Contents

4
Designing Control Systems

This book is concerned with the behavior of processes. The world we live in can be described in terms of a set of interacting processes. In the previous two chapters, we discussed how to model the behavior of processes. In this chapter, we begin to consider how to influence that behavior.

Some processes are easier to control than others. For instance, someone typing at a word processor generally has a fair amount of control over what characters appear on the screen. Other processes are influenced by a large number of factors, only a few of which we are able to directly observe or influence. In sending an electronic mail message, for example, the speed with which the message arrives at its destination is determined in part by the path provided and in part by the traffic on the networks specified in that path. Electronic mail users can directly control the former but have little control over the latter. If you could somehow predict the traffic on the network, then you might be better prepared to specify a path that would speed your message to its destination. Unfortunately, predicting network traffic flow is itself a complicated and time consuming task.

In studying the control of processes, it is often convenient to describe the world in terms of two processes: one of which we have absolute control over, and a second process that we wish to control. The first is called the *controlling process* and the second the *controlled process*. In the automatic control literature, the controlling process is called the *controller* or *regulator*, and the controlled process is called the *plant*. The behavior of the controlling process is determined in part by the control-system designer. Given some desired behavior for the controlled process, the task is to design a device that realizes the controlling process and forces the desired behavior in the controlled process.

The interaction between controlling and controlled processes can be quite complex. We generally think of the controlling process as calling all the shots, but the control exerted by the controlling process over the controlled process is seldom complete. Factors that influence the controlled process but

are not under the control of the controlling process have to be accounted for. The controlled process can, and in many cases must, influence the controlling process in order to bring about the desired behavior. This influence is mediated through the use of special devices used by the controlling process to *observe* the behavior of the controlled process.

Information about the observed behavior of the controlled process is often used by the controlling process in determining what action to take next. This basic idea that the responses of the controlling process are computed from the observed behavior of the controlled process is generally referred to as *feedback* control. In some cases, the need for observation can be reduced or even eliminated by using models to *predict* the behavior of the controlled process.

In this chapter, we consider techniques drawn primarily from control theory and control systems engineering. We focus on the role of feedback in the design of control systems with an emphasis on representations and techniques that stress computational issues. We introduce criteria for controllability, observability, stability, and optimality, and consider a variety of problems to illustrate these concepts. We then consider some basic feedback controllers and how they might be embedded in a computational framework. In the context of discussing feedback control, we introduce programming approaches that are well suited to building control systems that have to be particularly responsive to change. We end this chapter by considering a problem in robotics that lies at the boundary between those problems traditionally considered within the purview of control theory and problems associated with artificial intelligence. The objective here is not to provide a comprehensive survey of control techniques, but rather to draw on the control disciplines for insights and general techniques that apply to the full range of planning and control problems. Before launching into the more technical discussions drawing on results from control theory, we consider a particular problem to illustrate some basic issues.

4.1 Basic Issues

Consider the following control problem. Suppose that you want to control a robot to move from one location to another in a city. The robot has to travel using city streets that are arranged as an irregularly-spaced grid of two-way streets (see Figure 4.1). You have to devise a control algorithm to direct the robot to move from its present location to a destination location defined in

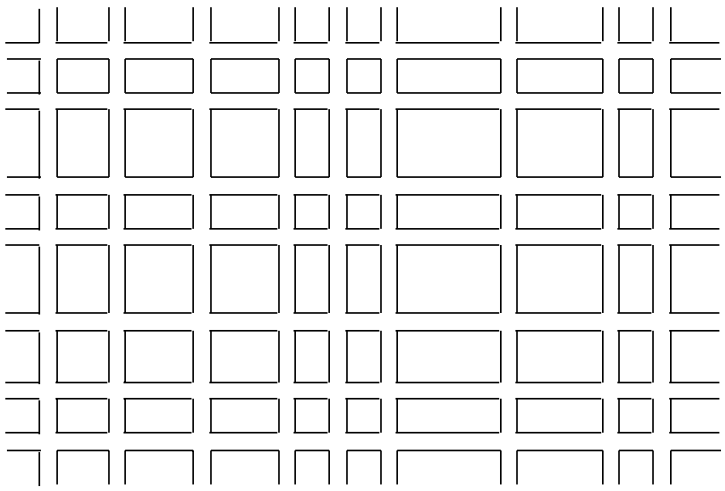

Figure 4.1: A city street layout

terms of global coordinates. Of course, the problem is not yet well enough specified that you can run off and start writing down an algorithm. There are a number of other factors that we have to consider.

First, what sort of control can we exert over the robot? Most likely there will be some means of controlling the robot's speed and direction of travel, but it is not likely that the robot will move exactly where we tell it nor will it move at precisely the speed that we specify. If we indicate that the robot is to move due south at 12 kilometers per hour and there is a brick wall in the way, then we might expect some difference between the specified and the actual speed and heading. Usually, however, the differences between actual and specified control variables are more subtle. Errors accumulate and combine in executing a sequence of control actions. Sooner or later it becomes necessary to compare the actual effect against the intended effect, and this is where sensors enter into the picture.

Sensors are used to monitor the progress of the robot and to determine the state of the environment. Sensors can determine and correct for movement error. For instance, the robot might be equipped with shaft encoders for determining how many revolutions the drive wheels have turned or what direction the wheels are pointing. From this information, we can compute an estimate of where the robot is relative to where it started out. Sensors and the estimates derived from sensor data are also subject to errors. Somehow or another we have to take such errors into account. For instance, it may

be that the errors are known to satisfy a particular statistical distribution from which we can calculate a measure of how certain we are in the inferences derived from sensor data. If our confidence in our inferences is low, then that could mean that we lack sufficient information to formulate a good answer to the control problem we are faced with. In some cases, being left with insufficient information is unavoidable and we must proceed to schedule critical control actions with whatever information we have at hand. In other cases, we can use sensors to gather additional information so as to make inferences that we are more confident in.

Sensors tell us about more than just the state of the robot; they tell us about the state of the larger world in which the robot is embedded. In the simplest robot navigation tasks, the only thing that changes is the robot itself and its position in the world. The environment is said to be static. If we know something about the fixed state of the environment, then we can take advantage of this in designing a control algorithm. Knowledge of the environment might take the form of a map labeled with street names, whether or not traffic moves in one direction or both, and whether there are stop signs or other impediments to traffic flow.

In more realistic problems, the environment changes; there are other vehicles on the road, traffic lights change, roads are blocked by construction, and pedestrians occasionally dart out into traffic. The static map may still be useful, but often we can supplement our knowledge of the environment to account for dynamic phenomena. For instance, we might have access to a construction schedule indicating where and when certain streets will be closed to traffic. In some cases, we might be able to model certain disturbances as predictable processes. A construction crew might be laying new gas pipe under a particular street at the rate of one block per night so that at most one block-long section of the street is impassable on any given night. If you notice the crew laying pipe on any two nights, you can predict what block will be closed off for any subsequent night.

While some processes are predictable, others are either difficult to predict (*e.g.,* jay-walking pedestrians) or not worth the trouble (*e.g.,* traffic lights). In order to deal with such processes, the control algorithm has to be alert to changes in the environment that indicate the existence of processes whose behavior might have an impact on the performance of the robot. The robot has to be continually alert for evidence of certain processes (*e.g.,* pedestrians straying into the street in front of the robot). Other processes need only be monitored in certain circumstances. For instance, the robot has to check for the state of the traffic light at the next intersection only as it approaches that

intersection. The design of the control algorithm must take into account the sensors available and the tasks they are to be put to. Sensors often constitute a scarce resource in need of careful management.

There is another aspect of the control of our mobile robot that we have carefully avoided up until now, and that concerns how the algorithm that we devise is to be implemented. In order to implement a control algorithm, we need to specify the algorithm in terms of a language, and we have to provide a compiler for that language, and a target machine for the code generated by the compiler. In fact, it generally is difficult to specify a control algorithm without some specific implementation in mind.

How long a series of program statements takes to execute on a particular machine may be critical in determining the consequences of a control action. For instance, suppose that you want to compute how to respond in the case in which a pedestrian runs out into the street in front of the robot. Certainly it would be a good idea to apply the brakes as soon as possible if indeed that is an appropriate thing to do. How long the algorithm takes to compute whether or not to apply the brakes will have a profound impact on the health of the pedestrian in question. If the robot is to swerve in an attempt to avoid hitting the pedestrian, then the direction in which the wheels are turned will depend upon the time that they are turned, and this will depend upon the time it takes to compute the direction.

In some cases, we can just assume that the time required to compute responses is shorter than the time available for computation. For instance, suppose that at time t the robot interprets its sensor data as indicating a pedestrian standing in the street five meters directly in front of it. The robot attempts to compute what action to take at time $t+\Delta$. The control algorithm is implemented so that the time required to compute such a response is less than Δ. Having computed an appropriate answer, the control algorithm might simply wait out the remaining time, or hand the action and the time it is to be executed to a sequencer responsible for executing actions at specified times. Of course, if the robot is traveling at a meter a second and Δ is longer than a couple of seconds, then the response will likely be too late to be of any use.

Some of the decisions concerning how long to spend computing an appropriate response in a given set of circumstances can be carried out at design time. Other decisions concerning how long to compute are better left until run time when the allocation of computational resources can be based on more data about the situation at hand. If the lead time for responding to a certain sort of phenomena varies, then having a rigid scheme for comput-

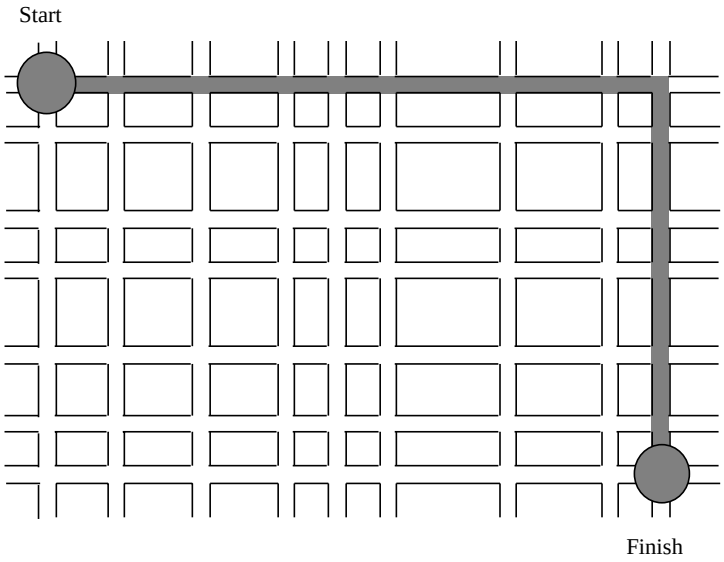

Figure 4.2: A path generated by dead reckoning

ing a response may lead to poor performance on average. Jamming on the brakes is only appropriate as a last resort. In situations where more time is available to arrive at a decision, a more careful analysis is often called for. In this chapter, we ignore many of the issues that relate to the run-time allocation of processor time to optimize decision making. Chapter 8 directly addresses these issues. In this chapter, we take a conservative approach to ensure that the algorithms that we develop perform reasonably for even the worst-case situations anticipated.

So far we have considered several factors that are important in specifying control problems. Now, we consider some specific control problems. In an ideal world, when the robot is told to turn left 15° and move forward at two meters per second for five seconds, the robot ends up exactly 10 meters from its original position facing 15° counter-clockwise from its original heading. Consider the problem involving a static environment in which all of the streets allow two-way traffic and are obstacle free, and the robot is standing in the center of an intersection and is instructed to move to the center of a second intersection specified in x and y coordinates in the frame of reference of the robot's initial position. In this case, an appropriate control algorithm would direct the robot to complete the traversal in two steps following the

paths indicated by the x and y offsets (see Figure 4.2).

In the above ideal world, the robot is said to direct itself by "dead reckoning." Aside from a clock to measure the passage of time, and thereby gauge the distance traveled, the robot requires no sensors to direct its motion. Suppose that we relax the requirement that the robot be able to control its velocity precisely. In this case, it is possible that the robot's estimates of distance traveled are subject to error. How is the problem changed as a consequence? If the errors are small relative to the length of a city block, a simple variation on the dead-reckoning approach will work just fine. If the errors are large, then the problem may be impossible to solve since the robot will have no way to determine if it reaches its destination. Even if the robot has some other means of detecting that it has arrived at its sought-after destination, significant movement errors may force the control algorithm to randomly choose paths.

Suppose that the robot can determine its position at any time in some global coordinate system. Now movement errors can be corrected by what is generally referred to as *feedback*. The control algorithm attempts to move five meters to the left; it checks to see how far it actually moved; it attempts to correct for the error observed. As long as the errors are some fraction of the distance attempted, this technique will converge quickly on the desired distance. If determining global position is fast enough, then this technique reduces to the previous dead-reckoning method.

Now suppose that not all streets are passable; some streets are one-way and others are blocked by construction equipment. The dead-reckoning approach will obviously not work, but a simple path-following strategy will suffice to find a path if one exists. Figure 4.3 shows the streets traversed by the robot under the control of a simple path-following algorithm that tries to shorten the Euclidean distance to the destination whenever possible, backing up only when its way becomes blocked. The problem is that directing the robot using the simple path-finding strategy causes the robot to traverse streets that it might not have if it possessed a more global perspective of the city.

Suppose that the robot has an accurate map of the city indicating one-way streets and construction road blocks. Rather than actually traversing the streets, the control algorithm could use the map to *simulate* traversing the streets and thereby find a short path. Computing the shortest path between any two locations can be done in $O(n^2 \log n)$ time using a best-first search algorithm [2], assuming a square grid of streets with n streets along each axis of the grid. Figure 4.4 shows the streets traversed by the robot

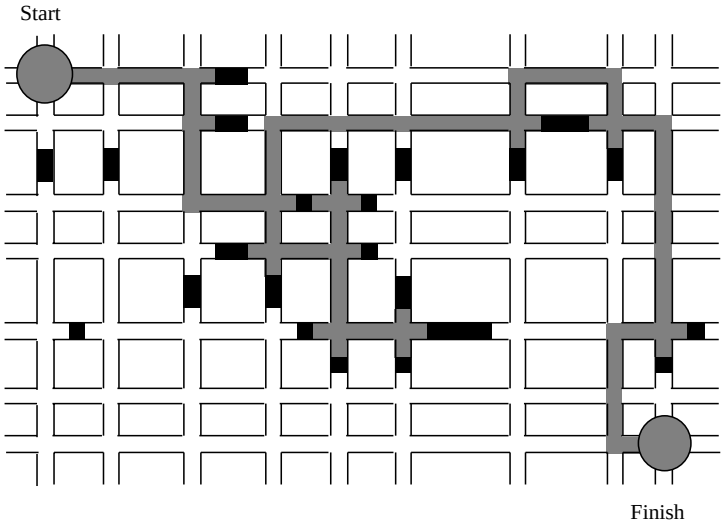

Figure 4.3: Navigation without the aid of a map

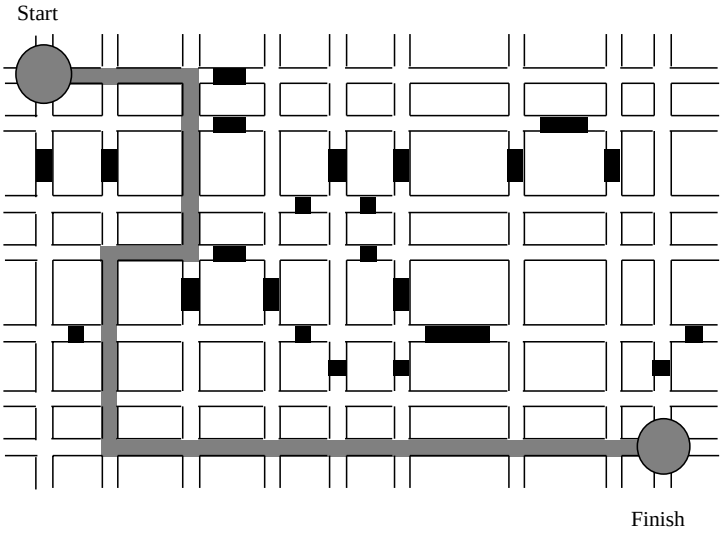

Figure 4.4: Navigation using path planning and a global map

under the control of an algorithm with access to a map. This method of simulating the behavior of the robot in order to eliminate unnecessary work or avoid an undesirable effect represents an instance of *feedforward*. The

control algorithm generates and analyzes possible actions and their consequences so that it can choose among the available options.

The use of feedback and feedforward are common in the design of control systems. Feedback compensates for a system's inability to accurately predict the effects of a control action on the behavior of a controlled process. Feedback relies on being able to accurately monitor the behavior of a process. Feedforward enables a system to anticipate both desirable and undesirable consequences and take steps to, respectively, take advantage of or avoid them. Feedforward relies on a system having an accurate model for the process being controlled.

Feedforward and feedback complement one another. In situations in which the controlled process cannot be accurately predicted but can be closely monitored, tight feedback loops enable a control algorithm to generate control actions on the basis of immediate past performance. Such a scheme is likely to work assuming that the factors influencing the process at one point in time are similar in type and magnitude to the factors influencing the process a short time previously. In situations in which the controlled process cannot be accurately monitored but can be accurately predicted, control actions are generated in response to predictions concerning the behavior of the process. If the process cannot be monitored at all, then control proceeds blindly relying on the accuracy of the predictive model.

Traditional methods in planning stress the use of feedforward methods whereas traditional methods in control stress the use of feedback. The reason for their different emphases is easy to explain. First of all, planning is by definition concerned with predicting the future in order to guide behavior. Much of the early work in planning was concerned with processes that interact with one another in a complex manner, and, hence, influencing the behavior of these processes required anticipating these interactions. This early work generally assumed that the controlled process, while complex, was understood well enough to be accurately modeled. More recent work has begun to relax this assumption by either using feedback to supplement predictions or using stochastic models that take uncertainty into account.

In contrast with the work in planning, much of the early work in control assumed that the controlled process was subject to a multitude of factors that either were not well understood or required run-time data that simply was not available. Precise adjustments to the control parameters were needed to achieve the desired behavior requiring that the controlling process be able to generate the necessary control actions at a high rate. A more complex algorithm for determining the next control action lowers the rate

at which control actions can be generated, whereas, the more inaccurate the models are in predicting the effect of control actions, the more frequently the controlling process has to be monitored and the control parameters adjusted to compensate for the inaccuracies of the model. In the past, many industrial control applications have favored trading model complexity for increased reliance on feedback and higher parameter-adjustment rates. As computers become faster and our modeling techniques more reliable, there has been a tendency to incorporate more and more complex modeling techniques into industrial controllers. If this trend continues, industrial controllers will begin to look more like planners.

As the control community begins to realize the advantages of increased computational power for supporting complex modeling, so the planning community is beginning to realize the problems in relying solely on the predictions of a complex model. Correcting these problems is not simply a matter of building an interpreter that executes a sequence of actions generated by a traditional planner and occasionally senses the environment to see if the actions have had their desired effect. The problem with this approach is that the controlled and the controlling processes are often out of synch with one another.

A control action generated one moment may be deemed inappropriate at the next as new information becomes available. To simply generate a sequence of actions and expect that the sequence can be carried out without modification is for many problems absurd. In asking directions in Boston, a local may tell you to turn left on Commonwealth Avenue and follow it for three blocks until you get to Massachusetts Avenue, but if you find four fire trucks tying up traffic on Commonwealth Avenue, then you would be well advised to disregard their directions and find an alternative route. There was nothing wrong with the directions provided given what was known at the time they were solicited, but knowledge changes over time and such changes should be taken into account when deciding how to act.

Of course, the preceding paragraph shouldn't be taken as an argument against planning; we've already seen that path planning can lead to improved performance in certain circumstances. What we have to beware of is blindly executing plans in the face of information that warns against their use. The traditional notion of a plan as a sequence of actions has to be rethought. Plans should be interpreted as suggestions about how to behave. Some suggestions require a long time to generate, but the processes that they are designed to help control may proceed at a similarly slow pace. In real-world problems, there are any number of processes that require some

amount of control. Some processes proceed slowly and require attention only at widely-spaced intervals (*e.g.,* the pipe-laying process discussed earlier). Other processes are faster paced and require almost constant attention (*e.g.,* pedestrian traffic). The trick is to deal effectively with the fast-paced processes (*e.g.,* steer clear of pedestrians and stop at appropriate traffic signals) while at the same time directing behavior so as to take into account suggestions regarding the slower processes (*e.g.,* avoid routes that are believed to be obstructed by construction) and suggestions generated offline as it were regarding faster-paced processes (*e.g.,* if you see a ball rolling out into the street, brake hard as a child may be following closely behind).

In the following, it will be useful to separate out two kinds of control algorithm. One that generates suggestions concerning certain low-level behaviors and that is likely to perform out of synch with the processes whose behavior it is meant to influence, and a second that is closely tied to the processes that it is meant to influence. The distinction is artificial; it serves primarily to identify two distinct mind sets that have to be merged in order to develop a coherent theory of control. To provide a label for the two kinds of control and identify the source for the corresponding mind sets, we call the first *high-level planning* and the second *low-level control*. An example of a high-level planning algorithm would be a path planning algorithm designed to influence the movement of the robot. An example of a low-level control algorithm would be the algorithm that directs the speed and heading of the robot as it traverses the city streets avoiding obstacles and maneuvering around corners.

One possible architecture for a system integrating high-level planning and low-level control might consist of two components: a tactical component that determines what to do at the next instant, and a strategic component that attempts to mediate the behavior of the tactical component by imposing constraints on the behavior of the low-level systems. It is up to the tactical component to interpret the constraints provided by the strategic component so as to adjust the behavior of the low-level systems while at the same time maintaining real-time performance.

In this chapter, we are primarily interested in what we have called low-level control. Toward the end of this chapter, however, we begin to address high-level control issues as prologue to the next chapter which will deal almost exclusively with high-level strategic planning. Now, we draw upon the disciplines of control theory and control systems engineering to develop some terminology and explore techniques that will be used in subsequent chapters.

4.2 Controllability

Consider the following time-invariant discrete-time dynamical system.

$$\begin{aligned} x(k+1) &= f(x(k), u(k)) \\ y(k) &= g(x(k)) \end{aligned}$$

The state transition function, f, completely determines the state of the system at time $k+1$ given the state and the input at time k. Initially, we assume that the state of the system is directly observable, and so the output function, g, is defined

$$g(x(k)) = x(k).$$

In solving a particular control problem, we are interested in generating appropriate inputs so as to constrain the behavior of the dynamical system. In Chapter 1, we introduced a general formulation of the control problem, representing the behavior of a dynamical system in terms of the set of possible state-space trajectories,

$$H_X \stackrel{\triangle}{=} \{h_X : T \to X\}.$$

In this formulation of the problem, the desired behavior of the system is specified in terms of a *goal* set,

$$G \subset H_X.$$

There are several special cases of this formulation that we consider in the following sections.

In the *servo problem*, we are given a *reference trajectory*, and expected to repeat or *track* that trajectory as closely as possible. In the *set-point regulation problem*, the objective is for the system to achieve and maintain a particular state or set of states starting from any initial state. In the terminology of Chapter 2, we wish to find some input function $v \in \{v : T \to U\}$ so that for any initial time $\tau \in T$ and initial state $x(\tau) \in X$ there exists $t > \tau$ such that for all $t' > t$ we have

$$f(x(t'), v(t')) \in C,$$

where $C \subset X$ is the set of target states.

We can generalize on our formulation of the set-point regulation problem to restrict not only the final states of the system, but the intermediate

states as well, thereby restricting the motions (state space trajectories) of the system. For instance, we might require that the system avoid a certain set of states, by stipulating that for all $t > \tau$ we have

$$f(x(t), \upsilon(t)) \notin Q,$$

where $Q \subset X$ is the set of states to avoid and $C \cap Q = \emptyset$.

Among the qualitative properties of dynamical systems and their controllers, the following notion of *controllability* is particularly relevant to the set-point regulation problem. An event $\langle \tau, x \rangle$ in the phase space defined by $T \times X$ is said to be *controllable with respect to a set of target states*, $C \subset X$, if and only if there is some time t and some input υ which moves $\langle \tau, x \rangle$ into the set $\{t : t \geq \tau\} \times C$. A dynamical system is *completely controllable with respect to* C if and only if every event in $T \times X$ is controllable with respect to C. This notion of complete controllability with respect to a set of target states provides necessary and sufficient conditions for there being a solution to the set-point regulation problem.

As was mentioned in Chapter 2, one of the best developed areas of modern control theory concerns the analysis of dynamical systems that can be modeled as linear multivariable systems. In this chapter, we illustrate the power of linear systems theory by defining three important qualitative properties of dynamical systems, and stating simple mathematical criteria for these properties to be satisfied.

We begin with the notion of controllability. Criteria for controllability are generally specific to a particular method of modeling dynamical systems. In general, we are interested in whether or not it is possible to transfer any state $x(t_0) \in X$ to any other state in X in a finite amount of time $t_1 - t_0$ where $t_0 < t_1$ by appropriately choosing $u(t)$ for $t_0 \leq t \leq t_1$. If such arbitrary transfers are possible, we say that the system is *completely controllable* (no restriction to a particular set of target states).

Consider the following linear time-invariant system represented by

$$\begin{aligned} \dot{\mathbf{x}}(t) &= A\mathbf{x}(t) + B\mathbf{u}(t) \\ \mathbf{y}(t) &= C\mathbf{x}(t) \end{aligned}$$

where $\mathbf{x}$ is the n-dimensional state vector, $\mathbf{u}$ is the p-dimensional input vector, $\mathbf{y}$ is the q-dimensional output vector, and A, B, and C are, respectively, $n \times n$, $n \times p$, and $q \times n$ real constant matrices. There are a number of relatively simple mathematical conditions for such a system being completely controllable. One of the simplest is provided by the following theorem which

is stated here without proof (see Chen [11] or Gopal [17] for proofs and related theorems).

Theorem 1 *The system is completely controllable if and only if the rank*[1] *of the* $n \times np$ controllability matrix, $[B|AB|\cdots|A^{n-1}B]$, *is* n.

As a simple example, the dynamical system for the single-degree-of-freedom robot introduced in Chapter 2 with state equation,

$$\dot{\mathbf{x}}(t) = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \mathbf{x}(t) + \begin{bmatrix} 0 \\ 1/M \end{bmatrix} \mathbf{u}(t),$$

is completely controllable since the rank of its controllability matrix,

$$[B|AB] = \begin{bmatrix} 0 & 1/M \\ 1/M & 0 \end{bmatrix},$$

is 2. However, the system described by

$$\dot{\mathbf{x}}(t) = \begin{bmatrix} 0 & C_1 \\ C_2 & 0 \end{bmatrix} \mathbf{x}(t) + \begin{bmatrix} 1 \\ 1 \end{bmatrix} \mathbf{u}(t),$$

has a controllability matrix,

$$[B|AB] = \begin{bmatrix} 1 & C_1 \\ 1 & C_2 \end{bmatrix},$$

indicating that the system is controllable only if $C_1 \neq C_2$.

There are other similarly concise and equivalent conditions stated in the literature. Both Chen [11] and Gopal [17] provide similar results for linear time-varying systems, as well as constructive proofs that identify the appropriate input functions. It is testimony to the power of linear systems theory that such precise conditions can be stated for such a general class of dynamical systems.[2]

[1] The *rank* of an $n \times m$ rectangular matrix, A, is defined as the maximum number of linearly independent column vectors, or, equivalently, the order of the largest square array whose determinant is non-zero, where the square array is obtained by removing rows and columns from A.

[2] As was noted in Chapter 2, it is standard practice in engineering control systems to model real-world nonlinear systems using linear approximations. Since small perturbations of the elements of the matrices A and B may signal the difference between controllability and its lack, it should be noted that statements of system controllability must be carefully weighed in the process of design.

It should be noted that the above stated notion of controllability places no constraint on the input (controller) or on the trajectory followed by the system. A system may be determined as uncontrollable by the above criterion, while being controllable in most practical respects. For instance, the system may move to any given state from all initial states that will arise in practice. As another example, we may not care about certain components of the state vector; it may be that we are only concerned with controlling the output of the system.

To investigate further the notion of controllability, we consider some examples of dynamical systems that can be represented in terms of finite-state automata. These dynamical systems are referred to as *discrete event systems* in the literature [30]. We represent a discrete event system as an automaton, $G = (U, X, f, x_0)$, where, in keeping with our previous notation, U is the set of inputs (think of U as a set of primitive events), X is the set of states, $f : U \times X \to X$ is the state transition function, and x_0 is the initial state.

We partition U into two sets: U_c, the set of *controllable* events, and U_u, the set of *uncontrollable* events. An *admissible control* for such a dynamical system consists of a subset $\gamma \subseteq U$ such that $U_u \subseteq \gamma$. Let $\Gamma \subseteq 2^U$ represent the set of all admissible controls. If $\gamma \in \Gamma$ and $u \in \gamma$, we say that u is *enabled* by γ, otherwise we say that it is *disabled*. A controller for a given dynamical system is specified as a map,

$$\eta : X \to \Gamma.$$

The idea is that disabled events are prevented from occurring and enabled events are allowed to occur if permitted by the underlying dynamics. The stipulation that $U_u \subseteq \gamma$ for all $\gamma \in \Gamma$ captures the intuition that the controller cannot prevent the uncontrolled events from occurring if the dynamics dictates otherwise. An issue arises regarding what happens if all of the events for a given state are disabled. We resolve the issue by simply requiring that the controller ensure that for any state there is at least one enabled event for which the transition function is defined; the system can remain in the same state only if that is permitted by the dynamics.

Consider the dynamical system depicted in Figure 4.5 in which $U = \{a, b, c\}$, $X = \{0, 1, 2\}$, $x_0 = 0$, and f is defined so that

$$(0, a) \mapsto 1, (0, b) \mapsto 2, (1, c) \mapsto 2, \text{ and } (2, a) \mapsto 2.$$

Let $U_c = \{a, b\}$ and suppose that we wish to design a controller that achieves

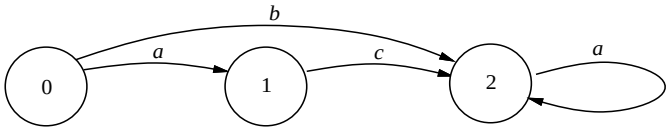

Figure 4.5: A dynamical system represented as a finite-state automaton

$\{2\}$ while avoiding $\{1\}$. The controller defined by

$$0 \mapsto \{b\} \text{ and } 2 \mapsto \{a\}$$

will suffice to do exactly what we want. The same controller will work if $U_c = \{a\}$. However, if we have $U_c = \{b\}$, then there is no controller satisfying the requirements given.

There is an alternative approach to characterizing the behavior of discrete event systems modeled as finite-state automata. In formal language theory, a finite-state automaton can be viewed as a generator for a language. Let U^* denote the set of all finite strings of elements of the set U. A subset $L \subseteq U^*$ is called a *language* over U. The automaton described above is a generator for the language

$$L = ba^* + aca^*,$$

indicating the union of the set of strings consisting of b followed by a finite number of a's, and the set of strings consisting of a followed by c followed by a finite number of a's. Instead of asking if we can design a controller that achieves $\{2\}$ while avoiding $\{1\}$, we ask if we can design a controller for the automaton so that it generates the language $L' = ba^* \subseteq L$.

Ramadge and Wonham [30] define a *supervisor* for a discrete event system as a map

$$\eta : L \to \Gamma,$$

where L is the language (or *behavior*) generated by the discrete event system. The *prefix closure* of $L \subseteq U^*$ is that subset $\bar{L} \subseteq U^*$ defined by

$$\bar{L} = \{u : uv \in L \text{ for some } v \in U^*\}.$$

A language $K \subseteq L$ is said to be *controllable* with respect to a given discrete event system if

$$\bar{K}U_u \cap L \subseteq \bar{K},$$

where $\bar{K}U_u$ represents the set of all strings consisting of a string from the prefix closure of K concatenated with an event from U_u. In [30], Ramadge

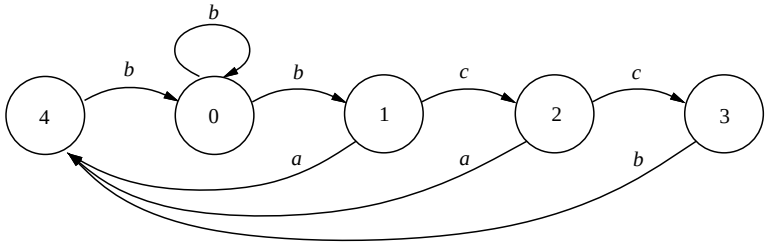

Figure 4.6: One component of a product system

and Wonham prove the following, thus providing necessary and sufficient conditions for the existence of supervisors for discrete event systems.

Theorem 2 *For any discrete event system A with closed behavior L and any subset $K \subseteq L$, there exists a supervisor that serves to restrict A to exactly K if and only if $\bar{K} = K$ and K is controllable.*

In some cases, it is convenient to represent a dynamical system as a collection of finite-state automata loosely coupled through the state space resulting from taking the cross product of the state spaces for the individual automata. As an example, suppose that we wish to model a collection of n identical chemical processes. Each individual process is modeled by an automaton $G_i = (U_i, X_i, f_i, x_{0_i})$ where the ith automaton is defined by $U_i = \{a_i, b_i, c_i\}$, $X_i = \{0_i, 1_i, 2_i, 3_i, 4_i\}$, $x_{0_i} = 0$, and f_i is as indicated in Figure 4.6. Let $U_{c_i} = \{a_i, c_i\}$. Suppose that all n processes run independently of one another with one important exception: state 4 involves the use of a piece of equipment with limited capacity such that only one process can be in state 4 at a time. We wish to design a controller that will guarantee this. Note that once a process enters state 1, we can exercise some control over *when* it enters state 4, but we can only delay this event, we cannot prevent it from happening.

To represent the combined behavior of the collection of processes, we define the *product generator* $G = \{U, X, f, x_0\}$ where $U = \cup_{i=1}^{n} U_i$, $X = \Pi_{i=1}^{n} X_i$, $U_c = \cup_{i=1}^{n} U_{c_i}$, $x_0 = (x_{0_1}, x_{0_2}, \ldots, x_{0_n})$ and for each $u \in U_i$ we have

$$f(u, (x_1, x_2, \ldots, x_i, \ldots, x_n)) = (x_1, x_2, \ldots, f_i(u, x_i), \ldots, x_n)).$$

The objective is to build a controller for G such that at most one of the chemical processes is in the state requiring the piece of equipment at any given point in time.

In the worst case, all of the processes will simultaneously arrive at state 1 in their respective state spaces. At this point, exactly one process can transition to state 4, while the $n-1$ remaining processes are forced to enter state 2. The same simple analysis applied to state 1 can be applied to state 2 with the conclusion that $n-2$ processes are forced to enter state 3. The controller has no control over the processes in state 3, and hence we conclude that there exists a controller for the product system if and only if $n \leq 3$.

Discrete event systems can be used to model manufacturing systems, communication networks, vehicular traffic problems, and a variety of other dynamical systems requiring coordination and control. In addition to answering mathematical questions concerning the existence of supervisors, the current theory provides constructive methods for realizing certain classes of supervisors. In the best circumstances, these methods require time and storage polynomial in the size of the state space. For practical problems, one generally has to be clever in searching the space of possible controllers for one that satisfies the domain constraints.

4.3 Observability

So far, we have had little to say about the role of the system output function. In fact, we initially assumed that $y(t) = g(x(t)) = x(t)$, so that the state of the system was directly observable as output. In general, the entire system state will not be directly observable. If the controller requires either the entire system state vector or specific components of this vector, then an additional module has to be added to the control system in order to recover the state by observing the system output. Such modules are generally referred to as *observers*. If the function g is known and invertible, then the construction of an observer is trivial. Generally, g is not invertible and the state has to be recovered by observing the output of the system over some interval of time. In the following, we consider a notion of observability which, at least in the case of linear multivariable systems, turns out to be closely related to controllability.

A system is said to be *completely observable* if it is possible to identify any state $x(t_0) \in X$ by observing the output $y(t)$ for $t_0 \leq t \leq t_1$ where $t_0 < t_1$. The problem stated is traditionally called the *observation* problem, but it is just one of several so-called *state-determination* problems. The observation problem involves determining the state from future outputs. There is a related problem called the *reconstruction* problem that involves identi-

fying the state from past outputs: identify the state $x(t_1) \in X$ by observing the output $y(t)$ for $t_0 \leq t \leq t_1$ where $t_0 < t_1$. As in the case of controllability, there are simple mathematical criteria for observability in linear multivariable systems (see Chen [11] or Gopal [17] for proofs and equivalent conditions).

Theorem 3 *The system is completely observable if and only if the rank of the* $nq \times n$ observability matrix,

$$\begin{bmatrix} C \\ CA \\ \vdots \\ CA^n - 1 \end{bmatrix} \text{ is } n.$$

Given the similarity of the statement of Theorems 1 and 3 one might suspect that there is a rather deep relationship between controllability and observability for linear multivariable systems. It would be particularly convenient if one could prove that a system is observable if and only if it is controllable. This happens to be true in a somewhat convoluted mathematical sense as we see in the following theorem.

Theorem 4 *(The Principle of Duality) The system represented by*

$$\begin{aligned} \dot{\mathbf{x}}(t) &= A\mathbf{x}(t) + B\mathbf{u}(t) \\ \mathbf{y}(t) &= C\mathbf{x}(t) \end{aligned}$$

is controllable (observable) at time t_0 *if and only if the dual system represented by*

$$\begin{aligned} \dot{\mathbf{z}}(t) &= -A'\mathbf{z}(t) + C'\mathbf{v}(t) \\ \mathbf{w}(t) &= B'\mathbf{z}(t) \end{aligned}$$

is observable (controllable) at t_0*, where the prime (*e.g., B'*) indicates matrix transposition, and the second system (called the* adjoint*) is mathematically closely related to the first.*

One practical consequence of Theorem 4 is that once you have constructed a controller (observer), you have done all the necessary work required to construct the associated observer (controller); the algorithms required for one task are almost identical to the algorithms required for the

other task. It is also interesting to note that observability and controllability in linear systems can be considered independently. The two problems of building a controller and building an observer can be pursued independently of one another. The two problems are said to be *separable* and the associated dynamical system is said to satisfy the *separation property*. This separation property does not hold in general.

Results similar to that of Theorem 4 hold for linear systems corrupted with Gaussian noise. In Chapter 6, we consider the problem of building a deterministic regulator (controller) and a stochastic estimator (observer) for dynamical systems modeled as linear systems corrupted with Gaussian noise. It turns out that these two problems are also separable; you can construct an optimal control system by coupling an optimal deterministic regulator to an optimal stochastic estimator.

It should be emphasized that the notion of observability introduced in this section is quite strong. In general, a controller need not reconstruct the entire system state in order to provide satisfactory performance for a given control problem. In many cases, the task of reconstructing the entire system state would impose a significant computational burden. Practically speaking, we are interested in *demand-driven observation strategies* that allocate resources to measurement and interpretation in keeping with the immediate demands on the system. The task-based planning methods presented in Chapter 5 employ this sort of demand-driven observation strategies.

4.4 Stability

When we first introduced the notion of controllability in Section 4.2, we were interested in the ability to first achieve a given state or set of states in a finite amount of time, and then maintain the system in that state of set of states for all time hence. When we subsequently considered controllability criteria for linear systems, we dropped the latter requirement. In many applications, however, it is not enough for a controller to simply move the system to a particular state. Neither is it reasonable to expect that the controller maintain a given state in the face of arbitrary disturbances or perturbations of the dynamical system. Stability is a property of dynamical systems which implies that small changes in input or initial conditions do not result in large changes in system behavior. Stability is not a prerequisite for being able to control a system, but it makes the task of designing a control system somewhat easier. The system describing the inverted pendulum presented

in Chapter 2 is not stable by the criteria that we will present shortly, but it is controllable. The concept of stability introduced in the following is attributed to the Russian mathematician A. M. Lyapunov.

We will be concerned with the same linear multivariable system introduced earlier.

$$\begin{aligned} \dot{\mathbf{x}}(t) &= A\mathbf{x}(t) + B\mathbf{u}(t) \\ \mathbf{y}(t) &= C\mathbf{x}(t) \end{aligned}$$

Let $\mathbf{u}(t) = \mathbf{u}_c$ be any constant input. If there exists a point $\mathbf{x}_e \in \mathbf{R}^n$ such that

$$A\mathbf{x}_e + B\mathbf{u}_c = 0,$$

then $\mathbf{x}_e$ is said to be an *equilibrium point* of the system corresponding to the input $\mathbf{u}_c$. We assume that the system has only one equilibrium point, and, without loss of generality, take the origin of the state space to be that equilibrium point. Finally, we consider only the case in which $\mathbf{u}_c = 0$ so that

$$\dot{\mathbf{x}}(t) = A\mathbf{x}(t).$$

This system is *stable in the sense of Lyapunov* at the origin if, for every $\epsilon > 0$, there exists $\delta > 0$ such that $\|\mathbf{x}(t_0)\| \leq \delta$ implies $\|\mathbf{x}(t)\| \leq \epsilon$ for all $t \geq t_0$, where $\|\mathbf{x}\|$ denotes the Euclidean norm for a vector $\mathbf{x}$ of n components $x_1, x_2, \ldots, x_n$ defined by

$$\|\mathbf{x}\| = (x_1^2, x_2^2, \ldots, x_n^2)^{1/2}.$$

The hyperspherical region defined by the set of all points such that $\|\mathbf{x}\| \leq \epsilon$ serves to ensure a bound on the system response.

We say that the above system is *asymptotically stable* at the origin if

1. it is stable in the sense of Lyapunov, and
2. there exists a real number $r > 0$ such that

$$\|\mathbf{x}(t_0)\| \leq r \text{ implies } \mathbf{x}(t) \to 0 \text{ as } t \to \infty.$$

The asymptotic stability of a linear multivariable system can be determined using a relatively simple mathematical test provided in the following theorem (see [17] for proof).

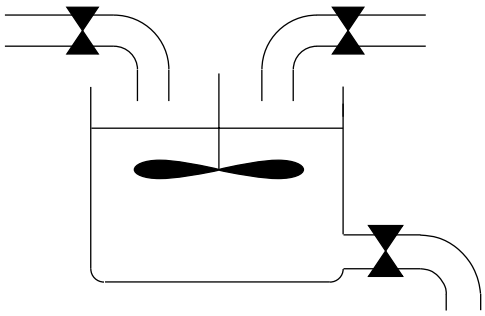

Figure 4.7: Controlling the concentration of chemicals in a mixing tank

Theorem 5 *The system described by the state equation,*

$$\dot{\mathbf{x}} = A\mathbf{x}(t) + B\mathbf{u}(t),$$

is asymptotically stable if and only if all of the eigenvalues of the matrix A have negative real parts.

Recall that the eigenvalues of a matrix A correspond to those values of λ such that $\mathrm{Det}(\lambda I - A) = 0$, where I is the identity matrix and $\mathrm{Det}(M)$ indicates the determinant of the matrix M. One particularly convenient advantage of the stability test introduced in Theorem 5 is that it does not require one to solve the system state equations.

As a simple example of an asymptotically stable system, consider the following state equation,

$$\dot{\mathbf{x}}(t) = \begin{bmatrix} C/2 & 0 \\ 0 & C \end{bmatrix} \mathbf{x}(t) + B\mathbf{u}(t),$$

where $C < 0$ and B is not important for our analysis. This state equation can be used to model a process in which two chemical solutions are mixed in a tank (see Figure 4.7). We assume that a stirring mechanism maintains a uniform concentration of the two chemicals throughout the tank. In attempting to control the concentration of the mixture flowing out of the tank, the controller varies the rate at which the chemical solutions flow into the tank through their respective supply pipes (see [17] for details).

In the case of the chemical-mixing system, the eigenvalues correspond to solutions of the following equation,

$$\mathrm{Det}\left(\begin{bmatrix} -C/2 + \lambda & 0 \\ 0 & -C + \lambda \end{bmatrix} \right) = \frac{C^2}{2} - \frac{3C\lambda}{2} + \lambda^2 = 0,$$

referred to as the *characteristic* equation. In this case, the characteristic equation has two solutions, $C/2$ and C, which are negative given the statement of the problem, indicating that the system is asymptotically stable.

In the case of the system corresponding to the inverted pendulum described in Chapter 2,

$$\dot{\mathbf{x}}(t) = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & -0.5809 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 4.4532 & 0 \end{bmatrix} \mathbf{x}(t) + \begin{bmatrix} 0 \\ 0.9211 \\ 0 \\ -0.3947 \end{bmatrix} \mathbf{u}(t),$$

the characteristic equation,

$$\text{Det}\left(\begin{bmatrix} \lambda & 1 & 0 & 0 \\ 0 & \lambda & -0.5809 & 0 \\ 0 & 0 & \lambda & 1 \\ 0 & 0 & 4.4532 & \lambda \end{bmatrix} \right) = \lambda(\lambda(\lambda^2 - 4.4532)) = 0,$$

has one positive solution, $+\sqrt{4.4532}$, indicating that the corresponding system is not asymptotically stable.

Given that balancing an inverted pendulum for any length of time takes a certain amount of coordination, it might seem intuitively obvious that the dynamical system corresponding to the cart and the pendulum is not asymptotically stable. However, asymptotic stability can also run counter to intuition. In particular, asymptotic stability does not distinguish between the single-degree-of-freedom robot and the cart-and-pendulum system. In the case of the single-degree-of-freedom robot, the characteristic equation,

$$\text{Det}\left(\begin{bmatrix} \lambda & 1 \\ 0 & \lambda \end{bmatrix} \right) = \lambda^2 = 0,$$

has a single non-negative solution, 0, indicating that the corresponding system is not asymptotically stable. This result seems more plausible when you consider that the position of the robot can diverge arbitrarily far from the starting position given a constant velocity.

Intuitively, you might think that it would be easier to control the position of the single-degree-of-freedom robot than the angle of the pendulum on the cart. Indeed, there is a sense in which this is true. In designing a control system for an unstable dynamical system, an engineer adds components that expend energy to stabilize the system. These components have the effect of

shifting the eigenvalues of the composite system (the original system plus the added components) to the left (*i.e.,* making them more negative). Since the only eigenvalue of the single-degree-of-freedom robot is 0, it should take less energy to stabilize the single-degree-of-freedom robot than it does the inverted pendulum. In practice, this need not always be the case, but similar, though somewhat more sophisticated arguments form the justification for many subtle tools in the repertoire of the control engineer.

Before we leave the subject of stability, it is worth mentioning one particularly useful technique referred to as the *root-locus method* developed by W. R. Evans for investigating the stability of linear systems. The root-locus method is most closely associated with what is called classical control theory which, as was mentioned in Chapter 2, is based primarily upon the use of the Laplace transform and analysis in the frequency domain.

Many control systems have a single input variable and a single output variable. The input is referred to as a *reference signal* indicating the desired value for the output or controlled variable. The *transfer function* of such a control system is defined to be the ratio of the Laplace transform of the input variable to the Laplace transform of the output variable. Consider the spring-mass-dashpot system described in Chapter 2, and suppose that we allow an external force to act on the block. The equation of motion of the block is

$$M\frac{d^2x}{dt^2} + C\frac{dx}{dt} + Kx = u(t),$$

where the output of the system is defined to be x and the input is u. The Laplace transform of the equation of motion is

$$Ms^2X(s) + CsX(s) + KX(s) = U(s),$$

assuming the initial conditions,

$$x(0) = x_0, \qquad \frac{dx(0)}{dt} = 0.$$

The transfer function for the above system is

$$T(s) = \frac{X(s)}{U(s)} = \frac{1}{Ms^2 + Cs + K}.$$

By analyzing the system's *poles* (the roots of the denominator or *characteristic equation* of the transfer function) and *zeros* (the roots of the numerator of the transfer function), one can tell a great deal about the transient

response characteristics of the control system. For instance, it is well known [12] that, for a system to be asymptotically stable, it is necessary and sufficient that all of the poles of the system transfer function have negative real parts.[3]

Figure 4.8 shows the relation between the poles of the transfer function for a second order system and the system's corresponding behavior in the time domain. In Figure 4.8, each plot on the left hand side indicates one particular placement of the poles in the complex s-plane, and the corresponding plot on the right indicates the resulting performance in the time domain. This method of analyzing control systems by determining the placement of poles is known as the *root-locus method*.

Not surprisingly, there is a close connection between the frequency- and time-domain methods for determining stability. In the case of multiple-input, multiple-output systems, we have to generalize on the notion of a transfer function, which is defined only for single-input, single-output systems. The *transfer matrix* of a linear multivariable dynamical system as introduced in the beginning of this section is uniquely defined by

$$T(s) = C\,(sI - A)^{-1}\,B,$$

where I is the identity matrix. It should be noted that there is information lost in this conversion. In particular, the state and input equations specify the internal state as well as the input/output behavior of the dynamical system, whereas the transfer matrix only specifies the latter. It turns out that the poles of the system represented by the transfer matrix are exactly the eigenvalues of the matrix A [35].

One convenient property of transfer functions and transfer matrices is that, in certain cases, such representations can be obtained experimentally by subjecting the dynamical system to sinusoidal inputs and measuring the steady-state response. The close connection between frequency- and time-domain methods allows the engineer to shift back and forth between these two perspectives as the problem dictates. It should be noted, however, that the connection between frequency- and time-domain notions of stability can only be made for linear time-invariant systems; otherwise, the notion of frequency domain is not even well defined.

Stability can simplify the design of control systems; it is not, however, a prerequisite for control. The linear system for the inverted pendulum is

[3]The Laplace variable is a complex variable and hence the roots of the characteristic equation are generally complex as well.

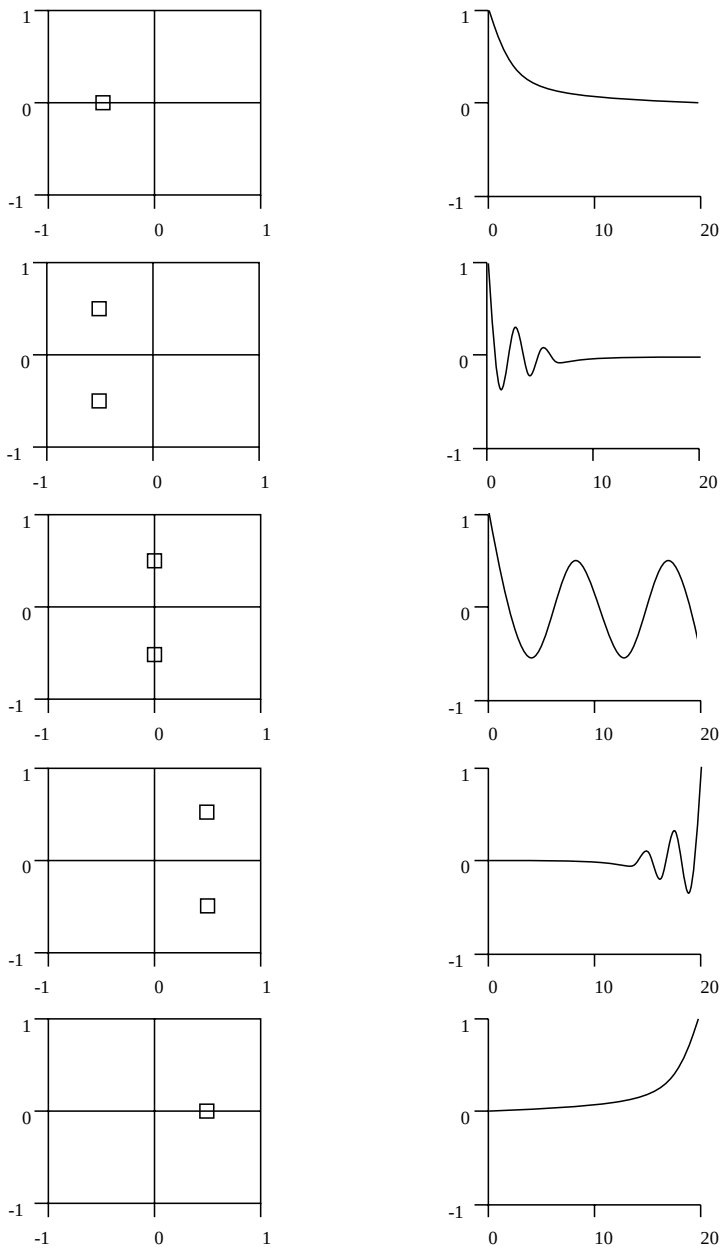

Figure 4.8: The connection between pole placement in the complex s-plane and performance in the time domain.

not stable, but it is controllable. If we are designing a device, it is generally worthwhile to design it in such a way that its corresponding dynamical system is stable. In cases in which the plant (environment) is given, we have little choice and must proceed whether or not the associated system is stable.

4.5 Optimality

In previous sections, we have stressed primarily the qualitative properties of dynamical systems (*e.g.,* controllability, observability, and stability). With the exception of criteria concerning whether or not a given controller can achieve a particular state from some arbitrary initial state, we have had very little to say about the performance of a control system. In this section, we consider control problems in which some quantitative measure (or *index*) of performance is provided. It is natural within this context to consider problems of *optimal control* that involve maximizing or minimizing such a performance index.

In describing optimal control problems, we generally restrict our attention to some restricted interval of time, either continuous, $[t_0, t_1]$, or discrete, $[1, n]$. The behavior of the dynamical system is described by either a set of differential equations,

$$\dot{\mathbf{x}}(t) = f(x(t), u(t)), \text{ restricted to } t_0 \leq t \leq t_1,$$

in the continuous case, or a set of difference equations,

$$x(k+1) = f(x(k), u(k)), \text{ restricted to } 1 \leq k \leq n,$$

in the discrete case. In addition to the model for the dynamical system, it is often convenient to place restrictions on both the inputs (*e.g.,* you might want to place a bound on control torques to keep the cost of servo motors within budget constraints) and the outputs (*e.g.,* you may want to restrict the trajectories of a robot arm to a confined work space). The input restrictions define a set of *admissible controls* (see the discussion in Section 4.2 on admissible controls for discrete events systems). Finally, it will be necessary to formulate a performance index in terms of a scalar *value* function, V.

The choice of performance index is largely subjective, but generally a particular application will suggest something reasonable. In some cases, it may make sense simply to minimize time,

$$\mathrm{V} = \int_{t_1}^{t_2} 1 \, dt = t_1 - t_0.$$

In other cases, there may be an obvious cost function, $c(x, u)$, such as the amount of fuel or other resource spent,

$$\mathrm{V} = \int_{t_1}^{t_2} c(x(t), u(t))\, dt.$$

For the set-point regulation and servo problems a good measure of performance is the squared error,

$$\mathrm{V} = \int_{t_1}^{t_2} (x(t) - x^*(t))^2\, dt,$$

where $x^*(t)$ is the desired state at time t. The squared error index is an example of a quadratic performance index.[4] More generally, the performance index is defined as

$$\mathrm{V} = h(x(t_1)) + \int_{t_1}^{t_2} g(x(t), u(t))\, dt.$$

where h and g are scalar functions meant to capture the value of the terminal state and the state/input trajectory respectively. The problem of designing optimal controls consists of finding an admissible control that minimizes (maximizes) the performance index, V.

There are two classes of optimal control problems involving linear multivariable systems for which general results have been obtained. The first class involves the use of a quadratic performance index as in the example of the minimum squared error index, and includes optimal versions of the linear set-point regulation and servo problems. In the second class of problems, the objective is to minimize the time required to drive the system to a desired state. In both of these two classes of problems, optimal controllers can make use of feedback, which, as covered in the next section, provides for more robust control in the presence of external disturbances and errors in modeling. The optimal linear minimum-time controller is of a particularly simple form; it can be viewed as a function that simply switches between the extreme values dictated by the class of admissable controls. A controller that operates at a constant level either in one mode or another (*e.g.*, $\forall\, t,\ u(t) \in \{-1, 0, 1\}$) is called a *bang-bang* controller.

[4]The function $\mathrm{V} = \int f(t)\, dt$ is a quadratic performance index if $f(t) = \mathbf{x}(t)' A \mathbf{x}(t)$ where A is an $n \times n$ matrix with $a_{ij} \in \mathbf{R}$ and $\mathbf{x} \in \mathbf{R}^n$.

Most of the work on optimal control builds upon basic techniques in the calculus of variations [15]. The method of Lagrange multipliers[5] for finding extrema of functions subject to constraints is one technique from the calculus of variations that students typically encounter in college calculus courses.

As a simple example illustrating the use of the method of Lagrange multipliers, let $\varphi(x, y)$ and $\varsigma(x, y)$ be functions of two variables. The object is to find values of x and y that maximize (or minimize) the *objective* function $\varphi(x, y)$ while at the same time satisfying the constraint equation, $\varsigma(x, y) = 0$. We replace $\varphi(x, y)$ with an auxiliary function of three variables called the *Hamiltonian* function, $\Phi(x, y, \lambda)$, defined as

$$\Phi(x, y, \lambda) = \varphi(x, y) + \lambda\varsigma(x, y).$$

The new variable, λ, is called a *Lagrange multiplier*. The *Euler-Lagrange multiplier theorem* [15] implies that, if we locate all points (x, y, λ) where the partial derivatives of $\Phi(x, y, \lambda)$ are all 0, then among the corresponding (x, y) we will find all of the points at which the function $\varphi(x, y)$ will have a constrained extremum.

In the method of *Lagrange multipliers*, we solve for x, y, and λ in the equations formed by setting the partial derivatives to 0:

$$\frac{\partial\Phi}{\partial x} = 0, \quad \frac{\partial\Phi}{\partial y} = 0, \text{ and } \quad \frac{\partial\Phi}{\partial\lambda} = 0.$$

Since $\partial\Phi/\partial\lambda = \varsigma(x, y)$, if we find a solution (x, y, λ) to the above three equations, the constraint equation $\varsigma(x, y) = 0$ will automatically be satisfied.

To illustrate how to apply the method of Lagrange multipliers to problems in optimal control, consider the discrete-time system,

$$x_{k+1} = f(x_k, u_k),$$

and the performance index defined by

$$\mathrm{V} = \sum_{k=1}^{n} g(x_k, u_k),$$

[5]Leonard Euler (1707–1783) developed the basic approach to solving constrained extremum problems. Joseph Lagrange (1736–1813) studied Euler's approach and worked out the details for some important special cases. The basic method is generally referred to as the method of *Lagrange multipliers*, but in some texts the equations are referred to as the *Euler-Lagrange equations* recognizing Euler's contributions.

where we have changed our notation somewhat, $x(k) = x_k$ and $u(k) = u_k$, to simplify subsequent equations. The only constraint that we impose is that the optimal solution obey the state difference equations. We enforce this constraint by augmenting the performance index as follows

$$\mathrm{V}' = \sum_{k=1}^{n} \left[g(x_k, u_k) + \lambda_{k+1}(f(x_k, u_k) - x_{k+1}) \right].$$

We define the Hamiltonian somewhat differently from above as

$$\Phi_k = g(x_k, u_k) + \lambda_{k+1} f(x_k, u_k),$$

so that we can rewrite the augmented performance index as

$$\mathrm{V}' = \sum_{k=1}^{n} \left[\Phi_k - \lambda_{k+1} x_{k+1} \right].$$

By the Euler-Lagrange multiplier theorem, the change in the total derivative, $d\mathrm{V}'$, defined as

$$d\mathrm{V}' = \sum_{k=1}^{n} \left[\left(\frac{\partial \Phi_k}{\partial x_{k+1}} - \lambda_k \right) dx_k + \left(\frac{\partial \Phi_k}{\partial \lambda_{k+1}} - x_k \right) d\lambda_k + \frac{\partial \Phi_k}{\partial u_k)} du_k \right],$$

should be zero at a constrained minimum. As a consequence, the necessary conditions for a constrained minimum are defined by

$$x_{k+1} = \frac{\partial \Phi_k}{\partial \lambda_{k+1}} = f(x_k, u_k), \quad 1 \leq k \leq n,$$

referred to as the *state equations*,

$$\lambda_k = \frac{\partial \Phi_k}{\partial x_{k+1}}, \quad 1 \leq k \leq n,$$

referred to as the *costate equations*,

$$0 = \frac{\partial \Phi_k}{\partial u_k)}, \quad 1 \leq k \leq n,$$

referred to as the *stationary conditions*, and, finally, we require that x_1 be the initial state. The state and costate equations are coupled difference equations, and together they define a two-point boundary value problem.

In the special case of linear systems with quadratic performance indices, numerical solutions can be obtained rather easily.[6]

In general, it can be quite difficult to solve the two-point boundary value problems resulting from Lagrange multiplier formulations. However, in some cases, finding global maxima or minima can still be achieved by searching the space defined by the variational variables (*e.g.*, x and y in the case of minimizing $\varphi(x, y)$). One approach is to use numerical methods to solve the original equations relating to the performance index and constraints, and then search the resulting surface looking for global extrema. The *gradient*, defined as

$$\nabla\varphi = \begin{bmatrix} \partial\varphi/\partial x \\ \partial\varphi/\partial y \end{bmatrix}$$

in the case of $\varphi(x, y)$, is used to guide search in a method that proceeds by taking many small steps, each one in the direction indicated by the (negated) gradient. This search method is called *gradient descent*. If the surface has a single (global) minimum, then gradient descent search is guaranteed to find it. If, however, there are many local minima, as is often the case, then one has to be a lot more clever in directing the search. It is this aspect of optimal control involving search in a space of possible controls that primarily interests us in this section.

In some cases, we can resort to exhaustive search. For instance, if x and y are bounded, we might try to discretize the domain of φ, allowing each of x and y to take on $r \in \mathbf{Z}$ possible values. In this case, there are only r^2 points at which to evaluate φ; however, in the case of m variational variables each having r possible values, there will be r^m points to evaluate. As we will see, the dimensionality, m, of a control problem is a critical factor in the design of optimal control systems.

Bellman [5] and Pontryagin [29] were largely responsible for formulating the necessary problems and developing many of the basic approaches to solving optimal control problems. The requisite mathematics is complicated enough that the background required to even state the basic theorems does not seem warranted for our treatment here. Suffice it to say that the results for linear systems are extensive, and that, additionally, there are powerful

[6] Specifically, it is possible to derive open-loop (the system state is not employed in computing the next input) controllers for the case in which the final state is specified (fixed) in advance, and closed-loop (the system state is employed in computing the next input) controllers for the case in which the final state is not specified (free) in advance [25].

numerical methods that have proved successful for a range of nonlinear systems. For a good overview of the field, the reader is encouraged to consult the text by Athans and Falb [4]. In the remainder of this section, we focus on a particular class of optimal control problems called *multistage decision processes*, and a particular approach to solving such problems optimally due to Richard Bellman.

Consider a deterministic discrete-time n-stage process consisting of an initial state x_1, a sequence of inputs $u_1, u_2, \ldots, u_n$, and a sequence of resulting states $x_2, x_3, \ldots, x_n$ such that

$$x_{k+1} = f(x_k, u_k).$$

Following standard practice, the $\{u_k\}$ and $\{x_k\}$ are treated as variables ranging over U and X respectively. We introduce a performance index,

$$\mathrm{V}(u_1, \ldots, u_n; x_1, \ldots, x_n).$$

We wish to find input sequences that maximize V.

As we indicated earlier, in general, this problem of maximizing a function of n variables is computationally quite hard. In the worst case, it will be necessary to search through the set of $|U|^n$ possible sequences of length n in order to choose the sequence with the highest value. In some cases, however, we can do much better. In the following, we consider some easier problems that result from introducing restrictions on V. In particular, we consider the case in which at any stage in the process, say the kth stage, the effect of the remaining $n-k$ stages on the total value depends only on the state of the system following the kth decision and the subsequent $n-k$ decisions [6]. Let $\mathrm{R} : U \times X \to \mathbf{R}$ represent a *reward* function, where $\mathrm{R}(u, x)$ corresponds to the (immediate) benefit derived from performing action u in state x. We write $\mathrm{R}(u, x)$ if both the input and the state matter in determining the amount of reward and $\mathrm{R}(x)$ if only the state matters. As an example of the sort of performance functions we are interested in, we might have

$$\mathrm{V}(u_1, \ldots, u_n; x_1, \ldots, x_n) = \sum_{k=1}^{n} \mathrm{R}(u_k, x_k)$$

in which we are interested in the sum of rewards (referred to in the sequel as *separable control*), or

$$\mathrm{V}(u_1, \ldots, u_n; x_1, \ldots, x_n) = \mathrm{R}(x_n)$$

in which we are interested only in the reward associated with the final state (referred to as *terminal control*).

We proceed by generating a sequence of functions, $\{\mathrm{V}_n\}$, so that

$$\mathrm{V}_n(x_1) = \max_{u_k} \sum_{k=1}^{n} \mathrm{R}(u_k, x_k).$$

Expanding, we have

$$\begin{aligned} \mathrm{V}_n(x_1) &= \max_{u_k} \sum_{k=1}^{n} \mathrm{R}(u_k, x_k) \\ &= \max_{u_k} [\mathrm{R}(u_1, x_1) + \mathrm{R}(u_2, x_2) + \cdots + \mathrm{R}(u_n, x_n)] \\ &= \max_{u_1} \max_{u_2} \ldots \max_{u_n} [\mathrm{R}(u_1, x_1) + \mathrm{R}(u_2, x_2) + \cdots + \mathrm{R}(u_n, x_n)]. \end{aligned}$$

Rearranging, we obtain

$$\begin{aligned} \mathrm{V}_n(x_1) &= \max_{u_1} [\mathrm{R}(u_1, x_1) + \\ &\quad \max_{u_2} \max_{u_3} \ldots \max_{u_n} [\mathrm{R}(u_2, x_2) + \mathrm{R}(u_3, x_3) + \cdots + \mathrm{R}(u_n, x_n)]]. \end{aligned}$$

Note that

$$\mathrm{V}_{n-1} = \max_{u_2} \max_{u_3} \ldots \max_{u_n} [\mathrm{R}(u_2, x_2) + \mathrm{R}(u_3, x_3) + \cdots + \mathrm{R}(u_n, x_n)].$$

Substituting, we have in the case of separable control,

$$\mathrm{V}_n(x_1) = \max_{u_1} [\mathrm{R}(u_1, x_1) + \mathrm{V}_{n-1}(x_2)],$$

or just

$$\mathrm{V}_n(x) = \max_{u} [\mathrm{R}(u, x) + \mathrm{V}_{n-1}(f(x, u))]$$

for $n \geq 2$, and

$$\mathrm{V}_1(x) = \max_{u} \mathrm{R}(u, x).$$

for $n = 1$. For the case of terminal control, we have

$$\mathrm{V}_n(x) = \max_{u} [\mathrm{V}_{n-1}(f(x, u))], \quad \text{for} \quad n = 2, 3, \ldots$$

and

$$\mathrm{V}_1(x) = \mathrm{R}(x).$$

X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
X															X
X															X
X				X											X
X				X											X
X		G		X											X
X				X											X
X				X				X	X	X	X				X
X								X							X
X								X							X
X								X							X
X								X	X	X	X				X
X															X
X															X
X															X
X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X

Figure 4.9: A 16×16 grid world

The time to compute $\mathrm{V}_i(x)$ for all $x \in X$ given that invoking V_{i-1} has unit cost is $O(|X||U|)$. From this observation, it follows that the time required to compute $\mathrm{V}_n(x)$ for all $x \in X$ given that invoking V_1 has unit cost is $O(n|X||U|)$.

This general method of computing the performance index recursively is called *dynamic programming*. The basic constrained minimization variational problem essentially involves choosing a point in an n-dimensional phase space. Dynamic programming involves decomposing the problem into making n choices each of which involves a one-dimensional phase space [6].

To illustrate the basic technique involved in dynamic programming, we consider a simple robot control problem. A *grid world* is represented as an $n \times n$ grid. One cell of the grid is designated as the goal. Certain other cells (a total of m) are designated as obstacles. In particular, all of the perimeter cells are designated as obstacles. Initially, the robot is located in a cell which is not an obstacle. Figure 4.9 depicts a 16×16 grid world in which the goal is indicated by the letter "G", and the obstacles are indicated by the letter "X."

There are $n^2 - m$ states each one corresponding to the robot being in a

X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
X	-5	-4	-5	-6	-7	-8	-9	-10	-11	-12	-13	-14	-15	-16	X
X	-4	-3	-4	-5	-6	-7	-8	-9	-10	-11	-12	-13	-14	-15	X
X	-3	-2	-3	X	-7	-8	-9	-10	-11	-12	-13	-14	-15	-16	X
X	-2	-1	-2	X	-8	-9	-10	-11	-12	-13	-14	-15	-16	-17	X
X	-1	G	-1	X	-9	-10	-11	-12	-13	-14	-15	-16	-17	-18	X
X	-2	-1	-2	X	-8	-9	-10	-11	-12	-13	-14	-15	-16	-17	X
X	-3	-2	-3	X	-7	-8	-9	X	X	X	X	-16	-17	-18	X
X	-4	-3	-4	-5	-6	-7	-8	X	-20	-19	-18	-17	-18	-19	X
X	-5	-4	-5	-6	-7	-8	-9	X	-21	-20	-19	-18	-19	-20	X
X	-6	-5	-6	-7	-8	-9	-10	X	-22	-21	-20	-19	-20	-21	X
X	-7	-6	-7	-8	-9	-10	-11	X	X	X	X	-18	-19	-20	X
X	-8	-7	-8	-9	-10	-11	-12	-13	-14	-15	-16	-17	-18	-19	X
X	-9	-8	-9	-10	-11	-12	-13	-14	-15	-16	-17	-18	-19	-20	X
X	-10	-9	-10	-11	-12	-13	-14	-15	-16	-17	-18	-19	-20	-21	X
X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X

Figure 4.10: $\mathrm{V}(\langle x, y\rangle)$ for the grid world

particular cell not designated as an obstacle. There are five possible actions not all of which are necessarily available for a given state: the robot can remain in its current cell or move to any one of four adjacent cells ($\uparrow$, $\rightarrow$, $\downarrow$, and $\leftarrow$) as long as the destination cell is not designated as an obstacle. We use the value function for separable control where the reward is defined as follows:

$$\mathrm{R}(u, x) = \begin{cases} 0 & \text{if } x \text{ is equal to the goal} \\ -1 & \text{otherwise} \end{cases}$$

We compute V_1, V_2, up to V_k such that $\mathrm{V}_k = \mathrm{V}_{k-1}$ and set $\mathrm{V} = \mathrm{V}_k$. Figure 4.10 shows $\mathrm{V}(\langle x, y\rangle)$ for each state (location $\langle x, y\rangle$) in the grid world of Figure 4.9.

If you look carefully at the numbers shown in Figure 4.10, you will notice that by always moving to the neighboring location with the highest value you will eventually end up at the goal location no matter what location you start out in. This property can illustrated graphically by considering the elevation map shown in Figure 4.11 defined using $\mathrm{V}(\langle x, y\rangle)$ as the elevation at coordinates $\langle x, y\rangle$ in the grid with interior obstacles represented as small negative values. Notice that the goal location is a global maximum in the

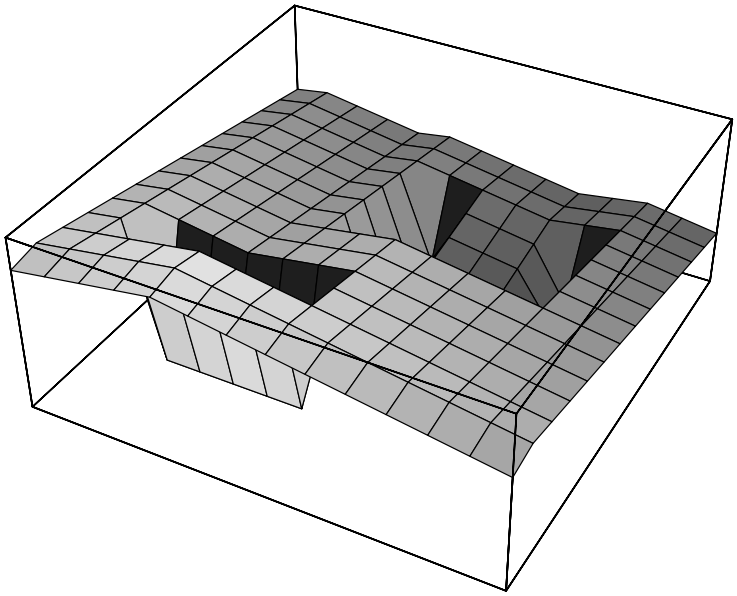

Figure 4.11: Representation of $\mathrm{V}(\langle x, y \rangle)$ as an elevation map

elevation map. This will always be the case no matter what the arrangement of obstacles. It turns out that the strategy of always moving to the location with the highest value is optimal in the following sense.

We define a control law or *policy* as a mapping from states to actions:

$$\eta : X \to U.$$

We are interested in policies that are optimal according to the following *principle of optimality* due to Bellman. Bellman states ([6] page 57) that "An optimal policy has the property that whatever the initial state and the initial decision are, the remaining decisions must constitute an optimal policy with regard to the state resulting from the first decision." Given Bellman's principle of optimality, the policy,

$$\eta(x) = \arg\max_u \mathrm{V}(f(x, u)),$$

is optimal.

Figure 4.12 shows the optimal policy for the grid world shown in Figure 4.9, where $\rightarrow$, $\leftarrow$, $\uparrow$, and $\downarrow$ indicate the direction of movement for the indicated state as specified by the optimal policy.

X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
X	↓	↓	←	←	←	←	←	←	←	←	←	←	←	←	X
X	↓	↓	←	←	←	←	←	←	←	←	←	←	←	←	X
X	↓	↓	←	X	↑	←	←	←	←	←	←	←	←	←	X
X	↓	↓	←	X	↑	←	←	←	←	←	←	←	←	←	X
X	→	G	←	X	↑	←	←	←	←	←	←	←	←	←	X
X	↑	↑	←	X	↓	←	←	←	←	←	←	←	←	←	X
X	↑	↑	←	X	↓	←	←	X	X	X	X	↑	←	←	X
X	↑	↑	←	←	←	←	←	X	→	→	→	↑	←	←	X
X	↑	↑	←	←	←	←	←	X	↑	↑	↑	↑	←	←	X
X	↑	↑	←	←	←	←	←	X	↑	↑	↑	↑	←	←	X
X	↑	↑	←	←	←	←	←	X	X	X	X	↓	←	←	X
X	↑	↑	←	←	←	←	←	←	←	←	←	←	←	←	X
X	↑	↑	←	←	←	←	←	←	←	←	←	←	←	←	X
X	↑	↑	←	←	←	←	←	←	←	←	←	←	←	←	X
X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X

Figure 4.12: An optimal policy

Because the transitions in state space are so localized in the grid world, we can use a much more efficient dynamic programming algorithm for computing the optimal policy than the one described above. In particular, we compute V_2 only for grid cells corresponding to one of the four neighbors of the goal adjacent along the grid axes, and, in so doing, treat V_1 as undefined for all cells other than the goal. In general, we compute V_i only for previously unconsidered grid cells corresponding to one of the four neighbors of cells considered in $i-1$th iteration, and treat V_{i-1} as undefined for all cells not considered in the $i-1$ or earlier iterations. If k is the last iteration in which there are unconsidered cells, then V_k is defined for all cells in the grid, and we set $V = V_k$. This specialized dynamic programming algorithm runs in $O(|X|)$.

The example application of dynamic programming given above involves a discrete deterministic dynamical system. Dynamic programming can be applied to continuous dynamical systems to achieve solutions of arbitrary accuracy using a variety of numerical techniques. Dynamic programming can be seen as a method of efficiently solving variational problems involving multiple local minima by cleverly guiding the search. Dynamic programming

can also be applied to stochastic processes, and we will return to this subject in Chapter 6.

Here as elsewhere the dimensionality of the problem severely restricts the application of this and most other methods to generating solutions efficiently. Dynamic programming is often referred to as an "approach" rather than a "method," where the distinction generally made is that an approach provides a way of looking at problems that still requires considerable creativity to actually apply, whereas a method is more a matter of turning a crank. Dynamic programming suggests that we try to view optimization problems as multistage decision problems in which the performance index is some simple (*e.g.,* additive) function of the state and input at each stage. If it is possible to view a problem thus, we can effectively reduce the dimensionality of the problem, thereby availing ourselves of substantial computational savings. Unfortunately, there are many aspects of a problem that serve to determine its dimensionality. For example, at best, the solution methods that we considered above involved computations linear in the size of the state space, and the dimensionality of the state space is determined by the number of state variables that comprise the state vector. In practical problems, methods that require quantifying over the entire state space can be computationally prohibitive. In subsequent chapters, we consider methods that allow us to decompose certain problems into independent subproblems, each of which requires quantifying over only a small portion of the state space.

4.6 Feedback Control Systems

In Section 4.2 on controllability, we considered a controller as a function from states to inputs (control actions). While there are many different types of controllers mentioned in the literature, this particular formulation is perhaps the most common. It is so common, in fact, that traditionally a *control law* is defined to be a function $\eta : T \times X \to U$,

$$u(t) = \eta(x(t), t).$$

However, in the problems we will be considering, η will not depend on the current time.

This basic idea that the inputs to a dynamical system should be computed from the state is quite important. Kalman ([20] page 46) describes it as "the fundamental idea of control theory," and "a scientific explanation of

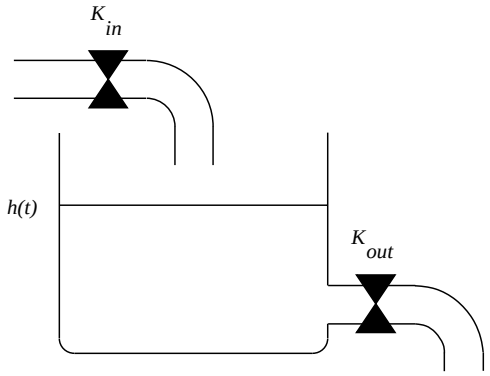

Figure 4.13: Controlling the level of fluid in a tank

the great invention known as 'feedback,' which is the foundation of control engineering."

It is worth asking why, if we have an accurate model of the process that we are trying to control, must we resort to sampling the state of this process on a continual basis. The answer is that uncertainty can and, generally, does arise from several sources besides the dynamical model. For instance, we have to sample the state of the system at some point in order to supply the initial conditions to the model. If there is any error in our measurement of the state variables, then that error will likely be exacerbated with the passage of time and as a consequence of inappropriate inputs generated on the basis of incorrect state information. Even if we are able to observe the state precisely, there will inevitably be some delay between our observation of the state and our initiation of a control action. This delay may be due to time spent in computing inputs, the response time of the actuators used to realize an input, or lags introduced by the sensors. We return to these issues in Chapter 6 when we consider the problems that arise in dealing with uncertainty in control.

In the following, we consider the application of feedback control to the fluid-flow problem introduced in Chapter 1. We begin by considering the problem of regulating the level of fluid in a tank using a closed-loop feedback controller. Figure 4.13 depicts the tank and its associated input and output pipes.

We model the controlled process as a first-order differential equation,

$$K_{in}\theta(t) - K_{out}h(t) = A\frac{d\,h(t)}{dt},$$

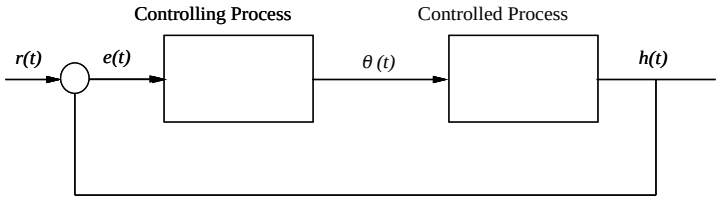

Figure 4.14: Block diagram for a closed-loop process controller

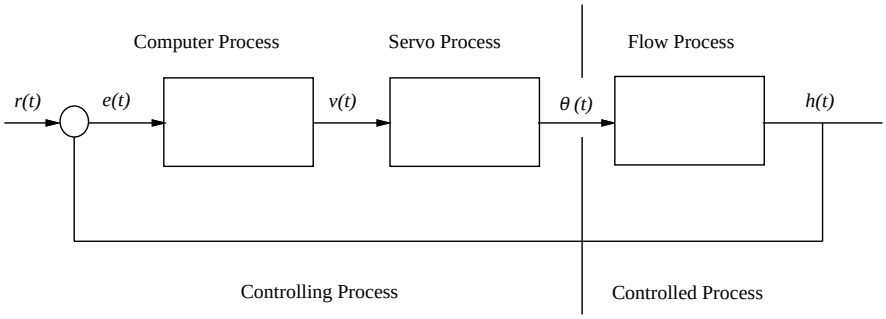

Figure 4.15: Decomposing the controlling process into subprocesses

where K_{in} is the flow constant in cubic meters per degree minute for the valve governing flow through the input pipe, K_{out} is the flow constant in square meters per minute for the output pipe, A is the surface area of the tank, $\theta(t)$ is the position of the valve governing flow through the input pipe at time t, and $h(t)$ is the height of the fluid in the tank at time t.

Now we have to specify a controlling process that changes θ in order to cause changes in h. In the simplest model, the controlling process directly determines θ by looking at the difference between the reference (or target) level and last measured value of h; this difference is referred to as the *error*. The block diagram shown in Figure 4.14 depicts this model with $r(t)$ indicating the reference and $e(t)$ indicating the error.

In Chapter 1, we defined a control algorithm that could cause instantaneous changes in θ. Needless to say, the typical interface between the controlling and controlled processes is more complex. In a somewhat more realistic model, the control computer might determine a voltage that is input to a servo system consisting of an amplifier and a DC motor attached to the input valve. The servo system is just another process, and we might model

it using the equation,

$$\frac{d\theta(t)}{dt} = K_g\, v(t),$$

where $v(t)$ is the input voltage and K_g is a constant that depends on the characteristics of the servo. Figure 4.15 provides a block diagram of this more complex model.

To define a process that determines the voltage input to the servo, we employ a standard technique from control theory. In many control schemes, the output of the controller is a simple function of the error. For controlling certain processes, an effective controller can be designed in which the output of the controller, $v(t)$ in this case, is directly proportional to the error,

$$v(t) = K_p\, e(t),$$

where K_p represents the controller proportionality constant. Not surprisingly, this sort of control is called *proportional control*.

For a control algorithm running on a digital computer, we have to specify a *discrete controller* that samples the output of the controlled process and outputs a control action at discrete intervals. The discrete proportional controller is just a computer program running on a specific machine that samples the output of the controlled process every so many clock cycles and outputs a value proportional to the computed error.

Though we will not dwell on such issues in this book, it should be noted that designing a discrete controller from the analysis of a continuous one is not a trivial exercise. If the sample period is short enough,[7] then the parameters determined for the continuous controller may determine a discrete controller that will serve as a good approximation to the continuous one. However, many problems require a much more complicated analysis, as it is not possible to sample quickly enough due to delays caused by computation and the physical limitations of sensors.

To maintain the level of fluid in the tank depicted in Figure 4.13 at two meters, we might use the following simple procedure.

```
do forever
   wait_for_delay
   height ← read_fluid_height
   error ← 2.0 − height
   servo_voltage ← Kp ∗ error
```

[7] Borrie [8] suggests as a rule of thumb that a good approximation to the continuous controller can be obtained by choosing a sampling period of one-tenth to one-fifth of the shortest time constant or cycle period of the highest frequency component of the dynamical system.

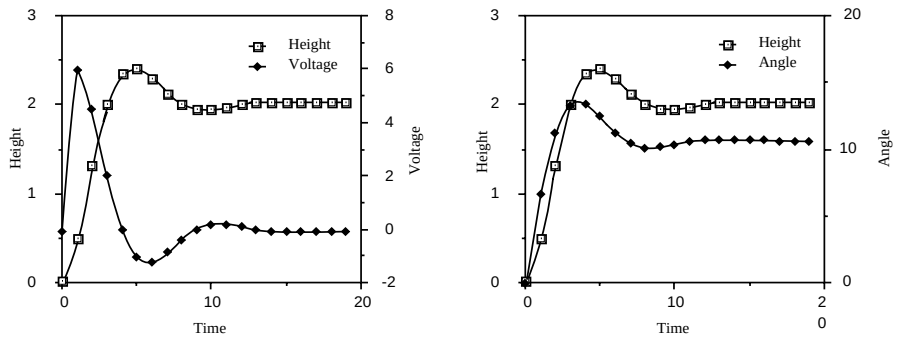

Figure 4.16: The behavior of the discrete proportional controller

where `read_fluid_height` reads the height sensor, `wait_for_delay` causes the controller to pause for the specified sample period, and `servo_voltage` is a machine register that directly determines the voltage fed to the servo. Figure 4.16 shows two graphs describing the behavior of the above control algorithm with a sample period of 1 minute and a proportionality constant of 3.0. One graph compares changes in h with changes in v, and a second compares changes in h with changes in θ. The particular proportionality constant 3.0 was chosen after a small amount of experimentation.

Proportional controllers are suitable for controlling only a limited class of processes. Two other popular forms of control are *integral control* and *derivative control*. The output $u(t)$ of an integral controller is proportional to the accumulated error,

$$u(t) = K_i \int_0^t e(t)\, dt,$$

whereas the output of a derivative controller is proportional to the change in the error,

$$u(t) = K_d \frac{d\, e(t)}{dt}.$$

The proportional-plus-integral-plus-derivative (or *PID*) controller generalizes the above three types of controllers:

$$u(t) = K_p\, e(t) + K_i \int_0^t e(t)\, dt + K_d \frac{d\, e(t)}{dt}$$

The fluid-flow problem involves a very simple, forgiving physical system. As we have modeled it, the flow process is stable, evolves slowly over time,

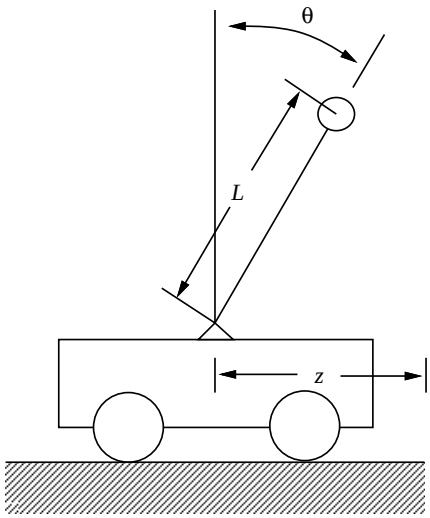

Figure 4.17: Inverted pendulum mounted on a cart

and manifests no delays between the execution of an action and its effects. By sampling very quickly, the controller can determine and respond to the effects of its actions almost immediately. In practice, however, there generally will be delays between when an action is executed and when that action has an effect on the state variables of interest. In the control literature, such delays are referred to as *transport lag* or *dead time*.

As an example of transport lag, if there is a long pipe between the input valve and the place where the pipe leads into the storage tank, a change in the angle of the input valve will take some time to have an effect on the level of fluid in the tank. Transport lag has a destabilizing influence on dynamical systems making them more difficult to control. If at all possible, transport lag should be avoided in designing equipment that has to be controlled by people or machines. By appropriately choosing the parameters for a *PID* controller, reasonable performance can often be achieved for a variety of physical systems exhibiting transport lag [8].

For the simple tank-filling process, proportional control is quite adequate. Other, less stable processes may require an integrator or a differentiator to damp oscillations and compensate for delays and disturbances. To illustrate, we consider the problem of balancing an inverted pendulum mounted on a cart that can move back and forth on a horizontal track. The arrangement of the cart and pendulum is shown in Figure 4.17.

We assume that the controller can observe the angle, $\theta(t)$, of the pendulum, and apply a force, $u(t)$, directly to the cart so as to control its position, $z(t)$. The motion of the cart and pendulum are determined by the following system of differential equations,

$$\begin{aligned} M\ddot{z}(t) + m\ddot{z}(t) + mL\ddot{\theta}(t)\cos\theta(t) - mL\dot{\theta}^2\sin\theta(t) &= u(t) \\ \frac{4}{3}mL^2\ddot{\theta}(t) + mL(\ddot{z}(t)\cos\theta(t) + L\ddot{\theta}(t)) - mLg\sin\theta(t) &= 0 \end{aligned}$$

where

$$M = 1 \text{ kg}, \;\; m = 0.15 \text{ kg}, \;\; L = 1 \text{ m}, \;\; g = 9.81 \text{ m/sec}^2.$$

Cannon [10] presents an analytical study of the dynamical system under proportional control, showing that the system is highly unstable. In accord with Cannon's results, we were unable to find a proportional controller that could maintain the pendulum in an upright position for more than a few minutes of simulated time. Figure 4.18.i shows one of the better performing proportional controllers that we were able to find; the system is shown failing to recover from an initial situation in which the pendulum is just short of vertical with zero initial velocity. The proportional controller is defined by the function,

$$u(t) = 12.25e(t),$$

where the error, $e(t)$, is in radians. The graph in Figure 4.18.i assumes a sample period of 1/30 of a second. The proportional controller managed to balance the pendulum for about 2.5 minutes sampling 30 times per second, and longer with higher sampling rates. However, even with the higher sampling rates, the controller eventually failed.

Figure 4.18.ii shows a proportional-plus-derivative controller successfully recovering from an initial situation in which the pendulum is set at an angle 15° from the vertical with zero initial velocity. The proportional-plus-derivative controller is defined by the function,

$$u(t) = 38e(t) + 140\frac{e(t) - e(t-\Delta)}{\Delta},$$

where the sample period, Δ, is 1/10 of a second for the graph shown in Figure 4.18.ii. The steady-state error for the proportional-plus-derivative controller after 10 seconds is less than 10^{-10} starting from an initial situation in which the pendulum is 15° from the vertical, and assuming zero initial velocity. The proportional-plus-derivative controller was able to recover from an initial position of up to 80° from the vertical.

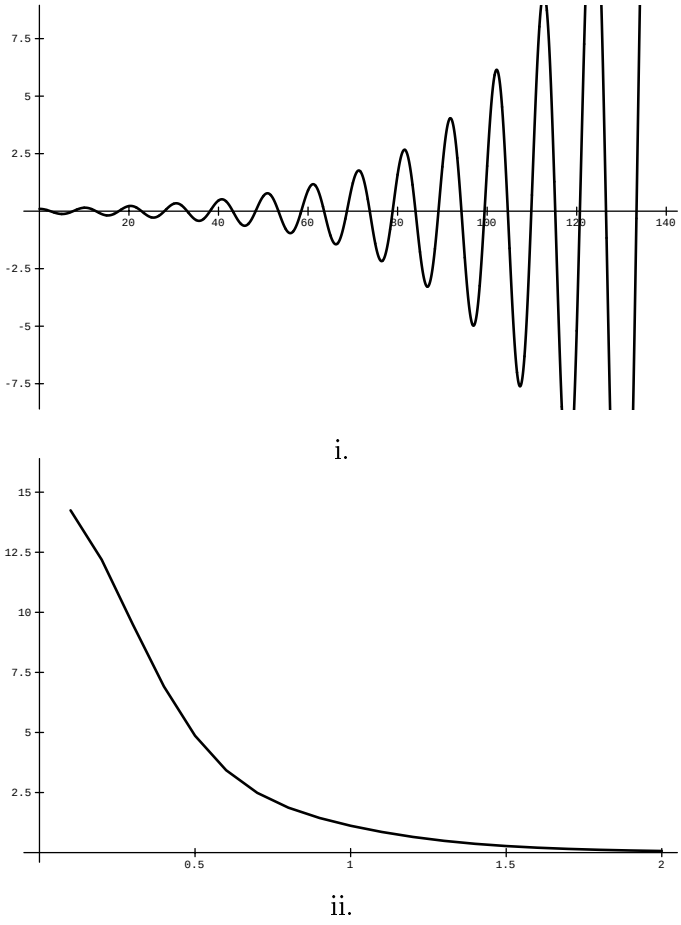

Figure 4.18: Balancing an inverted pendulum on a moving cart with (i) proportional and (ii) proportional-plus-derivative control

The performance of a control system can often be improved by introducing additional components into the system to compensate for deficiencies. This process is generally referred to as *compensation*, and the added components, often electrical or mechanical devices, are referred to as *compensators*.[8] For instance, in a system prone to undesirable oscillation, the system de-

[8]System compensation is sometimes defined more narrowly as designing a controller that achieves acceptable transient response while maintaining a specified steady-state accuracy.

signer might incorporate a hydraulic damper or increase the friction between sliding parts to damp the unwanted oscillations. Adding a derivative or integral term to an existing proportional controller represents a particular form of compensation.

In the examples above, we selected the control constants experimentally. Control theory, however, has developed powerful tools for analyzing and synthesizing control systems. In previous sections, we described some of the analytical tools for establishing stability, controllability, and observability. While it is beyond the scope of this book to delve deeper into control systems engineering, in the following sketch we attempt to convey a flavor of how a systems engineer might solve a control problem.

Suppose that the systems engineer is confronted with the same fluid-flow problem described earlier except that the target level changes over time. The controller is required to track the target level. Specifically, we require that (1) the controller respond to a one meter step change in the target by reducing and holding the error to less than 10 centimeters of the target within five minutes of the change in target level, and (2) the steady-state error be less than five centimeters for the above step change or for a continuous change in the target of 10 centimeters per minute. There are many ways of proceeding; we consider a typical approach.

The systems engineer begins by assuming a proportional feedback controller, and attempts to determine if it is possible for such a simple controller to meet the design requirements. The engineer constructs a dynamical model of the system, takes the Laplace transform of the equations of motion, and obtains a transfer function for the closed-loop controller by performing appropriate algebraic manipulations.

By analyzing the roots of the characteristic equation of the transfer function, the engineer can determine the steady-state performance of the system for different values of the proportionality constant. By applying an inverse transform to the transfer function, the engineer can determine the system's transient response to a step input. This combination of frequency- and time-domain methods allows the engineer to completely analyze the system with respect to the specified design requirements. In addition, the above sort of analysis can determine just how close a simple proportional controller can come to realizing the design requirements.

If simple proportional control cannot meet the design requirements, the engineer may want to compensate the feedback controller to improve performance. One method of compensation useful for improving transient response to a step input is to add a derivative term to the feedback controller. Deriva-

tive control has the effect of speeding up the system's response to a change in the target. If the steady-state error is unacceptable, then adding an integral term can often improve performance. Analysis would proceed as in the previous paragraph using both frequency- and time-domain analysis to determine appropriate values for the proportional, derivative, integral control constants.

For more realistic problems, the engineer might have to consider problems arising from sampling, delays due to processing and actuation, unmodeled behavior (*e.g.,* structural resonant modes in robot arms), and a host of other complications. For such problems, the mathematical tools of control theory are still useful, but they require increased skill and artistry on the part of systems engineer to apply them effectively.

The mathematical discipline of control theory is largely concerned with the formal analysis of control systems. As was mentioned in Section 4.5, in some cases, optimal control processes can be derived analytically providing that accurate models of the controlled processes are available. Since the characteristics of the controlled processes rarely are known precisely, control theorists are interested in systems that are insensitive to minor deviations in the models used in the design process. In cases where significant deviations are likely, or the models are known to be incomplete, *adaptive systems* are designed to compensate by adjusting the model as information becomes available.

Adaptive control techniques attempt to cope with uncertainty about the process being controlled by automating certain aspects of controller design. The basic idea is quite simple. The designer generally has some sort of model of the process (plant) that he is trying to build a controller for. This model, while it is known to provide only a rough idea of the behavior of the plant, is sufficient to determine the form of the basic controller (*e.g.,* a parameterized *PID* controller). The designer then builds a program that refines the basic controller as it observes this controller attempting to control the plant. In the case of a *PID* controller, refinement consists of adjusting the control coefficients. Adaptive control is one approach to making controllers more responsive to a complex and often unpredictable environment. Adaptive control also provides a means for coping with complexity in the design process by allowing a control system to monitor its own behavior and adjust accordingly. Chapter 9 deals with some aspects of adaptive control in the context of a discussion of learning techniques. Now we turn our attention to some of the programming issues involved in building control systems.

4.7 Computational Issues in Control

Control systems are complex devices that involve the interaction of mechanical and computational processes. In considering the computational aspects of control, it is important to keep in mind that someone has to write the programs or design the circuits that perform the necessary computations. For problems like controlling a power plant or an automated assembly line, these programs and circuits can become quite complex. Despite our best efforts, large programs develop organically as a process only partly under the control of any one individual. Continual redesign is impractical, and sooner or later the designer has to commit to a specific implementation of a module, interface, or subroutine. Once in a while, a designer has the luxury of rewriting an interface, optimizing an algorithm, or consolidating several functions in a single module, but often enough he or she has to make do with whatever is available. It would be convenient if control knowledge could be encapsulated in small general-purpose functional units that could be applied in a wide variety of circumstances. This has long been a dream of researchers in artificial intelligence, and, in the following, we consider some possible approaches to realizing that dream. Two critical issues that have to be addressed in the context of controlling processes are:

1. Can general-purpose control knowledge be used to support real-time control of interesting processes?

2. Can disparate behaviors be made to cooperate so as to achieve coordinated behavior across a range of situations?

In attempting to address these issues, we consider a class of programming techniques referred to as *embedded control systems* that were specifically designed to address shortcomings in classical approaches to planning relying primarily on offline computation and perfect information. Embedded control systems are meant to be responsive to the processes being controlled. They tend not to employ any complicated predictive mechanisms in order to avoid the computational overhead generally associated with such mechanisms. An embedded control system has to be prepared to respond quickly to changes perceived in the controlled process. If the system is engaged in a complex and time-consuming computation, it will likely miss opportunities to generate appropriate responses. In the applications for which embedded control systems are best suited, it should be possible to achieve the desired behavior using simple models that can be quickly computed.

Much of the work on embedded control systems done in artificial intelligence has been concerned with building systems that are capable of representing and manipulating precompiled procedural knowledge about how to control things. Different behaviors can be separately realized in terms of distinct procedures, each making use of the available sensors and effectors as needed. The differences between such systems usually revolve around the complexity of the primitive operations allowed by a given procedure and the means whereby procedures are selected, coordinated, and allowed to communicate with one another. In the following, we consider two approaches to building embedded control systems. For the most part, the two approaches look like programming languages, and our analysis concerns what features of the different languages make them more or less suitable for writing and thinking about control systems.

Every programming language is designed to support a particular level of abstraction. High-level languages can introduce barriers to abstraction by forcing the programmer to adopt a particular way of thinking. For instance, a language that provides only sequential control constructs can make it difficult to deal with parallel or asynchronous processes. Low-level languages can also introduce barriers to abstraction simply by failing to provide the programmer with adequate means to deal with the complexity of programming large systems. Of course, one can simulate any computational process given any Turing-equivalent machine/language combination. In looking at approaches designed to facilitate controlling processes, we should be alert to notice features that allow us to naturally map our understanding of control problems onto computational processes.

Almost every programming language provides support for procedures of one sort or another. Procedures encapsulate procedural knowledge: how to go about achieving certain tasks. In speaking about the control of processes, procedures are usually associated with specific behaviors. The first approach to implementing embedded control systems that we look at is called a *procedural reasoning system* [16]. A procedural reasoning system consists of a set of procedures and a *scheduler* for selecting what procedures to run and when. Each procedure has associated with it a specific task-achieving behavior that it implements, and an invocation condition or *goal* specifying what the procedure is meant to achieve.

Procedures are represented as *labeled transition graphs.labeled transition graph*. A labeled transition graph is a directed graph whose arcs are labeled with statements in some logic or programming language. In the following, we use Prolog statements to label arcs. The statements are examined by

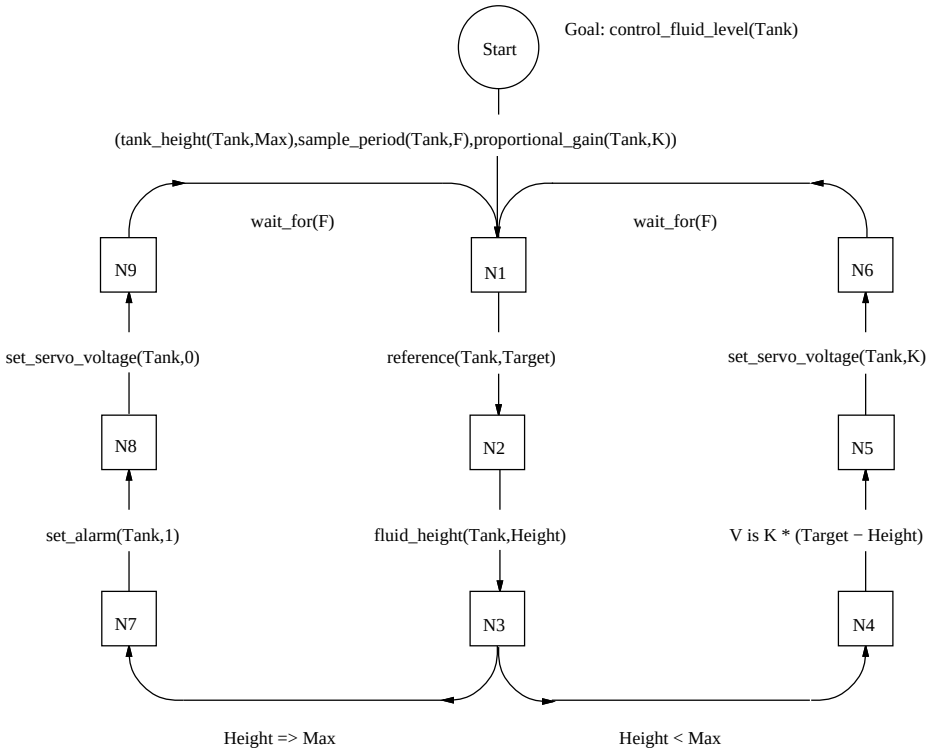

Figure 4.19: Labeled transition graph for a proportional controller

the scheduler to determine transitions from one node in the graph to some adjacent node in the graph. Each node in a labeled transition graph has one or more arcs leading out of it. Some statements correspond to predicates or queries and others have an imperative content. The statements labeling arcs are generally seen as giving rise to the goals of the system.

The scheduler is charged with keeping track of what goals the system has and invoking whatever procedures are appropriate to achieving those goals. At any given moment, the scheduler has some number of active procedures that it is employing to pursue its present goals. For each of those procedures, the scheduler maintains a pointer to some node in the associated labeled transition graph. The scheduler chooses a particular procedure to work on and attempts to transit to a new node by examining the statements on the arcs leading out of the node currently associated with the chosen procedure. An example should help clarify.

Figure 4.19 shows a labeled transition graph implementing the discrete proportional controller discussed earlier. The procedure shown also implements an overflow test to issue an alarm if the fluid runs over the top of the tank. Statements labeling arcs such as `fluid_height(Tank,Height)`, and `V is K * (Target - Height)` correspond to queries: "What is the current height of the fluid in the tank?" and "What voltage is `K` times the difference between the current height and reference value?" Statements such as `set_servo_voltage(Tank,V)` and `set_alarm(Tank,1)` correspond to imperatives to adjust parameters used by the procedures associated with the servo attached to the input valve and the alarm device.

Both queries and imperatives can be seen as giving rise to additional goals. For some of these goals, the scheduler invokes additional procedures. For other goals, special-purpose systems may kick in to try to satisfy the goal. For a given goal there may be many different procedures running. A procedure can be revoked if its associated goal becomes satisfied or if some competing goal becomes satisfied. Most labeled transition graphs have terminal nodes indicating exit conditions for the associated procedure. The scheduler is responsible for starting new procedures and terminating old ones. Procedures communicate with one another by posting goals to a global database in a manner similar to that used in blackboard systems [18]. A possible scheduling algorithm for a procedural reasoning system is described as follows. The scheduler maintains two queues ACTIVE and PENDING to keep track of procedures that are in various stages of processing.

1. Choose a procedure p from ACTIVE.
2. Post goals corresponding to each statement labeling an arc emanating from the current node of the procedure p.
3. Move p from ACTIVE to PENDING.
4. Add to ACTIVE each procedure whose invocation condition matches a goal posted in Step 2.
5. For each procedure q in PENDING such that any of the posted goals corresponding to the statements labeling arcs emanating from the current node of q are satisfied:
   (a) Choose one satisfied goal g.
   (b) Retract the other posted goals and remove any associated procedures from ACTIVE and PENDING.

(c) Set the current node of q to be the node terminating the arc labeled with the statement corresponding to g.

(d) Remove q from PENDING.

(e) If the current node of q is not a terminal node, move q to ACTIVE.

6. Go to Step 1.

It is important to note that the scheduler never waits around to compute anything; the scheduler simply posts new goals, invokes procedures where required, and notices when posted goals are satisfied. Suppose that the procedure shown in Figure 4.19 is the only active procedure and its current node is `N2`. The scheduler posts the goal `fluid_height(Tank,Height)` with `Tank` bound and `Height` unbound, and the procedure is moved to the list of pending procedures. The subsystem responsible for monitoring the level of fluid in the tank notices the posted goal, reads the sensor for fluid level, and marks the goal `fluid_height(Tank,Height)` as satisfied with `Height` bound to whatever the sensor read. The next time the scheduler looks at the pending procedures it notices the satisfied goal, updates the procedure's current node to `N3`, and places the procedure back on the list of active procedures.

The procedural reasoning system supports subroutine calls in that a transition in one procedure may require invoking a second procedure. Several procedures can run in parallel and communicate asynchronously by posting goals to the global database. As an example of how two procedures might work together in parallel, we consider a type of feedforward control that can be implemented easily in a procedural reasoning system.

The reference or target value specified in a control problem can be thought of as a command for the controller to achieve a particular condition (*e.g.,* a fluid level of the specified height). In many problems, the reference changes—sometimes continuously—over an interval. The controller has to track these changes so as to minimize errors. If the reference changes can be predicted or are simply provided in advance, the controller can take advantage of this information and use feedforward control to help improve transient response. For example, if the controller for a robot arm knows the exact trajectory it is to move the end effector along, it can often precompute a sequence of control actions, and then execute an error-free path without any feedback control whatsoever. In most cases, however, feedforward and feedback are used in conjunction, with feedforward taking advantage

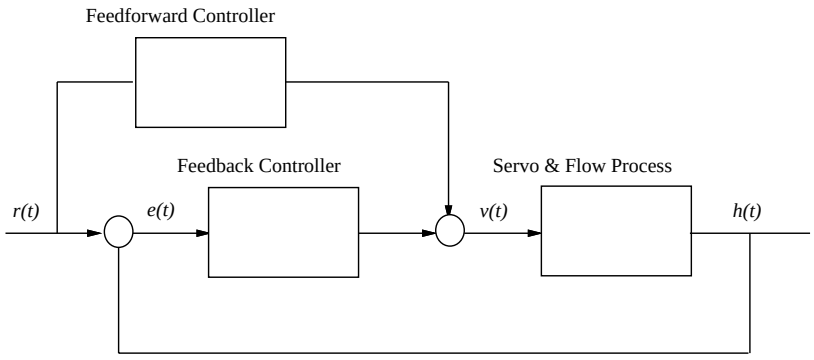

Figure 4.20: Block diagram for a controller with command feedforward

of known changes in the target value, and feedback compensating for the inevitable errors that arise in dealing with real-world processes.

In the case of our tank-filling process, a feedforward controller could be added to the feedback controller of Figure 4.15. The feedforward controller anticipates the next reference value and mediates the output of the feedback controller if a change is detected. This sort of controller is referred to as a *command feedforward* controller and its block diagram is shown in Figure 4.20.

To implement command feedforward control in a procedural reasoning system, we define a new procedure to monitor changes in the reference value. This procedure specifies a value proportional to the change in reference to be added to that specified by the feedback controller. The labeled transition graph for the command feedforward procedure is shown in Figure 4.21. The two procedures shown in Figure 4.19 and Figure 4.21 run at the same time. The servo process operates on a voltage which is the sum of that specified by each of the two procedures. This control scheme works particularly well for tracking a continuously changing reference; for instance, if you wanted the level in the tank to decrease to 0 at a fixed rate.

In describing the command feedforward control system above, we started with an existing feedback control system and then added a feedforward controller without changing the basic architecture of the feedback control system. *Hierarchical control systems* generalize on this basic idea. A hierarchical control system is constructed of several layers so that each layer serves as a controller for the layer immediately below and is controlled by the layer immediately above. There are different types of hierarchical control systems.

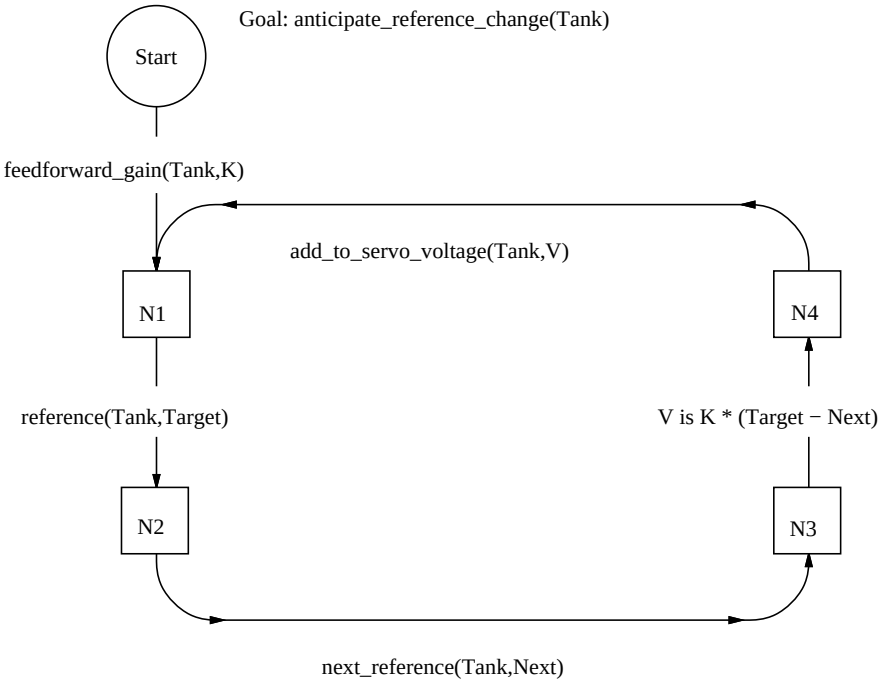

Figure 4.21: Labeled transition graph for a command feedforward controller

They differ in how the various layers are controlled by and impose control on the layers immediately above and below. As our second approach to building embedded control systems, we consider a hierarchical control system in which one layer is allowed to impose control on a lower layer by modifying control signals used for communicating between components of the lower layer [9].

Figure 4.22 depicts the general form of the sort of hierarchical control system we are considering. Each level is composed of a set of components each of which is responsible for a simple primitive behavior. The components communicate with one another by passing signals. For the most part, the signals consist of bit or byte streams. The components can be implemented any way that you prefer, but it is a good discipline to think of them as very simple computing devices. For instance, the components might be implemented as regular finite state machines augmented with a small amount of local state, a combinatorial circuit, and a local clock. The combinatorial circuit and local state are used to keep track of signals originating from other

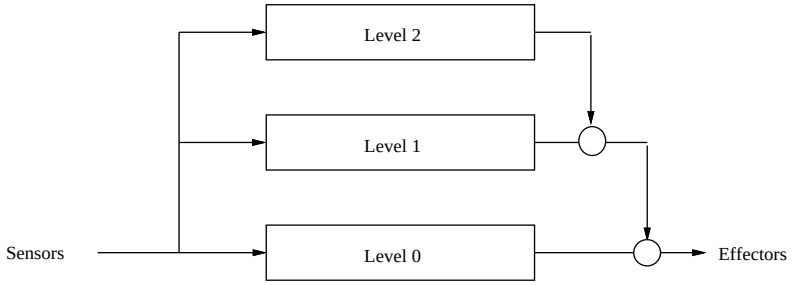

Figure 4.22: A hierarchical control system

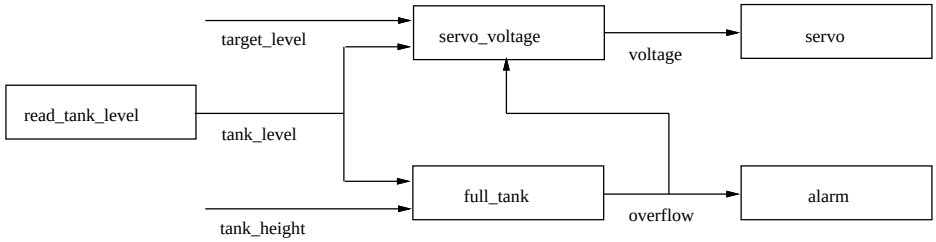

Figure 4.23: A single-level control system

components. The clock is used to provide simple timing capabilities. There is no global state and the different components communicate asynchronously by writing values into the local memory of other components.

Figure 4.23 shows a single-level control system for maintaining the fluid level in a holding tank. The component labeled `read_tank_level` continuously samples the sensor indicating the level of fluid in the holding tank and outputs the value read on the wire labeled `tank_level` which subsequently appears in registers in the components labeled `servo_voltage` and `full_tank`. The `servo_voltage` component implements the same procedure as the labeled transition graph of Figure 4.19. The `full_tank` component detects when the level in the tank is equal to the height of the tank and passes this information on to the `servo_voltage` component and to the `alarm` component which is responsible for sounding an alarm.

To illustrate how one level in a hierarchical control system might influence a lower level in the same system, we consider a second form of feedforward control referred to as *disturbance feedforward* control. A disturbance is a process that affects the controlled process but is not taken into account by the controlled process model. In the fluid-level process we have been

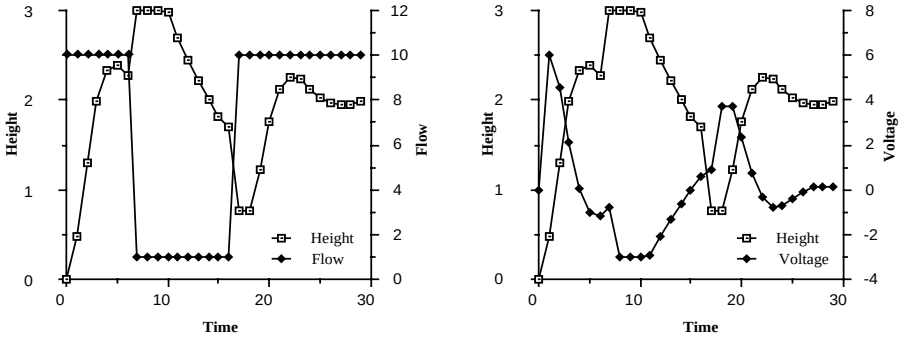

Figure 4.24: Overflow due to a disturbance restricting outflow

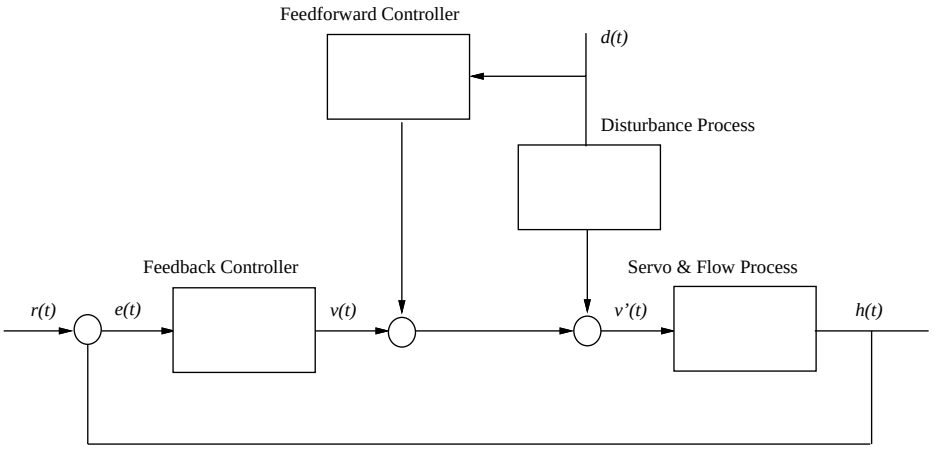

Figure 4.25: Block diagram for a controller with disturbance feedforward

considering, we might model a process restricting the flow through the pipe leading out of the tank shown in Figure 4.13 as a disturbance. Suppose that the output pipe is being used to fill containers that are moved into position under the pipe using a conveyor system. When a container is filled, the flow through the output pipe is temporarily restricted so that a new container can be positioned under the pipe. Figure 4.24 shows how a simple proportional controller reacts to a brief restriction in the output flow; the reduced flow effectively reduces the gain of the proportional controller and fluid spills over the top of the tank before the controller can react and appropriately compensate.

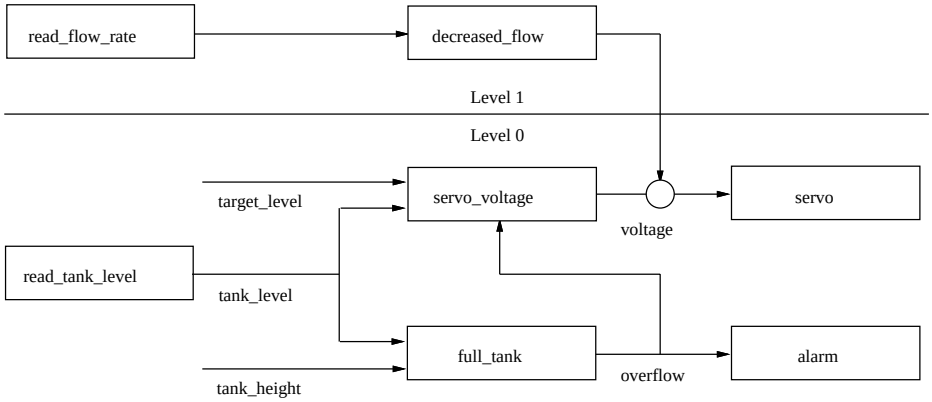

Figure 4.26: A two-level system with disturbance feedforward control

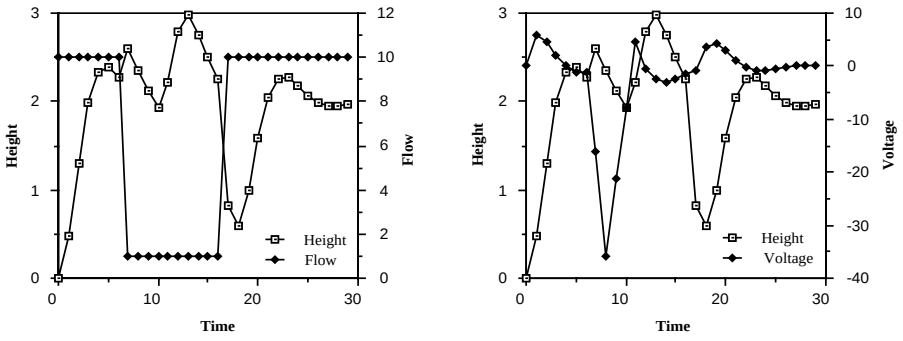

Figure 4.27: Disturbance feedforward controller preventing overflow

Let us suppose that it is possible to anticipate a restriction in the output flow as would be the case for the container-filling example described above. Figure 4.25 shows a block diagram for a disturbance feedforward controller for the fluid-level problem. We assume that it is possible to sense restrictions in the output flow and use this information to increase the voltage fed to the servo motor thereby temporarily increasing the gain of the feedback controller.

Given the single-level proportional controller shown in Figure 4.23, we can add a second control level in order to reduce or eliminate the amount of spillage resulting from momentary restrictions. The resulting two-level system is shown in Figure 4.26.

The performance of the two-level system is somewhat less than optimal; as indicated in Figure 4.27, the two-level system does avoid spilling any fluid, but the fluid height is somewhat erratic around the time of the restriction. We might be able to further tune the feedforward component to eliminate or reduce this erratic behavior. However, it is often the case that, in building on top of an existing control system, we simply have to accept the limitations of what we started out with, or do it over. The hierarchical system described above makes it rather easy to build on an existing control system. Given the discipline described earlier for building modular stand-alone computational components, adding new functionality or enhancing old often consists of simply adding some new components and wiring them together with the old ones. To the extent that this can be realized in practice, it makes building and experimenting with control systems remarkably easy.

The procedural reasoning system and the hierarchical control system described above are similar in many respects. Both support multiple processes running in parallel. Both support procedural abstraction and asynchronous control. There are some differences, however. The procedural reasoning system encourages the explicit representation of intentions, behaviors, and goals. The hierarchical control system encourages one to think in terms of evolving control systems and distributed computation. We say "encourage" as both systems are no more than general-purpose programming languages. Unless you specify a compiler and a target machine, the two systems are essentially equivalent.

There are other approaches to building embedded control systems some of which will be discussed in subsequent chapters. In some cases, the embedded control system looks more like the sort of planning systems that we will investigate in Chapter 5 in that it manipulates a representation of its pending tasks imposing ordering constraints and dealing with certain classes of interactions between tasks [14]. In others cases, the system is realized as a boolean circuit [1, 31] or as a network of processes that communicate using a specialized message passing protocol [28]. The process of compiling embedded control systems from behavioral specifications is of particular interest, and we will return to this issue in Chapters 8 and 10.

In the previous section on feedback control, we were concerned with the transient response and steady-state error of a controller. It should be noted that similar issues arise in all sorts of planning and control problems. An embedded control system can exhibit undesirable oscillation by getting into a loop in which it repeatedly performs an action and then turns around and negates an intended effect of that action. For instance, consider the classic

blocks-world problem in which a robot is required to stack block A on top of block B and B on top of block C where in the initial situation A is already on B and C is on the table with nothing on top. You can easily imagine a short-sighted embedded control system that continually removes A from B in preparation for lifting B on top of C, but then immediately follows this action by putting A back on top of B to satisfy one half of the required goal.

Recognizing and avoiding such self-defeating behavior can involve some rather complicated reasoning in the general case. We would not want to require this more complicated reasoning as a prerequisite to acting. Instead, we might design a quick, but near-sighted action component that occasionally gets itself into self-defeating cycles, coupled with a slower, but more far-sighted reasoner that has the ability to detect such cycles and force the action component back onto a productive track. Both the procedural reasoning system and the hierarchical control system described in this section, provide for such a division of labor. In Chapter 10, we consider general systems-architecture issues that relate to this sort of hybrid control system.

4.8 Navigation and Control

Traditionally, the problem of navigation, involving spatial and geometrical modeling, and the problem of control, involving kinematics and dynamical modeling have been considered separately. The former is believed to be in the realm of planning; the latter in the realm of control. In the first problem, we are given a geometrical model describing a robot, the objects surrounding it, their current relative positions and orientations, and some goal state describing a final position of the robot, and we are asked to generate a trajectory or path through the associated space of possible configurations of the robot and the surrounding objects. In the second problem, we are given a dynamical model of the robot and asked to generate a feedback control law that issues torques to manipulator joints and drive wheels in order to track a supplied reference trajectory. In this section, we consider a unified approach that addresses both of these problems.

To represent the state of the robot with respect to its environment, we introduce the idea of *configuration space* taken from mechanics and adapted for use in robotics [26]. Following Latombe [24], we represent the robot, $\mathcal{A}$, and the objects—we will refer to them as *obstacles*—in its environment, $\mathcal{B}_1, \mathcal{B}_2, \ldots, \mathcal{B}_m$, as closed subsets of the *work space*, $\mathcal{W} = \mathbf{R}^n$, where $n = 2$ or 3. Both the robot and the obstacles in the workspace are assumed to be

rigid. Let $\mathcal{F}_{\mathcal{A}}$ and $\mathcal{F}_{\mathcal{W}}$ be Cartesian frames of reference embedded in $\mathcal{A}$ and $\mathcal{W}$ respectively. $\mathcal{F}_{\mathcal{A}}$ is a moving frame while $\mathcal{F}_{\mathcal{W}}$ is fixed.

A *configuration*, q, of an object is a specification of the position and orientation of $\mathcal{F}_{\mathcal{A}}$ with respect to $\mathcal{F}_{\mathcal{W}}$. The *configuration space*, $\mathcal{C}$, is the set of all configurations of $\mathcal{A}$. We employ the Euclidean metric and the following distance function to induce a topology on $\mathcal{C}$. The distance between two configurations, $q, q' \in \mathcal{C}$, is defined as

$$\texttt{distance}(q, q') = \max_{a \in \mathcal{A}} \|a(q) - a(q')\|,$$

where $\|x - x'\|$ denotes the Euclidean distance between any two points, $x, x' \in \mathbf{R}^n$, and $a(q)$ is the point in $\mathcal{W}$ occupied by $a \in \mathcal{A}$ when $\mathcal{A}$ is in configuration q. We define the *free space*, $\mathcal{C}_{free}$, to be

$$\mathcal{C}_{free} = \{q | q \in \mathcal{C} \wedge \mathcal{A}(q) \cap (\bigcup_{i=1}^{m} \mathcal{B}_i) = \emptyset\},$$

where $\mathcal{A}(q)$ is that subset of $\mathcal{W}$ occupied by $\mathcal{A}$ in configuration q. A *free path* (or just a *path*) of $\mathcal{A}$ from some initial configuration, q, to the *goal* configuration, q^*, is a continuous map

$$\pi : [0, 1] \rightarrow \mathcal{C}_{free},$$

subject to the constraints that $\pi(0) = q$ and $\pi(1) = q^*$.

The literature is full of approaches to solving the problem of finding obstacle-free paths in configuration space. In the following, we consider the *artificial potential field* approach first introduced to the robotics community by Khatib [21] which unifies navigation (or path planning) and control. Our treatment here borrows the notation of Latombe [24], as well as some of the insights of Koditschek [23] on the connections between planning and control. To simplify the subsequent discussion, we assume that the robot is a point object and the workspace is $\mathbf{R}^2$. In this case, it is meaningless to talk about the robot's orientation, and, hence, the configuration space is identical to the work space.

We wish to design an artificial potential field so that the robot will be attracted toward the goal configuration in $\mathcal{C}$ and repulsed by obstacles. This field of forces is modeled as a function, F, defined by

$$F(q) = -\nabla U(q),$$

where $U : \mathcal{C}_{free} \rightarrow \mathbf{R}$ is a differentiable potential function, and the gradient, ∇, is defined in the case of $\mathcal{C} = \mathbf{R}^2$ as

$$\nabla U = \begin{bmatrix} \partial U/\partial x \\ \partial U/\partial y \end{bmatrix}.$$

We represent the potential function as a sum of attractive and repulsive component potential functions:

$$U(q) = U_{att}(q) + U_{rep}(q).$$

Generally, the attractive force is represented either as a conic potential well using the Euclidean distance, as in

$$U_{att}(q) = \xi \|q - q^*\|,$$

where ξ is a positive scaling factor, or as a parabolic potential well using the Euclidean distance squared, as in

$$U_{att}(q) = \frac{1}{2}\xi \|q - q^*\|^2,$$

where the constant $1/2$ is just to make ∇ come out a little neater. In the former case, we have

$$\nabla U_{att}(q) = \xi \frac{(q - q^*)}{\|q - q^*\|},$$

and in the latter

$$\nabla U_{att}(q) = \xi (q - q^*).$$

There are advantages and disadvantages to both approaches to representing the attractive potential. In some cases, it is useful to define a hybrid potential using a parabolic potential within some fixed radius of the goal (facilitating gradient descent search in the proximity of the goal) and a conic potential outside that radius (keeping the potential value smaller at points far from the goal) [24].

We decompose the repulsive component of the potential function into m additive components, one for each obstacle. In designing a repulsive field for a particular obstacle, we want to make it impossible for the robot to come in contact with the surface of the obstacle while allowing movement to proceed unimpeded when the robot is sufficiently distant from the obstacle. For a convex object, $\mathcal{B}_i$, the following potential function performs well

$$U_{\mathcal{B}_i}(q) = \begin{cases} \frac{1}{2}\eta \left(\frac{1}{\rho_i(q)} - \frac{1}{\zeta}\right)^2 & \text{if } \rho_i(q) \leq \zeta \\ 0 & \text{if } \rho_i(q) > \zeta \end{cases},$$

where ζ is a positive scalar called the *distance of influence*, and ρ_i is defined as

$$\rho_i(q) = \min_{q' \in \mathcal{B}_i} \|q - q'\|,$$

where we do not bother to distinguish between the configuration space and the work space, since in the cases considered here they are the same.

The gradient of $U_{\mathcal{B}_i}$ is defined by

$$\nabla U_{\mathcal{B}_i}(q) = \begin{cases} \eta \left(\frac{1}{\rho_i(q)} - \frac{1}{\zeta}\right) \frac{1}{\rho_i^2(q)} \nabla \rho_i(q) & \text{if } \rho_i(q) \leq \zeta \\ 0 & \text{if } \rho_i(q) > \zeta \end{cases},$$

where $\nabla \rho_i(q)$ is defined as follows. Let q_c be the unique configuration in $\mathcal{B}_i$ such that $\|q - q_c\| = \rho_i(q)$. $\nabla \rho_i(q)$ is the unit vector pointing away from $\mathcal{B}_i$ in the direction determined by the line passing through q and q_c.

We combine the repulsive fields for the set of obstacles, $\{\mathcal{B}_1, \mathcal{B}_2, \ldots, \mathcal{B}_m\}$, by taking a simple sum,

$$U_{rep}(q) = \sum_{i=1}^{m} U_{\mathcal{B}_i}(q).$$

The gradient of the sum is simply the sum of the gradients,

$$\nabla U_{rep} = -\sum_{i=1}^{m} \nabla U_{\mathcal{B}_i}(q).$$

Combining the attractive and repulsive force fields, we have

$$F(q) = \nabla U_{att} + \nabla U_{rep}.$$

Figure 4.28 shows a 2-D configuration space, the resulting potential fields, and several equipotential contours indicating that the potential field has a single minimum. The attractive potential is modeled as a parabolic potential well.

The potential field approach was originally conceived of as a method for real-time obstacle avoidance. The basic idea was to regard the robot in configuration space as a particle moving under the influence of the field, $F = -\nabla U$. The acceleration is determined by $F(q)$ for every $q \in \mathcal{C}$. Given the dynamics of $\mathcal{A}$ and assuming perfect sensing and motors that deliver exact and unlimited torque, we can compute the torques that should be issued to each of the actuators so that the robot behaves exactly as the particle metaphor predicts.

Figure 4.28: A 2-D configuration space (i) containing two obstacles. The attractive potential field (ii), along with the repulsive potential fields (iii and iv) for each of the two obstacles, the sum (v) of the attractive and repulsive potential fields, and a 2-D plot (vi) showing several equipotential contours.

Consider a very simple robot with one degree of freedom (*e.g.*, a prismatic (sliding) joint). We assume that its position (configuration), $q \in \mathcal{C} = \mathbf{R}$, and velocity, $\dot{q}$, can be measured precisely by a perfect sensor and controlled by a servo that delivers exact and unlimited force, $\mathcal{F}$. We model the dynamical system using Newton's second law of motion,

$$M\ddot{q} = \mathcal{F},$$

where M is the mass of the robot. The object is to move the robot from its present configuration to some final configuration q^*.

In the potential field approach described above, we address the geometrical side of the problem in terms of optimizing a cost function disguised as a potential function. This approach is quite similar to the dynamic programming example that we investigated in Section 4.5. The cost function that we are trying to minimize in this case is just the attractive potential function introduced earlier

$$\varphi = \frac{1}{2} K_P \|q - q^*\|^2,$$

where K_P is any positive scalar. To simplify the present discussion, we ignore the problem of avoiding obstacles. From this equation, we obtain

$$\dot{q} = -\nabla\varphi = -K_P(q - q^*),$$

and note that, since in this case q^* is the only minimum of φ, this linear differential equation generates a solution to the geometric problem of finding a path from any initial starting configuration to q^*. Now we set out to derive a control law that will serve to track the path (or reference trajectory) so defined.

Having interpreted φ in terms of potential energy, we define the kinetic energy, κ, as

$$\kappa = \frac{1}{2} M \dot{q}^2,$$

and obtain the total energy, λ, as the difference of the kinetic and potential energies

$$\lambda = \kappa - \varphi.$$

A dynamical model can be obtained using the Lagrangian formulation of Newton's equations defined by

$$\frac{d}{dt}\left(\frac{\partial \lambda}{\partial \dot{q}}\right) - \frac{\partial \lambda}{\partial q} = \mathcal{F}_{ext},$$

where $\mathcal{F}_{ext}$ represents all of the external (non-conservative) forces acting on the robot. The resulting Newtonian law of motion is

$$M\ddot{q} - K_P(q - q^*) = \mathcal{F}_{ext}.$$

Let us assume that $\mathcal{F}_{ext}$ represents a dissipative force (we can add this if necessary) proportional to the velocity,

$$\mathcal{F}_{ext} = -K_D \dot{q},$$

where K_D is a positive scalar. The resulting system is asymptotically stable, and converges to the goal q^* from all initial configurations $q \in \mathcal{C}$.

Finally, we have

$$M\ddot{q} + K_D\dot{q} - K_P(q - q^*) = 0.$$

Returning to our original dynamical model,

$$M\ddot{q} = \mathcal{F},$$

we can obtain the following control law,

$$\mathcal{F} = -K_D\dot{q} + K_P(q - q^*),$$

an instance of proportional derivative feedback control. The proportional component captures the essence of a simple one-dimensional planning system that determines an appropriate reference trajectory in configuration space. The derivative component enables the controller to respond appropriately to the behavior of the two-dimensional (one spatial and one temporal dimension) physical system.

Khatib's motivation for employing artificial potential fields was to provide real-time obstacle avoidance capability for multi-link manipulators [21]. In his original formulation, it was assumed that there would exist a higher level of control that would compute a global strategy in terms of intermediate goals. The low-level system would produce the necessary forces to achieve these goals, accounting for the detailed geometry, kinematics, and dynamics in real time. In the following, we say a bit more about the high-level problem of computing a global strategy corresponding to a path from the current configuration to the goal configuration.

The approach to building potential fields described earlier has a number of problems, some of which can be easily remedied and others of which are more difficult to overcome. We address some of these problems now, beginning with the easiest ones, working our way up to the more difficult.

The repulsive field for obstacles in the workspace was defined only for convex objects. We can extend the method to handle more general objects by decomposing each obstacle into some number of (possibly overlapping) convex objects, associating a repulsive potential with each component, and summing the result. There are some subtleties with this approach (see [24]), but this basic method of decomposition works well in practice.

The next problem concerns the assumptions regarding the dimensions of the workspace and the degrees of freedom of the robot. For the idealized point robot operating in two dimensions, the two-dimensional configuration space was equivalent to the Euclidean plane. In general, the number of parameters required to describe the configuration of the robot will determine the dimension of the configuration space. For a rigid robot operating in three dimensions, it takes six parameters to describe the configuration of the robot. For manipulators consisting of rigid links serially connected by single-degree-of-freedom joints (*e.g.,* revolute (rotating) and prismatic (sliding) joints), the number of parameters required is equal to the number of joints. For existing mobile robots and manipulators, it is possible to construct the requisite configuration spaces and extend the techniques described above to handle the resulting motion planning problems. However, assuming P $\neq$ NP, the complexity of planning free paths is exponential in the dimension of the

configuration space.

In general, computing free paths for multi-link manipulators and mobile robots in cluttered environments can be quite expensive [33]. From the perspective of computational complexity, this high-level geometric planning problem is typical of the sort of problems that we will encounter in the next chapter. Solutions to problems involving a significant number of constraints (*e.g.,* an environment cluttered with obstacles) and many alternative control actions (*e.g.,* robots with several degrees of freedom) tend to be computationally prohibitive. For real-time applications involving such problems, it is generally necessary to make simplifying assumptions thereby decreasing the complexity of the resulting decision problem while at the same time sacrificing generality and possibly risking soundness or completeness.

Another problem with the artificial potential function approach outlined earlier concerns the problem of multiple extrema in potential fields. In general, a potential field for a cluttered work space may include several extrema. Under such conditions, using the gradient to guide search may result in paths that terminate at extrema other than the one corresponding to the goal configuration. Concave objects are one potential source of misleading local extrema (see Figure 4.29.iii), but such extrema can also result in the case of closely situated convex obstacles if the distance of influence, ζ, is greater than twice the distance between the obstacles (see Figure 4.29.i).

In order to avoid getting stuck in local minima, it is necessary to employ more sophisticated search methods than simple gradient descent. In the following, we consider one such method for finding collision-free paths in a two-dimensional configuration space.[9]

We begin by tessellating the configuration space to form a grid of equally sized cells. In the case of a point robot on a planar surface, the discretized configuration space, $\mathcal{C}_{\mathbf{Z}}$, is a subset of the integer plane, $\mathbf{Z} \times \mathbf{Z}$:

$$\mathcal{C}_{\mathbf{Z}} = \{(i, j) | 0 \leq i, j \leq r\},$$

where r is an integer parameter used to bound the size of the configuration space. The potential at the coordinates, (i, j), in the integer plane is $U(il, jl)$ where l is the length of the side of a cell. We assume that both the initial and the goal configurations are configurations in $\mathcal{C}_{\mathbf{Z}}$, and that, if two configurations are neighbors in $\mathcal{C}_{\mathbf{Z}}$ and both of them belong to $\mathcal{C}_{free}$, then the straight line segment connecting them also lies in $\mathcal{C}_{free}$.

[9] The method for searching two-dimensional configuration space described here can be extended to higher-dimensional configuration spaces with little modification, but is only practical for dimension $\leq$ 4 [24].

Figure 4.29: Two potential fields with multiple extrema: one (i) resulting from two closely situated convex obstacles, and a second (iii) resulting from a single concave obstacle. Corresponding equipotential contours are shown (ii and iv) for each of the two potential fields.

In the following, T is a tree whose nodes are configurations in $\mathcal{C}_{\mathbf{Z}}$. We define a *best-first path planning algorithm* as follows.

1. Initialize T to be the tree consisting of the single (root) node corresponding to the current configuration.

2. Choose a leaf node, q, of T with unexplored neighbors in $\mathcal{C}_{\mathbf{Z}}$ whose potential value is equal to or less than the potential value of all the other leaves in T with unexplored neighbors.

3. Add to T as children of q all configurations not already in T whose potential value is less than some (large) threshold. (This threshold is set to avoid paths that get too close to obstacles. Recall that at the surfaces of obstacles the potential is infinite.)

4. If q^* is a leaf node in T, then go to Step 6.

5. If there are no leaf nodes in T with unexplored neighbors, then return failure, else go to Step 2.

6. Return the path from the root of T to q^*.

The algorithm described above is guaranteed to find a free path if one exists or report failure otherwise. The algorithm deals with multiple extrema by following a discrete approximation to gradient descent until reaching a local minimum. Once in a local minimum, it proceeds to "fill in" the well of this minimum by exploring the surrounding cells until a saddle point is reached and the local minimum is avoided. By adding simple optimizations to facilitate finding the next node to explore, it is possible to achieve a running time of $O(mr^m \log r)$ for a configuration space of dimension m. The algorithm works for configuration spaces of arbitrary dimension, but for dimension much greater than four the running time is prohibitive.

It should be noted that the best-first planning algorithm will find a path if one exists, but not necessarily the shortest path or the optimal path by any given metric. The discretized configuration space can be used as part of a dynamic programming approach to finding optimal paths. Indeed, using a dynamic programming approach, we can design an algorithm that will construct a potential field with a single minima at q^* in $O(mr^m)$. Using this potential field, one can generate the shortest path from any initial location to q^* using a discrete approximation to gradient descent in time linear in the length of the path.

Koditschek [22] provides a method of generating potential functions that he calls *navigation* functions which have a single global minimum. The advantage is that simple local methods (*e.g.*, gradient descent) suffice for navigation and control. However, as with other approaches to motion planning, the cost of generating navigation functions can be quite high in the case of cluttered environments and robots with many degrees of freedom.

This section was meant as a bridge between the central issues of this chapter and those of the next. In this chapter, we considered basic properties of dynamical systems such as controllability, observability, and stability

that are critical in the design of control systems. We investigated the fundamental idea of feedback control and considered the use of performance measures in optimal control. Finally, in this section we considered the idea of providing higher-level direction for control in the context of navigation problems. In particular, we considered methods for encoding navigation tasks in terms of potential functions that provide a convenient basis for the control of manipulators and mobile robots. The next chapter considers the issues involved in encoding high-level tasks in much more detail. Like the problems involved in motion planning, the problems we look at in the next chapter are computationally complex.

4.9 Further Reading

The literature on control systems theory and practice is vast. In the following, we point out some books and articles that we have found particularly useful in understanding the basic control issues and their attendant mathematical formulations. For a good overview of classical and modern approaches to control, the introductory text by Dorf [12] is excellent. Most control texts assume a relatively high level of mathematical sophistication. In particular, some familiarity with linear systems analysis is generally assumed. The text by Chen [11] provides a good introduction to linear systems theory. Gopal's book [17] on the control of linear multivariable systems is an excellent introduction to that subject. For more of an engineering perspective on control, the interested reader is advised to consult Bollinger [7] or Borrie [8]. Cannon [10] provides a number of case studies demonstrating how an engineer goes about analyzing and designing control systems. The high-level description of the process of designing a controller for the fluid-flow problem is modeled after Cannon's analysis of an electromechanical "repeater" used to monitor the orientation of a remote device such as a wind-direction indicator.

The survey article by Ramadge and Wonham [30] provides a good introduction to work in the area of discrete events systems. Optimal control texts generally rely on a good background in the differential and integral calculus, and, in particular, the calculus of variations [15]. Athans and Falb [4] provide an introduction to optimal control. There have been many books written on dynamic programming. The original text by Bellman [5] is still generally available and provides a good introduction to the subject with plenty of illustrative examples.

Much of the work done in artificial intelligence has concentrated on languages for expressing control knowledge and techniques for compiling decision procedures from behavioral specifications. The procedural reasoning system of Section 4.7 is modeled after the system designed by Georgeff and Lansky [16]. The procedural reasoning system is also closely related to the blackboard architecture of Hayes-Roth [18]. The hierarchical control system described in the same section is modeled after the subsumption architecture of Brooks [9]. Rosenschein and Kaelbling [31], Agre and Chapman [1], and Nilsson [28] all describe methods for building control systems that can be ascribed the semantics of boolean circuits. Firby [14] defines a language and interpreter for that language that are particularly well suited for programming robots that operate in uncertain environments. Schoppers [32], Kaelbling [19], and Drummond [13] describe methods for constructing policies from a set of goals the system is required to achieve. Miller [27] describes techniques for compiling robot decision procedures that account for sensors as potentially scarce resources.

For a careful treatment of the configuration space representation and a variety of approaches to finding free paths in configuration space, the reader is encouraged to read Latombe's book on robot motion planning [24]. For a survey of complexity results pertaining to motion planning, see Schwartz, Sharir, and Hopcroft [34]. Koditschek [23] provides a technical and historical survey of navigation techniques using potential functions including a discussion of stability issues. Arkin [3] describes a system in which high-level commands are communicated to low-level control systems in terms of potential functions that are used to construct an artificial potential field for guiding a robot.

References

[1] Philip E. Agre and David Chapman. Pengi: An implementation of a theory of activity. In *Proceedings AAAI-87*, pages 268–272. AAAI, 1987.

[2] Alfred V. Aho, John E. Hopcroft, and Jeffrey D. Ullman. *Data Structures and Algorithms*. Addison-Wesley, Reading, Massachusetts, 1983.

[3] Ronald R. Arkin. Motor schema based navigation for a mobile robot: An approach to programming by behavior. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 264–271, 1987.

[4] Michael Athans and Peter L. Falb. *Optimal Control: An Introduction to the Theory and Its Applications.* McGraw-Hill, New York, 1966.

[5] Richard Bellman. *Dynamic Programming.* Princeton University Press, 1957.

[6] Richard Bellman. *Adaptive Control Processes.* Princeton University Press, Princeton, New Jersey, 1961.

[7] John G. Bollinger and Neil A. Duffie. *Computer Control of Machines and Processes.* Addison-Wesley, Reading, Massachusetts, 1988.

[8] John A. Borrie. *Modern Control Systems: A Manual of Design Methods.* Prentice-Hall, Englewood Cliffs, New Jersey, 1986.

[9] Rodney A. Brooks. A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, 2:14–23, 1986.

[10] Robert H. Cannon. *Dynamics of Physical Systems.* McGraw-Hill, New York, 1967.

[11] C. T. Chen. *Introduction to Linear System Theory.* Holt, Rinehart, and Winston, New York, 1970.

[12] Richard C. Dorf. *Modern Control Systems.* Addison-Wesley, Reading, Massachusetts, 1989.

[13] Mark Drummond. Situated control rules. In Ronald J. Brachman, Hector J. Levesque, and Raymond Reiter, editors, *Proceedings of the First International Conference on Principles of Knowledge Representation and Reasoning.* Morgan-Kaufmann, Los Altos, California, 1989.

[14] R. James Firby. An investigation in reactive planning in complex domains. In *Proceedings AAAI-87*, pages 202–206. AAAI, 1987.

[15] I. M. Gelfand and S. V. Fomin. *Calculus of Variations.* Prentice-Hall, Englewood Cliffs, New Jersey, 1963.

[16] Michael P. Georgeff and Amy L. Lansky. Reactive reasoning and planning. In *Proceedings AAAI-87*, pages 677–682. AAAI, 1987.

[17] M. Gopal. *Modern Control System Theory.* Halsted Press, New York, 1985.

[18] Barbara Hayes-Roth. A blackboard architecture for control. *Artificial Intelligence*, 26:251–321, 1985.

[19] Leslie Pack Kaelbling. Goals as parallel program specifications. In *Proceedings AAAI-88*, pages 60–65. AAAI, 1988.

[20] R. E. Kalman, P. L. Falb, and M. A. Arbib. *Topics in Mathematical System Theory*. McGraw-Hill, New York, 1969.

[21] Oussama Khatib. Real-time obstacle avoidance for manipulators and mobile robots. *International Journal of Robotics Research*, 5:90–99, 1986.

[22] D. Koditschek. Exact robot navigation by means of potential functions: Some topological considerations. In *IEEE International Conference on Robotics and Automation*, pages 1–6, 1987.

[23] D. Koditschek. Robot planning and control via potential functions. In Oussama Khatib, John H. Craig, and Tomás Lozano-Pérez, editors, *Robotics Review 1*, pages 349–367. MIT Press, Cambridge, Massachusetts, 1989.

[24] Jean-Claude Latombe. *Robot Motion Planning*. Kluwer Academic Publishers, Boston, Massachusetts, 1990.

[25] Frank L. Lewis. *Optimal Control*. John Wiley and Sons, New York, 1986.

[26] Tomás Lozano-Pérez. Spatial planning: A configuration space approach. *IEEE Transactions on Computers*, 32:108–120, 1983.

[27] David P. Miller. Planning by search through simulations. Technical Report 423, Yale University Computer Science Department, 1985.

[28] Nils J. Nilsson. Action networks. In Josh Tenenberg, Jay Weber, and James Allen, editors, *Proceedings from the Rochester Planning Workshop: From Formal Systems to Practical Systems*, pages 36–68, 1989.

[29] L. S. Pontryagin, V. G. Boltyanskii, R. V. Gamkrelidze, and E. F. Msichenko. *The Mathematical Theory of Optimal Processes*. John Wiley and Sons, New York, 1962.

[30] Peter Ramadge and Murray Wonham. The control of discrete event systems. *Proceedings of the IEEE*, 77(1):81–98, 1989.

[31] Stan Rosenschein and Leslie Pack Kaelbling. The synthesis of digital machines with provable epistemic properties. In Joseph Y. Halpern, editor, *Theoretical Aspects of Reasoning About Knowledge: Proceedings of the 1986 Conference*, pages 83–98. Morgan Kaufmann, 1986.

[32] Marcel J. Schoppers. Universal plans for reactive robots in unpredictable environments. In *Proceedings IJCAI 10*, pages 1039–1046. IJCAII, 1987.

[33] J. T. Schwartz and M. Sharir. On the pianos movers' problem: I. *Communications on Pure and Applied Mathematics*, 36:345–398, 1983.

[34] J. T. Schwartz, M. Sharir, and J. Hopcroft. *Planning, Geometry, and Complexity of Robot Motion.* Ablex, Norwood, New Jersey, 1987.

[35] William A. Wolovich. *Linear Multivariable Systems.* Springer-Verlag, New York, 1974.