CHAPTER 2

The campaign for real time: robot bodies and brains

Wobbler had written an actual computer game like this once. It was called "Journey to Alpha Centauri." It was a screen with some dots on it. Because, he said, it happened in real time, which no-one had ever heard of until computers. He'd seen on TV that it took three thousand years to get to Alpha Centauri. He had written it so that if anyone kept their computer on for three thousand years, they'd be rewarded by a little dot appearing in the middle of the screen, and then a message saying, "Welcome to Alpha Centauri. Now go home." (Pratchett, 1992a)

This work was implemented on two robots, Cog and Kismet (see Figure 2-1), developed at the Humanoid Robotics Group at the MIT AI Lab by various students over the past decade. More accurately, it was implemented on their sprawling "brains" – racks of computers connected to the bodies by a maze of cables, an extravagant sacrifice offered up to the gods of real-time performance. This chapter dips into the minimum detail of these systems necessary to understand the rest of the thesis. The interested reader is referred to the excellent theses of Williamson (1999), Breazeal (2000), and Scassellati (2001), on whose shoulders the author stands (or is at least trying to peer over).

2.1 Cog, the strong silent type

Cog is an upper torso humanoid that has previously been given abilities such as visually-guided pointing (Marjanović et al., 1996), rhythmic operations such as turning a crank or driving a slinky (Williamson, 1998a), and responding to some simple forms of joint attention (Scassellati, 2000). For a good overview of the research agenda that Cog embodies, see Brooks et al. (1999).

2.1.1 Low-level arm control

Cog has two arms, each of which has six degrees of freedom organized as shown in Figure 2-2. The joints are driven by series elastic actuators (Williamson, 1995) – essentially a motor connected to its load via a spring (think strong and torsional rather than loosely coiled). The arm is not designed to enact trajectories with high fidelity. For that a very stiff arm is preferable. Rather, it is designed to perform well when interacting with a poorly characterized environment. The spring acts as a

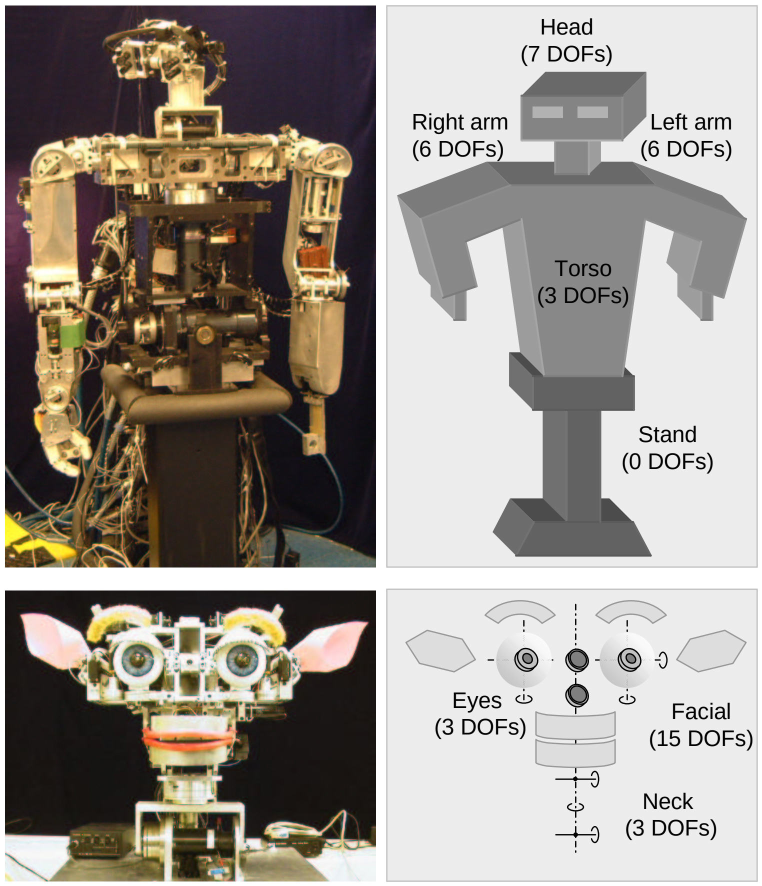

Figure 2-1: The robots Cog (top) and Kismet (bottom). Kismet is an expressive anthropomorphic head useful for human interaction work; Cog is an upper torso humanoid more adept at object interaction.

low pass filter for the friction and backlash effects introduced by gears, and protects the gear teeth from shearing under the impact of shock loads. A drawback to the use of series elastic actuators is that they limit the control bandwidth in cases where the applied force needs to change rapidly. The force applied by an electric motor can normally be changed rapidly, since it is directly proportional to the current supplied. By putting a motor in series with a spring, this ability is lost, since the motor must now drive a displacement of the spring's mass before the applied force changes. For the robot's head, which under normal operation should never come into contact with the environment, and which needs to move continuously and rapidly, series elastic actuators were not used. But for the arms, the tradeoff between control bandwidth and safety is appropriate. Robot arms are usually employed for the purposes of manipulation, but for this work they instead serve primarily as aides to the visual system. The target of a reaching operation is not assumed to be well characterized; in fact the reaching operation serves to better define the characteristics of the target through active segmentation (see Chapter 3). Hence the arm will habitually be colliding with objects. Sometimes the collisions will be with rigid, more or less unyielding structures such as a table. Sometimes the collisions will be with movable objects the robot could potentially manipulate. And sometimes the collisions will be with people. So it is important that both the physical nature of the arms, and the manner in which they are controlled, be tolerant of "obstacles".

The arms are driven by two nested controllers. The first implements force control, driving each

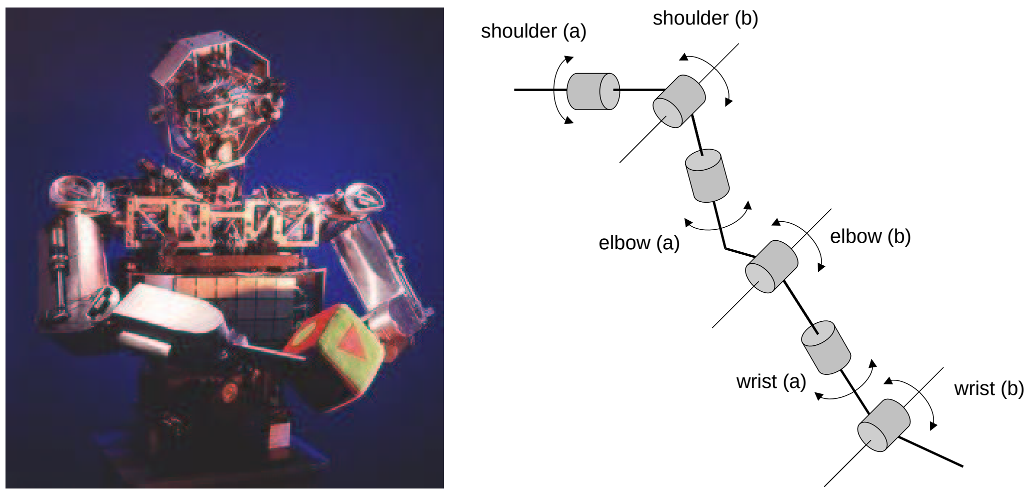

Figure 2-2: Kinematics of the arm, following Williamson (1999). There are a total of six joints, divided into a pair for each of the shoulder, elbow, and wrist/flipper.

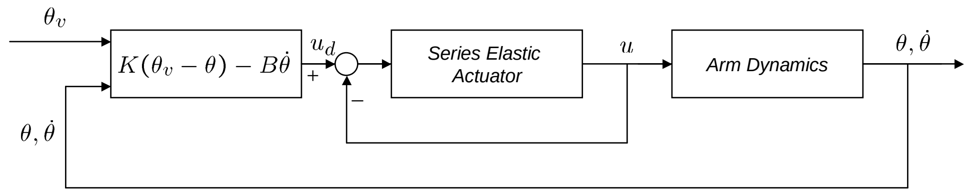

Figure 2-3: Control of a joint in the arm, following Williamson (1999). An inner loop controls the series elastic actuator in terms of force, working to achieve a desired deflection of the spring as measured by a strain gauge. An outer loop controls the deflection setpoint to achieve a desired joint angle, as measured by a potentiometer.

motor until a desired deflection of the associated spring is achieved, as measured by a strain gauge. This high-speech control loop is implemented using an 8-axis motor controller from Motion Engineering, Inc. A second loop controls the deflection setpoint to achieve a desired joint angle as measured by a potentiometer. Figure 2-3 shows this second loop, following (Williamson, 1999). Various extensions and modifications to this basic approach have been made, for example to incorporate a feed-forward gravity compensating term, but the details are beyond the scope of this thesis.

2.1.2 Low-level head control

Figure 2-4 shows the degrees of freedom associated with Cog's head. In each "eye", a pair of cameras with different fields of view provides a step-wise approximation to the smoothly varying resolution of the human fovea (Scassellati, 1998). The eyes pan independently and tilt together. The head rolls and tilts through a differential drive. There is a further pan and tilt associated with the neck. There are a number of redundancies in the degrees of freedom to permit rapid movement of the eyes followed by a slower compensating motion of the relatively massive head. The head contains a 3-axis inertial sensor to simplify gaze stabilization.

The motors of the head are connected to optical encoders and driven by an 8-axis motor controller from Motion Engineering, Inc. The motor controller is configured to permit both position

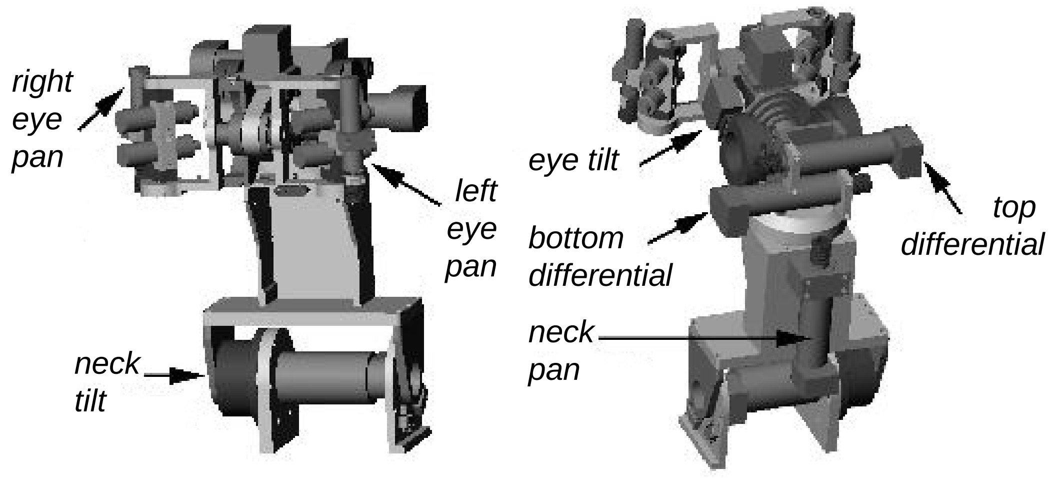

Figure 2-4: The motors of Cog's head, following Scassellati (2001). The degrees of freedom are loosely organized as pertaining to either the eyes, head, or neck. Pan and tilt (but not roll) of the eyes can be achieved at high speed without moving the mass of the head.

Figure 2-5: Kismet, the cute one.

and velocity control. Much has been written about both the low-level and strategic control of such a head – see, for example Scassellati (2001) – so the details will be omitted here.

2.2 Kismet, the cute one

Parts of this work were developed on and reported for Kismet. Kismet is an "infant-like" robot whose form and behavior is designed to elicit nurturing responses from humans (Breazeal et al., 2001). It is essentially an active vision head augmented with expressive facial features so that it can both send and receive human-like social cues. Kismet has a large set of expressive features - eyelids, eyebrows, ears, jaw, lips, neck and eye orientation. The schematic in Figure 2-1 shows the degrees of freedom relevant to visual perception (omitting the eyelids!). The eyes can turn independently along the horizontal (pan), but turn together along the vertical (tilt). The neck can turn the whole head horizontally and vertically, and can also crane forward. Two cameras with narrow fields of view rotate with the eyes. Two central cameras with wide fields of view rotate with the neck. These cameras are unaffected by the orientation of the eyes.

The reason for this mixture of cameras is that typical visual tasks require both high acuity and a wide field of view. High acuity is needed for recognition tasks and for controlling precise visually guided motor movements. A wide field of view is needed for search tasks, for tracking multiple

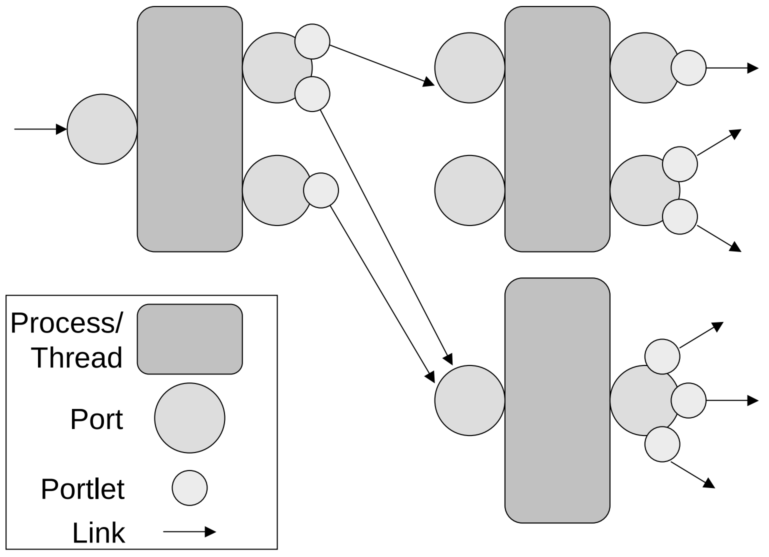

Figure 2-6: Communications model. Every process or thread can own any number of ports. Every port can be directed to send data to any number of other ports. Since different processes will access this data at different rates, it is useful to consider each port as owning several “portlets” that manage each individual link to another port. Given the most conservative quality of service settings, data will persist in the communications system as long as is necessary to send it on the slowest link.

objects, compensating for involuntary ego-motion, etc. A common trade-off found in biological systems is to sample part of the visual field at a high enough resolution to support the first set of tasks, and to sample the rest of the field at an adequate level to support the second set. This is seen in animals with foveate vision, such as humans, where the density of photoreceptors is highest at the center and falls off dramatically towards the periphery. This can be implemented by using specially designed imaging hardware, space-variant image sampling (Schwartz et al., 1995), or by using multiple cameras with different fields of view, as we have done.

2.3 The cluster

Cog is controlled by a network of 32 computers, with mostly 800 MHz processors. Kismet is controlled by a similar but smaller network and four Motorola 68332 processors. The network was designed with the demands of real-time vision in mind; clearly if it was acceptable to run more slowly (say, one update a second instead of thirty) then a single machine could be used. The primary engineering challenge is efficient interprocess communication across the computer nodes. We chose to meet this challenge by using QNX, a real-time operating system with a very clean, transparent message-passing system. On top of this was build an abstraction to support streaming communications and modular, subsumption-like design.

2.4 Cluster communication

Any process or thread can create a set of Ports. Ports are capable of communicating with each other, and shield the complexity of that communication from their owner. As far as a client process or thread is concerned, a Port is a fairly simple object. The client assigns a name to the Port, which gets registered in a global namespace. The client can hand the Port a piece of data to transmit, or read data the Port has received either by polling, blocking, or callback. There are some subtleties in the type of service required. The client can specify the kind of service required – a sender can specify whether transmission be guaranteed, or whether new data override data not yet sent; a receiver can independently specify whether, of the data the sender attempts to pass on, reception should be guaranteed or whether new data received by a Port should override data not yet read by its owner, and under what conditions the sending Port should be made to wait for the receiver.

Objects passed to the communications system obey a Pool interface. They can be cloned and recycled. The Port will clone objects as necessary, with an associated Pool growing to whatever size is required. This will depend on the rates at which all the links attached to the Port (via Portlets) read data at. By default, the owner of the Port is insulated from needing to know about that. For simple objects, cloning can be achieved with simple copies. The complex objects, such as images, a reference counting approach is worth using. Overall, this approach avoids unnecessary copies, and minimizes the allocation/deallocation of objects in the communications system. It is compatible with the existence of "special" memory areas managed by other entities, such as a framegrabber.

Ports and Portlets either use native QNX messaging for transport, or sockets if running on or communicating with a non-QNX system. The name server used is QNX's native `nameloc` service, or a simple socket-based `wide_nameloc` service for communicating with a non-QNX system. By default, these issues are transparent to client code. The system can operate transparently alongside other methods of communication, since it doesn't require any special resources such as control of the main process.