Layout of the Section

This section is divided into eight chapters, one to define robotics and the other seven to intertwine both the theory and practice associated with each paradigm. Ch. 2 describes the Hierarchical Paradigm and two representative architectures. Ch. 3 sets the stage for understanding the Reactive Paradigm by reviewing the key concepts from biology and ethology that served to motivate the shift from Hierarchical to Reactive systems. Ch. 4 describes the Reactive Paradigm and the architectures that originally popularized this approach. It also offers definitions of primitive robot behaviors. Ch. 5 provides guidelines and case studies on designing robot behaviors. It also introduces issues in coordinating and controlling multiple behaviors and the common techniques for resolving these issues. At this point, the reader should be almost able to design and implement a reactive robot system, either in simulation or on a real robot. However, the success of a reactive system depends on the sensing. Ch. 6 discusses simple sonar and computer vision processing techniques that are commonly used in inexpensive robots. Ch. 7 describes the Hybrid Deliberative-Reactive Paradigm, concentrating on architectural trends. Up until this point, the emphasis is towards programming a single robot. Ch. 8 concludes the section by discussing how the principles of the three paradigms have been transferred to teams of robots.

End Note

Robot paradigm primitives.
While the SENSE, PLAN, ACT primitives are generally accepted, some researchers are suggesting that a fourth primitive be added, LEARN. There are no formal architectures at this time which include this, so a true paradigm shift has not yet occurred.

1 *From Teleoperation To Autonomy*

Chapter Objectives:

- Define *intelligent robot.*
- Be able to describe at least two differences between AI and Engineering approaches to robotics.
- Be able to describe the difference between *telepresence* and *semi-autonomous control.*
- Have some feel for the history and societal impact of robotics.

1.1 Overview

This book concentrates on the role of artificial intelligence for robots. At first, that may appear redundant; aren't robots intelligent? The short answer is "no," most robots currently in industry are not intelligent by any definition. This chapter attempts to distinguish an intelligent robot from a non-intelligent robot.

The chapter begins with an overview of artificial intelligence and the social implications of robotics. This is followed with a brief historical perspective on the evolution of robots towards intelligence, as shown in Fig. 1.1. One way of viewing robots is that early on in the 1960's there was a fork in the evolutionary path. Robots for manufacturing took a fork that has focused on engineering robot arms for manufacturing applications. The key to success in industry was precision and repeatability on the assembly line for mass production, in effect, industrial engineers wanted to automate the workplace. Once a robot arm was programmed, it should be able to operate for weeks and months with only minor maintenance. As a result, the emphasis was

The term AI is controversial, and has sparked ongoing philosophical debates on whether a machine can ever be intelligent. As Roger Penrose notes in his book, The Emperor's New Mind: "Nevertheless, it would be fair to say that, although many clever things have indeed been done, the simulation of anything that could pass for genuine intelligence is yet a long way off."[115] Engineers often dismiss AI as wild speculation. As a result of such vehement criticisms, many researchers often label their work as "intelligent systems" or "knowledge-based systems" in an attempt to avoid the controversy surrounding the term "AI."

A single, precise definition of AI is not necessary to study AI robotics. AI robotics is the application of AI techniques to robots. More specifically, AI robotics is the consideration of issues traditional covered by AI for application to robotics: learning, planning, reasoning, problem solving, knowledge representation, and computer vision. An article in the May 5, 1997 issue of Newsweek, "Actually, Chess is Easy," discusses why robot applications are more demanding for AI than playing chess. Indeed, the concepts of the reactive paradigm, covered in Chapter 4, influenced major advances in traditional, non-robotic areas of AI, especially planning. So by studying AI robotics, a reader interested in AI is getting exposure to the general issues in AI.

1.3 What Can Robots Be Used For?

Now that a working definition of a robot and artificial intelligence has been established, an attempt can be made to answer the question: what can intelligent robots be used for? The short answer is that robots can be used for just about any application that can be thought of. The long answer is that robots are well suited for applications where 1) a human is at significant risk (nuclear, space, military), 2) the economics or menial nature of the application result in inefficient use of human workers (service industry, agriculture), and 3) for humanitarian uses where there is great risk (demining an area of land mines, urban search and rescue). Or as the well-worn joke among roboticists goes, robots are good for *the 3 D's*: jobs that are dirty, dull, or dangerous.

THE 3 D'S

Historically, the military and industry invested in robotics in order to build nuclear weapons and power plants; now, the emphasis is on using robots for environmental remediation and restoration of irradiated and polluted sites. Many of the same technologies developed for the nuclear industry for pro-

try; processing immune suppressant drugs may expose workers to highly toxic chemicals.

Another example of a task that poses significant risk to a human is space exploration. People can be protected in space from the hard vacuum, solar radiation, etc., but only at great economic expense. Furthermore, space suits are so bulky that they severely limit an astronaut's ability to perform simple tasks, such as unscrewing and removing an electronics panel on a satellite. Worse yet, having people in space necessitates more people in space. Solar radiation embrittlement of metals suggests that astronauts building a large space station would have to spend as much time repairing previously built portions as adding new components. Even more people would have to be sent into space, requiring a larger structure. the problem escalates. A study by Dr. Jon Erickson's research group at NASA Johnson Space Center argued that a manned mission to Mars was not feasible without robot drones capable of constantly working outside of the vehicle to repair problems introduced by deadly solar radiation.[51] (Interestingly enough, a team of three robots which did just this were featured in the 1971 film, *Silent Running*, as well as by a young R2D2 in *The Phantom Menace*.)

Nuclear physics and space exploration are activities which are often far removed from everyday life, and applications where robots figure more prominently in the future than in current times.

The most obvious use of robots is manufacturing, where repetitious activities in unpleasant surroundings make human workers inefficient or expensive to retain. For example, robot "arms" have been used for welding cars on assembly lines. One reason that welding is now largely robotic is that it is an unpleasant job for a human (hot, sweaty, tedious work) with a low tolerance for inaccuracy. Other applications for robots share similar motivation: to automate menial, unpleasant tasks—usually in the service industry. One such activity is janitorial work, especially maintaining public rest rooms, which has a high turnover in personnel regardless of payscale. The janitorial problem is so severe in some areas of the US, that the Postal Service offered contracts to companies to research and develop robots capable of autonomously cleaning a bathroom (the bathroom could be designed to accommodate a robot).

Agriculture is another area where robots have been explored as an economical alternative to hard to get menial labor. Utah State University has been working with automated harvesters, using GPS (global positioning satellite system) to traverse the field while adapting the speed of harvesting to the rate of food being picked, much like a well-adapted insect. The De-

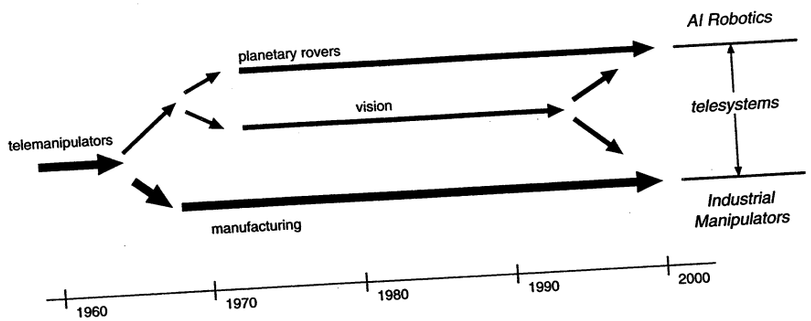


Figure 1.1 A timeline showing forks in development of robots.

placed on the mechanical aspects of the robot to ensure precision and repeatability and methods to make sure the robot could move precisely and repeatable, quickly enough to make a profit. Because assembly lines were engineered to mass produce a certain product, the robot didn't have to be able to notice any problems. The standards for mass production would make it more economical to devise mechanisms that would ensure parts would be in the correct place. A robot for automation could essentially be blind and senseless.

Robotics for the space program took a different fork, concentrating instead on highly specialized, one-of-a-kind planetary rovers. Unlike a highly automated manufacturing plant, a planetary rover operating on the dark side of the moon (no radio communication) might run into unexpected situations. Consider that on Apollo 17, astronaut and geologist Harrison Schmitt found an orange rock on the moon; an orange rock was totally unexpected. Ideally, a robot would be able to notice something unusual, stop what it was doing (as long as it didn't endanger itself) and investigate. Since it couldn't be preprogrammed to handle all possible contingencies, it had to be able to notice its environment and handle any problems that might occur. At a minimum, a planetary rover had to have some source of sensory inputs, some way of interpreting those inputs, and a way of modifying its actions to respond to a changing world. And the need to sense and adapt to a partially unknown environment is the need for intelligence.

The fork toward AI robots has not reached a termination point of truly autonomous, intelligent robots. In fact, as will be seen in Ch. 2 and 4, it wasn't until the late 1980's that any visible progress toward that end was made. So what happened when someone had an application for a robot which needed

real-time adaptability before 1990? In general, the lack of machine intelligence was compensated by the development of mechanisms which allow a human to control all, or parts, of the robot remotely. These mechanisms are generally referred to under the umbrella term: teleoperation. Teleoperation can be viewed as the "stuff" in the middle of the two forks. In practice, intelligent robots such as the Mars Sojourner are controlled with some form of teleoperation. This chapter will cover the flavors of teleoperation, given their importance as a stepping stone towards truly intelligent robots.

The chapter concludes by visiting the issues in AI, and argues that AI is imperative for many robotic applications. Teleoperation is simply not sufficient or desirable as a long term solution. However, it has served as a reasonable patch.

It is interesting to note that the two forks, manufacturing and AI, currently appear to be merging. Manufacturing is now shifting to a "mass customization" phase, where companies which can economically make short runs of special order goods are thriving. The pressure is on for industrial robots, more correctly referred to as industrial manipulators, to be rapidly reprogrammed and more forgiving if a part isn't placed exactly as expected in its workspace. As a result, AI techniques are migrating to industrial manipulators.

1.2 How Can a Machine Be Intelligent?

ARTIFICIAL INTELLIGENCE

The science of making machines act intelligently is usually referred to as *artificial intelligence*, or AI for short. Artificial Intelligence has no commonly accepted definitions. One of the first textbooks on AI defined it as "the study of ideas that enable computers to be intelligent,"[143] which seemed to beg the question. A later textbook was more specific, "AI is the attempt to get the computer to do things that, for the moment, people are better at."[120] This definition is interesting because it implies that once a task is performed successfully by a computer, then the technique that made it possible is no longer AI, but something mundane. That definition is fairly important to a person researching AI methods for robots, because it explains why certain topics suddenly seem to disappear from the AI literature: it was perceived as being solved! Perhaps the most amusing of all AI definitions was the slogan for the now defunct computer company, Thinking Machines, Inc., "... making machines that will be proud of us."

partment of Mechanical and Material Engineering at the University of Western Australia developed a robot called *Shear Majic* capable of shearing a live sheep. People available for sheep shearing has declined, along with profit margins, increasing the pressure on the sheep industry to develop economic alternatives. Possibly the most creative use of robots for agriculture is a mobile automatic milker developed in the Netherlands and in Italy.[68;32] Rather than have a person attach the milker to a dairy cow, the roboticized milker arm identifies the teats as the cow walks into her stall, targets them, moves about to position itself, and finally reaches up and attaches itself.

Finally, one of the most compelling uses of robots is for humanitarian purposes. Recently, robots have been proposed to help with detecting unexploded ordinance (land mines) and with urban search and rescue (finding survivors after a terrorist bombing of a building or an earthquake). Humanitarian land demining is a challenging task. It is relatively easy to demine an area with bulldozer, but that destroys the fields and improvements made by the civilians and hurts the economy. Various types of robots are being tested in the field, including aerial and ground vehicles.[73]

1.3.1 Social implications of robotics

While many applications for artificially intelligent robots will actively reduce risk to a human life, many applications appear to compete with a human's livelihood. Don't robots put people out of work? One of the pervasive themes in society has been the impact of science and technology on the dignity of people. Charlie Chaplin's silent movie, *Modern Times*, presented the world with visual images of how manufacturing-oriented styles of management reduces humans to machines, just "cogs in the wheel."

Robots appear to amplify the tension between productivity and the role of the individual. Indeed, the scientist in *Metropolis* points out to the corporate ruler of the city that now that they have robots, they don't need workers anymore. People who object to robots, or technology in general, are often called *Luddites*, after Ned Ludd, who is often credited with leading a short-lived revolution of workers against mills in Britain. Prior to the industrial revolution in Britain, wool was woven by individuals in their homes or collectives as a cottage industry. Mechanization of the weaving process changed the jobs associated with weaving, the status of being a weaver (it was a skill), and required people to work in a centralized location (like having your telecommuting job terminated). Weavers attempted to organize and [illegible] escalating vi-

LUDDITES

olence in 1812, legislation was passed to end worker violence and protect the mills. The rebelling workers were persecuted. While the Luddite movement may have been motivated by a quality-of-life debate, the term is often applied to anyone who objects to technology, or "progress," for any reason. The connotation is that Luddites have an irrational fear of technological progress.

The impact of robots is unclear, both what is the real story and how people interact with robots. The HelpMate Robotics, Inc. robots and janitorial robots appear to be competing with humans, but are filling a niche where it is hard to get human workers at any price. Cleaning office buildings is menial and boring, plus the hours are bad. One janitorial company has now invested in mobile robots through a Denver-based company, Continental Divide Robotics, citing a 90% yearly turnover in staff, even with profit sharing after two years. The Robotics Industries Association, a trade group, produces annual reports outlining the need for robotics, yet possibly the biggest robot money makers are in the entertainment and toy industries.

The cultural implications of robotics cannot be ignored. While the sheep shearing robots in Australia were successful and were ready to be commercialized for significant economic gains, the sheep industry reportedly rejected the robots. One story goes that the sheep ranchers would not accept a robot shearer unless it had a 0% fatality rate (it's apparently fairly easy to nick an artery on a squirming sheep). But human shearers accidently kill several sheep, while the robots had a demonstrably better rate. The use of machines raises an ethical question: is it acceptable for an animal to die at the hands of a machine rather than a person? What if a robot was performing a piece of intricate surgery on a human?

1.4 A Brief History of Robotics

Robotics has its roots in a variety of sources, including the way machines are controlled and the need to perform tasks that put human workers at risk.

In 1942, the United States embarked on a top secret project, called the Manhattan Project, to build a nuclear bomb. The theory for the nuclear bomb had existed for a number of years in academic circles. Many military leaders of both sides of World War II believed the winner would be the side who could build the first nuclear device: the Allied Powers led by USA or the Axis, led by Nazi Germany.

One of the first problems that the scientists and engineers encountered was handling and processing radioactive materials, including uranium and

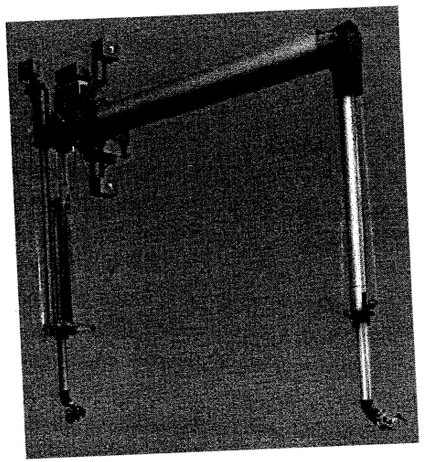


Figure 1.2 A Model 8 Telemanipulator. The upper portion of the device is placed in the ceiling, and the portion on the right extends into the hot cell. (Photograph courtesy Central Research Laboratories.)

plutonium, in large quantities. Although the immensity of the dangers of working with nuclear materials was not well understood at the time, all the personnel involved knew there were health risks. One of the first solutions was the *glove box*. Nuclear material was placed in a glass box. A person stood (or sat) behind a leaded glass shield and stuck their hands into thick rubberized gloves. This allowed the worker to see what they were doing and to perform almost any task that they could do without gloves.

But this was not an acceptable solution for highly radioactive materials, and mechanisms to physically remove and completely isolate the nuclear materials from humans had to be developed. One such mechanism was a force reflecting *telemanipulator*, a sophisticated mechanical linkage which translated motions on one end of the mechanism to motions at the other end. A popular telemanipulator is shown in Fig. 1.2.

:MANIPULATOR

A nuclear worker would insert their hands into (or around) the telemanipulator, and move it around while watching a display of what the other end of the arm was doing in a containment cell. Telemanipulators are similar in principle to the power gloves now used in computer games, but much

[illegible] technology of the time did not allow a perfect

tor had to make non-intuitive and awkward motions with their arms to get the robot arm to perform a critical manipulation—very much like working in front of a mirror. Likewise, the telemanipulators had challenges in providing force feedback so the operator could feel how hard the gripper was holding an object. The lack of naturalness in controlling the arm (now referred to as a poor Human-Machine Interface) meant that even simple tasks for an unencumbered human could take much longer. Operators might take years of practice to reach the point where they could do a task with a telemanipulator as quickly as they could do it directly.

After World War II, many other countries became interested in producing a nuclear weapon and in exploiting nuclear energy as a replacement for fossil fuels in power plants. The USA and Soviet Union also entered into a nuclear arms race. The need to mass-produce nuclear weapons and to support peaceful uses of nuclear energy kept pressure on engineers to design robot arms which would be easier to control than telemanipulators. Machines that looked more like and acted like robots began to emerge, largely due to advances in control theory. After WWII, pioneering work by Norbert Wiener allowed engineers to accurately control mechanical and electrical devices using cybernetics.

1.4.1 Industrial manipulators

Successes with at least partially automating the nuclear industry also meant the technology was available for other applications, especially general manufacturing. Robot arms began being introduced to industries in 1956 by Unimation (although it wouldn't be until 1972 before the company made a profit).[37] The two most common types of robot technology that have evolved for industrial use are robot arms, called industrial manipulators, and mobile carts, called automated guided vehicles (AGVs).

INDUSTRIAL MANIPULATOR

An *industrial manipulator*, to paraphrase the Robot Institute of America's definition, is a reprogrammable and multi-functional mechanism that is designed to move materials, parts, tools, or specialized devices. The emphasis in industrial manipulator design is being able to program them to be able to perform a task repeatedly with a high degree of accuracy and speed. In order to be multi-functional, many manipulators have multiple degrees of freedom, as shown in Fig. 1.4. The MOVEMASTER arm has five degrees of freedom, because it has five joints, each of which is capable of a single

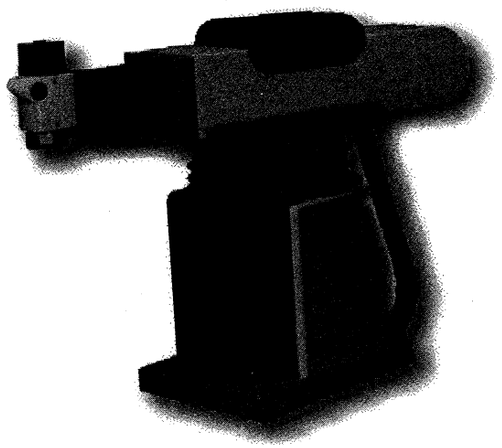


Figure 1.3 An RT3300 industrial manipulator. (Photograph courtesy of Seiko Instruments.)

bow, and wrist), two of which are complex (shoulder and wrist), yielding six degrees of freedom.

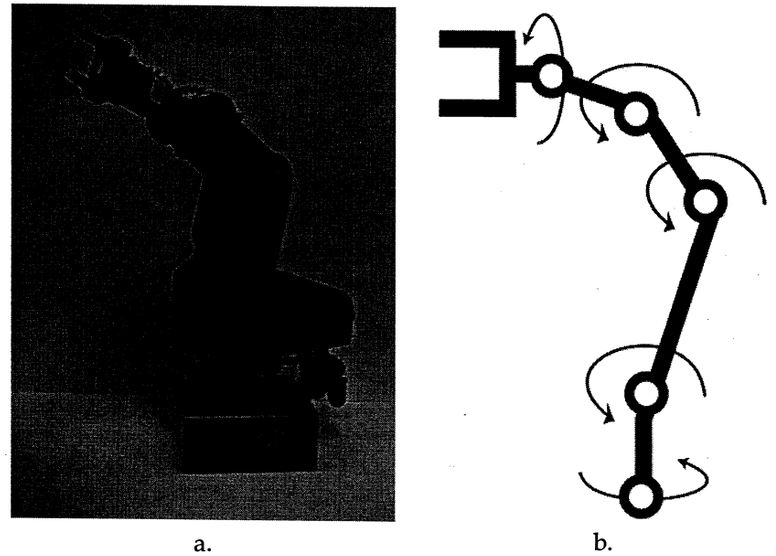
Control theory is extremely important in industrial manipulators. Rapidly moving around a large tool like a welding gun introduces interesting problems, like when to start decelerating so the gun will stop in the correct location without overshooting and colliding with the part to be welded. Also, oscillatory motion, in general, is undesirable. Another interesting problem is the joint configuration. If a robot arm has a wrist, elbow and shoulder joints like a human, there are redundant degrees of freedom. Redundant degrees of freedom means there are multiple ways of moving the joints that will accomplish the same motion. Which one is better, more efficient, less stressful on the mechanisms?

BALLISTIC CONTROL
OPEN LOOP CONTROL

It is interesting to note that most manipulator control was assumed to be *ballistic control*, or *open loop control*. In ballistic control, the position trajectory and velocity profile is computed once, then the arm carries it out. There are no "in-flight" corrections, just like a ballistic missile doesn't make any course corrections. In order to accomplish a precise task with ballistic control, everything about the device and how it works has to be modeled and figured into the computation. The opposite of ballistic control is *closed-loop control*,

CLOSED LOOP

and current position is noted by a sensor(s),



a. b.

Figure 1.4 A MOVEMASTER robot: a.) the robot arm and b.) the associated joints.

and a new trajectory and profile is computed and executed, then modified on the next update, and so on. Closed-loop control requires external sensors to provide the error signal, or *feedback*.

FEEDBACK

In general, if the structural properties of the robot and its cargo are known, these questions can be answered and a program can be developed. In practice, the control theory is complex. The dynamics (how the mechanism moves and deforms) and kinematics (how the components of the mechanism are connected) of the system have to be computed for each joint of the robot, then those motions can be propagated to the next joint iteratively. This requires a computationally consuming change of coordinate systems from one joint to the next. To move the gripper in Fig 1.4 requires four changes of coordinates to go from the base of the arm to the gripper. The coordinate transformations often have singularities, causing the equations to perform divide by zeros. It can take a programmer weeks to reprogram a manipulator.

One simplifying solution is to make the robot rigid at the desired velocities, reducing the dynamics. This eliminates having to compute the terms for overshooting and oscillating. However, a robot is made rigid by making it

H PENDANT

heavier. The end result is that it is not uncommon for a 2 ton robot to be able to handle only a 200 pound payload. Another simplifying solution is to avoid the computations in the dynamics and kinematics and instead have the programmer use a teach pendant. Using a *teach pendant* (which often looks like a joystick or computer game console), the programmer guides the robot through the desired set of motions. The robot remembers these motions and creates a program from them. Teach pendants do not mitigate the danger of working around a 2 ton piece of equipment. Many programmers have to direct the robot to perform delicate tasks, and have to get physically close to the robot in order to see what the robot should do next. This puts the programmer at risk of being hit by the robot should it hit a singularity point in its joint configuration or if the programmer makes a mistake in directing a motion. You don't want to have your head next to a 2 ton robot arm if it suddenly spins around!

MATIC GUIDED VEHICLES

Automatic guided vehicles, or AGVs, are intended to be the most flexible conveyor system possible: a conveyor which doesn't need a continuous belt or roller table. Ideally an AGV would be able to pick up a bin of parts or manufactured items and deliver them as needed. For example, an AGV might receive a bin containing an assembled engine. It could then deliver it automatically across the shop floor to the car assembly area which needed an engine. As it returned, it might be diverted by the central computer and instructed to pick up a defective part and take it to another area of the shop for reworking.

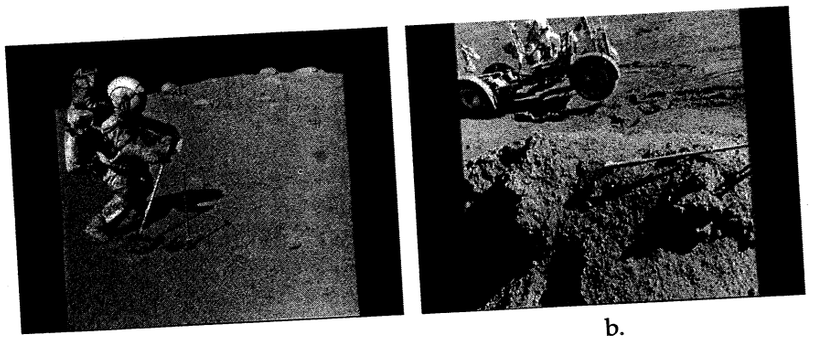
However, navigation (as will be seen in Part II) is complex. The AGV has to know where it is, plan a path from its current location to its goal destination, and to avoid colliding with people, other AGVs, and maintenance workers and tools cluttering the factory floor. This proved too difficult to do, especially for factories with uneven lighting (which interferes with vision) and lots of metal (which interferes with radio controllers and on-board radar and sonar). Various solutions converged on creating a trail for the AGV to follow. One method is to bury a magnetic wire in the floor for the AGV to sense. Unfortunately, changing the path of an AGV required ripping up the concrete floor. This didn't help with the flexibility needs of modern manufacturing. Another method is to put down a strip of photochemical tape for the vehicle to follow. The strip is unfortunately vulnerable, both to wear and to vandalism by unhappy workers. Regardless of the guidance method, in the end the simplest way to thwart an AGV was to something on its path. If the AGV did not have range sensors, then it would be unable to detect [illegible] of equipment or a person put deliberately in its path. A few costly collisions would usually led to the AGV's removal. If the AGV did have range sensors, it would stop for anything. A well placed lunch box could hold the AGV for hours until a manager happened to notice what was going on. Even better from a disgruntled worker's perspective, many AGVs would make a loud noise to indicate the path was blocked. Imagine having to constantly remove lunch boxes from the path of a dumb machine making unpleasant siren noises.

From the first, robots in the workplace triggered a backlash. Many of the human workers felt threatened by a potential loss of jobs, even though the jobs being mechanized were often menial or dangerous. This was particularly true of manufacturing facilities which were unionized. One engineer reported that on the first day it was used in a hospital, a HelpMate Robotics cart was discovered pushed down the stairs. Future models were modified to have some mechanisms to prevent malicious acts.

BLACK FACTORY

Despite the emerging Luddite effect, industrial engineers in each of the economic powers began working for a *black factory* in the 1980's. A black factory is a factory that has no lights turned on because there are no workers. Computers and robots were expected to allow complete automation of manufacturing processes, and courses in "Computer-Integrated Manufacturing Systems" became popular in engineering schools.

But two unanticipated trends undermined industrial robots in a way that the Luddite movement could not. First, industrial engineers did not have experience designing manufacturing plants with robots. Often industrial manipulators were applied to the wrong application. One of the most embarrassing examples was the IBM Lexington printer plant. The plant was built with a high degree of automation, and the designers wrote numerous articles on the exotic robot technology they had cleverly designed. Unfortunately, IBM had grossly over-estimated the market for printers and the plant sat mostly idle at a loss. While the plant's failure wasn't the fault of robotics, per se, it did cause many manufacturers to have a negative view of automation in general. The second trend was the changing world economy. Customers were demanding "mass customization." Manufacturers who could make short runs of a product tailored to each customer on a large scale were the ones making the money. (Mass customization is also referred to as "agile manufacturing.") However, the lack of adaptability and difficulties in programming industrial robot arms and changing the paths of AGVs interfered with rapid retooling. The lack of adaptability, combined with concerns over worker safety and the Luddite effect, served to discourage companies from investing in robots through most of the 1990's.



a. b.

Figure 1.5 Motivation for intelligent planetary rovers: a.) Astronaut John Young awkwardly collecting lunar samples on Apollo 16, and b.) Astronaut Jim Irwin stopping the lunar rover as it slides down a hill on Apollo 15. (Photographs courtesy of the National Aeronautics and Space Administration.)

1.4.2 Space robotics and the AI approach

While the rise of industrial manipulators and the engineering approach to robotics can in some measure be traced to the nuclear arms race, the rise of the AI approach can be said to start with the space race. On May 25, 1961, spurred by the success of the Soviet Union's Sputnik space programs, President John F. Kennedy announced that United States would put a man on the moon by 1970. Walking on the moon was just one aspect of space exploration. There were concerns about the Soviets setting up military bases on the Moon and Mars and economic exploitation of planetary resources.

Clearly there was going to be a time lag of almost a decade before humans from the USA would go to the Moon. And even then, it would most likely be with experimental spacecraft, posing a risk to the human astronauts. Even without the risk to humans, the bulk of spacesuits would make even trivial tasks difficult for astronauts to perform. Fig. 1.5a shows astronaut John Young on Apollo 16 collecting samples with a lunar rake. The photo shows the awkward way the astronaut had to bend his body and arms to complete the task.

Planetary rovers were a possible solution, either to replace an astronaut or assist him or her. Unfortunately, rover technology in the 1960's was limited. Because of the time delays, a human would be unable to safely control a rover over the notoriously poor radio links of the time, even if the rover went very

slow. Therefore, it would be desirable to have a robot that was autonomous. One option would be to have mobile robots land on a planetary conduct preliminary explorations, conduct tests, etc., and radio back the results. These automated planetary rovers would ideally have a high degree of autonomy, much like a trained dog. The robot would receive commands from Earth to explore a particular region. It would navigate around boulders and not fall into canyons, and traverse steep slopes without rolling over. The robot might even be smart enough to regulate its own energy supply, for example, by making sure it was sheltered during the planetary nights and to stop what it was doing and position itself for recharging its solar batteries. A human might even be able to speak to it in a normal way to give it commands.

Getting a mobile robot to the level of a trained dog immediately presented new issues. Just by moving around, a mobile robot could change the world-for instance, by causing a rock slide. Fig. 1.5b shows astronaut Jim Irwin rescuing the lunar rover during an extra-vehicular activity (EVA) on Apollo 15 as it begins to slide downhill. Consider that if an astronaut has difficulty finding a safe parking spot on the moon, how much more challenging it would be for an autonomous rover. Furthermore, an autonomous rover would have no one to rescue it, should it make a mistake.

Consider the impact of uncertain or incomplete information on a rover that didn't have intelligence. If the robot was moving based on a map taken from a telescope or an overhead command module, the map could still contain errors or at the wrong resolution to see certain dangers. In order to navigate successfully, the robot has to compute its path with the new data or risk colliding with a rock or falling into a hole. What if the robot did something broke totally unexpected or all the assumptions about the planet were wrong? In theory, the robot should be able to diagnose the problem and attempt to continue to make progress on its task. What seemed at first like an interim solution to putting humans in space quickly became more complicated.

Clearly, developing a planetary rover and other robots for space was going to require a concentrated, long-term effort. Agencies in the USA such as NASA *Jet Propulsion Laboratory (JPL)* in Pasadena, California, were given the task of developing the robotic technology that would be needed to prepare the way for astronauts in space. They were in a position to take advantage of the outcome of the *Dartmouth Conference*. The Dartmouth Conference was a gathering hosted by the Defense Advanced Research Projects Agency (DARPA) in 1955 of prominent scientists working with computers or on the theory for computers. DARPA was interested in hearing what the potential

uses for computers were. One outcome of the conference was the term "artificial intelligence"; the attending scientists believed that computers might become powerful enough to understand human speech and duplicate human reasoning. This in turn suggested that computers might mimic the capabilities of animals and humans sufficiently for a planetary rover to survive for long periods with only simple instructions from Earth.

As an indirect result of the need for robotics converging with the possibility of artificial intelligence, the space program became one of the earliest proponents of developing AI for robotics. NASA also introduced the notion that AI robots would of course be mobile, rather than strapped to a factory floor, and would have to integrate all forms of AI (understanding speech, planning, reasoning, representing the world, learning) into one program—a daunting task which has not yet been reached.

1.5 Teleoperation

TELEOPERATION

Teleoperation is when a human operator controls a robot from a distance (*tele* means "remote"). The connotation of teleoperation is that the distance is too great for the operator to see what the robot is doing, so radio controlled toy cars are not considered teleoperation systems. The operator and robot have some type of master-slave relationship. In most cases, the human operator sits at a workstation and directs a robot through some sort of interface, as seen in Fig. 1.6.

The control interface could be a joystick, virtual reality gear, or any number of innovative interfaces. The human operator, or teleoperator, is often referred to as the *local* (due to being at the local workstation) and the robot as the *remote* (since it is operating at a remote location from the teleoperator). The local must have some type of display and control mechanisms, while the remote must have sensors, effectors, power, and in the case of mobile robots, mobility.[141] The teleoperator cannot look at what the remote is doing directly, either because the robot is physically remote (e.g., on Mars) or the local has to be shielded (e.g., in a nuclear or pharmaceutical processing plant hot cell). Therefore, the *sensors* which acquire information about the remote location, the *display* technology for allowing the operator to see the sensor data, and the *communication link* between the local and remote are critical components of a telesystem.[141]

LOCAL
REMOTE

SENSORS
DISPLAY
MMUNICATION LINK

Teleoperation is a popular solution for controlling remotes because AI technology is nowhere near human levels of competence, especially in terms of

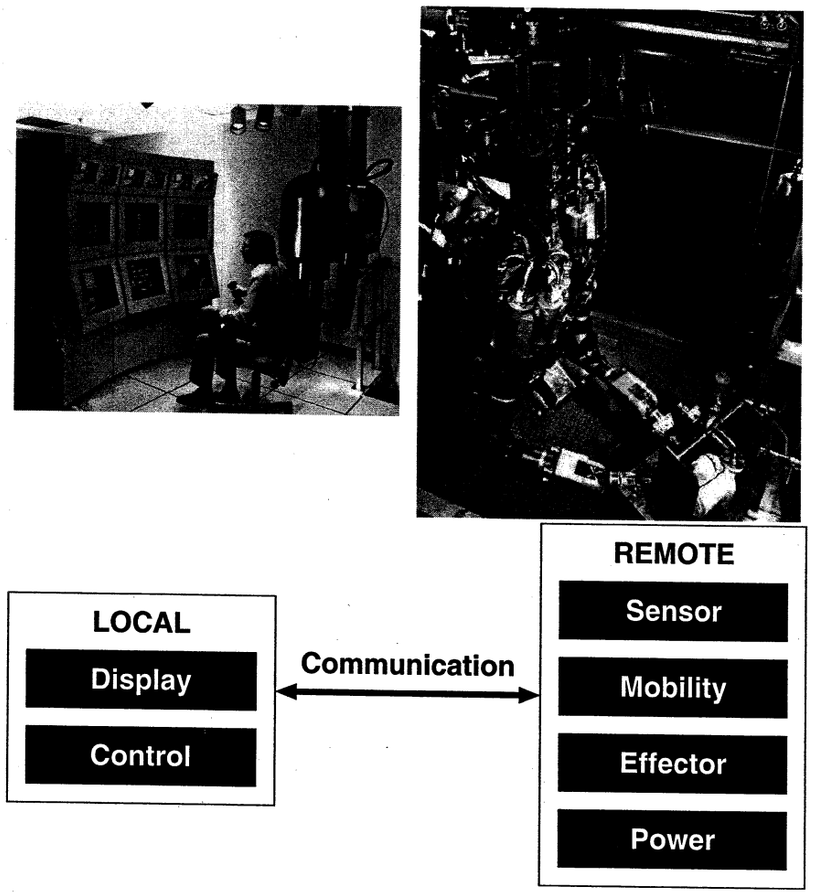


Figure 1.6 Organization of a telesystem. (Photographs courtesy of Oak Ridge National Laboratory.)

perception and decision making. One example of teleoperation is the exploration of underwater sites such as the Titanic. Having a human control a robot is advantageous because a human can isolate an object of interest, even partially obscured by mud in murky water as described by W. R. Uttal.[141] Humans can also perform dextrous manipulation (e.g., screwing a nut on a bolt), which is very difficult to program a manipulator to do.

Figure 1.7 Sojourner Mars rover. (Photograph courtesy of the National Aeronautics and Space Administration.)

Another example is the Sojourner robot (shown in Fig. 1.7) which explored Mars from July 5 to September 27, 1997, until it ceased to reply to radio commands. Since there was little data before Sojourner on what Mars is like, it is hard to develop sensors and algorithms which can detect important attributes or even control algorithms to move the robot. It is important that any unusual rocks or rock formations (like the orange rock Dr. Schmitt found on the Moon during Apollo 17) be detected. Humans are particularly adept at perception, especially seeing patterns and anomalies in pictures. Current AI perceptual abilities fall far short of human abilities. Humans are also adept at problem solving. When the Mars Pathfinder craft landed on Mars, the air bags that had cushioned the landing did not deflate properly. When the petals of the lander opened, an airbag was in the way of Sojourner. The solution? The ground controllers sent commands to retract the petals and open them again. That type of problem solving is extremely difficult for the current capabilities of AI.

But teleoperation is not an ideal solution for all situations. Many tasks are repetitive and boring. For example, consider using a joystick to drive a radio-controlled car; after a few hours, it tends to get harder and harder to pay attention. Now imagine trying to control the car while only looking [illegible] front. The task becomes much harder

because of the limited field of view; essentially there is no peripheral vision. Also, the camera may not be transmitting new images very fast because the communication link has a limited bandwidth, so the view is jerky. Most people quickly experience *cognitive fatigue*; their attention wanders and they may even experience headaches and other physical symptoms of stress. Even if the visual display is excellent, the teleoperator may get *simulator sickness* due to the discordance between the visual system saying the operator is moving and the inner ear saying the operator is stationary.[141]

COGNITIVE FATIGUE

SIMULATOR SICKNESS

Another disadvantage of teleoperation is that it can be inefficient to use for applications that have a *large time delay*.[128] A large time delay can result in the teleoperator giving a remote a command, unaware that it will place the remote in jeopardy. Or, an unanticipated event such as a rock fall might occur and destroy the robot before the teleoperator can see the event and command the robot to flee. A rule of thumb, or *heuristic*, is that the time it takes to do a task with traditional teleoperation grows linearly with the transmission delay. A teleoperation task which took 1 minute for a teleoperator to guide a remote to do on the Earth might take 2.5 minutes to do on the Moon, and 140 minutes on Mars.[142] Fortunately, researchers have made some progress with *predictive displays*, which immediately display what the simulation result of the command would be.

TELEOPERATION HEURISTIC

PREDICTIVE DISPLAYS

The impact of time delays is not limited to planetary rovers. A recent example of an application of teleoperation are unmanned aerial vehicles (UAV) used by the United States to verify treaties by flying overhead and taking videos of the ground below. Advanced prototypes of these vehicles can fly autonomously, but take-offs and landings are difficult for on-board computer control. In this case of the Darkstar UAV (shown in Fig. 1.8), human operators were available to assume teleoperation control of the vehicle should it encounter problems during take-off. Unfortunately, the contingency plan did not factor in the 7 second delay introduced by using a satellite as the communications link. Darkstar no. 1 did indeed experience problems on take-off, but the teleoperator could not get commands to it fast enough before it crashed. As a result, it earned the unofficial nickname "Darkspot."

Another practical drawback to teleoperation is that there is at least one person per robot, possibly more. The Predator unmanned aerial vehicle has been used by the United States for verification of the Dayton Accords in Bosnia. One Predator requires at least one teleoperator to fly the vehicle and another teleoperator to command the sensor payload to look at particular areas. Other UAVs have teams composed of up to four teleoperators plus a fifth team member who specializes in takeoffs and landings. These teleop-

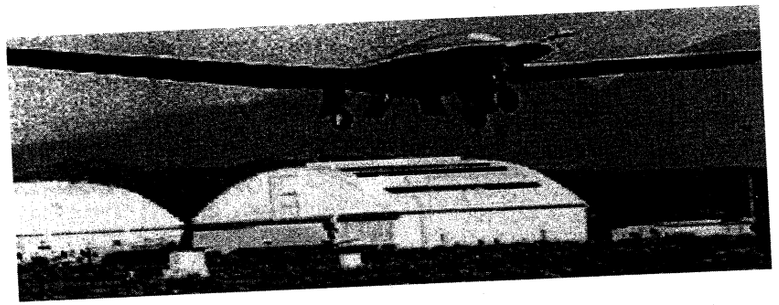


Figure 1.8 Dark Star unmanned aerial vehicle. (Photograph courtesy of DefenseLink, Office of the Assistant Secretary of Defense-Public Affairs.)

erators may have over a year of training before they can fly the vehicle. In the case of UAVs, teleoperation permits a dangerous, important task to be completed, but with a high cost in manpower.

TASK CHARACTERISTICS

According to Wampler,[142] teleoperation is best suited for applications where:

1. The tasks are unstructured and not repetitive.
2. The task workspace cannot be engineered to permit the use of industrial manipulators.
3. Key portions of the task intermittently require dextrous manipulation, especially hand-eye coordination.
4. Key portions of the task require object recognition, situational awareness, or other advanced perception.
5. The needs of the display technology do not exceed the limitations of the communication link (bandwidth, time delays).
6. The availability of trained personnel is not an issue.

1.5.1 Telepresence

An early attempt at reducing cognitive fatigue was to add more cameras with faster update rates to widen the field of view and make it more consistent with how a human prefers to look at the world. This may not be practical

for many applications because of limited bandwidth. Video telephones, picture phones, or video-conferencing over the Internet with their jerky, asynchronous updates are usually examples of annoying limited bandwidth. In these instances, the physical restrictions on how much and how fast information can be transmitted result in image updates much slower than the rates human brains expect. The result of limited bandwidth is jerky motion and increased cognitive fatigue. So adding more cameras only exacerbates the problem by adding more information that must be transmitted over limited bandwidth.

TELEPRESENCE

VIRTUAL REALITY

One area of current research in teleoperation is the use of *telepresence* to reduce cognitive fatigue and simulator sickness by making the human-robot interface more natural. Telepresence aims for what is popularly called *virtual reality*, where the operator has complete sensor feedback and feels as if she were the robot. If the operator turns to look in a certain direction, the view from the robot is there. If the operator pushes on a joystick for the robot to move forward and the wheels are slipping, the operator would hear and feel the motors straining while seeing that there was no visual change. This provides a more natural interface to the human, but it is very expensive in terms of equipment and requires very high bandwidth rates. It also still requires one person per robot. This is better than traditional teleoperation, but a long way from having one teleoperator control multiple robots.

1.5.2 Semi-autonomous control

SEMI-AUTONOMOUS CONTROL

SUPERVISORY CONTROL

Another line of research in teleoperation is *semi-autonomous control*, often called *supervisory control*, where the remote is given an instruction or portion of a task that it can safely do on its own. There are two flavors of semi-autonomous control: continuous assistance, or *shared control*, and *control trading*.

SHARED CONTROL

In continuous assistance systems, the teleoperator and remote share control. The teleoperator can either delegate a task for the robot to do or can do it via direct control. If the teleoperator delegates the task to the robot, the human must still monitor to make sure that nothing goes wrong. This is particularly useful for teleoperating robot arms in space. The operator can relax (relatively) while the robot arm moves into the specified position near a panel, staying on alert in case something goes wrong. Then the operator can take over and perform the actions which require hand-eye coordination. Shared control helps the operator avoid cognitive fatigue by delegating boring, repetitive control actions to the robot. It also exploits the ability of a

human to perform delicate operations. However, it still requires a high communication bandwidth.

)L TRADING

An alternative approach is *control trading*, where the human initiates an action for the robot to complete autonomously. The human only interacts with the robot to give it a new command or to interrupt it and change its orders. The overall scheme is very much like a parent giving a 10-year old child a task to do. The parent knows what the child is able to do autonomously (e.g., clean their room). They have a common definition (clean room means go to the bedroom, make the bed, and empty the wastebaskets). The parent doesn't care about the details of how the child cleans the room (e.g., whether the wastebasket is emptied before the bed is made or vice versa). Control trading assumes that the robot is capable of autonomously accomplishing certain tasks without sharing control. The advantage is that, in theory, the local operator can give a robot a task to do, then turn attention to another robot and delegate a task to it, etc. A single operator could control multiple robots because they would not require even casual monitoring while they were performing a task. Supervisory control also reduces the demand on bandwidth and problems with communication delays. Data such as video images need to be transferred only when the local is configuring the remote for a new task, not all the time. Likewise, since the operator is not involved in directly controlling the robot, a 2.5 minute delay in communication is irrelevant; the robot either wrecked itself or it didn't. Unfortunately, control trading assumes that robots have actions that they can perform robustly even in unexpected situations; this may or may not be true. Which brings us back to the need for artificial intelligence.

Sojourner exhibited both flavors of supervisory control. It was primarily programmed for traded control, where the geologists could click on a rock and Sojourner would autonomously navigate close to it, avoiding rocks, etc. However, some JPL employees noted that the geologists tended to prefer to use shared control, watching every movement. A difficulty with most forms of shared control is that it is assumed that the human is smarter than the robot. This may be true, but the remote may have better sensor viewpoints and reaction times.

1.6 The Seven Areas of AI

Now that some possible uses and shortcomings of robots have been covered, ... of artificial intelligence and how they could be used to overcome these problems. The Handbook of Artificial Intelligence[64] divides up the field into seven main areas: *knowledge representation, understanding natural language, learning, planning and problem solving, inference, search,* and *vision.*

KNOWLEDGE REPRESENTATION

1. **Knowledge representation**. An important, but often overlooked, issue is how does the robot represent its world, its task, and itself. Suppose a robot is scanning a pile of rubble for a human. What kind of data structure and algorithms would it take to represent what a human looks like? One way is to construct a structural model: a person is composed of an oval head, a cylindrical torso, smaller cylindrical arms with bilateral symmetry, etc. Of course, what happens if only a portion of the human is visible?

UNDERSTANDING NATURAL LANGUAGE

2. **Understanding natural language**. Natural language is deceptively challenging, apart from the issue of recognizing words which is now being done by commercial products such as Via Voice and Naturally Speaking. It is not just a matter of looking up words, which is the subject of the following apocryphal story about AI. The story goes that after Sputnik went up, the US government needed to catch up with the Soviet scientists. However, translating Russian scientific articles was time consuming and not many US citizens could read technical reports in Russian. Therefore, the US decided to use these newfangled computers to create translation programs. The day came when the new program was ready for its first test. It was given the proverb: the spirit is willing, but the flesh is weak. The reported output: the vodka is strong, but the meat is rotten.

LEARNING

3. **Learning.** Imagine a robot that could be programmed by just watching a human, or by just trying the task repeatedly itself.

PLANNING, PROBLEM SOLVING

4. **Planning and problem solving.** Intelligence is associated with the ability to plan actions needed to accomplish a goal and solve problems with those plans or when they don't work. One of the earliest childhood fables, the Three Pigs and the Big, Bad Wolf, involves two unintelligent pigs who don't plan ahead and an intelligent pig who is able to solve the problem of why his brothers' houses have failed, as well as plan an unpleasant demise for the wolf.

INFERENCE

5. **Inference**. Inference is generating an answer when there isn't complete information. Consider a planetary rover looking at a dark region on the ground. Its range finder is broken and all it has left is its camera and a fine AI system. Assume that depth information can't be extracted from

the camera. Is the dark region a canyon? Is it a shadow? The rover will need to use inference to either actively or passively disambiguate what the dark region is (e.g., kick a rock at the dark area versus reason that there is nothing nearby that could create that shadow).

SEARCH

6. **Search.** Search doesn't necessarily mean searching a large physical space for an object. In AI terms, search means efficiently examining a knowledge representation of a problem (called a "search space") to find the answer. Deep Blue, the computer that beat the World Chess master Gary Kasparov, won by searching through almost all possible combinations of moves to find the best move to make. The legal moves in chess given the current state of the board formed the search space.

VISION

7. **Vision.** Vision is possibly the most valuable sense humans have. Studies by Harvard psychologist Steven Kosslyn suggest that much of problem solving abilities stem from the ability to visually simulate the effects of actions in our head. As such, AI researchers have pursued creating vision systems both to improve robotic actions and to supplement other work in general machine intelligence.

Finally, there is a temptation to assume that the history of AI Robotics is the story of how advances in AI have improved robotics. But that is not the case. In many regards, robotics has played a pivotal role in advancing AI. Breakthroughs in methods for planning (operations research types of problems) came after the paradigm shift to reactivity in robotics in the late 1980's showed how unpredictable changes in the environment could actually be exploited to simplify programming. Many of the search engines on the world wide web use techniques developed for robotics. These programs are called *software agents*: autonomous programs which can interact with and adapt to their world just like an animal or a smart robot. The term *web-bot* directly reflects on the robotic heritage of these AI systems. Even animation is being changed by advances in AI robotics. According to a keynote address given by Danny Hillis at the 1997 Autonomous Agents conference, animators for Disney's *Hunchback of Notre Dame* programmed each cartoon character in the crowd scenes as if it were a simulation of a robot, and used methods that will

SOFTWARE AGENTS
WEB-BOT

1.7 Summary

AI robotics is a distinct field, both historically and in scope, from industrial robotics. Industrial robots has concentrated on control theory issues, particularly solving the dynamics and kinematics of a robot. This is concerned with having the stationary robot perform precise motions, repetitively in a structured factory environment. AI robotics has concentrated on how a mobile robot should handle unpredictable events in an unstructured world. The design of an AI robot should consider how the robot will represent knowledge about the world, whether it needs to understand natural language, can it learn tasks, what kind of planning and problem solving will it have to do, how much inference is expected, how can it rapidly search its database and knowledge for answers, and what mechanisms will it use for perceiving the world.

Teleoperation arose as an intermediate solution to tasks that required automation but for which robots could not be adequately programmed to handle. Teleoperation methods typically are cognitive fatiguing, require high communication bandwidths and short communication delays, and require one or more teleoperators per remote. Telepresence techniques attempt to create a more natural interface for the human to control the robot and interpret what it is doing and seeing, but at a high communication cost. Supervisory control attempts to delegate portions of the task to the remote, either to do autonomously (traded control) or with reduced, but continuous, human interaction (shared control).

1.8 Exercises

Exercise 1.1

List the four attributes for evaluating an architecture. Based on what you know from your own experience, evaluate MS Windows 95/98/2000 as an architecture for teleoperating a robot.

Exercise 1.2

Name the three primitives for expressing the components of a robotics paradigm.

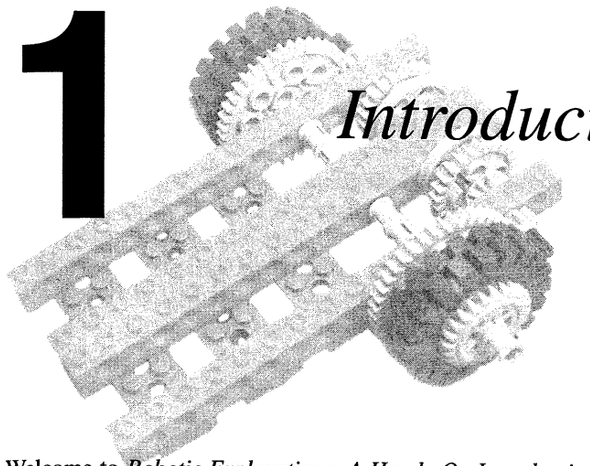
Exercise 1.3

Name the three robotic paradigms, and draw the relationship between the primitives.

Exercise 1.4

What is an intelligent robot?

1 Introduction



Welcome to *Robotic Explorations: A Hands-On Introduction to Engineering*. On the surface, this text is about the practice, technique, and theory behind mobile robotic systems, but it is different from most texts because you have to build a robot to learn about how they work. In doing so, you learn not only about robots, but about technological systems of all kinds—electrical, mechanical, and computational. More important, you engage in engineering design and problem solving as you work on activities and projects inspired by your own ideas.

Soon enough, we'll have the soldering irons, motors, gears, and chips out to play with, but to get started, let's take a quick tour through years of thinking about *feedback control*—a way of understanding systems that is of central importance to our explorations of robotics.

Next in this introduction, we examine an entirely different sort of history—a collection of toys. These toys are of strong personal meaning because they are toys I owned as a child. But they are unusual toys, and relevant to our discussion, because they were designed to let young learners explore key ideas in technology like process, program, and algorithm.

The "toy stories" lead into a discussion of some more advanced toys, including some designed at the Massachusetts Institute of Technology (MIT) beginning in the 1960s. Like the toys I played with as a child, these MIT toys were designed to help kids learn.

That is what this book is about: a yet more sophisticated set of toys (call it educational technology if you like) that is specifically designed to lead students into thought-provoking and challenging engineering situations. In other words, while the set of toys you will be using alongside this text will let you build mobile robots, the real work is in your hands, thinking hard about issues and problems in the design of *your* robot.

Thus, this text is part reference, part project guide, part philosophy, and part theory. You will design and build a mobile robot, but, more important, you will learn about engineering design and the process of invention because, although all of the pieces are

ion

of Feedback Control, the use of feedback in engineered systems is traced from ancient times to the modern world [May70].

Originally conceived in ancient Greece and built into time-measuring devices, the idea of feedback control was forgotten and had to be rediscovered in Renaissance Europe. By way of introduction to the idea of feedback, let's briefly examine its progression from these ancient origins to the present day.

Some of the earliest inventions that incorporated feedback control were water clocks. To produce a constant flow of water, a float valve regulated the amount of water in a large holding tank. This body of water applied a constant pressure through a precise orifice, resulting in an even flow of water. Various mechanisms measured this water flow, which translated into a measure of elapsed time.

The float valve worked by detecting the level of water in a tank and controlling the amount of water that would flow into the tank. When the water fell below a certain level, a valve was opened, allowing water to fill the holding tank. Essentially, these early water clocks—ingenious devices of their time—employed the same mechanism now commonly found in the household toilet.

Float valves were then likely the first instance of a self-regulating device: that is, a mechanism that senses the quantity to be regulated and then takes an action that controls that same quantity. Without the float valve, there would have been no way of providing a steady flow of water, and the clocks would not have been able to keep accurate enough time to be of any use.

The feedback concept embedded in the early water clocks progressed into water clocks built in Arabic cultures until about 1200 A.D., but it did not spread to Europe. Instead, the idea of feedback control had to be rediscovered by Western inventors much later.

The first feedback system developed in modern Europe was a temperature-regulating oven invented by Cornelis Drebbel in the early 1600s. Drebbel was a prolific inventor; his other inventions included telescopes, microscopes, and a submarine. He worked on the regulated oven to help in his chemical experiments (which included pyrotechnic materials and a scarlet dye). Drebbel's son-in-law, the physician Johan Sibertus Kuffler, attempted to commercialize the heat-regulated furnace, but was largely unsuccessful.

Drebbel's furnace worked on a similar principle as the float valve water regulators. A chamber inside the furnace held a quantity of mercury; as the mercury expanded due to increasing heat inside the furnace, it rose in the chamber and caused a damper to close off the flow of air to the coals, reducing the amount of heat being generated.

Although Drebbel's work was unpublished, others wrote about his invention and spread the concept to other engineers. In the mid-1700s, the physicist Réaumur revived the heat-regulated furnace for the purpose of incubating chickens. Interestingly, much of the subsequent work done on heat regulation over the next 50 years was based on the problem of hatching chickens.

The turning point for temperature-regulation devices came with the work of the French [illegible] Bonnemain vastly improved on the mechanism poultry substantially lessened). Bonnemain's heat-regulated incubators were the first such devices that were true industrial equipment and not just laboratory demonstrations.

After the incubators/ovens, feedback control was designed into windmills and boilers. In windmills, the fan tail pointed the main wheel into the wind and regulated the pressure of the grindstones on the grain; in steam boilers, a feedback system controlled the pressure in the boiler unit. But it was as a speed regulator in the steam engine that feedback control achieved its breakthrough in terms of engineering effectiveness and public recognition.

At Boulton & Watt, a steam mill firm in London, James Watt, the chief engineer, adopted a centrifugal control mechanism (which he had seen on windmills) for the purpose of controlling the output engine speed of his firm's steam engines. The "governor," as he called it, accomplished this by measuring the actual speed of the engine and throttling the steam inlet valve of the engine, allowing more steam to enter if the speed had slowed down and less steam if the engine was going too fast. Figure 1.1 represents Watt's early drawings of the invention.

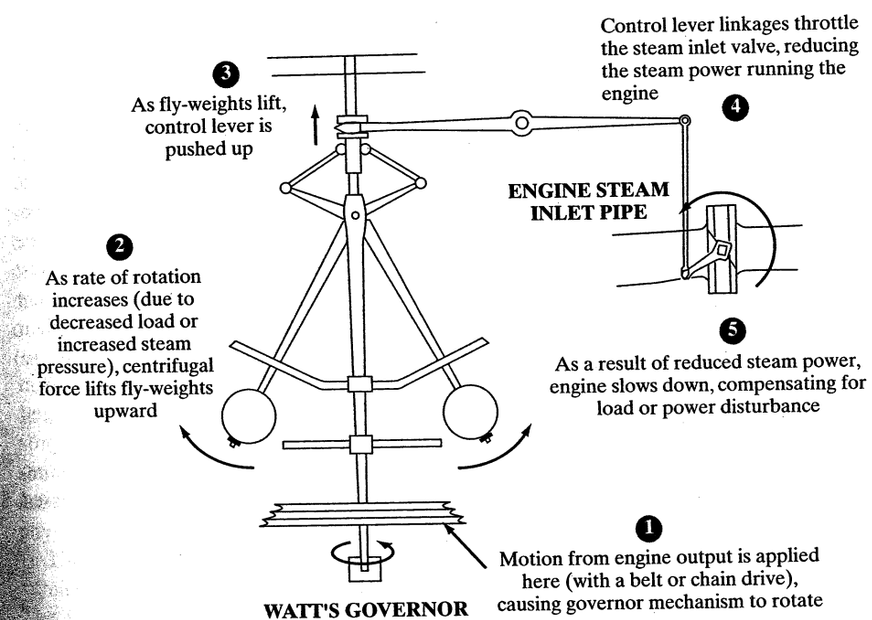


FIGURE 1.1 James Watt's Governor for Feedback Control of Steam Engine Speed

At the time, Watt thought he was applying a known principle to a new application; he did not take credit for having invented a new idea. Rather, he wrote that the

tion

of the Watt governor spread through public consciousness as the Industrial Revolution gained momentum; Watt's governor became iconified as the symbol of our collective ability to harness the power of technology.

Up until the 20th century, the technology of automatic control was a specialty of mechanical engineering. By the turn of the century, classical mathematics theory—i.e., differential equations—formalized the understanding of speed control. In the early 1900s, spurred by new applications in electrical engineering, new theory was developed. Scientists such as Laplace and Nyquist helped to create techniques that are still used today in the analysis of automatic control systems.

The next big surge in control engineering came during World War II, with research and development in munitions such as radar, guided missiles, and automatic pilots. After the war ended, the research became generally available to the larger scientific community. One of the leading war scientists, Norbert Wiener, turned his attention from using feedback controls in weaponry to thinking about its larger implications—specifically, the role of feedback in biological systems. Wiener's 1948 book, *Cybernetics: Control and Communication in the Animal and the Machine*, was a seminal contribution to modern thinking; fields of inquiry such as social sciences, psychology, biology, and engineering were affected by Wiener's work [Wie48].

Wiener coined the term *cybernetics*, representing the entire field of control and communication theory, from the Greek word $\chi\upsilon\beta\epsilon\rho\nu\acute{\eta}\tau\eta\varsigma$, meaning "steersman." This was to recognize the central importance of the centrifugal governor popularized in Watt's steam engines.

Wiener's work was part of a larger societal recognition of the role of feedback in social, economic, and biological systems in our world. In the late 1940s and early 1950s, W. Grey Walter, a British neurophysiologist, developed the first true "robotic animals"—mobile electronic robots specifically designed to mimic the behavior of living creatures [Wal51]. Walter's robots, built before the invention of the transistor, were powered by batteries and used vacuum tubes for control. With just six vacuum tubes, Walter's robots were able to perform behaviors like light-seeking (i.e., "tropism"), wandering about randomly, and finding their hutch to recharge their batteries.

To emphasize the creature-like characteristics of his robots, Walter called them "turtles" and gave them playful, pseudoscientific names like *Machina docilis* and *Machina speculatrix*. Walter's turtles later inspired Seymour Papert to call his programmable robots—designed for use by children—"turtles" and influenced Valentino Braitenberg's robot-like "vehicles." (The work of both of these people is discussed shortly.)

1.2 Toys to Think With

...childhood was in the 1960s and 1970s. During that time, there was great popular ... there

drive forward and backward and steer like a regular car. The difference was that this car was not directly controlled, but rather was *programmed* by placing pegs on a control wheel.

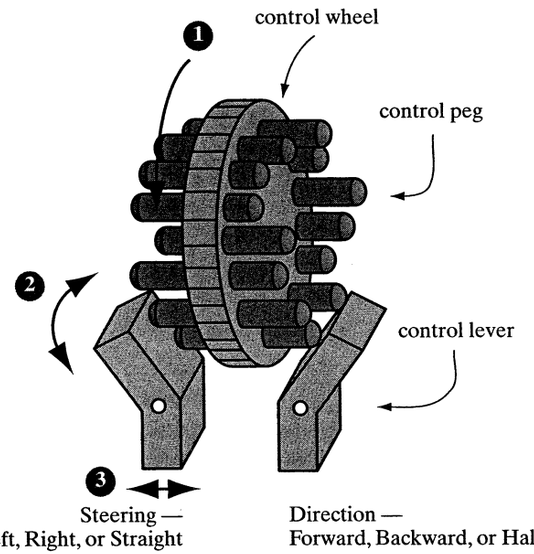


FIGURE 1.2 Control Mechanism from Mechanically Programmable Toy Car

The mechanism "under the hood" of the programmable toy car. The control wheel was turned (indicated by arrow 1) by a pinion gear (not shown). Control pegs, of varying lengths, sequentially pressed against control levers open (2), which were mechanically connected to steering and direction mechanisms (3).

The central control wheel was driven by a gear and slowly turned at a fixed rate. To program the car control pegs of three different lengths were placed onto the control wheel. As the wheel turned, the pegs would successively press against control levers. One control lever connected mechanically to the car's steering system, and by choice of peg the car would go straight, to the left, or to the right. The other control lever controlled the car's movement as either forward, backward, or halted. By coordinating the choice of steering and direction pegs, the car could be programmed to drive in a specific path (see Figure 1.2).

For very young children, the typical radio control car is a level of abstraction beyond direct body motion. The child manipulates a steering wheel that guides the movement of an artifact external to the child. But my programmable car encouraged thinking about movement in a yet more abstract way. Rather than directly controlling the car's movement, I programmed a series of actions in advance. This program had a tangible, physical instantiation in the pegs on the control wheel.

The mechanical car was not a big commercial success, but 10 years later an electronic version of the same idea—the Big Trak by Milton-Bradley—was immensely popular. The Big Trak, one of the first microprocessor-based toys, was a tank-like vehicle with a keypad for entering movement sequences. It was fundamentally the same concept as the mechanical car I had owned, with the programming done electronically rather than mechanically. The Big Trak is no longer manufactured, but the Roamer, an egg-shaped robot designed by Valiant Technology, has similar programmable movement capabilities.

a powerful idea—that of constructing a series of actions into a program—immediate and accessible.

The mechanically programmed car is my first recollection of a programmable toy, but it is by no means the last.

Whereas the programmable car was probably designed simply as a toy, the Digi-Comp Computer was created by computer scientists specifically as an educational device. The Digi-Comp was also a mechanically programmable toy, but rather than controlling the state of another device (like the steering of the car), Digi-Comp operated on its own internal state—as does a real computer. As proclaimed on the cover of its manual, the Digi-Comp was "the first real operating digital computer in plastic."

Comp

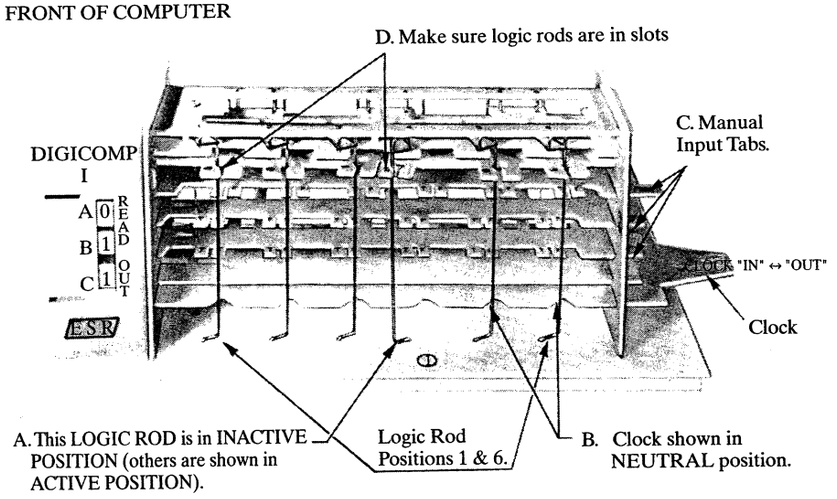


Figure 1.3 shows a photograph of the Digi-Comp device. The computer consisted of three sliding plates, each of which could be in one of two different positions (representing the "0" state and the "1" state). The user placed plastic tubes (not shown) onto tabs that jutted out from the plates. A clocking mechanism, activated by the user, caused a set of vertical bars to swing back and forth. When Digi-Comp was clocked, these bars knocked into the tubes and caused the plates to move.

The Digi-Comp was programmed by judicious placement of the tubes. As a child playing with it, I did not really understand how precisely the tubes caused the Digi-Comp to perform different functions, although it was clear to me that they were central. Looking over the instruction manual as a computer scientist, the mechanism is less mysterious, although it is still quite clever. Fundamentally, the tubes cause each plate of the Digi-Comp to act as a set-reset flip-flop.

Despite its humble plastic construction and arcane programming method, the Digi-Comp really was a computer. The examples in the manual, which I worked through as a 10-year-old, included Boolean operations like "and" and "or," binary counting, bit-shift operations, and other such classics.

The Digi-Comp was of course limited, but it was remarkable in what it accomplished with just plastic and a few metal rods. For me personally, it certainly served its design-

The TRS-80 computer was in the first generation of machines sold as "home computers," although of course they were really hobbyist playthings. I was fortunate to have owned the model with "Level II BASIC," which supported features like floating point numerics and an interface to assembly language routines. The machine came with only 4K of memory, which I later expanded to a generous 16K.

A big difference between these early computers and contemporary machines was that commercial productivity software, as it were, did not yet exist, so the main application of the early home computers was for their owners to explore how they worked. Essentially, every computer owner was also a computer programmer.

Another important difference was that these early computers were simple enough that a dedicated hobbyist could completely understand every layer of functionality that was built into them, from the CPU instruction set, to memory-mapped hardware and video display, to system software built into the machine's permanent memory. Today's machines, in terms of both their hardware and software designs, are far too complex to be thoroughly understood. As a pedagogical tool, the early computers had a distinct advantage.

When I first got the TRS-80, I began by learning the BASIC language; I then progressed to programming in the Z-80 machine language of the computer, learning about its hardware subsystems, and finally learning how to patch the operating system and various ways and directly call subroutines located in the computer's built-in operating program.

The set of projects I always wanted to work on, but did not have the expertise to do so at the time, was to interface the computer to the real world. That would have to wait until college. I knew that I wanted to do work in the field of robotics; as a sophomore, I was fortunate enough to join the research group of Warren Seering, who as a member of the Mechanical Engineering faculty had a research group located at MIT's Artificial Intelligence (AI) Laboratory.

Seering's research group was working on an experimental robot arm, a "Cartesian" manipulator with linear X, Y, and Z axes. It was a large, powerful robot arm designed for exploring new robotic manufacturing techniques. The robot was strong enough to be dangerous; it could easily crush a person's hand if it came between the robot's body and its limit stops.

My work involved writing assembly language code that served as the low-level feedback positioning controller. While I was working on this project, my coursework included the formal study of feedback control. It was a remarkable experience to be learning the theory behind feedback control while my code was controlling a large and potentially dangerous piece of equipment. The robot was bolted down to the concrete ninth floor of the building that housed the AI Lab; legend had it that when previous students' control code caused unstable oscillations of the arm, it literally shook the whole building. I soon discovered these stories to be accurate. Fortunately, the robot came with a large, red emergency stop button.

The reason for telling all of these personal learning stories is to give a sense of the type of learning experience I instinctively sought out as a young learner. The exercises and projects proposed in this book are ideally done in the same spirit of hands-on, engaged experimentation. When I became a master's student, I began working with Seymour Papert, a pioneer in the area of what he calls "constructionist learning"—the

Seymour Papert is a mathematician, computer scientist, and, in the 1960s, was co-director of MIT's AI Laboratory. Papert was a leader of the community of AI researchers during a period of intense creativity and intellectual excitement. AI researchers had developed the Lisp programming language and were engaged in writing programs to test, explore, and embody their ideas about the nature of human intelligence.

Papert wanted to bring this spirit of inquiry to the world of children. He believed that children weren't fluent with mathematics because of a lack of challenging and engaging "mathematical stuff" in the world of a child. In the late 1960s and early 1970s, Papert led the development of Logo, a programming language designed for children. Logo shared many ideas with the Lisp programming language used in the MIT research laboratories, like an interactive interface, but had a simplified syntax to make it easy to learn.

In *Mindstorms*, his seminal book on the experiences of children programming with Logo, Papert made an analogy between trying to learn a foreign language (say French) in the classroom versus learning it in a place where it is spoken (France). It's easy to learn to speak French while living in France because speaking the language is a natural, contextualized activity with real-life relevance. If you want to eat an ice cream cone, you must ask for it in French! In contrast, speaking in a classroom is based on role-play conversation, which cannot have the same personal relevance. Papert hoped to create an environment—the Logo programming language—where children could work with mathematical ideas that have the same personal meaning as does speaking French in France.

As the Logo language was developed, it came to have at least two characteristics that distinguished it from other contemporary computer programming environments. The first was its interactivity, which it shared with Lisp, the language on which Logo was based. When children were sitting in front of a Logo console, they could type a Logo command and the computer would execute it immediately. This was a completely different sort of interface to a computer than what was typical in those days, namely, batch mode programming. With the Logo approach, a budding programmer could immediately see the result of an interaction with the computer. This encouraged a whole different style of work on the computer, one that was more oriented to exploration and play—children's natural ways of thinking—than abstract symbolic thought.

The other feature of Logo projects was that they expanded the realm of the computer beyond data manipulation. Most computer interactions, including early Logo projects, were based on some sort of data transformation. Numeric or textual data would be processed by the computer and the result would be displayed for the programmer. For example, a Logo program might take lists of English nouns, verbs, and adjectives and string them together into nonsense sentences—a word play experiment.

Papert and his colleagues began experimenting with robots connected to computers running Logo. Rather than being fixed arms or XY-tables with Cartesian geometries, these robots were mobile robots that could be understood with a relative, robot-centric geometry. Children would first play with robot movement using button-boxes to control the robot's motion. Then they would use Logo primitives to control the robots, typing statements like `FORWARD 50` to make the robot move forward 50 "steps" or `RIGHT` [illegible] make the robot turn in place 90 degrees. By making sequences of these movement [illegible] to themselves as the robot and literally walk themselves through a Logo program by moving about as the robot would. Papert felt strongly that the way children were able to think about the robot by using their bodies, what he called a "body syntonicity," was key to making Logo accessible to children with a broader range of intellectual styles. More children would become engaged in projects based on robot control than data manipulation because they were able to "think with their bodies" in doing so.

The Logo robot became a defining feature of the Logo environment. Inspired by Walter's vacuum tube robot turtles, the Logo robot was dubbed the "turtle" because early robots were shaped vaguely like turtles, because they moved sort of like a turtle would, and to give children (and adult researchers for that matter) a playful and familiar object to hold in their mind when thinking about how the device would behave.

As computers developed video display technology that replaced line printer interfaces, the Logo turtle moved "off the floor" and "onto the screen." This is to say that the physical electromechanical Logo robots were supplanted by iconic images of turtles on the video display screen. When a child gave a command to a screen turtle, it would move about and draw on the display screen rather than on the floor.

The screen turtles had certain advantages and drawbacks with respect to the floor turtles. Perhaps the most important advantage of the screen turtles was that they could be used on any computer with a video display so they could reach many more children. Also, screen turtles could move very precisely and rapidly—there were no mechanical slippage problems—so children could easily create complex geometric displays.

On the other hand, screen turtles were more conceptually abstract than their floor turtle ancestors. Children had more difficulty understanding rotations from screen turtles—in some implementations of Logo, the turtle would snap immediately to its new position rather than step through a series of rotations to accomplish a movement. A child could not get up and walk around a screen turtle. When the turtle was facing downward on the screen, children would often become confused about the meaning of "turning left" (counter-clockwise rotation) versus "turning right" (clockwise rotation).

Still, the screen turtles were a valuable invention that greatly accelerated children's ability to relate to computer programming activities. It was much easier for children to understand how to program a turtle to make a drawing than to use cartesian geometry, the "native" language of the computer hardware, as was typically available in early microcomputer versions of the BASIC language.

1.2.1 LEGO/Logo

In 1971, Papert and colleague Cynthia Solomon published a short memo entitled "Twenty Things to Do with a Computer" [PS71]. When this paper was written, the contemporary interface to a computer was the TeleType—a line-oriented typewriter-like system, in which the user and computer alternately typed information onto a continuous scroll of paper.

Papert and Solomon's paper suggested 20 computer-based activities that assumed that many different kinds of electromechanical devices—similar in spirit to the Logo floor turtles—could be attached to the computer. Several of the activities involved the Logo turtles, but several others imagined more sophisticated and versatile possibilities [illegible]

difference between this work and the early Logo floor turtle experiments: With the newer work, children were able to not only write the programs to control electromechanical devices, but could actually build those devices as well. The vision laid out by Papert and Solomon's memo of 15 years ago was being realized.

The LEGO/Logo project, as it became called, used a recently developed set of LEGO parts called *LEGO Technic*. The LEGO Technic set included not only the familiar plastic LEGO building blocks, but newer pieces like gears, beams, wheels, and motors, which enabled a LEGO builder to create animated and exciting mechanical projects. The range of devices that could be constructed with LEGO Technic was as wide as the imagination: Children created ferris wheels, toasters, elevators, walking robots, and animated sculptures, to name just a few.

Coincidentally, as the MIT researchers were building prototype interfaces that allowed the Logo language to control LEGO devices, the president of the LEGO company read *Mindstorms*, Papert's book. Sensing a shared set of ideals about the role of children's play in learning and the value of constructive materials in children's hands and minds—whether they be the grammatic building blocks of the Logo language or the physical building blocks of a LEGO set—Papert's group and principals at the LEGO company arranged a meeting. A sponsored research project resulted, and Resnick and Ocko performed research and development that led to the commercialization of the LEGO/Logo system as a product for the educational market. The LEGO company named the first generation of the product *LEGO TC logo* ("tc" for Technic Control) and has been selling it since the late 1980s. Around 1990, an updated version, *LEGO DACTA*™ *Control Lab*™, was introduced; currently, it is estimated that 20,000 schools in the United States have the materials and more than 1 million children have used them.

1.2.2 The Programmable Brick

Shortly after Ocko and Resnick had finished their work on what became the *LEGO TC logo* product, they began looking for new directions to extend the LEGO/Logo concept. One limitation of the commercial product was that LEGO constructions needed to be tethered with wire to the electronic interface sitting aside the controlling desktop computer. The system tended to encourage the construction of stationary machines like a merry-go-round, rather than mobile machines like a LEGO floor turtle. Researchers in Papert's group were also interested in exploring children's work with mobile creature-like robots that exhibited cybernetic characteristics.

Resnick and Ocko experimented with remote control technology in which the controlling computer broadcasted commands to a LEGO machine that carried an infrared- or radio-based receiver. The system was functional, but it too had its limitations: Reliable, bidirectional communications (needed so the LEGO machine could report sensor data back to the host computer) were difficult to implement, and a system that would be suitable for classroom use, in which there might be a dozen or more simultaneous projects, would make the communications technology unwieldy. Additionally, it seemed like poor aesthetics for a big desktop computer to be controlling a little LEGO machine. Why not build a miniature computer that could be embedded into the LEGO machine

In about a year's time, we created a prototype Programmable Brick and manufactured about a half-dozen copies of them. The "Brick" had outputs to control four LEGO motors and inputs to receive data from four sensors. Existing LEGO sensors—a touch switch and light sensors—as well as custom sensors could be used. The Brick was based on a version of the 6502 microprocessor—the same device as was used in the Apple II series of computers, which were prevalent at the time. Brian Silverman ported the commercial version of Logo, written for the Apple II computer, to run on our Programmable Brick.

To program the Brick, it would be hooked up to a host computer using a serial line connection. Then the user could type commands to the brick or download Logo procedures to it. After downloading, Logo procedures could be invoked by giving commands through a "command center" on the computer screen or by pressing a button on the Programmable Brick, which would run a specially named Logo procedure.

This work on the MIT Programmable Brick has now come full circle. In 1998, the LEGO Group announced *LEGO Mindstorms*™, a new brand, and its flagship product, Robotics Invention System and the RCX Programmable LEGO Brick. The LEGO MindStorms brand represents the company's belief in the entertainment and educational value of robotics and its commitment in bringing it to children. The RCX Brick, inspired by the line of work at MIT beginning with our first Programmable Bricks in the late 1980s, is part of the first robotics system truly designed for children ever to reach the mass market.

1.2.3 The MIT 6.270 Robot Design Competition

Our work on the Programmable Brick for children led to the creation of the MIT Robot Design course and competition, through the intervention of an MIT undergraduate, Michael Parker, who worked in our LEGO/Logo lab at the time that the Programmable Brick was being developed.

Parker, a computer science student, had just taken MIT's Introduction to Mechanical Design class. In this class, created in the 1960s by Woodie Flowers, students are given an assorted kit of mechanical parts, including metal, plastic, wood, and cardboard, and the specification for a contest challenge. Over the duration of the term, students use the materials in their kit to design and build a machine that will satisfy the contest game. The course culminates in a contest in which the students' machines play against one another in an elimination-style event.

Parker had participated in the course and was excited by the concept. Having seen the Programmable Brick technology, Parker recruited me and Randy Sargent, another computer science student, to develop a version of the Brick for use in a class with MIT students.

It took several years to develop the format and technology for the project, including several revisions of the hardware and software provided to the students. Taking inspiration from Flowers' design course, we gave students a complete robot-building package, which included LEGO Technics elements for mechanical construction, various electronic sensors, and a controller board that could be programmed to operate autonomously. Like the mechanical design course, our project culminated in a final contest challenge. The big difference was that our project, which we called the LEGO Robot Design Competition, included electronics and software as part of the design space and required students to embed their control ideas in a program, rather than being able to control their machines

in more details about the evolution of the MIT course, including a study of students' learning, are referred to my Ph.D. dissertation, *Circuits to Control: Learning Engineering by Designing LEGO Robots* [Mar94].

Now let's look at the technology we use in this book. Much of it grew directly out of work done for the MIT LEGO Robot Design project.

1.3 About the Technology

In developing the projects for this book, I chose to base them around a particular set of hardware and software. This should make it much easier for those who do adopt these materials, while providing concrete examples that should also serve to illuminate the issues for any choice of technology. At the same time, I attempted to create examples and projects that would be applicable to being implemented with a variety of robotic technologies.

These materials were originally developed for the MIT Robot Design project and since have been further refined and also tested by an extended community of users who have adopted them for similar purposes. Although the materials obviously cannot satisfy all needs, they are very well suited as robot-design technology for students at a variety of levels of advancement.

The technology has three different components: hardware, software, and mechanism.

Hardware. For the electrical and computational hardware, we use the *Handy Board*, a hand-held, microprocessor-based robot control board. The Handy Board includes a variety of features that make it suited for controlling small robots, including outputs for DC motors, inputs for numerous sensors, an LCD screen, and an integrated battery pack.

Software. A custom software environment for the Handy Board, called *Interactive C*, has been developed. Interactive C is a multitasking implementation of the C programming language that is designed to run on a small, 8-bit microprocessor like the one that the Handy Board uses. Interactive C also has a console window that allows the user to interact with the robot while it is executing programs. This latter feature distinguishes Interactive C from other versions of the C language that are available for typical small microprocessor systems.

Mechanism. For the body of the robot, the LEGO Technic system is the basis of the examples and exercises presented in this book. LEGO Technic is an extension of the widely known and popular LEGO building brick system. Technic includes axles, gears, wheels, motors, and other parts to facilitate the construction of intricate and functional mechanical systems, including robot movement drivetrains and other effectors.

Following is a brief introduction to each of these materials; as we come to each of them in the main course of the text, much additional detail is presented. Also discussed here are some suggestions for alternatives.

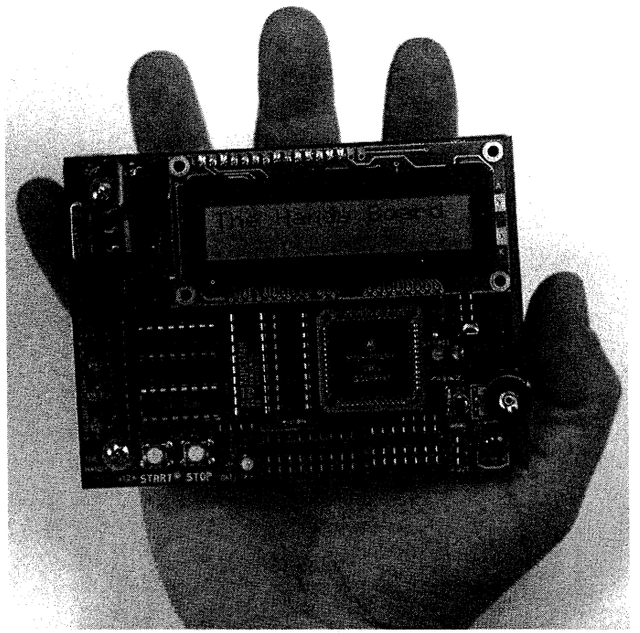


FIGURE 1.4 The Handy Board

Overview

The Handy Board's primary features include:

- *32K of main system memory.* The Handy Board has 32K of static RAM memory. This may not sound like a lot with today's desktop computers with tens of megabytes of memory, but remember that the Handy Board does not need to support a graphical user interface or a modern operating system. Further, the approach used by the Interactive C compiler uses the memory in an efficient manner.

  The Handy Board's memory is battery-protected, meaning that its contents are preserved even when power is switched off. This allows you to download a program to the robot, disconnect it from the computer, and carry it over to a play environment without worrying about losing the contents of the robot's memory.

- *Output drivers for four DC motors.* Up to four standard DC motors can be driven directly from the Handy Board. The drivers provide nine volts and about one ampere of current; this is ideal for powering many small motors.

  Under software control, the power provided to the motors can be varied in strength, providing a rudimentary speed control.

- *Inputs for analog and digital sensors.* Using a compact sensor connector, the Handy Board will accept up to seven analog (continuously varying) sensors and nine digital (switch type) sensors. The standard sensor connector is a three-wire circuit that can provide a five-volt power source to the sensor. This allows easy connection of active sensors, such as infrared reflectance sensors, that require power to

- *LCD screen.* The Handy Board comes with a 16-character, two-line liquid crystal display screen, which can display textual and numeric information under user control. The LCD screen greatly facilitates debugging of robot programs because sensor values and internal program state can be readily displayed on the screen.

For more information about the Handy Board design, see Appendix A.

Alternatives

The main features that the Handy Board provides that make it ideal for robotics projects are the integrated motor driver circuits, the 32K bytes of memory (which allows the use of Interactive C), its small size, and the sensor connector that allows the easy attachment of multiple, individually wired sensors. Most commercial microprocessor boards for embedded control applications do not allow easy attachment of individual motors and sensors.

Its limitations include the 32K bytes of memory (for some kinds of projects, this is not enough) and the fact that it is based on an 8–bit microprocessor (some applications will require more CPU horsepower). These concerns are discussed shortly.

For some applications, a board with even less capability than the Handy Board may be acceptable. This would be true if the intent were to support robot projects of limited complexity or if it were desired to program primarily in assembly language for pedagogical reasons. In this case, a controller board like the Mini Board, which has only 2K bytes of memory and no display screen, would be suitable (see Appendix G, *Resources*, for more information on the Mini Board). The Mini Board cannot support Interactive C, so it is typically programmed either with a traditional C compiler or in assembly language.

Most people will find the 2K byte memory limitation of a board like the Mini Board to be too constraining for open-ended, experimental work. Alternatively, there are dozens of inexpensive controller boards available, based on the 68HC11 or other embedded control microprocessors, that include 32K, 64K, or more memory. These designs are not typically targeted for robotic applications, however, and do not include circuitry for powering DC motors, convenient connectors for sensors, or a collection of software drivers for useful robotic peripherals. This is where the Handy Board is especially valuable.

There are applications for which the power of the Handy Board will not be adequate. For example, sophisticated programming languages like Lisp or other interpreted environments cannot be implemented with the limited memory and CPU power. Robot tasks that involve the collection of large amounts of data will exhaust the 32K bytes of memory. Finally, some sensory processing tasks, like visual recognition, require newer and more powerful microprocessors.

This should not be seen as selling short the versatility of the Handy Board, however. For a wide range of projects, including beginning through intermediate robotics work, C-language and assembly language programming, and electronic design and interfacing, the Handy Board is an ideal solution.

Indeed, the 8–bit microprocessor is far from having outlived its usefulness, despite the barrage of information to the contrary coming from the chip industry. Certainly on the desktop, faster and faster processors continue to make radical changes in the nature of the computing experience. Yet do we really need a 32–bit microprocessor running [illegible]? Probably not. In recent years in the embedded controls field, there actually [illegible]

1.3.2 Interactive C

The Handy Board can be programmed using a variety of standard 68HC11 development tools, including C-language compilers and assemblers for writing directly in 68HC11 code. The standard software environment used in this book, however, is *Interactive C*, a special kind of C-language compiler developed for educational robotics applications.

Interactive C was created by Randy Sargent and the author originally for use in the MIT Robot Design project. There were several considerations in the design of Interactive C that make it ideal for robotics projects:

Interactivity. With a conventional C-language compiler, it is difficult to interact with one's program while it is running. The typical development sequence consists of writing code, compiling it, and then running it. Such programs are difficult to debug because it is hard to test the various subprograms independently.

Interactive C instead provides a command-line console that allows the user to type expressions and function calls interactively, even while other programs are running. This capability, the hallmark of interpreted computer language environments, makes it much easier to try out ideas as they are hatched.

Stability. In the traditional C-language system, the compiler does not protect the programmer from making mistakes that will crash the computer. For example, if the programmer references an array element that is out of bounds of the predetermined size of the array, the computer will gladly oblige and allow errant data to corrupt system memory (leading to a crash).

In working with our students in the MIT Robot Design project, we found that debugging was especially difficult when not only software errors but also electronic hardware failures could cause a system crash. Therefore, we designed Interactive C to report a runtime error for common programming problems (e.g., divide-by-zero, out-of-bounds array reference) rather than crashing the system.

Multitasking. For many of the ideas we wished students to explore with their robotics projects, the capability to have multiple programs running simultaneously was important. Interactive C has built in the ability to multitask up to about a dozen user programs.

There are two versions of Interactive C: a free version that is based on the original MIT source code, and a commercial version that is a product of Newton Labs, a robotics firm that Sargent has since founded. The projects and examples prepared in this book can all be completed with the free version. The commercial version provides additional features, as well as an improved user interface that would be of interest to the advanced user. Both the free and commercial versions are available for a number of popular computer operating systems, including MS-DOS, Windows, Macintosh, and Unix.

Alternatives

Some users may wish to program their Handy Board directly in 68HC11 machine language. This is generally a prerequisite when writing drivers for new devices, but also has great pedagogical value as a way to deeply understand the interactions between the microprocessor and the hardware of the Handy Board. Generally, it is more difficult to

assembler program and load it onto the Handy Board, are also freely available. See the *Resources* appendix.

For programming in C, there are a number of high-quality conventional C-language compilers available for the 68HC11 chip. Some of these are targeted at industrial users and cost USD$1000 or more. These tend to be complex and difficult to use, and it is not necessary to spend anywhere near that amount to purchase a reliable product.

One of the best of the inexpensive C-language compilers for the 68HC11 is ICC11 by ImageCraft, Inc. MS-DOS and Windows versions are available, and a Macintosh version is planned. ICC11 is officially supported as a compiler for use with the Handy Board. Another good compiler product is Micro-C by Dunfield Development Systems. Micro-C is available in MS-DOS and Windows versions only.

1.3.3 LEGO Technic

For the structural and mechanical aspects of robot design, this book is based on the use of LEGO Technic. *Technic* is the brand name for the mechanized portion of the LEGO product line. LEGO Technic includes axles, gears, motors, and other parts that allow the construction of complex and functional mechanical systems.

LEGO Technic is a wonderful material because it combines the playful appeal of the basic LEGO brick with a sophisticated and powerful set of mechanical design components. LEGO Technic allows rapid idea generation, prototyping, and evaluation. For many projects, Technic is satisfactory as a final building material, meaning that the robot built as a prototype from Technic can also serve as a valid experimental platform.

Technic is ideal for both structural purposes—building the frame or other support elements of a project—as well as the gear reduction and power transmission mechanism. Because of this, LEGO Technic can be used to create a huge variety of different mechanisms and mechanical systems that are designed from the ground up. It can be a profoundly satisfying feeling to design, build, and debug a mechanical device that performs as intended. With LEGO Technic, this can be done without knowing how to use the tools in a machine shop, greatly accelerating the creative process.

Most of the examples in this book are demonstrated using the LEGO Technic system. Although some of these ideas are particular to the Technic product, most of the mechanical design principles apply to any type of mechanical design. Because of this versatility and expressive power, the LEGO Technic system is adopted as the standard building material for this book.

Considerations

LEGO Technic sometimes has a reputation for being expensive and not structurally sound.

Cost. Although it is certainly true that LEGO Technic is not inexpensive, it is comparable in price to other high-quality building systems, such as *Fischertechnik*. Compared with products on a retail toy shelf, LEGO Technic is more expensive, but the quality of the LEGO manufacturing process—which uses only the best plas- [illegible] precision molding techniques—far surpasses copycat products. In short, if [illegible] overpriced, and

with cross-beams that will lock their designs in place. These secrets are (of course) revealed in this book and change perceptions of LEGO materials as not being capable of building sturdy structures.

Alternatives

Fischertechnik, manufactured by the German company of the same name, is a precision building system that shares many qualities with the LEGO Technic product.[3] While LEGO Technic evolved from the famous rectangular building brick, *Fischertechnik* started out as a system for engineers to prototype factory layout designs. As such, *Fischertechnik* excels at constructing rectilinear structures; it has a locking joint that allows these frame-like structures to be built with great rigidity.

Fischertechnik now includes elements for mechanized structures, such as gears, axles, and other such parts. Projects designed with *Fischertechnik* tend to be larger than their LEGO Technic counterparts and have a more industrial appearance.

Although this author prefers LEGO Technic for its play value and greater versatility, both LEGO Technic and *Fischertechnik* are very high-quality, powerful building systems. For those who have prior experience with *Fischertechnik* or otherwise find it appealing, it is an excellent choice.

Fischertechnik and LEGO Technic cost approximately the same, so neither are solutions if cost is the issue. For building robots on a budget, several possibilities exist:

Modified Radio Control Cars. Inexpensive toy radio control (RC) cars can make excellent robot platforms because they are durable and well designed for basic locomotion. There are some challenges involved in using them, centering around the problem of interfacing to their high-current motors, but once these are solved, the RC car is a good alternative.

Scrap Materials. For those who are comfortable working with basic construction stock like wood, metal, and plastic, robot building can be a very rewarding endeavor. It is certainly the case that building a robot from these "raw" materials is much less expensive than using the commercial building systems. For the mechanized aspect of the designs, several companies sell gears, chain link, and other such components, or such parts can be removed from junk devices like portable cassette stereos and RC cars.

Teaching how to design from scrap materials is beyond the scope of this book, but many of the design principles that are discussed in this book are relevant to builders who are working in the machine shop.

1.4 An Overview of the Book

The text has been organized with a set of chapters and appendices to encourage nonlinear use. That is, the progression of ideas from the first chapter to the last appendix does not follow a sequential progression, and readers are encouraged to skip around and pursue their own path through the material.

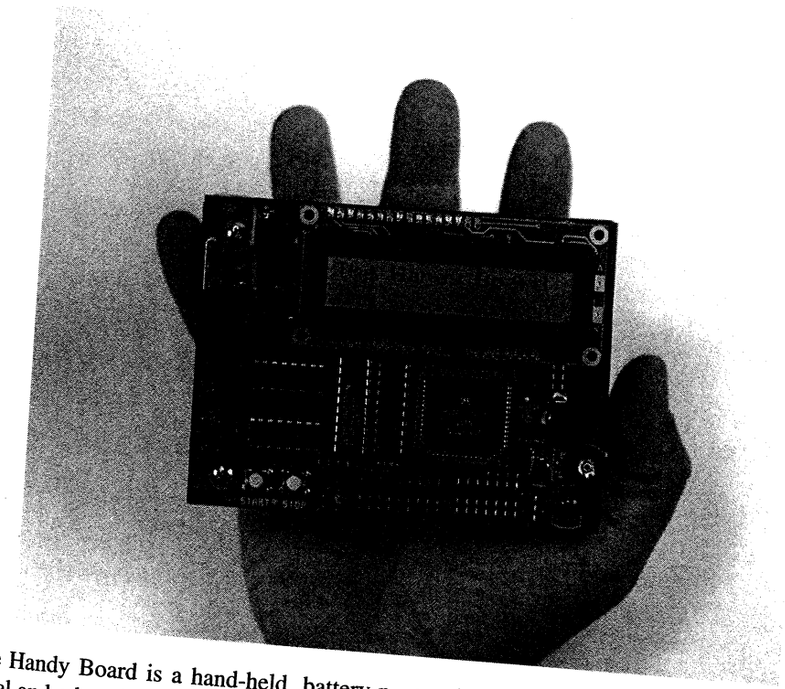
The book is filled with various exercises and examples to help illustrate the ideas presented. Although it is certainly possible to be an "armchair roboticist" and enjoy the [illegible] project work goes hand in hand with the

ing of

```
/*
  datacoll.c
  data collection and printing

  requires printdec.c, serialio.c
*/

int SAMPLES=1000;
char data[1000];

void main()
{
  disable_pcode_serial();

  printf("press Start to  collect data\n");
  start_press();
  collect_data();
  beep();

  printf("press Start to  dump data\n");
  start_press();
  dump_data();
  beep();

  printf("done.\n");
}

void collect_data()
{
  int i;

  for (i= 0; i< SAMPLES; i++) {
    data[i]= analog(0);
    /* to slow down capture rate, add msleep here */
  }
}

void dump_data()
{
  int i;

  for (i= 0; i< SAMPLES; i++) {
    printdec(data[i]);
```

APPENDIX

D Handy Board Specification



The Handy Board is a hand-held, battery-powered microcontroller board ideal for personal and educational robotics projects. Based on the Motorola 68HC11 microprocessor, the Handy Board includes 32K of battery-backed static RAM, outputs for four DC motors, inputs for a variety of sensors, and a 16×2 character LCD screen. The Handy Board runs Interactive C, a cross-platform, multitasking version of the C programming language.

The Handy Board is distributed under MIT's free licensing policy, in which the design may be licensed for for personal, educational, or commercial use with no charge.

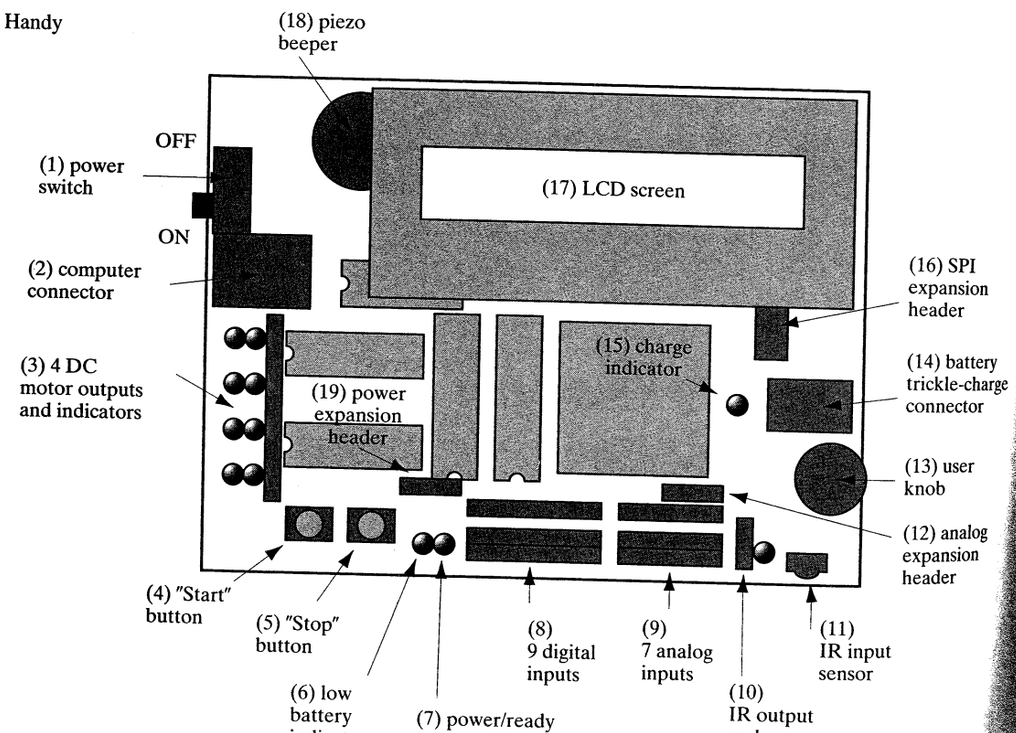
- two user-programmable buttons, one knob, and piezo beeper.
- powered header inputs for seven analog sensors and nine digital sensors.
- internal 9.6v nicad battery with built-in recharging circuit.
- hardware 38 kHz oscillator and drive transistor for IR output and onboard 38 kHz IR receiver.
- 8-pin powered connector to 6811 SPI circuit (1 Mbaud serial peripheral interface).
- expansion bus with chip selects allows easy expansion using inexpensive digital I/O latches.
- board size of 4.25 × 3.15 inches designed for a commercial, high-grade plastic enclosure that holds battery pack beneath the board.

D.2 Ports and Connectors

Figure D.1 shows a labeled view of the Handy Board's ports, connectors, inputs, and outputs. In the following, each of these is briefly described.

1. **Power Switch.** The power switch is used to turn the Handy Board on and off. The Handy Board retains the contents of its memory even when the board is switched off.

1 Labeled Handy agram



the motor connects to the outer two pins and the center pin is not used. Red and green LEDs indicate motor direction. From top to bottom, the motor outputs are numbered 0 to 3.

4. **Start Button.** The Start button is used to control the execution of Interactive C programs. Also its state may be read under user program control.

5. **Stop Button.** The Stop button is used to put the Handy Board into a special bootstrap download mode. Also its state may be read under user program control.

6. **Low Battery Indicator.** The red Low Battery LED lights when for a brief interval each time the Handy Board is switched on. If this LED is on steadily, it indicates that the battery is low and that the CPU is halted.

7. **Power/Ready Indicator.** The green Power/Ready LED lights when the Handy Board is in normal operation and flashes when the Handy Board is transmitting serial data. If the board is powered on and this LED is off, then the Handy Board is in special bootstrap mode.

8. **9 Digital Inputs.** The bank of digital input ports is here. From right to left, the digital inputs are numbered 7 to 15.

9. **7 Analog Inputs.** The bank of analog input ports is here. From right to left, the analog inputs are numbered 0 to 6.

10. **IR Output and Indicator.** The IR output port is here. The red indicator LED lights when the output is enabled.

11. **IR Input Sensor.** The dark green-colored IR sensor is here.

12. **Analog Expansion Header.** The analog expansion header is a 1×4 connector row located above analog inputs 0 to 3.

13. **User Knob.** The user knob is a trimmer potentiometer whose value can be read under user program control.

14. **Battery Trickle-Charge Connector.** The battery charge connector is a coaxial power jack to accept a 12-volt signal for trickle-charging the Handy Board's internal battery.

15. **Charge Indicator.** The yellow charge indicator LED lights when the Handy Board is charging via the coaxial power jack.

16. **SPI Expansion Header.** The SPI expansion header is a 2×4 pin jack that allows connection with the 6811's *serial peripheral interface* circuit. See the CPU and memory schematic diagram for a pin-out of this connector.

17. **LCD S**

D.3 Battery Maintenance

The Handy Board has a 9.6v, 600 mA battery pack consisting of eight AA-cell nickel-cadmium rechargeable batteries.

D.3.1 Battery Charging

There are three ways to charge the internal battery:

1. *Adapter plugged directly into the HB.* Just plug the adapter into the power jack on the HB and the yellow "CHARGE" LED on the HB will light. This is a trickle-charge mode, which means that (1) the Handy Board will fully charge in about 12 to 14 hours, and (2) the HB may be left in this mode indefinitely.
2. *Adapter plugged into the Serial Interface/Battery Charger board; HB connected via telephone wire; "NORMAL CHARGE" mode selected.* The yellow "CHARGE" LED on the interface board will light. This is a trickle-charge mode, which means that (1) the Handy Board will fully charge in about 12 to 14 hours, and (2) the HB may be left in this mode indefinitely.
3. *Adapter plugged into the Serial Interface/Battery Charger board; HB connected via telephone wire; "ZAP CHARGE" mode selected.* The yellow "CHARGE" LED on the interface board will *not* light. The ZAP CHARGE will fully charge the HB's battery in just 3 hours, *after which time the battery will become warm and should be removed from charge or placed into either of the two trickle-charge modes.*

When using one of the trickle-charge modes, the Handy Board should be turned off so that the charge current goes toward charging the battery and not simply running the board. In Zap charge, there is enough charge current to operate the board and charge the batteries at the same time (assuming that the board is not driving motors or other external loads).

D.3.2 Adapter Specifications

The specifications of the Handy Board's DC adapter are as follows:

- 12-volt, 500 mA DC output
- 2.1 mm inside, 5.5 mm outside diameter barrel-type plug
- center conductor negative

Most "universal" type adapters will work properly at one of their settings. Look for the yellow charge LED to light up indicating proper charge (make sure the Charge Rate switch is set to "Normal" mode).

Please be careful not to get an adapter that is *overpowered*. Problems have been

D.4 Part Listing

Circuit: hbsch12
Date: Thursday, November 30, 1995 - 9:58 AM

Device Type	Num.	Value	References	Price Ea.	Catalog No.	Supplier
8 cell AA nicad pack	1		BAT1	19.28	P227-L024-ND	Digikey
2% polyprop cap	1	0.0068 uF	C6	0.49	P3682-ND	Digikey
monolithic cer cap	4	0.1 uF	C5 C7 C9 C14	0.21	P4917-ND	Digikey
mini radial 'lytic	4	10uF	C10 C11 C12 C13	0.08	P6248-ND	Digikey
monolithic cer cap	2	22 pF	C1 C2	0.18	P4841-ND	Digikey
telephone cable	1	4-wire	CAB1	1.60	17MP007	Mouser
tantalum	2	4.7 uF	C4 C8	0.29	P2011-ND	Digikey
mini axial 'lytic	2	47 uF	C15 C16	0.29	P5972-ND	Digikey
mini axial 'lytic	1	470 uF	C3	0.65	P6305-ND	Digikey
power diode	1	1N4001	D3	0.15	333-1N4001	Mouser
signal diode	1	1N914	D1	0.15	333-1N914	Mouser
bridge rectifier	1	DB101	D2	0.62	DB101-ND	Digikey
AC or DC adapter	1	12v, 500mA	DC1	3.95	100087	Jameco
CPU board enclosure	1		ENCL1	5.12	537-402-RD	Mouser
interface enclosure	1		ENCL2	1.94	400-5043	Mouser
PolySwitchfi fuse	1		F1	1.32	RUE250-ND	Digikey
Coax Power Jack	2	2.1mm ID	J11 J12	0.34	CP-002A-ND	Digikey
RJ11 top entry	1	6/4	J5	1.08	154-UL6642	Mouser
RJ12 side entry	1	6/6	J10	1.28	154-UL6661	Mouser
10-pin female header	1		J3			
12-pin female header	1		J4			
14-pin female header	1		J14			
14-pin male header	1		J15			
3 pcs 9-pin female hdr	1		J2		[FEMALE HEADER IS CUT	
3 pcs. 7-pin female hdr	1		J1		FROM 36-PIN HEADER	
3-pin female header	1		J7		LISTED AT END OF PAGE]	
4-pin header	2		J8 J13			
4x2 header, female	1		J6			
DB-25 female connector	1		J9			
iron core inductor	1	1 uH	L1	1.54	152-3425	Mouser
high-eff red LED	7	HLMP1700	LED1 LED2 LED3 LED4	0.84	M7010-ND	Digikey
			LED9 LED11 LED13	0.282	HLMP-1700QT-ND	Digikey
hi-eff yellow LED	2	HLMP1719	LED14 LED15	0.282	HLMP-1719QT-ND	Digikey
hi-eff green LED	6	HLMP1790	LED5 LED6 LED7 LED8	0.282	HLMP-1790QT-ND	Digikey
			LED10 LED12			
NPN darlington	1	ZTX614	Q1	0.59	ZTX614-ND	Digikey
	2	10K	R3 R7	0.0235	10KEBK-ND	Digikey
	3	1K	R2 R5 R10	0.0235	1KEBK-ND	Digikey
	1	2.2K	R9	0.0235	2.2KEBK-ND	Digikey
	1	2.2M	R1	0.0235	2.2MEBK-ND	Digikey
	1	2.94K	R4	0.11	2.94KXBK-ND	Digikey
1% precision res	1	20K	VR1	0.72	569-91AR-20K	Mouser
trimpot	3	47K	R6 R8 R15	0.0235	47KEBK-ND	Digikey
	2	47	R12 R13	0.0235	47EBK-ND	Digikey
	2	47 , 1/2W	R11 R14	0.06	47H-ND	Digikey
RPACK6	1	1Kx4	RP4	0.21	592-8A-1K	Mouser
RPACK9	1	1Kx5	RP2	0.16	592-6S-1K	Mouser
14-pin DIP socket	2	47Kx9	RP1 RP3	0.27	592-10S-47K	Mouser
16-pin DIP socket	2		DIP4 DIP5	0.57	ED3114-ND	Digikey
20-pin DIP socket	4		DIP6 DIP7 DIP8 DIP9	0.65	ED3116-ND	Digikey
28-pin DIP socket	2		DIP1 DIP2	0.81	ED3120-ND	Digikey
52-pin PLCC socket	1		DIP3	1.13	ED3728-ND	Digikey
piezo beeper	1		PLCC	2.03	A2123-ND	Digikey
SPDT slide switch	1		SPKR1	1.90	P9957-ND	Digikey
SPDT switch	1		SW1	4.47	CKN5006-ND	Digikey
pushbutton switch	1		SW4	1.10	SW101-ND	Digikey
32K static CMOS RAM	2		SW2 SW3	0.20	P8006S-ND	Digikey
hex inverters	1	62256-100LP	U2	3.95	42833	Jameco
quad Schmitt NANDs	1	74HC04	U9	0.29	570-CD74HC04E	Mouser
3-to-8 decoder	1	74HC132	U7	0.46	511-M74HC132	Mouser
tristate bus driver	1	74HC138	U6	0.46	570-CD74HC138E	Mouser
transparent octal latch	1	74HC244	U5	0.70	570-CD74HC244E	Mouser
octal latch	1	74HC373	U3	0.68	570-CD74HC373E	Mouser
voltage monitor	1	74HC374	U8	0.61	570-CD74HC374E	Mouser
infrared demodulator	1	DS1233-10	U12	1.25	manufacturer	Mouser
motor driver	1	IS1U60	U15	3.00		Dallas Semi
voltage	2	L293D				