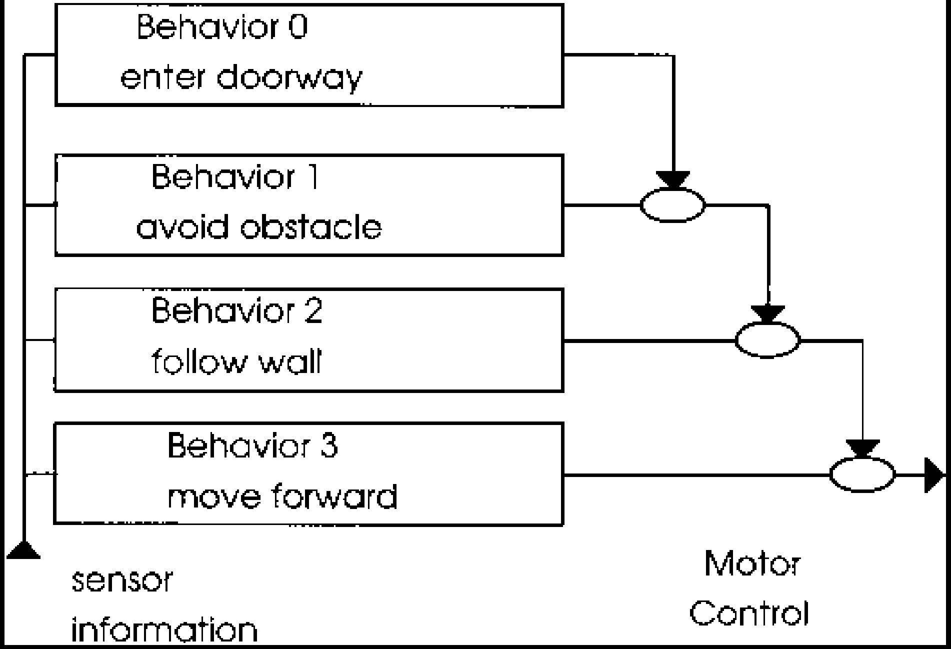
How to create your own Behaviour Based Architecture ?

5. Decide on Arbitration / Coordination

•Since multiple behaviors may activate simultaneously, decide **how to combine them**:

- **Competitive (Winner-takes-all)** – only one behavior executes (e.g., strongest activation).
  - **Subsumption (priority)** – higher priority behaviors override lower ones.
- **Collaborative** – behaviors output vectors, final action is a weighted combination.
- Combination.





4. Exercise

6. Implement Incrementally

- Start with the most critical behavior (usually **avoid obstacle**).
- Add others one by one, testing after each step.
- This incremental build shows students how **complexity emerges gradually**.

7. Test in Scenarios

- Place the robot (real or simulated) in different situations:
  - Obstacle ahead.
  - Goal visible but blocked.
  - Battery low.
- Observe: which behavior dominates, and how does the robot adapt?

8. Evaluate Emergent Behavior

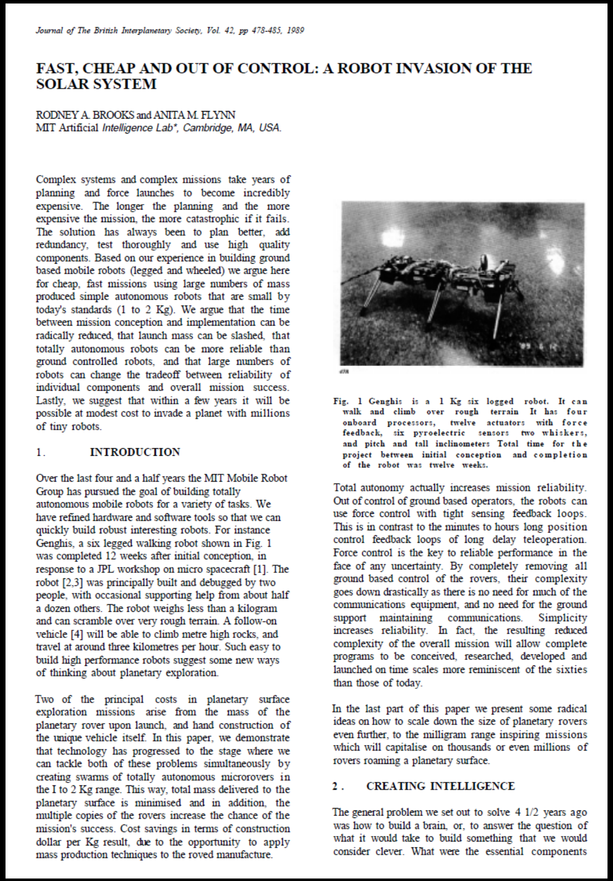
- Ask: does the robot achieve the task without centralized planning?
- Discuss trade-offs: robustness, adaptability, lack of global memory.

Further reading…

Springer Handbook of Robotics

Siciliano
Khatib
Editors

Springer



Journal of The British Interplanetary Society, Vol. 42, pp 478-485, 1989

FAST, CHEAP AND OUT OF CONTROL: A ROBOT INVASION OF THE SOLAR SYSTEM

RODNEY A. BROOKS and ANITA M. FLYNN
MIT Artificial *Intelligence Lab*, Cambridge, MA, USA.*

Complex systems and complex missions take years of planning and force launches to become incredibly expensive. The longer the planning and the more expensive the mission, the more catastrophic if it fails. The solution has always been to plan better, add redundancy, test thoroughly and use high quality components. Based on our experience in building ground based mobile robots (legged and wheeled) we argue here for cheap, fast missions using large numbers of mass produced simple autonomous robots that are small by today's standards (1 to 2 Kg). We argue that the time between mission conception and implementation can be radically reduced, that launch mass can be slashed, that totally autonomous robots can be more reliable than ground controlled robots, and that large numbers of robots can change the tradeoff between reliability of individual components and overall mission success. Lastly, we suggest that within a few years it will be possible at modest cost to invade a planet with millions of tiny robots.

1. INTRODUCTION

Over the last four and a half years the MIT Mobile Robot Group has pursued the goal of building totally autonomous mobile robots for a variety of tasks. We have refined hardware and software tools so that we can quickly build robust interesting robots. For instance Genghis, a six legged walking robot shown in Fig. 1 was completed 12 weeks after initial conception, in response to a JPL workshop on micro spacecraft [1]. The robot [2,3] was principally built and debugged by two people, with occasional supporting help from about half a dozen others. The robot weighs less than a kilogram and can scramble over very rough terrain. A follow-on vehicle [4] will be able to climb metre high rocks, and travel at around three kilometres per hour. Such easy to build high performance robots suggest some new ways of thinking about planetary exploration.

Two of the principal costs in planetary surface exploration missions arise from the mass of the planetary rover upon launch, and hand construction of the unique vehicle itself. In this paper, we demonstrate that technology has progressed to the stage where we can tackle both of these problems simultaneously by creating swarms of totally autonomous microrovers in the I to 2 Kg range. This way, total mass delivered to the planetary surface is minimised and in addition, the multiple copies of the rovers increase the chance of the mission's success. Cost savings in terms of construction dollar per Kg result, due to the opportunity to apply mass production techniques to the roved manufacture.

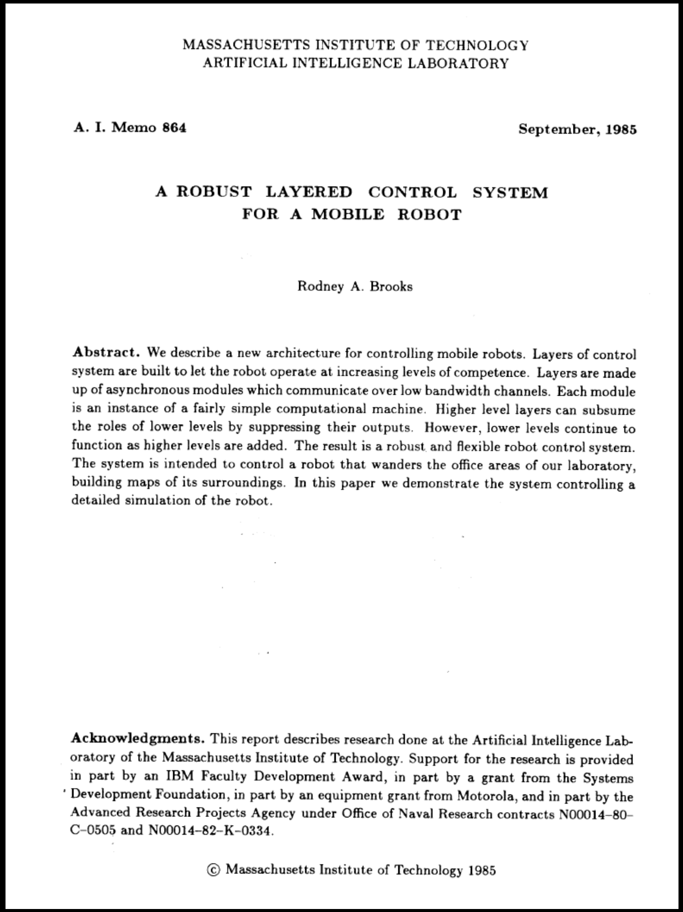
Fig. 1 Genghis is a 1 Kg six logged robot. It can walk and climb over rough terrain It has four onboard processors, twelve actuators with force feedback, six pyroelectric sensors two whiskers, and pitch and tall inclinometers Total time for the project between initial conception and completion of the robot was twelve weeks.

Total autonomy actually increases mission reliability. Out of control of ground based operators, the robots can use force control with tight sensing feedback loops. This is in contrast to the minutes to hours long position control feedback loops of long delay teleoperation. Force control is the key to reliable performance in the face of any uncertainty. By completely removing all ground based control of the rovers, their complexity goes down drastically as there is no need for much of the communications equipment, and no need for the ground support maintaining communications. Simplicity increases reliability. In fact, the resulting reduced complexity of the overall mission will allow complete programs to be conceived, researched, developed and launched on time scales more reminiscent of the sixties than those of today.

In the last part of this paper we present some radical ideas on how to scale down the size of planetary rovers even further, to the milligram range inspiring missions which will capitalise on thousands or even millions of rovers roaming a planetary surface.

2. CREATING INTELLIGENCE

The general problem we set out to solve 4 1/2 years ago was how to build a brain, or, to answer the question of what it would take to build something that we would consider clever. What were the essential components



MASSACHUSETTS INSTITUTE OF TECHNOLOGY
ARTIFICIAL INTELLIGENCE LABORATORY

A. I. Memo 864 September, 1985

A ROBUST LAYERED CONTROL SYSTEM FOR A MOBILE ROBOT

Rodney A. Brooks

Abstract. We describe a new architecture for controlling mobile robots. Layers of control system are built to let the robot operate at increasing levels of competence. Layers are made up of asynchronous modules which communicate over low bandwidth channels. Each module is an instance of a fairly simple computational machine. Higher level layers can subsume the roles of lower levels by suppressing their outputs. However, lower levels continue to function as higher levels are added. The result is a robust and flexible robot control system. The system is intended to control a robot that wanders the office areas of our laboratory, building maps of its surroundings. In this paper we demonstrate the system controlling a detailed simulation of the robot.

Acknowledgments. This report describes research done at the Artificial Intelligence Laboratory of the Massachusetts Institute of Technology. Support for the research is provided in part by an IBM Faculty Development Award, in part by a grant from the Systems Development Foundation, in part by an equipment grant from Motorola, and in part by the Advanced Research Projects Agency under Office of Naval Research contracts N00014-80-C-0505 and N00014-82-K-0334.