3

How a Robot Works

What is covered in this chapter:

- The basic hardware and software components that a robot consists of.
- The techniques we can apply to make a robot ready for interacting with people.

As a way of thinking about how a robot works, let us role-play by imagining being a robot. We might think we can do a lot of things, but we soon find out our capabilities are severely limited. If we are newly built robots, without appropriate software, our brains are entirely empty. We cannot do anything—move, know where we are, understand what is around us, even ask for help. We find the experience of being a robot rather strange and difficult to imagine. The main source of strangeness is that the new robot's brain is nothing like a human brain, not even an infant's. The robot has no basic instincts, no goals, no memory, no needs, no learning capabilities, and no ability to sense or act. To make a robot system, we need to integrate, and at least partially develop, hardware and software together to enable the robot to sense and act in the world.

This chapter is written for readers who have a limited technical background in intelligent interactive robotics. It describes the common components of a robot and how they are connected to enable participation in interaction. Section 3.1 explains basic ideas about the components needed to build a robot. Section 3.2 explains the types of hardware. Section 3.3 covers the integration of hardware and software and addresses the perception (e.g., computer vision), planning, and action control of the robot. Section 3.4 introduces sensors, such as cameras, range finders, and microphones, and Section 3.5 introduces actuators. Section 3.6 discusses software specifically designed for connecting other pieces of software to form one coherent program. Section 3.7 covers how to model interaction between the robot's program and the environment, whereas Section 3.8 goes specifically into artificial intelligence (AI) and machine learning. Finally, Section 3.9 discusses the most pressing limitations of robotics.

3.1 The making of a robot

To build a robot, one of the first steps is to establish connections between the robot's sensors, computer, and motors so that the robot is able to sense, interpret what it senses, plan actions, and then act them out. Once the robot is connected, for example, to a camera, its computer can read the data the camera provides. But the camera image is nothing more than a large table of numbers, similar to the following table:

9	15	10
89	76	81
25	34	29

From these numbers, can you guess what the robot is seeing? Perhaps a ball, an apple, or a fork? Assuming that each value in the table represents the lightness value of one sensor element in the camera, we can translate those numbers into a graphic that is more meaningful to humans (see Figure 3.1), but the graphic remains meaningless to the robot.

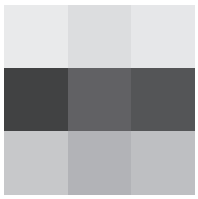

Figure 3.1 The camera's data translated into a grid of grayscale pixels.

You might be able to see a line in the image shown in Figure 3.1, but a robot has no understanding of what a line is. This line might be the edge of a cliff from which the robot could fall and damage itself. But the robot does not have a concept of height or gravity. It would not comprehend that it could fall if it crossed this line. It does not know that if it fell, it would likely come to rest upside down. Without the appropriate sensors, it would not register that it would be falling, nor that it abruptly came to a stop as it encountered the ground. It would not even recognize that its arm would be broken. In other words, even concepts that are vitally important for interacting with and surviving in the world around us that are innate in humans have to be explicitly programmed in a robot.

A robot, in essence, is a computer with a body. Any functionality needs to be programmed into the robot. A problem that all robots have to deal with is that although their sensors and motors are sufficient for operating in this world, their intelligence is not. Any concept of interest to roboticists needs to be programmed into the robot. This requires a lot of time and effort and often involves many cycles of trial and error. The analogue world out there is converted into a digital world, and translating tables of numbers into meaningful information and meaningful responses is one of the core goals of AI. Being able to identify a face from a large table of values, recognizing if a person has been seen before, and knowing that person's name are all skills that require programming or learning. Thus, the progress of human–robot interaction (HRI) is constrained by the progress that is made in the field of AI. Robotics engineers integrate sensors, software, and actuators to enable the robot to make sense of and interact with its physical and social environment. An engineer might, for example, use accelerometer sensors, which can detect acceleration and Earth's gravitational pull, to read the orientation of the robot and determine if it has fallen. A cliff sensor, consisting of a small infrared light source pointing down and a light sensor, can be used by the robot to avoid falling down a staircase.

Typical problems that robot engineers have to solve for the robot include the following:

- What kind of body does the robot have? Does it have wheels? Does it have arms?
- How will the robot know its location in space?
- How does the robot control and position its body parts—for example, arms, legs, wheels?
- What does the space around the robot look like? Are there obstacles, cliffs, doors? What does the robot need to be able to perceive about this environment to move safely?
- What are the robot's goals? How does it know when it has achieved them?
- Are there people around? If so, where are they, and who are they? How will the robot know?
- Is a person looking at the robot? Is someone talking to it? If so, what does the robot understand from these cues?
- What is the human trying to do? What does the person want the robot to do? How can we make sure the robot understands this?
- What should the robot do, and how should the robot react?
- Does the robot have enough battery power?

To address these questions, HRI researchers need to build or choose appropriate hardware and an appropriate morphology for the robot and then develop relevant programs—the software—that can tell the robot what to do with its body.

3.2 Robot types

At the time of this writing, a number of robots have been produced for the consumer market. Section 2.3 introduced some of the landmark robots, although this list is far from complete. For a more complete overview, we refer to the databases created by Anthropomorphic roBOT (ABOT) (www.abotdatabase.info) and the Institute of Electrical and Electronics Engineers (IEEE) (https://robotsguide.com/robots/). Although not all consumer robots become domestic staples, these commercial robots are often suitable platforms for HRI research. Commercially available robots can be categorized in a number of ways, including the following: social robots and drones, humanoids, androids, zoomorphic robots, virtual agents, telepresence and tele-operation robots, projection robots, and industrial robots. We will discuss these types in this section.

As covered in Chapter 1, *social robots* are robots that are designed to interact with humans (Hegel et al., 2009). This does not necessarily mean that a robot has a humanlike shape; as will be explained in Section 4.2 and Chapter 8, humans will readily perceive humanlike traits in other agents if they give off certain social cues or behave in certain ways. Thus, even a robot as simple as the Keepon (see Figure 2.7) can be considered a social robot because

its behavior creates the impression of a social presence. Obviously, different social robots will have different levels of complexity in their interaction. Paro, the baby seal robot (see Figure 2.8), can move its tail and open and close its eyes based on haptic feedback, but it does not communicate in other ways. In contrast, the iCub is shaped like a child (see Figure 3.8) and can display a variety of facial expressions, both of which create a myriad of ways to engage in social interaction.

Drones, and in particular, social drones, are flying robots that co-share space with humans (Obaid et al., 2020; Baytas et al., 2019; Johal et al., 2022) and can be used for applications in the household or education, among other use cases. Contrary to humanoid robot types, which will be discussed next, social drones commonly do not have a humanlike appearance.

Humanoid robots are robots that follow a general humanlike outline for their hardware. This means that generally speaking, the robot will be bipedal (although sometimes the legs are merged into a shaft on wheels, as is the case with Wakamaru and Pepper; see Figures 2.6 and 6.4); have a torso with a set of arms; and have a head with at least some facial features, such as eyes and a mouth. Well-known examples of humanoid robots include Nao, Pepper, Asimo, Robovie, and iCub.

Further humanlike in appearance are the *android robots*, which aim to mimic human looks as closely as possible. Although creating an exact replica of a human face and body out of silicon may be doable, animating it in such a way that it moves in a natural and humanlike way comes with its own set of challenges and issues, as discussed in greater detail in Section 4.2.1. Well-known androids include Kokoro and the Geminoid HI 4 robots (see Figure 4.5; see also Figure 4.7). Rather than trying to follow a human outline, *zoomorphic robots* are modeled after an animal shape. This can be an existing animal: for example, the Aibo is modeled after a dog (see Figures 2.10 and 11.1), the Paro after a baby seal (see Figure 2.8), and the Pleo after a baby sauropod dinosaur (see Figure 11.5). However, the robot's designer can also take some artistic freedom in their design and come up with their own fantasy animal, as was done in the creation of the Furby (see Figure 3.2).

Figure 3.2 The Furby (1998–2016) is a commercial zoomorphic robot that was particularly popular in the late 1990s.

An interesting in-between form of virtual assistants and embodied robots is the *projection robots*. These robots consist of a physical husk on which features are projected (see Figure 3.3). The benefits of this layout are that it becomes possible to mimic subtle movements, such as facial expressions, and that the appearance of the robot (e.g., skin color, gender) can be easily changed. At the same time, the animations of this robot remain projected rather than actual movements, and to our knowledge, there is no robot available yet that combines a projection with animated embodiment that would allow the robot to manipulate its environment.

Strictly speaking, *virtual agents* aren't robots: they are animated representations of an agent that is presented on a screen (e.g., a computer, tablet, or smartphone). Often, these agents are linked up with AI programs that can process spoken or written language commands and provide a response. These

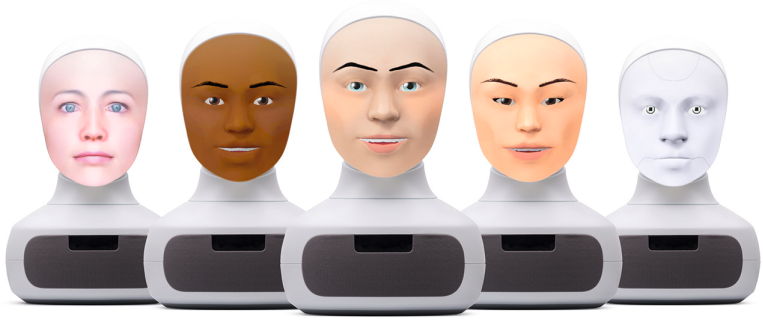

Figure 3.3 The Furhat robot combines a virtual face with a hardware embodiment through projection. (Source: Furhat Robotics)

applications have found their way into many areas, such as customer service, healthcare, sales, and education (Lugrin et al., 2022).

Telepresence robots can also be used as platforms for HRI research. Many different types exist on the market, including mobile versions, such as the Beam, and desktop versions like Kubi. Small mobile robots carrying a screen displaying a friendly face are being developed, soon to be ready for release in the consumer market.

Although commercially available robot hardware provides a wide variety of morphologies and sensing and programming capabilities, every robot is limited in what it can do; its appearance and capabilities constrain the interactions it can engage in. Researchers, therefore, also conceive and build their own robots, which range from simple desktop and mobile platforms with or without a manipulator to very humanlike android robots. The choice of a particular morphology for a robot to be used in HRI research often depends on the capabilities needed for the expected task (e.g., whether it needs to be able to pick up objects), the type of interaction (e.g., petlike interactions can benefit from an animal-like robot), and people's expectations and perceptions of different morphologies (e.g., humanoids may be expected to behave and be intelligent in ways similar to humans).

3.3 System architecture

All the hardware components of the robot need to be connected to a computer so that they can become interactive. The architecture of such a system can typically be divided into layers. Each layer typically only communicates with its direct neighbors (see Figure 3.4).

3.3.1 Hardware layers

At the bottom of the system are the different hardware components, such as motors and sensors. They are connected with cables to one or more computers. Some robots do all processing on board, but many robots will offload processing to other computers. In more recent robot software, the speech recognition, computer vision, and storage of user data often happen in the cloud, transmitted by internet-connected software services, typically operating on a pay-per-use basis. The advantage of cloud-based computing is that the

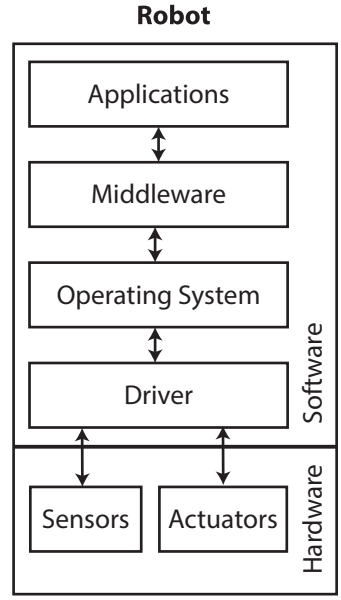

Figure 3.4 System architecture for robots.

robot has access to much more computing power and storage space than it could ever carry on board. Smart speakers, such as Google Home and Amazon Alexa, rely on cloud-based computing. However, a disadvantage is that when a robot relies on cloud-based computing, it needs robust communication with the cloud server. This is not necessarily guaranteed, particularly when a robot is mobile. Thus, time-critical computing and computing used to guarantee safety (e.g., emergency stops) are usually done on board.

3.3.2 Software layers

Above the hardware layers are the software layers. All the currently available robots are controlled by software running on one or several computers. The computers receive data from sensors and periodically send commands to the actuators.

On the computer, there is an operating system (e.g., Windows, Linux), which acts as the general platform allowing the software to access the general hardware of the computer, such as access to disks and files, and manages resources like memory and the central processing unit (CPU). The drivers enable the operating system to communicate with specific hardware components. These drivers normally come from the manufacturer of the hardware components, but some of them might already come integrated into the operating system. For example, when you plug a mouse into your computer, you normally do not need to install any drivers.

Although application software can directly run on the operating system, robotic applications often are run through *middleware*, consisting of many small pieces of software modules. Middleware is considered a "software glue," being in the middle of software modules and the operating system (see Section 3.6 for a more in-depth discussion).

3.4 Sensors

Most social robots are equipped with sensors that allow them to gauge what is happening in their environment. Many commonly used sensors are related to the three most commonly used modalities in human interaction—vision, hearing, and touch—but robots are not at all limited to human modes of sensing. It is often helpful, therefore, to consider what types of information the robot needs to perceive and what the most accurate and expedient ways are for it to do so, rather than focusing on reproducing human capabilities.

3.4.1 Vision

Camera

A camera consists of lenses that focus an image onto a sensor surface. The sensor surface is implemented using either a charge-coupled device (CCD) or, more often, a complementary metal-oxide-semiconductor (CMOS) technology. The basic element of a camera is a light sensor consisting mainly of silicon that converts light into electrical energy. A camera consists of an array of millions of these light sensors. Typically, color in a camera image is represented using three values, red (R), green (G), and blue (B). Hence, a camera is commonly referred to as an *RGB camera*. The sensors on the sensor surface are not sensitive to the color of the light hitting them; they are only sensitive to light intensity. To make an RGB camera, small color filters are placed on top of the sensor surface, with each filter letting through only red, green, or blue light (see Figure 3.5). Cameras are the richest and most

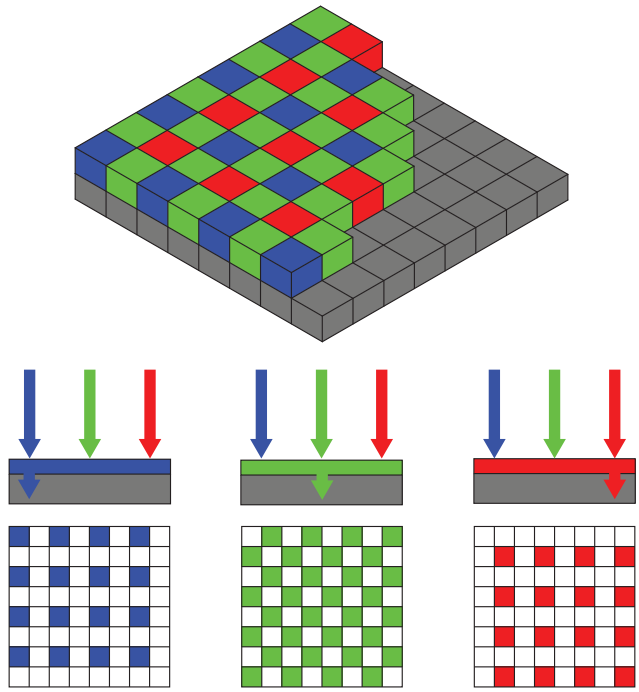

Figure 3.5 Array of CCDs in RGB camera.

complex sensors available to robots, and through the wide adoption of the RGB camera in digital cameras and smartphones, it has become miniaturized and very cheap.

Most cameras have a more restricted field of view than human vision. Whereas people can see more than 180 degrees, a typical camera might only see 90 degrees, thus missing a lot of what is going on in the periphery. A robot with a single camera will have a limited field of view and might have to rely on other sensors, such as laser range finders or microphones, to give it a sense of what is going on around it.

Most importantly, the camera image needs to be processed using computer-vision algorithms in order for the robot to be able to respond to its visual environment (see Section 3.8.2).

In computer-vision research, investigators often put cameras in the environment to facilitate accurate vision. Although this is one of the realistic approaches to yielding stable performance from computer vision, in the HRI setting, it is sometimes discouraged because people can feel uncomfortable around cameras. For example, in a project in which elderly people were being assisted in their home by a robot, the engineers would have loved to have cameras on the robot and in the home because it would have allowed the robot to accurately track and interact with people. However, the elderly participants were quite firm in their refusal of the installation and use of cameras, forcing the team to use localization beacons and laser range finders instead (Cavallo et al., 2014).

Depth sensors

Just as human vision uses stereo vision, knowledge about objects, and self-motion to figure out the distance to objects, so can computer-vision algorithms be used to extract a three-dimensional (3D) image from two-dimensional (2D) information. Stereo cameras have been the technology of choice for a long time, but in recent years, other technologies have emerged that allow us to see depth directly, without the need for computer vision. These "depth sensors" output a "depth image" or RGBD image (standing for red, green, blue, and *depth*), a map of distances to objects in view of the camera.

Typically, a depth sensor can measure the distance to objects a few meters away. Depending on the strength of the emitted infrared light, most depth sensors only work reliably indoors. There are several ways of making such depth sensors. One of the typical mechanisms is time of flight (TOF), in which a device transmits invisible infrared light pulses and measures the time taken between the moment when it transmitted the light and the moment when it received the light's reflection. Because the speed of light is so high, the camera would need to record the timing of the returning light with a precision that is out of reach of current electronics hardware. Instead, the camera emits pulses of infrared light and measures the phase difference between the light leaving the camera and the light returning to

Figure 3.6 The Microsoft Kinect Azure DK for Windows sensor. (Source: Used with permission from Microsoft)

the camera. The Microsoft Kinect One, the second iteration of Microsoft's game controller, is based on this principle (see Figure 3.6). Despite being developed as a game controller, it was quickly adopted by robot builders and is now widely used to give robots a sense of depth. Combined with appropriate software, the Kinect sensor can also perform skeleton tracking, which is helpful for figuring out where people are and what they are doing. Smaller devices are now available that return RGBD images based on a range of different technologies, including TOF, structured light, and stereo vision.

Laser range finders

Depth sensors are appropriate for measuring distances up to a few meters. In order to measure distances at longer ranges, researchers frequently use a laser range finder, also known as *light detection and ranging* (LiDAR). A typical laser range finder can measure distances to objects up to 30 meters away, and it samples the environment between 10 and 50 times per second. The accuracy of laser range finders is within a few centimeters.

The basic mechanism of this type of sensor is also TOF (as explained earlier, under *Depth sensors*). A laser range finder transmits a single beam of infrared laser light and measures the distance by measuring the time between the moment it transmits the laser beam and the time it receives its reflection. Typically, the transmitter and receiver are on a rotating platform, sweeping the laser beam around the environment. Thus, the device only measures distance in a single 2D plane, that is, the plane of rotation of the rotating platform.

Robots can have range finders mounted at different heights to scan for objects on a horizontal plane. Range finders close to the ground can sense objects on the floor and people's legs, whereas range finders that are set higher up can be used to sense objects on a table or counter (see Figure 3.7).

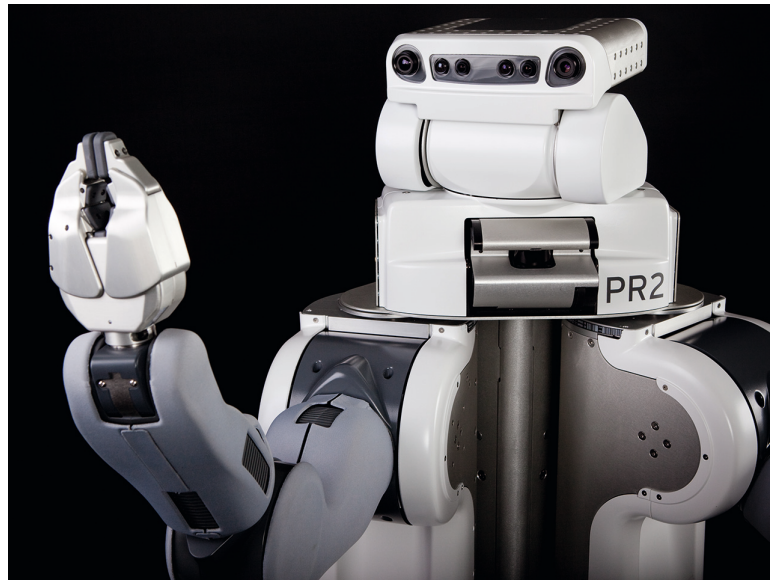

Figure 3.7 The PR2 robots (2010–2014): Can you tell where the range finder is? (Source: Willow Garage)

3.4.2 Audio

Microphones are commonly used devices for auditory sensing, and they convert sound into electrical signals. Microphones have different sensitivity profiles; some are omnidirectional, picking up all sounds in the environment, whereas others are directional, only picking up sounds in a cone-shaped area in front of the microphone. Combining multiple microphones into an array allows us to use "beam-forming" techniques, which can separate sound signals coming from a specific direction from ambient noises. Microphone arrays are used for sound-source localization, that is, getting an accurate reading on the angle of a given sound source with respect to its position in relation to the microphone array.

3.4.3 Tactile sensors

Tactile sensors can be important in HRI, for example, when the robot is physically guided by the user. Many different implementations exist, from physical buttons or switches to capacitive sensors, such as those found on touch screens.

The most commonly used tactile sensor is a mechanical push switch. It is often used together with a bumper. When a robot collides with an object, the switch is closed, allowing the robot to detect the collision. Pressure sensors and capacity sensors, like the ones reading your finger's position on a touch screen, can also be used to detect physical contact with the environment. Pressure sensors can be implemented using a range of technologies but usually contain a material that changes its electrical properties (resistance or capacitance) when force is applied (see Figure 3.8).

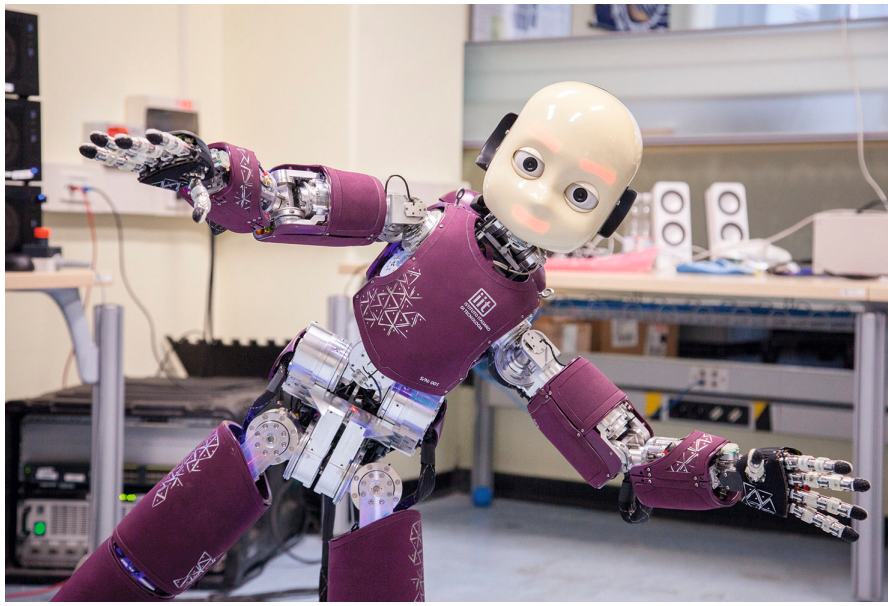

Figure 3.8 The iCub (2004–present) humanoid has capacitive tactile sensors worked into its fingers, palms, and torso. (Source: IIT Central Research Lab Genova)

Pressure sensors can help robots recognize whether and how hard they are touching a person or object. They are also very useful for enabling robots to pick up and handle objects appropriately. Tactile sensors can furthermore be used to allow the robot to know whether someone is touching it, and the robot can be programmed to respond accordingly. For example, the seal-like Paro robot has a tactile sensor net all over its body that allows it to sense the location and pressure with which a person is touching it and react by cooing for soft strokes and crying out after a harder hit.

3.4.4 Other sensors

Various other sensors exist, many of which can be relevant to HRI. Light sensors read the amount of light falling on the sensor and can be used to sense a sudden change in light, signaling that something has changed in the environment. When combined with a light source, they can be used to detect objects. A simple and very effective obstacle sensor combines an infrared light-emitting diode (LED) light with an infrared light sensor; when light bounces back from objects in front of the sensor, it can determine the distance to objects. This not only is used to detect obstacles in front of the robot but can also be used to sense when people are approaching the robot.

In recent years, the inertial measurement unit (IMU) has become a popular sensor. It combines three sensors—an accelerometer, a gyroscope, and a magnetometer—and is used to read the rotation and motion of the sensor or, more accurately, the rotational and translational acceleration. Recent advances in micro-electrical manufacturing have allowed these sensors to be miniaturized down to a few millimeters. They have become ubiquitous in

mobile phones and miniature drones, and when used in a robot, they allow the robot to sense if it falls or to keep track of where it has moved over time.

Far-infrared (FIR) sensors are cameras that are sensitive to long-wavelength infrared light, which is emitted by warm bodies. They can be used to detect the presence of people, as used in burglar alarms, or when integrated into an FIR camera, they can be used to record an image of the temperature of the room. FIR sensors are still expensive and are mainly used for thermal imaging, but eventually, they may allow the robot to see people at night or in cluttered environments.

It is important to realize that, unlike our own senses, sensors do not necessarily need to be mounted on the robot. A robot might rely on a ceiling-mounted camera to interpret the social environment, or it could use a wall-mounted microphone array to localize who is speaking. The whole environment could, in a sense, be considered part of a robot system.

3.5 Actuators

An actuator converts electrical signals into physical movements. A system with one actuator typically realizes motion either on one straight line or on one rotational axis. This means that the system has one degree of freedom (DOF). By combining multiple motors, we can develop a robot that has motion with multiple DOFs, allowing for navigation of a 2D plane or gesturing with humanlike arms.

3.5.1 Motors

The standard actuator for robots is a direct-current (DC) servo motor (see Figure 3.9). It typically consists of a DC motor and a microcontroller, with a sensor such as a potentiometer or an encoder, which outputs the absolute or relative position of the motor's output axis. To control the speed, the controller typically sends pulse-width modulation (PWM) signals to the DC motor. PWM is an on/off pulse, literally switching the motor on for a few milliseconds and then back off. This is done up to 100 times per second, and the duration of the on phase against the off phase (known as the *duty cycle*) determines the speed at which the motor rotates. The PWM signal controls the speed of the motor, and the controller sets the position of the motor. This is done through feedback control, where the controller continuously reads the position of the motor and adjusts the motor's PWM and direction to reach or maintain a desired position. For motors used in a robot's arms and head, the controller typically performs position control to rotate the motor toward a given commanded angle. For motors used in wheels on a mobile base, the controller typically performs velocity control to rotate the motor at the commanded velocity.

Robots can have different configurations and numbers of motors, depending on the body shape and the functions they are meant to perform. Commercially available cleaning robots, such as Roomba, typically have two motors driving

Figure 3.9
Connecting servo motors to each other allows robots to move around in various ways, such as in this robot arm. (Source: Trossen Robotics)

the wheels and one tactile sensor for moving around the room. Thus, Roomba has two DOFs. A simple nodding robot may have one motor to control its head direction, meaning that it has one DOF. A better-equipped humanoid may have three DOFs for its head, controlling pan, tilt, and yaw; two arms with four to seven DOFs; a mobile base with at least two motors; and sensors for visual, auditory, and tactile sensing. A robot arm, such as the KUKA (see Figure 3.10), must have at least six DOFs to manipulate an object. Three DOFs are necessary to locate its end effector (e.g., hand) to be in a position within a reachable range of the object, and another three DOFs are needed to grasp the object from any direction. A human arm can be approximated as an arm having seven DOFs, with an additional redundant one DOF beyond the necessary six DOFs for manipulation.

To grasp objects, a robot arm must have some type of end effector attached at the end. A 1-DOF gripper can be used to grasp an object, but more complex robot hands can have as many as 16 DOFs. Android robots, designed to closely resemble humans, typically have many more (e.g., 50 or more) DOFs and are able to control their facial expressions and other bodily movements in relatively nuanced ways compared to simpler robots.

Motors come in many different sizes, speeds, and strengths and thus have differing power needs. It is therefore important to consider from early on in the design process how the motor specifications relate to the robot's design and what kinds of actions a robot will need to make, such as whether it will need to pick up a one-kilogram bag or just needs to wave its arms, how big the robot can be while still fitting in well with its environment, how quickly it needs to respond to stimuli, and whether it needs to have a portable power bank or can be plugged into the wall.

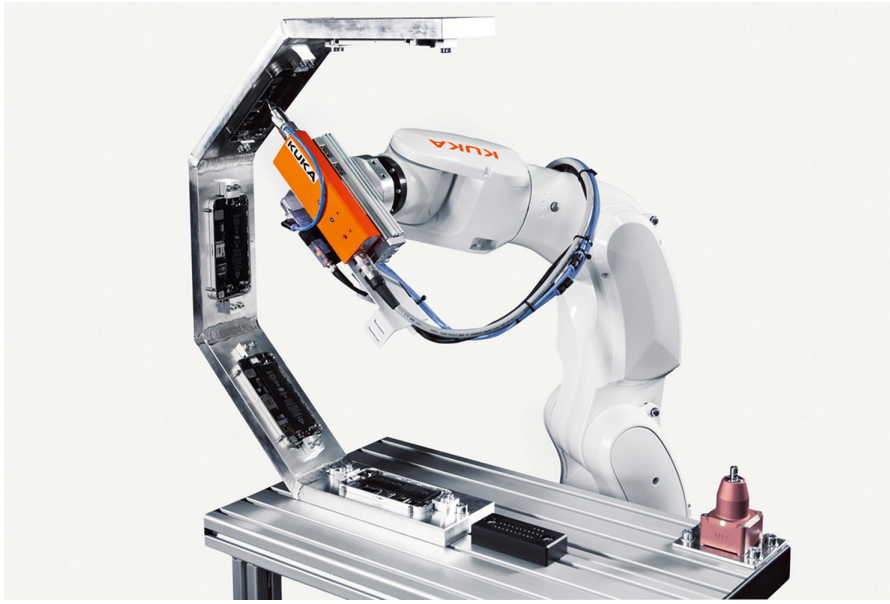

Figure 3.10 KUKA robot arm. (Source: KUKA)

3.5.2 Pneumatic actuators

A pneumatic actuator uses a piston and compressed air. Air is delivered from a compressor or from a vessel containing high-pressure air, which needs to be attached to the robot in some way. Pistons typically can extend and contract, depending on which valves are opened to let in the compressed air. As opposed to electric motors, pneumatic actuators produce linear motion, which is somewhat similar to human muscle motion, and are able to produce accelerations and speeds that are difficult to achieve using electric motors. Hence, they are often preferred for humanoid robots and android robots that need to gesticulate at humanlike acceleration and velocity (see Figure 3.11). The compressors that they need to operate can be quite loud, so it is important to consider how to give the robot access to compressed air without marring the interaction experience.

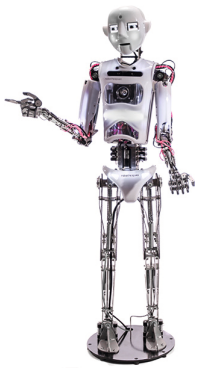

Figure 3.11 RoboThespian (2005–present) uses pneumatic actuators to achieve the acceleration required to deliver a convincing theatrical performance. The robot can run for around a day on a scuba tank's worth of compressed gas, although it can also be attached to a compressor. (Source: Photo copyright Engineered Arts)

3.5.3 Speakers

To generate sounds and speech, standard loudspeakers are used. Speakers are perhaps the cheapest actuator on the robot, but in terms of HRI, they are indispensable. Where to place a speaker or speakers in the robot's body is an important factor to consider when designing a robot that will interact with people. For example, Takayama (2008) showed that the relative height from which the voices of a user and an agent interacting with each other are projected can influence who is seen to be dominant in the interaction.

3.6 Middleware

3.6.1 What is a middleware?

Middleware is software that sits among software components, such as commonly available library modules and the application modules that the developers created for a specific purpose, as well as the operating system of the robot's computer. It is often considered as the "software glue" because its function is to ease the connection of those software components.

One of the functions of robotics middleware is to deal with the heterogeneity of hardware. Some applications are flexible about the type of sensors the robot uses as long as similar sensor data are provided. For instance, a 3D LiDAR could provide 3D distance data, but these data can be converted into the kind of 2D data that a 2D laser range finder provides. On the middleware, we can standardize a data format for 2D laser range finders so that we can use 2D laser range finders from different companies, as well as other sensors that output distance information, such as depth sensors and 3D LiDAR, in a similar way.

Another function of the robotics middleware is to help developers deal with complexity and reuse software modules. Almost all robotic applications are overly complex. It is unrealistic to create the whole application from scratch. Moreover, applications are often not really interested in the raw sensory data. They want to know abstracted information, for example, if a person is standing in front of the robot. Therefore, once someone creates a functional software module that enables the detection of a person in front of the robot, other developers would hope to reuse such an established module for many other robot applications, which would all be composed of somewhat different software and hardware components. Thus, "modules" (software components) are often shared within a community in which developers maintain and reuse various well-behaving modules.

To better understand the benefits of middleware, we need to take a closer look at how robots are built and how they work. Let's assume that we have two different robots, Marvin and S2E2. Both have two wheels to move around, but S2E2's wheels are 10 cm in diameter and Marvin's wheels are 20 cm. Thus, these robots are similar to the degree that they use the same methods to move forward, backward, and around, but they are different in wheel size.

Programmers might want these two robots to move between the fridge and the couch to deliver a beverage to their human user. For this purpose, the robots need to drive forward for two meters. The motors themselves can only be switched on or off. The wheels need a rotation sensor to detect how often they have turned. It would be really useful if the behavior to deliver the beverage developed for Marvin could also be used for S2E2. The middleware makes this possible by abstracting the robots. It translates the two-meter distance to 6.37 rotations for S2E2 and 3.18 rotations for Marvin.

Driving a robot straight ahead for two meters may look like an easy task, but it is not. It is possible for the wheels to slip, or a cat might dash across the path. Hence, the robot requires sensors to measure its location within the room. Marvin could have an ultrasound sensor mounted in front to measure

the distance between itself and the couch ahead. S2D2 might have a LiDAR sensor to measure the distance. Again, robots are similar, yet different. The middleware abstracts the two sensors to simply the distance from itself to the couch. The programmer can then monitor the progress of the robot and adjust the duration for which the motors are switched on and off.

But what about the cat that crosses paths with the robot? Both robots need to be able to move around an obstacle to drive to the couch. The problem of dynamically planning and adjusting a path toward the couch requires yet more sensors and software. These components should be able to communicate with each other so that they can trigger, for example, evasion behaviors. Middleware allows the different components to directly communicate with each other. Moreover, the problem of navigating the living room can be abstracted to both robots so that the software developed becomes reusable. This dramatically speeds up the process of software development because solutions for common problems can be shared. Path planning, obstacle avoidance, and localization have all been solved as problems in themselves, independent of the specific robot.

3.6.2 Robot Operating System

The Robot Operating System (ROS) is a middleware platform commonly used in the robotics and HRI community.[1] The name is somewhat misleading because ROS actually is not an operating system, such as MacOS, Linux, or Windows. Rather, it is a collection of software modules and tools. It deals with communications between sensors and modules and offers libraries and tools to support frequently used robot abilities, such as localization and navigation. ROS has a large community of users who often share modules on public open-source software repositories. The more developers use and extend this middleware to different sensors and actuators, the more attractive this platform becomes.

Some robot hardware developers decided not to develop their own software platforms for their robots as, for example, Aldebaran did for its Nao and Pepper robots. Instead, they offer modules for ROS to control and program their robots. PAL Robotics is an example of a company that offers ROS modules for its robots, such as TIAGo (see Figure 3.12).

Although ROS is playing an important role in the robotics and HRI communities, it still remains middleware that requires technical expertise to install, configure, and use. It is primarily useful for developers who are already familiar with code editors, repositories, and libraries. For these, ROS offers tools to launch code, introspection, debugging, visualization, plotting, logging, and playback. It does not, however, contain animation tools (see Section 3.7.2) or behavior editors (see Section 3.7.1). Unfortunately, there is no visual programming environment that would allow users without technical knowledge to click together behaviors and interactions.

[1] See www.ros.org

Figure 3.12 The TIAGo robot family uses ROS. (Source: TIAGo Family by PAL Robotics, © PAL Robotics S. L. 2024)

3.7 Applications

A robot is much more than a computer with a body. A computer operates in a clean digital environment, whereas a robot needs to interface with the messy, buzzing confusion of the real world. Not only does it need to make sense of the world, but it also needs to do so in real-time. This environment requires a radically different approach to robot software.

Architecture models

How should software for a robot be organized? A first rule of thumb, which is applicable to any software, is that messy program code should be avoided. Researchers and developers ideally aim to modularize software. One typical approach is to follow the "sense-plan-act" model (see Figure 3.13), in which inputs from sensors are processed using software modules specific to perception, which then convert sensor streams into high-order presentations. For example, audio recordings of speech are converted into a text transcription, or camera images are analyzed to report on the location of faces. Next, there is a section that deals with "planning," which plans the robot's next actions using information gleaned from the sensing process, then outputs commands to modules for action.

For instance, a person-finding perception module reports on the location of people detected in a 2D camera image and also returns the size of the heads, indicative of how close people are to the robot. Next, the planning module computes the head orientation for the robot to face the nearest speaker and sends a command to move the head to the output modules. The output modules then calculate which angle is needed for the robot's neck motors and send these to the low-level motor controllers.

The sense-plan-act approach is also known as the *deliberative approach* because the robot deliberates its next action. Quite often, we want a robot to respond quickly to external events, without spending a lot of time pondering what to do next. In this case, we often program simple "behaviors" for the robot (Brooks, 1991). Behaviors are tightly coupled sensor–action processing

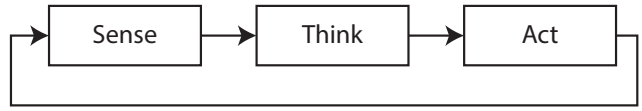

Figure 3.13
Sense-plan-act model.

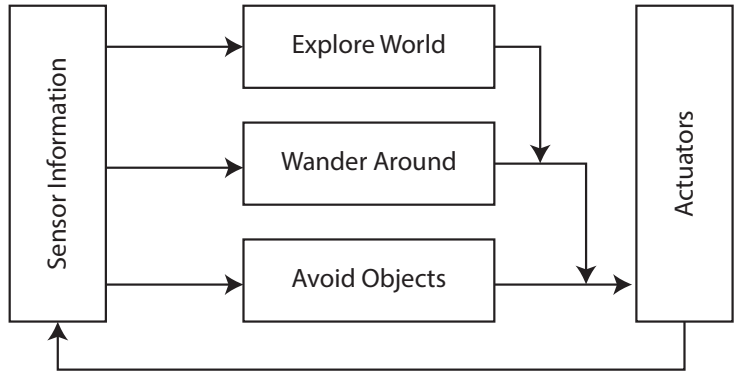

Figure 3.14 The subsumption behavior-based architecture.

loops that immediately respond to an external event. These can be used to make an emergency stop when the robot is about to drive down the stairs, but they can serve equally well in social interaction. When a loud bang is heard or when a face appears in view, we want the robot to respond as fast as possible. Act first; think later. Often, there are dozens of behaviors running on the robot, and mechanisms exist to mediate between which behaviors are active and which are not. One such mechanism is the subsumption architecture, which organizes behavior into hierarchies, allowing a behavior to activate or inhibit others (Brooks 1986; see Figure 3.14).

With this approach, even though the robot does not have an explicit "representation" of the world, it can still behave in an apparently intelligent way. For instance, if a cleaning robot uses two behaviors in parallel, one that avoids the wall and another that makes it have a slight pull to the right, the resulting, or emergent, behavior is that of wall following. Even though wall following wasn't programmed explicitly, it emerges from the interaction between two simpler behaviors. The vacuum robot Roomba has been developed with such an idea in mind.

In HRI studies, we typically find ourselves looking for a middle ground between deliberative and reactive approaches. We want a reactive control layer, which responds quickly to subsecond social events, followed by a deliberative layer, which formulates a coherent response to slower elements of the interaction, such as conversation.

In light of this, it is important to develop software that can be decomposed into a number of smaller modules. Even if the complete wealth of a sense-plan-act model is not needed, it is still common practice to separate modules into perception, planning, and action.

Planning is diverse in terms of components and complexity and depends heavily on the robot and the application. A cleaning robot may need to compute the next location to clean, whereas a companion robot may need

to make a decision on how it should initiate a conversation with a user. The software on a Roomba vacuum will therefore be radically different from that on a Pepper humanoid robot. For interactive robots, various forms of HRI knowledge will be embedded into the various software modules.

Action modules take care of the actuation and social output of the robot, such as nonverbal utterances, speech, hand gestures, and locomotion. For instance, the speech-synthesis module may receive text and convert this into spoken words, together with timing information that allows the robot to accentuate its speech with appropriate gestures.

3.7.1 Behavior editors

A robot has to be programmed in order for it to behave in the way we want it to. This can occur at different levels of detail. We could tell the right wheel to switch on for two seconds. Many of these detailed instructions can be combined into a more complex animation. When combining the robot movements with sensory input, we can describe them as behaviors. Such a behavior could be "greet the user when you see them for the first time." These behaviors can reuse many of the lower-level actions. For example, waving of the arm could be used for the "greeting behavior" but also for the "call for help" behavior.

The lower levels of programming are typically done on the top of the middleware layer, as described in Section 3.6. Working at these lower levels normally requires technical knowledge of the hardware and software of a robot. Experts on human and robot behavior often have more expertise in psychology and design but less experience with programming. Hence, it is desirable to have behavior design software that can be used without the need for in-depth programming skills.

Unfortunately, there are currently no open-source or commercial software programs that fulfill this need for multiple social robots. The developers of robots can provide tools for their specific robots, but these tools cannot be used for other robots. A good example is the Choregraphe software from Aldebaran (see Figure 6.9), which can be used to program the Nao and Pepper robots without the need for writing code. Users can drag and drop boxes, such as "Stand Up" or "Say Hello," to the canvas and connect them with lines to control the flow of the actions. This visual way of controlling the robot is, strictly speaking, still a form of programming, but it is often considered more intuitive. Children are often targeted by these visual programming paradigms, such as through Scratch (Sweigart, 2016) from the Massachusetts Institute of Technology (MIT; see Figure 3.15) or Blockly (Lovett, 2017) from Google. Sony's current fourth-generation Aibo dog (see Figure 2.10) uses a Blockly-like environment to enable owners to program its behavior. It does not, however, include the far superior MEdit motion editor (Cannon et al., 2007) and programming options (R-Code and Open-R) used from the first to third generations. These essential programming tools made it possible for Aibo to

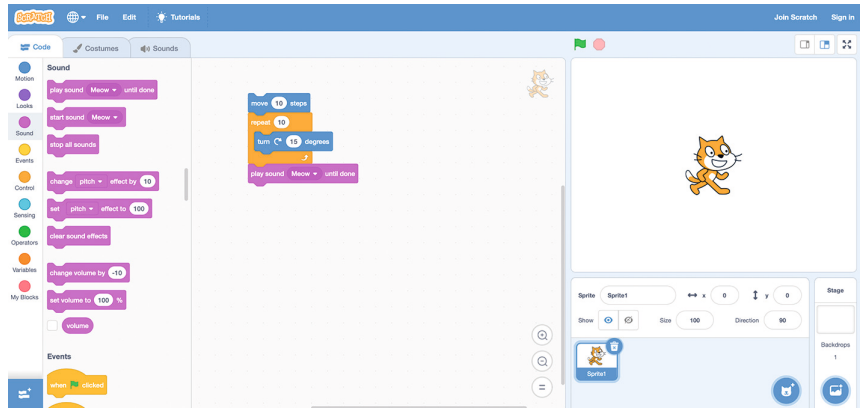

Figure 3.15 The Scratch programming environment. (Source: MIT Media Lab)

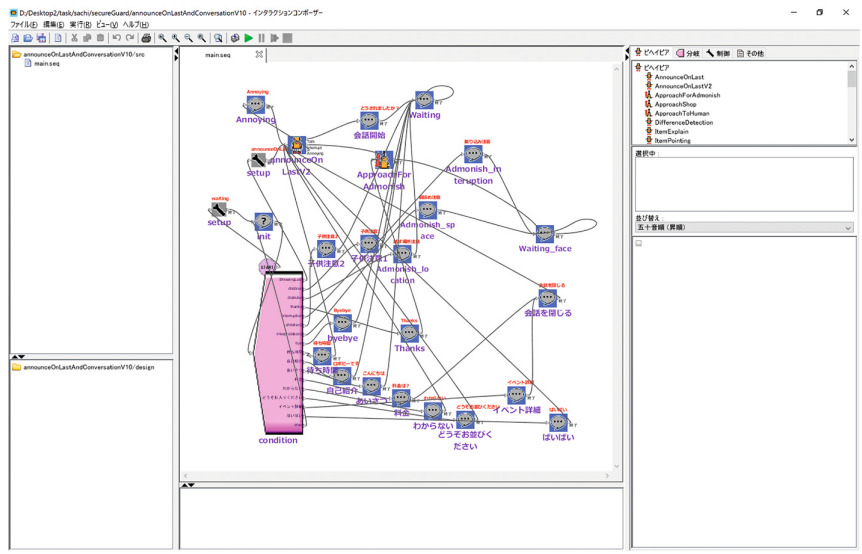

Figure 3.16 The Interaction Composer programming environment. (Source: ATR)

be used in the Robocup competition from 1999 to 2008. Current Aibo dogs are limited to home applications.

Another example of a platform-specific behavior editor is the Interaction Composer by ATR. It is used to control the Robovie (see Figure 4.14) series of robots. It has been used and further developed for over 14 years (Glas et al., 2016). It uses the visual design programming paradigm in which users connect elements through lines (see Figure 3.16). Although this software is abstract enough to potentially be applied to other robots, in practice, it is still closely linked to a couple of specific robots. Similar to Choregraphe, there are currently no plans to further open this behavior editor for other robots. Neither of them is available as open source, and hence others are also unable to achieve this.

Simulations and virtual representations of robots are used to test the robots' behavior before downloading and executing them on the actual robots. Gazebo

software,[2] for example, is widely used to simulate a robot in an environment. It does not, however, easily include human users. The HRI community has developed other simulation software that is specifically targeted at HRI, such as MORSE (Lemaignan et al., 2014c). From here, it is only a small step to building virtual robots in game engines.

Those working with modern game engines, such as Unity and Unreal, have many similar challenges as HRI researchers. They have to program an agent, either a robot or a character in a game, to interact with the user. This includes animations, conversations, and interactions with the environment. Game engines have already advanced tools for this purpose, and hence HRI researchers can use them for the design and control of robot behavior. The USARSim, for example, uses the Unreal Engine (Lewis et al., 2007), whereas the Robot Engine is based on Unity (Bartneck et al., 2015b), and MORSE is based on the open-source Blender Game Engine (Lemaignan et al., 2014c). Connecting the robot's hardware to the game engine can also easily be achieved using serial port communication to an Arduino microcontroller. As with any simulation of reality, it does not capture the noise and complexity of the real world. Moreover, the most difficult part, humans, is not easily included in the simulation. There are approaches to simulate simple behaviors of humans in the simulations (Kaneshige et al., 2021), or virtual reality (VR) techniques can be used to let human users interact with robots in the simulation world (Inamura et al., 2021), although those tests are so far rather limited, only serving as a pretest. Hence, it remains necessary to test the simulated behavior in the real world. Robots are not, for example, able to move as fast as their virtual counterparts.

Many of the behavior editors described in this section also include tools to manage the spoken dialogue between human and robot. Section 7.3.3 in Chapter 7 describes the functioning of dialogue managers in more detail.

3.7.2 Animation editors

Most of the animation software used to design the movement of robots borrows from the classical principles of keynote animations that are widely used in 2D and 3D animations. The animator uses a timeline and adds key frames to it. The positions of all the robot's actuators are defined as poses in these key frames. The pose of the robot can be set by using software to remotely control the robot into the right position, or users can simply move the physical robot into the desired position.

The movement between these key-frame poses can then be interpolated through the use of curves. One of the most popular curves is the fade-in and fade-out curve, where the movement slowly accelerates at the start and decelerates toward the end (see Figure 3.17).

[2] See https://gazebosim.org/home.

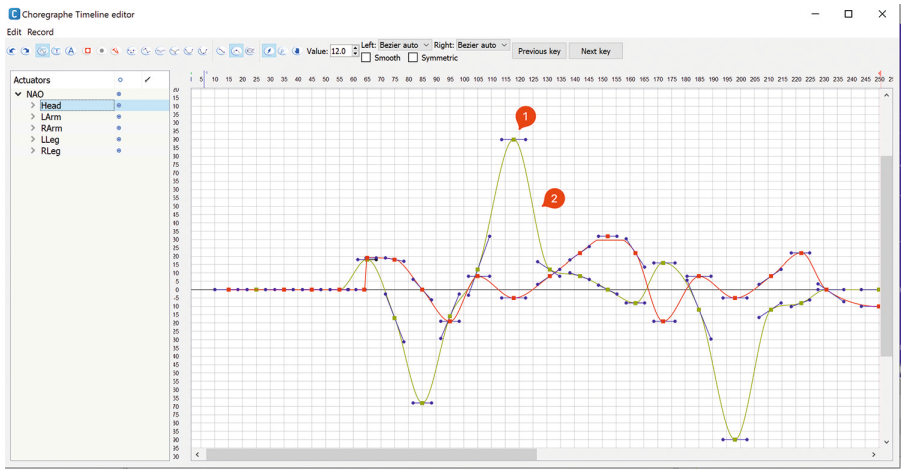

Figure 3.17 Key-frame animation in the Choregraphe software. (1) Shows a key frame, and (2) shows the interpolated movement. (Source: Software from Aldebaran, screenshot by Christoph Bartneck)

3.8 Artificial intelligence and machine learning

Many modules in the software perform some kind of intelligent processing. Those often benefit from techniques known as *artificial intelligence* (AI) or *machine learning*.

Although AI and machine learning are broad techniques, here we will focus on introducing some key concepts most relevant to HRI. We provide a basic introduction to supervised learning, followed by computer vision, which is one of the typical applications of supervised learning (see Section 7.2 of Chapter 7 for another important application, speech recognition). Other types of machine learning, such as generative models (to be used for speech synthesis and language generation; see Chapter 7) and reinforcement learning, will also be introduced.

There is a lot of recent attention to deep learning. In the media, the term *AI* is sometimes used interchangeably with the term *machine learning*. However, AI covers a broad set of techniques that perform any form of intelligent processing like humans do or beyond that. For instance, search algorithms that are used for motion planning are part of AI techniques, but they do not employ machine learning. Although general intelligence is one of the ultimate goals of AI research, it is still far out of reach.

Machine learning represents diverse algorithms that gain benefit ("learn") from data. Among them, *supervised learning* is most typically used in HRI applications. In this context, "supervised" concerns the fact that human developers manually provide labels to the training data. Supervised learning is usually used to address pattern-recognition problems, acquiring simple symbols (labels) from complex data, such as computer vision and speech recognition.

3.8.1 Supervised learning

Supervised learning is one type of machine learning; it specifically requires training data with correct labels. To understand what it is, let's try to imagine a specific task: classification of emotion in a human face. Using its camera, a

robot took an RGB image of a human face. How can it tell whether this person is wearing a happy or a surprised expression?

To solve this task, the robot should have a classifier program that is already well trained. The classifier converts input data (i.e., a face) into some kind of feature vector. Then, the trained classifier outputs the label (e.g., happy, surprised, etc.) based on the input of the feature vector. For simplicity, let's assume that the feature vector is a list of motions of various facial muscles; that is, there is one for each lip, another one for each eyebrow, and so forth. We know that if people are happy, their lip corners typically go up, and when surprised, their eyebrows are raised. Instead of explicitly programming these rules, in supervised learning, we let the classifier acquire them from the data. (Note that for the sake of simplicity, we made this example rather straightforward. However, identifying and specifying the relation between an input vector and a label is usually not at all simple. Thus, performance from supervised learning usually clearly outperforms explicit programming of such rules.)

What we provide to the classifier is *training data*. In our example, these would be a lot of human faces with correct labels—that is, a lot of happy faces, with all of them labeled as "happy," and a lot of surprised faces, with all of them labeled as "surprised." Typically, providing labels to all these instances constitutes intensive human labor. One by one, human workers have to check each image of a face and add the appropriate labels. Then, with some training algorithm (if successful), the classifiers acquire appropriate parameters or rules that enable them to (mostly) correctly classify unseen data. This process typically takes a huge amount of computation time and also requires a lot of additional labor by developers, who would work with hyperparameters (e.g., in the case of a neural network, how many layers, how those layers connect, how input vectors are represented, number of iterations of updating parameters, etc.).

Next, we explain the key elements and techniques for supervised learning.

Data sets

Machine learning requires data from which the robot can learn. This training data set should contain a large number of examples of the thing to be learned, which may be data from sensors or text, and generally has been manually annotated by humans. For instance, there can be a data set with camera images of human faces, and for each image, the emotion of the person is labeled, such as "neutral," "happy," or "angry." Such a set of example data and labels is referred to as a *data set*. Typical data sets contain hundreds of thousands or even millions of examples. The appropriate size of a data set varies depending on the complexity of the target machine-learning problem. Nevertheless, typically, larger data sets yield better performance.

Because the labeling process usually requires extensive labor, developers often rely on crowdsourcing data (e.g., using Amazon Mechanical Turk). However, we should be careful about the quality of the data as well as

the quantity of the data. Having ambiguous or wrong labels will harm the performance.

Because machine learning heavily relies on the amount and quality of data, sharing data sets, as well as sharing classifier modules (e.g., speech-recognition module), is a great community contribution. Researchers sometimes publish data sets together with their classification algorithm/system. There are specific websites for sharing data sets, such as Kaggle.[3]

Feature extraction

To aid machine learning, sensor data are often preprocessed by converting the sensor data into a more suitable representation and by extracting salient features from the data. This process is called *feature extraction*. There are many algorithms to extract features from raw sensor input. For instance, edge detection highlights the pixels in an image where the intensity abruptly changes, and a segmentation algorithm identifies regions in an image where the colors are all similar, which can indicate a face, hair, or an eye (see Figure 3.18).

Figure 3.18 Canny edge detection of a user operating the buttons on a robot.

Features are, in essence, numbers. These features are often placed into a feature vector, a row of numbers ready for processing. For instance, one could count up the number of pixels detected as an edge and use it as one of the variables of the feature vector. Researchers often manually analyze their data sets and identify salient features. For instance, with careful observation, one might find that a child fidgets more than an adult does; once such a feature is found, one can add variation of motion to the feature vector.

Classification based on training

Supervised learning is often used for *classification* problems. In classification, an algorithm decides, based on training data, what class an unknown data point belongs to. For example, given a camera image of a person, the classifier decides what emotion the person's face shows. (Note: Another frequent approach is regression, in which an algorithm provides a continuous number from unknown data, such as estimating the age of a person based on their face.)

Suppose we can compute a one-dimensional (1D) feature vector representing people's height and have a data set with two classes, "child" and "adult" (i.e., each data point in the training data will have a label saying whether the data point is a "child" or an "adult"). The classifier learns a threshold value from the training data set (e.g., 150 cm) to distinguish the two classes.

In this case, the feature vector contains only a single feature, the height of the user. We call this a *1D feature vector*. Classification algorithms typically work with thousands of features and try to recognize several classes, but sometimes up to thousands of classes. Classification errors are more or less inevitable. For instance, a tall child or short adult would be classified incorrectly with the previously described 1D feature vector.

[3] See www.kaggle.com/datasets

This material has been published by Cambridge University Press as Human-Robot Interaction by
Christoph Bartneck, Tony Belpaeime, Friederike Eyssel, Takayuki Kanda, Merel Keijsers, and Selma Sabanovic.
ISBN: 9781009424233 (https://www.cambridge.org/9781009424233).

Classification algorithms perform better (yield fewer errors) when having access to more data. Ideally, we want classification algorithms to "generalize," meaning they correctly handle data that they have never been exposed to. However, classification algorithms sometimes "overfit" to only the training data. When this happens, the algorithm does really well on the training data, but it performs poorly when confronted with new unseen data that are not included in the training data.

There are various algorithms available for classification problems. Support vector machines (SVMs) were traditionally often used with handcrafted features. Nowadays, it is more standard to use deep learning if a large amount of data is available. For the purpose of explainability, other algorithms, such as a decision tree, are sometimes used.

Deep learning

Deep learning is a family of neural-network techniques enabled by the increased availability of computational power. For instance, deep neural networks (DNNs) rely on artificial neural networks with a large number of layers of interconnected artificial neurons—hence the name "deep."

When input is 2D (typically, an image), *convolutional neural networks* (CNNs) are used. A CNN also has deep layers of neural networks. However, it has specific typological constraints among neurons, representing the convolution procedure in image processing. It is good at the task of identifying whether a target pattern exists somewhere within 2D data. For instance, in an object-detection task, it is more important whether there is a "dog" in the image or not rather than whether a "dog" is in the top left of the image. The classifier using a CNN better generalizes to various objects regardless of their locations in the image. See Section 3.8.2 for more information about computer vision.

In cases where the input is a time series, a family of *recurrent neural networks* (RNNs) is usually used. An RNN is a neural network, typically with deep layers, and also has a mechanism to keep internal states (i.e., a memory). At each time step, it receives an input, then provides an output label under the condition of its own memory. Long short-term memory (LSTM) is one of the famous RNNs. An RNN is often used for automatic speech recognition (ASR). In an ASR task, the words to be recognized usually depend on what was already spoken (e.g., if "how are ..." is already spoken, it is highly likely to hear "you" as the next word). See Section 7.2 of Chapter 7 for more information on speech recognition.

Another important deep-learning model is a *transformer*. It is used for variant lengths of input, similar to RNNs, but it does not have a memory mechanism. Instead, it comprises an encoding–decoding mechanism, along with an attention mechanism that focuses on the important part of the encoded input. It is often used for natural-language processing (NLP). The famous examples are *language models*, such as *BERT* and GPT-3. When given a sequence, they predict the next word, and most encode the sequence in doing

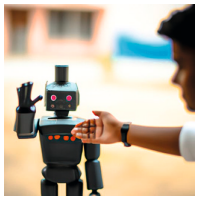

Figure 3.19 AI generated this image using the Dall-e platform. The text prompt used was "human playing with robot."

so. Such a model is trained in an unsupervised manner. This encoding is often used as *embedding* (a kind of feature vector) for other tasks by using fine-tuning techniques (see Section 3.8.1). Moreover, it is also used for generative tasks, such as image generation from natural-language input (see Figure 3.19), for which the learning process involves huge pairs of images and text, typically obtained from publicly available data (e.g., from Instagram).

Are you struggling with writing an essay or scientific paper? No problem—let a computer create it for you! Although students may be most familiar with ChatGTP, other language models have been trained specifically for generating scientific papers, such as SCIgen[a] (generates computer science papers, including figures and references) and Galactica (can generate scientific papers for any field of study; Taylor et al., 2022). It is important to note that language models do not understand the text they generate; in essence, they are a slightly shinier version of the "text completion" feature on your phone. Thus, although these programs return texts that sound confident, professional, and overall convincing, they often are factually incorrect.

Although these automatically generated papers are unlikely to pass the peer-review process of good journals, they could still be used to generate misinformation, and they thereby pose a serious threat to the integrity of science. After only two days online, Meta decided to shut down the demo web page for Galactica. The model itself is still available,[b] and people familiar with computer science can continue to (ab)use it.

[a] See https://pdos.csail.mit.edu/archive/scigen/.
[b] See https://github.com/paperswithcode/galai

For any of the aforementioned deep learning, it takes a large amount of computational power for training, but recent progress in using parallel computing and graphical-processing units (GPUs) has allowed us to train these networks within a reasonable amount of time.

Deep learning usually does not require careful feature extraction by hand. Instead, deep learning discovers the relevant features from the data by itself. A drawback is that deep learning requires huge amounts of data: typically, millions of data points are needed to train an algorithm. For instance, Google collected an enormous data set containing more than 230 billion data points to train its speech-recognition algorithm. GPT-3 was trained on 45 terabytes (TB) of text data from Wikipedia and books.

The complexity of deep learning makes it difficult to know exactly what the network bases its decisions on (e.g., we may not know what features it has identified or how it decided to use these features to come to a classification), which can be particularly problematic for HRI outside of the laboratory when we need to trust that the system will be robust, safe, and predictable. If the robot does something wrong, we need to be able to figure out how to debug and correct the system, as in the case of an autonomous Uber vehicle that had

trouble classifying a person crossing the road and ran over the person as a result (Marshall and Davies, 2018).

Transfer learning

The need for large data sets is a significant challenge for HRI because it is difficult to collect large amounts of data in which humans and robots are interacting.

The problem is more evident in deep learning. To moderate the problem, there is a technique known as *transfer learning* or *fine-tuning*, which reuses part of an existing trained network (often, embedding) and adds a small amount of labeled data to only tune a small part of the neural network (often near the output layer). By doing so, it learns new skills with a relatively small data set.

For instance, big language models, such as BERT and GPT, which are typically trained with trillions of sentences, are used for *intent recognition* via transfer learning using possibly less than a hundred sentences Huggins et al. (2021) (see Section 7.3 for a discussion of how intent recognition is used in HRI).

3.8.2 Computer vision

Computer vision is an important area for HRI. In essence, computer vision interprets a 2D array of numbers when working with single images, or a series of 2D images recorded over a period of time when working with video data. Computer vision can be rather straightforward and still very effective in the context of HRI. Motion detection, for example, can be achieved by subtracting two camera images taken just a fraction of a second apart. Any pixels that captured motion will have a nonzero value, which in turn can be used to calculate the region with the most motion. When used on a robot, a motion detector lets the robot orient itself toward the areas with the most motion, providing the illusion that the robot is aware of things moving, which, in the context of HRI, often involves people gesturing or talking.

Another computer-vision technique relevant to HRI is the processing of faces. The ability to detect faces in an image has advanced and can be used, for example, to let the robot look people in the eye. Face recognition (i.e., identifying a specific person in an image) is still a challenge, however. Impressive progress has been made in the last decade, mainly fueled by the evolution of deep learning, and it is now possible to reliably recognize and distinguish between hundreds of people when they are facing the camera. But face recognition typically fails when the user is seen from the side.

Skeleton tracking is another technique relevant to HRI. In skeleton tracking, the software attempts to track where the user's body and limbs are. This technique was first used in gaming on the Microsoft Xbox console, with software specific to the Kinect RGBD sensor, but is now a staple in many HRI applications. Several software solutions exist, but deep learning has enabled

the reading of skeletons of dozens of users in complex scenarios from a single simple camera image, without the need for an RGBD sensor. The software for this, called OpenPose, is freely available and often used in HRI studies (Cao et al., 2017).

There are many commercial and free software solutions that offer a range of out-of-the-box computer-vision functionality. OpenCV is perhaps the best-known offering; it is a free software library developed over 20 years. It can be used for facial recognition, gesture recognition, motion understanding, object identification, depth perception, and motion tracking, among others.

Because computer vision often requires a considerable amount of computational power, which is not realistic on small or cheaper robots, sometimes the computer-vision process is addressed on the cloud. In this case, the video stream of the robot is sent over an internet connection to servers on the cloud. There are commercial-based cloud solutions for face recognition, person identification, and image classification being sold on a per-use basis.

3.8.3 Reinforcement learning

Reinforcement learning is a very different approach to machine learning. It does not require any training data prepared in advance and does not necessarily need human supervision. Instead, a robot learns from successes and failures by really trying to act. What it learns is the optimal *policy*, the best *action* for each given *state*, which yields the best *reward*.

To understand how it works, imagine an example of a crawling robot that has an arm with two DOFs (search on YouTube using such keywords as "crawling robot Q-learning" to find examples). For simplicity, assume that the robot has a choice of only four distinctive actions: stretch out its hand, touch the ground, fold its arm, and lift its arm from the ground. The question here is which action the robot should choose.

This is somewhat complex question because the best action depends on the current pose of the robot. We know that to move forward, it should stretch out its hand, touch the ground, then fold its arm (here, it moves forward by "crawling"), and then it needs to detach from the ground. By repeating this, it can keep moving forward.

Reinforcement-learning algorithms learn such actions if rewards are designed appropriately. For a crawling robot, we would need a sensor to sense how much the robot moved forward. Then, the output from the sensor can be used as a reward. The reward (the fact that the robot moved forward) is only acquired when the previously described actions are performed in the correct order. If executed in the wrong order, no reward or even a negative reward would be given (e.g., if it folded its arm, touched the ground, and then stretched out). Many reinforcement-learning algorithms start from a random search, trying various actions from various states (here, we could use the current pose of its arm as the state), and memorize the rewards obtained for a given state. By repeating trials, the algorithm hopefully converges to find the best policy.

Various reinforcement-learning algorithms are available. Among them, the most famous is *Q-learning*, which is designed to remember the best reward for each "Q-state," defined as the combination of action and state. One expansion of Q-learning with deep-learning techniques is referred to as *deep Q-learning* (DQN), which uses a deep neural network to represent the Q-state.

Reinforcement learning typically takes time to repeat thousands of trials until it converges, even for relatively simple problems like the aforementioned crawling robot. Moreover, during the trial-and-error process, a robot unfortunately needs to fail a lot in order to "unlearn" all the specific ways in which it can be wrong. Sometimes, researchers try to use physics simulations to moderate these learning costs. For example, it took several months with seven robot arms learning in parallel for the arm to learn how to grasp various objects, whereas a model using a simulation and previously collected data took only a few days to learn the same behavior Ibarz et al. (2021). Applying reinforcement learning to HRI problems is not straightforward–because of the cost of failures, the time involved, and the difficulty in using simulation—yet researchers have started to try to find methods to make it feasible for HRI (e.g., Mitsunaga et al. 2008, McQuillin et al. 2022).

3.8.4 Adaptation

Both the user and the robot are adaptive systems. Humans have sophisticated skills to adapt their behavior and communication to their environment and others. When talking to a child, for example, adults tend to use simpler words and sentence structures, a phenomenon often referred to as *motherese* (Wrede et al., 2005; Rohlfing et al., 2005). Similarly, users also tend to adapt their communication when talking to a robot. They tend to speak slower and louder, particularly if the speech-recognition system does not seem to work properly (Kriz et al., 2010).

In return, robots are commonly expected to adapt their behavior to their users (Rossi et al., 2017) to optimize the interaction. Allan et al. (2022), for example, showed that users benefit from different types of praise from a robot based on their implicit self-theory. Users who consider self-attributes, such as intelligence, as malleable (incremental theory) prefer praise for their effort, whereas users who consider it to be unchangeable (entity theory) prefer to receive praise for their ability.

The robot needs to collect considerable data about each user to be able to infer characteristics such as these. Applying reinforcement learning would be one of the possible implementations for doing so. However, these data can only be collected in real-time, and hence data collection is limited. Therefore, it is still quite challenging research.

In both of these adaptations, the human and the robot change their own behavior. Humans have the additional option of explicitly changing the robot to their preferences. They may, for example, switch to a male voice or prefer a certain color of plastic over another. This adaptation is referred

to as *customization*. It does not require any sophisticated machine-learning techniques other than the adjustment of certain parameters.

3.9 Limitations of robotics for HRI

There are several limitations of robotics, some of which are specific to HRI and some of which apply to robotics in general. One general challenge is that a robot is a complex system that needs to translate between the analogue world and the digital internal computation of the robot. The real world is analogue, noisy, and often very changeable, and the robot first needs a suitable digital representation of the world, which the software then uses to make decisions. Once a decision is made, this is translated back into analogue actuation, such as speaking a sentence or moving a leg.

Another major challenge applicable to robotics at large is that of learning. Currently, machine learning needs to iterate through millions of examples to slowly nudge itself toward performing a task with a reasonable level of skill. Despite speedups due to advances in DNNs and GPUs, at the time of writing, computers need days or often weeks to learn, and this is only when all the learning can happen internally, for example, in simulation or using prerecorded data. Learning from real-time data that a robot samples from the world is still virtually impossible. Related to this is the challenge of *transfer*, or the performance of one skill transferring to another. For example, people can learn to play one game of cards and will then be able to transfer that knowledge to quickly pick up another game of cards with different rules. Machine learning typically struggles with this task and needs to start the learning of a new challenge from scratch.

The seamless integration of the various systems on a robot also represents a major challenge. Speech recognition, natural-language understanding, social-signal processing, action selection, navigation, and many other systems all need to work together in order to create convincing social behavior in a robot. On simple robots, this is manageable, but on more complex robots, the integration and synchronization of these various skills are still beyond our grasp. Face detection, emotion classification, and sound-source localization might each work well in isolation, but bringing the three together to make the robot respond in a humanlike manner to people approaching the robot is still a challenge. Greeting people who smile at the robot, looking up when the door slams, or ignoring people who show no interest in the robot may sound easy, but it is difficult to build such behavior that consistently works well. The challenge becomes formidable once further skills are added. Conversational robots, which aim to interact with people using natural language in addition to using their full suite of sensors to react in an appropriate manner, are only now being attempted in research labs across the world. It is unlikely that a robot will be built in the next decade that can handle a conversation as well as people can.

Robots and AI systems in general struggle with semantics: they often do not truly understand what happens around them. A robot might seem to respond

well to a person approaching it and asking for directions, but this does not mean that the robot understands what is happening—that the person is new to the space or where the directions it gives actually lead to. Often, the robot has been programmed to face people when they come near and to respond to the key words it hears. Real understanding is, at the moment, still exclusive to humans. Although there are research projects on imbuing AI systems with a sense of understanding (Lenat, 1995; Navigli and Ponzetto, 2012), there are not yet robots that can use their multimodal interaction with the world to understand the social and physical environment.

The reasons why AI has not yet achieved a humanlike general intelligence level are manifold, although conceptual problems were identified right from the outset. Searle (1980) pointed out that digital computers alone can never truly understand reality because they only manipulate syntactical symbols that do not contain semantics. In his *Chinese room thought experiment*, a slip of paper with Chinese symbols is slid under the door of a room (Searle, 1999). A man inside the room reads the symbols and comes up with a response by applying a set of rules he finds in a book full of instructions containing more Chinese characters. He then writes the response in the form of other Chinese characters and slides it back under the door. The audience behind the door might be under the impression that the man in the room understands Chinese, whereas in reality, he just looks up rules and has no understanding of what those symbols really mean. In the same manner, a computer also only manipulates symbols to come up with a response to input. If the computer's response is of humanlike quality, does that mean the computer is intelligent?

According to Searle's line of argument, IBM's chess-playing computer Deep Blue does not actually understand chess, and DeepMind's AlphaGo does not understand the game of Go. Both programs may have beaten human masters of the game, but they did so only by manipulating symbols that were meaningless to them. The creator of Deep Blue, Drew McDermott, replied to this criticism: "Saying Deep Blue doesn't really think about chess is like saying an aeroplane doesn't really fly because it doesn't flap its wings" (1997). That is, he debated that as far as it functions as it is supposed to, a new machine or AI does not need to replicate all the details of humans, animals, or birds. This debate reflects different philosophical viewpoints about what it means to think and understand and is still underway today. Similarly, the possibility of developing general AI remains an open question. All the same, progress has been made. In the past, a chess- or Go-playing machine would have been regarded as intelligent. But now it is regarded as the feat of a calculating machine—our criteria for what constitutes an intelligent machine have shifted along with the capabilities of machines.

In any case, no sufficiently intelligent machine has yet been built that would provide a foundation for many of the advanced application scenarios that have been imagined for robots. Researchers often fake the intelligence of the robot by applying the Wizard-of-Oz method (see Section 10.6.1 on page 182).

However, there are also some basic problems that we do not expect to find a solution for in the near future. One of the most basic limitations of HRI is the battery capacity. Most robots cannot operate for longer than an hour before having to recharge. This is a major constraint for mobile robots, particularly those that navigate in unstructured environments. For instance, once a robot is on its way somewhere, it already has to plan its return. Furthermore, this limitation makes it difficult for humans to experience longer-term interactions. Finally, robots like Nao cannot return to their charging station autonomously, meaning that either the user or the experimenter has to manage its battery charging.

Another physical limitation concerns the speed at which the robot can move. Here, we mean not only the robot's speed of driving around but also the speed with which a robot is moving its arms and head. Piumsomboon et al. (2012), for example, tried to motion-capture a Haka dancer and mapped his movements in real-time to several Nao robots (see Figure 3.20). The robot could only keep up with the human dancer if the dancer moved unnaturally slowly. Once the dancer unleashed his powerful Haka dance, the robots fell hopelessly behind. Another often-overlooked limitation of robots is that they cannot move silently. Humans can move their arms without making any noticeable sound. Robots, on the other hand, use electrical motors, gears, or pneumatic actuators. Although this inability to sneak around might be welcomed by some, it can be rather disturbing when humans want to go to sleep.

If robots were to create their top 10 list of things they hate about the world, then gravity would certainly be among the top entries. Simply refraining from

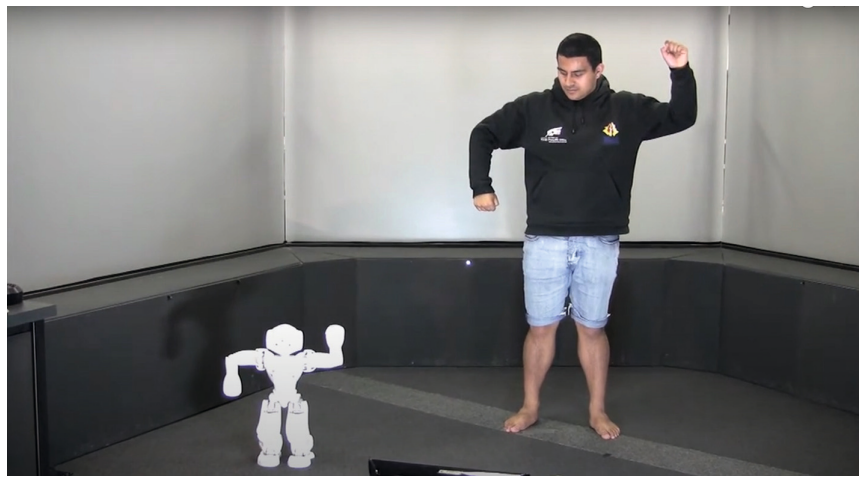

Figure 3.20 Nao robot trying to imitate a Haka dancer.

falling over is incredibly hard; in humans, this process requires an extremely fine-tuned close collaboration between different senses, such as vision, the vestibular system, haptics, and the body's sense of where one is in space (Wolfe et al., 2006). Science has yet to artificially recreate a similar kind of balance system. Furthermore, keeping in balance while moving around is even harder, particularly when the floor is uneven or when the robot has no method of getting up again.

From a human perspective, we are left wanting for the robot support that we have been promised for so many years. Even today, emptying a dishwasher remains an impossible task for robots. Over half a century ago, in 1966, the British Broadcasting Corporation (BBC) created a short film about Able Mabel, the Robot Housemaid.[4] It promised that robots would soon be able to handle many household tasks. Meredith Thring argued in this film that it would only take 1 million pounds to produce the first prototype. Needless to say, this vision of the future was far too optimistic.

The requirements of HRI often imply unrealistic assumptions about what can be achieved with current technology, and novice research and the public should be aware of the limitations of robotics and AI.

3.10 Conclusion

Robots are made from multiple software modules connected with sensors and actuators. Software design requires HRI knowledge, and conversely, HRI researchers need to have a basic understanding of software in order to provide useful knowledge for future HRI developers. For a robot to be successful, the different components need to be chosen and integrated with an eye toward the specific HRI application and its needs. Despite limitations, however, robots can be designed to interact successfully with humans in various types of short-term, and sometimes longer, interactions.

Questions for you to think about:

- Chapters 2 and 3 introduced various robot types that are available on the market. What sensors do these robots have? What actuators do they have? What hardware components do you think are crucial?
- Imagine a scenario where you want to use a smart social robot. Which sensors and actuators should it have? What skills should the robot have, and is software available to deliver these skills?
- What kind of data set would be needed to train a machine-learning algorithm for a new interaction capability of a robot, such as distinguishing your face from others?

[4] See www.bbc.co.uk/archive/mabel-the-robot-housemaid-1966/zhnvxyc

3.11 Exercises

The answers to these questions are available in the Appendix.

** **Exercise 3.1 Sensors** A list of technologies follows. Which ones are typically used as sensors on robots? Select one or more options from the following list:

1. Camera
2. Loudspeaker
3. Microphone
4. LED light
5. LiDAR
6. Servo motor
7. Ultrasound sonar

* **Exercise 3.2 Pepper's sensors, part 1** Have another look at Pepper (page 15). What sensor technologies does this robot have? Select one or more options from the following list:

1. Radar
2. Depth camera
3. Capacitive touch sensor
4. Global Positioning System
5. Inertial measurement unit
6. Oxygen sensor

*** **Exercise 3.3 Pepper's sensors, part 2** Based on your answer to the previous question, what functions do you think these sensors serve?

** **Exercise 3.4 How do sensors work?** Which of the following statements are correct? Select one or more options from the following list:

1. The light sensor in a camera can see only brightness.
2. A TOF infrared light sensor can measure depth up to 300 meters.
3. Inertial measurement units combine an accelerometer, microphone, and gyroscope.
4. Typical cameras can see up to 90 degrees.
5. An RGBD sensor is a camera that can estimate the distance to objects.
6. Omnidirectional microphones pick up sound from all around.

*** **Exercise 3.5 How do servo motors work?** Hobby servos are simple motors found in cheap robots. Which of the following statements are true? Select one or more options from the following list:

1. The position of a servo is controlled by the duty cycle of the control signal.
2. The speed of a servo is controlled by the voltage.
3. The servo motor continuously changes direction to maintain its set position.
4. The position and speed of a servo are controlled by switching it on and off.

5. An external position sensor is used to control the position of the servo.
6. A servo motor has two output axes.

*** Exercise 3.6 Finger** Have a look at your pointing finger. How many DOFs does it have?

**** Exercise 3.7 Degrees of freedom** What is the minimum DOFs that a robot needs to drive to every location in a room?

**** Exercise 3.8 Grasping** What is the minimum DOFs that a robot arm needs to grasp an object in reach from any direction?

**** Exercise 3.9 Linear actuators** What type of linear actuators are often used in social robots? Select one of the following options:

1. Hydraulic actuators
2. Pneumatic actuators
3. Aquatic actuators
4. Bimorph actuators

**** Exercise 3.10 Control model** What model is typically used to control a robot? Select one of the following options:

1. $Act \rightarrow think \rightarrow sense$
2. $Sense \rightarrow think \rightarrow act$
3. $Sense \rightarrow act \rightarrow think$

*** Exercise 3.11 Middleware** Which of the following are **not** middleware? Select one or more options from the following list:

1. Windows
2. Linux
3. ROS

**** Exercise 3.12 Middleware functions** This question focuses on robot middleware, such as ROS. Which statements are true? Select one or more options from the following list:

1. Offers basic functions to access hardware, like access to storage or input/output (IO) port
2. Uses different hardware (e.g., sonar and LiDAR) in an interchangeable way
3. Automatically creates code to realize HRI without the need for explicit coding
4. Provides standardized environments for programmers to share and reuse their modules
5. Helps programmers visualize what is communicated between modules

***** Exercise 3.13 Machine learning** Imagine we are going to build a classifier using deep learning to, for example, identify whether there is a person in a camera image or not. Which of the following statements are true? Select one or more options from the following list:

1. If we can maintain the quality of the data, more data will result in better performance.
2. Thanks to deep learning, we can train a classifier from scratch with only a small amount of data.
3. Thanks to deep learning, we do not need to handcraft features. We can directly use raw image data.
4. Thanks to deep learning, we do not need to provide labels. We can assign random labels to the data to start training.
5. We do not need to care about the topology of the neural network. Anything like DNN, CNN, RNN, or transformer can be chosen for this simple image-classification task, as long as it is a deep-learning method.

*** **Exercise 3.14 Robots that work with people** Watch this video, and then answer the question that follows.

Andrea Thomaz, "Next Frontier in Robotics: Social, Collaborative Robots," https://youtu.be/O1ZhWv84eWE

1. Thomaz demonstrates a robot meant to work together with people in everyday environments. Looking at Thomaz's robot, describe what kinds of technical components and capabilities it has that allow it to interact with people. What are the different social cues that the robot uses, and how do its components work together to produce those cues in the course of an interaction? The description does not have to go into great detail, but do describe how you think different components (e.g., gaze, manipulation, movement in space) work together in these interactions.

Future reading:

- For basic AI:
  Russell, Stuart, and Norvig, Peter. *Artificial Intelligence: A Modern Approach*. Pearson, Essex, UK, 4th edition, 2022. ISBN 978-1292401133. URL www.worldcat.org/oclc/1242911311
- For recent machine learning:
  Goodfellow, Ian, Bengio, Yoshua, and Courville, Aaron. *Deep Learning*. MIT Press, Cambridge, MA, 2016. ISBN 9780262035613. URL www.deeplearningbook.org
- For basic robotics:
  Matarić, Maja J. *The Robotics Primer*. MIT Press, Cambridge, MA, 2007. ISBN 9780262633543. URL www.worldcat.org/oclc/604083625
- For diverse topics in robotics:
  Siciliano, Bruno, and Khatib, Oussama. *Springer Handbook of Robotics*. Springer, Berlin, 2016. ISBN 9783319325507. URL www.worldcat.org/oclc/945745190