2
Autonomous Systems Architectures

In this chapter, we provide an introduction to *architectures* for autonomous systems, specifically the software managing all the components within, and interactions between components of, an autonomous system. The description here is in no way comprehensive, and there are many other sources, for example Brooks (1999) and Winfield (2012), that provide much more detail. We will touch upon some of the theoretical aspects, such as layered/behavioural versus symbolic/component views or continuous control versus discrete control, will discuss practical architectural styles, such as hierarchical control systems and hybrid architectures, and will highlight practical robotic middle-ware, such as the *Robot Operating System* (ROS) (Quigley et al., 2009).

This background will help to motivate and explain the development of *hybrid agent architectures*. This style of autonomous system architecture will then be the one we primarily use throughout the rest of this book.

2.1 Architectures for Autonomous Systems

Embodied autonomous systems comprise many different physical components. Typically, these are *sensors* (such as cameras, infrared detectors, and motion trackers), components for *propulsion* (such as wheels, engines, legs, and wings), *communication* (such as Bluetooth, GPS, screens, and voice), and more general *actuators* (such as arms, magnets, and lifting aids). All these must be brought together and controlled to achieve the tasks associated with the overall system and, as is standard in Engineering, each of these physical components will likely have some form of software *control system* that manages its behaviour. Predominantly, *feedback* control systems (which use sensors to monitor the state of the system and then adjust actuators based on how far the sensor readings are from some ideal) are used at this level to adapt the behaviour of the component to that of its function and its environment.

Once we have such control components, they can be organised into numerous different architectures, some of which are briefly described.

Symbolic Artificial Intelligence: Sense–Plan–Act. A straightforward and modular approach, rooted in methods for logical reasoning by machines (often referred to as symbolic artificial intelligence), is that of the *sense–plan–act* architecture. Here, *sensing* is carried out to gather information about the system's environment, typically by invoking the system's sensors. This is then used to construct an internal 'world model', and then symbolic *planning* is invoked upon this world model in order to attempt to achieve the system's goals. The plan constructed is subsequently transformed into *actions* that are sent to the lower-level components to undertake. And then the activity cycles back to sensing.

Engineering: Hierarchical Control Systems. Once we have a collection of controllers managing the system's physical interactions, we can, in turn, generate a control system that, itself, manages a set of such sub-components. We can think of the basic control systems as sitting at the bottom of some layered data structure. Above these are the first layer of controllers, each of which manages some subset of the systems at the bottom layer. Further control elements, in the next layer up, handle sets of manager controllers, and so on. These layers can be viewed as a mathematical tree structure, with 'leaves' in the bottom layer each connected to only one controller 'node' in the layer above, and these nodes are connected to a single node in the layer above them. Each controller handles data/activity from the layer below, packages these up, and then sends them up to the node above. Similarly, a node interprets commands from above and delegates them to subordinate nodes below. It is typical that nodes nearer to the root of this tree deal with higher-level, more abstract, behaviours, while leaf nodes typically characterise detailed sensor or actuator control.

Robotics: Subsumption Architecture. The above idea of an abstraction hierarchy also influences *subsumption architectures* that are popular in robotics. As in hierarchical control systems, the higher layers represent more abstract behaviours than lower layers. Individual layers in a subsumption architecture form finite state machines, all of which take input directly from the sensors. Data is transmitted between the nodes of each layer in order to form a behaviour. A higher layer can *subsume* a lower layer by inserting new information into the data connections between its nodes (Brooks, 1986). So, for example, any decision made by a node may take into account decisions from nodes above it because of data inserted by the higher-level node, but if no information comes

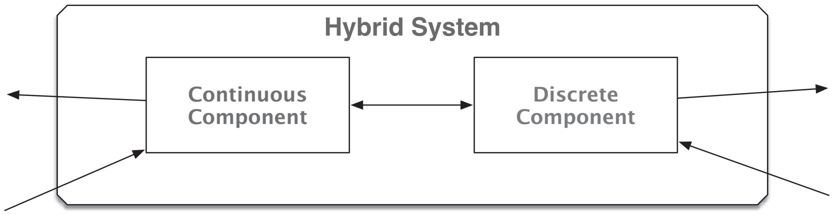

Figure 2.1 Hybrid agent architecture

from higher up, a node acts independently. As this approach was developed in part as a response to top-down planning approaches (such as sense–plan–act, mentioned earlier in the chapter), there is no identifiable planning node. Instead, behaviours at a node are activated through a combination of sensor inputs and behaviours from above.

To some extent, we can see the aforementioned sense–plan–act and subsumption approaches as mirror structures; each is hierarchical: one being driven primarily by the control/input aspects, the other being driven primarily by abstract/planning aspects. Neither of these approaches is without problems. The sense–plan–act approach, in its simplest form, can potentially be too slow when limited computational resources are available since it represents a centralised approach to reasoning about the current situation in its entirety. The tightly constrained cycle can restrict flexibility and efficiency, especially if the system becomes embroiled in time-consuming planning over complex models. Subsumption architectures are typically more efficient, but are often seen as being quite *opaque*; decisions are made somewhere in the hierarchy, but the reasons for these decisions can be hard to isolate.

There are a range of variations and approaches that attempt to take elements from each of these. These can often be classed under the umbrella term *hybrid architectures* (Figure 2.1), which typically involve the interaction between *continuous* components that deal with reasoning with numbers often involving differential equations and produce outputs that are numbers and that are often continuous in a mathematical sense – meaning that similar inputs generate similar outputs. Discrete components deal with more logical reasoning over whether facts are true or false and whether their outputs tend to be zeros and ones (or trues and falses).

Artificial Intelligence: Three–Layer Architectures. These architectures are again hierarchical, with the higher level being concerned with abstract planning and deliberation, while the lowest-level layer is concerned with feedback control mechanisms (Firby, 1990). The activity within the layers of a three-layer architecture tends to be much more autonomous than that prescribed by the fixed sense–plan–act cycle. So, sensing can be going on while deliberation occurs and both might be occurring in the background while external communication is being undertaken.

Engineering: Hybrid Control Systems. In a traditional hierarchical control system, all layers use continuous control controllers. *Hybrid* control systems typically mix continuous and discrete components. The continuous nodes are feedback control systems, as often with mathematically continuous behaviour. The discrete nodes control activity amongst the continuous nodes and, often, provide *discontinuous* changes in behaviour switching between quite different continuous controllers depending upon the situation – so two similar sets of inputs may produce very different behaviours. This kind of behaviour is difficult to produce using hierarchies of continuous controllers. Continuous controllers tend to be efficient at optimising with respect to the system's environment but, if the situation changes radically, this optimisation might be inappropriate. The discrete component gives the possibility of making a significant change, potentially between distinct continuous control regimes, either provoked by a big change in the environment or possibly by an internal choice.

These hybrid architectures are both flexible and very popular, but again have problems. From our viewpoint, their primary problem is that, as with subsumption architectures, the reasons for decisions are often very hard to discern. Recently, and particularly in autonomous vehicles, the use of *hybrid* **agent** *architectures* has increased.

2.2 Agent Architectures

An 'agent' is an abstraction developed to capture autonomous behaviour within complex, dynamic systems (Wooldridge, 2002). It is defined by Russell and Norvig (2003) as something that 'can be viewed as **perceiving** its environment through **sensors** and **acting** upon that environment through **effectors**'. There are many versions of the 'agent' concept (Franklin and Graesser, 1996), including ones in which the environment it perceives and acts upon is a software system, but the basic concept lends itself naturally to robotic and autonomous systems in which a decision-making component must perceive and act upon some external environment.

In agent architectures for autonomous systems, we encapsulate decision-making as a component programmed as an agent within the larger system.

Since the core new aspect of such a system is autonomous decision-making we can see how this decision-making agent component has a key role in this type of architecture. The agent must make decisions based on the perceptions it has from its environment (which is often dynamic and unpredictable) and under hard deadlines. In principle, the agent might *learn* new high-level behaviours besides making choices about behaviours. As suggested by Wooldridge (2002), an agent must fundamentally be capable of *flexible autonomous action*. Since we are encapsulating decision-making within these components, we require them to be, as much as possible, *rational* agents. These are loosely defined as agents that 'do the right thing' (Russell and Norvig, 2003).

Since the 1980s, the agent approach, and the concept of rational agents in particular, has spawned a *vast* range of research (Bond and Gasser, 1988; Bratman et al., 1988; Cohen and Levesque, 1990; Davis and Smith, 1983; Durfee et al., 1989; Shoham, 1993), not only regarding the philosophy behind autonomous decision-making but also programming languages/frameworks and practical industrial exploitation.[1] It has become clear that the agent metaphor is very useful in capturing many practical situations involving complex systems comprising flexible, autonomous, and distributed components.

Yet, it turns out that the 'rational agent' concept on its own is still not enough! Continuous control systems, neural networks, genetic algorithms, and so on, can all make decisions (and in many cases are designed to make these decisions rationally) and so are autonomous. However, while agents comprising such components can indeed act autonomously, the reasons for their choice of actions are often opaque. Consequently, these systems are very hard to develop and control. From our point of view, they are also difficult to formally analyse, and hence difficult to use where reliability and transparency are paramount.

In reaction to these issues, the *beliefs–desires–intentions* model of agency has become more popular as a mechanism for implementing decision-making. Agents following this model are sometimes referred to as *cognitive* agents. Again, there are many variations on this (Bratman, 1987; Rao and Georgeff, 1992; Wooldridge and Rao, 1999), and we will examine these further in Chapter 3, but we consider a cognitive agent to be one which

must have explicit reasons *for making the choices it does, and should be able to explain these if necessary.*

[1] IFAAMAS — The International Foundation for Autonomous Agents and Multiagent Systems – `www.ifaamas.org`.

Rational cognitive agents typically provide

1. *pro-activeness*
   that is, the agent is not driven solely by events and so it takes the initiative and generates, and attempts to achieve, its own goals
2. *social activity*
   that is, the agent interacts with other (sometimes human) agents and can cooperate with these in order to achieve some of its goals
3. *deliberation*
   that is, the agent can reason about its current state and can modify its subsequent actions and future goals according to its knowledge about situation.

Crucially, rational cognitive agents adapt their autonomous behaviour in an analysable fashion to cater for the dynamic aspects of their environment, requirements, goals, and knowledge.

Example 2.1 Spacecraft Landing
Imagine a cognitive agent controlling a spacecraft that is attempting to land on a planet. The agent has:

control of dynamic activity . . . . for example, thrust, direction, and so on;

information (i.e., 'knowledge'/'belief') . . . . for example, about the terrain and target landing sites;

motivations (i.e., 'goals'). . . for example, to land soon, and to remain aloft until safe to land.

The cognitive agent must dynamically

- *assess*, and possibly *revise*, the information held
- *generate* new motivations or *revise* current ones
- *decide* what to do, that is, *deliberate* over motivations/information

So, the requirement for *reasoned* decisions and explanations has refined the basic *hybrid agent architecture* approach (Figure 2.2) in order to require that the discrete agent component is actually a *cognitive* agent. These autonomous systems are based on the hybrid combination of

1. *cognitive agent* for **high-level** autonomous (discrete) decisions and
2. traditional *control systems* for **low-level** (continuous) activities.

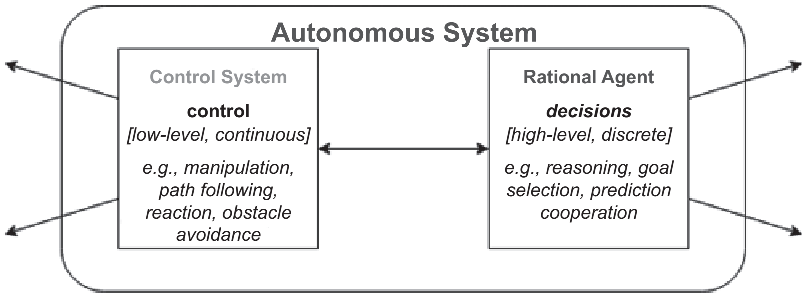
Autonomous System
Control System
control
[low-level, continuous]
e.g., manipulation, path following, reaction, obstacle avoidance
Rational Agent
decisions
[high-level, discrete]
e.g., reasoning, goal selection, prediction cooperation

Figure 2.2 Hybrid agent architecture

These can be easier to *understand*, *program*, and *maintain* and, often, be much more *flexible*.

It is important to note that, while there is typically just one cognitive agent, as autonomous (and particularly robotic) systems have become more complex, such architectures have involved an increasing number of other components. These could be multiple sensor control systems (including sophisticated image classification systems for perception), learning systems, planners, schedulers, monitors, navigation components, and so on. These will not all be traditional control systems, but can be composed of neural networks, genetic algorithms, or any mechanism for finding solutions to adaptive problems. We will touch on the wider issue of verification of these complex modular systems in Chapter 5.

Aside: Governor/Arbiter Agents. While these hybrid agent architectures are increasingly popular, many autonomous systems are still constructed using complex control hierarchies. We will see later that the possibility of identifying a high-level decision-making component can be advantageous for deep analysis, particularly where legal, safety, or ethical arguments need to be made. However, even for more opaque control architectures, there is a useful option. If the idea of an agent is not built into the architecture, we can, in some cases, add an agent to the system as an arbiter or governor (Figure 2.3). Here, the agent decides whether to *approve* or *reject* any proposed course of action for the autonomous system. This means, for example, that decisions about the safety, legality, or ethics of any course of action are assessed in a discrete (and analysable) way.

This is the approach taken by Arkin in his proposed *ethical governor* (Arkin, 2008) used in military uncrewed aerial vehicle (UAV) operations. This governor conducts an evaluation of the ethical appropriateness of a plan prior to

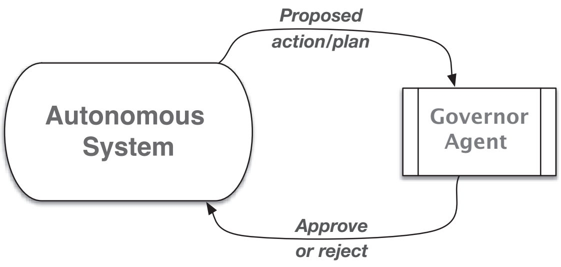

Figure 2.3 Governor architecture

its execution, prohibiting plans deemed unethical. Similarly, Woodman et al. (2012) developed a *protection layer*, which aims to filter out unsafe actions.

Though not explicitly constructed with an agent at its core, we can still view these system–agent pairs as hybrid agent systems.

We will see examples of these governor architectures in Chapters 10 and 13.

2.3 Modularity in Modern Robotic Software

It is widely recognised that robotic systems need to be programmed in a modular fashion, allowing software packages from different suppliers to be integrated into a single system. There are a number of technologies available to support this integration. One of the better known of these technologies is the Robot Operating System (ROS) (Quigley et al., 2009). This system enables individual software components, termed *nodes*, to specify their behaviour in terms of the messages they exchange with other parts of the system and provides support for nodes to communicate via ROS *core* nodes. The ROS distribution provides APIs for programming nodes in C and Python, and there is also support for programming ROS nodes in other languages (Crick et al., 2011).

The Robot Operating System does not impose any particular architecture upon a system built using it. Nodes may operate in publisher–subscriber or client–server modes (or both at once for different functionalities). Links may be made freely between any nodes in the system, as it is the structure of these links that will determine the particular architecture of any system built using ROS. However, the abstraction imposed by ROS upon an autonomous system – that of self-contained nodes that communicate via messages – has proved powerful and appealing, enabling the easy reuse of particular components for, for instance, image processing across multiple systems.

2.4 Practical Systems

In this book, we discuss the development of rational cognitive agent languages for programming autonomous systems, particularly in aerospace and robotics, and so we focus on the use of hybrid agent architectures. The cognitive agent can be programmed in any of the many agent programming languages, almost all of which have interpreters implemented in Java. Examples of such systems include

Autonomous Convoying System The autonomous convoy is a queue of vehicles in which the first is controlled by a human driver, but subsequent vehicles are controlled autonomously (Kamali et al., 2017).[2] The autonomously controlled 'follower' vehicles maintain a safe distance from the vehicle in front. When a human driving a vehicle wishes to join a convoy, they signal their intent to the convoy lead vehicle, together with the position in the convoy they wish to join. Autonomous systems in the lead vehicle then instructs the vehicle that will be behind the new one to drop back, creating a gap for it to move into. When the gap is large enough, the human driver is informed that they may change lane. Once this is achieved, autonomous systems take control and move all the vehicles to the minimum safe convoying distance. Similar protocols are followed when a driver wishes to leave the convoy.

Maintenance of minimum safe distances between vehicles is handled by two low-level control systems. When the convoy is in formation, control is managed using distance sensors and wireless messages from the lead vehicle. These messages inform the convoy when the lead vehicle is braking or accelerating and so allow smooth responses from the whole convoy to these events. This reduces the safe minimum distance to one where fuel efficiency gains are possible. In some situations control uses sensors alone (e.g., during leaving and joining). In these situations the minimum safe distance is larger.

A cognitive agent system manages the messaging protocols for leaving and joining, and switches between the control systems for distance maintenance. For instance, if a communication breakdown is detected, the agent switches to safe distance control based on sensors alone. A simulation of the vehicle control systems was created in MATLAB[3] and connected to the TORCS[4] racing car simulator.

[2] Software available from `github.com/VerifiableAutonomy`.
[3] `uk.mathworks.com`.
[4] `torcs.sourceforge.net`.

We will discuss this system in more detail with a focus on its verification in Chapter 11.

An Autonomous Robotic Arm The autonomous robotic arm system performs sort and segregate tasks such as waste recycling or nuclear waste management[5] (Aitken et al., 2014, 2017). The system is required to view a set of items on a tray and identify those items. It must determine what should be done with each one (e.g., composted, used for paper recycling or glass recycling) and then move each item to a suitable location.

The system integrates computer vision, a robot arm, and agent-based decision-making. It is implemented in ROS (Quigley et al., 2009). Computer vision identifies items on a tray (Shaukat et al., 2015). Their identities and locations are communicated to the agent that makes decisions about what should be done with each object. These decisions involve, for instance, sending anything that is plant matter to be composted, paper for recycling, and bricks for landfill. These decisions are then enacted by control systems.

Satellite Formations Traditionally, a satellite is a large and very expensive piece of equipment, tightly controlled by a ground team with little scope for autonomy. Recently, however, the space industry has sought to abandon large monolithic platforms in favour of multiple, smaller, more autonomous, satellites working in teams to accomplish the task of a larger vehicle through distributed methods. The system described here embeds existing technology for generating feedback controllers and configuring satellite systems within an agent-based decision-maker. The agent relies on *discrete* information (e.g., ‘a thruster is broken’), while system control uses *continuous* information (e.g., ‘thruster fuel pressure is 65.3’).

In the example a group of satellites can assume and maintain various formations while performing fault monitoring and recovery. This allows them to undertake collaborative surveying work in environments such as geostationary and low Earth orbits and among semi-charted asteroid fields. The system and scenarios are described more fully in Lincoln et al. (2013). We describe its verification in Chapter 8.

Lego Rovers The LEGO Rovers system (Figure 2.4) was developed to introduce the concepts of abstraction and cognitive agent programming to

[5] The robotic arm system involves proprietary software developed jointly by the universities of Liverpool, Sheffield, and Surrey and National Nuclear Labs. Requests for access to the code or experimental data should be made to Profs Fisher, Veres, or Gao.

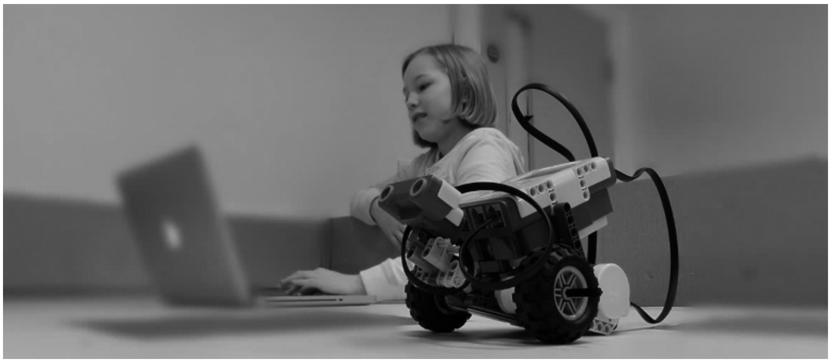

Figure 2.4 The LEGO Rovers system. Image courtesy of Phil Jimmieson and Sophie Dennis. Used with permission

school children.[6] It is used in science clubs by volunteer members of the STEM Ambassador scheme,[7] and has also been used in larger-scale events and demonstrations. The activity introduces the user to a teleoperated LEGO robot and asks them to imagine it is a planetary rover. The robot's sensors are explained; the user is shown how the incoming data is abstracted into beliefs such as *obstacle* or *path* using simple thresholds and can then create simple rules for a software agent, using a GUI, which dictates how the robot should react to the appearance and disappearance of obstacles, and so on. Aspects of the LEGO Rovers architecture are described in Dennis et al. (2016).

[6] `legorovers.csc.liv.ac.uk/` software available at `github.com/legorovers`.
[7] `www.stemnet.org.uk/ambassadors/`.