

Introduction to Robotics

9. Architectures.

1

What is a Robot Architecture?

A robot 'architecture' primarily refers to the software and hardware framework for controlling the robot.

Robotic systems are complex and tend to be difficult to develop.

They **integrate** multiple sensors with effectors, have **many degrees of freedom** and must reconcile hard real-time systems with systems which cannot meet real-time deadlines.

System developers have typically relied upon robotic architectures **to guide the construction of robotic devices** and for providing computational services (e.g., communications, processing, etc.) **to subsystems and components.**

2

Robot Architecture

To be successful a system designer has to decide how (in what order? with what priority?) does he put together multiple feedback controllers in a principled fashion and how to *scale up* control to more complex robots, which generally have to deal with many behaviors at once.

How would you put multiple feedback controllers together?

How would you decide which one to use when and for how long and in what priority relative to the others?

3

Robot Control Architecture.

There are many different ways in which a **robot control program** can be put together. **In order to program a robot in a structured and principled fashion, we use an appropriate robot control architecture.**

We will associate control architectures with software for control.

These architectures have tended thus far to be **task and domain specific** and have lacked suitability to a broad range of applications.

For example, an architecture well suited for direct teleoperation tends not to be easily controlled for supervisory control or for autonomous use.

4

Robot Control Architecture

A **control architecture** provides a set of principles for **organizing a control system**.

It provides structure and constraints which aid the designer in producing a well-behaved controller.

The term *architecture* is used here in the same sense as in *computer architecture*, where it means the set of principles for **designing computers out of a collection of well-understood building blocks**.

Similarly, in robot architectures, we will soon see that a set of building blocks or tools is at our disposal, and we can use them appropriately to program a robot in a reliable fashion.

5

Summary.

Definition.

Robotic Architecture is the discipline devoted to the design of highly specific and individual robots from a collection of common software building blocks.

6

Computational Perspective: Turing Universality

Since we are talking about software, this means that every architecture is implemented in some programming language.

Turing Universality. Any programming language that contains: sequencing, conditional branching, and iteration can compute the entire class of computable functions.

Programming languages are implementation **tools and not architectures!**

Generally, they are Turing-universal: **(in theory)**, *any programming language can be used to implement any architecture.*

In practice, some programming languages are more convenient than others for specific tasks:

A programming language can be designed so as to facilitate certain architectural practices (e.g., reactivity, tight feedback, symbolic representations, numerical computation, etc.)

7

Our interest in robot architectures.

We will discuss

the history of what has been used in the past, and

what is state of the art, and

what are the available options.

There are infinitely many possible robot programs, **but there are in fact a finite, and rather small, number of truly distinct robot architectures.**

8

Robot Architecture Major Classes/Categories

Intuitively, this means that there are infinitely many ways to structure a robot program, but they all fall into one of **major classes/categories of control**:

- 1) **deliberative** look-ahead: think/plan, then act
- 2) **reactive** no look-ahead: react
- 3) **behavior-based** distribute thinking over acting
- 4) **hybrid** combine 1+2, think slowly, react quickly

Pure deliberative control is practically no longer in use.

9

Misconceptions.

1. Programming languages are implementation tools and not architectures.
2. The issue of fundamental power or expressiveness of a robot control architecture: claims have been made about one control architecture being able to *compute* fundamentally more than another.

This **cannot be true** if we understand that all are grounded in Turing-complete programming languages. Since any language can compute anything that another language can, the architecture on top of it cannot further constrain it.

However, the above **is not to say that all architectures are the same**. On the contrary, **architectures impose strong constraints on how robot programs are structured, and the resulting control software ends up looking very different.**

10

The Choice of the Control Architecture

In many cases, it is impossible to tell, just by observing a robot's behavior, what control architecture it is using. Only for simple robots, it is often the case.

However, **when it comes to more complex robots**, i.e., robots that have to deal with complex environments and complex tasks, the control architecture becomes very important.

The different **properties of an environment** that will impact the robot's controller (and therefore the choice of control **architecture**):

noisy,

speed/response time of sensors and effectors total/partial hidden state/
observable

discrete v. continuous state ; static v. dynamic ...

Similarly, the properties of the robot's task impact the choice of the control architecture. **The task requirements can constrain the architecture choice.**

11

Parallel Processing Paradigm.

As robot control is engaged to deal **with more complex problems, centralized supervisory architectures encounter barriers to real time performance** caused by computational complexity coupled with **insufficient computing power and sensor resources.**

Despite startling advances in hardware and software technology and similarly surprising cost reductions, these fundamental **barriers remain unchanged.**

The parallel-processing paradigm may be the only technology to challenge this fact.

12

Sense - Plan - Act

Centralized supervisory robot control architectures are usually based on variants of the sense-plan-act (SPA) cycle.

In the mid-1980's it was the dominant view in the AI community:

- A control system for an autonomous mobile robot should be decomposed into three functional elements:

a sensing system,
a planning system, and
an execution system.

- The job of the sensing system is to translate raw sensor input (usually sonar or vision data) into a world model.

- The job of the planner is to take the world model and a goal and generate a plan to achieve the goal.

- The job of the execution system is to take the plan and generate the actions it prescribes.

13

Sense - Plan - Act (SPA).

The sense-plan-act (SPA) approach has two significant architectural features.

First, the flow of control among these components is unidirectional and linear. Information flows from sensors to world model to plan to effectors, never in the reverse direction.

Second, the execution of a SPA plan is analogous to the execution of a computer program.

The information content is in the composition structure, not the primitives.

The intelligence of the system (such as it is) lives in the planner or the programmer, not the execution mechanism.

SPA had numerous shortcomings.

**Planning and world modeling turned out to be Very Hard Problems, and
Open-loop plan execution was clearly inadequate in the face of
environmental uncertainty and unpredictability.**

14

Deliberate Reasoning

Based on **SPA model**.

Planning requires a relatively complete knowledge about the world, as well as some predictions about the outcome of its action.

In a dynamic world, objects may move, so it is dangerous to rely on past information, that may no longer be valid.

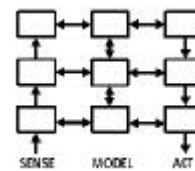
Representational World Models are constructed from both prior knowledge about the environment and incoming sensor data in support of deliberation.

Characteristics:

Hierarchical in structure, similar to the organization of commercial businesses or military command.

Control occurs in a predetermined manner, flowing through the hierarchy **up and down**.

Planning scope changes descending the hierarchy as time requirements are shorter and spatial considerations more local.



15

Behavior Based Systems.

One recent trend in robotic architectures has been a focus on behavior-based systems.

Behavior based refers to the fact that these systems exhibit various behaviors, some of which are emergent.

These systems are characterized by tight coupling between sensors and actuators, minimal computation, and a task-achieving "behavior" problem decomposition.

Behavior representation: how is action represented?

Granularity of behavior: what time scale is used for action?

Behavior interaction and coordination: how are actions/behaviors chosen?

Basis for behavior specification: is a biological model used?

Programming methods: is software reusable, supported, user-friendly?

16

Asynchronous and Synchronous processes

The other leading architectural trend is typified by a mixture of asynchronous and synchronous control and data flow.

Asynchronous processes are characterized as loosely coupled and event-driven without strict execution deadlines.

Synchronous processes, in contrast, are tightly coupled, utilize a common clock and demand hard real-time execution.

17

Some Criteria for Selecting a Control Architecture

support for parallelism: the ability of the architecture to execute parallel processes/behaviors at the same time.

hardware targetability:

how well the architecture can be mapped onto real-robot sensors and effectors

how well the computation can be mapped onto real processing elements (microprocessors)

run-time flexibility: does the architecture allow run-time adjustment and reconfiguration? It is important for adaptation/learning.

modularity: how does the architecture address encapsulation of control, how does it treat abstraction?

Does it allow many levels, going from feedback loops to primitives to agents?

Does it allow re-use of software?

18

Some Criteria for Selecting a Control Architecture

niche targetability: how well the architecture allows the robot to deal with its environment

robustness: how well does the architecture perform if individual components fail? How well does it enable and facilitate writing controllers capable of *fault tolerance*?

ease of use: how easy to use and accessible is the architecture? Are there programming tools and expertise?

performance: how well does the robot perform using the architecture? Does it act in real-time? Does it get the job done? Is it failure-prone?

The above issues allow us to compare and evaluate different architectures relative to specific robotic designs, tasks, and environments. **But not all tasks, environments, and designs are comparable.**¹⁹

Time Scale.

Time-scale is an important way of distinguishing control architectures.

Reactive systems respond to the real-time requirements of the environment,

while **deliberative system** look ahead (plan) and thus work on a longer time-scale.

Hybrid systems must combine the two time-scales in an effective way, usually requiring a middle layer; consequently they are often called *three-layer architectures*.

Finally, **behavior-based systems** attempt to bring the different time-scales closer together by distributing slower computation over concurrent behavior modules.²⁰

Representation

Another key distinguishing feature between architectures is **representation** of the world/environment, also called **world modeling**.

Some tasks and architectures involve storing information about the environment **internally**, in the form of an **internal representation** of the environment.

For example, while exploring a maze, a robot may want to remember a sequence of moves it has made (e.g., "left, left, right, straight, right, left"), so it can back-track and find its way.

Thus, the robot is constructing a representation of its path through the maze.

The robot can also build a *map* of the maze, by drawing it using exact lengths of corridors and distances between walls, etc. .

This is also a representation of its environment, a model of the world.

If two robots are working together, and one is much slower than the other, if the fast robot remembers/learns that the other is always slower, that is also a type of a model of the world, in this case, a model of the other robot.

21

Different World Models.

There are numerous aspects of the world that a robot can represent/model, and numerous ways in which it can do it, including:

- * **spatial** metric or topological: maps, navigable spaces, structures
- * **objects** instances of detectable things in the world
- * **actions** outcomes of specific actions on the self and environment
- * **self/ego** stored proprioception: sensing internal state, self- limitations, etc.
- * **intentional** goals, intended actions, plans
- * **symbolic** abstract encoding of state/information

22

Amount and Type of Representation

The amount and type of representation or modeling used by a robot is **critically related to the type of control architecture** it is using.

Some models are very elaborate; they take a long time to construct and are therefore kept around possibly throughout the lifetime of the robot's task (for example detailed metric maps).

Others may be relatively quickly constructed and transient, used quickly and discarded or updated (for example the next few steps in a short plan, the immediate goal, etc.)

How long it takes to construct/build a model is an important aspect of the robot's controller.

23

Amount and Type of Representation

How long it takes to use it is equally important.

Consider maps again:

- * it takes a long time to construct an accurate and detailed metric map, because it requires exploring and measuring the environment.
- * furthermore, it takes time to use such a map as well (even if it took no time to construct it, but it was given to the robot by the designer);
- * one must find the free/navigable spaces in the map, and then
- * search through those to find the best path to the goal.

Similarly, **any internal model can require time to construct and be used, and these timing requirements directly affect the time-scale of the controller.**

24

Control Architectures and Internal States/Representations.

A control architecture can make it easy or difficult to store internal models (just as a programming language can make it more or less convenient to build and store structures) **and manipulate them, i.e., compute with them.**

How **internal state**, i.e., information a robot system keeps around, relates to **representation**.

- In principle, any internal state is a form of representation.
- What matters is the form and function of that representation.
- The reason two different terms are employed is as follows: **state** refers to the "**status**" of the system itself, whereas "**representation**" refers to **arbitrary information** that may be contained in the system.

25

A Foraging Example

Foraging is a well studied problem in robotic navigation.

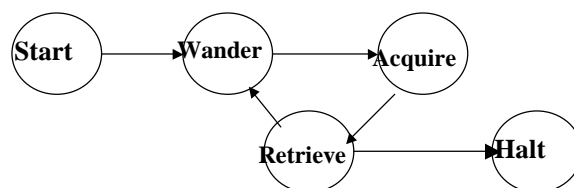
Several high level behavioral requirements to accomplish the task include:

Wander: move through the world in search of an attractor.

Acquire: move toward the attractor when detected.

Retrieve: return the attractor to the home base once acquired.

Finite State Acceptor (**FSA**) diagram for foraging



26