Robot Control Architectures

DT211/1 Applied Computing
DIT School of Computing
In order to implement a solution to the team challenge you will need to consider how to divide up and structure the program that controls your robot.

In preparation for this, today’s lecture will review how people have approached this problem already.

In other words we will look at the history of robot architectures.
What is a robot architecture?

- The term **robot architecture** is used to refer to how a system is divided into subsystems and how those subsystems interact.
- Robot architectures are distinguished from other software architectures because of the special needs of robot systems.
What are the special needs of robot systems?
Robot systems operate in complex dynamic real-time environments. These systems have to:

- to control diverse sensors and actuators in real time,
- in the face of significant uncertainty and noise
- while monitoring for, and reacting to, unexpected situations
- and do all this concurrently and asynchronously

Moreover, robot systems need to respond at varying temporal scopes from millisecond feedback control to minutes, or hours, for complex tasks.
Robot architectures and programming began in the late 1960s with the **Shakey** robot at Stanford University.

**Figure**: The Shakey robot. Shakey had a camera, a range finder, and bump sensors, and was connected to DEC PDP-10 and PDP-15 computers via radio and video links.
Shakey’s architecture was decomposed into three functional elements: **sensing, planning, and executing**.

1. **The sensing system** translated the camera image into an internal world model.

2. **The planner** took the internal world model and a goal and generated a plan (i.e., a series of actions) that would achieve the goal.

3. **The executor** took the plan and sent the actions to the robot.
This approach has been called the sense-plan-act (SPA) paradigm.

**Figure:** The sense-plan-act paradigm.

The components of the robot in this case are said to be horizontally organized. Information from the world in the form of sensor data has to filter through several intermediate stages of interpretation before finally becoming available for a response.
The main architectural features of the SPA paradigm are that sensing flowed into a world model, which was then used by the planner, and that plan was executed without directly using the sensors that created the model.
The emphasis in these early systems was in constructing a detailed world model and then carefully planning out what steps to take next.

The problem was that, while the robot was constructing its model and deliberating about what to do next, the world was likely to change.

So these robots exhibited the odd behavior that they would look (acquire data, often in the form of one or more camera images), process and plan, and then (often after a considerable delay) they would lurch into action for a couple of steps before beginning the cycle all over again (look and lurch behaviour).
Problems with SPA

1. Planning in any real world domain took a long time, and the robot would be blocked, waiting for planning to complete.
2. Execution of a plan without involving sensing was dangerous in a dynamic world.

Often an SPA system would produce a plan, but before this plan could be executed in full, it would become invalidated by changes in the real world.
In 1986 Rodney A. Brooks published an article which described a type of reactive architecture called the **subsumption architecture**.

### Implementation

- A subsumption architecture is built from **layers** of interacting **finite-state machines** - each connecting sensors to actuators directly.
- These finite-state machines were called **behaviors** (leading some to call the subsumption architecture behavior-based or behavioral robotics).
- Since multiple behaviors could be active at any one time, Subsumption had an **arbitration mechanism** that enabled higher-level behaviors to override signals from lower-level behaviors.

The subsumption architecture became the dominant approach within the reactive robot architectures.
**Figure:** The subsumption architecture.
The Subsumption Architecture (Brooks, 1986) was characterised by:

1. a lack of representation of the outside world,
2. the analysis of the architecture on a task rather than a functional basis,
3. the subsuming of behaviours by higher level behaviours,
4. a tight coupling between sensors and actuators.
For example, the robot might have a behavior that simply drives the robot in random directions. This behavior is always active and the robot is always driving somewhere.

A second, higher-level behavior could take sensor input, detect obstacles, and steer the robot away from them. It is also always active.

In an environment with no obstacles, the higher-level behavior never generates a signal. However, if it detects an obstacle it overrides the lower-level behavior and steers the robot away.

As soon as the obstacle is gone (and the higher-level behavior stops sending signals), the lower-level behavior gets control again.

Multiple, interacting layers of behaviors can be built to produce more and more complex robots.
Comparison of Subsumption with SPA

**Scalability:** Brooks claimed that where SPA architectures had to be redesigned substantially to allow for the inclusion of new abilities in the robotic system, the Subsumption Architecture allowed new abilities to be added by simply adding a new behaviour levels to the system that could override or subsume the lower levels behaviors where necessary without having to interfere or redesign the lower level behaviors.

**Performance:** Whereas SPA robots were slow and ponderous, Subsumption robots were fast and reactive. A dynamic world did not bother them because they constantly sensed the world and reacted to it.
However, behavior-based robots soon reached limits in their capabilities.

1. It proved very difficult to compose behaviors to achieve long-range goals
2. It proved almost impossible to optimize robot behavior.
In essence, robots needed the planning capabilities of the early architectures wedded to the reactivity of the behavior-based architectures.

This realization led to the development of layered, or tiered, robot control architectures.
These Hybrid architectures may be characterised by a layering of capabilities, where low level layers provide reactive capabilities, and high level layers provide the more computationally intensive deliberative capabilities.

The most popular variant on these hybrid architectures are Three Layered Architectures

1. Controller or Reactive Layer
2. Sequencer or Executive Layer
3. Planner or Deliberative Layer
The **Controller** (aka **Reactive**) layer provides low level control of the robot.

- It is characterized by a tight sensor-action loop.
- Its decision cycle is often on the order of milliseconds.
- In terms of software engineering the controller would be a series of drivers with basic responses, while from a biological point of view the controller is the set of nerve connections to muscles and other organs.
- Controller elements should have low computational complexity to allow them to react quickly to stimuli, and execute basic behaviours speedily.
The **Sequencer** (aka Executive) layer is between the low-level controller layer and the higher level planner layer.

It accepts directives from the planner layer and sequences them for the reactive layer.

Naturally, this ordering cannot be a simple linear list, since the environment being operated in can change unexpectedly, and primitive Behaviours in the Controller can fail.

For example, the executive layer might handle a set of via-points generated by a deliberative path planner, and make decisions as to which reactive behavior to invoke.
The **sequencer** layer is also responsible for integrating sensor information into an internal state representation.

For example, it might host the robot's localisation and mapping routines.

Decision cycles at the executive layer are usually in the order of a second.
The Planner or Deliberative Layer contains the heaviest computational components, traditionally containing an exponential or high polynomial state space searcher.

The planner generates global solutions to complex tasks.

Its decision cycle is often in the order of minutes.

The planner uses models for decision making. These models usually utilize state information gathered at the executive layer.
Overview of Robot Control Architectures

Figure: A Timeline of Robot Control Architectures.
Sense, Plan, Act was an early robot control procedure commonly abbreviated SPA. Today we use its fundamental concepts to remind us of the three critical capabilities that every robot must have in order to operate effectively:

**SENSE:** The robot needs the ability to sense important things about its environment, like the presence of obstacles or navigation aids.

**PLAN:** The robot needs to take the sensed data and figure out how to respond appropriately to it, based on a pre-existing strategy.

**ACT:** Finally, the robot must actually act to carry out the actions that the plan calls for.
Where is the S, P and A in this program?
SENSE: The robot uses a Touch Sensor to sense whether it has collided with an object.
PLAN: The overall strategy is to run forward unless something is in the way, which is detected using the Touch Sensor. If the Touch Sensor is pressed, the robot will turn away from the obstacle. This is all captured in the program, which runs on the robot, reading the sensor’s data and issuing the appropriate motor commands.
ACT: The robot acts by moving its motors in response to the given motor commands, which are given in combinations that produce forward movement and turns as appropriate.
What today’s lecture highlights is that there are different ways to structure your control program and that it is important to match the structure of your program to the task required.

This study of how to structure your programs to match the task required is what you will study in software engineering.

As regards the team project, what I want you to take from today’s lecture is that it is often quicker to take some time at the start of a project to make a design for the structure of your program rather than to dive straight into the coding.